Towards a Systematic Test for Embedded Automotive Communication Systems

Eric Armengaud, Andreas Steininger, and Martin Horauer

Abstract— **The introduction of computer-controlled intelligent safety and comfort features has turned cars into complex distributed computing systems. In such a system the proper operation of the communication backbone as well as the proper interaction of components from different vendors must be ensured for all configurations and operating conditions. This system-level test goes far beyond the (isolated) test of single components and represents a substantial problem, that seems to be still largely unsolved, although its solution is crucial for maintaining the consumers' trust in modern automotive electronics.**

In this paper we concentrate on the test of distributed systems based on FlexRay, the protocol that is envisioned as the communication backbone for future automotive systems. The cornerstones of our approach are a decomposition of the system into layers and mechanisms, and a versatile strategy for monitoring and stimulation under various conditions. Our concept can be adapted to diverse needs ranging from an early debugging with full access to the system, over non-intrusive online testing during inter-operability tests, to maintenance testing that is restricted to a remote access only. We give detailed discussions of the requirements and present our solutions for the various issues involved. Selected use cases demonstrate the usefulness of the taken approach.

Index Terms— **System Test, Embedded Systems, Time-Triggered Communication, Automotive Electronics, FlexRay.**

I. Motivation

DOZENS of complaints like *"while overtaking another car the engine unexpectedly went into emergency mode and dropped to a crawl"* or *"windows open automatically when in heavy rain"* were reported to a local Austrian automotive magazine by its readers for a survey on problems with electronics in new cars [1]. The fact that virtually all brands were affected, from low-cost to luxury, clearly indicates substantial problems with the integration of electronic systems in general. Still nobody is seriously talking about returning to the "good old days". Electronics has been the key innovation driver for automotive systems throughout the last decade, and this situation is not going to change in the near future. The vehicle networks are transforming automotive components into truly distributed electronic systems, e.g. with up to 72 units interconnected with various different fieldbus systems in the new Mercedes S-class, see [2]. Replacing rigid mechanical components with dynamically configurable electronic elements triggers an almost organic, system wide level of integration. As a result, the automotive industry expects savings resulting in improved environmental tolerance and cost reduction. Sophisticated and more complex features for safety and comfort such as chassis control and smart sensors are expected to be developed faster and more reliably and will likely become mainstream. Unfortunately, all these functionalities translate into higher performance (resource usage), reliability and maintainability requirements for future automotive systems, while stringent cost constraints and the need to allow the interchangeable use of components from different vendors rule out solutions found in other domains. It seems that this development has brought us to the point where today's technology is not prepared to keep automotive embedded systems at a quality level that is traditionally expected from a car in general.

A cornerstone for the enormous success of embedded systems in automotive applications is their role as an enabler for a tight coupling and interaction of multiple sensors and functions. This ability has paved the way to services (e.g. the electronic stability program) that are impossible to build upon traditional mechanical solutions alone. The automotive industry has recognized the central role of the communication system in this context and recently introduced time-triggered communication to succeed and supplant the presently employed event-triggered controller area network (CAN) [3]. In particular, for advanced distributed control applications ("by-wire systems") an industrial consortium of leading automotive and electronic manufacturers established a very promising protocol termed FlexRay, cf. [4].

While enabling even more complex applications, the time triggered approach itself requires a higher effort during the design phase and introduces numerous new configuration parameters. Moreover, compared to traditional protocols FlexRay is very young, and hence field experiences are scarcely available. This creates an unprecedented challenge with respect to quality assurance, and an innovative, systematic approach for testing and debugging on the system level is urgently needed for application development and maintenance.

In this paper (i) we present an approach that facilitates a partitioning of the test problem of a time-triggered communication subsystem – and thus a systematic treatment – while still retaining the system-test scope. This allows a considerable reduction of test complexity. (ii) we stress the option of transparent remote testing that makes our approach even suitable for demanding test purposes. The key here is the careful exploitation of the clock synchronization service as a loop-back for diagnostic information. We present some use cases to demonstrate the practical applicability of our

E. Armengaud and A. Steininger are with the Vienna University of Technology, Treitlstr. 3, A-1040 Vienna, {armengaud, steininger}@ecs.tuwien.ac.at.

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

This work has been supported within the FIT-IT research projects STEACS (807146), ExTracT (810834), and the FHplus project DECS (811414) all managed by the Austrian Research Agency FFG.

approach.

II. FlexRay Basics

A FlexRay based system consists of nodes ("*self-contained computers with their own hardware and software*" [5] p.75) which cooperate to deliver services for their environment. Each node comprises a communication controller that is in charge of sending and receiving data over a communication network via two channels that can be used in a redundant or non-redundant fashion; see [4] for details. FlexRay supports different bus topologies including linear bus or star topologies, and hybrids thereof, cf. Fig. 1.

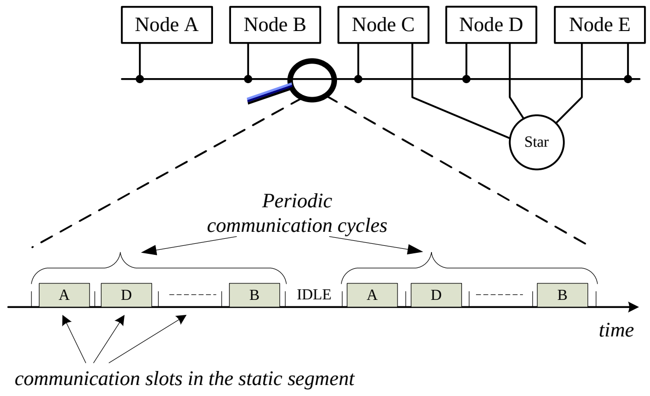
Node A
Node B
Node C
Node D
Node E
Star
Periodic communication cycles
A
D
B
IDLE
A
D
B
time
communication slots in the static segment

Fig. 1. Example Architecture of a FlexRay System

Time-triggered communication protocols such as FlexRay implement a Time Division Multiple Access (TDMA) scheme and divide the bus access into periodic, a-priori defined time windows which are statically assigned among the nodes for message transmission. This yields bounded and deterministic communication delays independent of the traffic load. In addition to the mandatory static segment the FlexRay protocol also implements an optional dynamic segment to allow for more flexibility. In this segment, however, bounds for communication delays cannot be guaranteed anymore. The dynamic part is intended for communication that is not safety critical.

