A structured approach for the systematic test of embedded automotive communication systems

Conference Paper · December 2005

DOI: 10.1109/TEST.2005.1583956 · Source: IEEE Xplore

Eric Armengaud
AVL LIST GMBH

Andreas Steininger
TU Wien

Martin Horauer
Fachhochschule Technikum Wien

A Structured Approach for the Systematic Test of Embedded Automotive Communication Systems

E. Armengaud[a], F. Rothensteiner[a], A. Steininger[a], Roman Pallierer[b], M. Horauer[c] and M. Zauner[c]

[a]Vienna University of Technology, ECS Group E182-2, Treitlstr. 3/2, 1040 Vienna, Austria
[b]DECOMSYS GmbH, Stumpergasse 48/28, 1060 Vienna, Austria
[c]University of Applied Sciences Technikum Wien, Höchstädtplatz 5, 1200 Vienna, Austria

Abstract

We present a systematic test strategy for the communication subsystem of a distributed automotive system. Key points are (1) system decomposition into layers and services and (2) integration of fault injection and monitoring within this framework.

1. Introduction

The automotive industry has introduced a wealth of new and emerging electronic features in recent years with a lot of innovations still to come, e.g. X-by-Wire technologies. Among the benefits of these new technologies are increased safety, more efficient resource usage, less environmental impacts and an increase of comfort [1]. At the same time these new approaches create a lot of new challenges. Considering that modern cars already contain up to 70 electronic control units and thus represent highly complex distributed architectures, one of these challenges is certainly to ensure the dependability required for the safety critical applications under the given tight cost constraints. Car manufacturers have soon identified the network architecture as the crucial point in this context. To that end, an industrial consortium has established the communication protocol FlexRay [2] that provides the flexibility to operate the communication system in a time- and/or event-triggered mode, respectively.

Despite all the wealth of properties to provide a reliable communication testing of such complex distributed architectures remains an issue. Even though we can rely on proven traditional test approaches for ensuring the functional integrity of all involved components (in isolation), their proper interoperation remains to be verified – in particular, with respect to the large number of product variants, configurations and their ever increasing complexity. Therefore, some means for testing are mandatory in order to detect faults and resulting errors and/or failures. A solution for an entire product lifecycle, i.e., during development, operation, and maintenance is desirable. Our STEACS[1] project addresses some of these problems by developing suitable concepts and prototype implementations for diagnosis and fault injection.

In this paper we describe our basic approach for a monitoring, replay and fault injection tool-suite that is applicable for system tests of distributed automotive networks. In addition, we detail some results obtained with our demonstrator system to show the applicability and benefits of our approach.

2. System Architecture

Nowadays, automobiles operate several bus-systems using different protocols each tailored to specific application needs. Figure 1 illustrates a possible automotive network consisting of several functional modules that are mapped to some node(s).

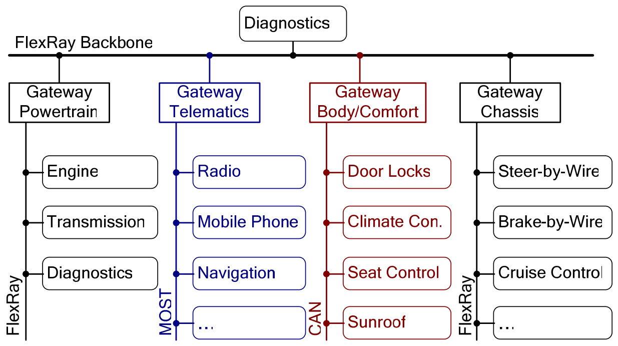

Figure 1: Typical automotive network with several bus domains

Testing and diagnosis of such distributed systems are confronted to three main problems, which are:

(1) Geographic diversity: Each node is a discrete physical entity distant to the other one. This results in reduced observability and controllability of the whole

[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 Research Promotion Agency (FFG) under grant 807146. See http://embsys.technikum-wien.at/steacs.html for further information.

system, aggravated by the desire for a purely remote test. Furthermore, communication across different bus domains must be considered, since the combination will likely be required for future applications, e.g. to combine navigation and/or speed with steering information.

