A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems

Eric Armengaud, Andreas Steininger
Vienna University of Technology
Embedded Computing Systems Group E182-2
Treitlstr. 3, A-1040 Vienna
{armengaud, steininger}@ecs.tuwien.ac.at

Martin Horauer
University of Applied Sciences
Technikum Wien
Höchstädtplatz 5, A-1200 Vienna
horauer@technikum-wien.at

Roman Pallierer
Dependable Computer Systems
DECOMSYS GmbH
Stumpergasse 48/28, A-1060 Vienna
pallierer@decomsys.com

Abstract

This paper presents a layer model tailored for the test of distributed systems that rely on the time-triggered paradigm, such as the FlexRay protocol that is currently employed in the automotive industry. The presented layer model is applied for the generation of a fault model, aids in the inspection of fault propagation throughout the distributed system under consideration and is used for fault diagnosis of defective electronic control units. To that end, this systematic test and diagnosis approach will provide a solid basis for analyzing and verifying future by-wire systems with respect to their communication properties.

1 Introduction

Automotive electronics is the key innovation driver in the automotive domain. Analysts estimate that more than 80 percent of all automotive innovations now stem from electronic systems [5]. High-end cars such as the BMW 7, already contain up to 70 electronic control units (ECUs), and even middle-class cars comprise up to 40 ECUs. The use of these ECUs and especially their interoperation allow the establishment of increased and improved functionality in comparison with stand-alone components (e.g., combining speed with steering information). For non safety-critical communication (central locking, seat motion control, power windows, etc.), automotive multimedia and personal computer networking a number of fieldbus protocols are already in wide-spread use, such as CAN, LIN or Byteflight. For safety-critical and advanced control applications (”X-by-wire systems”) an industrial consortium of leading automotive and electronic manufacturers is establishing a common communication protocol, termed FlexRay. With these by-wire systems it will be possible to replace rigid mechanical and hydraulic components with configurable electronic elements. They make vehicles lighter, cheaper, safer, and more fuel-efficient and, in addition, will enable an unprecedented boost of functionality of such systems. Customers, however, will probably hesitate to accept this completely new technology. Therefore it is of utmost importance to establish confidence in automotive electronic systems by all possible means. Undoubtedly, one of these means will be testing [7]. Efficient methods for test and diagnosis will be required to check the functional integrity of all involved components and hence prevent failures, see [4, 9] for some recent examples. Successful approaches for fault diagnosis and testing as employed in the semiconductor industry – see [8] for a survey – may be applicable to the automotive industry as well when tailored to the strict cost and safety constraints.

Since communication is the most important enabler for distributed functionality, the focus of our STEACS[1] project is set on diagnosis and testing methods for the communication subsystem. One main scientific challenge lies in the elaboration of a systematic test approach. While methods for testing of the computing nodes on the one side and the bus on the other side do exist, a unified and systematic test approach is required that does not only consider the function of these singular components in isolation. Experience

[1]The STEACS-project received support from the Austrian ”FIT-IT[embedded systems” initiative, funded by the Austrian Ministry for Traffic, Innovation and Technology (BMVIT) and managed by the Austrian Industrial Research Promotion Fund (FFF) under grant 807146.
See http://embsys.technikum-wien.at/steacs.html for further information.

shows that problems with interaction of ”fault-free” components are becoming increasingly relevant in practice. The problem is further aggravated by the large number of new product variants.

From a conceptual point of view this testing problem is particularly interesting, since the usual ”divide and conquer” test approach cannot be applied here – in some sense it can even be viewed as the root of the problem. Another structuring method must be found that goes beyond the physical borders and allows the inclusion of distributed services.

In this paper we propose a layer-based approach for structuring the test process of the communication system. Following a brief summary of our motivation in Sec. 2, we present a generic method for structuring the layers of a communication system and illustrate its usage by presenting a detailed layer model of a FlexRay-based communication system along with a short description. To demonstrate the usefulness of the model we apply it to some example problems in Sec. 4. Finally, in Sec. 5 we conclude the paper.

