μTTCN – An Approach to Continuous Signals in TTCN-3

Jens Bräuer, Henning Kleinwechter, Andreas Leicher

(jens.braeuer|henning.kleinwechter|andreas.leicher)@carmeq.com

Abstract: Today, development and implementation of automotive embedded systems is based on a model-based process. A major problem of respective model-based toolchains can be traced back to inadequate testing tools, which often are not standardized, incomplete and cumbersome in usage. Due to these shortcomings, test specifications are neither exchangeable nor reuseable. In this paper, we propose μTTCN for test specifications of continuous embedded systems. μTTCN targets at exchangeable and reuseable test specifications based upon TTCN-3, as well as convenient handling of testcase implementations.

1 Introduction

Today, most innovations in automotive embedded systems are realised by software. The development and implementation of this software rests upon a model-based development process, which satisfies cost-effectiveness and short development cycles as imposed by the automotive market [Gri03][EKL07]. Tools such as MATLAB/Simulink/Stateflow (ML/SL/SF) [Mat06] facilitate fast development in this market. They provide structural abstractions, improved behavioural modeling, visualisation and code generation. However, a major obstacle in today's tool chain of continuous embedded systems can be traced back to inadequate support for test specification and implementation. Existing tools for test implementation often are not standardised, do not support all the necessary information of test specifications, e.g expected results of a testcase, and are cumbersome in usage. Due to these shortcomings, test specifications are neither exchangeable nor reusable. Furthermore, test implementation is time consuming and error prone.

In this paper, we propose μTTCN as an extension for the Testing and Test Control Notation (TTCN-3) language to support continuous embedded systems. The focus of the extension will be on an accelerated textual implementation of test specifications that also permits reuse and exchange.

The paper is structured as follows: Section two provides an overview of existing concepts of software test as well as a description of the testing process that is used by the model-based development team at Carmeq GmbH [car07]. Section three introduces our extension to TTCN-3. Section four gives a respective example. Related work will be presented in section five.

2 Test Concepts and their Implementation for Model-Based Development at Carmeq

A typical test process implemented by the automotive industry includes the following steps: 1. a test case specification based on the requirements is created, 2. a review for correctness of the test specification is performed, 3. the test case is implemented in the designated target environment, 4. the implementation is reviewed for correctness, 5. test cases are executed against the system under test (SUT), and 6. test results are evaluated against the test case specification. In general, the designated target environment as well as adopted development tools determine the needed test environment. For example, a typical development project at Carmeq uses ML/SL/SF for design and TargetLink for code generation. For this scenario, tests are implemented with CTE/ES [raz03] and executed with dSpace MTest [dsp06].

A major concern regarding the present test process is the diversity of development tools and target environment. Each project possibly changes the design tools and the target environment, even if it designs and implements a product with identical requirements. Consequently, testing tools often need to be exchanged. In this respect, test specifications need to be independent of the target environment in order to be exchangeable and reusable. Unfortunately, this requirement is not fulfilled with industrial deployed testing tools.

However, research provides alternatives. The Testing and Test Control Notation (TTCN-3) is a standardised testing language supporting multiple graphical representations [ETS05]. TTCN-3 and its predecessors have been extensively used within the telecommunication industry and by the European Standards Telecommunication Institute [ets07]. A key feature of TTCN-3 is the abstraction of a testcase's technical details in terms of communicating with the system under test (SUT). In technical terms, ports are used to communicate with the SUT. They hide technical details such as protocols being used or byte encodings. Those details are implemented in codecs and adapters separated from the testcase. Therefore, TTCN-3 is an ideal candidate to support the model-based development process in the automotive industry as it could support different testing targets (HIL, PIL, SIL, MIL) without modifying the testcases.