(2) The large system state space: Every module can be modeled by an individual state machine; hence, the global system state is the product of all parallel state machines. As a result, testing all states of the system is an elaborate operation that requires the synchronization of the individual state machines. Due to the large number of product variants and configurations it is impossible to completely represent the system by a prototype; optional upgrades make the system structure even dynamic over life time.

(3) The large fault range: Faults may affect any function unit in a communication system and in a different failure behavior, e.g. omissions, timing failures, arbitrary failures, etc.

Concluding, system tests for communication networks in the automotive area are of utmost importance for revealing problems early in the design and implementation phase as well as during operation and maintenance. The huge variety of possible configurations makes it imperative to explicitly test the inter-operation of nodes within the assembled system rather than the individual nodes in isolation. In particular, a structured approach is mandatory to tackle the complexity that arises from the distributed nature of these systems.

3. Related Work

A standard procedure for testing is the decomposition of the test object into smaller sub-units [3] in order to reduce complexity and improve accessibility. Usually this decomposition implies a *physical* separation of sub-units. In distributed systems such a physical decomposition is indeed useful for ensuring that the individual nodes participating in the communication are working properly (testing the nodes in isolation). At the same time this approach is not sufficient to test distributed services provided by a collective of nodes. Therefore, a *logical* decomposition into layers is often proposed [4, 5].

Using a similar approach, we defined a layer structure for FlexRay communication systems, see [6]. Herein we decompose the complex (overall) communication service into single sub-services located on different layers.

In every test approach two fundamental sub-tasks can be identified, namely stimulation of the target and observation of its response.

In most existing approaches [4, 7, 8] observation is based on some form of packet filtering [9, 10] for the network traffic performed on an observer node. While this high-level approach is very flexible and does not require any dedicated hardware support, it does not provide sufficient diagnostic resolution for problems originating from hardware and lower level protocol services as expected in the automotive domain. Therefore, our tester is a standard node with enhanced capabilities (direct access to bus traffic on lower layers, e.g.). A similar solution has been used in [5] for a Profibus tester.

For complex communication systems a purely “artificial” generation of stimuli by a tester is elaborate and time-consuming. Therefore, it has become common practice to observe the target system during normal operation and “superpose” stimulation only to force it into desired execution paths and states [4, 8]. This is particularly necessary to include rarely exercised paths like error handling mechanisms in the test. The related techniques are commonly termed “fault injection”.

According to [11] fault injection approaches can be classified according to their aims: *Fault removal* is intended to check whether the fault tolerance mechanisms are designed and implemented properly, while the aim of *fault forecasting* is to determine the parameters required for the system’s reliability model. The focus of our project is definitely fault removal, which means that the selection of our faults must be aimed at high test coverage. With a fault distribution that is representative for the field conditions our tool can, however, support fault forecasting as well.

The main issue with fault injection into complex distributed systems is the huge fault space. Several approaches for the fault injection into a communication system have already been published (that essentially solve only part of the problem). One example is the Orchestra approach proposed in [4]. The aim there is to create a portable test tool that is applicable for a large variety of distributed systems. The insertion of a fault injection layer below the targeted protocol layer allows mutilating, inserting, or deleting messages on different abstraction levels. This approach, however, is purely software-based and hence does not allow investigating lower-layer services.

In [8] a versatile concept for dependability assessment in distributed systems is presented. The proposed tool NFTAPE can be viewed as a kind of control system for different distributed fault injection methods. Dedicated fault injector hardware and/or processes on the target allow various types of faults to be injected directly into the target. For our remote test approach, however, we do not want to modify the target. In fact, we map all possible types of node misbehavior for whatever reason to its bus behavior and hence inject faults to the bus traffic only. Given that the communication media is the only interconnection between the individual nodes this must be possible, although this restriction makes it harder to selectively target services in higher layers.