2 Rationale

A fundamental property of time-triggered systems is their composability in the temporal domain that, in principle, simplifies the integration and coexistence of different nodes. In practice, however, wrong configuration parameters or slightly out of specification values of the employed nodes restrict us from simply performing the test in a node-by-node manner. Performing an unstructured functional test of the configured system, on the other hand, is not reasonable, since the complexity of the distributed system is substantially higher than that of a single node. As test efforts rise more than linearly – typically $O(n^2)$ cf. [2] – with system complexity this would either result in excessive test duration or in poor test coverage. Thus, some kind of structured test is mandatory.

The fundamental purpose of a test is to check whether the system under test provides all services as specified. Should any service deviate from the specification or be completely missing, we have encountered a failure, and this is exactly what the test is intended to discover. If we want to structure the test, it will probably be a good strategy to identify and separate all services, break them into sub-services (which we call mechanisms) as far as possible and focus the test to each of these mechanisms separately. The rationale why we chose to employ a layer model as the central point of our test concept is the following:

1. The layer definition is generally based on the notion of services (see Sec. 3) and hence suits naturally to our intended test strategy. In particular, the layer model reflects all services provided by the system with a fine – ideally atomic – granularity.

2. In the process of model elaboration the decomposition of the complex global communication service into smaller mechanisms is performed in a systematic manner. This aids in achieving a complete picture of all relevant mechanisms.

3. The comprehensive layer model shows the individual services and mechanisms and also their interrelations. In that way it is possible to identify all relevant inputs of the particular service or mechanism as well as the receivers of the service outputs in the next level. This can substantially simplify diagnosis and helps in identifying potential sources of error (fault model).

4. By using the layer model one can determine the abstraction level at which monitoring should be performed to test a particular mechanism.

5. If potential error signals issued by a mechanism are included in the model, a hierarchy of error signals can be constructed that further eases diagnosis.

6. Furthermore, a detailed model will enable an inspection of fault propagation, cf. [3].

In order to test the fault tolerance- and error detection capabilities of the communication system in our project we will employ fault injection. Again the layer model can be helpful to provide a systematic structure for planning these experiments, however, we will not cover these topics in this paper.

3 A layer model of a time-triggered architecture

The OSI reference model provides a standardized way to structure the functional entities of a communication system into a set of layers. In practice, however, the majority of modern communication systems typically only use a subset of the proposed layers due to reasons of efficiency demanded by every particular implementation.

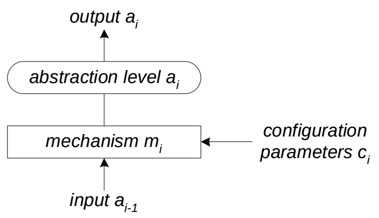

Figure 1. Abstraction Level and Mechanism

For a systematic test and diagnosis of a communication system it is desirable to have at least one or a small set of dedicated nodes with extended monitoring and some fault-injection capabilities. The idea herein is to monitor the communicated data stream at functional boundaries within the receiver of a test node that is assumed to behave correctly. When an erroneous condition is detected it will be possible to draw conclusions from the location where the fault was detected to the originator of the misbehavior, see Sec. 4. To extract useful information, however, a detailed structuring of the involved layers is required. To that end, we introduce the concept of abstraction levels a_i and mechanisms m_i. An abstraction level represents a detailed view of the system whereas a mechanism provides simple – ideally atomic – services that are controlled by several inputs a_{i-1} and optional configuration parameters c_i and produce one or more outputs a_i, cf. Fig. 1.

3.1 The FlexRay example

Fig. 2 presents a detailed layer model of a FlexRay communication system (see [6]) based on the functional entities ”abstraction level” and ”mechanism” described above and structured according to the OSI reference model.

Herein, the lowest layer is the physical layer whose service is to transmit raw bits from a node to another through a communication channel. Its interface to the upper layers are serial bit streams received from or to be sent through the communication system.

The next layer above is the data link layer whose services organize the bit stream into a frame structure, detect potential transmission errors, and control the medium access. In contrast to the data link layer defined in the OSI reference model, the FlexRay protocol specification does not include an acknowledgement system to automatically initiate re-transmission of faulty frames. In addition, this layer provides a view of the network synchronized time that handles the medium access control (MAC) service.

