Fault Containment and Error Detection in TTP/C and FlexRay

August 28, 2002

H. Kopetz

Research Report 23/2002
Version 1.5

Technical University of Vienna
hk@vmars.tuwien.ac.at

Abstract: TTP/C and FlexRay are two protocols that have been designed for use in time-triggered systems for safety-critical applications. This paper investigates the fault-containment and error-detection mechanisms of safety-critical time-triggered systems that are based on these two protocols. The following critical failure modes of a fault-containment region are introduced and analyzed in detail: babbling idiot failures, masquerading failures, slightly-off-specification (SOS) failures, crash/omission (CO) failures, and massive transient disturbances. After a short description of the two time-triggered protocols TTP/C and FlexRay this paper tries to show how the two protocols handle these failure modes at the architecture level.

Key Words: fault containment, error detection, time-triggered protocols, FlexRay, TTP/C, safety-critical systems

1. INTRODUCTION

Real-time systems that are designed to replace mechanical or hydraulic components in safety-critical automotive applications, such as "drive-by-wire" applications, must achieve a level of safety that is higher or equal to the level of safety of the systems they replace. In the literature, the failure rate of better than 10^{-9} critical failures per mission interval is demanded in these ultra-dependable applications (Suri, Walter et al. 1995). Such a low failure rate can only be achieved if the system is distributed and tolerates the arbitrary failure of any one of its subsystems. During the design care must be taken that the impact of a fault is contained within the subsystem that is directly affected by the fault, and that the consequences of a fault, the ensuing error, is detected before it can propagate to another independent subsystem. Only then it is possible to develop a safety argument for these applications that relies on the independence assumption of the subsystems.

In order to achieve the required level of safety, the system must be partitioned into independent fault-containment regions (FCR) (Hopkins, Smith et al. 1978). Errors that develop within an FCR (as the consequence of the fault) must be detected and isolated by error-detection mechanisms at the boundaries of the FCR such that the errors cannot damage the computational state in any other FCR. The architecture must assure that no single faulty FCR can knock out the complete safety-critical control system, even if such an event has a very low probability. The error-detection mechanisms should form an integral part of the distributed architecture for the following reasons:

(i) The effect of some faults, for example those of *babbling idiot failures* or *slightly-off-specification (SOS) failures*, can corrupt the distributed platform in such a way that further coordinated processing becomes impossible. These faults must be isolated at the architecture level.

(ii) The error-detection latency that can be achieved by hardware implemented error-detection and diagnosis algorithm is *shorter* than the error-detection latency of software implemented algorithms. Since the system is in an inconsistent state between the instant of error occurrence and the instant of consistent error detection, the duration of this sometimes safety-critical inconsistency interval can be reduced.

(iii) The reduction of the greatly complexity of the application software. Already, the complexity of the application software is a topic of concern and the cause of many system design errors leading to system outages. Therefore every effort must be taken to simplify application programming. Ideally a distributed architecture should provide a consistent distributed computing platform where the application software can be developed under the assumption that all actions achieve their desired effect. Any loss of consistency, caused by a fault, must be immediately reported to the application in order that appropriate error-handling mechanisms can be taken at the application level.

(iv) Many of the distributed error-diagnosis algorithms, such as membership, are intricate and require a considerable effort for their correct design and validation (Rushby 2001). By placing these validated algorithms in the hardware of the architecture the problem is solved once-and-for-all. The application programmer is

relieved from redeveloping these algorithms within every application system. The intermixing of the system-level error-detection mechanisms with the application software has as a consequence that the application software becomes more complex and error-prone.

(v) There are also good cost-reasons for implementing generic algorithms, that are required in many different applications, within the hardware of an architecture. If the development of the ratio between the cost of a square millimeter of silicon and the cost for an hour of application programming are extrapolated into the future, it seems wise to replace programming effort by silicon, wherever possible.

In addition to the architecture-based error-detection mechanisms, any safety-critical real-time application must contain error-detection mechanisms at the application level. The *end-to-end argument* from Saltzer (Saltzer, Reed et al. 1984) applies even more to safety-critical real-time systems than to many other systems (Leveson 1995). However, it is much easier to implement an application specific *end-to-end protocol* if a consistent distributed computing base can be assumed than if no such assumption about the consistency of the distributed computing base can be made (Maier et al. 2002). The focus of this paper is, however, on architecture-based fault containment and error detection and not on application-specific *end-to-end* algorithms in safety critical systems.

This paper is organized as follows. Section 2 presents some of the design principles of the time-triggered architecture, focusing on the concepts of time and state. Section 3 introduces some basic concepts required for describing dependable distributed systems and discusses the relationship between fault containment and error detection. It is shown that in an architecture for high-dependability applications, the error-detection mechanisms must be in a different FCR from the subsystem where the fault occurs. Section 4 introduces a set of critical failure modes that must be addressed in any architecture for safety-critical real-time applications. Section 5 gives an overview over essential properties of the two protocols, TTP/C (TTTECH, 2002) and FlexRay (Berwanger, Ebner et al. 2001; FlexRay-group 2002). Section 6 investigates how these two protocols compare in their error-handling capability at the architecture level. Finally, the paper terminates with a conclusion in Section 7.

The reader should note that at present there is a certain competition between FlexRay and TTP/C on the market. Although every effort has been made to support all claims by rational arguments, this paper, nevertheless, represents the view of TTP/C, since it has been written by the author of TTP/C. A more balanced picture would evolve if similarly detailed papers, written by the authors of FlexRay and representing the view of FlexRay, were available. The making of this comparison has been further complicated by the fact that the precise specification of FlexRay—in contrast to the TTP/C specification--is not publicly available and that the available publications contain incomplete and sometimes inconsistent statements about the detailed operation of FlexRay. Any open debate by the scientific community about the detailed safety mechanisms in FlexRay is hindered as long as the FlexRay consortium treats the precise FlexRay specification as a confidential document.

2. THE TIME-TRIGGERED ARCHITECTURE

The time-triggered architecture (TTA) is a distributed architecture for the implementation of hard real-time applications. It consists of a set of nodes interconnected by a TDMA (time-division multiple access) based real-time communication system. The TTA provides the following basic services to the distributed application at the architecture level

(i) a fault-tolerant global sparse time base of known precision at all nodes.

