THE FAULT HYPOTHESIS FOR THE TIME-TRIGGERED ARCHITECTURE

H. Kopetz[1]

[1]*Vienna University of Technology, Austria*
hk@vmars.tuwien.ac.at

Abstract A precise fault-hypothesis is essential for the design and validation of a safety-critical computer system. The fault-hypothesis must specify the fault-containment regions (FCRs), the assumed failure modes of the FCRs with their respective failure frequencies, the error detection latency and the time-interval that is required in order that an FCR can repair the state corruption that has occurred as a consequence of a transient fault. After a general discussion of the detailed contents of the fault-hypothesis document, this paper presents the fault-hypothesis that has formed the basis for the design of the time-triggered architecture. The time-triggered architecture is a distributed architecture that has been developed for the control of safety-critical embedded applications.

Keywords: Safety-Critical Systems, Fault-Hypothesis, Error Detection, Fault-tolerance, Error-Containment, State Repair

1. Introduction

Ultra-dependable computer systems that are deployed in safety-critical applications are expected to exhibit a mean-time-to-failure (MTTF) of better than 10^9 hours [Walter et al., 1995], i.e. more than 100 000 years. Although this number has its origin in the dependability requirements of a *fly-by-wire system,* it is applicable to the automotive domain as well. It has been stipulated that the dependability requirements for a *drive-by-wire* system are even more stringent than the dependability requirements for fly-by-wire systems, since the number of exposed hours of humans is higher in the automotive domain.

It is *impossible* to gain confidence about a system reliability of 100 000 years by testing [Littlewood and Strigini, 1993]. A consequence of the *untestability* of 10^{-9} systems is the need to analyze critical algorithms by formal methods in order to convince certification authorities of the correctness of the algorithms within the specified operational envelop. Since the observed *mean-time to fail* of hardware components is two orders of magnitude below the aimed-for reliability at the system level, the safety argument must be based on experimental data about the component reliability and analytical arguments taking into account the redundancy in the fault-tolerant system structure. Even a very low correlation in the failure probabilities of replicated subsystems has a significant effect on the system reliability of ultra-dependable systems. The system design must thus assure that replicated subsystems of an architecture for ultra-dependability systems *fail independently* of each other.

It is the objective of this paper to present the fault hypothesis of the time-triggered architecture (TTA). In the following section we argue why a precise fault-hypothesis is essential for the design of a safety critical system and outline the contents of such a precise fault hypothesis. Section three gives a short overview of the time-triggered architecture. Section four presents the fault hypothesis of the TTA with respect to hardware faults, while Section five is devoted to the discussion of the fault-hypothesis with respect to software faults. The paper finishes with a conclusion in Section six.

2. The Fault Hypothesis

The fault hypothesis is a statement about the assumptions made concerning the types and number of faults that a fault-tolerant system is expected to tolerate [Laprie, 1992]. The fault hypothesis divides the fault space into two disjoint partitions: the partition of *covered faults* and the partition of *uncovered faults.* The covered faults are those faults that are contained in the fault-hypothesis and are addressed during the system design. The occurrence of a covered fault during system operation should not have an adverse effect on the availability of the *safety-critical* system functions. The occurrence of an uncovered fault can lead to critical system failure, since no mechanisms are provided to protect against uncovered faults. During system validation it must be shown that uncovered faults are *rare events.*

2.1 Why do we need a precise Fault Hypothesis?

Before the design of a safety-critical system can commence, a precise fault hypothesis is needed for the following reasons:

1 **Design of the Fault-Tolerance Algorithms:** Without a precise fault-hypothesis it is not known which fault-classes must be addressed during the system design [Avizienis, 1997].

2 **Assumption Coverage:** There is a probability that the assumptions that are contained in the fault hypothesis are not met by reality. This probability is called the assumption coverage [Powell, 1992]. The *assumption coverage* predicts the probability of failure of a *perfectly designed* fault-tolerant system. Without a precise fault-hypothesis, the probability for

the occurrence of uncovered faults cannot be predicted. The assumptions that form the fault-hypothesis must be carefully scrutinized and in a safety-critical system it must be demonstrated that the assumption coverage is significantly better than 10^{-9}/hour.