Evidently, the establishment of a global time base is of utmost importance for time-triggered systems. This global time base is required, among other things, for collision free communication in the above mentioned TDMA scheme and for obtaining a consistent view of the environment among the different nodes. However, the nodes are self-contained and have their own (imperfect) quartz. Consequently, periodic re-synchronization is required to correct the different deviations and establish a system-wide synchronized global time base. A distributed clock synchronization algorithm serves this purpose. An instance of this algorithm is executed on every node, where it determines the difference between the scheduled and the actual reception time of so called "sync messages". Based on the mismatch observed for these sync messages it computes a correction for its local time according to the fault tolerant midpoint algorithm [6].

Like other protocols FlexRay provides CRC protection for the messages, and diverse additional syntax checks are performed on received frames that can be used to identify an erroneous message (i.e. one sent by an erroneous sender or one mutilated during transmission). In contrast to most other protocols, however, the time triggered nature of FlexRay allows performing checks in the temporal domain as well. In particular, a message is only regarded valid by a receiver, if it arrives within a defined window around the expected arrival time (defined by the global schedule and the respective receiver's local notion of time), and if it fits within the sender's assigned time slot in the TDMA schedule. This accurate interface definition both in the time and in the value domain enables *composability*, i.e. existing services are not disturbed by the introduction of a new one [7].

In principle, testing of time-triggered systems benefits from so called temporal firewalls [8]. They split the distributed system into multiple fault containment regions that can be tested independently in isolation, e.g., using node tests and a test of the communication subsystem, respectively. On top of that, however, a system-level test plays an important role. One reason is that FlexRay features more than 90 configuration parameters that need to be configured properly for every node, which makes a systematic approach for a system test mandatory to cover the huge test space. Further arguments for the need of a system test will be given below.

III. Elaboration of Requirements

A. Terminology

Testability: The quality of a test result attainable with given efforts strongly depends on the testability of the system under test (SUT). The two key aspects in this respect namely (i) *controllability* and (ii) *observability* are largely determined by the *accessibility* of the SUT. In short, good accessibility leads to good testability.

Remote Testing/Monitoring: A test is called remote if it does not require direct physical access to test points on the target. Instead, a communication path serves as the only interface for the tester to control and observe the SUT. As the name implies, a remote tester may be located distant from the SUT.

Monitoring vs. Testing: The normal test procedure is a stimulus/response measurement: Well chosen test vectors are applied to the SUT to exercise all relevant functions. The response is observed and compared to a known good reference [9], [10]. This approach by its nature causes an interference with a potentially ongoing operation of the SUT. However, under certain conditions the normal system operation can be considered a sufficient set of test vectors, and hence the requirement to explicitly apply stimuli can be relaxed. It is then sufficient to monitor the ongoing system operation instead and check whether it meets the expectations [11]. Usually, without explicit stimulation some exceptional states (e.g., emergency handling) are very likely not to be entered and the associated resources hence not to be exercised and tested. This limited test coverage may cause problems in the context with latent errors [12]. The major advantage of monitoring is that – since no active stimulation occurs – it is not necessarily intrusive. Furthermore, in time-triggered communication one can distinguish between a null-frame (when a node has no data to transmit) and a missing frame. Hence, a failing node or a broken link is easily identified.

On-line vs. Off-line Testing: In the classical sense the test is a very specific mode of operation: The SUT is taken off-line and stepped through a sequence of states to thoroughly execute all relevant functions. The theory for this off-line test approach is well developed and these tests usually achieve good coverage, since having complete control over the system facilitates good testability. On-line testing, in contrast, is transparently performed while the SUT is still providing its service [13]. The main advantages of continuous on-line testing are its very fast detection of errors – as compared with off-line tests that are usually performed after long periods of (untested) system operation only – and the continuous collection of a comprehensive and representative test record for statistical assessment and projections. Such a record may be useful for preventive maintenance [16].

On-line testing needs to be transparent for the application (non intrusive both in the time and in the value domain), which is a benefit as such but makes it extremely hard to implement.

B. Purposes of testing

According to [14] "*testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results*". Depending on the current stage in the life cycle of the system this common definition can be applied to testing of distributed automotive real-time systems with different goals in mind:

- Verification (correctness wrt. certain properties [15])
- Conformance test (correctness wrt. a given standard)
- Robustness test (operate under stressful conditions [16])
- Inter-operability test (nodes from different vendors [17])
- Performance test (benchmarking [18])
- Identification of configuration parameters [19]
- Maintenance test (find ageing defects [15])

Obviously these different testing aims span a large scope. During protocol and application development experienced engineers are using the test tool in a lab environment for debugging a prototype or even a simulation model, and flexibility is the key issue. In context with maintenance testing, on the other hand, repairpersons with comparatively low protocol expertise are using it in a rough environment with limited access, and hence standardization and ease of use are most crucial. Verification, conformance, inter-operability, and maintenance tests are qualitative tests with the purpose of proving whether an assumption is correct or wrong, whereas parameter identification as well as robustness and performance tests are quantitative tests that aim at deriving a numerical characterization for a given attribute.

C. Starting Point and Scope of Work

Monitoring of distributed embedded systems is frequently used for debugging, logging and other analysis on the data level; see [20], [21], [22] for some recent advances. Commercial tools for the analysis and diagnosis of the most popular automotive or real-time communication protocols are available. Examples are CANalyzer[1] for the CAN protocol, TTView[2] for the TTP protocol, or the BusDoctor[3] and the CANalyzer expansion for the FlexRay protocol. Implementation issues of these and similar tools along with some use-case scenarios can be found, e.g., in [23], [24].

All these solutions, however, only enable the monitoring of the bus traffic on top of the data link or higher layers employing COTS network controllers and device drivers in a promiscuous mode (if available) where even corrupt frames are forwarded to the processing CPU (plus some error flags). For the systematic, in-depth testing of a FlexRay based communication subsystem this needs to be complemented by bus monitoring and dedicated measurements on lower abstraction levels, as well as an appropriate data analysis.