(ii) mechanisms for the precise operational specification of the interfaces among the nodes in the domains of time and value. These interfaces are called "temporal firewalls".

(iii) a communication service that transports messages from on node to another node with nearly constant delay and minimal jitter.

Both TTP/C and FlexRay provide these basic services. If the TTA is used in safety critical applications, the state of the art of safety critical computing (Lala and Harper 1994) (Thurner and Heiner 1998) requires *that any identified single fault of an FCR that has the potential to impact the correct operation of the remaining FCRs* must be detected and handled in order that the remaining FCRs can continue to provide their safety-critical service. Otherwise, the complete control system can fail as a consequence of the occurrence of a single fault. To achieve this additional capability the following supplementary services should be provided at the level of the architecture in a safety-critical TTA application:

(i) Prompt detection of errors in order to avoid the loss of state consistency in FCRs that have not been directly affected by a fault.

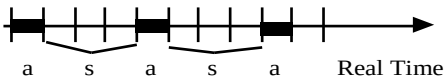
(ii) Error containment mechanisms such that the consequences of arbitrary FCR failures can be contained and tolerated.

(iii) Mechanisms that support the transparent implementation of fault-tolerance.

2.1 Global Sparse Time

For most applications, a model of time based on Newtonian physics is adequate. In this model, 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 at an instant is thus an event. The *time-stamp* of an event is established by assigning the state of the local clock of the observer to the event immediately after the event occurrence. Due 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 occurring: clock in component *j* ticks, event *e* occurs, clock in component *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. The finite precision of the global time-base and the digitalization of the time make it impossible in a distributed system to order events consistently on the basis of their global time-stamps based on a *dense* time. This problem can be solved by the introduction of a *sparse time base* (Kopetz 1997), p.55 into the TTA. 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 Figure 1. The activity intervals form a synchronized system-wide *action lattice.*



a duration of activity
s duration of silence

Figure 1: Sparse time base

From the point of view of temporal ordering, all events that occur within a duration of activity of the action lattice are considered to happen *at the same time*. Events that happen in the distributed system at different components at the same global clock-tick are thus considered *simultaneous.* Events that happen during different durations of activity (at different points of the action lattice) and are separated by the required interval of silence (the duration of this silence interval depends among others, on the precision of the clock synchronization (Kopetz 1992)) can be temporally ordered on the basis of their global timestamps. The architecture must make sure that significant events, such as the sending of a message, or the observation of the environment, occur only during an interval of activity of the action lattice. The time-stamps of events that are based on a sparse time base can be mapped on the set of positive integers. It is then possible to establish the temporal order of events by integer arithmetic.

The timestamps 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 lattice point of the action lattice by an *agreement protocol.* Agreement protocols are also needed to come to a system-wide consistent view of analogue values that are digitized by more than one analogue-to-digital converter.

2.2 Distributed State

In *abstract system theory*, the notion of *state* is introduced in order to separate the *past* from the *future* (Mesarovic and Takahara 1989) 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 word, 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 system-wide 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 above, makes it possible to define a system-wide notion of time, which is a prerequisite for an indisputable borderline between the past and the future, and thus the definition of a system-wide distributed state. The "interval of silence" on the sparse time base forms a system wide 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. 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 to determine in such a model the precise state of a system at an instant of physical time at which voting on replicated copies of the state must be performed.

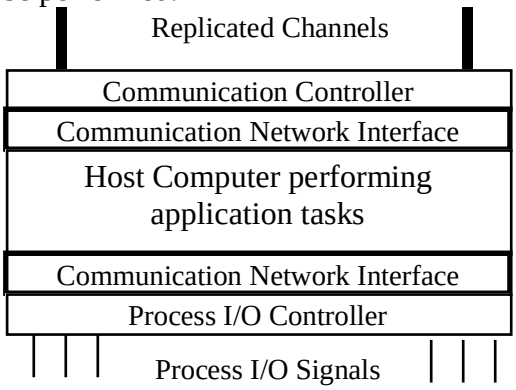


Figure 2: Internal Structure of a Node of the TTA

2.3 TTA Nodes

A node of the TTA is composed of three subsystems: a communication controller to the time-triggered communication system that contains two independent multicast communication channels, a host computer to perform the application tasks, and a communication controller to access the process input/output (Figure 2).

At the architecture level there are two operationally fully specified (in time and value) interfaces between the three subsystems within a node, called communication network interfaces (CNIs). The CNIs form temporal firewalls that eliminate control error propagation by design (Kopetz and Suri 2002). The communication system transports state messages from the CNI in the sending node to the CNIs in the other nodes within a cluster of nodes via the replicated communication channels. Since state messages are not consumed on reading and a new version of a state message overwrites the previous one, the CNI for a state message can be placed in a dual-ported memory. The data flow and control flow between a sending host computer in one node and a receiving host computer in another node is shown in Figure 3.

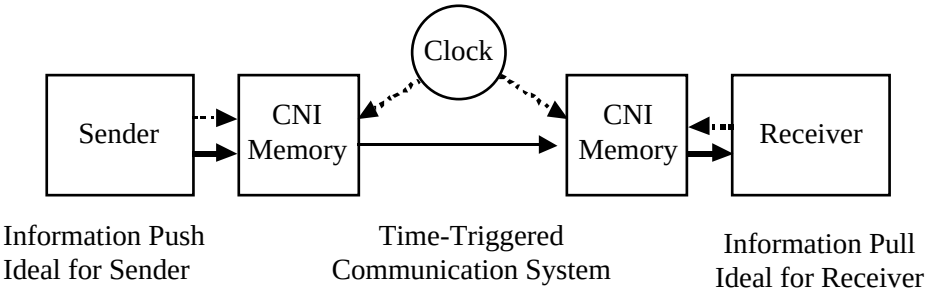


Figure 3: Data flow (full line) and control flow (dashed line) across a temporal firewall interface

The instants when messages are fetched from the sender's CNI and are delivered at the receiver's CNI are known *a priori* and are common knowledge to all communicating partners within the TTA. These instants establish the temporal specification of the CNIs.

