Test Specification and Test Methodology for Embedded Systems in Automobiles

Juergen Grossmann, Diana Serbanescu, Ina Schieferdecker

{juergen.grossmann,diana.serbanescu,ina.schieferdecker}@fokus.fraunhofer.de

Abstract: Despite the past few years' intensive efforts of automobile manufacturers and their suppliers to enhance the quality assurance of their products, the problems of testing systems that, like automobiles, steadily increase in complexity and interconnectedness are still not solved. The variety of proprietary test systems and solutions do not allow an integrated definition, transfer, re-use and execution of tests for automobile manufacturers, suppliers and test equipment manufacturers. In contrast to the automotive industry, the telecommunication industry uses an integrated, manufacturer independent and standardized test technology: the **T**esting and **T**est **C**ontrol **N**otation (TTCN-3). The advantages of this technology have already been recognized by the automotive industry and therefore, AUTOSAR decided to use TTCN-3 for the definition of functional tests. However, the components of automobiles do not only have functional requirements. In addition to functional input-output behavior, also continuous data streams and real-time behavior have to be tested. The current version of TTCN-3 has only limited capabilities for testing such non-functional properties. To overcome these limitations, a consortium from industry and academia started the research project **TE**st specification and test **M**ethodology for **E**mbedded systems in **A**utomobiles (TEMEA). TEMEA is targeted on developing a TTCN-3-based test specification and test implementation methodology for embedded systems in automobiles. The project will develop solutions to test continuous data streams and real-time behavior with TTCN-3 and also investigate software engineering related issues like quality assessment and quality improvement of large test specifications. This presentation will introduce the TEMEA project and will present the first results of the initial requirements capturing phase of the project. It should stimulate discussions on future enhancements of the TTCN-3 language.

1 Introduction

The development of modern vehicles went through a major change regarding the realization of innovative functions and technologies in recent years. In former times improvements were based on mechanical and electrical systems in particular. Nowadays, innovations are realized particularly by electronic systems and software-driven functions.

Also current challenges – consumption reduction, effective drives, active safety functions, driver assistance – are not realizable without additional support by software-driven, often

cross-linked control systems.

Software systems in the automotive industry typically are embedded, distributed and – caused by the different areas of application (telematics, infotainment, power control, comfort electronics) – heterogeneous. A modern vehicle comprises between 30 and 80 different controllers, which communicate with one another over different bus systems. A comprehensive and systematic quality assurance of such complex systems is – still for the automotive industry, which is technologically and methodically very well equipped – a large challenge. Within the research project TEMEA [TEM08] we bear this challenge and started – under participation of our automotive partners – to develop a uniform and standardizable test specification platform for embedded real-time systems, which will be tailored to the requirements and technologies of the automotive industry. Our approach is based on the world-wide only standard for test specification and implementation: the Testing and Test Control Notation (TTCN-3) [ETS07a].

1.1 The Motivation

Both, car manufacturers (OEMs) and their suppliers, have invested strongly into the quality assurance of their products. Like that it is not remarkable that the quality of the vehicles and, in particular, their electronic components improved clearly in comparison to the last decade. Nevertheless, the quality assurance of software-intensive systems in the automotive industry is still characterized by high manual portions (i.e. a low level of automation) and a multiplicity of often proprietary test systems and test platforms. The latter rely on diverse description and specification techniques, that are only weakly formalized and normally not designed for the exchange between different test systems. Particularly for the automotive industry, which is characterized by strongly distributed development processes, not a good premise.

In contrast to this AUTOSAR [AUT08], a standardized platform for software architectures and software components, is currently established in the automobile industry. The standard is defined by a consortium, which is founded by a majority of European and international car manufacturers and their suppliers. In the telecommunication industry, which is confronted already for decades with the integration of devices and systems based on telecommunication standards, a standardized test technology was established. TTCN-3 provides a manufacturer-independent test environment consisting of modular service components (e.g. test management component, test logging component, platform adapters etc.) and a formalized test specification language.

Meanwhile, the advantages of this technology were recognized by the automotive industry. Thus, AUTOSAR decided to use TTCN-3 in the context of functional tests for basic software modules.

1.2 TEMEA Goals and Structure

While standardized technologies and model-based approaches already exist for the development of ECUs (electronic control units), a comparable standard is missing for automotive test systems. With TTCN-3, such a standard already exists for the telecommunication industry. In its current stage, TTCN-3 has limits in its application to automotive software. Particularly with regard to the classical automotive software domains, which by the majority address the development of real-time regulation systems and control systems (like e.g. drive electronics, driving assistance functions etc.), TTCN-3 lacks effective language means that provide a meaningful and effective support to describe the properties to be tested. Beyond that TTCN-3 is so far not integrated in the process chains and tool environments of the automotive industry.

