The Time-Triggered Architecture

HERMANN KOPETZ, FELLOW, IEEE AND GÜNTHER BAUER

Invited Paper

The time-triggered architecture (TTA) provides a computing infrastructure for the design and implementation of dependable distributed embedded systems. A large real-time application is decomposed into nearly autonomous clusters and nodes, and a fault-tolerant global time base of known precision is generated at every node. In the TTA, this global time is used to precisely specify the interfaces among the nodes, to simplify the communication and agreement protocols, to perform prompt error detection, and to guarantee the timeliness of real-time applications. The TTA supports a two-phased design methodology, architecture design, and component design. During the architecture design phase, the interactions among the distributed components and the interfaces of the components are fully specified in the value domain and in the temporal domain. In the succeeding component implementation phase, the components are built, taking these interface specifications as constraints. This two-phased design methodology is a prerequisite for the composability of applications implemented in the TTA and for the reuse of prevalidated components within the TTA. This paper presents the architecture model of the TTA, explains the design rationale, discusses the time-triggered communication protocols TTP/C and TTP/A, and illustrates how transparent fault tolerance can be implemented in the TTA.

Keywords*—*Distributed systems, embedded systems, real-time systems, safety-critical systems, time-triggered architecture (TTA), TTP/C.

I. INTRODUCTION

Computer architectures establish a blueprint and a framework for the design of a class of computing systems that share a common set of characteristics. The time-triggered architecture (TTA) generates such a framework for the domain of large distributed embedded real-time systems in high-dependability environments. It sets up the computing infrastructure for the implementation of applications and provides mechanisms and guidelines to partition a large application into nearly autonomous subsystems along small and well-defined interfaces in order to control the complexity of the evolving artifact [1]. Architecture design is thus interface design. By defining an architectural style that is observed at all component interfaces, the architecture avoids property mismatches at the interfaces and eliminates the need for unproductive "glue" code.

Characteristic for the TTA is the treatment of (physical) real time as a first-order quantity. The TTA decomposes a large embedded application into clusters and nodes and provides a fault-tolerant global time base of known precision at every node. The TTA takes advantage of the availability of this global time to precisely specify the interfaces among the nodes, to simplify the communication and agreement protocols, to perform prompt error detection, and to guarantee the timeliness of real-time applications.

Research work in the field of distributed dependable real-time computer architectures for safety-critical applications started more than 30 years ago with the design of the STAR computer [2] and the Software Implemented Fault Tolerance [3] and Fault Tolerant Multiprocessor [4] projects. These projects were carefully evaluated and gave rise to new designs about ten years later: Fault-Tolerant Parallel Processors [5], Multicomputer Architecture for Fault Tolerance [6], and the architectural concepts of the Airbus flight control system [7]. In 1992 the first paper on SAFEbus [8], the architecture that was later deployed in the Boeing 777 aircraft for flight control, became available. In excellent publications by Lala [9], Avizienis [10], Rechtin [11] and Laprie [12], the fundamental concepts and architectural principles for the design of dependable systems are clarified at about that time. For example, Lala states that field experience with approximate voting was not at all satisfying. At about the same time, a heated debate started concerning the cost efficiency of design diversity for the tolerance of design faults [13]–[15]. The important ARINC 178B standard [16], published in 1992, that deals with software development for safety-critical avionics systems contains no clear statement about the use of software design

Manuscript received December 20, 2001; revised August 31, 2002. This work was supported in part by the European Information Society Technologies projects NEXT TTA, Fault Injection for Time-Triggered Architectures, Systems Engineering for Time-Triggered Architectures, and Dependable Systems of Systems; in part by the Time-Triggered Sensor Bus project of the government of Austria; and in part by the Defense Advanced Research Projects Agency projects Model-Based Integration of Embedded Software and Networked Embedded Software Technology.

The authors are with the Vienna University of Technology, A-1040 Vienna, Austria (e-mail:hk@vmars.tuwien.ac.at; gue@vmars.tuwien.ac.at).

Digital Object Identifier 10.1109/JPROC.2002.805821

diversity. This issue has not been resolved until today. In Europe, DELTA 4 [17], a research project funded by the European Strategic Programme for Research in Information Technology (ESPIRIT), investigated fundamental issues in the design of distributed dependable architectures at the beginning of the 1990s and uncovered a number of fundamental concepts concerning state recovery in distributed systems. Although the research community at that time was in agreement that a conscientious architectural design phase that establishes the architectural style is of utmost importance for the development of large dependable distributed real-time systems, industrial praxis took a different view. The General Accounting Office's report [18] about the experiences with the air-traffic control project, presumably the largest distributed real-time system project of its time, paints a vivid picture of the practice of system development in that period.

Amid all these research activities, the work on the TTA started in 1979 at the Technical University of Berlin with the Maintainable Architecture for Real-Time Systems (MARS) project. A first report on the MARS project [19] appeared in 1982 and was later published at the IEEE's 15th International Symposium on Fault-Tolerant Computing in 1985 [20]. After 1982, different versions of the MARS architecture have been implemented at the Vienna University of Technology [21], [22], and it became clear that a hardware-supported fault-tolerant clock synchronization is a fundamental building block of a TTA. At about that time, the important concept of temporal accuracy of real-time information was introduced by Kopetz and Kim [23], [24]. The TTP/C protocol, which includes a clock synchronization service and a membership service, was first published in 1993 [25]. A prototype version of the TTA, including a new clock synchronization chip [26] was built in the context of the European Predictably Dependable Computing Systems project. This new prototype implementation has been subject to extensive fault injection experiments [27], [28]. From these experiments, it became evident that an independent guardian must be implemented in order to avoid "babbling idiot" failures in a distributed safety-critical system based on shared communication channels. In 1995, a research cooperation with DaimlerChrysler resulted in an industrial "proof of concept" by demonstrating a time-triggered protocol–equipped "brake-by-wire" car by 1997 [29]. In 1998, the first TTP/C communication controller chip, developed with the support of the European ESPRIT project TTA, was finished. Also in 1998, a high-tech spinoff company of the Vienna University of Technology was founded with the mission to further develop and market the time-triggered (TT) technology [30]. In 1999, Alcatel investigated the TTA and decided to use it in safety-critical train control applications. In 2000, Honeywell selected the TTA for flight control, and Audi decided to use the TTA in future "drive-by-wire" applications. In the last few years, the TT technology received increasing attention for the design of safety-critical real-time applications. A number of new time-triggered protocols, in addition to SAFEbus and TTP, have recently been published: TTCAN [31], FlexRay [32], and Spider [33]. An excellent recent survey by Rushby [34], [35] contains a comparison of some of these communication architectures.

This paper on the TTA is organized as follows. Section II deals with the architectural model of the TTA, presents the model of a sparse time base, elaborates on the important concept of temporal accuracy of real-time information, and discusses the fundamental differences between the event-triggered and TT view of reality. Section III presents the principles that guided the design of the TTA: the provision of a consistent distributed computing base, the unification of interfaces and the temporal firewall concept, composability in the domains of value and time, scalability, and openness to the integration of legacy systems and the information infrastructure, and the transparent implementation of fault tolerance in order to control the application software complexity in fault-tolerant real-time systems. Section IV deals with the communication infrastructure of the TTA: the TTP/C protocol, the TTP/A protocol, and the implementation of event channels on top of the basic TT communication service. Section V is devoted to the issue of transparent implementation of fault tolerance. Finally, Section III presents the two-phase design methodology of the TTA and discusses the architecture from the point of view of validation. The paper finishes with a conclusion in Section VII.

II. Architecture Model

The computational model that guides the design of the TTA is the TT model of computation [36].

A. Model of Time

The model of time of the TTA is based on Newtonian physics. Real time progresses along a dense timeline, consisting of an infinite set of instants, from the past to the future. A duration (or interval) is a section of the timeline, delimited by two instants. A happening that occurs at an instant (i.e., a cut of the timeline) is called an event. An observation of the state of the world is thus an event. The time stamp of an event is established by assigning the state of the node-local global time to the event immediately after the event occurrence. A fault-tolerant internal clock synchronization algorithm establishes the global time in the TTA. Owing to the impossibility of synchronizing clocks perfectly and the denseness property of real time, there is always the possibility of the following sequence of events: clock in node j ticks, event e occurs, clock in node k ticks. In such a situation, the single event e is time-stamped by the two clocks j and k with a difference of one tick. In a distributed system, the finite precision of the global time base and the digitalization of time make it—in general—impossible to consistently order events on the basis of their global time stamps. The TTA solves this problem by the introduction of a sparse time base [37, p. 55]. In the sparse-time model, the continuum of time is partitioned into an infinite sequence of alternating durations of activity and silence, as shown in Fig. 1. The duration of the activity interval, i.e., a granule of the global time, must be larger than the precision of the clock synchronization.

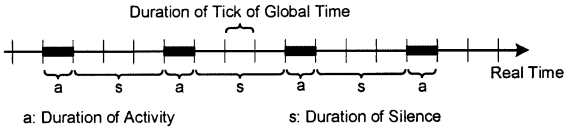

Fig. 1 Sparse time base.