3. FAULT-CONTAINMENT AND ERROR CONTAINMENT

Central to the field of fault-tolerant computing are the three notions of *failure*, *error*, and *fault (Laprie 1992)*: A *failure* occurs, if the service of a system deviates from its specification. An *error* is an unintended part of the *state* within a system that may lead to a future failure. At last, a *fault* is the cause of the error, and thus, indirectly, of the failure. In any fault-tolerant architecture it is important to distinguish clearly between *fault containment* and *error containment*. *Fault containment* is concerned with limiting the immediate impact of a *single fault* to a defined region within a system, *while error containment* tries to avoid the propagation of the consequences of a fault, the error, out of this defined region. It must be avoided that an error in any one fault-containment region propagates into another fault-containment region that has not been directly affected by the original fault.

3.1 Fault Containment

The notion of a *fault-containment region (FCR)* is introduced in order to delimit the immediate impact of a *single fault* to a defined subsystem of the overall system. *A fault-containment region* is defined as the set of subsystems that share one or more common resources and may be affected by a single fault. Since the immediate consequences fault in any one of the shared resources in an FCR may impact all subsystems of the FCR, the subsystems of an FCR cannot be considered to be independent of each other. (Kaufmann, B. et al. 2000). In the context of this paper the following shared resources that can be impacted by a fault are considered:

- Computing Hardware
- Power Supply
- Timing Source
- Clock Synchronization Service
- Physical Space

For example, if two subsystems depend on a single timing source, e.g., a single oscillator or a single clock synchronization algorithm, then these two subsystems are not considered to be independent and therefore belong to the same FCR. Since this definition of independence allows that two FCRs can share the same design, e.g., the same software, design faults in the software or the hardware are not part of this fault-model. In a safety-critical TTA a node (Figure 1) is considered to form a single FCR.

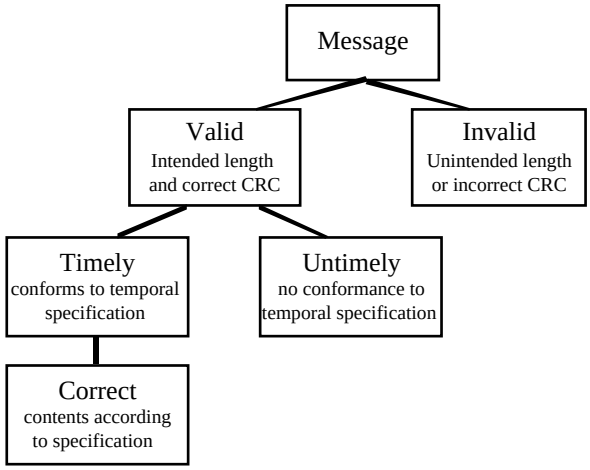


Figure 4: Classification of Messages

In the TTA nodes communicate by the exchange of messages across replicated communication channels. Each one of the two channels transports independently its own copy of the message at about the same time from the sending CNI to the receiving CNI. The start of sending a message by the sender is called the *message send instant*. The termination of receiving a message by the receiver is called the *message receive instant*. In the TTA, the *intended message send instants* and the *intended message receive instants* are *a priori* known to all communicating partners. A message contains an atomic data structure that is protected by a CRC. The TTA architecture makes the assumption that a CRC cannot be forged by a fault. The CRC is thus considered as a form of authentication signature of a message. A message is called a *valid message* if it contains a data structure with a correct CRC (see Figure 4). A message is called a *timely message* if it is a valid message and conforms to the temporal specification. A message with does not conform to the temporal specification is an *untimely message*. A timely message is a *correct message*, if its data structure is in agreement, both at the syntactic and semantic level (Kopetz and Suri 2002), with the specification. A message with a message length that differs from its specification or with an incorrect CRC is called an *invalid* message.

3.2 Error Containment

An error that is caused by a fault in the sending FCR can propagate to another FCR via a message failure, i.e., a sent message that deviates from the specification. A message failure can be a *message value failure* or a *message timing failure* (Cristian, Aghili et al. 1985). A message value failure implies that a message is either invalid or that the data structure contained in a valid message is incorrect. A *message timing failure* implies that the *message send instant* or the *message receive instant* are not in agreement with the specification.

In order to avoid error propagation by way of a sent message error-detection mechanisms that are in different FCRs than the message sender are needed. Otherwise, the error detection mechanism may be impacted by the same fault that caused the message failure. The TTA distinguishes between *timing failure detection* and *value failure detection*. A guardian that is part of the TTA performs the timing-failure detection. Value failure detection is in the responsibility of the host computer.

Timing failure detection: The guardian is an autonomous unit that has *a priori* knowledge of all intended message sent and receive instants. Each one of the two

replicated communication channels has its own independent guardian. A receiving node within the TTA judges a sending node as *operational,* if it has received at least one timely message from the sender around the specified receive instant. It is assumed that a guardian cannot forge a CRC and cannot store messages, i.e., it can only output a valid message across one of its output ports if it has received a valid message on one of its input ports within the last δ time units. A guardian transforms a message, which it judges untimely into an invalid message by cutting off its tail. Such a truncated message will be recognized as invalid by all correct receivers and will then be discarded. A judgement by the guardian may be due to a message timing failure or to a fault in the guardian. Figure 5 depicts a message flow from a sender to two different receivers via the two replicated communication channels, each one with its own guardian.

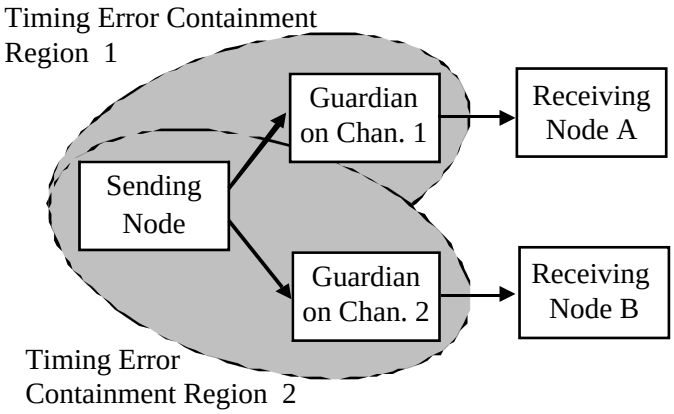


Figure 5: Fault Containment Regions (blocks) and
Timing Error Containment Regions (Ellipses)

Ideally, each one of the boxes in Fig. 5 should form its own independent fault-containment region (FCR), while the ellipses from two distinct timing-error containment regions. A timing-error containment region consists of two independent FCRs, one FCR that produces a message and another independent FCR for timing- failure detection. After a timing failure has been detected it can be isolated to avoid any error propagation.