Introducing **TTCN-3-embedded**, which is based on the existing TTCN-3 standard and previous research [GM06, SBG06, GSW08, SMD+08], we will develop a test specification language, which will be tailored specifically to the requirements of the automotive industry. The technology addresses particularly the interface between suppliers and car manufacturers, i.e. the exchange of formalized test specifications. The language supports testing of software components and vehicle functions on component level, integration level, and system level and will allow a detailed, objective, implementation and platform-independent definition of tests.

Thus both the degree of reuse and the degree of test automation can be increased substantially. TTCN-3-embedded will establish a common platform for the various test solutions needed by the automotive industry. Moreover, we will address new aspects like combined tests for discrete, real-time and continuous behavior of distributed and communicating software components. TTCN-3-embedded will facilitate the exchange of test specifications and creates an obligatory basis for the training of test designers and engineers. For a smooth integration into the development processes of the automobile industry we set on AUTOSAR. The aims of the project can be summarized as follows:

- Support for integrated testing of discrete and continuous behavior.
- Exchange of test definitions between different test- and simulation platforms e.g. Model in the Loop (MIL) platforms, Software in the Loop (SIL) platforms, and Hardware in the Loop (HIL) platforms.
- Support over the entire process of software integration and hardware integration.
- Analysis of real-time and reliability requirements.
- Testing distributed components according to AUTOSAR architecture.
- Analysis of the quality of tests.

The project TEMEA runs over two and a half years (2008 to 2010) and is split into different work packages (WPs). WP1 analyses the inventory of existing test technologies in the automotive industry and the requirements for TTCN-3-embedded. WP2 develops the language extensions for TTCN-3-embedded based on the requirements from WP1.

WP3 addresses the integration of TTCN-3-embedded with the concepts and artifacts of the AUTOSAR standard and will develop the TEMEA test system. WP4 compiles a suitable test methodology and WP5 provides a demonstrator that will validate the project developments by industrial applications.

1.3 Integration with Model-Based Engineering: An AUTOSAR Test Framework

Like in other industrial domains, software development in the automotive industry is affected by methods and technologies that provide a more abstract view on the software product itself. Whereas so far the ECU, that is the target hardware, has determined the perspective of development already in the early phases of the development process, model-based approaches and the use of transparent middleware platforms will lead to a much more hardware independent view on software.

The AUTOSAR [AUT08] consortium aims at the specification of a vendor-independent standard of a software platform and software components for the automotive industry. The respective AUTOSAR standard covers the definition of a software architecture dedicated to automotive ECUs, the definition of a modular run time environment and a transparent middleware infrastructure (AUTOSAR run-time environment), as well as a semi formal specification language (AUTOSAR templates) for the definition of hard- and software components and/or their aggregation to complex architectures. AUTOSAR supports a systematic separation of concerns. It supports a consistent modularisation of software by introducing software components and component clusters. It separates the specification of software components from its distribution and its deployment to the target hardware. AUTOSAR defines standardised communication interfaces and reusable components for basic software functionalities (e.g. memory management, thread management, communication stacks, drivers for sensors, actuators, etc).

Whereas the importance of AUTOSAR for software development is well understood, its potential for testing is not yet discovered. In TEMEA we act on the assumption that the successive initiation of AUTOSAR to the development processes will result in a whole number of new chances and challenges especially for the quality assurance processes. In principle, the complexity of the development products increases along the variability introduced by AUTOSAR. Flexible options for the configuration and distribution of individual software components will lead to enhanced integration processes that require new quality assurance and testing methods, which should be tailored directly to the AUTOSAR capabilities. On the other hand, standardisation and formalisation create a promising basis for more effective and automated approaches to test software components, ECUs, and their compositions.

TEMEA has started to provide an overall test methodology based on TTCN-3 that is tightly integrated with the AUTOSAR development process. We work currently on a consistent mapping between AUTOSAR and TTCN-3 concepts. Future work will address the systematic and automated generation of test architectures and simple test behavior from AUTOSAR specifications.

With the next AUTOSAR release (release 4 in 2009), AUTOSAR will provide enhanced support for the definition of real-time requirements and timing (AUTOSAR timing speci-

fication). For testing these requirements, the test environment and the respective test language must support means to adequately express timing and real-time requirements. Although TTCN-3 is already used to test AUTOSAR basic software components, TTCN-3 does provide only limited means for that. Presenting our first project results, the real-time enhancements of TTCN-3-embedded, we will show how TTCN-3 can be systematically extended to support real-time testing more adequately and can as such provide a future-proof basis for AUTOSAR testing.

