The Time-Triggered Architecture

H. Kopetz

Technical University of Vienna, Austria

hk@vmars.tuwien.ac.at

Abstract: The Time-Triggered Architecture (TTA) is a computer architecture for distributed real-time systems in safety critical applications, such as computer controlled brakes, or computer assisted steering in an automobile. The TTA is a composable architecture for the design of large real-time systems. Its main characteristics are a common notion of time in all elements of the architecture and the provision of fully specified interfaces, called temporal firewalls, between these elements. This paper gives an overview of the TTA, discusses the architectural principles, describes the sensor/actuator interfaces in the TTA and informs about the implementation of fault-tolerance in the TTA.

1. INTRODUCTION

A *real-time computer system* is a computer system in which the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical instant at which these results are produced. Large real-time systems can only be built constructively if the effort needed to reason about the logical and temporal properties of a particular system function is independent of the system size. This requires that a system can be split into a set of nearly autonomous subsystems that interact only via understandable, fully specified and stable interfaces (Courtois 1985,. Rechtin 1991). It is then possible to reason about a system function in a limited context: it is sufficient to look at the subsystem under consideration an its interfaces to the subsystem environment (i.e., the rest of the system). The subsystem environment is thus hidden behind the well specified subsystem interfaces. If the essential properties of the interfaces between the subsystems are fully specified in the value domain and in the temporal domain, and if these interface properties are not changed by the system integration, then the design is said to be *composable* with respect to the stated properties. In a composable architecture, important system properties, such as timeliness and testability, follow from the established subsystem properties.

The Time-Triggered Architecture is a composable architecture for the design of large distributed real-time systems is in safety critical applications, such as computer controlled brakes, or computer assisted steering in an automobile. Its main characteristics are a common notion of time in all elements of the architecture and the provision of fully specified interfaces between these elements. The TTA evolved out of many years of university research centered on the topic of distributed fault-tolerant real-time systems, carried out in the context of the MARS project (Kopetz, Damm et al. 1989) and the PDCS Project (Randell, Laprie et al. 1995).

In this paper, the time-triggered architecture is presented. After a general overview of the architecture the design principles are discussed in some detail. The concept of a temporal firewall, a well-specified stable interface between the components, is elaborated. Section 4 is devoted to the controlled object interface and introduces the time-triggered sensor bus as a new type of field bus. Finally, Section 5 deals with the implementation of fault-tolerance in the TTA.

2. Overview of the Architecture

A real-time computer system is always part of a larger system–this larger system is called a *real-time system*. A real-time system changes its state as a function of physical time, e.g., a chemical reaction continues to change its state even after its controlling computer system has stopped. It is reasonable to decompose a real-time system into a set of subsystems called *clusters* (Figure 1) e.g., the *controlled object* (the *controlled cluster*), the real-time computer system (the *computational cluster*) and the *human operator* (the *operator cluster*). We refer to the controlled object and the operator collectively as the *environment* of the real-time computer system.

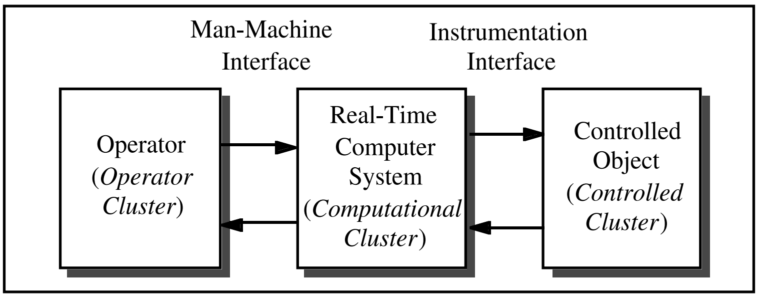

Figure 1: Real-time system.

Temporal Accuracy of Information

If time is frozen, the current state of the controlled object can be described by recording the values of its state variables at that moment. Possible state variables of a controlled object "car" are the position of the car, or the speed of the car. Normally one is not interested in *all* state variables, but only in the *subset* of state variables that is *significant* for the given purpose. A significant state variable is called a *real-time (RT) entity* (Kopetz 1997).