From the point of view of temporal ordering, all events that occur within an interval of activity are considered to happen at the same time. Events that happen in the distributed system at different nodes at the same global clock tick are thus considered simultaneous. Events that happen during different durations of activity and that are separated by the required interval of silence can be consistently temporally ordered on the basis of their global time stamps. The architecture must make sure that significant events, such as the sending of a message, occur only during an interval of activity. The time stamps of events that are outside the control of the distributed computer system (and therefore happen on a dense timeline) must be assigned to an agreed duration of activity by an agreement protocol.

In the TTA, there exists a uniform external representation of time that is modeled according to the global positioning system (GPS) time representation. The time stamp of an instant is represented in an eight-byte integer, i.e., two words of a 32-bit architecture. The three lower bytes contain the binary fractions of the second, giving a granularity of about 60 ns. This is the accuracy that can be achieved with a precise GPS receiver. The five upper bytes count the full seconds. The external TTA epoch assigns the value 2^{38} to the start of the GPS epoch, i.e., 00:00:00 Coordinated Universal Time on January 6, 1980. This offset has been chosen in order that also instants before January 6, 1980 can be represented by positive integers in the TTA. Thus, events that occurred between 8710 years before January 1980 and 26 131 years after January 1980 can be time-stamped with an accuracy of 60 ns. There are different internal time representations in the TTA that match the time format to the capabilities of the hardware (8-, 16-, or 32-bit architectures) and the requirements of the application. Since not all time stamps are based on a global time with a precision of 60 ns, an attribute field is introduced in the external representation indicating the precision of a time stamp [38].

B. Time and State

In abstract system theory, the notion of state is introduced in order to separate the past from the future [39, p. 45]: "The state enables the determination of a future output solely on the basis of the future input and the state the system is in. In other words, the state enables a "decoupling" of the past from the present and future. The state embodies all past history of a system. Knowing the state "supplants" knowledge of the past. Apparently, for this role to be meaningful, the notion of past and future must be relevant for the system considered."

Taking this view, it follows that the notions of state and time are inseparable. If an event that updates the state cannot be said to coincide with a well-defined tick of a global clock on a sparse time base, then the notion of a systemwide state becomes diffuse. It is not known whether the state of the system at a given clock tick includes this event or not. The sparse time base of the TTA, explained previously, makes it possible to define a systemwide notion of time, which is a prerequisite for an indisputable borderline between the past and the future, and thus the definition of a systemwide distributed state. The "interval of silence" on the sparse time base forms a systemwide consistent dividing line between the past and the future and the interval when the state of the distributed system is defined. Such a consistent view of time and state is very important if fault tolerance is implemented by replication, where faults are masked by voting on replicated copies of the state residing in different fault containment regions. If there is no global sparse time base available, one often recourses to a model of an abstract time that is based on the order of messages sent and received across the interfaces of a node. If the relationship between the physical time and the abstract time remains unspecified, then this model is imprecise whenever this relationship is relevant. For example, it may be difficult in such a model to determine the precise state of a system at an instant of physical time at which voting on replicated copies of the distributed state must be performed.

C. Real-Time Entities and Real-Time Images

In the TT model, a distributed real-time computer system is modeled by a set of nodes that are interconnected by a real-time communication system as shown in Fig. 2. All nodes have access to the global time. A node consists of a communication controller (CC) and a host computer. The common boundary between the CC and the host computer within a node is called the communication network interface (CNI) (the thick black line in Fig. 2), the most important interface of the TTA.

The dynamics of a real-time application are modeled by a set of relevant state variables, the real-time entities (RT entities) that change their state as time progresses. Examples of RT entities are the flow of a liquid in a pipe, the setpoint of a control loop, or the intended position of a control valve. An RT entity has static attributes that do not change during the lifetime of the RT entity, and has dynamic attributes that change with time. Examples of static attributes are the name, the type, the value domain, and the maximum rate of change. The value set at a particular instant is the most important dynamic attribute. Another example of a dynamic attribute is the rate of change at a chosen instant.

The information about the state of an RT entity at a particular instant is captured by the notion of an observation. An observation is an atomic data structure

$$\text{Observation} = \langle \text{Name, Value, } t_{\text{obs}} \rangle$$

consisting of the name of the RT entity, the instant when the observation was made (t_{obs}), and the observed value of the RT entity. A continuous RT entity can be observed at any instant while a discrete RT entity can only be observed when the state of this RT is not changing.

A real-time image (RT image) is a temporally accurate picture of an RT entity at instant t, if the duration between the

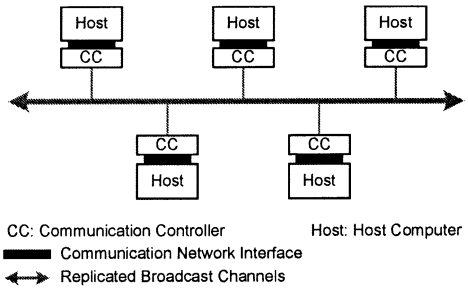

Fig. 2 Distributed real-time system with five nodes.

time of observation and the instant t is smaller than the accuracy interval d_{acc}, which is an application-specific parameter associated with the given RT entity. An RT image is thus valid at a given instant if it is an accurate representation of the corresponding RT entity, both in the value and the time domains [23]. While an observation records a fact that remains valid forever (a statement about an RT entity that has been observed at an instant), the validity of an RT image is time dependent and is invalidated by the progression of real time.

At the CNIs within a node, the pictures of the RT entities are periodically updated by the real-time communication system to establish temporally accurate RT images of the RT entities. The computational tasks within the host of a node take these temporally accurate RT images as inputs to calculate the required outputs within an *a priori*–known worst case execution time. The outputs of the host are stored in the CNI and transported by the TT communication system to the CNIs of other nodes at *a priori*–determined instants. The interface nodes transform the received data to/from the representation required by the controlled object or the human operator and activate control actions in the physical world.

D. State Information vs. Event Information

The information that is exchanged across an interface is either state information or event information, as explained in the following paragraphs. Any property of an RT entity (i.e., a relevant state variable) that is observed by a node of the distributed real-time system at a particular instant, e.g, the temperature of a vessel, is called a state attribute and the corresponding information state information. A state observation records the state of a state variable at a particular instant, the point of observation. A state observation can be expressed by the atomic triple

$$\langle \text{Name of variable, value, time of observation} \rangle.$$

For example, the following is a state observation: "The position of control valve A was at 75 ° at 10:42 A.M." State information is idempotent and requires an at-least-once semantics when transmitted to a client. At the sender, state information is not consumed on sending, and at the receiver, state information requires an update in place and a nonconsuming read. State information is transmitted in state messages.

A sudden change of state of an RT entity that occurs at an instant is an event. Information that describes an event is called event information. Event information contains the difference between the state before the event and the state after the event. An event observation can be expressed by the atomic triple

$$\langle \text{Name of variable, value difference, time of event} \rangle.$$

For example, the following is an event observation: "The position of control valve A changed by 5 ° at 10:42 A.M." Event observations require exactly once semantics when transmitted to a consumer. At the sender, event information is consumed on sending and at the receiver, event information must be queued and consumed on reading. Event information is transmitted in event messages.

Periodic state observations or sporadic event observations are two alternative approaches for the observation of a dynamic environment in order to reconstruct the states and events of the environment at the observer [40]. Periodic state observations produce a sequence of equidistant "snapshots" of the environment that can be used by the observer to reconstruct those events that occur within a minimum temporal distance that is longer than the duration of the sampling period. Starting from an initial state, a complete sequence of (sporadic) event observations can be used by the observer to reconstruct the complete sequence of states of the RT entity that occurred in the environment. However, if there is no minimum duration between events assumed, the observer and the communication system must be infinitely fast.

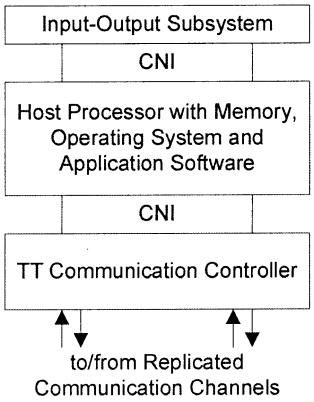

Fig. 3 Node of the TTA.

E. Structure of the TTA

The basic building block of the TTA is a node. A node comprises in a self-contained unit (possibly on a single silicon die) a processor with memory, an input–output subsystem, a TT communication controller, an operating system, and the relevant application software as depicted in Fig. 3.