The network layer, transport layer and session layer defined in the OSI reference model are not present in the presented model because the corresponding services are not particularly pronounced in FlexRay: With a communication based on broadcast channels, every frame transmitted is seen by all receivers throughout the distributed system and the addresses are implied by the bus schedule; therefore explicit routing as provided by the network layer is not needed. Typical services of a transport layer are fragmentation and re-combination of large data or the provision of reliable communication. At this particular layer these services, however, are not necessarily needed by applications in the automotive domain; hence, they were not implemented here. Finally, the session layer is omitted because establishment and control of sessions between two nodes are implied by the static schedule.

In FlexRay, the presentation layer is located upon the data link layer. Its services are to format the data contents according to the needs of the application and present a signal based interface to the application layer. Furthermore, the optional fault-tolerant communication paradigm as specified by the OSEK/VDX[2] joint project of the automotive industry are in the context of the presentation layer as well.

Finally the application layer is situated on top of the model and makes use of the services of the lower layers to communicate from a node to another.

Although these layers are arranged in a hierarchical order, the mechanisms within the layers are not necessarily hierarchically organized; in fact, some loops are present. These loops make the system more difficult to test and diagnose because the inputs of a mechanism might be indirectly dependent on its outputs. However, by breaking the loop the mechanism can be controlled directly at its inputs and observed at its outputs and, hence, totally tested.

[2]http://www.osek-vdx.org/

4 Application of the model

In Sec. 2 we have briefly summarized the aims we want to achieve with our model. Having given a brief overview and illustrated a detailed model for the FlexRay communication system in Sec. 3 we want to apply this model now to some practical problems in order to investigate, how far it actually serves our purpose. As fault hypothesis we consider single faults within the communication subsystem and presume a properly working tester node. Given this hypothesis we investigate how the presented layer model can be used for fault model generation, fault propagation analysis and diagnosis.

4.1 Fault model generation

With the fine grained structure of the time-triggered layer model generation of a fault model boils down to the systematic search for reasons why a particular mechanism should provide an incorrect output. If we perform this analysis for every mechanism, and if no hidden dependencies between mechanisms exist (i.e. ones are not included in our model), then we have finally developed an exhaustive fault model for our system (still excluding global sources of error like power supply or clock, however).

Looking at the structure of a mechanism in our model we can identify three reasons for a mechanism to produce an erroneous output (see Fig. 3)

(a) *The mechanism operates on erroneous inputs.* This case represents the propagation of faults and is hence not part of the fault model. A more thorough treatment of fault propagation can be found in Sec. 4.2.

(b) *The mechanism is provided with erroneous parameters.* This scenario can be attributed either to faulty configuration data (e.g. an old version) or a hardware fault in the configuration memory (e.g. a stuck-at error in a bit-cell).

If we consider the frame structuring mechanism in the receive path, e.g., an erroneous configuration of frame start sequence length or pattern would fall into this category.

(c) *A fault in the implementation or underlying hardware of the mechanism.* This may either be the consequence of a hardware defect, or of a design or implementation fault. E.g., for the frame structuring mechanism this would mean that valid frame start sequences are not properly detected (i.e. not according to the configuration); whereas other bit patterns might be erroneously identified as frame start sequence instead.