Value failure detection: Value-failure detection is not a responsibility of the TTA, but in the responsibility of the host computers (Bauer and Kopetz 2000). For example, value failure detection and correction can be performed in a single step by triple modular redundancy (TMR). In this case three replicated deterministic senders, placed in three different FCRs, perform the same operations in their host computers (Fig. 1). They produce—in the fault-free case—correct messages with the same content that are sent to three replicated receivers that perform a majority vote on these three messages (Actually, at the communication level six messages will be transported, one from each sender on each of its two channels).

Value-failure detection and timing-failure detection are not independent in the TTA. In order to implement a TMR structure at the application level in the host computers for value-failure masking, the integrity of the timing of the architecture must be assumed. An intact sparse global time-base is a prerequisite for the system-wide definition of the *distributed state* (see Section 3.2) which again is a prerequisite for masking value-failures by voting. The separation of timing-failure handling and value-failure handling has beneficial implications for resource requirements. To handle *k* arbitrary failing nodes

3k+1 nodes are required (Lamport, Shostak et al. 1982). All nodes of a cluster, independent of their involvement in a particular application system, can contribute to the handling of timing failures at the architectural level. Once a proper global time is available, TMR can be implemented for value-failure masking by only 2k+*1* synchronized nodes in a particular application subsystem. This separation of timing-failure and value-failure handling thus reduces the number of application subsystems needed for fault tolerance of an application from *3k+1* to *2k+1*.

To summarize, fault containment and error detection is achieved in the TTA in three distinct steps. At first, fault containment is achieved by proper architectural decisions concerning resource sharing in order to provide independent fault-containment regions. In a second step, timing error propagation out of an FCR is avoided at the architecture level by the guardians in the intelligent star couplers. In a third step, value failure handling is performed at the application level by voting.

4. CRITICAL FAILURE MODES

In this Section critical failure modes that should be addressed by any architecture for a safety-critical environment are discussed. A failure mode of an FCR is considered as *critical*, if it impacts the remaining correct nodes in such a way that the consistency of the distributed computing base among the nodes that are outside the affected FCR is lost. The focus is on a single fault during a fault-recovery interval Δd. After the recovery interval Δd the architecture has recovered from the consequences of this fault and can tolerate a further fault (provided enough resources remain operational). A set of nodes is defined as Δd-*consistent* if Δd time units after the occurrence of failure all remaining correct nodes have the same view about this failure event. The failure events that are of particular interest to this analysis are transient crash failures of hosts and communication system failures, because these failure events are the most probable in distributed systems. In the TTA the duration of Δd is less than two TDMA rounds. In the rest of this paper, whenever the term “consistent” is used, the above definition is implied.

If the TTA is used in safety-critical applications, *arbitrary* failures of an FCR should be addressed at the level of the architecture (Lala and Harper 1994), since it is difficult to constrain the failure modes of a system-on-a-chip (SOC). According to (Constantinescu 2002), the failure modes of future VLSI devices cannot be characterized by the “single-bit-flip” failure model. The further miniaturization of the devices on the silicon die leads to an increased rate of transient multi-bit failures and intermittent failures that are difficult to contain within a single die, which forms a single FCR.

In this paper, the following failure modes (failure modes (i) to (iv) are subcategories of the arbitrary failure mode) of an FCR (a single sytem-on-a-chip) are considered:

(i) Babbling idiot failures

(ii) Masquerading failures

(iii) Slightly-off-specification (SOS) failures

(iv) Crash/Omission (CO) failures

(v) Massive transient disturbances

In the analysis of failure modes (i) to (iv) it is assumed that a fault impacts a single FCR only. Failure mode (v) is special case that is explained at the end of this Section.

4.1 Babbling-Idiot Failures

A babbling idiot failure of an FCR occurs if the FCR starts sending *untimely messages* as defined in Section 3. In a multicast time-triggered communication topology that contains a broadcast channel such a babbling FCR can interfere with the communication of the correct nodes. If an FCR exhibits permanent babbling-idiot failures on both channels (this is in principle possible, since both channels are in the same FCR) any further communication among the correct nodes becomes impossible. The TTA detects and handles babbling-idiot failures of FCRs by the guardians in the communication system. The guardian will only open the sending channels during the *a priori* known time-interval that has been allocated to a node.

4.2 Masquerading Failures

A masquerading failure occurs if an erroneous node assumes the identity of another node and causes harm to the system. Systems that rely on names stored in a message to identify the transported message and the information contained therein are vulnerable to masquerading failures. It opens the possibility that a single faulty node can masquerade other nodes, without the receiver having a chance to detect the fault. For example, if a bit in the name of a message to-be-sent that is stored in the sending node is incorrect, this message could, after arrival at its destination, overwrite correct messages at correct receivers that have arrived from other correct nodes. This problem is discussed at some length in the safety critical SafeBus protocol (Driscoll and Hoyme 1993) p.36: *Any protocol that includes a destination memory address in a message is a space-partitioning problem.* Masquerading failures are difficult to diagnose if the receiver has no knowledge about the identity of the sender of a message. The internal state of any receiver can be corrupted by an incorrect message from a faulty sender that may not even have a functional relationship to this receiver. Masquerading failures pose a serious vulnerability in systems that achieve fault-tolerance by voting on replicated messages. A single faulty node can masquerade as a set of different nodes and thus invalidate the basic independence assumption of the senders. If a masquerading node remains undetected, its errors can propagate to all other nodes of the system. As a consequence the complete control system can be lost.

4.3 Slightly-off-Specification (SOS) Failures

Slightly-off-specification failures can occur at the interface between the analog and the digital world. Assume the situation as depicted in Figure 4. The specification requires that every correct node must accept an analog input signal if they are within a specified receive window of a parameter (e.g., timing, frequency, or voltage). Every individual node will have a wider actual receive window than the one specified in order to ensure that even if there are slight variations in manufacturing it can accept all input signals as required by the specification. These actual receive windows will be slightly different for the individual nodes, as shown in Fig. 6. If an erroneous FCR produces an output signal (in time or value) slightly outside the specified window, some nodes will correctly receive this signal,

while others might fail to receive this signal. Such a scenario will result in an inconsistent state of the distributed system.

