Specification-Based Testing of Embedded Control Systems

Ina Schieferdecker, TU Berlin/Fraunhofer FOKUS, Berlin
Jürgen Großmann, DaimlerChrysler AG, Berlin

Zusammenfassung

Eingebettete Systeme spielen in vielen Industriebereichen bei der Realisierung von komplexen Steuer- und Regelfunktionen eine wichtige Rolle. Dabei existiert eine große Bandbreite an Anforderungen bzgl. der Zuverlässigkeit von eingebetteten Systemen. Die Sicherstellung der Zuverlässigkeit erfordert allerdings für Hardware und Software unterschiedliche Methoden. Während die Zuverlässigkeit der Hardware vor allem hinsichtlich des Einsatzes unter extremen Betriebsbedingungen (Temperatur, mechanische Einflüsse, Strahlungen etc.) und möglicher Korrosionserscheinungen der Bauteile sichergestellt werden muss, so hängt die Zuverlässigkeit von Software in einem hohen Maße von der korrekten Implementierung der gewünschten Funktionalität ab.

Software-basierte Steuerungssysteme haben spezifische Eigenschaften, die – zumindest in ihrer Kombination – einmalig sind: sie sind typischerweise eingebettet, interagieren mit Sensoren und Aktuatoren mit kontinuierlichen Signalverläufen, steuern diskrete Kontrollflüsse, vermitteln und verarbeiten einfache und komplexe strukturierte Daten, kommunizieren über verschiedene Bussysteme und müssen hohen Sicherheits- und Echtzeitanforderungen genügen

Während verschiedene, auch modellbasierte Entwicklungsansätze und –methoden für eingebettete Systeme existieren, fehlt eine allgemein anerkannte Testtechnologie für die Analyse und Bewertung dieser Systeme, die zu qualitativ hochwertigen, sicheren und zuverlässigen Systemen führt.

Eine solche Testtechnologie soll die verschiedenen Aspekte von eingebetteten Systemen adressieren und das Testen des diskreten Verhaltens für die Kommunikation und Steuerung der Abläufe, des kontinuierlichen Verhaltens für die regelungstechnischen Abläufe und des hybriden Verhaltens für die steuerungstechnischen Abläufe in der Interaktion mit Sensoren/Aktuatoren und mit anderen Komponenten und dem Nutzer ermöglichen.

Dieses Papier stellt einen auf der standardisierten Testtechnologie TTCN-3 basierenden Entwurf einer solchen Testtechnologie für eingebettete Systeme vor.

Abstract

Embedded systems play an ever increasing role for the realisation of complex control functions in many industrial domains – resulting in a big variety of requirements with respect to their dependability and reliability.

Assuring dependability and reliability requires however different methods for hardware and software components. While the hardware reliability has to be guaranteed in particular for extreme operating conditions (temperature, mechanical influences, radiations etc.) and for possible material-dependent corrosion phenomena, the reliability of software components depend mostly on the correct implementation of the required functionality.

Software-based control systems have specific characteristics, which - at least in their combination - are unique: they are typically embedded, interact with sensors and actuators with continuous signal characteristics, supervise discrete control flows, obtain and process simple and complex structured data, communicate over different bus systems and have to meet high safety and real time requirements.

While different, also model-based development processes and methods for embedded systems exist, a generally recognized test technology for the analysis and evaluation of these systems, which lead to qualitatively high-quality, safe and reliable systems, is missing.

Such a test technology has to address the different aspects of embedded systems and to enable the testing of discrete behaviours for communication and control sequences, of continuous behaviours for the regulation sequences and of hybrid behaviours for the control sequence in interaction with sensors/actuators and with other system components and the user.

This paper presents a design of such a test technology for embedded systems, which is based on the standardized test technology TTCN-3.

1 Introduction

Model-based black box testing is an evolving technique for generating suites of test cases from system designs. It provides a systematic basis for system testing, gives quantifiable coverage for the test behavior and allows tests to be linked directly to the requirements [1][2][3]. Model-based testing gained a lot of momentum in the last years due to its automation potential.

However, there is no widely accepted common test technology: within every industrial domain for almost every product line there are proprietary, specifically developed test methods and tools. This is not only cumbersome, but requires also the re-invention of testing concepts common to the whole area of testing embedded systems over and over again. Development, support and maintenance of the tools as well as the education and training of testers for those tools have to be repeated for every new method or tools. Test artefacts can be reused only by one tool – but not by others across testing kinds, testing phases and alike.

In contrast to this, in the telecommunication domain a standardized and well-established test technology TTCN [5] has been developed over the years to enable the conformance and interoperability testing of telecommunication systems from different vendors and developed based on different platforms and in different technologies.