The Profibus tester framework proposed in [5] is based on a hardware solution similar to ours, but no discussion is given on how this tool is suitable for integration into a systematic test procedure. The Loki tool proposed in [7] is another (software based) fault injector for distributed systems; its focus, however, is the application level solely.

4. Layered Test Approach

4.1. Overview

Our approach is to connect a dedicated tester to the network of the bus domain under consideration; deliberately avoiding additional cabling to access the distributed nodes. Since the tester is a single dedicated node, one can afford a quite sophisticated design. This setup presents the following advantages:

- The intrusiveness of the system is kept low,
- the centralized and single tester still can eventually reach every node in its communication domain, and
- we can concentrate the required efforts to the tester node which allows us to afford probing deep into the communication services.

To tackle the test complexity, we first constructed a detailed layer model [6] of the FlexRay bus protocol – we choose FlexRay as our target communication system. Based on this layer model we structured our test approach using generic building blocks as illustrated in Figure 2 for one abstraction layer. The same approach can be re-used for every other abstraction layer within the communication system, and, in principle, can be targeted effortlessly to other bus-protocols as well. The key idea is to decompose the entire communication system into a collection of services that can each be entirely characterized by a set of *attributes*. Such attributes can either be protocol configuration parameters (e.g., duration of a static slot, payload length) or protocol constants (e.g., initialization value for the CRC calculation, etc.). Hence, 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.

In this model the monitoring path, resembling the *data monitoring*, the *data transformation* and the *data interpretation* modules, aims at providing means to observe and (automatically) extract the current system behavior. Important features of this path are the capability (i) to project the bus traffic monitored at the physical layer to any other abstraction level, (ii) to exhaustively test the services by way of testing their attributes, and (iii) to automatically evaluate the correctness of these attributes (e.g. whether they are within the expected limits).

The injection path, resembling the *stimulus activation*, *data generation* and *data injection* modules, provides means to (i) automatically select and modify one or a set of attribute(s) and (ii) transparently process bus traffic at any abstraction levels down to the physical layer. Consequently, any service deviation at a particular layer can be emulated, thus, gaining better controllability for the testing of the “remote” system under test.