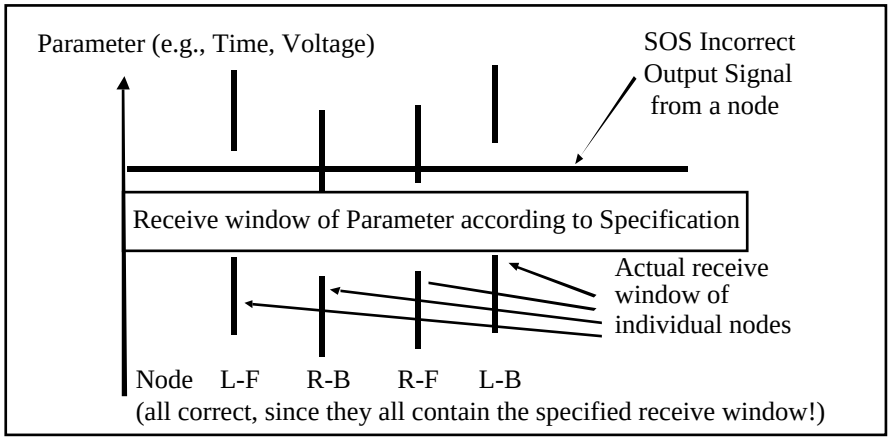


Figure 6: Slightly off Specification (SOS) failure

Example: Consider a brake-by-wire system where four receiving nodes are at the four wheels of a car (L-F: left front, R-B: right back, R-F: right front, L-B: Left back), as shown in Figure 6. In this example an SOS output failure of the "brake master" will cause confusion in the distributed system. According to this example, the *L-Front* and the *L-Back* node will accept a SOS message, while the *R-Back* and *R-Front* node will discard this message. In a brake-by-wire system, such an inconsistency can become safety relevant.

In the TTA the following three types of SOS failures can occur:

(i) SOS value failure

(ii) SOS coding failures

(iii) SOS send-instant failures.

An *SOS value failure* occurs if the signal level of the outgoing message is SOS faulty. Some receivers may still correctly decode such an SOS faulty signal, while others may not be able to decode this signal at all. Since both outgoing channels of an FCR depend on the same power supply, the probabilities of SOS value failures on both channels are correlated.

An *SOS coding failure* occurs if the bit stream from the sender is at the border of the coding specification, e.g., the frequency is SOS faulty. Since both channels are driven by the same oscillator the probabilities for the occurrence of an SOS coding failure on both channels are not independent.

An *SOS send-instant failure* occurs if the send-instant of a message transmission (see Section 3.1) is SOS faulty. A message that is SOS send-instant faulty may be accepted by some nodes and rejected by others. Again, SOS send-instant failures on the two channels are correlated.

4.4 Crash/Omission Failures

A widely accepted fault-model for distributed systems assumes that a fault manifests itself as either a crash failure of a node or an omission failure of the communication (CO

failure). CO failures are the most common failures in distributed system. According to this fault model a node either operates correctly or crashes. The communication system either transports a correct message or fails to transport any message. Most of the available communication protocols, such as for example TCP/IP, are designed to detect and, if possible, to correct CO failures. The consequence of a CO failures is a loss of consistency of the distributed computing base. In a point-to-point communication system an acknowledgement service is provided to detect CO failures. In multi-cast communication system, such as the TTA, a membership service can be designed to detect and identify CO failures. Another mechanism for CO failure detection is the acknowledgement mechanism of the CAN protocol (CAN 1990).

In a multicast environment it is important to distinguish between an omission failure at the sender and an omission failure at one of the receivers. If the sender learns promptly about a local omission failure it can often undo the state-change assumed to have taken place by rolling back to the state before the send operation. In case of an omission failure at one of the receivers, such a rollback is not possible.

Prompt CO failure detection and diagnosis at the architecture level is important in order to inform the application that consistency has been lost, and which unit is responsible for the loss of consistency. The application can then decide what corrective action must be taken.

4.5 Massive Transient Disturbances

Another important fault class in a distributed embedded system, particularly in the automotive domain, is concerned with massive transient disturbances, e.g., those caused by electromagnetic imission (EMI). A massive transient disturbance can cause the temporary loss of communication among otherwise correct nodes that reside in different FCRs. In such a situation the architecture can provide the service of prompt error detection in order that the nodes may take some local corrective action until the transient disturbance has disappeared and the communication service is reestablished. For example, (Thurner and Heiner 1998) report that in an automotive environment a temporary loss of communication of up to 50 msec can be tolerated by freezing the actuators in the positions that were taken before the onset of the transient disturbance. Massive transient disturbances are critical faults that must be avoided by proper quality engineering, e.g., by shielding the cables or installing fiber optics instead of copper. In a safety-critical distributed system massive transient disturbances must be rare events. From the point of view of the communication system, fast detection of a transient disturbance and fast recovery after the transient has disappeared are important.

The following Table 1 gives an overview of the effects of the critical failure modes:

Failure Mode	Babbling Idiot	Masquerading	SOS	Crash/Omission	Massive Disturb.
Probability of occurrence	very low	very low in hardware probable in software	very low	probable	depends on environment (EMI)
Failure effect	Total loss of communication if fault is permanent	Error propa-gation can cause loss of total system.	Inconsistent state in the distributed system	Inconsistent state in the distributed system	Transient loss of communication while disturbance is present
Possible fault handling at the architecture level	Isolation of faulty unit	Isolation of faulty unit	Isolation of faulty unit	Prompt detection and information of the application	Prompt detection and prompt resynchronization

Table 1: Characteristics of different critical failure modes

5. TIME-TRIGGERED PROTOCOLS

A time-triggered distributed architecture requires protocol and hardware support to establish a precise time base among the nodes of the distributed system. The first such protocol, TTP/C was published in 1993 (Kopetz and Gruensteidl 1993). After the implementation and evaluation of the original TTP/C protocol version, the protocol was refined and cast into the first silicon in 1998 in the context of the X-by-Wire project and the BriteEuram TTA project, both supported by the European Commission. The second improved version of TTP/C silicon, developed by TTTech in cooperation with Audi and Honeywell, has been available on the market since 2002. The advantages of the time-triggered paradigm for the design of dependable distributed systems had been widely appreciated not only by the research community, but also by the industrial community. A number of new time-triggered protocols have appeared on the market. Most notable are the FlexRay protocol, originally proposed by DaimerChrysler and BMW, and the TT-CAN protocol, an extension of CAN proposed by Bosch. Additionally, a time-triggered smart transducer protocol, TTP/A has been developed and is being standardized by the OMG. Out of these four protocols, only TTP/C and FlexRay claim to be designed for safety-critical applications. The analysis thus focuses on the fault-containment and error-detection capabilities of these two protocols.