The observation of a RT entity in the controlled object is stored as a *real-time image* in the computer system. A real-time (RT) image is a *current* picture of an RT entity. An RT image is valid at a given point in time if it is an accurate representation of the corresponding RT entity, both in the value and the time domains. While an observation records a fact that remains valid forever (a statement about an RT entity that has been observed at a particular point in time), the validity of an RT image is *time-dependent* and thus invalidated by the progression of real-time. The time interval, during which a RT image is temporally valid, is called the *temporal accuracy* of the RT image (Kopetz and Kim 1990).

The interface between the human operator and the real-time computer system is called the *man-machine interface,* and the interface between the controlled object and the real-time computer system is called the *instrumentation interface*. The man-machine interface consists of input devices (e.g., keyboard) and output devices (e.g., display) that interface to the human operator. The instrumentation interface consists of the sensors and actuators that transform the physical signals (e.g., voltages, currents) in the controlled object into a digital form and *vice versa*. A node with an instrumentation interface is called an *interface node.*

A real-time computer system must react to stimuli from the controlled object (or the operator) within time intervals dictated by its environment. The instant at which a result must be produced is called a *deadline*. If a result has utility even after the deadline has passed, the deadline is classified as *soft*, otherwise it is *firm*. If a catastrophe could result if a firm deadline is missed, the deadline is called *hard.* Consider a railway crossing a road with a traffic signal. If the traffic signal does not change to "red" before the train arrives, a catastrophe could result.

Computational Cluster

In the TTA the computational cluster is *distributed*. It consists of a set of (computer) *nodes* interconnected by a real-time communication network which is based on the Time-Triggered Protocol TTP (Kopetz and Grünsteidl 1994). Figure 2 depicts such a distributed computational cluster. The interface between the communication network and the node is called the communication network interface (CNI). The CNI between the TTP controller and the host computer is free of any temporal control signals, i.e., the CNI is data sharing interface implemented by a dual ported RAM memory.

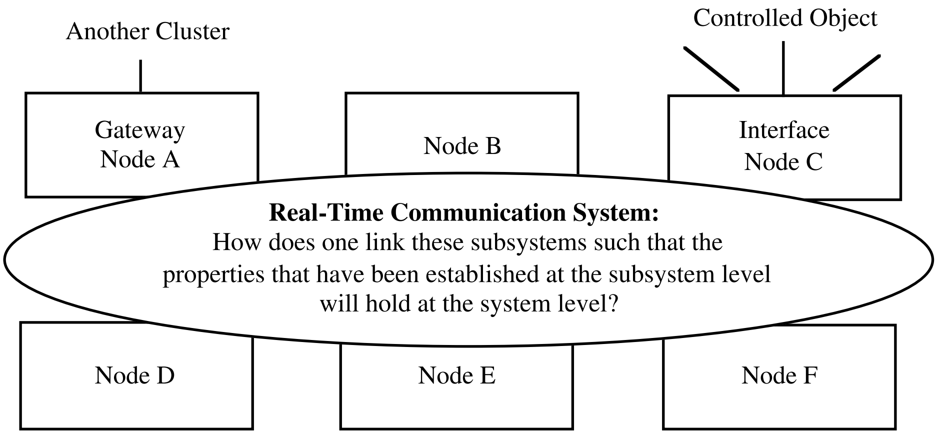

Figure 2**:** Structure of a computational cluster.

TTP is TDMA (time-division multiple access) type of protocol, where a sending slot is assigned *a priori* to every node. The protocol control information, i.e., the message descriptor list (MEDL) containing the points in time when a node is allowed to send a message, is contained in the TTP controller. TTP provides the following services:

(i) Message transport with low latency and minimal jitter,

(ii) Fault-tolerant clock synchronization,

(iii) Provision of a fault-tolerant membership service,

(iv) Provision of an immediate and deferred mode change service,

(v) Distributed redundancy management,

TTP provides flexibility as long as the determinism, i.e., the analytical predictability of the timeliness, can be maintained.

A TTA Node

In the TTA, a node can be viewed as an RT object that provides specified timely services across its CNIs (Kim and Kopetz 1994).The internal structure of a TTA node is depicted in Figure 3.