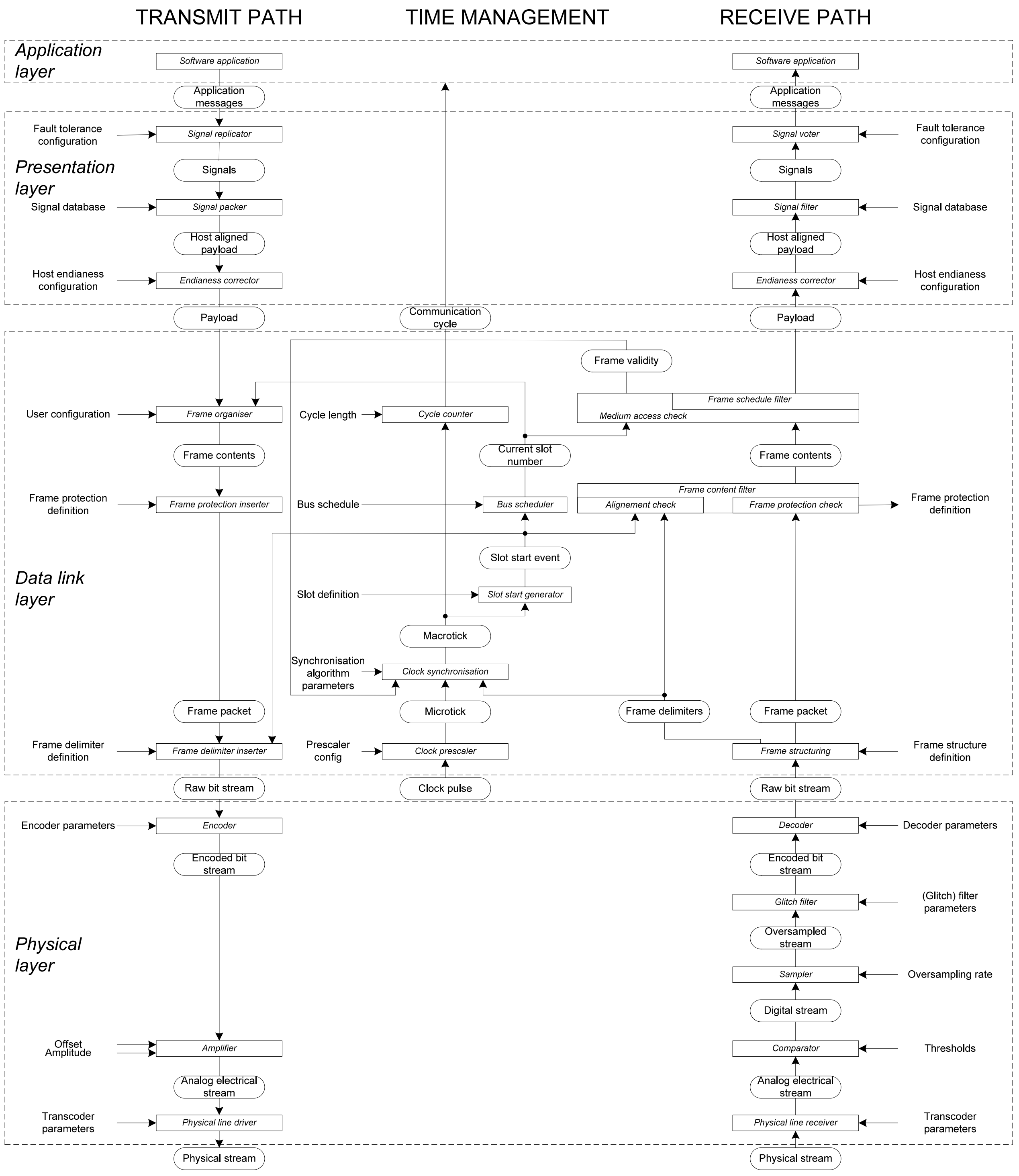

Figure 2. Layer Model for a Time Triggered Architecture (FlexRay)

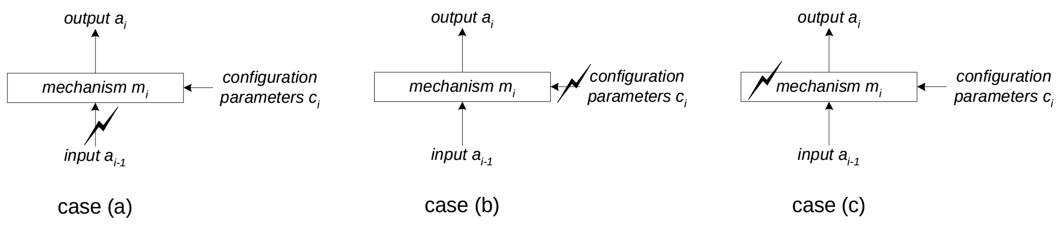

Figure 3. Reasons for a mechanism to produce an erroneous output

Type	Input	Output	Interpretation	
1	Ok	Ok	Normal Operation; the mechanism works properly	
2	Ok	Error	Mechanism Failure	Erroneous Parameters; case (b)
				Implementation Fault or HW defect; case (c)
3	Error	Error	Fault Propagation; case (a); see Sec. 4.2	
4	Error	Ok	Fault Masking; see Sec. 4.2	

Table 1. Interpretation of the input/output behavior of a mechanism

Tab. 1 summarizes all possible cases that can be distinguished based on an observation of the mechanism's input/output mapping.