In addition to these monitoring approaches, a mechanism in the reverse direction that allows some kind of stimulus generation and injection or replay is required. In principle, any fault injection tool serves this purpose; see [25] for a survey. Obviously, however, some protocol specific support is essential for performing efficient, well-aimed experiments that can be triggered by events on diverse protocol layers, e.g., and that also facilitates the generation of bus traffic with specific properties. Implementations in this regard are provided for existing automotive bus systems like LIN or CAN. Unfortunately, the concepts underlying these tools for event-triggered protocols can hardly be applied to the time-triggered case: The existence of a global time, the static schedule, and the high number of configuration parameters establish a completely different test scenario that requires a substantially different solution. The Disturbance Node for TTP/C represents such a solution for a time-triggered protocol. With its particular strength for injecting physical faults like bus noise, short circuits, delays, etc., its focus, however, is on the physical communication bus itself rather than the system.

AUTOSAR (AUTomotive Open System ARchitecture [26]) defines an open and standardized software architecture for vehicle applications. This platform includes two main error handling mechanisms: The *Development Error Tracer* is targeted for integration support and reports node internal error messages. The *Diagnostic Event Manager* enables the tracing of status and errors during field operation. These mechanisms present three main differences to our approach. First, they require explicit integration within the application and are thus application specific. Second, they require target node internal resources (CPU time and memory). Third, the gathered data is focused to high level information, and detailed information about the communication status is difficult to obtain.

For a system test of a distributed, time-triggered communication subsystem a tester approach is required that tightly integrates stimulation and observation to facilitate a well controlled stimulus/response measurement. At the same time we need maximum support for protocol related issues, and we need access to different protocol layers, both for monitoring and for stimulation. While suitable methods for testing of the

[1]http://www.vector-informatik.com
[2]http://www.ttautomotive.com
[3]http://www.decomsys.com

computing nodes themselves on the one side and the bus on the other side do exist, a unified, accurate and systematic test approach on the system level is required that does not only consider the function of these singular components in isolation. Experience 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 product variants. It is therefore the primary aim of our work to develop a modular test approach that is powerful enough to facilitate an efficient, well targeted check of system attributes and system behavior, such that the properties and services of interest for a given test purpose can be systematically investigated. Although, in principle, our approach shall be suitable for all test purposes mentioned above, we will focus the further presentation to those cases that pose specific problems due to limited accessibility and/or due to the need to perform the test on-line.

D. Preliminaries and requirements

To serve the various purposes listed above, an ideal tester for a distributed embedded system based on FlexRay has the following properties regarding its practical employment:

Remote Accessibility: For tests late in the product development cycle (conformance, maintenance) test points on individual nodes within the SUT are rarely accessible since they are deeply embedded and sealed. This makes a remote test approach via the communication network attractive.

Non-intrusiveness: The inter-operation of the nodes in a distributed system often suffers from very subtle effects, and this is where a probe effect is most troublesome. In this situation on-line monitoring suggests itself.

No explicit test support: It would be quite unrealistic to assume that specific "test hooks" will be appended to an established protocol standard, or that series applications will take care for providing support for our tester (like a shadow bus for diagnostic information). Therefore a guiding principle in our concept will be to come along without any supportive provisions or architectural changes in the SUT.

At this point it becomes obvious that it will not be possible to achieve all these contradicting goals at the same timewith a single, monolithic tool, a bundle of complementary techniques will rather be required that can be combined to serve a given purpose.

Throughout this paper we assume the tester to be fault-free. Since it is a somewhat unique component we can allow higher cost and efforts for the tester, and an extensive self-test can be performed before putting it into operation.

IV. Problem Decomposition

It is well known that test complexity rises by far more than linearly with the complexity of the SUT. The usual approach to combat this effect is "divide and conquer", i.e., partition the system into small portions that are relatively easy to test. In fact the ability to easily decompose a system into parts with well defined behavior is one of the major merits of the time-triggered approach (see Section II).

For a system test of a distributed, time-triggered communication subsystem, however, it is necessary to perform a test of the assembled system even though all components may have been tested in isolation already. Here, of course, a decomposition of the system into single components is counterproductive. Performing an unstructured functional test of the configured system, on the other hand, results in excessive test duration and poor coverage (this is essentially the problem system integrators are suffering from today). Thus, some different kind of test structuring is mandatory.

Rather than structuring the *physical* system into components we propose a two-step strategy instead:

1) Identify and separate all basic (i.e. top level) *services* that are expected from the SUT. In case of our communication system these are the sending and the reception of messages as well as clock synchronization.
2) Decompose each of these services into (ideally) independent functions which we call *mechanisms* M_k. Ideally, a mechanism has one single information input, one single information output and (maybe) a status output. Its operation is entirely described by a simple model in a generic way, and one or several *attributes* can be used to characterize it (see Figure 2). Attributes can either be protocol configuration parameters (e.g., number of static slots) or protocol constants (e.g., maximum clock frequency deviation). Consider the CRC generation in the transmit service as an example: Its input is the payload data, its output the associated CRC, and its functional model a linear feedback shift register implementing a given polynomial that is typically a protocol constant.

Note that these services and mechanisms are not necessarily bound to one physical node or component but may rather be distributed (like in the case of clock synchronization), and it is precisely these services (not the physical components) we want to test.

The proposed decomposition substantially simplifies the test problem (namely to a test of simple mechanisms), while still retaining the system level view. Given the functional model of a mechanism's operation, we can project our system test to a check whether all attributes are in the allowed range. The test of an error detection mechanism, in turn, implies observing the reaction of the system to a message that has been generated with the associated attribute(s) being erroneous. The decomposition can be performed in a systematic manner, which aids in achieving a complete picture, and facilitates a systematic exploration of the fault space along orthogonal single mechanism failures ("basic faults"). Note that the granularity of the fault model chiefly depends on the decomposition granularity. Physical faults that are not directly reflected by one basic fault can be mapped to unique combinations thereof ("syndromes"), thus aiding in the generation of a fault dictionary.