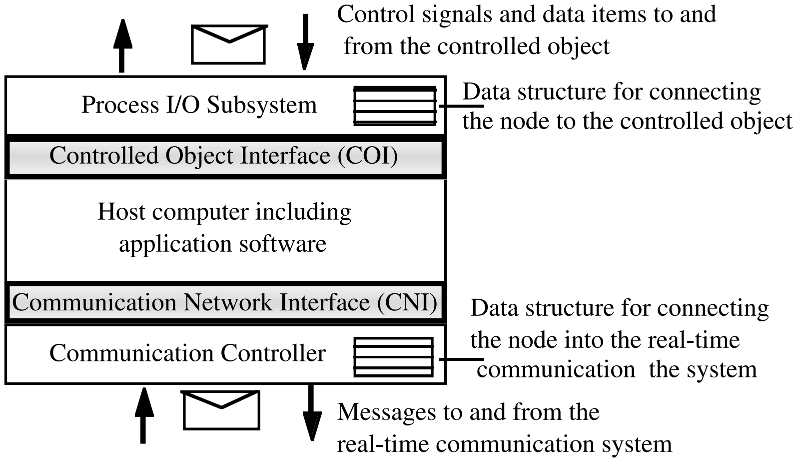

Figure 3: Structure of a node.

In general, the node hardware consists of a host computer, and one or two communication controllers (to the real-time bus and to the process I/O subsystem as shown in Figure 3). The host computer comprises the CPU, the memory and a real-time clock. The execution of the concurrently executing tasks within the host computer is controlled by the host operating system. The host computer shares the node internal CNI with the communication controller and the controlled object interface (COI) with the process I/O subsystem. The host accepts the temporally valid input messages and produces the intended and timely output messages via the CNI and the COI.

3. DESIGN PRINCIPLES

Global Time

In the TTA it is assumed that all elements of the architecture have access to a global time of known precision in the microsecond range (Kopetz and Ochsenreiter 1987). This global time is based on the metric of the physical second. Since a failure of this global time base can have catastrophic consequences, the global time is fault tolerant. The control signals for the activation of tasks or the sending of messages are derived from the progression of this global time base. The global time is also used to monitor the temporal accuracy of real-time data.

Temporal Firewalls

In the TTA, the interfaces between a node and its environment are formed by so-called *temporal firewalls* (Kopetz and Nosssal 1997).

*A **temporal firewall** is a unidirectional **data-sharing interface** with **state-data semantics** where **at least one** of the interfacing subsystems accesses the temporal firewall according to an **a priori known schedule** and where at all points in time the information contained in the temporal firewall is **temporally accurate** for at least d_{acc} time units into the future.*

The subsystem that accesses the temporal firewall according to the *a priori* known schedule is called the time-triggered (TT) subsystem. No control signal is crossing the temporal firewall. The information provider has to update the RT image in the temporal firewall according to the dynamics of the corresponding RT entity. If the information-providing subsystem ceases to operate, the information in the temporal firewall will be invalidated by the passage of time.

The following stable properties characterize a temporal firewall. Knowledge about these properties is available *a priori* to all interfacing subsystems:

(i) The addresses (names) and the syntactic structure of the data items in the temporal firewall. The meaning of the data items is associated with these names.

(ii) The points on the global time base when the data items in the temporal firewall are accessed by the TT subsystem. This information enables the avoidance of race conditions between the producer and the consumer. A race condition could lead to a loss of replica determinism in replicated temporal firewalls (Poledna 1995).

(iii) The temporal accuracy d_{acc} of the data items in the temporal firewall (Kopetz 1997). This knowledge is important to guide the information consumer about the minimum rate of sampling the temporal firewall. The absolute timepoints when the TT subsystem accesses the temporal firewall are reference points for the temporal accuracy of the information in the temporal firewall.

The common knowledge about the static properties of temporal firewall is used for error detection. Temporal firewall act as error containment interfaces within the TTA. Since a temporal firewall is free of control signals, the possibility of a control error propagation across a temporal firewall is ruled out by design.

Unification of Interfaces

In the TTA the temporal firewall is a generic interface. It unifies the interface properties of

(i) a host computer and the communication network,

(ii) a host computer and the controlled object, and

(iii) a gateway interface between two clusters.