Two replicated communication channels connect the nodes, thus forming a cluster. The cluster communication system (the gray-shaded area in Fig. 4) comprises the physical interconnection network and the communication controllers of all nodes of the cluster. In the TTA, the communication system is autonomous and executes periodically an *a priori*–specified time-division multiple access (TDMA) schedule. It reads a state message from the CNI at the sending node at the *a priori*–known fetch instant and delivers it to the CNIs of all other nodes of the cluster at the *a priori*–known delivery instant, replacing the previous

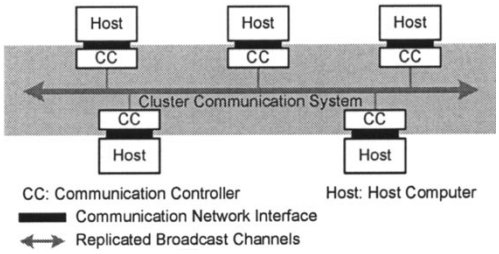

Fig. 4 Structure of a TTA cluster.

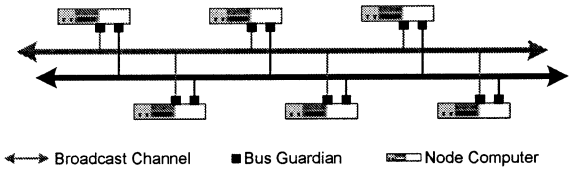

Fig. 5 Topology of TTA-bus.

version of the state message. The times of the periodic fetch and delivery actions are contained in the message scheduling table [the message descriptor list (MEDL)] of each communication controller.

Clusters can be connected by gateway nodes (Node D in Fig. 10). A gateway node is a member of two clusters and therefore contains two CNIs, as explained in Section III-D. A gateway node restricts the view of one cluster as seen by the other cluster in order to reduce complexity.

F. Interconnection Topology

The TTA distinguishes between two different physical interconnection topologies within a cluster, TTA-bus (Fig. 5) and TTA-star (Fig. 6).

In TTA-bus, the physical interconnection consists of replicated passive buses. At every physical node site there are three subsystems: the node and two guardians. The guardians are independent units that monitor the known temporal behavior of the associated node. If a node intends to send a message outside its *a priori*–determined time slot, the guardian will cut off the physical transmission path and thus eliminate this failure mode. Ideally, the guardians must be completely independent units with their own clock, power supply, and distributed clock synchronization algorithms. Furthermore, the guardians should be at a physical distance from the node they protect in order that the system becomes resilient to spatial-proximity faults. If all these requirements are implemented in TTA-bus, a complete TTA-bus node (including the guardians) must be composed of three independent chip packages (to be resilient to physical proximity faults), three independent clocks and three independent power supplies. This is quite expensive for mass-market applications. To reduce the implementation costs, the guardians are implemented on the same die as the node in the prototype implementation [41]. This is sufficient for fail-safe operation, since the TTA contains algorithms for the detection of the violation of the fault hypothesis (fail-silent nodes in TTA-bus) and can bring the application into the safe state in case the fault-hypothesis is violated. However, for fail-operational applications, the TTA-star

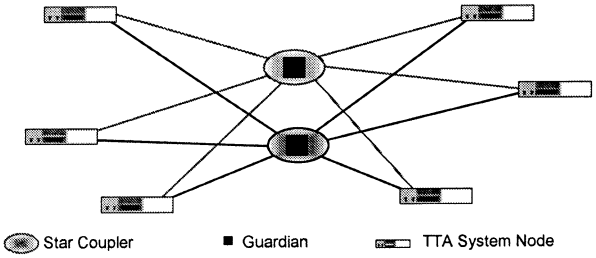

Fig. 6 Topology of TTA-star.

interconnection that tolerates arbitrary (i.e., byzantine) node faults is recommended [42], [43].

In the TTA-star configuration, the guardians are integrated into two replicated central star couplers as depicted in Fig. 6. This has the following advantages.

1) The guardians are fully independent and located at a physical distance from the nodes they protect.
2) In a cluster comprising n nodes, only $n+2$ (instead of $3n$ in TTA-bus when tolerating the same class of node failures) packages are needed.
3) The algorithms in the guardians can be extended to provide additional monitoring services, such as condition-based maintenance.
4) If the guardians reshape the physical signals, the architecture becomes resilient to arbitrary [e.g., slightly-off-specification (SOS)] node faults.
5) Point-to-point links have better electromagnetic interference characteristics than a bus and can easily be implemented on fiber optics.

III. Design Principles

The following sections will discuss the principles that guided the design of the TTA.

A. Consistent Distributed Computing Base

The main purpose of the TTA is to provide a consistent distributed computing base to all correct nodes in order that reliable distributed applications can be built with manageable effort. If a node cannot be certain that every other correct node works on exactly the same data, then the design of distributed algorithms becomes very cumbersome [44] because the intricate agreement problem has to be solved at the application level. The TTA exploits the short error-detection latency of a time-triggered protocol to perform immediate error detection at the protocol level and continuously executes a distributed agreement (membership) algorithm to determine if any node has been affected by a failure. By checking the membership of the nodes that are participating in a distributed application, an application at a particular node can make sure that all other nodes are correctly participating in the joint action.

The simple "brake-by-wire" system in a car (see Fig. 7) demonstrates the importance of a consistent membership view in a distributed real-time application. In this application, the four nodes that control the brakes at the four wheels of a car are connected by a fault-tolerant communication system. The R-Front and the L-Rear node accept the brake pedal pressure from one fail-silent brake pedal sensor, the

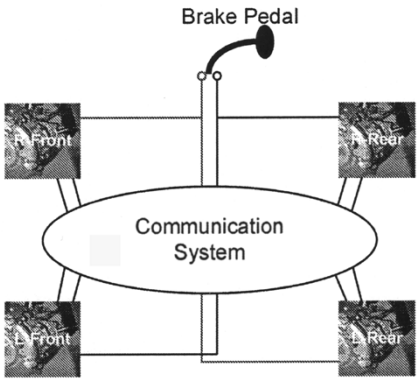

Fig. 7 Simple "brake-by-wire" application.

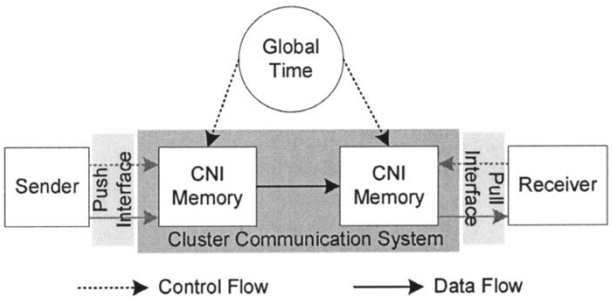

Fig. 8 Data flow and control flow at a TTA interface.

L-Front and R-Rear node accept the brake pedal pressure from the other fail-silent brake pedal sensor. Every wheel node informs all other nodes about its view of the brake pedal sensors, performs a distributed algorithm to allocate the brake force to each wheel, and controls the brake at its local wheel. The brake is assumed to be designed in a way that the brake autonomously visits a defined state (e.g., wheel free running—no brake force applied) in case the wheel node crashes or the electrical or mechanical mechanism in the local brake fails. As soon as the other three wheels learn about the failure at one wheel, they redistribute the brake force between them in a way that the car is stopped safely with three braking wheels. The time interval between the instant of brake failure and the instant of redistribution of the brake force, i.e., the error-detection interval, is a safety-critical parameter of this application. During this error-detection interval, the braking system is in an inconsistent state. We conjecture that there is a potential for a fatal accident if this inconsistent state is not detected and corrected within at most a few sampling intervals. Consider the scenario where the R-Rear node has an outgoing link failure. In this scenario, the other three nodes will assume the R-Rear node has failed (since they do not receive any message from the R-Rear node), while the R-Rear node will think it is operating correctly, since it receives all messages from the other nodes. This scenario illustrates the need for a distributed membership protocol to detect and eliminate safety-relevant inconsistencies in a distributed real-time system.

If the fault hypothesis of the TTA is violated and the distributed agreement protocol cannot achieve a consistent view, the TTA activates its clique avoidance algorithm to inform the application of the grave situation. In such a situation, it is up to the application to decide how to proceed: to perform a rapid restart thus reestablishing consistency as soon as possible (e.g., if a massive transient is assumed to have been the cause of the problem), or to continue with inconsistent data (which is not recommended).

B. Unification of Interfaces—Temporal Firewalls

A suitable architecture must be based on a small number of orthogonal concepts that are reused in many different situations in order to reduce the mental load required for understanding large systems. In a large distributed system, the characteristics of these interfaces between the identified subsystems determine to a large extent the comprehensibility of the architecture. In the TTA, the CNI (cf. Fig. 3) between a host computer and the communication network is the most important interface. The CNI appears in every node of the architecture and separates the local processing within a node from the global interactions among the nodes. The CNI consists of two unidirectional data-flow interfaces, one from the host computer to the communication system and the other one in the opposite direction.

We call a unidirectional data-flow interface elementary if there is only a unidirectional control flow [45] across this interface. An interface that supports periodic state messages with error detection at the receiver is an example of such an elementary interface. We call a unidirectional data-flow interface composite if even a unidirectional data flow requires a bidirectional control flow. An event message interface with error detection is an example for a composite interface. Composite interfaces are inherently more complex than elementary interfaces, since the correct operation of the sender depends on the control signals from all receivers. This can be a problem in multicast communication where many control messages are generated for every unidirectional data transfer, and each one of the receivers can affect the operation of the sender. Multicast communication is common in distributed embedded systems.

