Testing of Embedded Control Systems with Continous Signals

Ina Schieferdecker
Technical University Berlin
Fraunhofer FOKUS
schieferdecker@fokus.fraunhofer.de

Jürgen Großmann
DaimlerChrysler AG
juergen.grossmann@daimlerchrysler.com

Abstract
The systematic testing approaches developed within the telecommunication domain for conformance and interoperability testing of communication protocols have been extended and broadened to allow the testing of local and distributed, reactive and proactive systems in further domains such as Internet, IT, control systems in automotive, railways, avionics and alike. With the application of these testing principles it became apparent that the testing of systems with continuous systems is different to that of discrete systems. Although every continuous signal can be discretized by sampling methods (and hence mapped to the discrete signal paradigm), abstraction and performance issues in this setting become critical. This paper investigates the different options to support tests of embedded control systems in the framework of TTCN-3.

1 Introduction

Control systems in real-time and safety-critical environments are hybrid: they use both discrete signals for communication and coordination between system components and continuous signals for monitoring and controlling their environment via sensor and actuators. A test technology for control systems needs to be able to analyze the functional and timed behaviour of these systems being characterized by a set of discrete and continuous signals and their relations. While the testing of discrete controls is well understood and available via the standardized test technology TTCN-3[9], the specification-based testing of continuous control and of the relation between the discrete and the continuous control is not. A variety of test methods exist, which address different aspects – however, the concepts have not yet been consolidated. This is also shown by the fact that consortia like AutoSar, ESA or the MOST Forum are looking for an appropriate test technology. Appropriateness of a test technology is to be understood from different perspectives:

1. The adequateness of testing concepts and their syntactical presentation (textual or graphical) to the user
2. The applicability to various elements of the control systems, e.g. to discrete, continuous, local and distributed components
3. The applicability to various target systems, e.g. application components, communication components, control components, etc.

4. The portability to various target platforms, e.g. to platforms of the component providers and component integrators as well as to platforms of the test device vendors
5. The usability for different development approaches, e.g. HIL, SIL, MIL, etc.
6. The usability for different testing kinds, e.g. component level tests, system level tests, acceptance level tests, functional and non-functional tests, regression tests

TTCN-3 as such addresses already the majority of these aspects: it has constructs for

- message-based and procedure-based communication (making it applicable for asynchronous, decoupled and synchronous systems in client-server or peer-to-peer architectures),
- detailed test behaviours which allow to describe sequential, alternative, looping or parallel testing scenarios
- local and distributed test setups, so that physically distributed and/or heterogeneous systems with different access technologies can be tested
- an open and adaptable architecture to customize an executable test system to various target systems and target test platforms
- different presentation formats to highlight specific aspects of the test definitions or make it better fit to an application domain
- accessing different data formats of the system/component to be tested

It is a platform-independent, universal and powerful test specification technology, which allows selecting the abstraction level for the test specifications – from very abstract to very detailed, technical test specifications. It is also a test implementation language, which precisely define how to convert the abstract test specifications into executable tests. Also the tracing and evaluation of test results allows different presentation formats for textual and graphical logs. However, TTCN-3 lacks continuous signals as a language construct – which limits the power of TTCN-3 with respect to the adequateness of test concepts, the applicability to various target systems and the usability for different development approaches in the context of control systems. This paper discusses various options for testing continuous controls and analyses the use/extension of TTCN-3 for testing automotive control systems, in particular.

The paper is structured as follows: Section 2 reviews current approaches of testing control systems in the automotive domain. Section 3 discusses the basic concepts for testing continuous controls. Section 4 analyses how TTCN-3 can be used and extended to address the testing of continuous controls. Section 5 presents an example. Conclusions finish the paper.