This unification of the node interfaces simplifies the node software, since the same mechanisms can be used to access the controlled object data and the network data. Figure 4 depicts a multicluster system with temporal firewalls in each node. As mentioned before, the temporal firewalls act also as error containment interfaces.

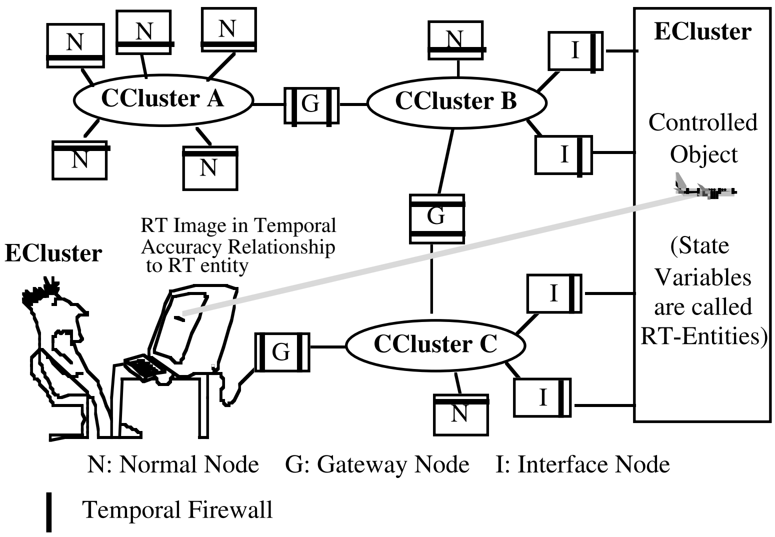

Figure 4: Example of a Multicluster System with Temporal Firewalls
(ECluster: Environment Cluster, CCluster: Computational Cluster)

In such a large system, the parameters of the temporal firewalls are developed during the architecture design phase. These parameters are then loaded into the TTP controllers and are inaccessible to the host computers. The host computer software can be developed and tested with respect to the fully specified temporal firewalls. Since the TTA architecture is composable, no unintended side effects will occur during system integration.

Real-Time Database

Conceptually, the distributed real-time data in the temporal firewalls, formed by the temporally accurate images of all relevant RT entities, is at the core of the time-triggered architecture. The real-time database is autonomously and periodically updated by the nodes of the cluster that observe the environment and produce RT images. The real-time data base contains a temporally valid "snapshot" of the current state of the cluster and the cluster environment. The data structures that control the updating of the real-time database are in the TTP communication controller, physically and logically separated by the CNI from the host software. These data structures are designed during the architecture design phase of a cluster. Even a malicious error in the host software cannot affect the communication pattern that updates the real-time database. This property of the architecture is important from the point of view of certification (Rushby 1993).

Scalability

Growth of a TTA architecture is unconstrained since there is no central element in the architecture. Nodes can be expanded into gateway nodes by implementing a second CNI interface in the node. The CNI interface to the original cluster is not affected by this node-local change. Understanding a large TTA system can be decomposed into understanding each cluster and each node of a cluster. Every node views all other elements as its "natural environment", not being able to distinguish a controlled-object cluster from a computational cluster. This architectural characteristic is important during software development and testing, because a test simulator will have exactly the same interface, both in the value and time domain, as the controlled object.

4. Interfacing to the Controlled Object

From the view of a host computer, the controlled object is hidden behind the controlled object interface COI that is a temporal firewall. This firewall contains temporal accurate state data of the sensors and actuators.

Intelligent Sensors

In the TTA it is assumed that sensors are intelligent, i.e., that a microcontroller with local processing capability is associated with each sensor and actuator. The local microprocessor performs the observation of the associated RT entity, signal conditioning, signal conversion to a standard format, and executes the time-triggered sensor bus protocol TTSB to transmit the data to a TTA node. Since a serial bus can only transport one message at a time, some observation messages will have to wait longer than other observation messages until they can be delivered at the TTA node. If these different waiting times cannot be controlled, a significant jitter can accumulate during the communication of the sensor observations that deteriorates the quality of control.