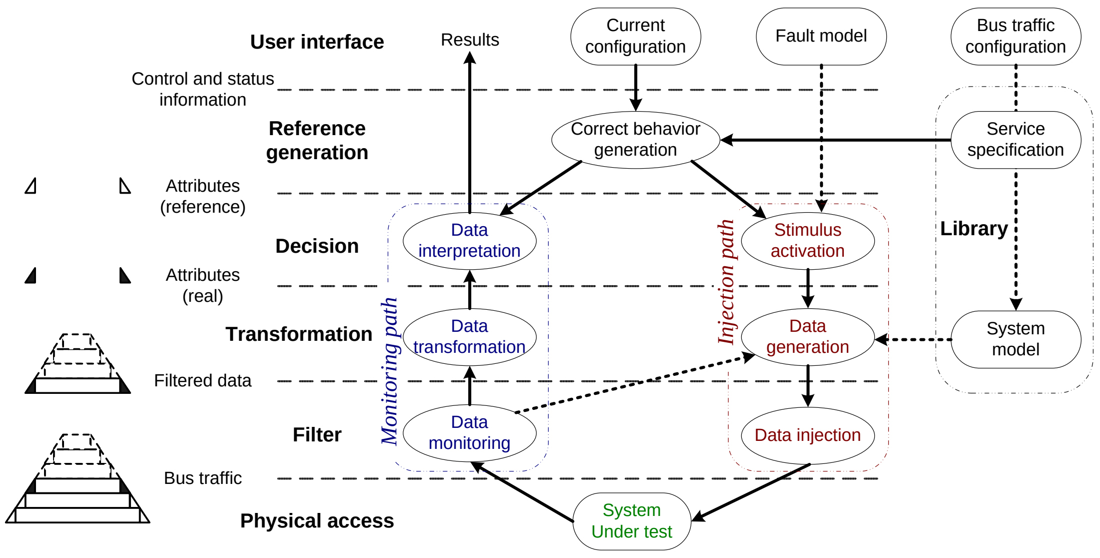

Figure 2: Proposed test approach illustrated for one layer

For either path reference values are provided by the *correct behavior generation* module. This reference behavior is built from data obtained on one hand by the *service specification library* and on the other hand by the *current configuration*. The specification of the system attributes (e.g. min/max values for every attribute) is constant over time and, consequently, can be linked as a library to the test environment. The configuration, however, is specific for the actual system setup and must be provided by the test engineer (e.g. the actual duration of a static slot). Finally, an optional *system model* can be used to generate a specification conform bus traffic and an optional *fault model* can be deployed to map a user defined fault model to the actual faults being injected.

4.2. The Monitoring Path

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 [12]. E.g., when the "*length of the static payload segment*" (*gPayloadLengthStatic* [2]) is the attribute of interest, the data monitoring module processes the information present at the physical layer, removes glitches, performs decoding, and presents the raw bit-stream to the data link layer. (For FlexRay frames the payload segment carries between 0 and 127 two-byte words of data. To indicate the size of the actual payload segment a separate payload length field is contained in the header segment.)

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). For the "*length of the payload segment*" attribute of a static transmission slot it removes the header and checksum, however, keeping the slot identifier and value of the payload length field. Next, it calculates the payload length from the remaining data, validates it with the payload length value, further removes the payload itself and, finally, forwards the obtained actual "*length of the payload segment*" value together with validity information and the slot identifier. The latter together with information from the system configuration allows the mapping of the attribute to a particular node.

The *data interpretation* module interprets the monitored attribute values and thus the according system behavior by comparing it with reference values. The result may be either a match/no-match (e.g. useful for conformance tests) or the distance from the observed towards the expected reference values (e.g. preferable for robustness tests). Furthermore, with this module 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 and modules.

4.3. The Injection Path

The *stimulus activation* module determines which and how attributes need to be modified. In particular, attributes are either selected based on information provided by a user specified fault model (i.e. for robustness tests) or all attributes relevant for the services are tested (i.e. for conformance tests). The valid range for every attribute is obtained from the correct behavior generation module, hence, the module can determine (with additional information from the user interface) how to manipulate the data in order to steer the attributes behavior.

The *data generation* module modifies a given "normal" bus traffic in order to emulate the malfunction of one (or several) services by altering its/their corresponding attributes for a defined interval. This data modification is performed at the location of the target service layer. It is accomplished by combining the information from the stimulus activation module either with recorded data obtained from the data monitoring module or a system model. Data obtained from a running cluster presents the advantage of producing representative test vectors, 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 [13]. While additional efforts are required to design the reference model, this second method typically allows more freedom in generating suitable bus traffic.

The *data injection* module takes as input the data file and performs all required processing down to the physical layer. Finally, it pushes the data onto the wires when the corresponding slots are present [12].

4.4. User Interface and Reference Generation

The user interface provides control of the entire test suite. It resembles a *library* module (*service specification*, *system model*) that models the system's expected behavior in accordance with the specification of the bus protocol and a set of configuration units (*current configuration*, *fault model* and *bus traffic configuration*). The latter is used on one hand to describe the current configuration of the cluster and on the other hand to describe the configuration of the test scenario. Furthermore, it presents the results to the test engineer for further post-processing.

The bus traffic configuration module provides the test engineer with an interface to enter and specify the actual data values that shall be sent (e.g. payload values, slot identifier, etc.). Next, the service specification module provides the specified ranges for all attributes and how they need to be instantiated. Thus, with the information from these two modules the system model is able to construct actual frames conforming to the specification.