2 Requirements Towards Testing of Reactive Real-Time Systems

Testing involves any activity aimed at evaluating attributes or capabilities of programs or systems, and finally, determining if they meet all of the requirements. Functional black-box testing is purely based on the requirements on the system under test (SUT) and mainly used for integration-, system- and acceptance-level testing. There exist several types of black-box testing which focus on different kinds of requirements or on different test goals, such as functional testing, conformance testing, interoperability testing, performance testing, scalability testing, and so forth. In the following, we consider conformance testing for embedded real-time systems and show how timing properties can be systematically integrated in the testing approach.

Figure 1 shows the setup for a simple functional test that does not consider timing requirements. A test component is used to stimulate the SUT. The outputs of the SUT are captured by the test component and matched against predefined message templates. If they coincide, the tested requirement is considered to be fullfilled, and the test passes. Otherwise, the test fails. Listing 1 shows an implementation using TTCN-3.

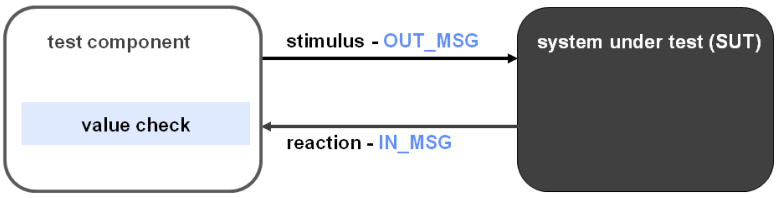

Figure 1: Simple black-box test system with message-based communication

Listing 1: Simple black-box test system: TTCN-3 code example

```
p_out.send(OUT_MSG);                                    1
alt {                                                   2
  [] p_in.receive(IN_MSG) {setverdict(pass);}           3
  [] p_in.receive(IN_MSG) {setverdict(pass);}           4
}                                                       5
```

In addition, real-time systems have to respect special requirements for timing [HS91]. Checking the message values and the message order is not sufficient here. Functional requirements are directly connected to the timing of the messages. That is functionality must be accomplished within a certain time interval, its starting or ending should be denoted by precisely defined time points or time spans that respect a given tolerance. The test components must be able to check whether a message has received in time.

Listing 2: Black-box test with time restrictions: TTCN-3 code example.

```
timer t;
p_out.send(OUT_MSG_1);
t.start(t_max);
alt{
    [] p_in.receive(IN_MSG_1) {t.stop; setverdict(pass)};
    [] p_in.receive {t.stop; setverdict(fail)};
    [] t.timeout(){setverdict(fail)}
}

t.start(t_wait)
t.timeout;
p_out.send(OUT_MSG_2);
alt{
    [] p_in.receive(IN_MSG_2){setverdict(pass)};
    [] p_in.receive{setverdict(fail)};
}
```

Moreover timing is not relevant for incoming messages only. A test component must as well be able to control the timing for the stimulation. Thus it has to ensure that a messages will be send at a defined point in time. These points in time may either be defined by absolute time values or relatively in respect to certain events (e.g. the receive of a message). Last but not least we shall be able to calculate and to compare time values with respect to the required exactness (e.g. microseconds).

An example of a test logic with timing requirements is given in Figure 2. Regarding time, there are two critical sections in this test example: first, there is t_{max}, a time constraint for the SUT that indicates the interval in which the reaction to the first stimuli should be received by the test component; the second is t_{wait}, which indicates the time that should elapse at the test component side, between the receiving of the first reactions from the SUT and the sending of the second stimulus to the SUT.

The test logic for this example is comprised with traditional TTCN-3 in Listing 2.

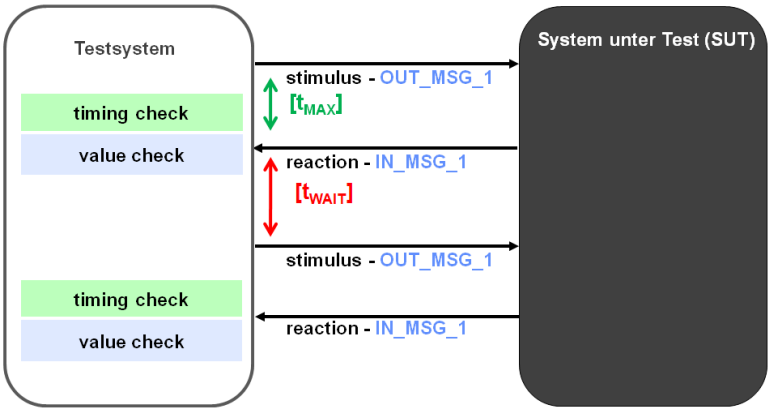

Figure 2: Black-box test with time restrictions

Obviously, traditional TTCN-3 already provides means to describe test cases that respect timing. Nevertheless, the solution denoted in Figure 2 has several disadvantages:

- The concept of a timer was not intended for suiting real-time properties, but conceived only for catching, typically mid- or longterm, timeouts. The specification of

Listing 3: Black-box test with time restrictions: TTCN-3-embedded code example

```
p_out.send(OUT_MSG_1) -> timestamp stamp;                                    1
alt{                                                                         2
[] p_in.receive(IN_MSG_1)                                                    3
        within (stamp..stamp+t_max) -> timestamp stamp;                      4
[] p_in.receive{setverdict(fail)}                                            5
}                                                                            6
p_out.send(OUT_MSG_2) at (stamp+t_wait);                                     7
alt{                                                                         8
[] p_in.receive(IN_MSG_2){setverdict(pass)}                                  9
[] p_in.receive{setverdict(fail)}                                            10
}                                                                            11
```

real-time properties using the concept of a timer is often clumsy and because of the non-real-time semantics of TTCN-3 not exact.