These features of TTCN-3 and the fact that it is a standardised testing language have led to several extension proposals for certain scenarios. For example, Real-time TTCN [WG97] focuses on the extension of TTCN to describe realtime constraints. TimedTTCN-3 [DGN02] extends TTCN-3 by elements to describe distributed time and introduces a logging-interface to support downstream assessment of testcases. The newest proposal, Continuous TTCN-3 [SBG06] focuses on an extension to conceptually describe continuous values, which are often used in the embedded world.
The introduction of a concept to describe continuous signals in an abstract manner is a key success factor of TTCN-3 in the embedded world.

3 Approach

In this section, an extension of TTCN-3 is suggested describing continuous signals. The domain of application was focused on the development of body comfort software. Major attributes of body comfort software are complex interfaces in terms of number of signals and complex temporal behaviour. Consequently, a test language should ease the choice of the relevant signal-subset per testcase and automatically set unused signals to default values. Also the definition of testcases containing identical test step sequences should be possible within the test language.

Due to the complexity of TTCN-3, a complete extension of the standard was not an option. Instead, we focused on the TTCN-3 Core Language and left out graphical representations and standardised interfaces (TCI, TRI). To introduce our changes, seven TTCN-3 grammar rules had to be changed and 45 new rules were introduced. This affected about 7% of the TTCN-3 Core Language grammar. Although, conceptual work was based on the complete TTCN-3 Core Language, our implementation considers only parts. The essential idea regarding the extension is a 'sample and hold' semantic. Only changes of a signal have to be defined, otherwise the current value is kept. In the following, we assume that the reader is familiar with TTCN-3. Otherwise [GHR+03] provides a short introduction.

The basis of μTTCN is the introduction of an additional kind of port, the streamport. Theoretical concepts behind it have been founded by Broy [Bro97], mapped onto testing of embedded devices and extended by Lehmann [Leh04] and were finally roughly introduced to TTCN-3 by Schieferdecker et. al [SBG06].

μTTCN only uses parts of the theory behind streamports which is outlined in the following: A stream of type T is a total function $s_T : \mathbb{R} \rightarrow T \cup \epsilon$, where $\mathbb{R}$ defines a dense time domain and T is a set of possible messages. ϵ is introduced to create a total function, so that $s(t) = \epsilon$ represents an undefined value of stream s at time t.
A TTCN-3 streamport c_T of type T (a 'channel' in terms of Lehmanns' theory), is a named variable a stream can be assigned to. Streamports were extended to define a default value α. α corresponds to the TTCN-3 value `omit` by default, if not otherwise defined by the user. Both the streamport and the stream are indexed with T to depict type-compatibility.
A set of assignments between streams and streamports is called steplet, which is defined for a specific interval of $\mathbb{R}$. Programmatic expressions in the body of a steplet are used to define sections containing different kinds of messages (T). Possible expressions defining those different sections are steps and sine/ramp interpolations. 'Sine' and 'ramp' are continuous continuations, so that one of the following statements has to hold: 1. The streamport defines an $\alpha \neq$ `omit` or 2. a previous section of the stream defines a value v ($v \in T$).

Based on [Con04, 72ff], the sections of the stream (function) may be defined as follows:

- `alpha @ t1 := x1`, interpreted as $\alpha(t_1) = x_1$
- `alpha @ t0 := ramp(x1, d)`, interpreted as $\alpha(t) = m \cdot t + n$ with $m = \frac{x_1 - x_0}{t_1 - t_0}, n = \frac{t_0 \cdot x_1 - t_1 \cdot x_0}{t_1 - t_0}, t_1 = t_0 + d$
- `alpha @ t0 := sine(x1, d)`, interpreted as $\alpha(t) = m \cdot \sin(n \cdot t + o) + p$ with $m = \frac{x_0 - x_1}{2}, n = \frac{\pi}{t_1 - t_0}, o = \frac{\pi}{2} - n \cdot t_0, p = \frac{x_0 + x_1}{2}, t_1 = t_0 + d$

For an in depth introduction to the original theory of streams, channels and assignments see [Leh04, chap. 3]. Efforts to completely (re)unite this theory with our extension in a mathematical sense have not been undertaken yet.