Furthermore, the output of the service specification is used together with the current configuration as input to the correct behavior generation module to model the services according to their attributes and to generate a systematic and exhaustive list of rules defining the system behavior conforming to the protocol specification. As an alternative approach one might consider using a "golden reference node" as in [14], however, such references are rarely available at all.

4.5. Projection to Other Layers

The attributes of a protocol are associated with different levels of abstraction (layers); hence, the test approach illustrated in Figure 2 must be applied to all layers of the communication system at the tester node. In particular, all modules need be tailored for the specific layers, except for the data monitoring and data injection modules that must be able to handle the de-encapsulation and encapsulation up to/down from every layer.

Our tester is able to monitor and/or replay bus traffic as well as to inject faults. Failures can be identified by observing invalid attributes at different layers. In particular, our approach enables the simultaneous testing of both low level mechanisms, e.g. glitch filtering on the bus, and high level mechanisms, e.g. correct signal exchange of a distributed application. Furthermore, the identified failures can be mapped with some probability to faults originating from remote nodes or the physical network, see [4].

5. Use Cases and Experimental Results

5.1. Experiment Setup

In this section we will use two communication attributes as examples to illustrate the usefulness of our approach for a systematic test. For these experiments the system under test consisted of four COTS FlexRay nodes and our tester node connected via a linear bus. The test procedure was the same in both campaigns:

I. *Reference trace generation:* By means of the data monitoring module the bus traffic generated by a test application is traced and stored into a file.

II. *Attribute modification:* The trace file is changed by means of the data generation module in a way that the considered attribute is consistently modified.

III. *Fault injection:* The modified trace file is replayed to the system under test by way of the data injection module, and the succinct reaction is observed (i.e. whether the modification was tolerated or an error or a failure occurred).

5.2. Systematic Test of the "Byte Start Sequence"

According to [2], "the byte start sequence" (BSS) is used to provide bit stream timing information to the receiving devices. In particular, the BSS is defined as a specific vector of two consecutive bits – one high and one low.

The goal of these experiments is to systematically test the BSS mechanism with respect to this model. Such a test approach can be used, e.g., in conformance testing in order to prove that the error detection mechanisms are operating as intended [13].

Our experimental campaign covers three fault scenarios:

1. *Value domain:* Apply all possible erroneous values to the BSS. (3 options)
2. *Time domain:* Shorten/extend the length of the BSS vector[2]. (1 bit / 3bit)
3. *Error position:* Vary the position of the erroneous BSS within the frame. (n options for n byte frame)

For these three experiments, the current configuration and the service specification provide the correct behavior generation module with the exact definition of a correct BSS. According to this information, the stimulus activation decides which attribute has to be modified (BSS) and which fault model should be used. The data generation module then alters the BSS as previously specified, and the data injection module injects the corrupted frame to the system.

Node	**Reported Results**
System under Test Nodes	Status Error
Tester Node	Erroneous BSS attribute

Table 1: "Byte Start Sequence" experiment results for all three campaigns

Table 1 summarizes the results of these experiments. In all cases the nodes operated conform to the specification.

[2] Notice that the BSS service builds upon the services of the data link layer, hence, the handling of glitches has no relevance for the BSS service test.

During the execution of the presented "byte start sequence" experiment 20 faults were injected; the experiment was executed in hardware in a single run lasting 3 seconds. In comparison, when executed by a simulator the same experiment lasted more than 30 minutes.

5.3 Systematic Test of the "Action Point Offset"

The Action Point Offset parameter describes the position of the frame within its communication slot as illustrated in Figure 3. The aims of these experiments were to shift the frame with positive and negative offsets in order to check the frame reception tolerance, and to test whether the boundary violation detection mechanisms operate as specified.