Ideally, all RT entities should be observed at the same point in time immediately before the start of the control algorithm task at the TTA node. In a time-triggered communication system the time-difference between the start of sending a message and the delivery of the message at the receiver is known *a priori*. This time-difference can be used by the intelligent sensor nodes to estimate the state of the observed continuous RT entity at the point in time when the control algorithm is started at the control node (Kopetz 1997). This state estimation can be performed on the basis of a simple Taylor expansion of the value change of the RT entity in the recent past or of some more sophisticated state estimation algorithm that is based on *a priori* knowledge about the RT entity and the long term observation of the RT entity.

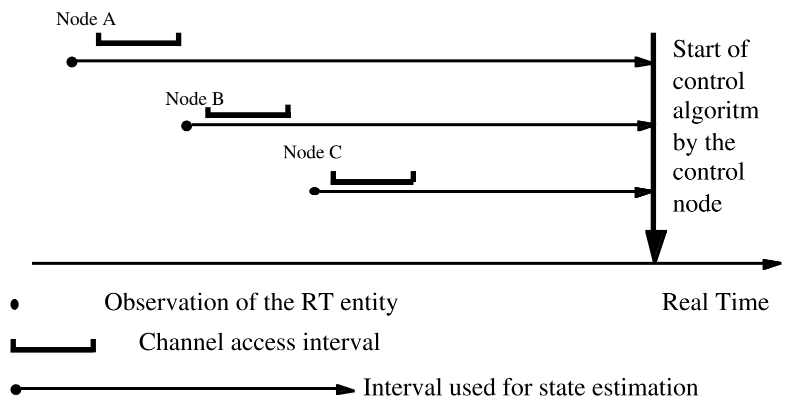

Figure 5: State-estimation intervals.

The state estimation procedure at each intelligent sensor node will use a different time delay (Figure 5) because of the sequential use of the single serial communication channel. However, since the individual access times to the channel are known *a priori* to each node, every sensor node can use its "personal" delay for the state estimation procedure. To achieve optimal temporal accuracy, the node with the most dynamic behavior should be configured to send its message at the latest possible time before the start of the control algorithm at the control node. At the receiving TTA node, the behavior of the system will be as if all sensor values have been observed at almost the same point in time immediately before the start of the control algorithm.

The whole sensor subsystem, including its temporal properties, is hidden behind the controlled object's *temporal firewall.*

The Time-Triggered Sensor Bus TTSB

The time-triggered sensor bus is based on the low-cost version of the TTP protocol family, the TTP/A protocol (Kopetz 1995). TTP/A is a multimaster protocol that can be implemented on any microprocessor with a UART (Universal Asynchronous Receive Transmit) interface. TTP/A is based on one byte messages. Most of these messages are data messages, only one special message, the Fireworks message, is a control message.

In TTP/A all communication activities are organized into rounds (Fig. 6). A round is the transmission of a sequence of one character messages that is specified *a priori* in a Message Description List (MEDL). A round starts with a special control byte, the Fireworks, that is transmitted by the active master. The Fireworks byte serves two purposes: It is the global synchronization event for the start of a new round and it contains the name of the active MEDL for this round. The Fireworks is followed by a sequence of data bytes from the individual nodes as specified in the active MEDL. A round terminates when the end of the active MEDL is reached. Every round is independent of the previous round.

To be able to differentiate between a Fireworks byte and a data byte, the Fireworks has characteristic features in the value domain and in the time domain that differentiate the Fireworks byte from the data bytes: The Fireworks byte has an odd parity while all data bytes have even parity. The intermessage gap between the Fireworks byte and the first data byte is significantly longer than the intermessage gap between the succeeding data bytes. These characteristic features make it possible for all nodes to recognize a new Fireworks byte, even if some faults have disturbed the communication during the previous round. The characteristic features of the Fireworks byte simplify the reintegration of repaired nodes--a repaired node monitors the network until a correct Fireworks byte is detected.

Since the sequence of messages is determined a priori by the definition of the active MEDL, it is not necessary to carry the identifier of a message as part of the message. All eight data bits of a message are true data bits. This improves the data efficiency of the protocol, particularly for the short one and two byte messages that are typical for control applications.

