A Development Framework for Ultra-Dependable Automotive Systems Based on a Time-Triggered Architecture

Bernd Hedenetz
Daimler-Benz Research, HPC T721, D-70546 Stuttgart, Germany
hedenetz@dbag.stg.daimlerbenz.com

Abstract

Today by-wire systems are well-known and utilised in the area of aircraft construction. In the last few years there has been an endeavour in the automotive industry to realise by-wire applications without mechanical or hydraulic backup systems in vehicles. The required electronic systems must be highly reliable and cost-effective due to the constraints of mass production.

A time-triggered architecture is a new approach that satisfies these requirements. The backbone of communication in this architecture is the fault-tolerant Time-Triggered Protocol (TTP), developed by the Vienna University of Technology and the Daimler-Benz Research. The TTP protocol has been designed due to the class C SAE [25] classification for safety critical control applications, like brake-by-wire or steer-by-wire.

For time-triggered architectures a new development process is required to handle the complexity of the systems, accelerate the development and increase the reliability. In this paper we present an approach for the development of distributed fault-tolerant systems based on TTP. The present approach is evaluated by a brake-by-wire case study.

1 Introduction

In the past few years there has been the tendency to increase the safety of vehicles by introducing intelligent assistance systems (e.g., ABS, Brake-Assistant (BA), Electronic Stability Program (ESP), etc.) that help the driver to cope with critical driving situations. These functions are characterised by the active control of the driving dynamics by distributed assistance systems, which therefore need a reliable communication network. The faults in the electronic components, which control these functions, are safety critical. However, the assistance functions deliver only an add-on service in accordance with a fail-safe strategy for the electronic components. If there is any doubt about the correct behavior of the assistance system, it will be switched off. For by-wire systems without a mechanical backup a new dimension of safety requirements for automotive electronics is reached. After a fault the system has to be fail-operational until a safe state is reached.

For the fail-operational assumption we demand that after any arbitrary fault the system is fully operational. The effective use of the redundancy is important, in order to reduce the production costs for automotive by-wire systems. A major goal is to increase the reliability of the system by adding additional redundancy without increasing the complexity of the system. Therefore, new electronic architectures have to be developed.

Distributed time-triggered architectures (TTA) can be realised through the Time-Triggered Protocol (TTP) which guarantees a global time synchronisation over the whole system and an adequate message transmission.

In this paper we present an approach for the development of distributed fault-tolerant systems based on TTP. This paper is organised as follows: Section 2 gives an overview of the general architecture and elaborates on the time-triggered approach and the communication subsystem. Section 3 gives a short overview about the lifecycle of safety related automotive systems. In Section 4 the development framework for TTA systems is presented. In Section 5 the development approach is demonstrated at the example of a brake-by-wire case study. The paper is concluded in Section 6.

2 Time-Triggered Approach

The TT paradigm of a real-time system is based on a distinctive view of the world: the observer (the computer system) is not driven by the events that happen in its environment. The system decides through the progression of time when to look at the world. Therefore, it is impossible to overload a time-triggered observer.

A TT system takes a snapshot of the world, an observation, at recurring predetermined points in time determined by the current value of a synchronised local clock. This snapshot is disseminated within the computer system by the communication protocol to update the state variables that hold the observed values. The semantic of the periodic messages transported in a TT system is a state-message semantic, i.e., a new version of a message overwrites the previous version and messages are not consumed on reading. This semantic is well suited to handle the transport of the values of the state variables used in control applications. The state message semantic provides a predefined constant load on the communication system and eliminates the problem of dynamic buffer management [23].

2.1 Time-Triggered Protocol (TTP)

For the realisation of a distributed time-triggered architecture a communication network is necessary that provides the features mentioned above. This type of communication belongs mainly to class C of the SAE classification [25].

None of the commonly used in-vehicle communication systems (CAN, A-BUS, VAN, J1850-DLC, J1850-HBCC [26]) meet the requirements for safety related by-wire systems since they were not designed for this case [17]. They are all lacking in being deterministic, in synchronisation and fault tolerance characteristics.

These missing properties are the motivation for developing new approaches for in-vehicle communication systems. As a new start we examine the Time-Triggered Protocol developed by the University of Vienna and Daimler-Benz Research. TTP is especially designed for safety related applications and fulfills these requirements.

TTP is an integrated time-triggered protocol that provides:

- a membership service, i.e., every single node knows about the actual state of any other node of the distributed system
- a fault-tolerant clock synchronisation service (global time-base),
- mode change support,
- error detection with short latency,
- distributed redundancy management.

All these issues are supported implicitly by the protocol itself. A comprehensive description of the TTP protocol is given in [13,14,15]. The TTP protocol has been designed to tolerate any single physical fault in any one of its constituent parts (node, bus) without an impact on the operation of a properly configured cluster [15].

The overall TTP hardware architecture is characterized by both the TTP system architecture and the TTP node architecture as shown in Figure 1. A TTP real-time system consists of a host subsystem, which executes the real-time application and the communication subsystem providing reliable real-time message transmission. The interface between these subsystems is realised by a dual ported RAM (DPRAM) called *Communication Network Interface (CNI)* [16]. The assembly of host and TTP-controller is called *Fail Silent Unit (FSU)*. Two FSUs form a single redundant *Fault-Tolerant Unit (FTU)*. The physical layer consists of two independent transmission channels.