Moreover, the global picture of how a given service is composed by individual mechanisms makes the interrelations between the involved mechanisms explicit and can hence substantially simplify diagnosis and an inspection of fault propagation, cf. [27]. If potential error signals issued by a mechanism (as a status output, e.g.) are included in this global

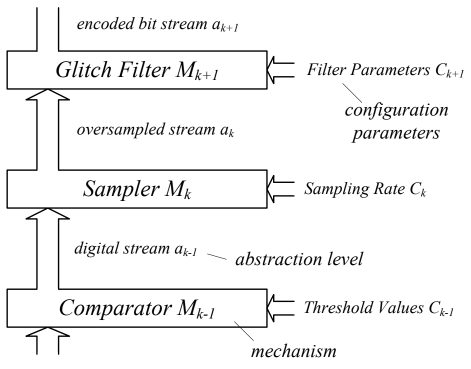

Fig. 2. Some Examples of Mechanisms

model, a hierarchy of error signals can be constructed that further eases diagnosis.

Our approach is somewhat similar to the OSI layer model, however, finer-grained. In [28] we have illustrated the complete model for the FlexRay communication services. The following key property of this model shall be highlighted that will become important later on: Mechanisms can be hierarchically ordered in levels, such that a high-level mechanism M_k builds upon the functionalities provided by the lower-layer mechanisms $M_{k-1} \dots M_1$. Therefore the function of mechanism M_k does not suffer from an erroneous behavior or incorrect configuration of a higher-layer mechanism M_{k+i}, while it does suffer from an error of a lower layer mechanism M_{k-i}.

Imagine the example of the channel decoder mechanism from the receive service here: Its output is expected to provide a (e.g. Manchester-) decoded bit stream representation of the message received on the physical line. Clearly this output will still be correct even if the CRC check mechanism that further processes this output should fail, while it will be incorrect if the sampling mechanism that supplies the decoder's input is configured with an erroneous clock rate, for example.

This property allows us to check the mechanisms one by one, starting with the lowest level mechanism and successively increasing the level. Thereby every step builds upon the results of the previous one(s).

Let us investigate how we can apply our structuring approach to the test of the FlexRay communication system. We have already identified 3 services, namely transmission of messages, reception of messages and clock synchronization. The nodes' transmit services can most easily be tested, since a tester that is connected to the bus, only needs to passively observe the traffic and draw the appropriate conclusions. Since the transmission service is composed of a linear chain of mechanisms, all mechanisms are sufficiently exercised during normal operation, such that plain monitoring is indeed a sufficient test here. The identification of a faulty node is easily possible due to the static TDMA schedule. Diagnosis with respect to the faulty mechanism can be based on the hierarchical structure of the mechanisms outlined above: Should, e.g., the sender's CRC generation mechanism be faulty (or incorrectly configured, in case of another mechanism), then all mechanisms below (encoding etc.) will still operate properly such that a message containing the incorrect CRC will finally be transmitted. The tester will receive this message, decode it, strip all of the framing information, etc., until it comes to CRC decoding. Here an error will be signalled. Assuming a fault free tester (and physical line) this error can be directly projected to the sender's CRC encoding mechanism. This correspondence between the sender's and the tester's abstraction levels is illustrated in Figure 3.

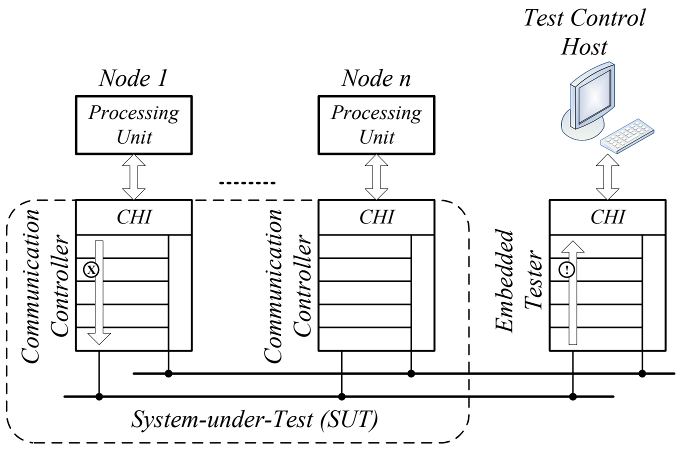

Fig. 3. Remote Testing

Testing the receive service, however, requires additional provisions. The first problem is that the operation of the receive service cannot be directly observed on the FlexRay bus. We have two options here: (a) Directly access diagnostic information on the nodes of the SUT. This is a very powerful approach but obviously inhibits a remote test. (b) Draw conclusions from the node's behavior that can be observed on the FlexRay bus. Here we can exploit the fact that a node that does not receive messages on the bus will run out of sync, since clock synchronization is based on the reception of (valid) messages. How to leverage the clock synchronization service as a virtual, transparent channel for such meta-information (and how to finally test it) will be treated in more detail in section V-B.

The second problem is that the receive service includes many error detection mechanisms that are not regularly executed during normal operation, i.e. the mechanisms are not linearly aligned within the service, but there are branches. Active stimulation will be required to exercise all these branches. This issue will also be treated below.

V. Implementation Concepts

A. Systematic test approach

With the layer approach we can decompose the entire communication system into a collection of services and mechanisms, and the remaining task is to check the values of the attributes that characterize every mechanism of interest. For this purpose we need access to its inputs and outputs as well as to the configuration interface for monitoring and – in some cases – for control. Beyond just this mere physical access, an elaborate strategy of how to interpret the observed information and how to prepare a stimulation is mandatory. Here our concept relies on generic building blocks as illustrated in Figure 4 for one abstraction layer. The same approach can be reused for every other abstraction layer within the communication system and, in principle, can be targeted effortlessly to other bus-protocols as well.