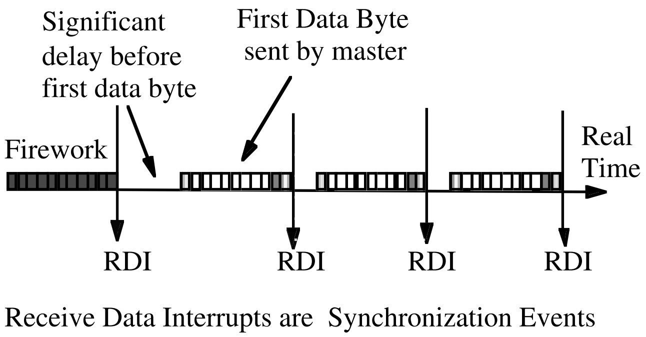

Fig. 6: Structure of a round.

From the point of view of the protocol operation, every round is independent of the previous round. In many applications, the termination of a round will cause the initiation of an identical next round by the active master. We call a sequence of identical rounds controlled by the same

MEDL a *mode*. With the start of every new round a mode change can be initiated by the active master by packing the name of a new MEDL into the Fireworks byte.

To increase the flexibility without compromising the predictability, the protocol supports an operational mode that allows the on-line downloading of a new MEDL after power up. Thus sensor nodes can be mass produced and receive the personality of a given application during the power-up phase. The master can enter the download mode by sending the proper Fireworks byte. It then broadcasts the new MEDL in a specified format. Every slave node will construct its local MEDL from the information received by the master. After the downloading activity is completed, the master can activate the downloaded MEDL by a mode change.

5. FAULT TOLERANCE

The only way for a large system to survive is to provide mechanisms that detect and mask faults without disturbing the system service (Avizienis 1997). One design goal of the TTA is the generic support of fault-tolerant operation without any modification of the application software. This approach avoids any increase in the complexity of the application software which is caused by introducing fault-tolerance.

Fault-Tolerant Units

A set of replica-determinate nodes can be grouped into a fault-tolerant unit (FTU) that will mask a failure of a node of the FTU without any effect on the external service provided by this FTU. The TTA supports the formation of different types of FTUs (selfchecking, TMR see Figure 7) and performs the redundancy management within the TTP controller such that the FTU CNI to the host computer is not affected by the replication of nodes and communication channels.

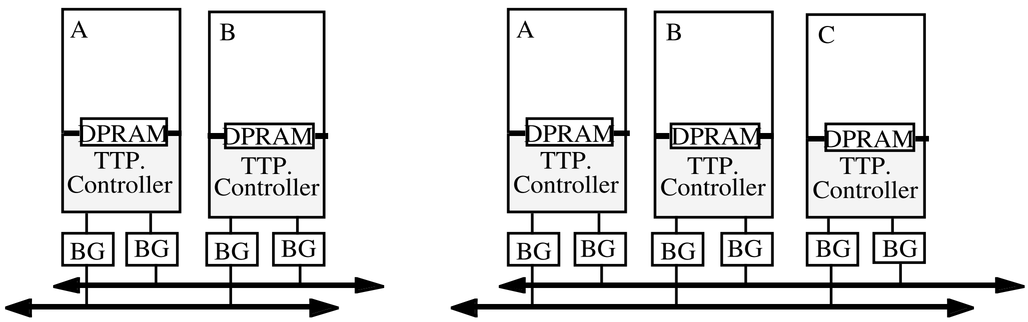

Figure 7: FTU consisting of two selfchecking nodes (left) or of three nodes in a triple modular redundant (TMR) configuration (right)

A necessary precondition for the implementation of active redundancy is the replica-determinate behavior of the host software. The TTA provides replica determinism at the CNI of a node, but it is up to the host software to guarantee replica determinism within the complete node.

Redundant Sensors

If the sensors need to be replicated to achieve fault-tolerance, then two separate field buses must be installed (Figure 8). Each one of those field buses is controlled by one of the TTA-nodes in the FTU. The other node is passive and listens to the field bus traffic to capture the sensor data.