Notice that cases (b) and (c) of Fig. 3 differ from case (a) in that the faulty output is generated from a correct input. Further notice that cases (b) and (c) cannot be distinguished from one another by simply observing the mapping between input and output, since they both show the same behavior to the outside, i.e. type 2.

A substantial advantage of the proposed mechanism-based approach to fault model generation is that we are dealing with symptoms rather than actual sources of errors: From the service point of view it is sufficient to find out whether the configuration parameters, e.g., are faulty. This drastically reduces the size of a fault model and hence makes an "exhaustive" approach feasible, albeit with a limited level of detail. Should a higher resolution be desired for a specific purpose, we can further investigate the liable physical effect by starting with the knowledge of the symptom resulting in a reduced search space. For the example of faulty configuration parameters this would mean identifying the reason why the configuration is faulty.

4.2 Error propagation

As a prerequisite for diagnosis we have to elaborate a thorough understanding of how errors propagate in the system. With the proposed model the study of error propagation involves the identification of a mechanism's possible reactions to an erroneous input. From an input/output point of view this is summarized by behavior types 3 and 4 in Tab. 1. In the following, however, we will derive a more detailed classification of the mechanisms' behavior.

For this purpose we must first of all separate between the transmit and receive path. These paths behave different, since in the transmit path additional information is appended to the actual application data to enable a transmission, while this information is successively stripped in the receive path.

4.2.1 Error propagation in the transmit path

A mechanism m_i of the transmit path receives its input a_i from the output of mechanism m_{i+1} and processes it, typically, by appending some additional information or by structuring the given input. While some constraints with respect to the format of a_i may apply (interface specification), in general no check or analysis can be performed for the input value. Mechanism m_i is usually not supposed to identify illegal values in m_{i+1}'s output, but rather processes all possible input values transparently. Since the source of the information is the chain of mechanisms above m_i , the source of the error must also be there, i.e. in mechanisms with index higher than i. E.g., the frame delimiter mechanism appends the required delimiters to any frame packet it receives at its input without being able to decide whether the packet is corrupted or not.

4.2.2 Error propagation in the receive path

Here the input a_{i-1} for mechanism m_i is provided by the output of mechanism m_{i-1}. Processing this input often involves some kind of dedicated rule checking or analysis in the value and time domains in addition to checks in the structural domain. Specifically, in the data link layer redundant portions of the incoming data are used for checking and are then discarded, or the consistency with a timing schedule is checked. The source of information is the chain of mechanisms below m_i, hence a potential source of an error must be assumed in the mechanisms with an index lower than i in the receive path of node N_r (the node under consideration), the bus or the transmit path of the sending node N_s.

Under the assumption of single faults, an error that is detected at the abstraction level a_i in the receive path of node N_r can not originate from a mechanism higher than m_i in the transmit path of the sending node N_s, see Fig. 4. The reason is as follows: Erroneous information originating from mechanism m_i in node N_s's transmit path will

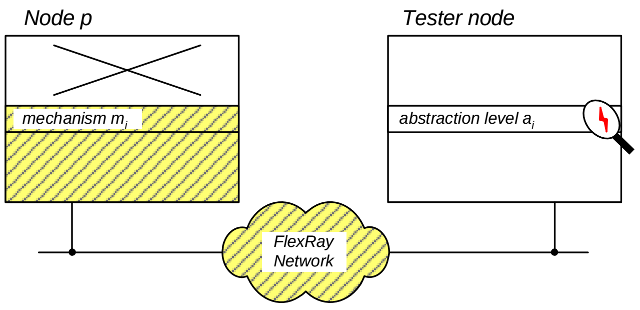

Figure 4. Remote diagnosis by a tester node

be properly "wrapped" by mechanism m_{i-1} and all subsequent mechanisms in the transmit path. Then it will be sent over the bus and unwrapped by node N_r's mechanisms in the receive path without problems until it reaches mechanism m_i. (From the point of view of a layer model, we could as well imagine that a virtual communication from mechanism m_{i-1} of node N_s to mechanism m_{i-1} of node N_r has taken place.). There is no reason why the unwrapping of the properly wrapped incorrect information should cause problems. Hence transmitter errors will show up for the first time in mechanism m_i of node N_r's receive path, or later, if the check performed by m_i is not sufficient to detect the error. Looking back from receiver to transmitter this means that an error detected by mechanism m_i cannot originate from mechanisms higher than m_i including the transmit path, which is essentially the statement we wanted to prove. E.g., an erroneous input to the frame structuring mechanism might be a raw bit stream with a corrupted start sequence. The origin of this error may be anywhere on the path between frame delimiter insertion mechanism of the transmitter and frame structuring mechanism at the receiver. A failure of the transmitter's endianness corrector, however, will never show up in the frame structuring mechanism.