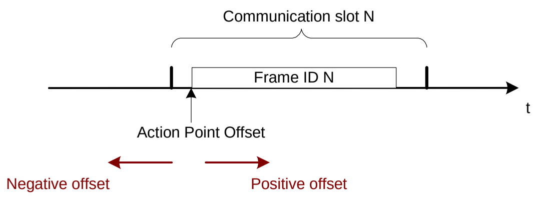

Figure 3: Action Point Offset mechanism

Three experiments were conducted for this test campaign.

1. *Minor offset:* Add increasing negative and positive offset values to the action point offset to slide the frame within a communication slot.
2. *Major offset:* Add increasing negative and positive offset values to the action point offset to slide the frame out of a communication slot, i.e. so that it overlaps two slots.
3. *Excessive offset:* Shift the action point far enough so that the entire frame is positioned into the previous/following slot, respectively.

Exp.	Node	Reported Results
1	SuT Nodes	Frames received correctly
	Tester Node	• Action Point within range • Offset to the boundary value
2	SuT Nodes	Frame Status Error
	Tester Node	• Action Point out of range • Offset to the boundary value
3	SuT Nodes	Frame Status Error
	Tester Node	• Action Point within range • Offset to the boundary value

Table 2: "Action Point Offset" experiment results

Table 2 summarizes the findings with our test approach. In the third experiment the tester identified the "Action Point Offset" as within bounds, although it was shifted to another slot. Instead, the "medium access check" attribute reported an error; hence, by way of other attributes this error could be uncovered.

The "Action Point Offset" experiment lasted 35 seconds; in total about 300.000 faults and/or modifications were injected and carried out respectively.

5.4. Industrial Application of the Approach

Our industry partner DECOMSYS is involved in the development and test of the FlexRay protocol and solutions for FlexRay based distributed systems. To maintain the high quality standards of their solutions in the highly competitive market of automotive electronics extensive and efficient product tests are required. Additionally, tools and methods for cluster testing are being demanded by customers. DECOMSYS is committed to provide them with a comfortable, efficient and flexible test environment that meets the high demands associated with the verification of a safety critical application. In this context the test strategy developed within the scope of the FlexRay Cluster Tests in 2002 has been recently upgraded with the BUDOCTOR[3] [12] that has been developed within the STEACS project, based on the concepts presented within this paper.

6. Benefits and Limitations

One key concept within our approach is to project the test of the communication system to a check of well defined attributes. The consequent implementation of this concept yields a highly efficient test approach in several respects: First of all, the decomposition of the complex communication system into (much simpler) services allows a substantial reduction of the target complexity. A systematic one-by-one check of the services requires substantially fewer test vectors than a less structured approach within the whole space of possible stimuli (and sequences thereof) as it is often applied. As a result our concept allows a much more systematic coverage of precisely targeted services than the ad-hoc test approaches often found in current practice. The structured search can be easily automated and provides high coverage within reasonable time. Still, however, all desired "ad-hoc" test scenarios can be composed as a superposition of attribute changes. In the same way diagnostic capabilities are improved: The systematic procedure allows to clearly identifying the erroneous service (and the erroneous node, if desired).

The crucial point within this concept, however, is the reduction of a service to its attributes. In fact, the attribute is used to parameterize a model implied in the data transformation module and the data generation module, respectively. Obviously the accuracy of this model limits

[3] http://www.decomsys.com/flyer/BUSDOCTOR.pdf

the quality of the whole approach and hence these models must be elaborated with special care (at this point the decomposition to simple services comes to the rescue again). All aspects that are not (correctly) reflected by this model will not be (correctly) covered by the test. Another crucial issue in this context is the mutual independence of the services: Decomposition and structured testing are only useful, if the services do not influence one another. It is the nature of a layer model to build the higher level services upon the lower level services – in this sense the independence of services can never be assumed. It is therefore necessary to employ a "bottom-up" test approach, thus ensuring the correct function of the lower levels before proceeding to the higher ones. Only then a correctly functioning lower level service becomes transparent with respect to testing the higher-level ones, as we have claimed in [6].