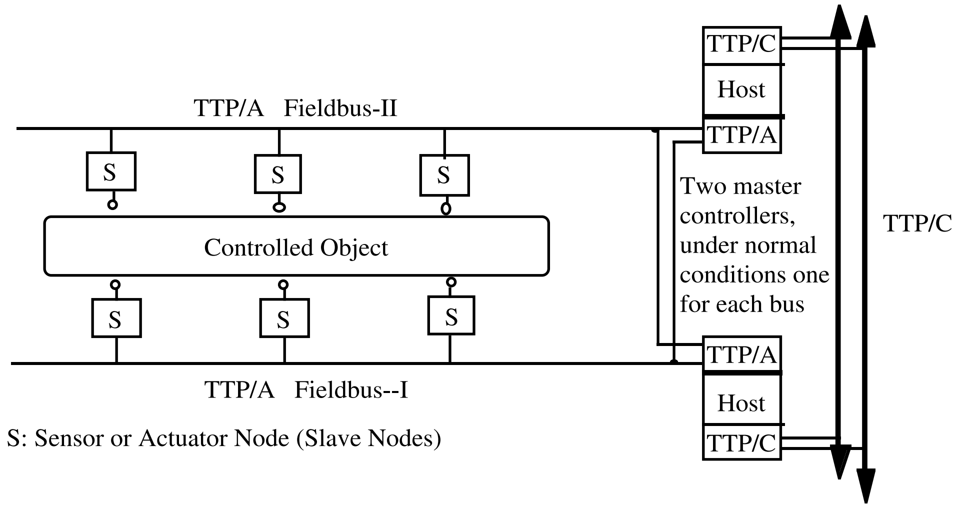

Figure 8: FTU configuration with replicated field buses.

An agreement protocol is executed in the controllers of the TTA-nodesto 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.

The High Error Detection Coverage Mode (HEDC)

If the active redundancy is based on the assumption of selfchecking nodes, then every critical computation is calculated twice on the host computers and the results of these computations are compared before a message is written into the CNI. Fault injection experiments have shown that this time redundancy is very effective in detecting transient faults (Karlsson, Folkesson et al. 1995).

The duplicate execution of application tasks is supported by the operating system by providing a special execution mode, the *High-Error-Detection-Coverage (HEDC)* mode, that is transparent to the application software. Additionally, an end-to-end CRC is calculated by the host operating system. In a safety-critical application, the messages are thus protected by two CRC fields, one at the communication level, and one at the end-to-end (application) level (Saltzer, Reed et al. 1984).

FTU Layer

In the TTA, the fault-tolerance functions are encapsulated in an FTU layer (fault-tolerant unit layer) between two temporal firewalls (Figure 9) within a node, the *basic CNI* and the *FTU CNI* (fault-tolerant unit CNI) (Kopetz and Millinger 1998). The basic CNI and the FTU CNI are syntactically alike and semantically closely related. From the point of view of the host software, only minimal changes are needed to port an application from a non-fault-tolerant implementation to a fault-tolerant implementation.

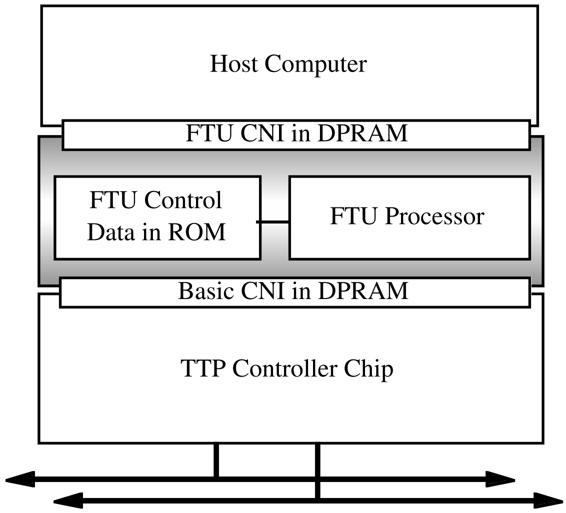

Figure 9: FTU Layer between the basic CNI and the FTU CNI.

At present we have implemented the FTU Layer in software but plan to develop a hardware based FTU layer implementation in the future.