- The use of a timer is affected by the TTCN-3 snapshot semantics and by the order in which receive and timeout statements are aligned in the alt statement. TTCN-3 in general makes no assumptions about the duration for taking and evaluating a snapshot which may vary, depending on different implementation solutions.

- The measured time point is in fact not the time point of message reception (i.e. the time a certain message has entered the input queue of the test system), but the time point of the evaluation of the queues by the test program. This kind of measurement approach is not exact.

- Moreover, time is consumed for the encoding and decoding of messages as well as for the process of message matching. This time consumpotion is neither controllable nor assessable by the tester and enters as inaccuracy to time measurements when using TTCN-3 timers.

- In TTCN-3, timer values (i.e. time) are represented by floating point numbers. Floating point numbers exhibit however inaccuracies in its limited number of digits and thus result in rounding errors.

- Last but not least, TTCN-3 is itself not real-time. The tester has no control of what happened between two instructions. For example, if the garbage collection starts between the receiving of a message and the start of the timer the measurement of time is no longer exact. These inaccuracies may vary.

The concepts that are introduced for TTCN-3-embedded are meant to overcome the limitations of traditional TTCN-3 mentioned above. Listing 3 informally introduces a TTCN-3 embedded implementation for the test task depicted in Figure 2. The timestamp operator (`-> timestamp`) writes the accurate point of transmission time of the message `OUT_MSG` into the variable `stamp`. This value can then be reused for the allocation of the within operator, which is parameterized by a time interval `(stamp..stamp+t_max)` that specifies the permitted arrival time of the message. The ***at*** operator in the following line specifies the exact point of message delivery. A more detailed specification will be given in the following sections.

Listing 4: Use of datetime and timespan values

```
var datetime starttime, actualtime; // declare datetime variables
var timespan distance, maximum;     // declare timespan variables

starttime:= testcasestart;          // using predefined symbols
actualtime:= now;                   // using the now operator
```

3 Real-time Extensions for TTCN-3

The Testing and Test Control Notation TTCN-3 [ETS07a, ETS07b, ETS07c] is a test specification language, which was originally developed to meet the requirements of testing telecommunication systems. It is standardized by the European Institute for Telecommunications Standards (ETSI). TTCN-3 is very expressive as a testing language, providing all the necessary features for writing reliable and maintainable test systems for a wide range of application domains. It is a modular and well-structured language for testing not only functional requirements like conformance or interoperability, but also other qualitative and quantitative requirements of the targeted systems. TTCN-3 allows parallel tests and distribution due to the concept of test components.

In the following, we discuss and introduce new instruments for dealing with real-time requirements in order to solve the problems presented in Section 2. Carefully selected new additions to the language were developed as a consolidation of former approaches [DGN02, Neu04, SMD+08]. The additions cover aspects like measuring and verifying timing information of the tested real-time system as well as controlling the timing of the test system itself. Our approach is based on the assumption that our test system posses a time deterministic behavior with respect to the given testing tasks. Thus, it has enough resources to schedule the testing tasks in time.

3.1 Representation of Time

In order to ease the manipulation of time values, two new abstract data types are introduced. The predefined types give general guidelines on how time values should look like and how they should be interpreted. The actual implementation of those time values depends on the coding techniques beneath the abstract level of TTCN-3. They can be built as user-defined types and added into the language via a separate module, without the need to change the core grammar itself:

The introduced data types for handling time are:

- ***datetime*** designates global time values. They uniquely specify the points in time when certain events occurred or should occur. Datetime values are either measured directly or denoted using a string representation according to the RFC 3339 [Net02], an RFC providing information about how to use time for internet applications, based on the ISO standard ISO 8601 [Int88].
- ***timespan*** designates time distances or intervals between different points in time. It is used to represent the amount of time that passed between events. Timspan values

Listing 5: Arithmetic and boolean expressions

```
distance:= now−starttime;          // calculating with time values
maximum:= 200∗millisec;             // literal of a timespan value

if(distance>maximum) {             // comparision of timspan values
  log("limit hurt")
}
```

Listing 6: Measurement of time

```
var datetime sendtime, receivetime;
port.send(MSG_OUT) -> timestamp sendtime;
port.receive(MSG_IN) -> timestamp receivetime;
```

are denoted using expressions that are constructed by integer values and a time unit representing constant.