We will use a small example to illustrate the algorithm defined by the μTTCN extension: Port and component definitions have been extended to provide the ability to define default values α.

```
type port T1 stream { out float := 0.0 }
type component ECU {
  port T1 C1_T1 := 10.0;
  port T1 C2_T1;
}
```

The above example defines the type `T1` of a streamport. This concretises set T, so streams operating on ports of type `T1` are functions $s_{T1} : \mathbb{R} \rightarrow \mathbb{R}$. The SUT described (the component definition) here consists of two streamports of type `T1` named `C1_T1` and `C2_T1`. The default value α may be overwritten in the component definition, so port `C1_T1` redefines $\alpha = 10.0$ whereas a default value of 0.0 applies to `C2_T1`. A steplet (a set of assignments, defined in sections) operating on the component may look like:

```
steplet S(12.0)() runs on ECU := {
    C2_T1@1 := sine(42, 2.5);
    C1_T1@4 := 3.0;
    C2_T1@6 := ramp(23, 5);
}
```

The specific time the steplet shall operate on the streamports is limited by the value given in the first parentheses (12.0 seconds in the example). Within the second pair of parentheses, the signature of parameters may be defined analogous to function parameters. Using the above steplet `S` with the μTTCN algorithm yields the following functions s_1 and s_2 which are then assigned to `C1_T1` and `C2_T2`:

$$s_1(t) = \begin{cases} \alpha & \alpha = 10.0 \text{ and } t \in [0, 2) \\ 3.0 & t \in [2, 12] \end{cases}$$

$$s_2(t) = \begin{cases} \alpha & \alpha = 0.0 \text{ and } t \in [0, 1) \\ m \cdot \sin(n \cdot t + o) + p & t \in [1, 3.5) \\ 42 & t \in [3.5, 6) \\ m \cdot t + n & t \in [6, 11) \\ 23 & t \in [11, 12] \end{cases}$$

Sections of the (stream) functions are defined along the time line for multiple ports. Different sections are merged to form total functions assigned to the appropriate port. Thus, signal values are described in a continuous manner rather than in discrete data-points. Consequently, it is possible to define data independent of the used sample rate. Nevertheless, discrete values are often needed for testing, so new statements were introduced to set the sample rate from within TTCN-3[1].

[1] Due to space limitations, we will not discuss these statements in the paper.

Comparing μTTCN to TTCN-3, a steplet can be seen as a hybrid between an 'altstep' and a 'template' expression. On the one hand, it defines data and allows modification (kind of inheritance) like a 'template'. On the other hand, it may be bound to a component and operates on different ports like an 'altstep'.

Code part	Advanced expression	Comment
`a@X := v;` `a@Y := default;`	`a@[X,Y] := v;`	Toggle signal
`a@X := v;` `b@X := v;`	`(a,b)@X := v;`	Same value on multiple ports
`a@X := v;` `b@X := w;`	`@X( a:=v, b:=w );`	Different values but same time

Table 1: μTTCN advanced expressions

Furthermore, μTTCN allows the inclusion of one steplet into another. Also, three advanced types of expression have been defined, allowing to parenthise streamports and values and easily 'toggle' values. This advanced forms were introduced to avoid redundances and to shorten the program code. Table 1 shows code parts, which can be abbreviated using the short forms. Multiple steplets may be used within one testcase, executed one after another. Each steplet defines it's own local time starting at zero.

Relating μTTCN's above theory to practice, the chosen algorithm allows an implicit definition of the testcases' interface. Taken that all streamports are assigned to default values $\alpha \neq$ `omit`, only continuations have to be defined.
An algorithmic merge of continuations is possible as the number of valid expressions is limited. In return, the tester is allowed to define functions along a timeline in steps, e.g. first speed rises like a sine, then a window opens linear, and so on. From our own experience, this corresponds to the way testcases are defined in practice, in contrast to total definitions of functions per streamport as imposed by other tools.
The set of allowed expressions was selected on the basis of requirements from practical work of testers: When testing body comfort software, there is no need for sophisticated functions. In contrast, the focus is laid on a set of practical relevant expressions and the ease of usage.