3 **Validation:** The implementation of the fault-tolerance mechanisms can only be validated, if it is precisely known which faults must be tolerated by the given system and which faults are not expected to be tolerated, since they are outside the scope of the given implementation.

4 **Certification:** Without a precise fault hypothesis it is impossible to certify the correct operation of a fault-tolerant system.

5 **Design of the Never-Give-Up (NGU) Strategy:** In a safety-critical application, the control system should never *give up,* even if the fault-hypothesis is violated by reality. In a properly designed fault-tolerant system chances are high that a violation of the fault hypothesis is caused by a correlated shower of *external transient faults* or by a *Heisenbug* and that a fast restart of the system will be successful. The activation of the restart mechanism must be activated by an NGU algorithm. Such an NGU algorithm can only be designed if a precise fault hypothesis is available.

2.2 Contents of the Fault Hypothesis

In the following Section we elaborate on the required contents of a fault hypothesis w.r.t. hardware faults of a distributed real-time control system that is intended for safety-critical applications. A safety critical distributed real-time system consists of a set of node computers that are interconnected by replicated communication channels (see Figure 1).

Specification of the Fault Containment Regions (FCR). 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 a part of the system that may be affected by a *single fault.* The probability of failure of two different FCRs failing at the same time should be independent, i.e. there should not be any correlation of the failure probabilities of different FCRs, Since the immediate consequences of a 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 and cannot be considered to form their own FCRs [Kaufmann and Johnson, 2000]. In the context of this paper the following shared hardware resources that can be impacted by a fault are considered:

- Computing Hardware

- Power Supply
- Timing Source
- Cock Synchronization Service
- Physical Space

For example, if two subsystems depend on a single timing source, e.g., a single oscillator, 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 distributed real-time system consisting of a set of SoCs (System on a Chip) node computers, a complete node computer must be considered to form a single FCR, since all correlated failures of two subsystems residing on the same silicon die cannot be eliminated: the subsystems residing on a single die share the same physical space, the same silicon substrate, the same manufacturing mask and manufacturing process, the same ground and power supply, probably the same timing source etc. There is a non-negligible probability that a fault in any one of these resources will affect both subsystems simultaneously.

A communication channel connecting the nodes of the distributed system can be formed by a bus, a ring, a star or any other interconnection structure. From the point of view of fault-containment, such a channel forms also a single FCR in a safety-critical environment.

Failure Modes. A failure mode specifies the type of failure that may occur if an FCR is impacted by a fault. In the literature different failure modes of an FCR are introduced from *restricted* to *unrestricted* [Laprie, 1992]. The most restricted failure mode is a fail-silent failure, i.e. where the assumption is made that an FCR either operates correctly or is silent. The most unrestricted failure mode is a Byzantine failure, where no assumptions is made about the behavior of a faulty component. Every restriction in the failure mode, i.e. every assumption about the behavior of a faulty component must be scrutinized w.r.t. the assumption coverage. It follows that a system that can tolerate an unrestricted failure mode of an FCR will lead to a better assumption coverage than a system with restricted failure modes.

Temporal Properties of Faults. Another classification considers the temporal properties of faults. We distinguish between the following five types of faults in the temporal domain:

1 **Transient fault:** A transient fault is caused by some random event. An example for a transient is a SEU (single event upset caused by a radioactive particle) [Constantinescu, 2002].

2 **Intermittent fault:** An intermittent fault is considered to be a *correlated sequence of transient faults* that is caused by some single physical degradation of a component. An example of an intermittent fault is the partial degradation of the junction of a transistor on a chip (e.g., caused by oxidation) that causes sporadic load dependent or data dependent errors. Experimental data show that an intermittent fault is likely to eventually produce a permanent fault [Normand, 1996].

3 **Soft permanent fault:** A soft permanent fault is a corruption of the h-state or of the i-state within a component [Kopetz, 1997] without causing any permanent damage to the component. For example, a corruption can be caused by a single-even-upset (SRU) [Kaufmann and Johnson, 2000]. The repair of the erroneous data structure eliminates the soft-permanent fault without any further effect on the hardware.

4 **Permanent fault:** A permanent fault occurs, if the hardware of an FCR brakes down permanently. An example for a permanent fault is a broken wire.