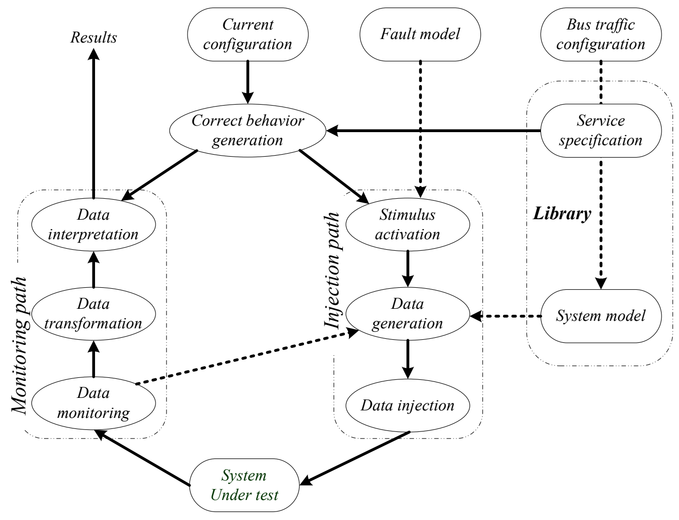

Fig. 4. Test Approach for one Layer

In this model the monitoring path aims at providing means to observe ongoing system operation, (automatically) extract the attributes of interest and evaluate their correctness (e.g. whether they are within the expected limits). The *data monitoring* module processes the received data up to the abstraction layer where the selected attributes can be best analyzed. Therefore, this module takes as input the bus traffic at the physical layer and de-encapsulates all data up to the particular layer by removing information used by the lower layers solely. The *data transformation* module takes the de-encapsulated information and reduces its dimensionality by extracting information relevant for processing the attribute values and for relating it to its source (node). The *data interpretation* module interprets the monitored attribute values and thus the according system behavior by comparing it with reference values. For more details see [29], [30].

The evaluation result may be either a match/no-match (useful for conformance tests) or the distance from the observed towards the expected reference values (useful for robustness tests). As an example consider the observation of a specific transmitter's bit width. The tester can analyze the received frame with respect to this attribute (using a precision oscillator in this case, of course) and check whether it is within the specified bounds.

Furthermore, due to the high data compression attained by the attribute extraction, one can easily record a result log for attributes over time, thus gaining insight to the evolution of attribute values. This feature is especially useful during system operation to obtain information for preventive maintenance, e.g. to identify the weakness of certain components. In the above example such a log might reveal an abnormal evolution of the bit width – even if it is still within bounds – that might point to a sneaking defect of the node's crystal oscillator or to thermal problems. Fixing is possible then even before a failure has actually occurred.

Another typical application of the monitoring path is for automatic configuration parameter identification, see also [19].

As outlined in Section IV monitoring may be sufficient for testing the transmit service, but active stimulation by the tester is definitely required for testing the receive service and the clock synchronization. More specifically our tester must be capable of generating traffic on the FlexRay bus in a well controlled manner. Therefore we have included a stimulation path in our concept. This path is shown in the center of Figure 4.

Depending on the purpose one can choose between two options for stimulus generation. One is to simply replay a previously recorded bus traffic, which presents the advantage of producing representative bus traffic, however, requires an operational system prototype and does not necessarily conform to the specification. This approach is well suited for robustness tests where good representativeness plays an important role and a running system is usually at hand anyway. In contrast, a "perfect" bus traffic can be generated from a reference model. This approach does not require a running cluster and provides bus traffic conform to the specification, which is more appropriate, e.g., for conformance tests. While additional efforts are required to design the reference model (*system model* in Figure 4), this second method typically allows more freedom in generating suitable bus traffic.

In either case the *data generation* module is responsible for (optionally) modifying attributes within this normal bus traffic according to a *fault model* and under control of the *stimulus activation* module. Such a modification may be desirable for exercising error detection mechanisms. Eventually, the *data injection* module processes the resulting bus traffic and transforms it into an appropriate representation to finally push it onto the wires with the appropriate timing.

A practical example here is the generation of frames with a temporal offset relative to their ideal position within the assigned slot. Such an experiment may be useful to test the receivers' tolerance with respect to such a shift in the scope of a robustness test.

Both previously discussed data generation approaches can be applied either on-line or off-line. While the on-line approach allows a study of the "live" behavior of the system, it is quite demanding for the tester's real time performance. A possible implementation of on-line modification is an active star capable of modifying data instead of simply passing it through. For more details see [29], [30].

B. Making the remote test transparent

So far our implementation concept has assumed accessibility of any desired point within the SUT. As outlined in Section III this may be a valid assumption during system development and debugging, but for purposes later in the life cycle a remote access is desirable. We have already motivated in Section IV that even straightforward remote monitoring of the network traffic can yield an appreciable amount of diagnostic information with respect to the transmit services of the individual nodes within the SUT, if it is supported by a careful interpretation that incorporates detailed protocol know-how. In contrast, remote diagnosis of the receive services is much more difficult, since neither its output nor its error status can be directly observed on the communication bus The most attractive solution here is to exploit the existence of an inherent loop-back that is established by the clock synchronization service as illustrated in Figure 5:

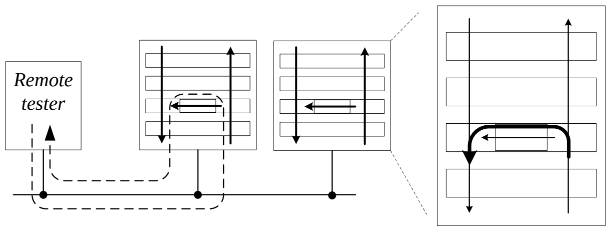

Fig. 5. Loop-back via the Clock Synchronization Service

The precise alignment of an outgoing frame within its associated TDMA slot is influenced by the sending node's local notion of time, which, in turn is determined by the clock synchronization service. As outlined in Section II this service evaluates the temporal position of all valid incoming messages, which closes the loop between sending and receiving: In principle it makes a difference on the temporal alignment of a node's outgoing frame, whether or not it has received a certain message during the previous round. This loop-back is very elegant and generic since it reuses existing standard mechanisms. Therefore, no additional service (and resource) is required, and this method can be applied with every FlexRay compliant system. There are several approaches to make use of this loop-back:

(a) The first approach is based on the observation that a node failing to receive valid messages will run out of synchronization. Seen in the reverse direction, the fact that a node stays synchronized with the others implies that it must have received (at least some) valid messages recently. Note that this exploitation of basic protocol properties facilitates a first assessment of the receive services by monitoring alone, i.e. remotely and completely non-intrusive.

(b) We can go one step further and have our tester send messages that will – if properly received – impact the global time in an a-priori known way (e.g. causing a speed-up). Note that adapting the global time to a faster or slower node is a normal procedure within the clock synchronization service, hence, such an impact does not necessarily represent a protocol violation. More specifically, we have shown that our test approach can be mapped to standard nodes with correlated, specification conform quartz drifts [31]. Since we are able to forecast the quartz drift a standard node would have to present the same output as our tester, we precisely know wether the test operation violates the quartz specification or not. On every node that receives the tester frame, the clock correction mechanism will exhibit the expected impact on the local time, whereas on those nodes that did not receive the tester frames or discarded it as invalid the impact will not occur. As outlined above, any variation of a node's local time is reflected by the position of its outgoing frames and can therefore be observed by a monitoring tool on the FlexRay bus. The tester can thus distinguish whether its stimulus has been received by a particular node.