If this can be taken for granted, the test of a service effectively comes down to a comparison of its observed attribute(s) with a single (or at most several) reference values. This not only minimizes the efforts for diagnosis but also supports the intuition and avoids error-prone interpretation of complex scenarios as it is sometimes required in traditional approaches. The same is true for the test of fault tolerance mechanisms and the construction of the required bus traffic. As a prerequisite our test framework must allow us to view a given message (sequence) from the abstraction level that is appropriate for the service under consideration, and must thus allow us to arbitrarily change between these levels. The data monitoring module and the data injection module, respectively, provide this functionality. The accuracy of these transformation models, however, limits the quality of the approach.

The ability to arbitrarily change between layers is also an important enabler for remote testing: By monitoring and modifying the bus traffic on the physical layer we have (indirect) access to all services within all nodes [6], provided that we do not consider Byzantine faults. Access to the physical layer is easily possible at any point of the communication network, while direct access to higher layers would involve physical access to each single node under test. Notice, however, that our remote test approach only allows us to observe node behavior as far as it affects bus behavior. This means that the triggering of an error detection mechanism or the suppression of an erroneous message is not necessarily noticeable by a remote tester. A simple and straightforward solution is to provide the tester with a more comprehensive access to the node (at the host side, e.g.), thus sacrificing the advantage of a purely remote access. Alternatively some form of loop-back of the high-level node behavior to the bus might be considered, and indeed several options exist to implement this and achieve membership-like functionalities. For example, identification of bus failures as described in [15] with redundant channels can be achieved with a non-redundant channel with our test approach. A further discussion of these options, however, goes beyond the scope of this paper.

A tester node that monitors the bus traffic during the mission can indeed improve system dependability through early detection of service failures. It is, however, not the dedication of the tester node to enhance the system's error detection capabilities like concurrent checking methods ([16], e.g.) do. Aliasing effects in the CRC check, e.g., are an accepted – albeit not desired – facet of the proper node function, hence they must be reflected by the model implemented in the tester node as well.

Similar aliasing effects may show up during fault injection as well: A fault that is severe enough to exceed the system's (limited) error detection capabilities may generate a seemingly correct input. In fact this input is valid on the considered layer but likely to cause an error on a higher layer. Remember our second experiment campaign (Section 5.3) where we have observed an example of this effect.

Although we have developed our approach for the specific problem of testing a FlexRay based communication system, it is applicable to different systems and different layers as well (in the sense that the concepts can be adopted, not the tools). However, for use in higher layers, like application level debugging, e.g., additional issues have to be considered (see [7]).

7. Conclusion

We have proposed a systematic test approach for distributed automotive communication systems for the FlexRay example. The key concept is the decomposition of the communication system into a set of (independent) services that are characterized by attributes. We have further presented a framework that allows extracting, checking and modifing all attributes in a systematic way. Our approach allows a very efficient and well targeted test of all attributes and hence all services within reasonable time. It also includes fault injection for the test of fault tolerance mechanisms. An additional advantage is that the involved procedure is easy to automate. We have illustrated our concepts by two experiment campaigns that tested specific attributes.

We have kept our approach very flexible and have minimized the requirements on the test setup in order to allow its application during various phases of the system's lifecycle. For example, test of the algorithms during system validation, fault removal during system development, conformance and interoperability tests during system verification, fault-forecasting and robustness tests during system evaluation or long-term monitoring during system maintenance can be easily accomplished.

Further issues that we will consider comprise the extension of our fault injector to on-line modification of bus traffic and the development of a GUI that presents the results in a condensed and meaningful way and that allows the user to fully exploit all facets of the framework.

8. References

[1] Leen, G; Hefferman, D; "In-Vehicle Networks, Expanding Automotive Electronic Systems", IEEE Transaction on Computers, January 2002, pp. 88-93.

[2] –, "FlexRay Communications Systems - Protocol Specification Version 2.0", FlexRay Consortium, 2004. (http://www.flexray.com).