2 Related Work

Established test tools in automotive from e.g. dSPACE [2], Vector[3] etc. are highly specialised for the automotive domain and come usually together with a test scripting approach which directly integrates with the respective test device. However, all these test definitions special to the test device and by that not portable to other platforms and not exchangeable. Some, e.g. ETAS [4], have already adopted TTCN-3, however, have taken it as it is without analysing further if the specific needs of the automotive domain are better to be reflected in the language itself. Hence, other platform independent approaches have been developed such as CTE [5], MTEST [6], and TPT [1]. CTE supports the classification tree method for high level test designs, which are applicable to partition tests according to structural or data-oriented differences of the system to be tested. Because of its ease of use, graphical presentation of the test structure and the ability to generate all possible combination of tests, it is widely used in the automotive domain. However, the detailed specification of test procedures and the direct generation of executable tests are out of consideration. MTEST is an extension of CTE to enable also the definition of sequences of test steps in combination with the signal flows and their changes along the test. By that it adds continuous signals to the test description, has however only limited means to express test behaviours which go beyond simple sequences, but are typical for control systems. This is addressed by TPT, which uses an automaton based approach to model the test behaviour and associate with the states pre- and post-conditions on the properties of the tested system (incl. the continuous signals) and on the timing. In addition, a dedicated run-time environment enables the execution of the tests. TPT is a dedicated test technology for embedded systems controlled by and acting on continuous signals, however, the combination with discrete control aspects is out of consideration. Therefore, this paper discusses first ideas on how to combine TPT concepts with TTCN-3 so as to gain the most from both approaches. It has been decided to adopt TPT concepts within TTCN-3 in order to make also advantage of TTCN-3 being a standardized test technology and having the potential to included dedicated concepts for continuous signals in later versions of TTCN-3.

3 TPT Concepts

TPT [1] uses for the modelling of test cases state machines combined with stream processing functions [11]. Every state describes a specific time quantified phase of a test. The transitions between states describe possible moves from one phase into the other. Every path through the automaton describes a possible execution of the test. The behaviour within the phases is formalized by expressions on the input and output flows of the channels the test is using. As these expressions are formal and not easy to understand, the graphical automaton uses informal annotations to the transitions and phases which explain the purpose, but do not give the precise definition which is hidden in the formal definitions. Semantically, an execution of such a hybrid system is a sequence of states and transitions, where the states describe the behaviour of the system up until the next transition is firing. This is reflected by the semantic concept of a component, which receives inputs via input channels and produce outputs via output

channels. A *channel* is a named variable, to which streams can be assigned. A *stream* s of type T is a total function s: $\mathbb{R} \rightarrow T \cup \varepsilon$, where $\mathbb{R}$ denote real numbers and represent a dense time domain, and ε[1] represents the absence of a value, meaning that for $s(t) = \varepsilon$ that there is no value of s at time t. Let C be a finite set of channels. Every channel $\alpha \in C$ has a type T_α. An *assignment* c of the channels C assigns to every channel $\alpha \in C$ a stream c_α: $\mathbb{R} \rightarrow T_\alpha \cup \varepsilon$. $\vec{C}$ is the set of all possible assignments of C.

A *component* is a relation F: $\vec{I} \leftrightarrow \vec{O}$, which defines the assignments of output channels $\vec{O}$ in dependence of the assignments of input channels $\vec{I}$. This relation has to have the time causality property: it exists a $\delta>0$, so that for all i, i' $\in \vec{I}$ and o, o' $\in \vec{O}$ with $o =_{<0} o'$, (i,o) $\in$ F and (i',o') $\in$ F and for all $t \geq 0$ the following holds:

$i =_{\leq t} i'$ and $o =_{\leq t-\delta} o'$ implies that $o =_{\leq t} o'$

Please note that $c =_{<t} c'$ and $c =_{\leq t} c'$ denotes the fact, that the assignments o and o' are equal in the time interval $(-\infty,t)$ and $(-\infty,t]$, resp.

TPT is based on the following model of hybrid systems: H = (V, E, src, dest, init, I, O, behaviour, cond) with

- dest, init) being a finite automaton having a finite set of states V, a finite set of transitions E, a function src: E → V, which assigns to every transition the source state, a function dest: E → V, which assigns to every transition the target state, a special initial state init ∈ V
- (I, O) being a signature with a finite set of input channels I, a finite set of output channels O and $I \cap O = \emptyset$, a function behaviour: V → $\vec{I} \leftrightarrow \vec{O}$ which assigns to every state v ∈ V a component $\vec{I} \leftrightarrow \vec{O}$, which is called behaviour, and a function cond: E → ($\overrightarrow{I \cup O} \rightarrow \mathbb{R} \rightarrow \mathbb{B}$), assigning to every transition e ∈ E a condition cond ∈ $\overrightarrow{I \cup O} \rightarrow \mathbb{R} \rightarrow \mathbb{B}$ for which the inputs O are delayed effective and where $\mathbb{B}$ are the Boolean values. The transition fires at the earliest (V, E, src, time when cond is true.