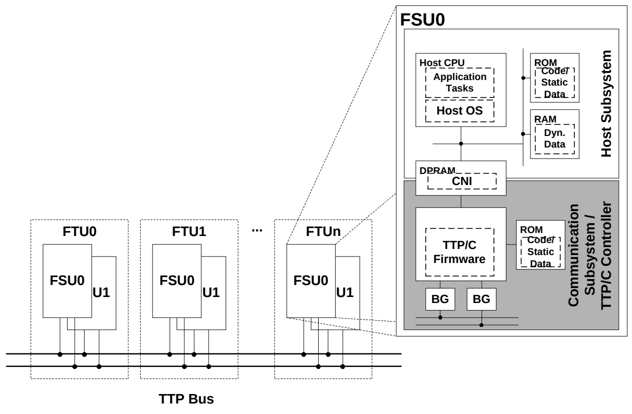

Figure 1: Architecture of a TTP-based fault-tolerant real-time system

2.2 Node Architecture

The overall aim of the node architecture is to fulfill the fail-silence assumption without developing special hardware for fault detection. We use software fault detection methods and low cost hardware mechanisms (such as watchdogs) and mechanisms provided by the CPU (bus error, address error, illegal op-code, privilege violation, division by zero,...). In [12] it is shown that a high degree of fault detection can be achieved by software fault detection mechanisms. We follow this approach for trying to fulfill the fail-silence assumption. Our architecture can be separated into three subsystems (see Figure 2):

- *Communication subsystem*: this part is responsible for the communication between distributed components.
- *Fault-tolerant subsystem*: this part contains safety critical and fault tolerance mechanisms. The safety related application is handled by this subsystem.
- *Application subsystem*: this part includes the safety related tasks, which build the application.

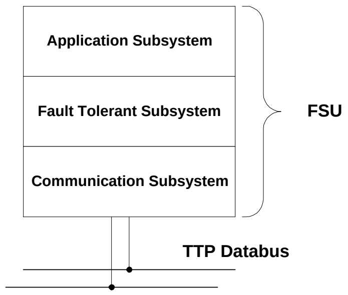

Figure 2: Subsystems of an FSU

2.3 Propagation of TTP

TTP is in discussion in the Brite-EuRam Project „Safety Related Fault Tolerant Systems in Vehicles" (acronym: „X-By-Wire") to be proposed as a European or International Standard [2].

In 1997 the Esprit Project „Time Triggered Architecture" (acronym TTA) has been started with the intention to develop a prototype TTP controller chip [9]. Other aims of the TTA project are the development of tools for the design of TTP systems and the formal verification of parts of the TTP protocol e.g. the clock synchronisation algorithm.

3 Lifecycle

The typical lifecycle for the development of safety related automotive systems consist of the following phases: *system specification*, *system design*, *design verification*, *implementation* and *integration* (see Figure 3). Several, more detailed descriptions of the lifecycle exist, for further study a large number of books [27] and standards are available [10,11]. All steps of the lifecycle have to be supported with tools to manage the complexity of the systems, accelerate the development process and increase the reliability. New tools have to be developed and common used tools have to be adapted to the requirements of TTA systems.

4 Development Process

For TTA systems a new development process is required which supports every phase of the lifecycle. Our aim is to devise a approach for designing complex distributed safety-related automotive systems using *commercial-off-the-shelf* tools to as high a degree as possible. Our approach based on verification of the system design by functional simulation, fault modeling in the models, functional test and fault injection in the real system architecture. Therefore we separate the development process in seven single steps (see Figure 3). During the first step - *requirement specification* - we specify the functional requirements, the time constrains and the reliability requirements of the system. In the second step - *architectural design* - the structure of the communication network, communication relations between the nodes and the schedule of the application tasks are defined due to the specification. In the next step - *functional design* - we realise the application, e.g., a control loop for an anti blocking system (ABS), as a functional model. For the actual realization of this part we use the tool Statemate^{TM} from i-Logix [7] and MATLAB$^{®}$/Simulink^{TM} from MathWorks [19]. In the following step - *functional simulation* - the functional models are verified through simulation and the reliability is examined by *fault modeling* into the functional models. In the step - *realisation and integration* - the real architecture is realised and integrated in a vehicle. Therefore we use the capability of automatic code generation of the tools. Additionally, we use monitoring tools to trace the system behavior. In the final step - *test and fault injection* - we execute test and fault injection experiments in the target system to verify the behavior of the real system with respect to the requirement specification.