5 **Massive transient disturbance:** A massive transient disturbance is a transient an external occurrence (e.g., a powerful imission of electromagnetic radiation) that results in the correlated failure of two or more communication channels and possibly some of the nodes. Whereas failure mode 1 to 4 relate to *internal* faults, failure mode 5 is concerned with an *external* fault. The probability for the occurrence of an external fault depends on the characteristics of the system environment, not on the design of the system per se.

Failure Frequency. The third part of the fault-hypothesis is concerned with the frequency of failures of the identified failure modes. Whereas failure rate data of electronic components w.r.t. permanent failures are available in the literature [Pauli et al., 1998], it is much more difficult to get consistent failures for intermittent and transient faults. One reason for this difficulty lies in the fact that transient failures often depend an physical location or on a particular geometry which is difficult to reproduce. For example, the SEU soft error rate caused by high-energy particles originating from the space depends on the altitude and the geographical location [Kaufmann and Johnson, 2000].

Error Detection Latency. The consequence of a *fault* is an *error* in the system state [Laprie, 1992]. The time it takes to detect the error is called the error detection latency. The *error detection latency* should be very short in order to be able to process the error before it propagates to a failure that impacts parts of the system that have not been disturbed by the original fault. *Knowing* *that **a failure has occurred** is more important than the actual failure* [Rechtin and Maier, 2002, p. 276].

Recovery Intervals. From the point of view of reliability modeling is important to know the time it takes the system to recover from a transient fault. For a permanent fault that has not caused spare exhaustion it is important to know the time it takes until all correct FCRs have a consistent view of the faulty FCR. For a transient fault there are thee intervals of importance:

- **Transient fault duration:** The time interval between the start of the transient fault and the instant when all communicating partners recognize that the transient fault has disappeared.
- **Protocol recovery interval:** The time it takes until the protocol has recovered and established a consistent view among all communicating partners (e.g., w.r.t. clock synchronization).
- **State repair interval:** The time it takes until an application has recovered from the transient fault and repaired the damage to its h-state (history state).

3. The Time-Triggered Architecture

The Time-Triggered Architecture (TTA) is a distributed architecture for the implementation of safety-critical applications [Kopetz and Bauer, 2003]. A large TTA system can be decomposed into a set of clusters. The structure of a typical single-cluster TTA system is depicted in Figure 1. Such a cluster consists of a *backbone core architecture* of node computers that are interconnected by two replicated communication channels. The media access to the communication channels is controlled by a time-division-multiple-access (TDMA) protocol. All correct nodes of a TTA system have synchronized clocks that are used to construct a fault-tolerant global *sparse time lattice* [Kopetz, 1992]. The guardians in the communication channels of Fig. 1 are needed in order to transform an *arbitrary timing failure* of a node into a *fail-silent failure.*

It is a goal of the TTA to reach–in a properly configured system–a service reliability at the system level of better than 10^{-9} failures/hour. This is achieved by

- structuring the system into a set of independent fault-containment regions (FCRs)
- Provide replica-deterministic operation of the node computers that can operate concurrently in a TMR (triple-modular-redundancy) mode.
- Tolerate an unrestricted (arbitrary) failure each FCR.

- Establish error-detection regions (EDR) such that the consequences of a fault, the ensuing errors, are detected before they corrupt the state of any other independent FCR [Kopetz, 2003]

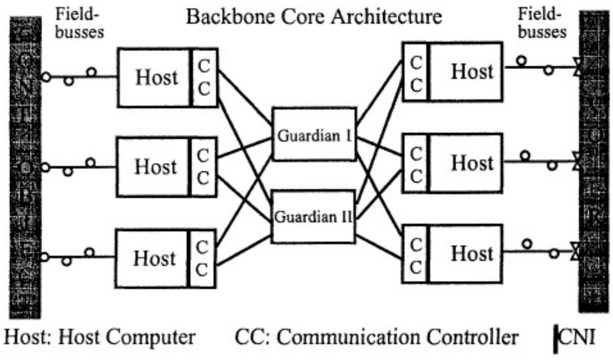

Figure 1. Structure of the Time-Triggered Architecture

A TTA node is supposed to be a *system-on-a-chip (SOC)*. From the hardware point of view, an SOC forms a single fault-containment region that can fail in an arbitrary failure mode. Each node computer contains a Time-Triggered Communication Controller (CC) and a host computer. The interface between the Communication Controller and the host computer is called the Communication Network Interface (CNI). A host computer can support local field-busses (e.g., CAN, LIN, or TTP/A) for the interconnection of the intelligent transducers (sensors and actuators) in the controlled object. The internal structure of a node is depicted in Fig. 2.