4.2.3 Error detection properties

Previously, we have mentioned that some mechanisms already perform checks on the structure and/or the content of the incoming information themselves. In general, such checks are infrequently encountered in the transmit path whereas they are very common in the receive path. If such a check detects an error, some kind of error indication signal is activated to trigger an appropriate error handling. With a perfect error detection on every single mechanism the interpretation of the error indication would be straightforward: An error indicated by mechanism m_i in the receive path of node N_r can either be traced to a failure of mechanism m_{i-1} in the receive path (in a perfect system failures of mechanisms with index $j < i-1$ would have been detected by the respective subsequent mechanism m_{j+1}) or originate from the transmit path of the sending node N_s. In particular, it can be assigned to mechanism m_i of N_s's transmit path for the following reason: We have already argued that failures of transmit mechanisms with index higher than i cannot be detected by receiver mechanism i. On the other hand, any failure of a transmitter mechanism with index j lower than i would be detected by the corresponding receiver mechanism m_j.

Unfortunately there are several reasons why error detection is not perfect in practice:

- Since mechanisms usually check specific properties of the incoming data only, errors that affect other properties are not accounted for. The frame alignment check, e.g., evaluates the temporal alignment of the frame only and does not consider the actual frame content.

- For several checks there is a non-zero probability that erroneous data – even if explicitly checked – is still regarded as valid (e.g. the aliasing probability of CRC checks).

- The check is performed somewhere between the input and output of the mechanism. As a result an error detected by mechanism m_i will be assigned to the output of mechanism m_{i-1}, although it may as well have originated in the input path of m_i before the actual check. A stuck bit at the input of the frame protection check, e.g., will have the same effect on the result of the CRC check as an erroneous frame packet coming from the frame structuring mechanism. A distinction can be made here by directly monitoring the frame packets.

4.2.4 Classification of the output behavior

Independent of an error indication, the mechanism will process the erroneous input and may react in several ways:

(A) View the erroneous input as valid and produce the respective output (direct error propagation), e.g., the endianness corrector.

(B) Suppress the erroneous input and produce no output at all (omission). In general this implies some error detection capabilities, e.g., the frame protection check.

(C) Produce a correct output in spite of the incorrect input (fault masking), e.g., the glitch filter. This will normally happen if error correction techniques are applied, in rare cases when the erroneous portion of the input does not contribute to generating the output or simply by chance.

(D) Produce any arbitrary output in an undefined (or at least unexpected) way (undefined behavior). In this case the mechanism will probably be upset by the erroneous input and its further function is questionable. This is normally viewed as a design flaw. E.g., a state machine within a decoder enters an illegal state due to an illegal encoded bit stream at the input.

Clearly the cases (B) and (C) are desirable. Notice, however, that even in case (A) the mechanism provides a correct mapping from a given input to the specified output from a local view, hence it cannot be considered faulty. Further notice that it is easy to distinguish cases (A) ...(D) if output and input of the mechanism are monitored, while this distinction may become fuzzy otherwise.

Tab. 2 summarizes the conceivable properties with respect to a mechanism's error propagation behavior.

4.2.5 The time management path

In addition to the transmit path and the receive path Tab. 2 also shows the Time Management path. Basically mechanisms within this path allow the same classification of output behavior: While the bus scheduler, e.g., simply propagates erroneous slot start events to erroneous slot numbers, fault masking is achieved by the inherent fault tolerance of the clock synchronization mechanism. No mechanism, however, in the Time Management path is designed to exhibit omission.

