A Model for Reactive Systems Supporting Varying Degrees of Synchrony

Wilfried Elmenreich
Vienna University of Technology
Vienna, Austria
wil@vmars.tuwien.ac.at

Roman Obermaisser
Vienna University of Technology
Vienna, Austria
ro@vmars.tuwien.ac.at

Philipp Peti
Vienna University of Technology
Vienna, Austria
php@vmars.tuwien.ac.at

***Abstract* – This paper presents a model for the integration of subsystems with a varying degree of synchrony into a heterogenous system. While the synchronous part supports hard real-time requirements, the quasi-synchronous part is layered on a synchronous part and inherits properties like bounded drifts, bounded clock value deviation and improved failure detection. The synchronous part will be realized with a time-triggered communication system that supports also an event-triggered channel for the quasi-synchronous part.**

The integration of event-triggered functions with relaxed timing properties makes sense if legacy systems or complex algorithms that cannot be temporally verified have to be used. Dependability is preserved by the failure detector subsystem that monitors the event-triggered part and extends the fault hypothesis to cover also performance failures.

The main benefit of the proposed architecture is the support of resource-efficient and economically feasible hard real-time dependable systems.

I. INTRODUCTION

Due to the many different and, partially, contradicting requirements, there exists no single model for building systems that interact with a physical environment. Well-known tradeoffs are predictability versus flexibility and resource adequate versus best-effort strategies. Thus, the chosen system model depends heavily on the requirements of the application.

For example, in safety-critical real-time control applications like X-by-wire systems in the automotive or avionic domain a system's inability to provide its specified services can result in a catastrophe involving endangerment of lives and/or financial loss an order of magnitude higher than the overall cost of the system. These hard real-time systems must be designed according to the resource adequacy policy by providing sufficient computing resources to handle the specified worst-case load and fault scenarios. In general such system are near to a synchronous approach, thus even during these worst-case scenarios, there are known bounds for computational and communication activities of correct processors.

In non-critical applications, however, dynamic resource allocation strategies and resource sharing may be preferred for economic reasons. Hence, such systems allow timing failures to occur during worst-case scenarios in order to establish cost-effective solutions. These systems can be represented by a model with a reduced level of synchrony.

Many real-time systems require an integration of both approaches due to economic constraints. Typically a system provides on of the following generic services, such as communicating data in a closed control loop, monitoring real-time values, system diagnosis, and system reconfiguration. All these services have different requirements regarding latency, jitter, bandwidth, dependability, consistency, etc. It will not be economically feasible to implement all these services the same way.

For example, consider an autonomous mobile robot that has strong real-time and fault-tolerance constraints such as actuator controls, sensor monitoring and reflex actions. Therefore a hard real-time communication for short messages with a short and constant dead time is necessary. On the other hand the robot system also runs complex path-planning and decision making algorithms, which are difficult to be brought into a synchronous scheme due to their complexity. Moreover, monitoring and maintenance tasks will require relaxed real-time properties but increased message sizes.

Another example for heterogeneous timing requirements is given by systems that contain legacy subsystems. A legacy subsystem has been developed according to their own rules and conventions and may therefore not conform to the overall system's synchrony properties.

It is the objective of this paper to present a model that supports the integration of subsystems with various degrees of synchrony. The expected benefits from such a model are the possibility of constructing efficient dependable hard-real time systems by supporting heterogeneous systems and the integration of legacy applications with respect to real-time behavior and fault containment of the overall application. Such a hybrid approach enhances the behavior of (legacy) subsystems with reduced synchrony assumptions by the support of improved failure detectors.

The remaining sections of this paper are organized as follows: Section II. reviews the basic concepts on distributed systems that are essential for the presented approach. Section III. presents a heterogenous model consisting of a quasi-synchronous event-triggered subsystem that is put on top of a synchronous time-triggered system layer. Section IV. shows the application of the presented ideas in a time-triggered sensor fusion model that can be used for an autonomous mobile robot. The paper is concluded in section V.