4 Example

This section will demonstrate the usage of μTTCN to test a subsystem of body comfort software: the flasher logic. The behaviour is modeled in ML/SL/SF and shown in Figure 1. Subsystems, shown as boxes in the figure, hide implementation details and structure the model. The left subsystem is realizing the preprocessing, while the subsystems in the middle implement the functional logic. The right one controls single flashlights.

In general, flashers are not only used to indicate directions but also to indicate hazards (hazard flashing) and crash detection. As an example, a requirement could impose that

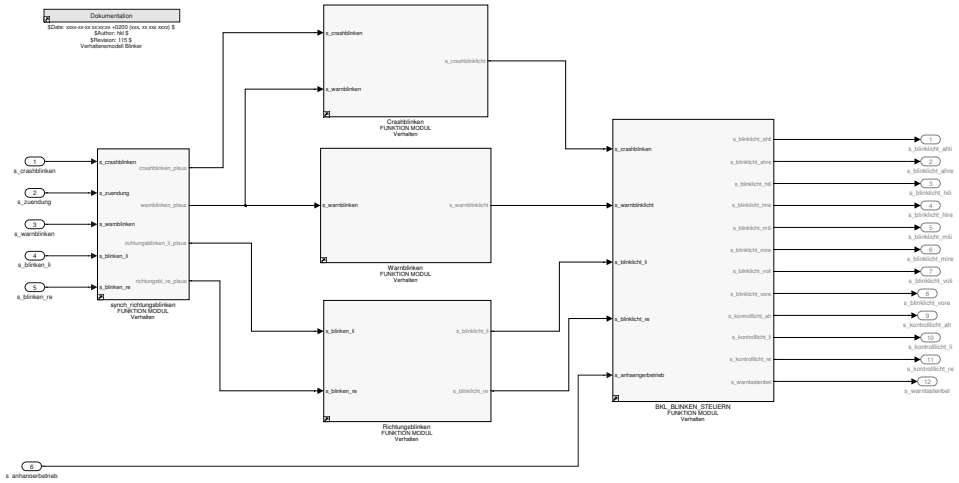

Figure 1: Flasher model

after a crash, hazard flashing has first to be turned on and then off to deactivate flashing. Within the test process (introduced in Section 2) a testcase might be defined as follows:

```
Target: Crash flashing implemented according to req. XX
Environment: Trailer mode disabled
Dependencies: Parameter phase_light, phase_dark

[0.5s] Ignition
[2s] Crash detected
[4s] Hazard flash activated
[10s] Hazard flash deactivated
```

The following μTTCN-code describes the stimuli needed to observe this behavior in the software:

```
testcase HazardFlash001() runs on ECU_MdlSubsys {
  (12)() := {
    initialize()@0;
    s_crashblinken@2 := true;
    s_warnblinken@4 := true;
    s_warnblinken@10 := false;
  }
}
```

The testcase uses an inline steplet-definition to generate stimuli for 12 seconds. At $t = 0$ the steplet named 'initialize' is included. It takes the appropriate steps to turn the ignition on. Two seconds after the test start, crash flashing is activated, followed by toggling the hazard flashing at 4 and 10 seconds.

When comparing the testcases' definition in text and μTTCN, one may notice the exact matching between text and program code. Due to the sectional definition of the stream,

this matching is also observed in real-world testcases and eases the review of the testcases' implementation.