[3] Bardell, P; McAnney, W; Savir, J; "Built-in Test for VLSI, Pseudorandom Techniques", John Wiley & Sons, New York 1987.

[4] Dawson, S.; Jahanian, F.; Mitton, T.; Teck-Lee Tung; "Testing of fault-tolerant and real-time distributed systems via protocol fault injection", Proceedings of the Annual Symposium on Fault Tolerant Computing, 1996., , 25-27 June 1996, pp. 404 – 414.

[5] Carvalho, J.; Portugal, P.; Carvalho, A.; "A framework for dependability evaluation of PROFIBUS networks", IEEE Int. Symposium on *Industrial Electronics, 2003* (ISIE '03) , Volume: 1 , 9-11 June 2003, pp. 466 - 471 vol. 1.

[6] Armengaud, E.; Steininger, A.; Horauer, M.; Pallierer, R.; "A Layer Model for the Systematic Test of Time-Trirggered Automotive Communication Systems", 5th IEEE International Workshop on Factory Communication Systems (WFCS'04), pp. 275 – 283, 2004.

[7] Chandra, R; Lefever, R.M.; Joshi, K.R.; Cukier, M.; Sanders, W.H.; "A global-state-triggered fault injector for distributed system evaluation", IEEE Transactions on Parallel and Distributed Systems, Volume: 15 , Issue: 7 , July 2004, pp. 593 - 605.

[8] Stott, D.T.; Floering, B.; Burke, D.; Kalbarczpk, Z.; Iyer, R.K.; "NFTAPE: a framework for assessing dependability in distributed systems with lightweight fault injectors", Computer Performance and Dependability Symposium, 2000 (IPDS 2000), 27-29 March 2000, pp. 91 – 100.

[9] McCanne, S; Jacobson, V.; "The BSD Packet Filter: A New Architecture for User-level Packet Capture", USENIX Conference, January 1993, pp. 259 – 269.

[10] Yuhara M; Bershad, B.N.; Maeda, C; Moss, J.E.B.; "Efficient Packet demultiplexing for multiple endpoints and large messages", USENIX Conference, January 1994, Second Edition.

[11] Arlat, J.; Aguera, M.; Amat, L.; Crouzet, Y.; Fabre, J.-C.; Laprie, J.-C.; Martins, E.; Powell, D.; "Fault injection for dependability validation: a methodology and some applications", IEEE Transactions on Software Engineering, Volume: 16 , Issue: 2 , Feb. 1990, pp. 166 – 182.

[12] Horauer, M.; Rothensteiner, F.; Zauner M.; Armengaud, E.; Steininger, A.; Friedl H.; Pallierer, R.; "An FPGA based SoC Design for Testing Embedded Automotive Communication Systems employing the FlexRay Protocol"; Austrochip 2004, pp. 119-125, Villach - Austria, October 2004, ISBN: 3-200-00211-5.

[13] Rasmussen, K.A.; "Open systems protocol testing techniques", IEEE International Conference on Communications 1990, (ICC 90), 16-19 April 1990, vol.4, pp. 1387 - 1391.

[14] Karlson, J; Folkesson, P; Arlat, J; Crouzet, Y; "Integration and Comparison of the Three Physical Fault Injection Techniques" Predictably Dependable Computing Systems, Springer, pp. 309-329, 1995.

[15] Temple. C.; "Identifying Bus Failures in a Time-Triggered Communication System Containing Redundant Communication Channels", The 2000 International Conference on Communications in Computing (CIC'2000), 26th - 29th June 2000, Monte Carlo Resort, Las Vegas, Nevada, USA.

[16] Rela, M; Madeira, H; Silva, J; "Experimental Evaluation of the Fail-Silent Behavior in Programs with Consistency Checks", Proc. 26th International Symposium on Fault-Tolerant Computing (FTCS-26), Sendai, Japan, June 1996, IEEE CS Press, pp. 394-403..