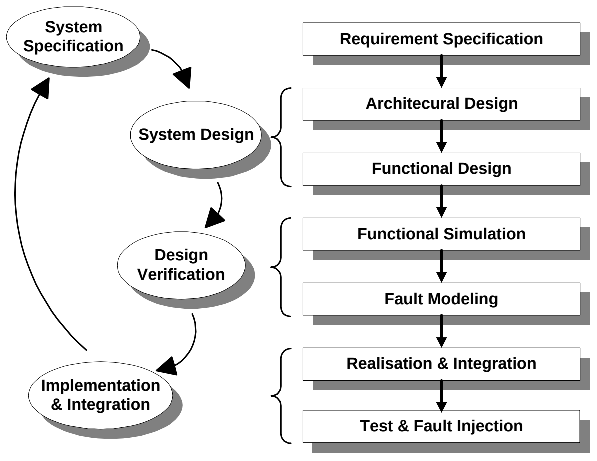

Figure 3: Overview about the development process

4.1 Requirement Specification

System specification is a very sensitive phase. The most faults that lead to critical failures are system-faults, and most design errors are not low-level implementation errors but errors committed at the system specification phase [18]. The description of the requirement specification has to include:

- Identification of the objects in the environment
- which produce or consume data flows.

- Definition of the input and output signals for these objects.
- Specification of the functional and time behavior of these objects.
- Specification of the reliability requirements.
- Definition of a fault hypothesis.

To verify the functional models and the system realisation we have to specify the system with a sufficient accuracy. The requirement specification is used to generate test pattern, fault pattern and environment models. Environment models are used as a reference for the functional simulation. Environment models can be described by:

- Statemate^{TM} models,
- Input, output signals (stimulation and reaction),
- Differential equations, and
- C-code.

4.2 Architectural Design

The complexity of a system depends on the number and types of elements and relations and on the amount of their inner states. A method to handle complex systems is the decomposition into smaller subsystems. A common way is to partition the entire system into subsystems with high inner connectivity and few relations crossing the subsystem boundaries [1]. For the design of time triggered architectures we follow this approach; first we design the highest subsystem level, the communication nodes in a TTP communication system. This step is called *global design* and consists of all steps associated with the overall system architecture:

- A time-triggered system is partitioned into a set of components connected by the TTP bus.
- In respect with the reliability requirements, critical components can be replicated.
- The communication relations between the components are defined.
- In the last step, a bus schedule is determined that fulfills the communication requirements of the previous steps.

These steps are typically done by the system manufacturer. The structure is commonly determined by the function which the system has to fulfill. For example, in an automotive steer-by-wire system, redundancy is required for measurement of the steering wheel angle and control of the steering actuator. The redundant nodes have to be distributed that there is no common mode failure - e.g. intruding of water in the ECU - can cause a fatal failure.

4.3 Functional Design

After the global design, the *local design* contains all steps associated with the specification of a single component. The bus message scheduling specified in the global design defines the interface of each component in the value and time domain. The local design consists of the following steps:

- Definition of the software structure of the components.
- Description of the application tasks and their communication relations and time constraints.
- Adding of fault tolerant schemes, e.g. double execution and variable protection through cycle-redundancy checks (CRC).
- Finally, a task schedule which fulfills the functional requirements and time constraints is determined.

The single components in the automotive environment are typically developed by sub-suppliers.

4.4 Functional Simulation

Models of real-time systems have to support different views of a system. A system can be described by four different views:

- Structural view: represents the structure of the subsystems.
- Functional view: describes the functions and processes of the system, the input and output signals, and the information flows between the functions and processes.
- Behavior view: describes the time and dynamic behavior and the internal states of the components.
- Implementation view: represents the realisation of the system by source code.

For the actual realisation of the functional models we use the tools Statemate^{TM} and MATLAB$^{®}$/Simulink^{TM}. Statemate^{TM} provides a hierarchical modeling approach for the specification and analysis of complex systems. The special feature of Statemate^{TM} is that it puts emphasis on the dynamic verification of the specification. This tool provides facilities for the model execution, in interactive or batch mode, and to instrument the models in order to collect statistics during execution. The model can be either connected to a software environment model (software-in-the-loop) or to a target hardware environment (hardware-in-the-loop). The source code can be generated automatically from the functional specification.

The main benefit of this approach is that the system behavior can be examined from a very early design phase and changes can be made with minor effort.

Statemate^{TM} uses three methods for system modeling, *module charts*, *activity charts*, and *statecharts* [5,6]. The module charts describe the structural view of the system while activity charts describe the functional view and are similar to conventional data flow diagrams. Activity charts illustrate the identified sub-functions and the information flows between them. Statecharts describe the behavioral view of the system. Through automatic code generation Statemate^{TM} also supports the implementation of the system.

4.4.1 Modeling a TTA System

The Statemate^{TM} approach is typically used for reactive systems. Time constraints are only an implicit part of the models. An important property in a TTA is the fulfillment of the time constrains. If the time boundaries are violated the system can not fulfill its duty. We solve this problem by building up a complete model of the TTA system in the Statemate^{TM} developing environment. The time constraints of the system architecture are guaranteed by the model of the TTP communication system. The time constraints on the node subsystem are guaranteed by the model of the fault tolerant subsystem.

The *system architecture* is the highest modeling level. This level represents the structure of the system. Every node is also modeled in detail. *Node* represents the node structure and consists of the *fault tolerant subsystem*, the *communication subsystem* and the *application subsystem* (corresponding to Figure 2). The fault tolerant subsystem builds the environment for the safety critical tasks, defined in the application subsystem. The communication subsystem handles the communication services provided by the TTP system (see Figure 4).