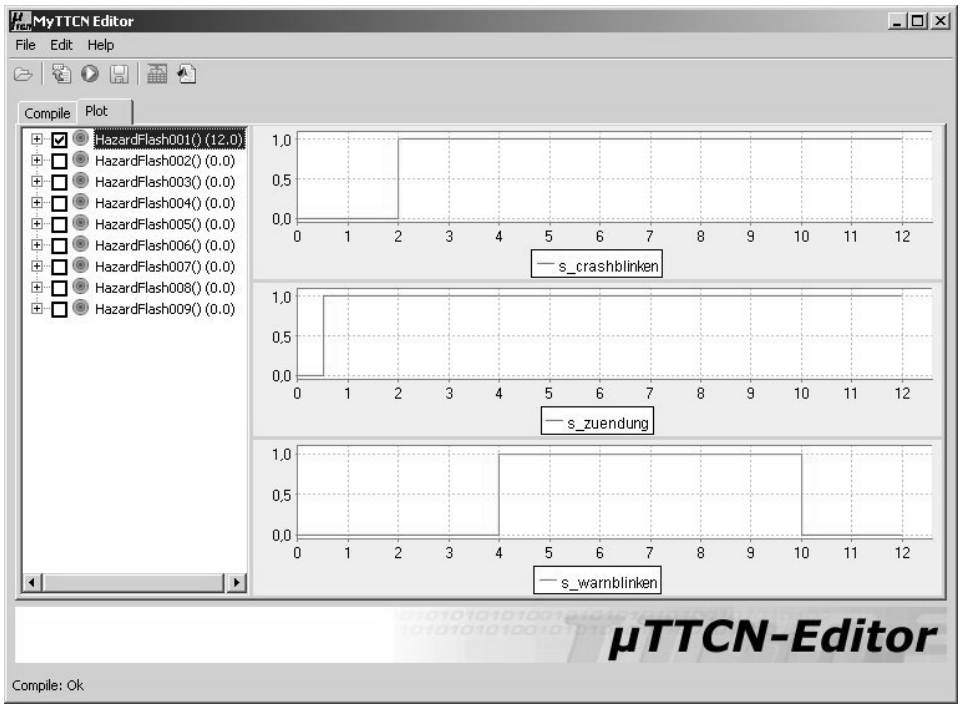

Figure 2: μTTCN-Editor - the tool implementing μTTCN

The μTTCN-Editor, which is shown in Figure 2, implements the μTTCN-compiler for testcase specification and implementation. The main area of the editor consists of two tabs, one showing the compiler output and the other (depicted in the figure) visualising the described stimuli. Visualisation helps the tester to avoid bugs when implementing testcases.

The testcase shown is typical for functional tests of body comfort software. Testcases are rather simple and there is no need for complex signals. Software complexity in this area mostly arises from the number of independent functions (like crash, hazard or direction flashing) and the variability of installed equipment. In addition, different local laws lead to a situation where functions are basically the same but behaviour is totally different. Also most values, like the length of a flash, can be parametrised.

These are the reasons why μTTCN supports parametrisation, externalisation and modification of sequences. Not shown in the above example but present in practice is the huge interface of the software. Some subsystems have about 100 inputs and more than 50 outputs, which imposes the need for centralised definition of default values.

5 Related Work

Testing embedded systems in general and in the automotive domain in special has been subject of many cooperations between industry and research institutions. Due to the complexity of the development of embedded automotive systems, project results often are not reused as they are fitted to special needs of the project partners.

The SiLEST project [DTBI+06] for instance focuses on test automation for model-based software development, using a proprietary XML-Format to describe testcases. Another XML-Format has been defined by the IMMOS project [DdF+06] and as part of the development of Time Partition Testing (TPT) [Leh04].

At the moment, research is ongoing in the field of test result assessments for embedded systems. While one approach is the development of advanced algorithms to compare actual results with reference data [FI06], another is the description of characteristics [GW07]. Statements on how to test software modeled with the open source framework Modelica [Mod06] are part of [GSE06].