This Section describes the essential functionality of TTP/C and FlexRay. For the TTP/C part, this description is based on the publicly available specification of the latest TTP/C controller chip (TTTech 2002) and (Kopetz, Bauer et al. 2001). For the FlexRay part, this description is based on the seminar handouts of the FlexRay seminar in Munich on April 16 and 17, 2002 in Munich (FlexRay-group 2002) and on other FlexRay publications that are in the publicly available such as (Berwanger, Ebner et al. 2001). It has to be noted, however, that the precise FlexRay not available to the public.

Both TTP/C and FlexRay provide a global time base by using a distributed fault-tolerant clock synchronization algorithm. Elements common to both algorithms are described in this paragraph. Global time is established by generating a synchronized macro-tick at

each node. This macro-tick is used to determine the sending slots of a node and the TDMA cycle. The ratio between the microticks of the node-local oscillator and the network-wide uniform macrotick is dynamically modified in order to synchronize the macro-ticks in the different nodes to within the required precision. Clock synchronization is partitioned into three phases (Kopetz 1997), p.61. In a first phase, the clock offset, measured in local microticks of a receiver, between the clock of the sender that sent a selected synchronization frame, and the clock of the receiver is measured and recorded. In a second phase, these clock differences are sorted, the extremes are discarded and a *synchronization ensemble* is formed. A synchronization algorithm calculates the local clock correction value out of the values contained in the synchronization ensemble. In the third phase the calculated value is used for the local clock adjustment by modifying the node-local microtick/macrotick relation.

5.1 TTP/C

Message Transmission: The TTP/C protocol uses a strict time-division-multiple-access (TDMA) pattern to control the media access to the two independent communication channels of a node. Time is partitioned into a sequence of slots of varying length (between 2 and 236 bytes of data). Every slot is statically assigned to a node, during which the node is allowed to send its frame. A TDMA round is formed by the sequence of slots. A sequence of predefined TDMA rounds forms a cluster cycle. After the end of a cluster cycle, the periodic access pattern is repeated. Nodes may send different messages during the different TDMA rounds of a cluster cycle. The frame contents, data rate and physical medium may be the same or may differ within the same slot on the two different channels. However, in a safety-critical application the same message must be sent on both channels. A node can only send once in a TDMA round and there is a *static link* between the slot identity and the identity of the sending node. This static link is exploited in the architecture for the purposes of error detection.

Guardian: In a safety-critical TTP/C configuration TTP/C requires an interconnection network with two intelligent star couplers, one for each channel. The guardians are integrated into the star coupler. Each star coupler forms an FCR with its own power supply, distributed clock synchronization system, and timing source. It has *a priori* knowledge of the sending slots that are assigned to each node and opens the gate only during the sending slot of a node. A star coupler must reshape the incoming signal stream based on its own clock and power supply in order to mask SOS failures of a sending node.

Clock Synchronization: In TTP/C the frames that are used for clock synchronization are selected on the basis of knowledge at the receiver's site in order to make it impossible for a faulty sender to deceive the receiver. TTP/C uses the fault-tolerant average (FTA) algorithm in order to calculate a fault-tolerant average out of four statically selected *time-keeping* clocks. If one of these pre-selected time-keeping clocks fails, a standby clock replaces the failed clock. TTP/C uses static rate correction but avoids dynamic rate correction of the time-keeping clocks in order to avoid oscillations in the time base. A theoretical analysis and extensive experiments have shown that dynamic rate correction is not needed if standard crystal oscillators are used for the time-keeping clocks (Schwabl 1988). Low quality clocks, that are not time-keeping clocks, may use dynamic rate correction since they do not influence the characteristics of the global time-base.

Error Detection: In TTP/C the receiver of a frame has knowledge about the physical identity of the sender of a frame. This knowledge is used to provide error-detection capabilities at the architecture level, such as a membership service and a clique avoidance service.

5.2 FlexRay

FlexRay is actually a combination of two protocols, the static time-triggered FlexRay protocol and the dynamic event-triggered Byteflight protocol. The operation of these two protocols is separated in the domain of time. The two protocols use different frame formats. Since only the time-triggered FlexRay protocol is recommended for use in safety-critical applications, this description deals with on the static time-triggered FlexRay protocol only.

Message Transmission: According to (FlexRay-group 2002) FlexRay distinguishes in its time-triggered part between *slots*, *cycles*, *frames* and *messages* and identifies a slot by a *slot-ID*, a cycle by a *cycle-ID*, a frame by a *frame-ID*, and a message by a *message-ID*. The cycle-ID, the frame-ID, and the optional message-ID are part of every time-triggered FlexRay frame.

In the time-triggered part of FlexRay, the communication pattern is periodic. It is partitioned into cycles and slots. Every slot of a cluster is identified by a unique slot-ID. All slots within a cluster must have the same duration. The slot timing is identical on both channels. Depending on the particular configuration, a statically defined number of slots forms a cycle which can be considered a TDMA round. Slots and cycles are increased in modular arithmetic as time progresses up to the maximum slot number (depends on the particular configuration) that form a cycle and the maximum cycle number. Global time is thus measured by a concatenation of cycle-ID and slot-ID.

A sending node transmits a frame with a given *frame-ID* whenever the current *slot-ID* matches the ID of the pending frame. The frame contents may be the same or may differ within the same slot on the two different channels. A slot remains empty if no frame is configured for it. The same node may occupy up to 16 slots within a cycle and may thus send up to 16 frames within one cycle. The *message-ID* is used for local filtering and has no role in the execution of the protocol.