For the design of a new system the developer has to define the application dependable parts, the architectural structure of the TTP cluster and the application subsystem, corresponding to the global and local design in Section 4.2 and 4.3. The other parts of the model are available in a model library and can be reused.

4.4.2 Application Functional Specification

Applications on the top of a TTA consist usually of the parts; periodically reading sensor signals, calculation of new system states, and activation of actuators. The function of applications are usually described through statemachines and the control algorithms. The Statemate^{TM} environment provide for the specification of statemachines the modeling concept of activity charts and statecharts. The control algorithms are developed with the help of MATLAB$^{®}$/Simulink^{TM} or MATRIX$_X$^{TM} [28]. The control algorithms can be integrated in Statemate^{TM} through C source code or in the case of MATRIX$_X$^{TM} directly, due to a common interface.

4.4.3 Application Development Environment

For the developing of applications we use a *software-in-the-loop* simulation environment (see Figure 5). We

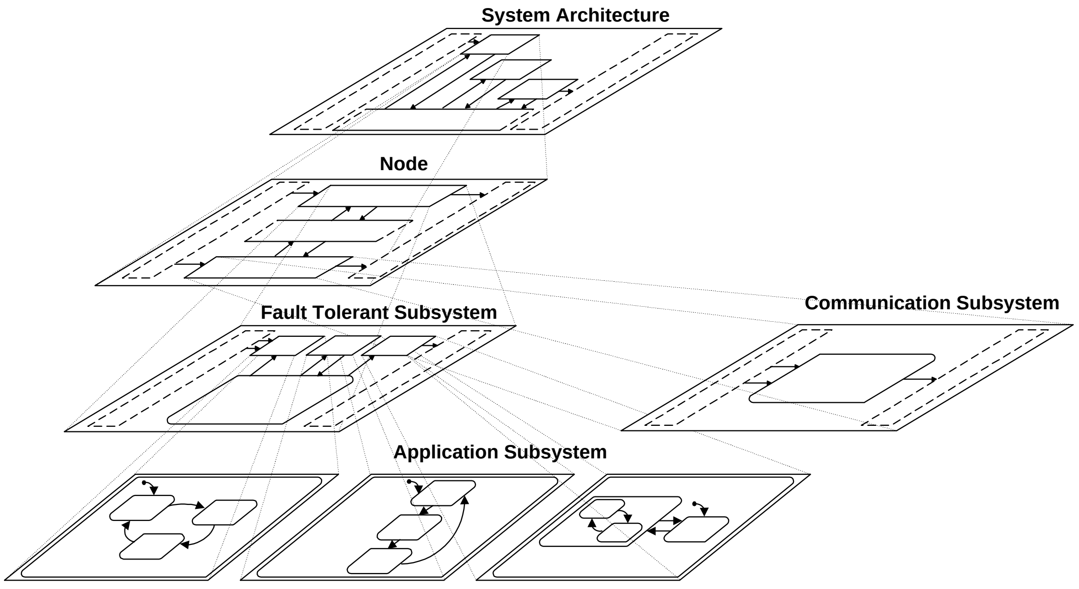

Figure 4: Functional model of a TTA

define a test environment out of the requirement specification, in which the functional model can be tested. We connect the functional model of the TTA system with the behavioral model of the vehicle. This method is called software-in-the-loop. To verify the functional model a set of test patterns have to be defined. The number of test patterns depend on the complexity of the system, the number of signals and the used test strategy. The environment model is realised by using MATLAB®/Simulink™ .

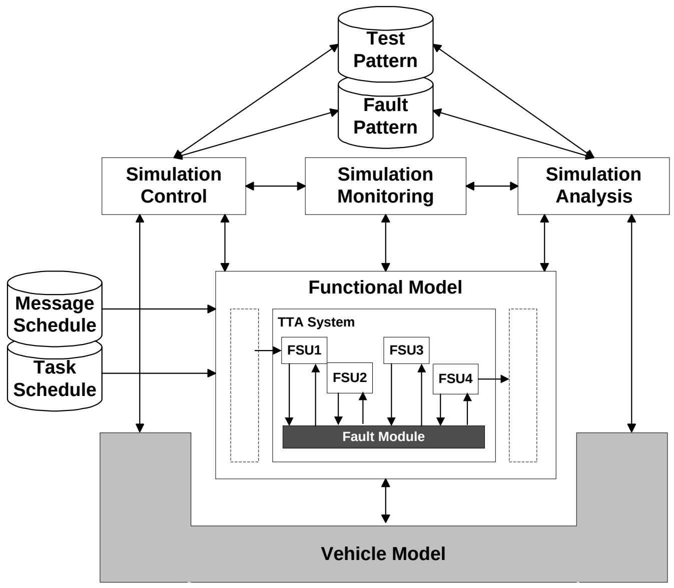

Figure 5: Overview of the simulation environment

The simulation environment consists of five components:

- The *vehicle model* is used as a test environment for the functional modeling. The vehicle model delivers the reactions of the environment to the functional model.
- The *simulation control* stimulates and controls the simulation. Additionally, the test pattern and the vehicle parameters are handled.
- The *simulation monitor* shows the current state of the simulation.
- The *simulation analysis* stores and analyses the reaction of the functional model to the different test and fault pattern.
- The *fault module* allows the fault injection into the functional model. Section 4.5 describes the fault module closer.