Research, related to the extension of TTCN-3 and applicable to embedded systems, has already been introduced in section 2. As part of the development of μTTCN, μTTCN has been mapped onto language elements introduced in Continuous TTCN-3 [SBG06]. In case further progress in research and implementation of Continuous TTCN-3 is made, testcases reactive to the behavior of the SUT[2] will be possible, using this mapping. The specification of the expected system output will be facilitated, too.
Comparing Continuous TTCN-3 and μTTCN two issues arise in which both approaches diverge: First the aim of the language design seems to be different. While Continuous TTCN-3 strives to use TTCN-3 as an 'execution environment' for testcases modeled in TPT and therefor may rely on code generation, one main goal of μTTCN development was code simplicity and readablily. Second, the approach on how to define the signals is different. μTTCN favours a stepwise definition, which is close to a textual description, e.g. 1st turn A on, then 2nd increase B to X and so on. Continuous TTCN-3 in contrast uses function-like definitions, which have to be total in the considered time interval.
However, a combination of Continuous TTCN-3's or μTTCN's capabilities with the downstream assessment of testcases, as introduced by TimedTTCN-3 [DGN02], would have great potential.

6 Conclusion

The introduced extension μTTCN has been designed and implemented as part of a diploma thesis in cooperation with Carmeq. Conceptual work on μTTCN was based on the complete TTCN-3 core language, the implemented tool however realises only parts of it. The tool has been used in industrial projects within the last few months. It turned out that a textual definition itself is not a problem at all and after an initial training, 50-75% time

[2] Testcases implemented in μTTCN and then mapped onto Continuous TTCN-3 will be 'reactive' in the sense that testcases stop when a deviation of expected and actual result of the SUT is detected.

savings were estimated when implementing testcases. Time savings where estimated by testers comparing former used test implementation by means of CTE/ES with μTTCN. Estimations took place four to six weeks after the introduction of μTTCN and after implementation of severals hundred testcases. An increase of errors due to the increased degrees of freedom was not detected.

Another error-prone and time-consuming task in testing embedded systems is the validation of the system reaction, not yet supported by muTTCN. There is great potential in further reducing test efforts (in terms of time) when solving the issue of automated signal validation in a way applicable to practice.

TTCN-3 with it's abstract testcase definition seems to be a good candidate for the exchange of testcase-specification. While there have already been field studies to apply TTCN-3 to the automotive domain [JT06] [Hen04], the lack of concepts for continuous signals prevents broad usage. μTTCN clearly considers these needs when testing models of functional body comfort software. Although it may not be applicable to embedded domains, which have a strong need for complex signal definitions. Instead it may be seen as one practical approach using the right basis, but still needing more work.

References

[Bro97] Manfred Broy. Refinement of Time. In *Transformation-Based Reactive Systems Developement*, volume 1231 of *Lecture Notes in Computer Science*, pages 44–63, Berlin / Heidelberg, 1997. Springer.

[car07] Carmeq. `http://www.carmeq.com`, 2007. Accessed: 25.01.2007.

[Con04] Mirko Conrad. *Auswahl und Beschreibung von Testszenarien für den Modell-basierten Test eingebetteter Software im Automobil*. Dr. Ing. Dissertation, Technische Universität Berlin – Fakultät IV - Elektrotechnik und Informatik, 2004.

[DdF+06] DaimlerChrysler, dSPACE, Fraunhofer FIRST, FZI Karlsruhe, and Universität Karlsruhe. IMMOS – Integrierte Methodik zur modellbasierten Steuergeräteentwicklung. `http://www.immos-projekt.de/`, 2004–2006. Accessed: 19.01.2007.

[DGN02] Zhen Ru Dai, Jens Grabowski, and Helmut Neukirchen. TimedTTCN-3 – A Real-Time Extension for TTCN-3. In *Testing Internet Technologies and Services – International Conference on Testing of Communicating Systems, 19.03-22.03.2002, Berlin*, pages 407–424, Dordrecht, Niederlande, 2002. Kluwer Academic.