The error detection behavior of this path is very interesting: Although the path is mainly ascending (such as the receive path), we could not identify any mechanisms that are able to generate an error indication by themselves. The time management, however, provides several services to the transmit path and the receive path, part of which is directly used for checking (alignment check and medium access check). Therefore failures of mechanisms within the time management are very likely to lead to an error indication within the receive path, i.e. in an indirect way. Similarly an incorrectly generated slot start event or slot number will cause transmission errors that are finally detected by the receiver node.

It is interesting to observe that the time management path has only one relevant input from the message stream, namely the frame delimiters. This input is very well protected (a) by the fault tolerant clock synchronization algorithm and (b) by the subsequent check of the frame validity (that ultimately relies on the error detection properties of the receive path), such that error propagation is inhibited. Due to its very limited local extension and its well protected inputs this path is often implemented without explicit protection by the other mechanisms.

4.3 Fault diagnosis

Basically, a fault that is not visible for the application is not relevant, and hence, not worth being traced since the system still completely fulfils its purpose in spite of this fault. There are, however, two good reasons why we still take the efforts to perform monitoring on a mechanism by mechanism base:

(I) Since many faults may become masked on higher layers, error detection latency is much higher compared to the case when we can perform monitoring directly at the output of the respective mechanisms. In particular, if we perform error detection on the application level only, it might take a long time until a constellation occurs in which the fault propagates to this layer.

(II) If we are interested in diagnosis rather than the mere detection of the error, we have to collect indications on the root of the problem, and this requires monitoring on different layers and abstraction levels.

Monitoring on different abstraction levels requires a special implementation of the communication controller, since access to specific points in the data processing pipeline is required. Furthermore, provisions must be made to direct the respective stream of monitoring data to a storage device (or an online evaluation). Currently we are developing such a monitoring hard+software in the scope of our STEACS project, see [1] for a first prototype.

While being a perfect goal in theory complete observability on all layers and abstraction levels causes excessive overheads in practice. Moreover, the mere collection of information by monitoring is not sufficient – an appropriate interpretation is crucial for diagnosis. Our layer model can aid us in both, determining the appropriate number and location of layers and abstraction levels to perform monitoring for a given problem, and finding the correct interpretation of the assessed information.

Concerning the interpretation of the collected data we can make use of the fault model and the error propagation analysis described above, and elaborate a diagnostic matrix that relates every fault with one or more syndromes. A syndrome can be viewed as the trace of an error when propagating through the layers and mechanisms. We can easily create a syndrome for a single mechanism by measuring the input and the output at the respective abstraction levels and additionally observing potential error indications. From this measurement we can derive the input/output behavior according to the proposed classification given in Tab. 1 and Tab. 2. Finally, we put these "local" syndromes for the single mechanisms in context to form a global syndrome. In this sense an example syndrome for the receive path with the observation at the physical layer would be

	case 1	case 2
physical → raw bit stream	*correct*	*erroneous*
frame packet:	*erroneous*	*omitted*
frame contents → signals	*omitted*	*omitted*
frame protection check error	*active*	*inactive*
all other error indications	*inactive*	*inactive*

Conclusion (case 1): A frame structuring mechanism failure occurred (correct input/erroneous output). The erroneous output is filtered out by the frame protection check.

Conclusion (case 2): The frame delimiter inserter at the sender has failed. The resulting error is filtered out by the frame structuring mechanism, thus, no error indication is given.

Path	Origin of fault detected in m_i	Error Indication	Output Behavior
Receiver	receive path: all m_j with $j < i$	Yes, (some mechanisms)	(A) Propagation
			(B) Omission
	transmit path of sender: all m_j with $j < i$		(C) Masking
			(D) Undefined
Transmitter	all m_j with $j > i$	No	(A) Propagation
			(D) Undefined
Time Management	Lower mechanisms within local MAC or sender's MAC	Indirect	(A) Propagation
			(C) Masking
			(D) Undefined

Table 2. Error propagation behavior

Based on the fine-grained measurement data and the thorough understanding of error propagation, it is quite easy to perform diagnosis. A systematic approach to this would be to build a table listing every fault in the fault model along with the related syndromes in advance and use this table in the reverse direction as diagnosis matrix. With this table we can immediately identify the cause (or a list of potential causes) for every syndrome we encounter. Notice that the effect of a fault may depend on circumstances like, e.g., data contents. As a consequence the respective syndrome also varies. In this way the set of different syndromes that are observed over time become quite characteristic for the fault. However, the syndrome alone does not allow us to distinguish between a parameter fault and an implementation or hardware fault in the frame structuring mechanism. This is because they all have the same effect on the input/output behavior.