If any fault is detected during the simulation, the fault has to be localized and corrected. In addition to the verification of the functional models, faults and open issues in the requirement specification can be detected.

4.5 Fault Modeling

To examine the reliability of the functional specification we introduced a fault module in our simulation environment, which allows us formal analysis of the system behavior in the presence of faults. In our fault hypothesis we claimed to tolerate any single arbitrary fault. To validate this property we use the fault module. The fault module is an optional part of the functional model and is controlled by the simulation control. The behavior of the fault module corresponds to the behavior of the fault injection device (see Section 4.7). So the result between fault modeling and fault injection can be compared and the fault modeling can be verified. Our focuses of fault injection activities on the real system focus on the disturbance of the communication on the TTP transmission channel. Thus the fault injection module is realised for the transmission channels only.

4.6 Realisation and Integration

After finishing the global and local design and verification of the behavior through functional simulation and fault modeling, the complete system is integrated by connecting all components, by downloading and execution of the application software.

For the system integration the opportunity to monitor the global behavior by tracing the bus is important. Therefore we developed, together with an outhouse partner a monitoring tool, which allows us during run-time to monitor and trace the messages on the communication bus, without influencing the system. The inner state of the components like task states, variable values, etc. can be observed via local monitoring. Local monitoring should be applied very carefully, because it changes the behavior, especially the timing behavior of the components.

In industrial projects parts of the system are developed by different project teams or outhouse partners. For the component developer the whole system is typically not available. For the test of a single component a tool which simulates missing nodes is required. Therefore we realised a first prototype tool [3].

4.7 Test and Fault Injection

We have three intentions with the test and the fault injection experiments:

1. Verify the behavior of the TTP bus against the preliminary TTP specification, the TTP protocol is still under current investigation.
2. Verify the behavior of the simulation models for the nodes, the communication subsystem and the fault

module during the developing phase of the simulation environment.
3. Verify the behavior of the real system with respect to the system requirement specification.

4.7.1 Test

The definition of the test pattern is very important for the quality of the test. The aim is to reach as high as possible coverage of the test room with a minimum number of test patterns. The test method has to be understandable and reproducible, therefore, we used a functional test method.

Functional test methods examine the functional behavior and not the code itself. The program is considered as a *black box* [22]. The aim of the functional test is to test the specified requirements as completely as possible. The test specification can be described in a formal and informal way [4]. Formal methods nowadays are only accepted for a few applications in the industry. The most important functional test methods are *equivalent class, boundary test* [20], the *category-partition method* [21], and the *classification tree method* [4]. We generate our test pattern with the help of the classification tree method. It supports the combination of different input signals to generate test patterns. In the first step the relevant classifications are identified with the help of the requirement specification. In the next step the classifications are separated into mathematical disjunctive classes. The classifications and classes can be defined hierarchically and form a classification tree. The test patterns are generated by the combination of the non classifiable basic classes, from each single classification one class is used.

4.7.2 Fault Injection

Fault injection is a method for testing the fault-tolerance of a system with respect to the specified behavior. Fault injection is needed for two different purposes: to test the correct operation of the fault tolerance algorithms/mechanisms and to predict the dependability of the system in a realistic scenario where faults are expected to occur and have to be handled properly [12].

Our present fault injection techniques concentrate on disturbing the transmission channel. Therefore we use a fault injection hardware called *TTP-Stress*, which allows to disturb the communication channel of TTP. In a distributed fault-tolerant system the communication between the nodes is of utmost importance. Faults which are injected:

- Faults of the Physical Communication Layer: short cuts between transmission wires, short cuts to ground or power supply, loss of connections, faulty bus termination, etc.
- Loss of Frames: the switch off and reconfiguration of nodes can be simulated.
- Change of bits: the message contents can be changed.

The disturbances can be injected for a defined time interval, periodically or permanently. The start trigger is set manually through the user or via an external device, e.g., from a monitoring tool.

TTP supports different physical transmission layers. The currently implemented physical layer is in conformance with the ISO/DIS 11898 CAN [24] standard, i.e., differential transmission on a two-wire broadcast bus with one dominant state and one recessive state. The higher layers defined in the CAN specification (e.g. arbitration) do not apply for TTP.

5 Brake-by-wire Case Study

Automotive applications like brake- or steer-by-wire are typical examples for the use of a time-triggered architecture. We selected a *brake-by-wire* application, which we realised as a case study to evaluate the TTP protocol [8]. This application has several advantages:

- Realistic workload in a hard real-time application.
- Reuse for future realisations.
- Experiences on a real automotive example.

5.1 Requirement Specification

The requirement specification consists of the parts: specification of the system and definition of the test environment.

We realise the system specification in textual form with in-house used methods. So we do not have the opportunity to execute consistence checks in the description of the specification. Therefore we use the system model, which represents a detailed functional specification of the system.

5.2 Architectural System Design

Our fault tolerant architecture consists of a set of two redundant ECU's for the *Brake-by-Wire-Manager (BBW-Manager, BBWM)* and 4 single ECU's, one for each brake (see Figure 6). The ECU's are connected by two replicated busses. In this case study the brake ECU's are not designed redundant, in order to reduce costs, since the failure of a single brake is not considered to be as severe as the failure of the BBW-Manager.