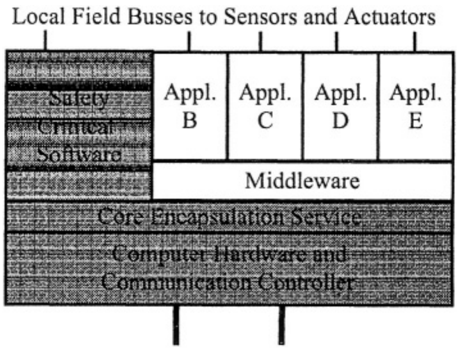
Local Field Busses to Sensors and Actuators
Safety Critical Software
Appl. B
Appl. C
Appl. D
Appl. E
Middleware
Core Encapsulation Service
Computer Hardware and Communication Controller

Figure 2. Structure of a TTA Node

The node of Figure 2 comprises five partitions: the upper leftmost partition provides a safety-critical service, while the other four partitions (B,C,D,E) provide non-safety critical services. Each partition has access to its local sensors and actuators via a local field bus. Given that the node hardware functions correctly and the shaded area is free of design faults, the core encapsulation service ensures that there cannot be any error propagation from the non-critical partitions to the critical partition. The certification for the safety-critical services is thus reduced to the shaded area of Figure 2.

The *middleware partition* in of Figure 2 establishes and monitors the encapsulated execution environments of the non-safety-critical jobs. In addition to a state message interface it provides event messages interfaces to the non-safety critical jobs and emulates legacy interfaces (e.g., a CAN controller interface) such that existing legacy software can be ported with minimal modifications.

The application software which resides within a partition is called a *job*. A set of cooperating jobs, each one possibly at a different node, forms a distributed application subsystem (DAS). The jobs of a DAS communicate via an encapsulated communication service with guaranteed temporal properties. If required, jobs of a DAS may be replicated at different nodes in order that the system provides a required level of fault tolerance in case a node fails. A TTA system may support many different encapsulated DASes that can interact via virtual gateways [Kopetz et al., 2004].

4. Fault Hypothesis w.r.t. Hardware Faults

With respect to hardware faults, the fault hypothesis of the the TTA consists of the following assumptions:

1 A node computer forms a single fault-containment region (FCR). From the point of view of hardware faults, a node is thus considered to be an atomic unit.

2 A physical communication channel including the central guardian forms a single FCR. All virtual channels that are implemented on a physical channel form a single unit of failure

3 A node computer can fail in an arbitrary failure mode. As long as only a single node computer fails, it is not relevant whether the failure is caused by a hardware fault or a software fault.

4 A central guardian distributes the messages received from the node computers. It can fail to distribute the messages, but cannot generate messages on its own (this is called the *distribution assumption*).

5 The permanent failure rate of a node computer or a central guardian is in the order of 100 FIT [Pauli et al., 1998] i.e. about 1000 years.

6 The transient failure rate of a node computer is in the order of 100 000 FIT, i.e, about 1 year. One important mechanism that causes transient failure is an SEU [Kaufmann and Johnson, 2000].

7 One out of about fifty failures of a node computer is *non-fail silent.* The relation of *silent* to *non-silent failures* of a node has been derived from fault injection experiments [Karlsson et al., 1995].

8 The central guardian transforms the non-fail-silent and the slightly-out-of-specification (SOS) failures of the node computers in the temporal domain to fail-silent failures in the temporal domain [Ademaj et al., 2003] (this is called the *SOS assumption*).

9 The detection of a single error is performed by a membership algorithm. The error detection latency is less then two TDMA rounds.

10 The detection of multiple errors is performed by a clique avoidance algorithm. The detection latency is less than two TDMA rounds.

11 The system can recover from a single transient fault within two TDMA rounds.

12 The system can recover from a massive transient that destroys the clock synchronization within 8 TDMA rounds [Steiner et al., 2003] after the transient has disappeared.

13 The state repair time of an application takes an application specific amount of time which must be derived from knowledge about the application software.