[dsp06] MTest - Feature list. `http://www.dspace.de/shared/data/pdf/flyer2006/dspace_flyer2006_MTest_1%-3_en.pdf`, 2006. Accessed: 25.08.2006.

[DTBI+06] DLR, TU-Berlin, IAV, Fraunhofer First, and webdynamix. SiLEST – Software in the Loop for Embedded Software Test. http://www.silest.de, 2004–2006. Accessed: 04.10.2006.

[EKL07] Hans-Peter Erl, Sascha Kirstan, and Lehrstuhl Software & Systemsengineering. Kosten-/Nutzenanalyse der modellbasierten Softwareentwicklung im Automobil. Study, Arthur D. Little and Technische Universität München, January 2007. Study not published yet.

[ETS05] ETSI. Testing and Test Control Notation. ETSI European Standard ES 201 873-X, 7 Parts, June 2005.

[ets07] European Telecommunications Standards Institute. `http://www.etsi.org`, 2007. Accessed: 25.01.2007.

[FI06] Fachgebiet Elektronische Mess- und Diagnosetechnik, TU-Berlin and IAV GmbH. Automatisierte Auswertung von Fahrzeugmessdaten mit Hilfe moderner Mustererkennungsmethoden. `http://www.mdt.tu-berlin.de/forschung/projekte/fahrzeugmessdaten`, 2006. Accessed: 23.01.2007.

[GHR+03] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, and Colin Willcock. An introduction into the testing and test control notation (TTCN-3). *Computer Networks*, 42(3):375–403, June 2003.

[Gri03] Klaus Grimm. Software Technology in an Automotive Company – Major Challenges. In *ICSE 2003: 25th International Conference on Software Engineering*, pages 498–503, Los Alamitos, USA, 03.05-10.05 2003. IEEE Computer Society.

[GSE06] Ulrich Grude, Friedrich Wilhelm Schröer, and Peter Enskonatus. TTCN–3 for .NET. In *T3UC 2006 – TTCN-3 User Conference, 31.05-02.06.2006, Berlin – Presentation*, 2006.

[GW07] Carsten Gips and Hans-Werner Wiesbrock. Notation und Verfahren zur automatischen Überprüfung von temporalen Signalabhängigkeiten und -merkmalen für modellbasiert entwickelte Software. In Mirco Conrad, Holger Giese, Bernhard Rumpe, and Bernhard Schätz, editors, *MBEES 2007 – Modellbasierte Entwicklung eingebetteter Systeme, 15.-18.01.2007, Dagstuhl*, pages 51–60, 2007.

[Hen04] Stefan Hendrata. Standardisiertes Testen mit TTCN-3. *automotive*, (9–10):64–65, 2004.

[JT06] Georg Janker and Dirk Tepelmann. MOST® goes TTCN-3: Putting two worlds together. In *T3UC 2006 – TTCN-3 User Conference, 31.05-02.06.2006, Berlin – Presentation*, 2006.

[Leh04] Eckard Lehmann. *Time Partition Testing – Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen*. Dr. Ing. Dissertation, Technische Universität Berlin – Fakultät IV - Elektrotechnik und Informatik, 2004.

[Mat06] MATLAB/Simulink/Stateflow. `http://www.mathworks.com/`, 2006. Accessed: 24.08.2006.

[Mod06] Modelica Association. Modelica. `http://www.modelica.org/`, 2006. Accessed: 23.01.2007.

[raz03] Classification Tree Editor for Embedded Systems. `http://www.razorcat.de`, 1998–2003. Accessed: 30.10.2006.

[SBG06] Ina Schieferdecker, Eckard Bringmann, and Jürgen Großmann. Continuous TTCN-3: testing of embedded control systems. In *SEAS '06: International Workshop on Software engineering for automotive systems*, pages 29–36, New York, 2006. ACM Press.

[WG97] T. Walter and J. Grabowski. Real-time TTCN for Testing Real-time and Multimedia Systems, 1997.