The basic TTA CNI as depicted in Fig. 8 is an elementary interface. The TT transport protocol carries autonomously—driven by its TT schedule—state messages from the sender's CNI to the receiver's CNI. The sender can deposit the information into its local CNI memory according to the information push paradigm, while the receiver will pull the information out of its local CNI memory. From the point of view of temporal predictability, information push into a local memory at the sender and information pull from a local memory at the receiver are optimal, since no unpredictable task delays that extend the worst-case execution occur during reception of messages. A receiver that is working on a time-critical task is never interrupted by a control signal from the communication system. Since no control signals cross the CNI in the TTA (the communication system derives control signals for the fetch and delivery instants from the progress of global time and its local schedule exclusively—cf. Section II-E), propagation of control errors is prohibited by design. We

call an interface that prevents propagation of control errors by design a temporal firewall [46]. The integrity of the data in the temporal firewall is assured by the nonblocking write concurrency control protocol [47].

From the point of view of complexity management and composability, it is useful to distinguish between three different types of interfaces of a node: the real-time service (RS) interface, the diagnostic and maintenance (DM) interface, and the configuration planning (CP) interface [48]. These interface types serve different functions and have different characteristics. For the temporal composability, the most important interface is the RS interface.

The RS Interface: The RS interface provides the timely RSs to the node environment during the operation of the system. In real-time systems it is a time-critical interface that must meet the temporal specification of the application in all specified load and fault scenarios. The composability of an architecture depends on the proper support of the specified RS interface properties (in the value and in the temporal domain) during operation. From the user's point of view, the internals of the node are not visible at the CNI, since they are hidden behind the RS interface.

The DM Interface: The DM interface opens a communication channel to the internals of a node. It is used for setting node parameters and for retrieving information about the internals of the node, e.g., for the purpose of internal fault diagnosis. The maintenance engineer that accesses the node internals by the DM interface must have detailed knowledge about the internal objects and behavior of the node. The DM interface does not affect temporal composability. Usually, the DM interface is not time-critical.

The CP Interface: The CP interface is used to connect a node to other nodes of a system. It is used during the integration phase to generate the "glue" between the nearly autonomous nodes. The use of the CP interface does not require detailed knowledge about the internal operation of a node. The CP interface is not time critical.

The CNI of the TTA can be directly used as the RS interface. On input, the precise interface specifications (in the temporal and value domain) are the temporal preconditions for the correct operation of the host software. On output, the precise interface specifications are the temporal postconditions that must be satisfied by the host, provided the preconditions have been satisfied by the host environment. Since the bandwidth is allocated statically to the host, no starvation of any host can occur due to high-priority message transmission from other hosts. As will be explained in Section IV-C, an event-triggered communication service is implemented on top of the basic TT service in the TTA to realize the DM and CP interfaces. Since the event-triggered communication is based on but not executed in parallel to the TT communication, it is possible to maintain and to use all predictability properties of the basic TT communication service in event-triggered communication.

C. Composability

In a distributed real-time system, the nodes interact by the communication system to provide the emerging RSs. These emerging services depend on the timely provision of the real-time information at the RS interfaces of the nodes. For an architecture to be composable in the temporal domain, it must adhere to the following four principles with respect to the RS interfaces:

1) independent development of nodes;
2) stability of prior services;
3) constructive integration of the nodes to generate the emerging services;
4) replica determinism.

Independent Development of Nodes: A composable architecture must meticulously distinguish between architecture design and node design. Principle 1) of a composable architecture is concerned with design at the architecture level. Nodes can be designed independently of each other only if the architecture supports the precise specification of all node services at the level of architecture design. In a real-time system, the RS interface specification of a node must comprise the precise CNI specification in the value domain and in the temporal domain and a proper abstract model of the node service, as viewed by the host of the node. Only then the node designer will be in the position to know exactly what can be expected from the environment at which time and what must be when delivered to the environment by the node. This knowledge is a prerequisite for the independent development of the node software.

Stability of Prior Services: Principle 2) of a composable architecture is concerned with the design at the node level. A node is a nearly autonomous subsystem that comprises the hardware, the operating system, and the application software. The node must provide the intended services across the well-specified node interfaces. The design of the node can take advantage of any established software engineering methodology, such as object-based design methods. The stability-of-prior-service principle ensures that the validated service of a node—both in the value domain and in the time domain—is not refuted by the integration of the node into a system. For example, the integration of a self-contained node, e.g., an engine controller (cf. Fig. 14), into the integrated vehicle control system may require additional computational resources (both in processing time and in memory space) of the node to service this new communication interface. Consider the case where the new communication interface contains a queue of messages that must be serviced by the host computer: memory space for the queue must be allocated by the node-local operating system, and processing time of the host processor for the management of the queue must be made available. In a rare-event situation, it may happen that these additional resource requirements that are needed for the timely interface service are in conflict with the resource requirements of the time-critical application software that implements the prior services of the node. In such a situation, failures in the node's prior services may occur sporadically during and after the integration. This is one reason why the TTA requires information-pull interfaces at the receiver.

Constructive Integration: Principle 3) of a composable architecture is concerned with the design of the communi-

cation system. Normally, the integration of the nodes into the system follows a step-by-step procedure. The constructive integration principle requires that if n nodes are already integrated, the integration of the $n+1$ node must not disturb the correct operation of the n already integrated nodes. The constructive-integration principle ensures that the integration activity is linear and not circular.

This constructive integration principle has severe implications for the management of the network resources. If network resources are managed dynamically, it must be ascertained that even at the critical instant, i.e., when all nodes request the network resources at the same instant, the timeliness of all communication requests can be satisfied. Otherwise, sporadic failures will occur with a failure rate that is increasing with the number of integrated nodes.

For example, if a RS requires that the network delay must always remain below a critical upper limit (because otherwise a local time-out within the node may signal a communication failure) then the dynamic extension of the network delay by adding new nodes may be a cause of concern. In a dynamic network the message delay at the critical instant (when all nodes request service at the same time) increases with the number of nodes. The system of Fig. 9 will work correctly with up to four nodes. The addition of the fifth node may lead to sporadic failures.

Replica Determinism: If fault tolerance is implemented by the replication of nodes, then the architecture and the nodes must support replica determinism. A set of replicated nodes is replica determinate [49] if all the members of this set have the same externally visible state, and produce the same output messages at points in time that are at most an interval of d time units apart (as seen by an omniscient outside observer). In a fault-tolerant system, the time interval d determines the time it takes to replace a missing message or an erroneous message from a node by a correct message from redundant replicas. The implementation of replica determinism is simplified if all nodes have access to a globally synchronized sparse time base. Replica determinism also decreases the testing and debugging effort significantly by eliminating Heisenbugs [50] by design.

D. Scalability

The TTA is intended for the design of very large distributed real-time applications. A large system can only be constructed if the mental effort required to understand a particular system function is independent of the system size. Despite the fact that a large system will support many more functions than a small system, the complexity of each individual function must not increase with the growth of the system. Horizontal layering (abstraction) and vertical layering (partitioning) are the means to combat the complexity of large systems. In the TTA, the CNIs encapsulate a function and make only those properties of the environment visible to this encapsulated function that are relevant for the correct operation of the function. This is a powerful application of these principles of partitioning and abstraction. Subsystems consisting of many nodes are connected by gateway nodes that provide a constrained view of each

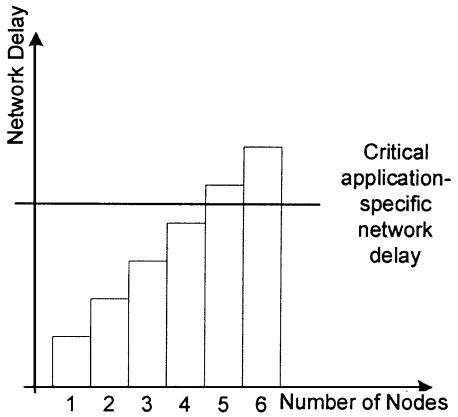

Fig. 9 Maximum network delay at critical instant as a function of the number of nodes.

subsystem—only those data elements that are needed for the emerging functions resulting from the cooperation of the subsystems are made available at the CNIs of the gateway node. There is no other central element in the TTA besides the global notion of time. If node *D* of Cluster 1 of Fig. 10 faces its computational limits, it can be expanded into a gateway node. The interface to Cluster 1 of the gateway node *D* remains unchanged—in value and time—while the internal processing of node *D* is now redistributed to the nodes *X*, *Y*, and *Z* of cluster 2. Seen from Cluster 2, the gateway node *D* provides only a limited view to the functions of Cluster 1 in order to restrict the complexity growth.