Guardian: In FlexRay the guardian is assigned to a particular node and is clocked by this node. The node indicates to the guardian the start of a cycle. From there onwards, the guardian maintains its own time-base derived from a local RC oscillator. The guardian has knowledge about the slot-ID that is assigned to its node and opens the "sending gate" only during this *a priori* assigned slot-ID. This knowledge is downloaded to the guardian during a configuration phase. Since the guardian does not contain an independent clock synchronization and an independent power supply, the node and the guardian form a single FCR.

Clock Synchronization: In FlexRay the frames that are used for clock synchronization are selected on the basis of the contents of a message (Sync Bit set). In FlexRay the synchronization ensemble is formed by the eliminating the j largest and j smallest clock values of the selected clocks. FlexRay uses the fault-tolerant midpoint (FTM) algorithm in order to calculate the correction value out of the synchronization ensemble. The FTM algorithm calculates its correction value by using only the two clocks at the extreme ends of the synchronization ensemble, while TTP/C's FTA algorithm calculates the correction

value by averaging all values of the synchronization ensemble. For four clocks, the FTM algorithm and the FTA algorithm give the same results. FlexRay performs a dynamic rate correction to compensate the drift-rate of low-quality clocks. It is not clear from the available documentation how the problem of an oscillation of the time base, caused by the dynamic rate correction, is avoided.

Error Detection: In FlexRay the receiver of a frame *has no knowledge about the physical identity of the sender* of a frame. This increases flexibility, but is traded against a reduction of error-detection capabilities at the architecture level, since the receiver can never identify a faulty sending FCR on the basis of a faulty or missing frame.

6. FAILURE HANDLING IN TTP/C AND FLEXRAY

Section 4 has introduced a set of critical failure modes of a single component that have the potential to impact the integrity of the distributed control system as a whole. Any architecture that is intended for use in safety-critical applications must address the system consequences of these failure modes. It must be convincingly shown that either the probability of such a *single failure* occurring is so low that the risk of the loss of the complete control system is acceptable or that mechanisms are provided that can contain the consequences of these failures. In this Section it is thus analyzed how the different critical failure modes introduced in Section 4 are handled by TTP/C and FlexRay at the architecture level.

6.1 Babbling Idiot Failures

TTP/C: In a safety-critical application, the sending node is implemented in an FCR that is different from the FCR of the guardian, i.e., in the intelligent star coupler, in order to isolate babbling sending nodes from the rest of the cluster. This architectural decision is substantiated by extensive fault-injection experiments (Ademai 2002).

In a prototype implementation of the TTA the sending node and the guardian were implemented in the same FCR in order to reduce the implementation cost by sharing the silicon die, a single power supply and a single version of the clock-synchronization algorithm for both, the sending node and the guardian. Two separate oscillators to control the timing of the service-providing unit and the guardian were deployed. This implementation was subjected to extensive fault-injection experiments (Ademai 2002), using software-implemented fault-injection, pin-level fault injection and heavy-ion fault injection. Some fail-silence violations were recorded in the heavy-ion fault experiments. It is assumed that in the heavy-ion experiments a single fault can be the cause of multiple errors that are difficult to detect systematically within a single FCR. In another set of experiments, the sending node and an autonomous intelligent guardian were implemented in two independent FCRs (Ademai 2002). In these experiments no fail-silence violation has been observed so far.

FlexRay: According to (FlexRay-group 2002) the guardian and the sending node are planned to be implemented in the same FCR (FlexRay-group 2002). There is no documented evidence provided, how the correlated failure of a node and the associated guardian is handled.

6.2 Masquerading Failures

TTP/C: The static *a prior* known association between sending slot and physical unit makes it impossible for a faulty physical unit to masquerade as another physical unit.

FlexRay: The dynamic association between sending slot and physical unit makes it possible for a faulty physical unit to masquerade as another physical unit. For example, the FlexRay clock synchronization requires that the Sync-Bit must be sent in the frames of both channels and at most one Sync-Bit frame per physical node may be sent. But which unit is capable to identify a faulty sender if the receiver has no knowledge about who is the physical sender of the faulty Sync-Bit frame? In the available documentation about the FlexRay architecture no mention of any mechanisms for the detection of masquerading faults was found.

6.3 Slightly-off-Specification (SOS) Failures

TTP/C: The central star coupler with its integrated guardian reshapes the incoming signal from a node in order to mask an *SOS value failure* or an *SOS coding failure*. If the guardian accepts the SOS frame from the sender correctly, it will send a correct message to all other receivers. If a guardian accepts the incoming SOS frame incorrectly, it will send an incorrect frame to all receivers. The guardian will thus transform a SOS faulty frame into either a consistently correct frame or a consistently incorrect frame. A SOS send instant failure (see Section 3.1) is detected by comparing the send instant of the incoming frame with the *a priori* known send instant and making the message invalid (by truncating the tail) if a send instant failure has been detected. An SOS failure of the guardian has no system-wide effect, since the replicated guardian can be assumed to be correct (single fault model). TTP/C requires only one correct channel for its operation.

FlexRay: In the available documentation about the FlexRay architecture there is no mention of any mechanisms for SOS failure detection or SOS failure masking.

6.4 Crash/Omission (CO) Failures

TTP/C: Crash/Omission Failures are detected by the TTP/C membership protocol which is an integral part of TTP/C. Under the assumption of a single fault, the membership protocol diagnosis an omission failure as an incoming or outgoing link failure. It is up to the application to decide how it should react to this evolving inconsistency.

FlexRay: In the available documentation about the FlexRay no mention of any mechanisms for CO failure detection at the architecture level was found. FlexRay recommends to perform CO failure-detection and diagnosis within each application system.

6.5 Massive Transient Disturbances

TTP/C: Massive transient disturbance are detected in TTP/C by the membership protocol and by the clique avoidance service. After the massive transient disturbance has disappeared the protocol can restart immediately with the time base that has been maintained within the host computer.

FlexRay: In the available documentation about the FlexRay architecture no mention of any mechanisms for protocol-based transient disturbance detection, other than the CRC, was found. Massive transient disturbance handling seems to be left to the application.