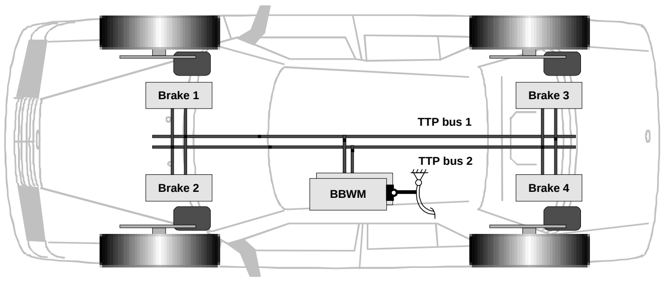

Figure 6: An Example for an TTA Architecture

The functional of the BBW-Manager is to read the sensor values of the brake pedal, the revolution counters of the wheels, the yaw-sensor, the acceleration sensors, and to calculate from these signals the brake force set points for the four brake actuators. The BBW-Manager also manages higher assistance functions like ABS, traction and driving dynamic control. The brake electronics get the brake force set points from the BBW-Manager.

The whole communication between the BBW-Manager, and the brake electronics is based on the fault tolerant TTP system.

In contrast to event triggered systems a TTP system is built upon a static message schedule. Figure 7 depicts the result of this phase, the static synchronous time division multiple access (TDMA) scheme and its constraint that each subsystem has to send exactly once in a TDMA cycle. The messages marked with ‘I’ are so called I-Frames, used for reintegration of rebooted nodes and do not transmit information for the application layer.

A TDMA slot has a length of about 1.2 msec. New brake force set points are sent every 7.2 msec, which is sufficient for an ABS control loop. The brake control ECU’s send their status and the current brake force. These messages are not so time critical as the transmission of the brake force set points. The brake ECU’s send their messages only once in a cluster cycle, each 12 TDMA slots. In the remaining slots the brake ECU’s send I-Frames for the network management.

5.3 Functional Design

As one example of a component with fail-silent property we describe in this section the realisation of the BBW-Manager. The four brake ECU’s are realised in a similar manner. The BBW-Manager has the functionality to calculate the four brake force set points. The brake force set points are safety related and have to be protected from transient and permanent faults. Figure 8 shows the schedule which is periodically executed on the BBW-Manager:

1. *Pedal signal measurement*, of the pedal signals from the pedal sensors.
2. *Pedal signal plausibility checks*, from the three pedal signals one valid value is calculated. This task is executed three times, to detect faults.
3. *Voter*, a voter task votes from the result of the three plausibility check tasks and starts an exception handling if a fault is detected.
4. *Brake force control*, the brake forces for the four actuators are calculated. This task is also executed three times.
5. *Voter*, a voter task votes from the result of the three brake force control tasks.
6. *TTP-communication*, the brake forces are send to the brakes via the TTP communication network.
7. *Diagnose*, a diagnose task is executed.
8. *Diagnose output*, the diagnose values are transmitted to an extern diagnose device.

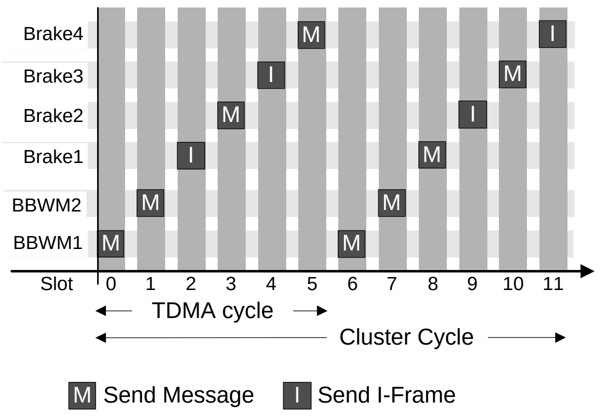

Figure 7: Communication Matrix of the Brake-by-wire Case Study

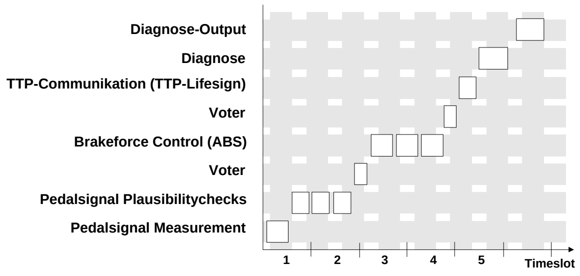

Figure 8: Local Task Schedule of the BBW-Manager

5.4 Functional Simulation

The system model represents the same structure of nodes as the real architecture. As an example we show the functional model of the system architecture (see Figure

9). We only present this part of the model, because the description of the whole model is beyond the scope of this paper.

The activity charts BBWM1_AC and BBWM2_AC represents the BBW-Manager nodes 1 and 2, similar the activity charts BRAKE_ACx the brake nodes. @ is part of the Statemate^{TM} syntax and means that a more detailed description of the activity exists. The external activities SENSORS and ACTUATORS represent the source and sink for the input and output data flows. The external activities MEDL_COMPILER and TADL_COMPILER, the configuration of the communication and task schedule of the nodes.