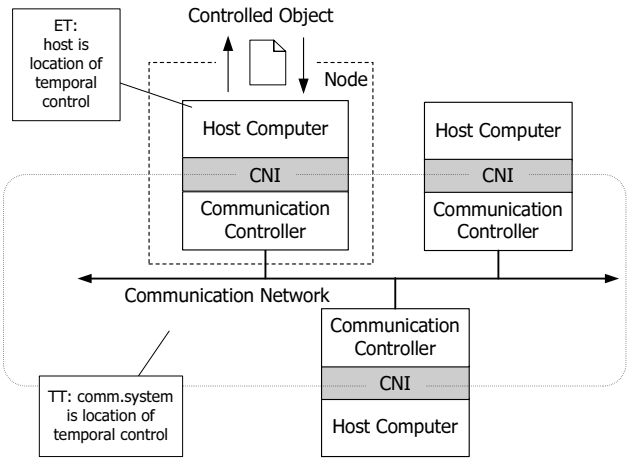

Fig. 1: Distributed Computing System

II. CONCEPTS

A. Distributed Systems

A distributed system (as depicted in Figure 1) consists of a set of nodes interconnected via a common network (communication system). A node can at least be partitioned into two subsystems, the communication controller (CC) and the host computer. The interface between these two is called Communication Network Interface (CNI) [1].
In the context of distributed embedded real-time systems a complete node seems to be the best choice for a component [2], since the component-behavior can then be specified in the domains of value and time. Thus, a component is considered to be a self-contained computational element with its own hardware (processor, memory, communication interface, and interface to the controlled object) and software (application programs, operating system), which interacts with its environment by exchanging messages across linking interfaces (LIFs) [3].

B. Asynchronous vs. Synchronous Systems

In synchronous systems there are known upper bounds of the durations of communication and processing activities. In the absence of such bounds, we speak of an asynchronous system [4]. According to [5] the degree of synchrony of a system is defined by the system's compliance to five conditions:

- S1 bounded and known processing speed
- S2 bounded and known message delivery delay
- S3 bounded and known local clock drift rate
- S4 bounded and known load pattern
- S5 bounded and known difference of local clocks

A synchronous system must satisfy all five conditions. An asynchronous system does not satisfy any of these conditions. Asynchronous and synchronous system are thus two extreme levels of synchrony. Many existing systems are based on a model with an intermediate level of synchrony. An example for such an intermediate level of synchrony is the *quasi-synchronous* model of computation [6]. In a quasi-synchronous system a subset of the five synchrony conditions is satisfied probabilistically, i.e. there is a known probability that an assumed bound does not hold.

C. The Concept of State

In system theory, the notion of state is fundamental for investigation of complex systems. In abstract system theory, the notion of state is introduced in order to separate (decouple) the past from the future. Assuming a deterministic world model, *the idea is that if one knows what state the system is in, he could with assurance ascertain what the output will be* [7, p. 45]. Hence, *the state embodies all past history of the given system*. Apparently, this definition of state by Mesarovic and Takahara is only meaningful, if the notion of past and future (time) is relevant for the considered system.

D. Failure Detectors

Inspired by the impossibility result of Fischer et al. [4] research has been focused on the question how much synchrony is needed to achieve consensus in a distributed system [8].
Chandra and Toueg [9] proposed an alternative approach to circumvent the impossibility result. In order to decide whether a process has actually crashed or is only very "slow", they augmented the asynchronous model of computation with *unreliable failure detectors*, i.e. an external failure detection mechanism that can erroneously indicate that a component has failed, only to correct the error at a later time. A distributed failure detector comprises a set of local failure detector modules. Each module maintains a list of processes which are currently suspected to have crashed and can add or remove processes from this list. Furthermore, the lists of suspects of two local failure detector modules can be different at any given time.
Failure detectors can be characterized according to the *completeness* and *accuracy* properties. Roughly speaking, completeness requires that every crashed process is eventually suspected and accuracy restricts the mistakes a failure detector can make.
In [10] it has been shown that in order to solve *consensus*, any failure detector has to provide at least as much information as the *eventually weak failure detector* $\diamond\mathcal{W}$. The definition of a class of failure detectors must be seen as a specification of the failure detection mechanism [11]. An optimal algorithm that implements the weakest failure detector is described in [12].
Chen et al. introduce a Quality of Service (QoS) model for failure detectors [13]. These QoS metrics should describe the properties of the failure detector in a quantitative way.
Arora [14] describes a failure detector as a system component that determines whether some state predicate is satisfied by the system state. According to [15], is in general a

difficult task to decide whether a predicate over the global system's state does or does not hold without a common time frame. The sparse time-base as introduced by Kopetz [16] offers the possibility to draw a irrefutable borderline between the past and the future, and thus the definition of a system-wide distributed state.