A component defines the values of (hybrid H): $\vec{I} \leftrightarrow \vec{O}$ for which holds: Let i ∈ $\vec{I}$ and o ∈ $\vec{O}$. Then is (hybrid H)(i,o) iff there is a finite or infinite sequence of states v ∈ V: φ= $v_1 \xrightarrow{e1} v_2 \xrightarrow{e1} \ldots$ where

- v_1= init and $src(e_n)=v_n$ and $dest(e_n)=v_{n+1}$, i.e. it is a correct path of the automaton H
- the concatenation behaviour(e1) –cond(e1)-> behaviour(e2) … is a sequentialation of H, and
- every other path φ ‘ which fulfils the above two properties is “slower” than φ

---

[1] ε will later be mapped to omit in TTCN-3.

Please refer to [1] for the complete definitions as only an outline of the basic semantics of TPT can be given here.

4 TTCN-3 for Hybrid Systems with Discrete and Continuous Signals

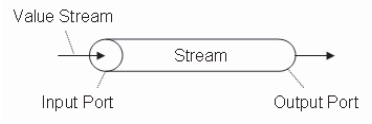
In order to enable the definition of tests for hybrid systems based on the semantic model of TPT, TTCN-3 needs to be extended with

- a concept of streams and channels (input and output channels and local channels),
- a concept of continuous time and sampling rates, and
- a concept of assignments to/evaluations of channels by direct definitions and by time partitions.

4.1 Streams and stream ports

The concept of channels is mapped to the concept of ports by adding an additional port kind for stream-based ports:

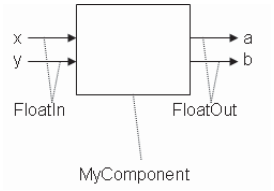
```
type port FloatIn stream {
        in float
}
type port FloatOut stream {
        out float
}
```



Stream ports are allowed to have an in or an out direction only, but not both (and also not an inout definition). Although TPT proposes to use float, integer and boolean for streams only, we leave it open to use any type, i.e. also user defined, structured type, as the basis of a stream-based port.

A test component having three input and output channels is then defined by a normal TTCN-3 test component type:

```
type component MyComponent {
        port FloatOut a,b;
        port FloatIn x,y
}
```



These leaves also the possibility open to combine continuous and discrete (i.e. message- or procedure-based) ports as the interface of a component type. In order to use streams for internal calculations (like TPT is using local channels), streams are defined as separate language concept:

```
stream float s;
```

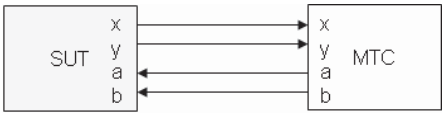
4.2 Evaluation of input streams

The essential concept is to find a way to define the generation of an output stream and the analysis of an input stream. For discrete ports, TTCN-3 uses combinations of send/receive, call/getcall, reply/getreply and raise/catch for the generation of outputs and analysis of inputs. The direct definition of a test uses a set of equations, which define how the local and output channels behave in dependence of the input channels: the set consists of equations g_{c1} ; g_{c2} ; … ; g_{cn}, where for each channel $c_i \in O \cup L$ an equation in the form $c_i(t) = E_i$ has to exist. The expressions E can use channels from $I \cup O \cup L$, where it is required that there is no cyclic dependency between the channels, but a delayed effectiveness, i.e. the set of equations can be resolved directly by term evaluation. This condition holds if for every $t \in \mathbb{R}$, the values of the output channels are defined by use of the values of the input channels for $t' \leq t$ and by use of the values of the output channels for $t'<t$ only. An example for such an equation system is given in the following:

```
x(t) = (t < 2) ? a(t) : x(t-2)
l(t) = b(t) + 2
y(t) = l(t/2) * sin(2 * t)
```

where l is a local stream.

Conceptually, we use this equation system as a characterization of the system under test (SUT), i.e. the input streams are to be interpreted as the inputs to the SUT which is required to react with outputs along the definition of the output stream:



This is reflected with the following testcase header

```
testcase MyTestCase() // a specific testcase
runs on MyComponent   // running on the main test component
system MyComponent    // testing a system of MyComponent
{ … }
```

Please note that the interpretation of the test system interface (defined with the system clause) is interpreted from the test system perspective, i.e. a and b are outputs from the test system and finally inputs to the system under test. x and y are the outputs from the SUT and inputs to the test system and being analyzed there.