Gateway nodes can also be used to integrate legacy systems. Assume that Cluster 1 of Fig. 10 is a legacy system and Cluster 2 is a new system that is developed according to a new architectural style, different from that of Cluster 1. Gateway node *D* can act as an interface system [51], and reconcile the property mismatches between the architectural styles of Cluster 1 and Cluster 2.

Maintenance and diagnostics require a focused view inside a subsystem to interrogate the correct operation of its low-level mechanisms. If all these internals were exposed at the subsystem boundaries, the complexity of a large system would explode. To solve this problem, the TTA supports facilities to build focused event-message channels into the internals of a selected subsystems by using the DM interface, similar to the boundary scan techniques used in the design of complex very large scale integration (VLSI) chips. A maintenance engineer who has detailed knowledge about the internals of a particular subsystem can thus look into the selected subsystem without exposing the encapsulated subsystem-internal information at the subsystem boundaries.

E. Transparent Implementation of Fault Tolerance

The TTA is intended for safety-critical real-time applications such as the control of an aircraft or “brake-by-wire” systems in automobiles. Active redundancy by replication and voting are the most appropriate fault-tolerance techniques for meeting these requirements. The realization of active replication demands a number of mechanisms, such

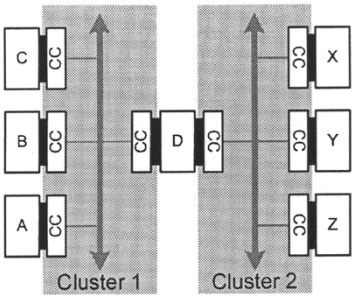

Fig. 10 Expansion of node *D* of cluster 1 into a new cluster 2.

as replica coordination, voting, membership fusion, internal state alignment, and reintegration of nodes after a transient failure. If these generic fault-tolerance mechanisms are intertwined with the application software of a node, then the resulting increase of the software complexity can be the cause of additional design faults.

In the TTA, the fault-tolerance mechanisms are implemented in a dedicated fault-tolerance layer (Fig. 11) possibly with its own middleware processor, such that the fault-tolerant CNI (FTU CNI) is identical in structure and timing to the basic nonfault-tolerant CNI. A properly structured application software can thus operate in a fault-tolerant system or a nonfault-tolerant system without any modifications. The fault-tolerance mechanisms remain transparent to the application in the TTA.

F. Openness

Large distributed real-time systems do not operate in isolation. They must be integrated into the global information infrastructure. The standardization of interfaces of the TTA by the Object Management Group (OMG) in order that TTA internal data can be accessed from any Common Object Request Broker Architecture (CORBA)–compliant client is in progress [38]. This interface standard proposal makes the three TTA interfaces, the RS interface, the DM interface and the CP interface available at a CORBA object request broker. The RS interface provides the real-time data with known delay and bounded jitter in order to support distributed control applications. The DM and CP interfaces are event-triggered interfaces that open event channels into the internals of a node for the purpose of maintenance and dynamic reconfiguration. Provided that the CORBA security clearance is passed, it is thus possible to investigate remotely (by the Internet) the internals of every TTA node while the system is delivering its RS.

IV. Communication

The instants at which information is delivered at or fetched from the CNIs of a node are determined *a priori* and are common knowledge to all nodes of a TTA. *A priori* common knowledge means that these instants are defined before the computation under consideration is started and that the instants are known to all nodes of a cluster beforehand. These instants are the deadlines for the application tasks within a host. Knowing these deadlines, it is in the responsibility of

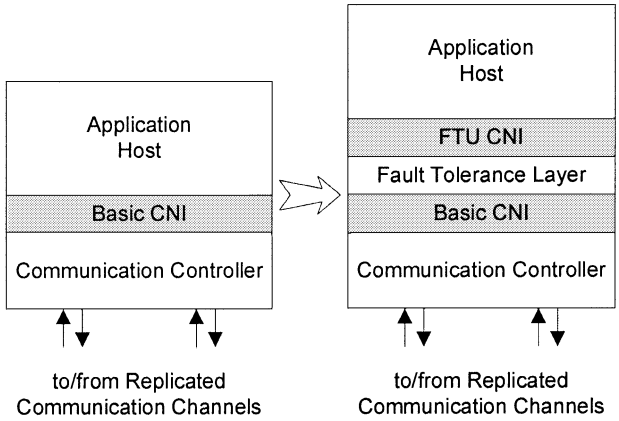

Fig. 11 Expansion of a nonfault-tolerant node into a fault-tolerant node.

the host to produce the required results before the deadline has passed. Any node-local scheduling strategy that will satisfy these known deadlines is "fit for purpose." It is the responsibility of the TT communication service to transport the information from the sending CNI to the receiving CNI within the interval delimited by these *a priori*–known fetch and delivery instants. The TTA contains two communication protocols that provide this communication service: the fault-tolerant TTP/C protocol and the low-cost fieldbus protocol TTP/A.

A. The TTP/C Protocol

The TTP/C protocol [25], [52] is a fault-tolerant time-triggered protocol that provides the following services.

1) Autonomous fault-tolerant message transport with known delay and bounded jitter between the CNIs of the nodes of a cluster by employing a TDMA medium access strategy on replicated communication channels.
2) Fault-tolerant clock synchronization that establishes the global time base without relying on a central time server.
3) Membership service to inform every node consistently about the "health-state" of every other node of the cluster. This service can be used as an acknowledgment service in multicast communication. The membership service is also used to efficiently implement the fault-tolerant clock synchronization service.
4) Clique avoidance to detect and eliminate the formation of cliques in case the fault hypothesis is violated.

In TTP/C, the communication is organized into rounds, where every node must send a message in every round. A particular message may carry up to 240 bytes of data. The data is protected by a 24–bit CRC checksum. The message schedule is stored in the MEDL within the communication controller of each node. To achieve high data efficiency, the sender name and the message name is derived from the send instant. The clock synchronization of TTP/C exploits the common knowledge of the send schedule: every node measures the difference between the *a priori*–known expected and the actually observed arrival time of a correct message to learn

about the difference between the sender's clock and the receiver's clock. This information is used by a fault-tolerant average algorithm to calculate periodically a correction term for the local clock in order to keep the clock in synchrony with all other clocks of the cluster. The membership service employs a distributed agreement algorithm to determine whether the outgoing link of the sender or the incoming link of the receiver has failed. Nodes that have suffered a transmission fault are excluded from the membership until they restart with a correct protocol state. Before each send operation of a node, the clique avoidance algorithm checks if the node is a member of the majority clique. The detailed specification of the TTP/C protocol can be found in [52].

B. The TTP/A Protocol

The TTP/A protocol is the TT fieldbus protocol of the TTA [53], [53], [38]. It is used to connect low-cost smart transducers to a node of the TTA, which acts as the master of a transducer cluster. In TTP/A, the CNI memory element of Fig. 8 has been expanded at the transducer side to hold a simple interface file system (IFS). Each interface file contains 256 records of four bytes each. The IFS forms the uniform name space for the exchange of data between a sensor and its environment (Fig. 12).

The IFS holds the real-time data, calibration data, diagnostic data, and configuration data. The information between the IFS of the smart transducer and the CNI of the TTA node is exchanged by the TT TTP/A protocol, which distinguishes between two types of rounds, the master-slave (MS) round and the multipartner (MP) round. The MS rounds are used to read and write records from the IFS of a particular transducer to implement the DM and CP interface. The MP rounds are periodic and transport data from selected IFS records of several transducers across the TTP/A cluster to implement the RS. MP rounds and MS rounds are interleaved, such that the time-critical RS and the event-based DM and CP service can coexist. It is thus possible to diagnose a smart transducer or to reconfigure or install a new smart transducer on-line, without disturbing the time-critical RS of the other nodes. The TTP/A protocol also supports a "plug-and-play" mode where new sensors are detected, configured, and integrated into a running system on-line and dynamically. The detailed specification of the TTP/A protocol can be found in [38].

C. Event Message Channels

In the TTA, event message channels are constructed on top of the basic TT communication service by assigning an *a priori*–specified number of bytes of selected TT messages to the event-triggered message transport service. These periodically transmitted bytes form a dedicated communication channel for the transmission of the dynamically generated event information. To implement the event semantics (cf. Section II-D) at the sender and receiver, two message queues must be provided in the CNIs: the sender queue at the sender's CNI and the receiver queue at the receiver's CNI. The sender pushes a newly produced event message into the sender queue, while the receiver must check the receiver queue to pull and consume the event message. An alternative design could produce an interrupt whenever a new event message arrives at the receiver, but such a design would violate the TTA principle of providing an information pull interface at the receiver and interfere with the principle of stability of prior services. Since in a cluster with n nodes every transmitted event message generates an event message at every receiver (i.e., a total of $n-1$ event messages in the distributed system), two additional services are provided to avoid a queue overflow at the receiver: a filter service and a garbage collection service. The filter service selects the incoming event messages according to filtering criteria established by the receiver and accepts only those event messages that pass the filter. The garbage collection service eliminates decayed event messages from the receiver queue based on the age of the message. A maximum queue storage duration must be statically assigned to each event message for this purpose. After this duration has elapsed, the message is eliminated from the receiver queue. The event message channels are used in the TTA to implement the nontime-critical DM and CP services. It is possible to implement widely used event-based protocols, such as Transmission Control Protocol/Internet Protocol or Controller Area Network, on the TTA event channels.