E. Event-Triggered and Time-Triggered Communication Systems

Two paradigms can be used for building the communication system. In an event-triggered (ET) system communication activities are initiated whenever a significant change of state occurs. Such a system exploits external control, i.e. the decision when a message is to be transmitted is within the sphere of control of the application software in the host. ET systems allow a flexible allocation of resources, which is attractive for variable resource demands. However, multiple nodes may contend for bus access as a reaction to event occurrences. In safety-critical applications it is necessary to guarantee a predictable communication with low latency and low jitter to all participants [17].
In a time-triggered (TT) system, activities are initiated at predetermined points in time. Such a system employs autonomous control. The communication controller decides autonomously when a message is transmitted. The CNI forms a *temporal firewall*, which isolates the temporal behavior of the host and the rest of the system. Local changes of a host cannot invalidate the correct temporal behavior of the communication system. Therefore a TT communication system provides composability with respect to the temporal control [18]. A TT system also helps achieving replica determinism [19], which is essential for establishing fault tolerance through active redundancy. The predetermined points in time of the periodic message transmissions allow error detection and establishing of membership information. Since the system load is independent of the number of events occurring in the controlled object, the latency jitter is minimal. The message transmission latency during peak load scenarios is identical to the latency during normal load.

III. INTEGRATING SYNCHRONOUS AND QUASI-SYNCHRONOUS SYSTEMS

The ET on top of TT service model depicted in Figure 2 aims at heterogeneous systems that are comprised of a strictly synchronous part and a part with relaxed synchrony assumptions. For the ET subsystem, we choose the layering of the ET communication channel on top of the synchronous time-triggered communication channel. In addition to the advantages of this approach described in [20], we enable failure detection at the ET communication channel through the underlying synchronous time-triggered communication channel. This model aims at critical systems that include non-critical subsystems. In case the overall system has evolved into a system with a higher level of criticality, these subsystems can be legacy systems providing non-critical functions. Consider for example electronics in the automotive industry, where computer systems have originally been used primarily for non-safety critical functions (e.g., body electronics). In the near future car models include controlling computer systems for safety-related functions such as brake-by-wire and steer-by-wire functionality [21]. For economic reasons new communication architectures (e.g., for X-by-wire applications) will not replace well established ones instantly.
In addition to the ability for reusing existing legacy applications, the heterogenous system model offers support for newly developed applications with relaxed synchrony guarantees. This is motivated by economic constraints, which can prevent the resource adequacy policy, which establishes the foundation for guaranteeing hard bounds for communication and computational activities as required for justifying the synchronous system model.
An example for a non-critical service in the automotive domain is the collection of statistical field data serving as engineering feedback, which goes beyond conventional maintenance logs [22].

A. Synchronous TT Part

The strict synchronous part is formed by a time-triggered subsystem, which in addition to satisfying the five defining conditions of synchrony also offers a synchronized global time base and derives all control signals from the progression of this global time [23]. A time-triggered communication protocol establishes a synchronous communication channel and links the components of the TT subsystem. The high level of temporal predictability simplifies the provision of fault-tolerance mechanisms, since components can exploit the global notion of time to reason about each other's progress and state. This fact renders the synchronous subsystem well-suited for critical system services.

B. Quasi-Synchronous ET Part

The event-triggered subsystem is a delimited, well-defined part of the overall system, which complies to the quasi-synchronous model of computation [6]. In other words, defining synchrony conditions are satisfied in only a probabilistic manner. The ET subsystem offers support for dynamic resource sharing and multiplexing of resources. It is aimed at non-critical applications, for which probabilistic guarantees for the durations of computational and communication activities are accepted for economic reasons. An event-triggered communication service establishes a quasi-synchronous communication channel, which provides a non-zero probability of a violation of timing bounds for communication latencies. Nevertheless, the event-triggered communication channel can offer a flexible and resource efficient communication service that allows applications to initiate communication activities in order to react to significant events.