The matching for the stream uses the equation system as proposed by TPT. The expressions in TPT use elements which are also available in TTCN-3. i.e. values, constants, type conversion (in contrast to TPT, TTCN-3 uses explicit conversion functions and not type casts), arithmetical operators and boolean operators. The conditional expression (_)?_:_ operator should be added to TTCN-3 in order to ease the definition of the expressions.

```
function Eval_DependentStream() runs on MyComponent {
        stream float l;        // local stream
        sample t(0.2); // sample rate for evaluation
        x@t == (t < 2) ? a@t : x@(t-2);
        l@t == b@t + 2 ;
        y@t == l@(t/2) * sin(2 * t) ;
}
```

For that, the @- and the ==-operator are introduced on stream ports: the @-operator represents the value of the stream at time t, which is taken from the sample rate defined in the sample statement, i.e. streams are discretized as defined by TPT for their evaluation. The ==-operator is the operator to define the equation system for the stream characterization. Whether evaluation is performed by an offline or online analysis depends on the test system realization and is currently outside the scope of consideration. Potentially, it could be useful to make it explicit in the test specification to require an online or offline evaluation, but this is for further study. In addition, a #-operator is used to get access to the value, which was in the stream before the current evaluation with the @-operator. Hence, x#(t+0.2) equals x@t as 0.2 is the sampling rate of t. Please note, that TPT uses for the @-operator a dot notation "." and for the #-operator a "-@" notation. The "." notation was not chosen here, as the dot has already a different meaning in TTCN-3. The "-@" notation was not chosen in order to have single sign operators only.

Another case is a set of streams that are independent from other streams, so that the equation is defined as an expression without referring to other streams:

```
function Eval_IndependentStream() runs on MyComponent {
        sample t(0.2); // sample rate for evaluation
        x@t == 2 * t - 3;
}
```

4.3 Generation of output streams

In the case that the test system has to provide streams as inputs to the SUT only, the equation systems as introduced above can be used. However, this time the equations are independent from other streams, i.e. the stream is defined by an equation and generated. Hence, the assignment-operator ":=" is used instead of the evaluation-operator "==". Both equation systems presented in Section 4.2 can e.g. be used (with an assignment operator and with output channels):

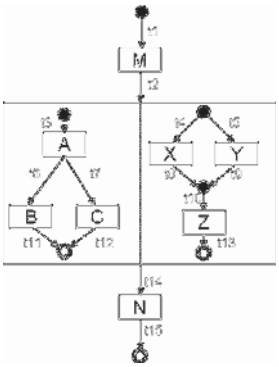
```
function Generate_IndependentStream() runs on MyComponent {
        sample t(0.2); // sample rate for generation
```

```
        a@t := 2 * t - 3;
}

function Generate_DependentStream() runs on MyComponent {
        stream float l;        // local stream
        sample t(0.2); // sample rate for evaluation
        a@t := (t < 2) ? x@t : a@(t-2);
        l@t := y@t + 2 ;
        b@t := l@(t/2) * sin(2 * t) ;
}
```

4.4 System characterization by time partitions

Test cases with time partitions use the model of hybrid systems introduced in Section 0. TPT state machines are represented by initial and final states, states with associated functions as defined in Section 4.2 and 4.3, junctions and parallelization of behaviours. The test components in TTCN-3 have sequential behaviours. Several test components are used for parallel behaviours. Control structures with conditionals and loops can be used to represent the junctions. The central question is how to combine evaluation and generation functions in the states with the control flow of the state machines. An example TPT definition is given on the right hand side.



We assume that every transition t has a condition c_t and for every state there is a function characterizing the continuous behaviour in that state as defined in Section 4.2 and 4.3. We propose two new TTCN-3 statements:

- The **try**-statement to check the fulfilment of the transition preconditions. It can check several conditions and is successful if one of the conditions evaluates to true at the current point in time.
- The **carry**-**until**-statement to perform a behaviour (a continuous, discrete or hybrid one) up until the given condition becomes true.