Critical Failure Mode	TTP/C Failure Handling	FlexRay Failure Handling
Babbling Idiot	Guardian in the star coupler, which forms an **independent** FCR, isolates babbling idiot failures	Relies on fault isolation by guardian which is in the **same** FCR as the sending node. Potential for correlated failures.
Masquerading	Static assignment of slot position to physical node identity eliminates possibility for masquerading faults	No provision within the published architecture
SOS	Reshaping of incoming signal stream by the star coupler masks SOS failures in time and value domain. Detection of send-instant faulty message by the star coupler	No provision within the published architecture
CO	Membership protocol detects and diagnosis CO failures promptly.	Error detection and error handling at the application level are proposed
Massive Transients	Membership protocol and clique avoidance protocol detect promptly the onset of massive transients	Error detection and error handling at the application level are proposed.

Table 2: Comparison of Critical Failure Mode Handling in TTP/C and FlexRay

7. CONCLUSION

An architecture for ultra-dependable distributed real-time systems must be driven by the concerns for fault containment and prompt error detection and isolation. In the time-triggered architecture it is possible to take a two-step approach to solve this problem: In a first step, timing errors are isolated at the architecture level in order to preserve the temporal integrity of the remaining distributed computing base. In a second step value errors are masked at the application level by making use of the resources that have not been affected by a fault. In order to maintain the temporal integrity of the distributed computing base, certain error-detection mechanisms must be provided at the architectural level. This paper has identified five critical failure modes of a fault-containment region that should be addressed at the level of the architecture. Table 2 shows how TTP/C and FlexRay handle these critical failure modes.

ACKNOWLEDGEMENT

This work has been supported in part the EU project NEXT TTA and the EU project FIT. I would like to thank members of the IFIP WG 10.4 on dependable computing for constructive comments on an earlier version of this paper, particularly concerning the importance of proper handling of masquerading faults.

REFERENCES

Ademai, A. (2002). FIT fault-injection results. Vienna, Technical University of Vienna.

Bauer, G. and H. Kopetz (2000). Transparent Redundancy in the Time-Triggered Architecture. Dependable Systems and Networks (DSN 2000), New York, IEEE Press.

Berwanger, J., C. Ebner, et al. (2001). FlexRay--The Communication System for Advanced Automotive Control Systems. SAE World Congress, Detroit, SAE Press paper 2001001-0676.

CAN (1990). Controller Area Network CAN, an In-Vehicle Serial Communication Protocol. SAE Handbook 1992, SAE Press. **SAE J1583:** 20.341-20.355.

Constantinescu, C. (2002). Impact of Deep Submicron Technology on Dependability of VLSI Circuits. Proc. of the 2002 International Conference on Dependable Systems and Networks, Washington D.C., IEEE Press.

Cristian, F., H. Aghili, et al. (1985). Atomic Broadcast: From simple message diffusion to Byzantine agreement. Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15), Ann Arbor, Michigan.

Driscoll, K. and K. Hoyme (1993). "SafeBus for avionics." IEEE Aerospace and Electronics Systems Magazine **8**(3): 34-39.

FlexRay-group (2002). Handouts of the International FlexRay Workshop. Munich, FlexRay Consortium at www.flexray-group.com.

Hopkins, A. L., T. B. Smith, et al. (1978). "FTMP: A Highly Reliable Fault-Tolerant Multiprocessor for Aircraft Control." Proc. IEEE **66**(10): 1221-1239.

Kaufmann, L. F., S. B., et al. (2000). Modeling of Common-Mode Failures in Digital Embedded Systems. Proc. of the Reliability and Maintainability Symposium 2000, Los Angeles, Cal., IEEE Press.

Kopetz, H. (1992). Sparse Time versus Dense Time in Distributed Real-Time Systems. Proc. 14th Int. Conf. on Distributed Computing Systems, Yokohama, Japan, IEEE Press.

Kopetz, H. (1997). Real-Time Systems, Design Principles for Distributed Embedded Applications; ISBN: 0-7923-9894-7, Third printing 1999. Boston, Kluwer Academic Publishers.

Kopetz, H., G. Bauer, et al. (2001). Tolerating Arbitrary Node Failures in the Time-Triggered Architecture. SAE 2001 World Congress, Detroit, Mich, SAE Press.

Kopetz, H. and G. Gruensteidl (1993). TTP - A Time-Triggered Protocol for Fault-Tolerant Real-Time Systems. Proc. 23rd IEEE International Symposium on Fault-Tolerant Computing (FTCS-23), Toulouse, France, IEEE Press.

Kopetz, H. and N. Suri (2002). Compositional Desing of Real-Time System: A Conceptual Basis for the Specification of Linking Interfaces, Technical University of Vienna.

Lala, J. H. and R. E. Harper (1994). "Architectural Principles for Safety-Critical Real-Time Applications." Proc. of the IEEE **82**(1): 25-40.

Lamport, L., R. Shostak, et al. (1982). "The Byzantine Generals Problem." ACM Trans. on Programming Languages and Systems **4**(3): 382-401.

Laprie, J. C., Ed. (1992). Dependability: Basic Concepts and Terminology - in English, French, German, German and Japanese. Dependable Computing and Fault Tolerance. Vienna, Austria, Springer-Verlag.

Leveson, N. G. (1995). Safeware: System Safety and Computers. Reading, Mass., Addison Wesley Company.

Mesarovic, M. D. and Y. Takahara (1989). Abstract Systems Theory, Springer Verlag.

Rushby, J. (2001). A comparison of Bus Architectures for Safety Critical Embedded Systems, SRI International. **2001**.

Saltzer, J., D. P. Reed, et al. (1984). "End-to-End Arguments in System Design." ACM Transactions on Computer Systems **2**(4): 277-288.

Schwabl, W. (1988). The Effect of Random and Systematic Errors on Clock Synchronizatin in Distributed Systems, Technical University of Vienna, A 1040 Vienna, Treitlstrasse 3/182.

Suri, N., C. J. Walter, et al., Eds. (1995). Advances in Ultra-Dependable Systems, IEEE Press.

Thurner, T. and Heiner (1998). Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. FTCS 28, IEEE Press.

TTTech (2002). Time-Triggered Protocol TTP/C High Level Specification Document. Vienna, Austria, TTTech. URL: www.tttech.com.