C. Fault Hypothesis for the System Model

For the ET service model, we extend the fault hypothesis described in [24], which assumes a single crash/omission

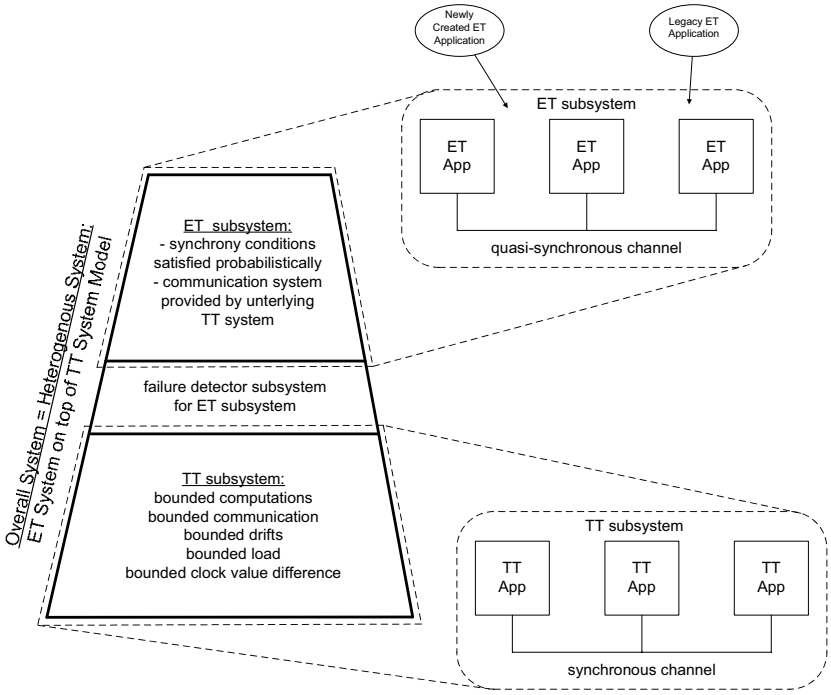

Fig. 2: Model for Integration of Synchronous and Quasi-Synchronous Systems

(CO) failure in any single of the constituent parts, namely:

- node-computers, which comprise a host computer and an associated communication controller
- and the communication channel.

A single failure implies that a minimum interval of time δ lies between successive CO failures. This interval of time allows the system to regain a consistent state after a CO failure, prior to the occurrence of the next failure. We extend the CO model by also considering single *performance failures*, thereby reaching a crash/omission/performance (COP) failure model.

A performance failure occurs, if a component fails to respond to input due to an imbalance of message production and consumption rates. A performance failure results from the loss of a message through the overflow of an event message queue. The queue at the sender's side overflows, in case the sender is requesting message transmissions at a rate exceeding the communication system's bandwidth, thereby accumulating more messages than fit into the outgoing queue. The event message queue at the receiver stores messages of time intervals in which the sender has produced messages at a higher rate than the receiver's rate of message consumptions. If more messages have accumulated than fit into the receiver's message queue, a performance failure occurs.

D. Error Containment Mechanisms

In order for the synchrony assumptions of the time-triggered subsystem to hold, rigorous error containment through the enclosing system for errors of the ET subsystem is essential. A violation of probabilistic bounds for computational or communication activities within the quasi-synchronous subsystem must not affect the correct temporal behavior of the synchronous part.

E. Failure Detection

Crash failures and omission failures are detected by the underlying synchronous system. Failure detectors for performance failures are employed for locally detecting imbalances of message production and consumption rates. By exchanging the local views with respect to errors via the synchronous channel, it is possible to achieve consensus regarding the set of correct components. This yields membership service as described in [25].

IV. APPLICATION OF THE MODEL

This section depicts an applications of the presented ideas in an architecture for an autonomous mobile robot. The basic model used a synchronous architecture and has been described as "time-triggered sensor fusion model" in [26].