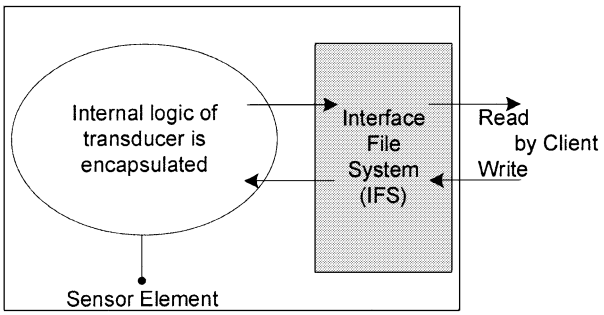

Fig. 12 Interface file system in a smart transducer.

Event message channels should not be used for time-critical or safety-critical functions. In case of a rare-event peak-load scenario, the event message service may be delayed or stopped in order to maintain the safety-critical TT service. It follows that the host tasks servicing the event channels can be scheduled according to the "best-effort" paradigm. Care must be taken that any software interaction between the event service and the safety-critical TT service inside the application software of the host is fully understood and no negative consequences on the replica determinism of the TT service can occur.

D. Performance Limits

As in any distributed computing system, the performance of the TTA depends primarily on the available communication bandwidth and computational power. In this Section we intend to investigate the temporal performance limits of the TTA. Because of physical effects of time distribution and limits in the implementation of the guardians [41], a minimum interframe gap of about 5 μs must be maintained between frames to guarantee the correct operation of the

guardians. If a bandwidth utilization of about 80% is intended, then the message-send phase must be in the order of about 20 μs, implying that about 40 000 messages can be sent per second within such a cluster. With these parameters, a sampling period of about 250 μs can be supported in a cluster comprising ten nodes. The precision of the clock synchronization in current prototype systems is below one microsecond. If the interframe gap and bandwidth limits are stretched, it might be possible to implement in such a system a 100 μs TDMA round (corresponding to a 10-kHz control loop frequency), but not much smaller if the system is physically distributed (to tolerate spatial proximity faults). The amount of data that can be transported in the 20 μs window depends on the bandwidth: In a 5-Mb/s system it is about 12 bytes; in a 1-Gb/s system it is about 2400 bytes. A prototype implementation of TTP/C using Gigabit Ethernet is currently under development. This prototype implementation uses commercial off-the-shelf (COTS) hardware and therefore is not expected to achieve the limiting performance. The objective of this project is rather to determine the performance that can be achieved without special hardware and to pinpoint the performance bottlenecks to face when using COTS components.

In a universal asynchronous receiver and transmitter TTP/A configuration with ten smart transducers, each one sending one byte of information, a typical round length (MP round plus MS round) is on the order of 250 bit-cells. If a low-cost single-wire connection with a transmission speed of 20 kb/s is used, the round duration will be 12.5 ms. If a more expensive physical layer that supports 1 Mb/s is selected, then the round duration will be 250 μs, supporting a control loop with a frequency of 4 kHz. The jitter in these applications is about one-third of the bitcell, i.e., in the previous example less than one microsecond. Since the TTA communication system provides full phase control, the jitter will not increase if the control loops are cascaded.

V. Fault Tolerance

As indicated in Section III-E, the TTA supports transparent implementation of fault tolerance.

A. Fault Hypothesis

Any fault-tolerant design activity starts with the specification of the fault hypothesis. The fault hypothesis states the types and number of faults that the system should tolerate. In the TTA, it is assumed that a chip is a single fault-containment region, since all functions of a chip share a common power supply, ground, oscillator, the same mask, the same manufacturing process, and are in close physical proximity. Given the feature size of today's VLSI circuits, it can happen that a single external event, e.g., an α particle, will affect a number of logic functions simultaneously. It is thus difficult to argue that two functions on the same chip will fail independently.

In a properly configured TTA star a cluster will tolerate an arbitrary failure mode of a single TTA node (chip). A faulty unit will be consistently detected by the membership protocol and isolated within two TDMA rounds. The TTA masks an asymmetric (byzantine) fault of any node by architectural means [42] and realizes an efficient temporal and spatial partitioning of the nodes at the cluster level.

B. Fault-Tolerant Units

For internal physical faults—an important fault class in any system—the preferred fault-tolerance strategy is active replication of independent nodes. The effective partitioning of the nodes and the masking of byzantine faults by the TTA provides the prerequisites to logically group a set of nodes or software subsystems that compute the same function into a fault-tolerant unit (FTU). The FTU will tolerate the failure of any of its independent constituent parts without a degradation of service. Note that the physical positions of the nodes composing an FTU should be far apart in order to tolerate physical proximity faults. The classic form of an FTU consists of three nodes (triple modular redundancy, referred to as TMR) that operate in replica determinism and present their results to a voter (physically located at every consumer of the result) that makes a majority decision.

A necessary prerequisite for the implementation of active redundancy in the described form is the replica-deterministic behavior of the host software [54], [49]. The TTA provides replica determinism at the CNI of a node, but it is up to the host software to ensure replica determinism within the complete node. The three replicas of the host software run synchronously on the three different host computers and produce their output simultaneously (within the precision of the clock synchronization) at their FTU CNI (cf. Fig. 11) before the *a priori*–known fetch instant [55]. The FTU layer distributes the messages to the other nodes of the cluster. Immediately before the *a priori*–determined delivery instant of a message at the FTU CNI of receiving nodes, the FTU layer at the respective nodes vote on the incoming messages from the nodes of the FTU and present the majority result to their hosts at the delivery instant. Periodically, every host must output its internal state for the same voting procedure by the other nodes of the FTU. An integrating node must wait until it receives a voted internal state before it can participate in an application.

The TTA also supports implementation of self-checking pairs of nodes. Since a self-checking node will only produce a result if it is correct in the temporal domain and in the value domain, a self-checking FTU can operate with two nodes (there is no need for voting) in order to tolerate a single node failure.

The host application must be designed such that the duration between the fetch instants and the delivery instants at the CNI is long enough to perform the fault-tolerance functions. If the timing at the host computer meets this requirement, the insertion of an FTU layer will be transparent to the host, since the fetch instant and the delivery instant may remain unchanged.

C. Never-Give-Up (NGU) Strategy

The fault-tolerant service described in Section V-B can be maintained only if the environment complies with the

fault hypothesis. If the environment violates the fault hypothesis—in a properly designed application, this must be a rare event—then the TTA activates an NGU strategy. The NGU strategy is initiated by the TTP/C protocol in combination with the application as soon as it becomes evident that there are not enough resources available any more to provide the minimum required service. The NGU strategy is highly application specific. For example, if the cause of the outage is a massive transient fault, then in some applications the NGU strategy may consist of freezing the actuators in their current state until a successful restart of the whole cluster has been completed.

D. Redundant Transducers

If the transducers need to be replicated to achieve fault tolerance, then at least two independent TTP/A fieldbuses must be installed (Fig. 13). Each one of those fieldbuses is controlled by one active TTP/A master in a TTP/C gateway node of the FTU. The other TTP/A master is passive and listens to the fieldbus traffic to capture the sensor data.

An agreement protocol is executed in the controllers of the TTA nodes to reconcile the values received from the replicated sensors. Then, a single agreed value from each redundant sensor set is presented to the host software at the CNIs. On output, the replicated results are transported on separate fieldbuses to a fault-tolerant actuator.

VI. TTA Design Methodology

Composability and the associated reuse of nodes and software can only be realized if the architecture supports a two-level design methodology. In the TTA, such a methodology is supported: the TTA distinguishes between the architecture design and the node design.

A. Architecture Design

In the architecture design phase, an application is decomposed into clusters and nodes. This decomposition will be guided by engineering insight and the structure inherent in the application, in accordance with the proven architecture principle of "form follows function." For example, in an automotive environment, a "drive-by-wire" system may be decomposed into functional units as depicted in Fig. 14.

If a system is developed "on the green lawn," then a top-down decomposition will be pursued. After the decomposition has been completed, the CNIs of the nodes must be specified in the temporal and in the value domain. The data elements that are to be exchanged across the CNIs are identified and the precise fetch instants and delivery instants of the data at the CNI must be determined. Given these data, the schedules of the TTP/C communication system can be calculated and verified. At the end of the architecture design phase, the precise interface specifications of the nodes are available. These interface specifications are the inputs and constraints for the node design.