⟨TimeSpan⟩ ::= ⟨Expression⟩ `"*"`
{`"picosec"` | `"nanosec"` | `"millisec"` | `"second"` | `"minute"` | `"hour"` | `"day"`}

In order to give the tester the right instruments of detecting relative time, some predefined symbols of the type datetime are introduced. Together with the ***now*** operator, that returns the current time value, several execution related timings can be deduced:

- ***testcasestart*** returns the time point when the test case execution started.
- ***testcomponentstart*** returns the point in time when the test component execution started.

Time values can be used in arithmetic and boolean expressions (see Listing 5) respecting a few simple rules: The substraction or addition of two timespan values results in a new timespan value. Timespans can be multiplied or divided by floats or integers and the result should be evaluated to timespan. The difference between two datetime values results in a timespan value. To enable a seamless integration of time types, either with each other as well as with existing TTCN-3 types, we introduce conversion function that provide conversions between time values, string values and/or float values.

3.2 Measurement of Time

In TTCN-3-embedded, the observation of time is directly connected to the reception and provisioning of messages at communication ports. We introduce a construct for automatically registering the time value at which a receive or send event occurred. The saving is indicated by redirect symbol **->** and the ***timestamp*** keyword. The value is stored as a datetime value in a variable.

Listing 7: Control of stimulation

```
// sends a message exactly 300 milliseconds after the start of the test case
port.send(MSG_OUT) at testcasestart+300*millisec;
```

The syntactic rules for this extension of a receive statement in TTCN-3 are:

⟨ReceiveStatement⟩ ::= ⟨PortOrAny⟩ `"."` ⟨PortReceiveOp⟩

⟨PortReceiveOp⟩ ::=
`"receive"` [`"("` ⟨ReceiveParameter⟩ `")"`] [⟨FromClause⟩] [⟨PortRedirect⟩]

⟨PortRedirect⟩ ::=
`"->"` ((⟨ValueSpec⟩ [SenderSpec] | ⟨SenderSpec⟩)[TimeStampSpec]) | ⟨TimeStampSpec⟩

⟨TimeStampSpec⟩ ::= `"timestamp"` ⟨DateTimeValue⟩

The timestamp-operator is available for message-based communications (i.e. for send, receive, and trigger statements) as well as for procedure-based communication[1] (i.e. for call, getcall, reply, getreply, raise, and catch statements).

3.3 Control of Application

The ***at*** operator is introduced to send certain messages at fixed points in time. It is associated with a datetime value, representing the point in time when the sending of a message should be performed. Following instructions are suspended during the waiting period. Two different situations are distinguished:

- If the requested time is not yet reached when the send statement is executed, the test component waits and guarantees that the message is dispatched exactly at the requested point in time.
- If the requested point in time is already exceeded, the test system will set an error verdict.

The at operator can be applied to other statements associated with outgoing communication, i.e. call, reply, and raise statements.

[1] For more information about the differences between message-based communication and procedure-based communication and the corresponding TTCN-3 statements please refer to [ETS07a].

Listing 8: Verification of incoming message

```
// the  message is valid when its value conforms to MSG_IN
// and is received at least 300 milliseconds after
// the start of the test case
port.receive(MSG_IN)
     within (testcasestart..testcasestart+300*millisec);
```

The extension introduced for adding the ***at*** operator to the *send* statement are as follows. The syntactic extensions for call, reply, and raise are similar.

⟨SendStatement⟩ ::= ⟨Port⟩ `"."` ⟨PortSendOp⟩

⟨PortSendOp⟩ ::=
`"send" "("` ⟨SendParameter⟩ `")"` [ToClause][TimeStampClause][`"at"`⟨DateTime⟩]

Finally, we provide functions to block the execution for or until a given point in time independent of communication means. The ***sleep*** function takes as a parameter a timespan value whereas the ***waituntil*** takes a datetime value:

⟨SleepConstruct⟩ ::= `"sleep" "("` ⟨TimeSpanValue⟩ `")"`

⟨WaitUntilConstruct⟩ ::= `"waituntil" "("` ⟨DateTimeValue⟩ `")"`

3.4 Time Verification

In order to verify whether certain messages were received in time, we introduce the ***within*** operator. The operator is associated with an interval of datetime values that represent the range for the allowed times of reception. The within operator extends the TTCN-3 matching mechanism. Thus in traditional TTCN-3, the matching mechanism is mainly responsible for the examination of value related properties. In TTCN-3-embedded, we additionally consider the timing information of a message. That is, a receive statement with a within operator is only processed successfully when the message value conforms to the value template and the reception time of the message is included in the given time interval.