There are two important assumptions in this fault hypothesis that must be further investigated the *distribution assumption* and the *SOS assumption.*

The distribution assumption states that the central guardian cannot distribute valid messages without having received a valid message. If the central guardian has no knowledge about how to generate a CRC of a message, the probability that a random fault will produce a random message that is syntactically correct, is generated at the proper time, is of the proper length and contains a proper CRC is far below the 10^{-9} limit.

The validity of the *SOS assumption* has been established by extensive fault-injection experiments [Ademaj et al., 2003]. In more than twenty thousand experiments that resulted in a node failure because of the radiation of the TTA node with α particles, no error propagation has been observed when the system was equipped with a central guardian. In contrast to this, a number of error propagations have been observed when the nodes where protected by a local guardian. Considering the low failure rate for permanent node errors and the results of these experiments it can be concluded that the SOS assumption is valid within the 10^{-9} limit if the system contains a central guardian.

A correctly configured TTA system will recover from a massive external transient that causes correlated transient faults in the worst case *within 8 TDMA* *rounds* after the transient has disappeared. This scenario has been investigated by model checking [Steiner et al., 2004].

Considering the failure rates that have been presented above, the probability that a second independent failure will happen before the recovery from the first failure has been completed is below the 10^{-9} limit.

To summarize, a properly configured TTA system tolerates a single arbitrary hardware failure of any one of its nodes within the 10^{-9} limit. At the moment the TTA implementation of TTTech [TTTech, 1998] is in the process of being certified by the FAA (Federal Aviation Authority) for aerospace applications that are in the highest criticality class.

5. Fault Hypothesis w.r.t. Design Faults

5.1 Bohrbugs versus Heisenbugs

In his classical paper [Gray, 1986], Jim Gray proposed to distinguish between two types of software design errors, *Bohrbugs* and *Heisenbugs.* Bohrbugs are design errors in the software that cause reproducible failures. An example for a Bohrbug is a logic error in a program that causes the program to always take an unintended branch if the same computation is repeated. *Heisenbugs* are design errors in the software that seem to generate quasi-random failures. An example for a Heisenbug is a synchronization error that will cause the violation of an integrity condition (e.g., only one process is active in its critical section) if the temporal relationship of two concurrent processes happens to cause a race condition. A minor change in the temporal interleaving of the two concurrent processes will eliminate this race condition and thus the manifestation of the software error. From a phenomenological point of view, a transient failure that is caused by a Heisenbug cannot be distinguished from a failure caused by transient hardware malfunction.

In a system with state, a Heisenbug can cause a permanent state error [Kopetz, 1997]. The correct operation of the node will resume, as soon as this state error has been eliminated (e.g., by voting over the state in a TMR triade).

Experience has shown that it is much more difficult to find and eliminate Heisenbugs than it is to eliminate the Bohrbugs from a large software system [Eisenstadt, 1997].

5.2 Safety-Critical Design

We assume that the *hardware design,* the *core encapsulation service* and the *safety-critical software* (the *safety-critical design*–see Fig. 2) is free of design errors. In order to justify this strong assumption, this *safety critical* design must be made as small and simple as possible in order that it can be analyzed formally. In order to reduce the probability of Heisenbugs, the safety critical design should be time-triggered. The control signals are derived from the progression of the *sparse global time base* which guarantees that all replicated components will visit the same state at about the same physical time.

Concerning Heisenbugs, the above assumption is more stringent than needed. If there would be a Heisenbug in the safety-critical design that manifests itself as an *uncorrelated failure* with a failure rate that is in the same magnitude as the failure rate for failures caused by transient hardware faults then a properly configured TTA architecture would mask such a failure.

5.3 Non-Safety-Critical Software

The non-safety-critical software of a node is encapsulated by the *encapsulation service* such that even a malicious fault in the non-safety critical software will have no effect on the correct function of the safety-critical software.

6. Conclusions

The fault hypothesis states the assumptions about the types and number of faults that a fault-tolerant system must tolerate. The fault-hypothesis must be established at the beginning of the design process, since it has a profound influence on the architecture of the emerging fault-tolerant system. It is thus one of the most important documents for the design process. Without a precise fault hypothesis it is impossible to decide which faults are covered and which faults are *uncovered* by a given design. In order to achieve a high *assumption coverage* the fault hypothesis should make minimal assumptions about the *behavior of faulty nodes.* These minimal assumptions must be carefully documented and scrutinized in order to establish that the *assumption coverage* is in agreement with the overall dependability objective of the intended system.