The model incorporates properties like cyclic processing, composable design, and introduces well-defined interfaces between its subsystems. Figure 3 depicts a control loop modelled by the time-triggered sensor fusion model. Interfaces are illustrated by a disc with arrows indicating the possible data flow directions across the interface. Physical sensors and actuators are located on the borderline to the process environment and are represented by circles. All other components of the system are outlined as boxes. The

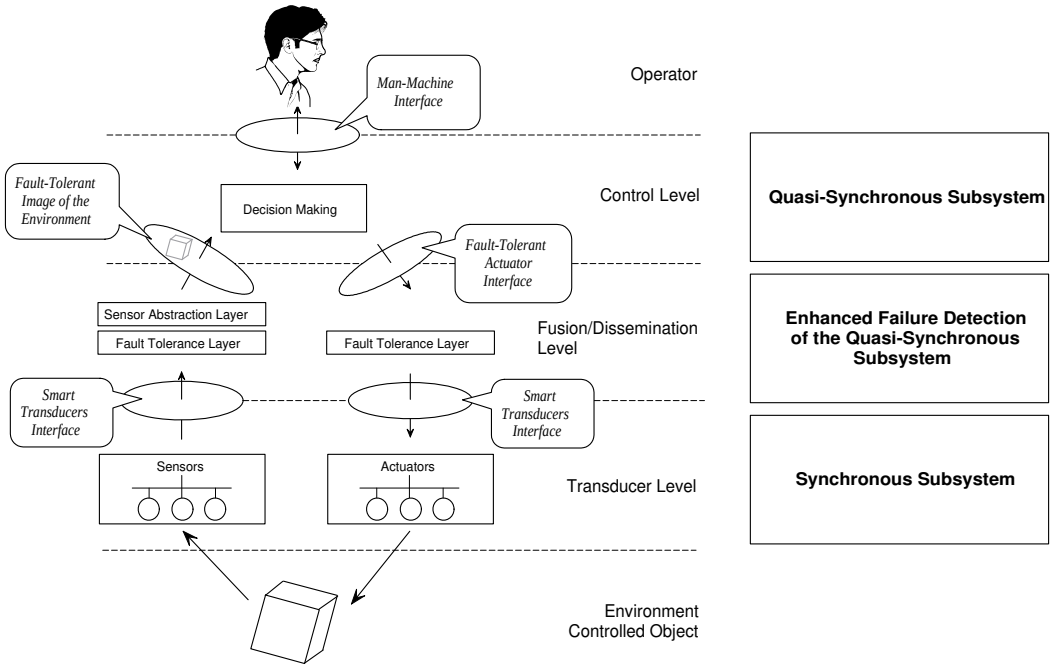

Fig. 3: Data flow in the time-triggered sensor fusion model

model distinguishes three levels of data processing with well-defined interfaces between them. The *transducer level* contains the sensors and actuators that interact directly with the controlled object. A *smart transducer interface* provides a consistent borderline to the above *fusion/dissemination level*. This level contains fault tolerance and sensor fusion tasks. The *control level* is the highest level of data processing within the control loop. The control level is fed by a dedicated view of the environment (established by transducer and fusion/dissemination level) and outputs control decisions to a *fault-tolerant actuator interface*. User commands from an operator interact with the control application via the *man-machine interface*.

The components of the transducer level consist of smart transducers with typically small predictable programs. These components are directly integrated into a synchronous time-triggered communication system providing hard real-time capabilities.

In contrast, the decision making systems at control level will typically consist of complex high-level algorithms (for example path planning for a mobile robot). When these part would be brought into a fully synchronous scheme, the necessary resources on computation time, which depend on the estimated worst-case execution time of the processes would lead to a unrealistic requirements in necessary computational time in order to make the system resource adequate. A more resource-efficient system can be build, when a quasi-synchronous model is applied. This model allows unbounded load and unbounded computations.

The disadvantage of an asynchronous model of computation lies in its inability to detect performance failures of processors. Our model overcomes this deficiency by providing enhanced failure detectors by taking advantage of the properties inherited from the synchronous layer. The failure detectors reside in the fusion/dissemination level together with a fault-tolerance and a sensor fusion layer.

V. CONCLUSION

We have motivated and discussed the integration of subsystems with a varying degree of synchrony into a heterogenous system. While the synchronous part supports hard real-time requirements, the quasi-synchronous part is layered on a synchronous part and inherits properties like bounded drifts, bounded clock value deviation and improved failure detection. The synchronous part will be realized by a time-triggered communication system that supports also an event-triggered channel as quasi-synchronous part.