The evaluation of the condition is done according to the current sample rate. These two statements allow to define junctions/states with several enabling conditions (such as t4 and t5) and states with several outgoing transitions (such as A) although this may require the double check of a condition (such as to enable the right hand parallel part and to select the X or the Y branch):

```
testcase TPT_TestCase()
runs on MyComponent system MyComponent {
        sample t(0.2);
        try {
        [] c_t1 {
                carry { M() } until c_t2;
```

```
                carry { FPar() } until c_t14;
                carry { N() } until c_15 ;
        }
}

function FPar() runs on MyComponent {
        var MyComponent l,r;
        l:= MyComponent.create;
        r:= MyComponent.create;
        l.start(FPar_l());
        r.start(FPar_r());
}
function FPar_l() runs on MyComponent {
        try {
        [] c_t3 {
                carry { A() } until (c_t6 or c_t7);
                try {
                [] c_t6 { carry { B() until c_t11: }
                [] c_t7 { carry { C() until c_t12; }
                }
        }
}
function FPar_r() runs on MyComponent {
        try {
        [] c_t4 {
                carry { X() } until (c_t8);
           }
        [] c_t5 {
                carry { Y() } until (c_t9);
           }
        }
        try {
        [] c_t 10 {
                carry { Z() } until (c_t13);
           }
        }
}
```

The try and carry-until statements assure that all control cases being proposed by TPT can be represented. We will further analyse the usage of these statements also in combination with the other TTCN-3 statements.

5 Summary

This paper reviews the requirements for a test technology for embedded systems, which use both discrete signals (i.e. asynchronous message-based or synchronous procedure-based ones) and continuous flows (i.e. streams). It compares the requirements with the only standardized test specification and implementation language TTCN-3 (the Testing and Test Control Notation [9]). While TTCN-3 offers the majority of test concepts, it has limitations for testing the aspects of continuous streams. Therefore, the paper reviews current approaches in testing embedded systems and identifies commonalities between TTCN-3 and TPT (the Time-Partition-Test method [1]) in terms of abstractness and platform-independence.

Subsequently, the paper investigates ways to combine TPT concepts with TTCN-3, so as to preserve the standard base of TTCN-3 and extend it to continuous signals. For that, TTCN-3 is being extended with concepts of streams, stream-based ports, sampling, equation systems for continuous behaviours and additional statements that allow control flows of continuous behaviours. The paper demonstrates the feasibility of the approach by providing a small artificial example. Future work will further refine the inclusion of TPT concepts into TTCN-3 and consider e.g. the combination of discrete and continuous control within one test behaviour function on a test component instance having both discrete and continuous ports. The concepts will be implemented in a TTCN-3 tool set [12] and applied to a real case study in the context of CAN-based engine controls in a car.

6 References

[1] E. Lehmann: Time Partition Testing, Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen, Promotion, Technischen Universität Berlin, November 2003.

[2] ControlDesk Test Automation Guide For ControlDesk Version 1.2. dSPACE GmbH, Paderborn (D), Sept. 1999

[3] B. Tettenborn, A, Leuze, S. Meißner: PXI basierendes Funktionstestsystem für Motorsteuergeräte im Rennsport, White Paper, May 2004.

[4] F. Tränkle: Testing Automotive Software with TTCN-3, 2nd TTCN-3 User Conference, June 6-8, 2005, Sophia Antipolis, France.

[5] M. Grochtmann, K. Grimm, J. Wegener: Tool-Supported Test Case Design for Black-Box Testing by Means of the Classification-Tree Editor. Proc. of 1. European Int. Conf. on Software Testing, Analysis and Re-view (EuroSTAR '93), London (GB), S. 169-176, Oct. 1993

[6] M. Conrad. Modell-basierter Test eingebetteter Software im Automobil: Auswahl und Beschreibung von Testszenarien. Dissertation, Deutscher Universitätsverlag, Wiesbaden (D), 2004

[7] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, I. Fey: Model-based Testing of Embedded Automotive Software using MTest. SAE World Congress 2004, Detroit (US), März 2004

[8] ETSI : TTCN-3 Homepage, http://www.ttcn-3.org, June 2005.

[9] ETSI ES 201 873-1 V3.1.1, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, Sophia Antipolis, France, July 2005.

[10] J. Zander, Z.R. Dai, I. Schieferdecker, G. Din: From U2TP Models to Executable Tests with TTCN-3 - An Approach to Model Driven Testing, TestCom'05, Canada, Montreal.

[11] M. Broy. Refinement of Time. In Transformation-Based Reactive System Development, LNCS 1231, Springer-Verlag, 1997.

[12] Testing Technologies IST GmbH: TTworkbench v1.3, http://www.testingtech.de, Oct. 2005.