Note that this is also a completely remote test procedure. It is intrusive in that the tester occupies time slots (sync messages to be more precise) in the TDMA schedule. The payload of the tester frames will be ignored by the application and is therefore irrelevant – it is just the framing that serves the purpose of testing the receive services. Therefore, the intrusiveness is very limited.

We have indeed been able to implement this loop-back and shown that this principle actually works in practice (in spite of noise and jitter). This kind of stimulation is a very powerful aid for testing error detection capabilities, e.g.: The tester can send frames that violate certain properties and then check whether these frames are actually discarded by a node. In a similar manner the tolerance intervals for certain attributes can be explored. Furthermore, this approach can be extended to having the tester send a sequence of stimuli in such a way that it forces the nodes in the SUT to follow a non-conventional but still valid progression of global time. This allows an assessment of attributes related to the clock synchronization service, e.g. (maximum tolerable rate correction, etc.). This issue will be further pursued in Section VI. Another possible application of our test approach is to perform on-line remote quartz measurement, see [32] for more information.

Considering the stringent constraints associated with remote testing the above approaches attain an appreciable coverage. Obviously, however, their coverage is not as good as in the unrestricted case. In particular, services situated above the clock synchronization in the layer model are not included in the loop-back and hence not covered.

VI. Use Cases

A. Test Environment

One issue while developing an application is the capacity to reproduce a scenario that led to a fault in the system. Herein, recording bus traffic and deterministic replay enable efficient *cyclic debugging*. Using these options it is possible to repeatedly re-execute a recorded scenario in order to track down the errors after a fault has been detected.

Likewise, the capacity to generate and/or alter bus traffic allows to *create new scenarios*. This on one hand enables the emulation of services that are not available at this point of development and on the other hand fault injection permits the validation of error detection/correction mechanisms. Furthermore, fault forecasting [15] for the purpose of robustness evaluation can be achieved, too. More generally, the generation of test scenarios and the capability to deterministically apply them to a system is required for every kind of validation and benchmarks.

To achieve these aims, our tester node is complemented with off-line software tools to generate different test scenarios. Our tester implementation presents three access points namely (1) at the *digital level* (serial bit stream), (2) at the *packet level* (FlexRay frames without checks of the timing information), and (3) at the *frame level* (FlexRay frames after timing checks). Concerning the software part of the tester, a bus traffic generation tool permits the generation of bus traffic from scratch. Additionally, a bus traffic modification tool allows the insertion of deviations by manipulating an existing bus traffic logfile. These deviations can model both disturbances from the environment as well as service deviations resulting from transient or permanent failures inside the communication controller, see [33] for details. An important feature of these software tools is the capability to process the data stream at different abstraction levels. Hence, the *physical access* (see Figure 4) is performed by the tester hardware while the *filtering* can be performed both by the hardware or software. The *transformation* (modification of the stream for fault injection) is performed by our software tools.

The interface between the on-line tester and the off-line tools is realized with the help of bus traffic logfiles, which completely describe the bus traffic behavior. In particular the communication scenario is divided into packets, each consisting of an identifier, a packet length descriptor, a time stamp, and the actual frame described at a defined abstraction level. This encapsulated format not only provides flexibility for further development, but builds a bridge between a simulation and hardware prototype environment. Both simulation and hardware prototyping complement each other for debugging purposes in order to track down errors to their roots.

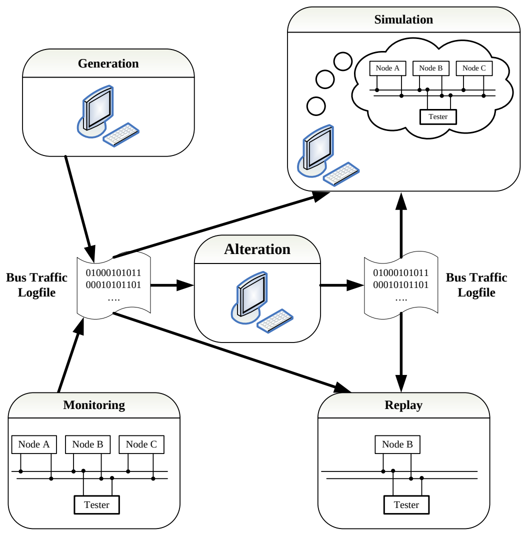

Fig. 6. Test Flow

Figure 6 shows a flow-chart for the possible test operations. First, a scenario is generated using bus monitoring or via the help of the bus traffic generation tool. In either case the resulting scenario is stored in a logfile that can be directly used for replay (in a real-world cluster or in a suitable simulator) or further altered. The aim of the alteration is to insert deviations into the bus traffic, in order to either explicitly inject failures (e.g. a wrong CRC) thus testing the fault detection mechanisms, or to test the system's tolerance margins (e.g. by shifting a frame towards the slot boundaries).

B. *Testing error detection mechanisms of the receive service*

In a first application we used this test environment to systematically test the error detection mechanisms of our node based tester prototype. Syntax errors were introduced by a tester node A at the digital level while boundary violations and content error were injected at the packet level. Table I summarizes the error flags as they were recorded in a test campaign using standard FlexRay controllers (column "FlexRay status") and in comparison with a finer differentiation with a different tester node B (column "Tester status"). Herein, the syntax errors were injected at the digital level while the boundary violations and content error were injected at the packet level, cf. Figure 4. All resulting errors have been successfully detected, therefore the detection coverage is 100% in this experiment.

TABLE I
TEST CAMPAIGN SUMMARY

# of deviations	# of cycles	FlexRay status	Tester status
234	5000	vSS!SyntaxError	CODERR
234	5000	vSS!SyntaxError	TSSVIOL
234	5000	vSS!SyntaxError	HCRCERR
234	5000	vSS!SyntaxError	FCRCERR
234	5000	vSS!SyntaxError	FESERR
1092	5000	vSS!BViolation	BVIOL
468	5000	vSS!BViolation	BVIOL
46	3000	vSS!BViolation	SWVIOL
46	3000	vSS!BViolation	NITVIOL
78	5000	vSS!BViolation	SOVERR
78	5000	vSS!ContentError	NERR
156	5000	vSS!ContentError	SSERR
78	5000	vSS!ContentError	FIDERR
46	3000	vSS!ContentError	CCERR
234	5000	vSS!ContentError	SPLERR

From Table I it is obvious that our tester reveals a much more detailed status information than a standard node.