For the functional simulation first we use the single step mode to verify the model being complete and consistent. After this, we define the test pattern and exercise the simulation in the batch mode. We instrument the simulation for control and collecting of statistics during execution.

5.5 Fault Modeling

The fault module is a optional part of the system model and is represented by the activity chart TTP_STRESS_AC. To compare the results between fault modeling and fault injection the fault model has the same function as the fault injection device (TTP-Stress). In the real system we use a broadcast bus, therefore TTP-Stress can be connected at an arbitrary point with the bus. In the system model every TTP message transmitted on the bus is piped through the fault module. For the verification of the fault module behavior we use the same setup in fault modeling and fault injection.

5.6 Realisation and Integration

For the realisation we use the capability of automatic code generation of Statemate^{TM} and MATLAB$^{®}$/Simulink™. By using automatic code generation we still have the problem that we use non validated code generators; this problem will be addressed in our future work. As target platforms we use a VxWorks [29] based system for the BBW-Manager and a controller platform without floating point unit for the brake ECU´s. Currently download and execution of automatic generated code is only possible for the BBW-Manager. The complexity of BBW-Manager is much higher than the Brake ECU´s.

For system integration we use bus monitoring and additionally every node sends its internal states via a diagnosis bus to an external monitoring device. So we have the capability of monitoring and logging of all important system states.

5.7 Test and Fault Injection

We use functional test for the verification of our implementation. To define the test pattern out of the

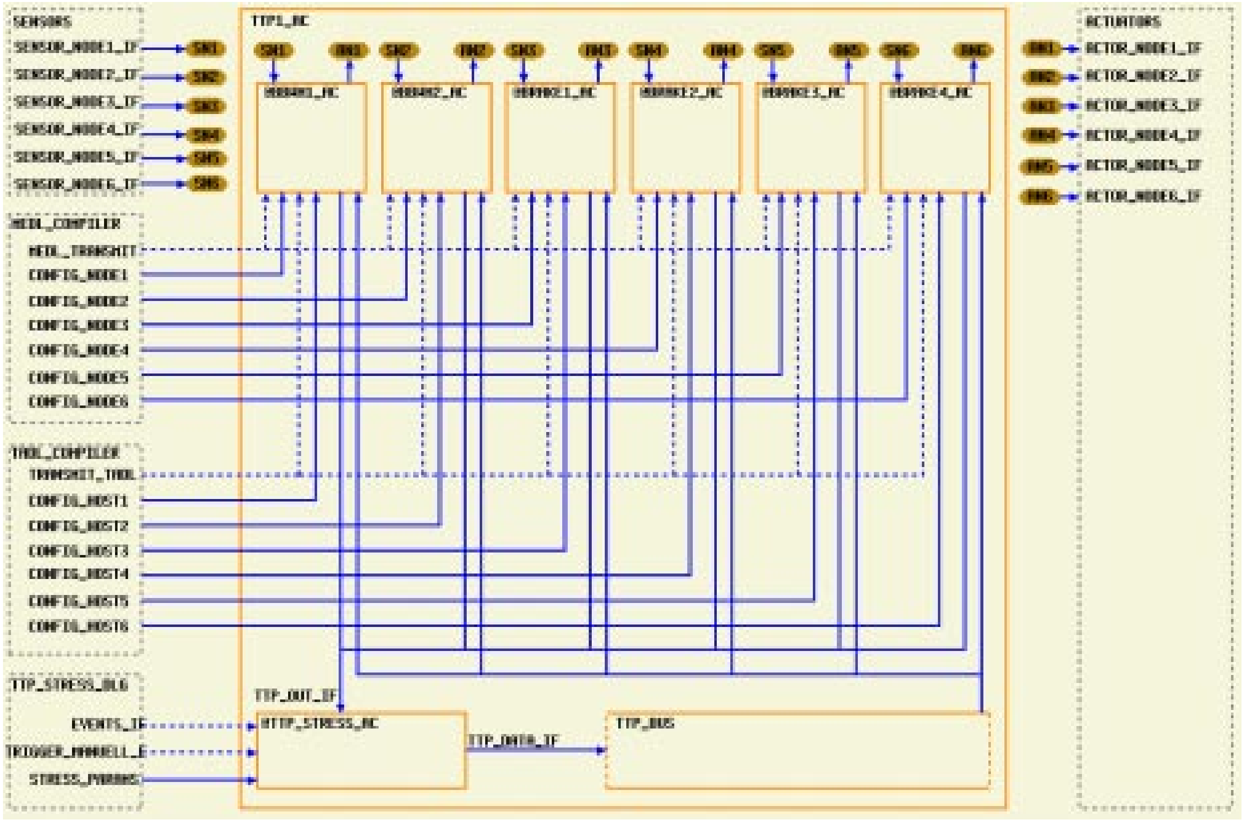

Figure 9: Functional Model of the System Architecture

requirement specification we use classifications trees [4].

The test pattern for the fault injection is separated into three groups; verification of the protocol behavior, verification of the fault, and verification of the system behavior under fault conditions.