The fault-hypothesis for the TTA states that a faulty node may fail in any failure mode, irrespective of whether the faulty behavior is caused by a physical fault in the hardware or a design fault in the hardware/software system. However, in the fault hypothesis of the TTA it is assumed that the failures of any two nodes are not correlated.

Acknowledgements

This work has been supported, in part by the European IST project NEXT TTA and the IST network of excellence ARTIST.

References

Ademaj, A., Sivencrona, H., Bauer, G., and Torin, J. (2003). Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proceedings of the 2003 International Conference on Dependable Systems and Networks,* pages 123–132.

Avizienis, A. (1997). Toward systematic design of fault-tolerant systems. *Computer,* 30(4):51–58.

Constantinescu, C. (2002). Impact of deep submicron technology on dependability of VLSI circuits. In *Proceedings of the International Conference on Dependable Systems and Networks,* pages 205–209. IEEE.

Eisenstadt, M. (1997). My hairiest bug war stories. *Commun. ACM,* 40(4):30–37.

Gray, J. (1986). Why do computers stop and what can be done about it? In *Proceedings of the 5th Symposium on Reliablity in Distributed Software and Database Systems.*

Karlsson, J., Folkesson, P., Arlat, J., Crouzet, Y., and Leber, G. (1995). Integration and comparison of three physical fault injection techniques. In Randell, B., Laprie, J., Kopetz, H., and Littlewood, B., editors, *Predictably Dependable Computing Systems,* pages 309–327. Springer Verlag, heidelberg edition.

Kaufmann, L.F. and Johnson, B.W. (2000). Modeling of common-mode failures in digital embedded systems. In *Proc. of the Reliability and Maintainability Symposium,* Los Angeles, Cal.

Kopetz, H. (1992). Sparse time versus dense time in distributed real-time systems. In *Proceedings of 12th International Conference on Distributed Computing Systems,* Japan.

Kopetz, H. (1997). *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, Dordrecht, London.

Kopetz, H. (2003). Fault containment and error detection in the time-triggered architecture. In *Proceedings of the Sixth International Symposium on Autonomous Decentralized Systems.*

Kopetz, H. and Bauer, G. (2003). The time-triggered architecture. *IEEE Special Issue on Modeling and Design of Embedded Software.*

Kopetz, H., Obermaisser, R., Peti, P., and Suri, N. (2004). From a federated to an integrated architecture for dependable embedded systems. Research Report 22/2004, Technische Universitat Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria.

Laprie, J.C. (1992). *Dependability: Basic Concepts and Terminology.* Springer Verlag, Vienna, Austria.

Littlewood, B. and Strigini, L. (1993). Validation of ultrahigh dependability for software-based systems. *Commun. ACM,* 36(11):69–80.

Normand, E. (1996). Single event upset at ground level. *IEEE Transactions on Nuclear Science,* 43(6):2742–2750.

Pauli, B., Meyna, A., and Heitmann, P. (1998). Reliability of electronic components and control units in motor vehicle applications. In *VDI Berichte 1415, Electronic Systems for Vehicles,* pages 1009–1024. Verein Deutscher Ingenieure.

Powell, D. (1992). Failure mode assumptions and assumption coverage. In *Proceedings of the 22nd IEEE Annual International Symposium on Fault-Tolerant Computing (FTCS-22),* pages 386–395, Boston, USA.

Rechtin, E. and Maier, M.W. (2002). *The Art of Systems Architecting.* CRC Press, Boca Raton, USA.

Steiner, W., Paulitsch, M., and Kopetz, H. (2003). Multiple failure correction in the time-triggered architecture. In *Proc. of the IEEE WORDS 2003 Conference,* CAPRI, Italy.

Steiner, W., Rushby, J., Sorea, M., and Pfeifer, H. (2004). Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. *The International Conference on Dependable Systems and Networks (DSN 2004).*

TTTech (1998). homepage of TTTech at `www.tttech.com`.

Walter, C. J., Hugue, M. M., and Suri, Neeraj (1995). *Advances in Ultra-Dependable Distributed Systems.* IEEE Computer Society, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720.