6. CONCLUSIONS

In this paper the design principles and the structure of the time-triggered architecture has been presented. The TTA is a composable architecture that supports a structured system design effort. At the architecture design level, the properties of the interfaces between the architectural units are specified and frozen. At the component design level, the components can be developed with respect to the specified interfaces. The composability properties of the architecture ensure that the system integration will proceed without unintended side effect.

The TTA has been implemented on COTS microcomputers with a special hardware board for the TTP controller. More than 100 TTA nodes have been installed and tested in a variety of application, predominantly in the field of automotive electronics. In the Brite Euram project X-by-wire (X standing for brake, steer, etc.) the TTA has been chosen as the base architecture for the implementation of a safety critical steer-by-wire application. Another ongoing ESPRIT project, the project TTA, has the goal to develop a VLSI chip of the TTP controller. We expect to have first silicon before the end of 1998.

ACKNOWLEDGEMENTS

This work has been supported, in part, by the ESPRIT LTR project DEVA, the ESPRIT OMI project TTA, and the Brite Euram project X-by-wire.

REFERENCES

Avizienis, A. (1997). Toward Systematic Design of Fault-Tolerant Systems. *IEEE Computer*. Vol. **30**. pp. 51-58.

Courtois, P.-J. (1985). On Time and Space Decomposition of Complex Structures. *Comm. ACM*. Vol. **28**. pp. 590-603.

Karlsson, J., P. Folkesson, et al. (1995). Integration and Comparison of Three Physical Fault Injection Techniques. *Predictably Dependable Computing Systems* B. Randell, J. L. Laprie, H. Kopetz and B. Littlewood, Eds. Heidelberg. Springer Verlag. pp. 309-327.

Kim, K. H. and H. Kopetz (1994). A Real-Time Object Model RTO.k and an Experimental Investigation of its Potential. *Proc. COMPSAC 94*, IEEE Press.

Kopetz, H. (1995). TTP/A -- A Time-Triggered Protocol of Body Electronics Using Standard UARTS. *Proc. SAE World Congress*, Society of Automotive Engineers, SAE Technical Paper 950039. pp. 1-9.

Kopetz, H. (1997). Component-Based Design of Large Distributed Real-Time Systems. *Control Engineering Practice–A Joural of IFAC, Pergamon Press*. accepted for publication.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7*. Boston. Kluwer Academic Publishers.

Kopetz, H., A. Damm, et al. (1989). Distributed Fault-Tolerant Real-Time Systems: The MARS Approach. *IEEE Micro*. Vol. **9**. pp. 25-40.

Kopetz, H. and G. Grünsteidl (1994). TTP-A Protocol for Fault-Tolerant Real-Time Systems. *IEEE Computer*. Vol. **24**. pp. 22-66.

Kopetz, H. and K. Kim (1990). Temporal Uncertainties in Interactions among Real-Time Objects. *Proc. 9th Symposium on Reliable Distributed Systems*, Huntsville, AL, USA. IEEE Computer Society Press. pp. 165-174.

Kopetz, H. and D. Millinger (1998). A Fault-Tolerance Layer for a Distributed Time-Triggered Real-Time System. Technische Universität Wien

Kopetz, H. and R. Nosssal (1997). Temporal Firewalls in Large Distributed Real-Time Systems. *Proceedings of IEEE Workshop on Future Trends in Distributed Computing*, Tunis, Tunesia. IEEE Press.

Kopetz, H. and W. Ochsenreiter (1987). Clock Synchronisation in Distributed Real-Time Systems. *IEEE Trans. Computers*. Vol. **36**. pp. 933-940.

Poledna, S. (1995). *Fault-Tolerant Real-Time Systems, The Problem of Replica Determinism*. Hingham, Mass, USA. Kluwer Academic Publishers.

Randell, B., J. C. Laprie, et al. (1995). *Predictably Dependable Computing Systems*. Heidelberg. Springer Verlag.

Rechtin, E. (1991). *Systems Architecting, Creating and Building Complex Systems*. Englewood Cliffs. Prentice Hall.

Rushby, J. (1993). Formal Methods and the Certification of Critical Systems. Computer Science Lab, SRI.

Saltzer, J., D. P. Reed, et al. (1984). End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*. Vol. **2**. pp. 277-288.