Listing 8 illustrates a small piece of code where the expected message should conform to `MSG_IN` and as an additional restriction, it should be received between the indicated points in time. These points refer to the start of the test case and are calculated by adding timespan values.

The syntactic extension for the receive statement is given below. The ***within*** operator can

also be applied to trigger, getcall, getreply, and catch statements.

⟨ReceiveStatement⟩ ::= ⟨PortOrAny⟩ `"."` ⟨PortReceiveOp⟩

⟨PortReceiveOp⟩ ::=
`"receive"` [`"("` ⟨ReceiveParameter⟩ `")"`] [FromClause]
[`"within" "("`⟨DateTime⟩`".."`⟨DateTime⟩`")"`] [PortRedirect]

4 Case Study

As a proof of concepts, we provide an example that is typical for the automotive industry. Figure 3 shows the setup for a blinker system that is composed of a central controller unit (master) and several actuator components that serve the indicator lights on the front and the backside of the car.

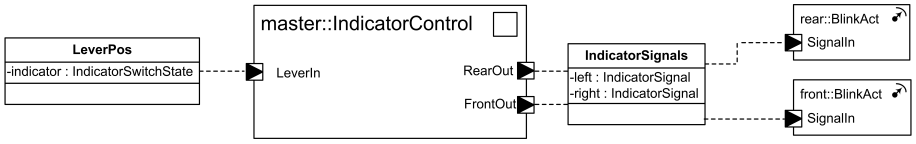

Figure 3: Simplified system setup for a blinker system modelled with AUTOSAR

Our simplified controller model has three ports: an input port (`LeverIn`) that receives the actual position of the lever, and two output ports (`RearOut, FrontOut`) that are used to communicate the indicator signals to different actuator components. Such an indicator system is usually realized as a distributed system. The central software component (master) is placed on a central ECU (e.g. SAM in the case of Mercedes or BCM in the case of VW) that usually hosts multiple applications. The actuator components can either be located on the central ECU (usually the ones for the front indicators) or on distributed controllers (e.g. rear controller, door controllers etc.). The functionality of our controlling component is supposed to be simple. It can be activated from the outside through the lever signal and – in response to that – it generates two output signals, one for the rear indicators and one for the front indicators.

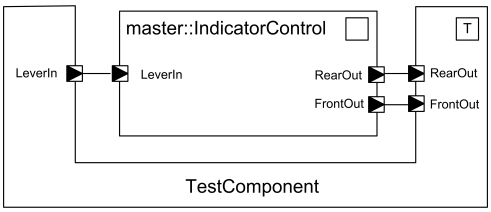

Figure 4: Test setup for the blinker example

The subject of testing is the timing of the signals as depicted in Figure 5. The timing aspects can be decomposed into the following three timing requirements:

- The outgoing signals should react on their activation at least `60ms` after the respective input signal has received. This requirement is tested by test case `tc_1` given in

Listing 9: Test interface definitions derived from the AUTOSAR model

```
module MainModule {                                                    1
  type enumerated IndicatorSwitchState {OFF, LEFT, RIGHT}              2
  type enumerated IndicatorSignal {OFF, ON}                            3
  type port LeverPos message {out IndicatorSwitchState}                4
  type port IndicatorSignals message {in IndicatorSignal}              5
                                                                       6
  type component IndicatorTestComponent {                              7
    port IndicatorSignals RearOut, FrontOut;                           8
    port LeverPos LeverIn}                                             9
}                                                                      10
```

Listing 10.

- Both outgoing signals should behave synchronously with a tolerance of `30ms`. This requirement is tested by test case `tc_2`, which is given in Listing 11.
- The length of the indicator period should last `600ms` (`300ms ON` and `300ms OFF`) with a tolerance of `20ms`. This requirement is tested by test case `tc_3` given in Listing 12.

For testing, we replace the actuator components and the input emitting component by a single test component. The proposed configuration of the test system is visualized in Figure 4.

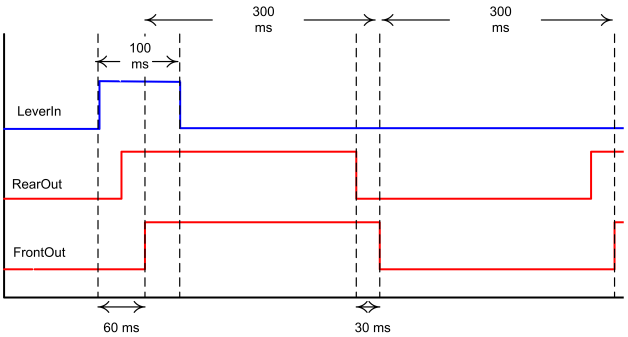

Figure 5: Signals and timing

The TTCN-3 test interface is directly derived from the AUTOSAR models (see Figure 3 and 4). The respective TTCN-3 code that specifies the ports and messages is depicted in Listing 9. The values of the messages are specified by enumeration types. The switch signal (`LeverIn`) is specified as an outgoing signal and the indicator signals (`RearOut`, `FrontOut`) as incoming signals.