Given a set of available nodes with their temporal specification (nodes that are available for reuse), a bottom-up design approach must be followed. Given the constraints of

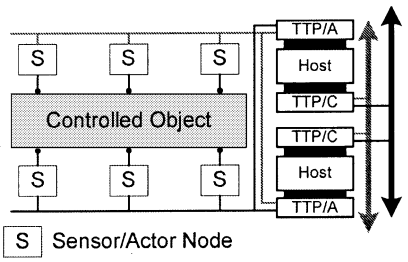

Fig. 13 Replicated fieldbuses.

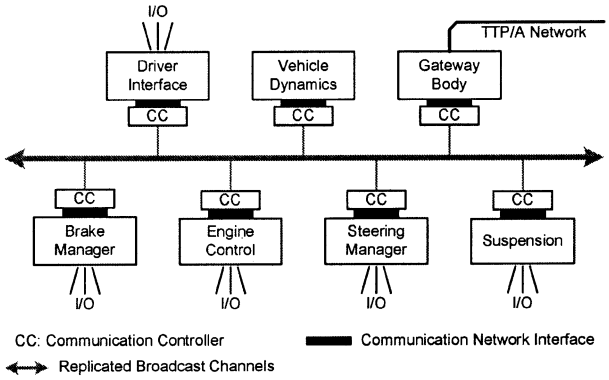

Fig. 14 Decomposition of a "drive-by-wire" application.

the nodes at hand (how much time they need to calculate an output from an input), a TTP/C schedule must be found that meets the application requirements and satisfies the node constraints.

B. Node Design

During the node design phase, the application software for the host computers is developed. The delivery and fetch instants established during the architecture design phase are the preconditions and postconditions for the temporal validation of the application software. The host operating system can employ any reasonable scheduling strategy, as long as the given deadlines are satisfied and the replica determinism of the host system is maintained.

Node testing proceeds bottom-up. A new node must be tested with respect to the given CNI specifications in all anticipated load and fault conditions. The composability properties of the TTA (stability of prior service, cf. Section III-C, achieved by the strict adherence to information pull interfaces) ensure that a property that has been validated at the node level will also hold at the system level. At the system level, testing will focus on validating the emerging services that are result of the integration.

C. Validation

Today, the integration and validation phases are probably the most expensive phases in the implementation of a large distributed real-time system. The TTA has been designed to reduce this integration and validation effort by providing the following mechanisms.

1) The architecture provides a consistent distributed computing base to the application and informs the application in case a loss of consistency is caused

by a violation of the fault hypothesis. The basic algorithms that provide this consistent distributed computing base (clock synchronization and membership) have been analyzed by formal methods and are implemented once and for all in silicon [56]–[61]. The application does not need to be concerned with the implementation and validation of the complex distributed agreement protocols that are needed to establish consistency in a distributed system.

2) The architecture is replica deterministic, which means that any observed deficiency can be reproduced in order to diagnose the cause of the observed problem.
3) The interaction pattern between the nodes and the contents of the exchanged messages can be observed by an independent observer without the probe effect [62]. It is thus possible to determine whether a node complies with its preconditions and postconditions without interfering with the operation of the observed node.
4) The internal state of a node can be observed and controlled by the DM interface.
5) In the TTA, it is straightforward to provide a real-time simulation test bench that reproduces the environment to any node in real time. Deterministic automatic regression testing can thus be implemented.

D. Design Tools

The TTA design methodology is supported by a comprehensive set of integrated design tools of TTTech AG [30]. The design engineer starts the architecture design by decomposing a cluster into nodes and by specifying the interaction patterns among the nodes: the data items that must be exchanged and the temporal constraints that must be observed. The design tool TTPplan calculates the message schedules and determines the precise fetch instant and delivery instant, i.e., the MEDL, for the CNI in each node. After the interaction pattern has been verified to meet the requirements of the application, the node design can commence. The node design takes the CNI specification developed at the architecture design phase as constraint and develops the task structure within a node. There is tool support for the automatic generation of the FTU layer, if fault tolerance is desired. In addition to the design tools there exist also download tools to download the developed software into the TTA nodes and monitoring tools to monitor the operation of a cluster.

VII. Conclusion

The TTA is the result of more than twenty years of research in the field of dependable distributed real-time systems. During this period, many ideas have been developed, implemented, evaluated, and finally discarded. What survived is a small set of orthogonal concepts that center around the availability of a dependable global time base. The guiding principle during the development of the TTA has always been to take maximum advantage of the availability of this global time, which is part of the world, even if we do not use it. The TTA spans the whole spectrum of dependable distributed real-time systems, from the low-cost deeply embedded sensor nodes to high-performance nodes that communicate at gigabits per second speeds, persistently assuming that a global time of appropriate precision is available in every node of the TTA.

Many of today's hardware architectures provide a single CPU to handle the middleware and the application software. The introduction of the FTU CNI of the TTA (see Fig. 11) suggests that a dedicated middleware CPU would enhance the temporal predictability of a node significantly by eliminating unnecessary temporal interactions between the middleware software and the host software. The middleware processor can perform the housekeeping and generic fault-tolerance functions, while the host processor would be dedicated solely to execute the application software in the host. Such a hardware architecture would make a significant contribution to solving the software reuse problem in distributed real-time systems, since an application could be ported to a new environment without any change in the application software interface, neither in the time, nor in the value domain.

At present, the TTA occupies a niche position, since in the experimental as well as in the theoretical realm of mainline computing, time is considered a nuisance that makes life difficult and should be dismissed at the earliest moment [63], [64]. However, as more and more application designers start to realize that real time is an integrated part of the real world that cannot be abstracted away, the prospects for the TTA look encouraging.

Acknowledgment

The authors would like to thank all Ph.D. students, master students, sponsors, and industrial partners that have contributed to the development of the TTA over a period of two decades.

References

[1] A. Simon, *Sciences of the Artificial*. Cambridge, MA: MIT Press, 1981.
[2] A. Avizienis, G. Gilley, F. Mathur, D. Rennels, J. Rohr, and D. Rubin, "The STAR (self-testing-and-repairing) computer: An investigation of the theory and practice of fault-tolerant computer design," *IEEE Trans. Comput.*, vol. C-20, pp. 1312–1321, Nov. 1971.
[3] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE*, vol. 66, pp. 1240–1255, Oct. 1978.
[4] A. Hopkins, T. Smith, and J. Lala, "FTMP—A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221–1239, Oct. 1978.
[5] J. Lala and L. Alger, "Hardware and software fault tolerance: A unified architectural approach," in *Proc. 18th Int. Symp. Fault Tolerant Comput.*, 1988, pp. 240–245.
[6] R. Kieckhafer, C. Walter, A. Finn, and P. Thambidurai, "The MAFT architecture for distributed fault tolerance," *IEEE Trans. Comput.*, vol. 37, pp. 398–405, Apr. 1988.
[7] P. Traverse, "AIRBUS and ATR system architecture and specification," in *Software Diversity in Computerized Control Systems*, U. Voges, Ed. Berlin, Germany: Springer-Verlag, 1988.
[8] K. Hoyme and K. Driscoll, "SAFEbus," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 8, pp. 34–39, Mar. 1993.
[9] J. Lala and R. Harper, "Architectural principles for safety-critical real-time applications," *Proc. IEEE*, vol. 82, pp. 25–40, Jan. 1994.
[10] A. Avizienis, "Toward systematic design of fault-tolerant systems," *IEEE Computer*, vol. 30, pp. 51–58, Apr. 1997.