C. *Testing the clock synchronization service*

The aim of this second use case is to test the distributed clock synchronization algorithm. Since this mechanism can not be tested in isolation (partitioning), methods to focus on the service itself (layering) as presented in this work are strongly required. The main advantage of our test concept is the capability for the user to directly access this mechanism. The other (lower) layers involved are automatically emulated by our test environment. During this test campaign, we applied a deterministic replay and varied the maximum rate- and step correction values until the nodes in the SUT switched to silent mode. For both experiments our tester generated two messages (ID 13 and ID 14) with the SYNC bit set, thus, dictating the clock synchronization in our setting.

For the rate correction experiment the tester linearly increased the rate of its logical clock. Concerning the step correction experiment we modified after about 300 communication cycles the step correction value of the tester by increasing it by $\delta_s = 25$ns (one microtick: nominal period of our internal quartz). Then we left this value constant for

40 communication cycles and set it back to the original value for the following 40 communication cycles. Afterwards we applied a step correction value increased by additional 25ns (i.e. $\delta_s = 50$ns) for the following 40 communication cycles before we set this value back once more. This procedure was applied periodically resulting in a stimulus as depicted in Fig. 7 (upper plot). At the same time we monitored the cycle length (the number of microticks after clock correction applied to the local node quartz) of the SUT nodes. These values followed the stimulus – as can be seen in Fig. 7 (lower plot) – up to a logical clock state deviation of $\delta_s = 450$ns. Afterwards, they were no longer able to follow suite; the nodes turned silent.

TABLE II
RESULTS OF THE CLOCK SYNCHRONIZATION EXPERIMENTS

Parameter	Results	Specification
max. rate correction	$4.5\mu s$	$4.5\mu s$
max. step correction	450 ns	500 ns

Table II summarizes our findings for both, rate correction and step correction. Herein the values given in the "Specification" column reflect the theoretical values from the FlexRay specification for the configuration of *pRateCorrectionOut* with 180μT and *pOffsetCorrectionOut* with 20μT (μT stands for microtick). As expected, the nodes of the SUT turned silent when the shift for the rate correction exceeded 4.5μs. For the step correction a value of $500ns$ was expected according to the specification, while our measurement yielded $450ns$. We have traced back this deviation to a correction "overshoot" that results from the need for a punctual correction of the accumulated clock state difference just after the immediate frequency jump dictated by our tester.

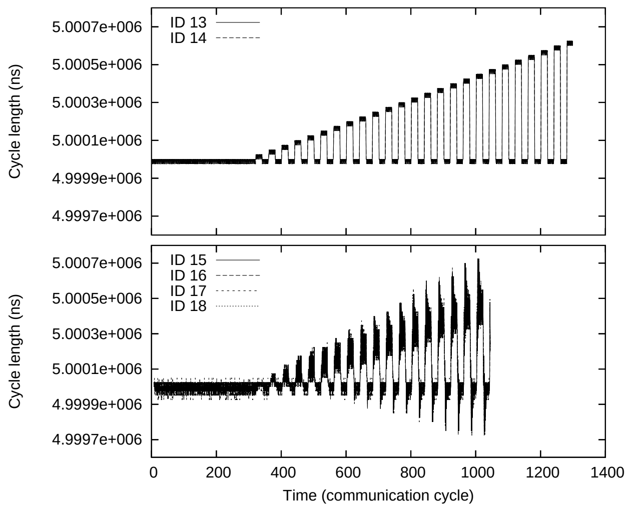

Fig. 7. Step Correction Experiment: Stimulus (upper plot) and Response (lower plot)

VII. CONCLUSION

Being the enabler for (cost) optimizations as well as new comfort and safety features distributed computer architectures will definitely play a central role in future car generations. We have motivated that system-level testing as well as testing the communication subsystem are crucial during development, integration and maintenance of these complex systems, and that systematic improvements to the current practice are urgently needed. With this motivation we have proposed a comprehensive test strategy for system-level testing of FlexRay based systems. Care has been taken to keep the approach flexible enough to be adapted to the partly contradicting needs of different test procedures during the life cycle of the system. This has been attained by providing a bundle of techniques that can be combined to fit the given purpose.

The foundation of our approach is a fine-grained layer model that allows horizontal decomposition of the system into services and mechanisms, but, in contrast to the usual component test approaches, without consideration of their physical allocation. The attributes that we introduced to characterize the function of the individual mechanisms are at the center of our stimulation and monitoring concepts: From the comprehensive stream of information collected on the FlexRay bus by monitoring, the attributes of interest are extracted, thus precisely reducing the information to the necessary. With respect to fault injection the attributes can be viewed as a fault space in which a systematic treatment becomes viable.

The bundle of methods we propose allows considering specific restrictions such as remote access or the avoidance of a probe effect. Our approach is based on (a) careful and detailed interpretation of the observed bus traffic, and (b) the exploitation of the clock synchronization service as an application transparent vehicle for stimulation and fault injection as well.

Our concept for stimulation allows both, the emulation of part of the system (faulty or non-faulty) as well as the introduction of faults into the bus traffic. The bus traffic required for emulation can be derived either from a bitstream actually observed on the bus or synthesized by virtue of a parameterized protocol model, and in both cases modifications can be applied for selected attributes. Depending on the purpose these procedures can be performed on-line or off-line.

Ongoing research is concerned with a refinement of the remote identification and check of the individual crystal oscillator frequencies of the nodes in the system by way of well-controlled stimulation of the clock synchronization service. Another research direction is the use of a star coupler test architecture to cover the important class of asymmetric faults.

REFERENCES

[1] K. Zeillinger, "Fehler im System," *auto touring, das ÖAMTC Magazin (in German)*, vol. 6, pp. 6–10, 2006.