Test case `tc_1`, listed in Listing 10 tests the activation of the indicator signals. After the `LeverIn` signal has switched to `LEFT` (see line 5), we expect to receive an activation (`ON`) signal from `FrontOut` and `RearOut` on the associated ports. Both signals should reach the test system within 60 milliseconds after the activation signal (`LeverIn`) was sent.

To check the respective timing we save the time point, when we sent the activation signal in the datetime variable `t_sig`, using the timestamp mechanism (line 5). This point in

Listing 10: Tests the initial activation of the indicator signals

```
testcase tc_1() runs on IndicatorTestComponent{                          1
  var datetime t_sig;                                                    2
  const timespan T_MAX = 60*millisec;                                    3
  // saving the time of sending the signal                               4
  LeverIn.send(LEFT) -> timestamp t_sig;                                 5
  alt{                                                                   6
    [] FrontOut.receive(ON) within (t_sig..t_sig+T_MAX){                 7
      RearOut.receive(ON) within (t_sig..t_sig+T_MAX)}                   8
    [] RearOut.receive(ON) within (t_sig..t_sig+T_MAX){                  9
      FrontOut.receive(ON) within (t_sig..t_sig+T_MAX)                   10
  }                                                                      11
}                                                                        12
```

Listing 11: Tests the synchronisation between the indicator signals

```
testcase tc_2() runs on IndicatorTestComponent{                          1
  var datetime t_sig;                                                    2
  const timespan T_MAX = 30*millisec;                                    3
  LeverIn.send(LEFT);                                                    4
  alt{                                                                   5
    [] FrontOut.receive(ON)-> timestamp t_sig{                           6
      RearOut.receive(ON) within (t_sig..t_sig+T_MAX)}                   7
    }                                                                    8
    [] RearOut.receive(ON) -> timestamp t_sig{                           9
      FrontOut.receive(ON) within (t_sig..t_sig+T_MAX)}                  10
  }                                                                      11
}                                                                        12
```

time is used as reference to check the incoming data in the alt statement. For each incoming message we check whether it has the correct value (e.g. `FrontOut.receive(ON)`) and whether the timing is correct (**within (t_sig..t_sig+T_MAX)**).

If the timing does not match the given time range, the message remains in the queue and the execution is blocked. To prevent a deadlock, we use a default timeout (not depicted in the listing) that sets the test verdict to fail and finalizes the test case.

With the second test case (see Listing 11), we check the synchrony of the indicator signals. The timing of the two indicator signals should not differ more than 30 milliseconds. We verify this requirement once at the moment when the signals switch back from `ON` to `OFF`. Afterwards, they are supposed to be transmitted normally, just keeping the delay.

At first, the triggering signal from the test system side is transmitted to the SUT through the `LeverIn` port (line 4). After that, the two ports for the incoming signals are checked. If the signal on `FrontOut` is sensed first, the timing of its reception is saved in the datetime variable `t_sig` using the timestamp instruction. The maximum relative time for the second signal is calculated based on this value. The second signal is expected to be received in the calculated time range (line 7). Analogue actions are taken if the signal expected on the `RearOut` port is received first.

In the third test case, we test the phase length of the indicator signals (see Listing 12). We expect that the output signals remain `ON` for 300 milliseconds with a tolerance of 20 milliseconds although the activating switch signal hast turned to `OFF` again. The test case begins with activating the trigger signal on port `LeverIn`. As a reaction to this command, the SUT should activate signals on `FrontOut` and `RearOut`. When the signals are received, their time of reception is stored in `t_on_sig_1` and `t_on_sig_2`,

Listing 12: Tests the duration of the indicator phase

```
testcase tc_3() runs on IndicatorTestComponent {                          1
  var datetime t_on, t_on_sig_1; t_on_sig_2;                              2
  const timespan T_MAX := 300*millisec;                                   3
  const timespan TOL := 20*millisec;                                      4
  const timespan T_WAIT := 100*millisec;                                  5
  var timespan upper, lower;                                              6
  LeverIn.send(LEFT)-> timestamp t_on;                                    7
  alt{                                                                    8
    []FrontOut.receive(ON)-> timestamp t_on_sig_1{                        9
      RearOut.receive(ON) -> timestamp t_on_sig_2}                        10
    []RearOut.receive(ON) -> timestamp t_on_sig_2{                        11
      FrontOut.receive(ON) -> timestamp t_on_sig_1}                       12
  }                                                                       13
  LeverIn.Send(OFF) at t_on+T_WAIT;                                       14
  lower:=T_MAX-TOL;                                                       15
  upper:=T_MAX+TOL;                                                       16
  alt{                                                                    17
    []FrontOut.receive(OFF)                                               18
      within(t_on_sig_1+lower..t_on_sig_1+upper){                         19
        RearOut.receive(OFF)                                              20
        within (t_on_sig_2+lower..t_on_sig_2+upper){}}                    21
    []RearOut.receive(OFF)                                                22
      within(t_on_sig_2+lower..t_on_sig_2+upper){                         23
        FrontOut.receive(OFF)                                             24
        within (t_on_sig_1+lower..t_on_sig_1+upper){}}                    25
  }                                                                       26
}                                                                         27
```