The integration of event-triggered functions with relaxed timing properties makes sense if legacy systems or complex algorithms that cannot be temporally verified have to be used. Dependability is preserved by the failure detector subsystem that monitors the event-triggered part and extends the fault hypothesis to cover also performance failures.

By the example of a synchronous architecture for autonomous mobile robots we have proposed a separation of functions in the model. Control functions with strong real-time and fault-tolerance constraints are mapped to the time-triggered part while complex navigation and decision algorithms reside in the event-triggered part.

The main benefit of the proposed architecture is the support of resource-efficient and economically feasible hard real-time dependable systems.

VI. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for valuable comments on the ideas of this paper. This work was supported in part by the European IST project NEXT TTA under contract No IST-2001-32111 and by the Hochschuljubiläumsstiftung der Stadt Wien via project CoMa (H-965/2002).

VII. REFERENCES

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

[2] H. Kopetz. Component-based design of large distributed real-time systems. *Control Engineering Practice - A Journal of IFAC*, 6:53–60, 1998.

[3] H. Kopetz and N. Suri. Compositional design of RT systems: A conceptual basis for specification of linking interfaces. Research report, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2002.

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

[5] P. Veríssimo. On the role of time in distributed systems. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 316–321. IEEE, October 1997.

[6] C. Almeida, J. Rufino, and P. Veríssimo. DDRAFT: Supporting dynamic distributed real-time applications with fault-tolerance. Technical Report CSTC RT-98-02, Centro de Sistemas Telemáticos e Computacionais do Instituto Superior Técnico, Lisboa, Portugal, February 1998.

[7] M. D. Mesarovic and Y. Takahara. *Abstract Systems Theory*, chapter 3. Springer-Verlag, 1989.

[8] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, 34(1):77–97, 1987.

[9] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[10] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM (JACM)*, 43(4):685–722, 1996.

[11] R. Guerraoui and A. Schiper. Consensus: the big misunderstanding [distributed fault tolerant systems]. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 183–188. IEEE, October 1997.

[12] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving consensus (brief announcement). In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, page 334. ACM Press, 2000.

[13] W. Chen, S. Toueg, and M.K. Aguilera. On the quality of service of failure detectors. *IEEE Transactions on Computers*, 51(1):13–32, January 2002.

[14] A. Arora and S. S. Kulkarni. Detectors and correctors: A theory of fault-tolerance components. In *18th International Conference on Distributed Computing Systems (18th ICDCS'98)*, Amsterdam, The Netherlands, May 1998. IEEE.

[15] F. C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys (CSUR)*, 31(1):1–26, 1999.

[16] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, Japan, Jun. 1991.

[17] J. Rushby. Bus architectures for safety-critical embedded systems. In Tom Henzinger and Christoph Kirsch, editors, *EMSOFT 2001: Proceedings of the First Workshop on Embedded Software*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, October 2001. Springer-Verlag.

[18] H. Kopetz and R. Obermaisser. Temporal composability. *IEE's Computing & Control Engineering Journal*, January 2002.

[19] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1996.

[20] R. Obermaisser. CAN emulation in a time-triggered environment. In *Proceedings of the 2002 IEEE International Symposium on Industrial Electronics (ISIE)*, volume 1. IEEE, January 2002.

[21] E. Bretz. By-wire cars turn the corner. *IEEE Spectrum*, 38(4):68–73, April 2001.

[22] A. Deicke. The electrical/electronic diagnostic concept of the new 7 series. In *Convergence International Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, October 2002. SAE.

[23] J. Rushby. Bus architectures for safety-critical embedded systems. In T. Henzinger and C. Kirsch, editors, *EMSOFT 2001: Proceedings of the First Workshop on Embedded Systems*, volume 2211 of *Lecture Notes in Computer Science*, pages 306–323, Lake Tahoe, CA, October 2001. Springer-Verlag.

[24] G. Bauer. *Transparent Fault Tolerance in a Time-Triggered Architecture*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2001.

[25] R. Obermaisser. Membership service for the ET on top of TT service model. Research Report 21/2003, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 1-3/182-1, 1040 Vienna, Austria, 2003.

[26] W. Elmenreich and S. Pitzek. The time-triggered sensor fusion model. In *Proceedings of the 5th IEEE International Conference on Intelligent Engineering Systems*, pages 297–300, Helsinki–Stockholm–Helsinki, Finland, September 2001.