[2] P. Hansen, "New S-Class Mercedes: Pioneering Electronics," *The Hansen Report on Automotive Electronics*, vol. 18, no. 8, pp. 1–2, Oct. 2005.

[3] "CAN Specification Version 2.0, available at http://www.semiconductors.bosch.de/pdf/can2spec.pdf," Robert Bosch GmbH, 1991.

[4] "Flexray Communications Systems – Protocol Specification Version 2.1, available at http://www.flexray.com," FlexRay Consortium, 2003.

[5] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[6] J. Lundelius-Welch and N. Lynch, “A New Fault-Tolerant Algorithm for Clock Synchronization,” *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[7] H. Kopetz and G. Bauer, “The Time-Triggered Architecture,” *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112 – 126, Jan. 2003.

[8] H. Kopetz and R. Nossal, “Temporal Firewalls in Large Distributed Real-Time Systems,” in *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, Oct. 1997, pp. 310–315.

[9] H. Thane and H. Hansson, “Towards Systematic Testing of Distributed Real-Time Systems,” in *Proceedings of the 20th IEEE Real-Time Systems Symposium*, December 1999, pp. 360–369.

[10] W. Schütz, “Fundamental Issues in Testing Distributed Real-Time Systems,” *Real Time System Journal*, vol. 7, no. 2, pp. 129–157, 1994.

[11] A. Steininger and C. Temple, “Economic Online Self-Test in the Time-Triggered Architecture,” in *Design and Test of Computers, IEEE*, vol. 16, July–Sept 1999, pp. 81–89.

[12] C. Scherrer and A. Steininger, “Dealing with Dormant Faults in an Embedded Fault-Tolerant Computer System,” in *IEEE Transaction on Reliability*, December 2003, pp. 512–522.

[13] M. Nicolaidis and Y. Zorian, “On-Line Testing for VLSI – A Compendium of Approaches,” *Journal of Electronic Testing: Theory and Applications*, vol. 12, no. 1-2, pp. 7–20, February 1998.

[14] W. Hetzel, *The Complete Guide to Software Testing, Second Edition*. Wiley, 1988.

[15] J.-C. Laprie and B. Randell, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004, fellow-Algirdas Avizienis and Senior Member-Carl Landwehr.

[16] A. Mukherjee and D. P. Siewiorek, “Measuring Software Dependability by Robustness Benchmarking,” *IEEE Transactions on Software Engineering*, vol. 23, no. 6, pp. 366–378, June 1997.

[17] S. Mosely, S. Randall, and A. Wiles, “Experience within ETSI of the Combined Roles of Conformance Testing and Interoperability Testing,” *The 3^{rd} Conference on Standardization and Innovation in Information Technology*, pp. 177–189, Oct 2003.

[18] J. Arlat, K. Kanoun, H. Madeira, J. Busquests, T. Jarboui, A. Johansson, and R. Linström, “State of the Art,” DBench project deliverables, Aug. 2001.

[19] E. Armengaud, A. Steininger, and M. Horauer, “Automatic Parameter Identification in FlexRay Based Automotive Communication Networks,” *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA’06)*, pp. 897–904, Sep. 2006.

[20] I. Smaili, “Real-Time Monitoring for the Time-Triggered Architecture,” Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[21] H. Thane, “Monitoring, Testing and Debugging of Distributed Real-Time Systems,” Ph.D. dissertation, Mälardalen Real-Time Research Centre (MRTC), Department of Computer Engineering, Mälardalen University (MDH), 2000.

[22] J. Tsai, Y.-D. Bi, S. Yang, and R. Smith, *Distributed Real Time System: Monitoring, Visualization, Debugging and Analysis*. New York, NY, USA: Wiley-Interscience, 1996.

[23] P. Peti, R. Obermaisser, W. Elmenreich, and T. Losert, “An Architecture supporting Monitoring and Configuration in Real-Time Smart Transducer Networks,” in *Proceedings of the IEEE Sensors 2002*, vol. 2, June 2002, pp. 1479–1484.

[24] M. S. Reorda and M. Violante, “On-Line Analysis and Perturbation of CAN Networks,” in *DFT ’04: Proceedings of the Defect and Fault Tolerance in VLSI Systems, 19th IEEE International Symposium on (DFT’04)*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 424–432.

[25] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, “Fault Injection Techniques and Tools,” *Computer*, vol. 30, no. 4, pp. 75–82, April 1997.

[26] H. Fennel, “Achievements and Exploitation of the AUTOSAR Development Partnership,” in *Convergence 2006*, October 2006, p. 10.

[27] M. Hiller, A. Jhumka, and N. Suri, “An Approach for Analysing the Propagation of Data Errors in Software,” in *DSN ’01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 161–172.

[28] E. Armengaud, A. Steininger, M. Horauer, and R. Pallierer, “A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems,” 5^{th} *IEEE International Workshop on Factory Communication Systems*, pp. 275–283, September 2004.

[29] E. Armengaud, A. Steininger, and M. Horauer, “Efficient Stimulus Generation for Remote Testing of Distributed Systems – The Flexray Example,” in *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA’05)*, L. L. Bello and T. Sauter, Eds., vol. 1, Catania, Italy, September 2005, pp. 763–770.

[30] E. Armengaud, F. Rothensteiner, A. Steininger, R. Pallierer, M. Horauer, and M. Zauner, “A Structured Approach for the Systematic Test of Embedded Automotive Communication Systems,” in *IEEE International Test Conference (ITC2005)*, Nov. 2005, pp. 1–8.

[31] E. Armengaud, M. Fuegger, and A. Steininger, “Safe Deterministic Replay for Stimulating the Clock Synchronization Algorithm in Time-Triggered Systems (to appear),” in 7^{th} *IEEE International Workshop on Factory Communication Systems (WFCS’08)*, Mai 2008.

[32] E. Armengaud, “Non-Intrusive Remote Oscillator Drift Measurement for Time-Triggered Systems,” Vienna University of Technology, Institue of Computer Engineering, Treitlstr. 3/3/182-1, 1040 Vienna, Austria (available at http://vmars.tuwien.ac.at/frame-papers.html), Tech. Rep. RR-62/2007, 2007.

[33] E. Armengaud, F. Rothensteiner, E. Schwaiger, and M. Zauner, “STEACS Test Environment,” Technical Report (Decomsys Confidential), 2005.