4.3.1 Application to remote diagnosis

Normally we would not like to perform extensive monitoring as suggested above on all nodes within a network. It would be much smarter if we could detect and diagnose mechanism failures on all nodes contributing in the communication by means of a single dedicated tester node which is the only one that is equipped with monitoring capabilities. In fact, it is possible to project an error encountered in the receive path on tester node N_t to the transmit path of the sender node N_s. Without further provisions, however, it is not possible to diagnose the receive path of node N_s in this setup. There are, however, two solutions to this problem:

(1) We can employ a specific "loopback" application on N_s, such that any information received by N_s is sent out to the bus and can hence be analyzed by N_t.

(2) We can make use of the loopback that exists anyway in the guise of the Time Management path: In Fig. 2 it can be seen that this path utilizes information from the frame structuring and medium access check mechanisms within the receive path to generate the required information for the transmit path. Hence careful analysis of the transmit behavior of N_s allows conclusions on the receive path. It should be considered, however, that the fault masking behavior of the clock synchronization mechanism renders this conclusion very vague.

5 Conclusion and future prospects

In this paper we presented a layer model tailored for the test and diagnosis of a time triggered distributed system architecture by using the FlexRay protocol as an example. The structure of the layer model is presented in accordance to the OSI model, however, it uses the functional entities "abstraction level" and "mechanism" to provide a finer granularity. Using this layer model we show how it aids us in the generation of a fault model that allows dealing with symptoms rather than actual sources of errors; a fact that drastically improves on the complexity and practicality. Next, we study the propagation of errors in the transmit, the receive and the time management path – the latter being specific for time triggered architectures – respectively. Furthermore, we investigate common error detection properties and classify the output behavior of the involved mechanisms. Finally, we show how our model can be applied to fault diagnosis, in particular how remote and node local faults can be identified with one test node.

In the next step we are planning to introduce faults into the communication system by means of fault injection with the purpose to test the error detection and fault-tolerance mechanisms and assess the robustness of our target system with respect to a given fault set. In this context we employ our presented models to identify where the modification should be made in order to efficiently check the target feature. The direct access to the different mechanisms that we have already implemented for the purpose of monitoring will allow a substantial improvement of controllability and observability in these experiments.

References

[1] E. Armengaud, A. Steininger, M. Horauer, R. Pallierer, and H. Friedl. A Monitoring Concept for an Automotive Distributed Network - The FlexRay Example. 7th *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'04)*, pages 173–178, 2004.

[2] C. F. Hawkins, H. T. Nagle, R. R. Fritzemeier, and J. R. Guth. The VLSI Circuit Test Problem - A Tutorial. *IEEE Trans. On Industrial Electronics*, 36(2):111–116, May 1989.

[3] M. Hiller, A. Jhumka, and S. N. An Approach for Analyzing the Propagation of Data Errors in Software. *Proc. of the International Conference on Dependable Systems and Networks (DSN'01)*, pages 161–170, 2001.

[4] N. Kandasamy, J. Hayes, and B. Murray. Time-constrained failure diagnosis in Distributed Embedded Systems. *Proc. of the International Conference on Dependable Systems and Networks*, pages 449–458, 2002.

[5] G. Leen and D. Hefferman. In-Vehicle Networks, Expanding Automotive Electronic Systems. *IEEE Transaction on Computers*, pages 88–93, January 2002.

[6] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann. FlexRay - The Communication System for Advanced Automotive Control Systems. *Society of Automotive Engineers (SAE) 2001 World Congress*, March 2001.

[7] G. Reichart. Systemintegration in der Elektrik/Elektronik. *7. EUROFORUM Jahrestagung Elektronik Systeme im Automobil*, February 2003. (in German).

[8] A. Steininger. Testing and Built-in-Self-Test - A Survey. *Journal of Systems Architecture*, 46:721–747, 2000.

[9] R. Tappe and D. Ehrhardt. Dynamic Tests in Complex Systems. *Proc. of the* 32^{nd} *International Test Conference ITC*, pages 609–614, 2001.