The verification of the protocol behavior, depends on the protocol specification. We generate the fault pattern manually through examining the TTP specification. For the examination of the system behavior under fault conditions, we use the results of classification tree analysis.

6 Conclusion

In this paper we present a framework for the development of ultra-dependable automotive systems based on a time-triggered architecture. In the first part of the paper we introduce the time-triggered architecture and the time-triggered protocol. We showed that time-triggered architectures are well suited for safety critical automotive by-wire applications. In the second part we describe an approach is for the development of TTP based systems. Our approach based on verification of the system design by functional simulation, fault modeling, functional test and fault injection in the real system. At the end we gave a brief example of how this approach can be used in a brake-by-wire case study.

References

[1] C. Alexander, *Notes On The Synthesis of Form*, Harvard University Press, Cambridge, Massachusetts and London, England, 1964.

[2] E. Dilger, L.A. Johansson, H. Kopetz, M. Krug, P. Lidén, G. McCall, P. Mortara, B. Müller, U. Panizza, S. Poledna, A.V. Schedl, J. Söderberg, M. Strömber, T. Thurner: *Towards an Architecture for Safety Related Fault Tolerant Systems in Vehicles*, Proceedings of the ESREL´97 International Conference, 1997.

[3] Fleisch, Ringler Th. and Belschner, R., *Simulation of Application Software for a TTP Real-Time Subsystem*. Proc. of European Simulation Symposium, Istanbul, Turkey, June 1997.

[4] K. Grimm, M. Grochtmann, *Classification Trees for Partition Testing*, in Software Testing, Verification and Reliablility, Bd. 3, No. 2, pp. 63-82, 1993.

[5] D. Harel, *Statecharts: a visual formalism for complex systems*, Science of Computer Programming, vol. 8, no. 3, pp. 231-274,1987.

[6] D. Harel et al., *On the formal semantics of Statecharts*, in Proc. 2nd IEEE Symposium on Logic in Computer Science, IEEE Press, NY, USA, pp. 54-64, 1987.

[7] D. Harel et al., *Statemate™: a working environment for the development of complex reactive systems*, IEEE Trans. On Software Engineering, vol. SE-16, no. 4, pp. 403-414, 1990.

[8] B. Hedenetz, R. Belschner, *Brake-by-wire without Mechanical Backup by Using a TTP-Communication Network*, SAE International Congress 1998.

[9] G. Heiner, T. Thurner, *Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems*, FTCS-28, June 1998.

[10] IEEE Std. 1074.1991, *IEEE Standard for Developing Software Lifecycle Processes*, The Institute of Electrical and Electronics Engineers, Inc., 1991.

[11] ISO/ICE 1508, *Functional safety: safety-related systems*, International Electrotechnical Commission, 1995.

[12] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, *Integration and Comparison of Three Physical Fault Injection Techniques*, Predictably Dependable Computing Systems, Springer Verlag 309-329, 1995.

[13] H. Kopetz, et. al., *A Prototype Implementation of a TTP/C Controller*, Proceedings SAE Congress 1997, Detroit, MI, USA, Febr. 1997. Society of Automotive Engineers, SAE Press. SAE Paper No. 970296.

[14] H. Kopetz, *Real-Time Systems - Design Principles for Distributed Real-Time Systems*, Kluwers Academic Publishers, 1997.

[15] H. Kopetz, G. Grünsteidl, *TTP - A Protocol for Fault-Tolerant Real-Time Systems*, IEEE Computer, pages 14-23, January 1994.

[16] A. Krüger, *Interface design for Time-Triggered Real-Time System Architectures*, doctor thesis, Institut für Technische Informatik, Vienna University of Technology, 1997.

[17] M. Krug, A. V. Schedl, *New Demands for Invehicle Networks*, Proceedings of the 23rd EUROMICRO Conference, pp. 601-606, 1997.

[18] N. Leveson, *Safeware - System safety and computers*, Addison-Wesley, Reading, MA, 1995.

[19] MathWorks, *MATLAB® - The Language of Technical Computing*, MathWorks Inc., MATLAB® 5.1, June 1997.

[20] G.J. Myers, *The Art of Software Testing*, Wiley-Interscience, Chichester, 1979.

[21] T.J. Ostrand, M.J. Balcer, *The Category-Partition Method for Specifying and Generating Functional Test*, in Communications of the ACM, Bd. 31, Nr. 6, Juni 1988.

[22] N. Parrington, M. Roper, *Softwaretest*; Mc Graw-Hill, Hamburg, 1990

[23] S. Poledna, *The Problem of Replica Determinism*, Fault-Tolerant Real-Time Systems, Kluwer Academic Publishers, 1996.

[24] SAE, *Control Area Network: an invehicle serial communication protocol*, SAE Information Report J1583, SAE Handbook, 1990.

[25] SAE, *Class C Application Requirement Considerations*, SAE Recommended Practice J2056/1, SAE, June 1993.

[26] SAE, *Survey of Known Protocols*, SAE Information Report J2056/2, SAE, April 1993.

[27] I. Sommerville, *Software Engineering*, Addison-Wesley Publishing Company, Wokingham, England, 3rd edition, 1989.

[28] URL: http://www.isi.com/products/matrixx/.

[29] URL: http://www.wrs.com/.