[11] E. Rechtin, *Systems Architecting, Creating and Building Complex Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[12] J. C. Laprie, *Dependability: Basic Concepts and Terminology*. Berlin, Germany: Springer-Verlag, 1992.
[13] A. Avizienis, "The N-version approach to fault-tolerance software," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1491–1501, Dec. 1985.
[14] J. Knight and N. Leveson, "An empirical study of failure probabilities in multi-version software," in *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, 1986, pp. 165–170.
[15] U. Voges, Ed., *Software Diversity in Computerized Control Systems*. Berlin, Germany: Springer-Verlag, 1988.
[16] *Software Considerations in Airborne Systems and Equipment Certification*, Standard RTCA/DO-178B , 1992.
[17] D. Powell, "Distributed fault tolerance—Lessons learnt from delta-4," *IEEE Micro*, vol. 14, pp. 36–47, Jan. 1994.
[18] "Air Traffic Control—Complete and Enforced Architecture Neded for FAA Systems Modernization," U.S. General Accounting Office, GAO/AIMD-97-30, 1997.
[19] H. Kopetz *et al.*. (1982) The Architecture of MARS. Technical Univ. Berlin, Berlin, Germany. [Online] Available: http://www.vmars.tuwien.ac.at
[20] H. Kopetz and W. Merker, "The architecture of MARS," in *Proc. 15th Int. Symp. Fault-Tolerant Comput.*, 1985, pp. 274–279.
[21] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger, "Distributed fault-tolerant real-time systems: The MARS approach," *IEEE Micro*, vol. 9, pp. 25–40, Jan. 1988.
[22] J. Reisinger, A. Steininger, and G. Leber, "The PDCS implementation of MARS hardware and software," in *Predictably Dependable Computing Systems*, H. Kopetz, B. Randell, J. L. Laprie, and B. Littlewood, Eds. Berlin, Germany: Springer-Verlag, 1995.
[23] H. Kopetz and K. H. Kim, "Temporal uncertainties in interactions among real-time objects," in *Proc. 9th Symp. Reliable Distributed Syst.*, 1990, pp. 165–174.
[24] K. Kim and H. Kopetz, "A real-time object model RTO.k and an experimental investigation of its potential," in *Proc. COMPSAC '94*, 1994, pp. 392–402.
[25] H. Kopetz and G. Grünsteidl, "TTP—A time-triggered protocol for fault-tolerant real-time systems," in *Proc. 23rd Int. Symp. Fault-Tolerant Comput.*, 1993, pp. 524–533.
[26] H. Kopetz and W. Ochsenreiter, "Clock synchronization in distributed real-time systems," *IEEE Trans. Comput.*, vol. C-36, pp. 933–940, Aug. 1987.
[27] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, and J. Reisinger, "Comparison and integration of three diverse physical fault injection techniques," in *Proc. PDCS 2: Open Conf.*, 1994, pp. 615–642.
[28] R. Hexel, "Validation of fault-tolerance mechanisms in a time-triggered communication protocol using fault injection," Ph.D. disseratation, Vienna Univ. Technol., Institut für Technische Informatik, Vienna, Austria, 1999.
[29] T. Thurner and G. Heiner, "Time-triggered architecture for safety-related distributed r eal-time systems in transportation systems," in *Proc. 28th Int. Symp. Fault-Tolerant Comput.*, 1998, pp. 402–407.
[30] *TTTech: Time-Triggered Technology* [Online]www.tttech.com
[31] *Road Vehicles—Controller Area Network (CAN)—Part 4: Time Triggered Comunication*, Standard ISO/CD 11 898-4, 2001.
[32] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lormann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann, "FlexRay—The Communication System for Advanced Automotive Control Systems," SAE, Doc. no. SAE 2001-01-0676, 2001.
[33] P. Miner. Analysis of the SPIDER fault tolerance protocols. presented at Lfm 2000: 5th NASA Langley Formal Methods Workshop. [Online]http://archive.larc.nasa.gov/shemesh/Lfm2000/Presentations/lfm2000-spider/
[34] J. Rushby, "Bus architectures for safety-critical embedded systems," in *Lecture Notes in Computer Science, Embedded Software*. Heidelberg, Germany: Springer-Verlag, 2001, vol. 2211, pp. 306–323.
[35] ——, "A Comparison of Bus Architectures for Safety-Critical Embedded Systems," SRI Int., Computer Science Laboratory, Menlo Park, CA, 2001.
[36] H. Kopetz, "The time-triggered (TT) model of computation," in *Proc. 19th IEEE Real-Time Syst. Symp.*, 1998, pp. 168–177.
[37] ——, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA: Kluwer, 1997.
[38] "Smart Transducer Interface," Object Management Group, Needham, MA, 2001.
[39] M. Mesarovic and Y. Takahara, *Abstract Systems Theory*. Berlin, Germany: Springer-Verlag, 1989.
[40] F. Tisato and F. DePaoli, "On the duality between event-driven and time-driven models," in *Proc. 13th Workshop on Distributed Comput. Contr. Syst.*, 1995, pp. 31–36.
[41] C. Temple, "Enforcing error containment in distributed time-triggered systems: The bus guardian approach," Ph.D. dissertation, Vienna Univ. Technol., Institut für Technische Informatik, Vienna, Austria, 1999.
[42] H. Kopetz, G. Bauer, and S. Poledna, "Tolerating Arbitrary Node Failures in the Time-Triggered Architecture," SAE, Doc. no. SAE 2001-01-0677, 2001.
[43] G. Bauer, H. Kopetz, and P. Puschner, "Assumption coverage under different failure modes in the time-triggered architecture," in *Proc. 8th IEEE Int. Conf. Emerging Technol. Factory Automat.*, 2001, pp. 333–341.
[44] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.
[45] H. Kopetz, "Elementary versus composite interfaces in distributed real-time systems," in *Proc. 4th Int. Symp. Autonomous Decentralized Syst.*, 1999, pp. 26–33.
[46] H. Kopetz and R. Nossal, "Temporal firewalls in large distributed real-time systems," in *Proc. IEEE Workshop Future Trends Distributed Comput.*, 1997, pp. 310–315.
[47] H. Kopetz and J. Reisinger, "The nonblocking write protocol NBW: A solution to a real-time synchronization problem," in *Proc. 14th Real-Time Syst. Symp.*, 1993, pp. 131–137.
[48] H. Kopetz, "Software engineering for real-time: A roadmap," in *Proc. 22nd Int. Conf. Software Eng.*, 2000, pp. 201–211.
[49] S. Poledna, *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Norwell, MA: Kluwer, 1995.
[50] J. Gray, "Why do computers stop and what can be done about it? ," in *Proc. 5th Symp. Reliability Distributed Software Database Syst.*, 1985, pp. 3–11.
[51] C. Jones, M. Killijian, H. Kopetz, E. Marsden, N. Moffat, M. Paulitsch, D. Powell, B. Randell, A. Romanovsky, and R. Stroud, "Revised Version of DSoS Conceptual Model," Technical University of Vienna, Institut für Technische Informatik, Vienna, Austria, 35/2001, 2001.
[52] Specification of the TTP/C Protocol. TTTech Computertechnik AG. [Online]www.tttech.com
[53] H. Kopetz, M. Holzmann, and W. Elmenreich, "A universal smart transducer interface: TTP/A," in *Proc. 3rd IEEE Int. Symp. Object-Oriented Real-Time Distributed Comput.*, 2000, pp. 16–23.
[54] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Comput. Surveys*, vol. 22, no. 4, pp. 299–319, 1990.
[55] G. Bauer and H. Kopetz, "Transparent redundancy in the time-triggered architecture," in *Proc. Int. Conf. Dependable Syst. Networks*, 2000, pp. 5–13.
[56] H. Pfeifer, D. Schwier, and F. von Henke, "Formal verification for time-triggered clock synchronization," in *Proc. 7th IFIP Int. Working Conf. Dependable Comput. Crit. Applicat.*, 1999, pp. 207–226.
[57] H. Pfeifer, "Formal verification of the TTP group membership algorithm," in *Proc. IFIP TC6/WG6.1 Int. Conf. Formal Description Techniques Distributed Syst. Communication Protocols (FORTE XIII) Protocol Specification, Testing and Verification (PSTV XX), FORTE/PSTV 2000*, 2000, pp. 3–18.
[58] D. Schwier and F. von Henke, "Mechanical verification of clock synchronization algorithms," in *Lecture Notes in Computer Science, Formal Techniques in Real-Time and Fault-Tolerant Systems*, A. Ravn and H. Rischel, Eds., 1998, vol. 1486.
[59] J. Lundelius and N. Lynch, "An upper and lower bound for clock synchronization," *Inform. Contr.*, vol. 62, no. 2/3, pp. 190–204, 1984.
[60] J. Rushby and F. von Henke, "Formal Verification of the Interactive Convergence Clock Synchronization Algorithm," SRI International, Computer Science Laboratory, SRI-CSL-89-3R, 1989.
[61] G. Bauer and M. Paulitsch, "An investigation of membership and clique avoidance in TTP/C," in *Proc. 19th IEEE Symp. Reliable Distributed Syst.*, 2000, pp. 118–124.
[62] W. Schütz, *The Testability of Distributed Real-Time Systems*. Norwell, MA: Kluwer, 1993.

[63] E. Lee, "Embedded Software—An Agenda for Research," Univ. California, Berkeley, CA, UCB/ERL no. M99/63, 1999.
[64] ——, "What's ahead for embedded software?," *IEEE Computer*, vol. 33, pp. 18–26, July 2000.

Hermann Kopetz (Fellow, IEEE) received the Ph.D. degree in physics "sub auspiciis praesidentis" from the University of Vienna, Vienna, Austria, in 1968.

He was Manager of the Computer Process Control Department at Voest Alpine, Linz, Austria, and Professor of Computer Process Control, Technical University of Berlin, Berlin, Germany. He is currently Professor of Real-Time Systems, Vienna University of Technology, Vienna, Austria, and a Visiting Professor at the University of California, Irvine, and the University of California, Santa Barbara. In 1993, he was offered a position as Director of the Max Planck Institute, Saarbrücken, Germany.

Günther Bauer received the Dipl. degree in electrical engineering and the Ph.D. degree in computer science from the Vienna University of Technology, Vienna, Austria.

He is currently Technical Coordinator of a European Union–funded Information Society Technologies research project in the Real-Time Systems Group at the Vienna University of Technology. His research interests include fault-isolation, fault-handling, and fault-tolerance aspects of distributed hard real-time systems. His current research work focuses on the design, implementation, and validation of a central guardian for the TTA.