respectively (lines 9-12). This is used to calculate time margins for the deactivation of both signals. After 100 milliseconds, another control signal is sent on the `LeverIn` port, indicating that the output signals should be turned off when the signal period expires. The expiration is calculated with the predefined tolerance in lines 18-25.

5 Summary

In this paper, we introduced the overall setup of the project TEMEA. The project develops a test technology and methodology for the automotive domain. Based on a standardized test technology, these enable the exchange pf test requirements, test cases, and test procedures between different parties involved in automotive systems development. Both methodology and technology are based on TTCN-3 and will contain several enhancements that adapt the already existing TTCN-3 standard to the needs of the automotive domain and especially to AUTOSAR. The project started with the specification of real-time enhancements to TTCN-3 that provide systematic support for testing timing requirements of automotive software applications under real-time conditions. We presented the real-time language features that were introduced to TTCN-3. We described their syntax and semantics and discussed their application by means of an example that reflects typical automotive timing requirements. In future work, we will provide a deeper integration with the AUTOSAR methodology and show how timing requirements can be derived directly from AUTOSAR software specifications.

References

[AUT08] AUTOSAR. www.autosar.org - web site of the AUTOSAR consortium, 2008.

[DGN02] Z. Dai, J. Grabowski, and H. Neukirchen. Timed TTCN-3 – A Real-Time Extension for TTCN. In *Testing of Communicating Systems, volume 14, Berlin, March 2002. Kluwer.*, 2002.

[ETS07a] ETSI: ES 201 873-1 V3.2.1. Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 1: TTCN-3 Core Language, Febr. 2007.

[ETS07b] ETSI: ES 201 873-4 V3.2.1. Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 4: TTCN-3 Operational Semantics, Febr. 2007.

[ETS07c] ETSI: ES 201 873-5 V3.2.1. Methods for Testing and Specification (MTS). The Testing and Test Control Notation Version 3, Part 5: TTCN-3 Runtime Interfaces, Febr. 2007.

[GM06] J. Grossmann and W. Mueller. A Formal Behavioral Semantics for TestML. In *Proc. of IEEE ISoLA 06, Paphos Cyprus*, pages 453–460, 2006.

[GSW08] Jürgen Großmann, Ina Schieferdecker, and Hans-Werner Wiesbrock. Modeling Property-Based Stream Templates with TTCN-3. In Suzuki et al. [SHUH08], pages 70–85.

[HS91] Wolfgang A. Halang and Alexander D. Stoyenko. *Constructing Predictable Real Time Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[Int88] International Organization for Standardization. *ISO 8601:1988. Data elements and interchange formats — Information interchange — Representation of dates and times*. International Organization for Standardization, Geneva, Switzerland, 1988. See also 1-page correction, ISO 8601:1988/Cor 1:1991.

[Net02] Network Working Group. Request for Comments: 3339, 2002.

[Neu04] Helmut Neukirchen. *Languages, Tools and Patterns for the Specification of Distributed Real-Time Tests*. PhD thesis, Georg-August-Universität Göttingen, 2004.

[SBG06] Ina Schieferdecker, Eckard Bringmann, and Juergen Grossmann. Continuous TTCN-3: Testing of Embedded Control Systems. In *SEAS '06: Proceedings of the 2006 international workshop on Software engineering for automotive systems*, pages 29–36, New York, NY, USA, 2006. ACM Press.

[SHUH08] Kenji Suzuki, Teruo Higashino, Andreas Ulrich, and Toru Hasegawa, editors. *Testing of Software and Communicating Systems, 20th IFIP TC 6/WG 6.1 International Conference, TestCom 2008, 8th International Workshop, FATES 2008, Tokyo, Japan, June 10-13, 2008, Proceedings*, volume 5047 of *Lecture Notes in Computer Science*. Springer, 2008.

[SMD+08] Diana Alina Serbanescu, Victoria Molovata, George Din, Ina Schieferdecker, and Ilja Radusch. Real-Time Testing with TTCN-3. In Suzuki et al. [SHUH08], pages 283–301.

[TEM08] TEMEA Project. web site of TEMEA (TEst specification and test Methodology for Embedded systems in Automobiles), 2008.