With its new version TTCN-3 [4] (now, Testing and Test Control Notation) the applicability of TTCN has been widened to functional and non-functional tests of generally reactive systems. Since its first publication in 2000, TTCN-3 has been extended to scale with industrial testing requirements and successfully applied to systems from various domains [8][9]. Consortia like AutoSar, ESA or the MOST Forum selected TTCN-3 only recently for case studies or test suite developments for their core technologies.

TTCN-3 has the potential to serve as a testing middleware that unifies tests of communicating, software-based systems, unifies the test infrastructure as well as the definition and documentation of tests. It provides concepts for local and distributed, platform- and technology-independent testing, which is adapted to concrete testing environments and concrete systems to be tested by means of an open test execution environment.

In order to use however the full potential of TTCN-3, it needs to be extended towards concepts dedicated to the testing of hybrid control systems. This paper presents in Section 2 TTCN-3 as such, gives an overview of the extended concepts being proposed already in Section 3 and adds in Section 4 additional concepts for result evaluation. A summary concludes the paper.

2 The standardized test technology TTCN-3

TTCN (Testing and Test Control Notation) is in the telecommunication domain a widely established and used test technology. In its new version, TTCN-3 has a wider scope and applicability. It cannot be used only for testing the conformance and interoperability of communication protocols but also the interaction of e.g. sensors, actuators and control units connected via bus systems. Therefore, TTCN-3 is now being used in other domains such as automotive, railways, avionics and security systems.

The TTCN-3 language was created due to the imperative necessity to have a universally understood (specification and implementation) language syntax able to describe test behaviours and test procedures. Its development was imposed by industry and science to obtain a single test notation for all black-box and grey-box testing needs. In contrast to earlier test technologies, TTCN-3 encourages the use of a common methodology and style which leads to a simpler maintenance of test suites and products. With the help of TTCN-3, the tester specifies the test suites at an abstract level and focuses on the test logic to check a test purpose itself rather than on the test system adaptation and execution details. A standardized language provides a lot of advantages to both test suite providers and users. Moreover, the use of a standard language reduces the costs for education and training, as a great amount of documentation, examples, and predefined test suites are available. It is obviously preferred to use always the same language for testing than learning different technologies for distinct testing kinds. Constant use and collaboration between TTCN-3 vendors and users ensure a uniform maintenance and development of the language.

TTCN-3 enables systematic, specification-based testing for various kinds of tests including e.g. functional, scalability, load, interoperability, robustness, regression, system and integration testing. TTCN-3 is a language to define test procedures to be used for black-box and grey-box testing of distributed systems. It allows an easy and efficient description of complex distributed test behaviours in terms of sequences, alternatives, and loops of stimuli and responses. The test system can use a number of test components to perform test procedures in parallel. TTCN-3 language is characterized by a well-defined syntax and operational semantics, which allow a precise execution algorithm. The task of describing dynamic and concurrent configurations is easy to perform. The communication can be realized either synchronously or asynchronously. To validate the data transmitted between the entities composing the test system, TTCN-3 supports definition of templates with a powerful matching mechanism representing test data. To validate the described behaviours, a verdict handling mechanism is provided. The types and values can be either described directly in TTCN-3 or can be imported from other languages (e.g. ASN.1, XML schema, or IDL). Moreover in TTCN-3, the parameterization of types and values is allowed. The selection of the test cases to be executed can be either controlled by the user or can be described within the execution control construct. The external configuration of a test suite through module parameters is possible.

Figure 1 shows an overview of the TTCN-3 language. The TTCN-3 core language defines the concept space of TTCN-3 [4] and its textual presentation format. Two other presentation formats, which allow the graphical and tabular definition of test artefacts, exist. TTCN-3 provides interfaces to reference data defined in other description languages. As the figure shows, one can import types and values specified in ASN.1, but other formats are also supported (IDL, XML schema etc).

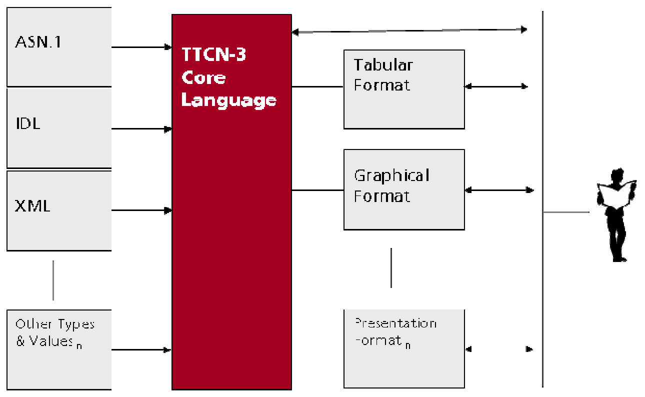

Fig. 1: *The TTCN-3 Language Architecture*

The ETSI standard for TTCN-3 comprises currently seven parts (described below) which are grouped together in the “Methods for Testing and Specification; The Testing and Test Control Notation version 3” document [3]:

1. TTCN-3 Core Language. This document specifies the syntax of TTCN-3 language.

2. Tabular Presentation Format. TTCN-3 offers optional presentation formats. The tabular format is similar in appearance and functionality to earlier versions of TTCN. It was designed for users that prefer the TTCN-2 style of writing test suites. A TTCN-3 module is presented in the tabular format as a collection of tables.

3. Graphical Presentation Format. It is the second presentation format of TTCN-3 and is based on the MSC format (Message Sequence Charts). The graphical format is used to represent graphically the TTCN-3 behaviour definitions as a sequence of diagrams.

4. Operational semantics. This document describes the meaning of TTCN-3 behaviour constructs and provides a state oriented view of the execution of a TTCN-3 module.

5. The TTCN-3 Runtime Interface (TRI). A complete test system implementation requires also a platform specific adaptation layer. The TRI document contains the specification of a common API interface to adapt TTCN-3 test systems to SUT.

6. The TTCN-3 Control Interfaces (TCI). This part provides an implementation guideline for the execution environments of TTCN-3. It contains the specification of the API the TTCN-3 execution environments should implement in order to ensure the communication, management, component handling, external data control and logging.

7. Use of ASN.1 in TTCN-3: This part provides guidelines and mappings rules for the combined use of ASN.1 (Abstract Syntax Notation One) and TTCN-specifications

Additional parts of the standards for the language mappings are currently under development.

2.1 The Concepts of TTCN-3

The TTCN-3 core language is a modular language and has a similar look and feel to a typical programming language. In addition to the typical programming constructs, it contains all the important features necessary to specify test procedures and campaigns for functional, conformance, interoperability, load and scalability tests like test verdicts, matching mechanisms to compare the reactions of the SUT with the expected range of values, timer handling, distributed test components, ability to specify encoding information, synchronous and asynchronous communication, and monitoring. A TTCN-3 test specification consists of imports from other modules; types, test data and templates definition, function, altstep and test case definitions for test behaviour; and control definitions for the execution of test cases (see Figure 2).

The top-level building-block of TTCN-3 is a module. A module contains all other TTCN-3 constructs, but cannot contain sub-modules. It can also import completely or partially the definitions of other modules. The modules are defined with the keyword module. The modules can be parameterized; parameters are sets of values that are supplied by the test environment at runtime. A parameter can be initialized with a default value.

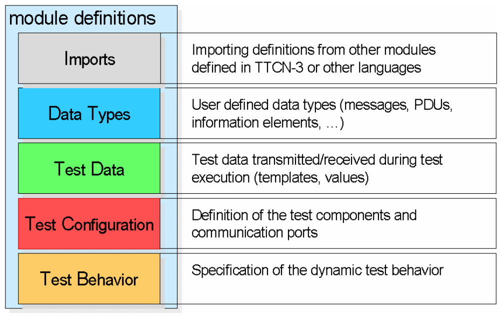
module definitions
Imports
Importing definitions from other modules defined in TTCN-3 or other languages
Data Types
User defined data types (messages, PDUs, information elements, ...)
Test Data
Test data transmitted/received during test execution (templates, values)
Test Configuration
Definition of the test components and communication ports
Test Behavior
Specification of the dynamic test behavior

Fig. 2: *TTCN-3 Module Structure*

A TTCN-3 module has two parts: the module definition part and the module control part. The definition part contains the data defined by that module (functions, test cases, components, types, templates), which can be used everywhere in the module and can be imported from other modules. The control part is the main program of the module, which describes the execution sequence of the test cases or functions. It can access the verdicts delivered by test cases and, according to them, can decide the next steps of execution. The test behaviours in TTCN-3 are defined within functions, altsteps and testcases. The control part of a module may call any testcase or function defined in the module to which it belongs.

2.2 Test System

A test case is executed by a test system. TTCN-3 allows the specification of dynamic and concurrent test systems. A test system consists of a set of interconnected test components with well-defined communication ports and an explicit test system interface, which defines the boundaries of the test system.

Within every test system, there is one Main Test Component (MTC). All other test components are called Parallel Test Components (PTCs). The MTC is created and started automatically at the beginning of each test case execution. A test case terminates when the MTC terminates, which implies also the termination of all other PTCs. The behaviour of the MTC is specified in the body of the test case definition. During the execution of a test case, PTCs can be created, started and stopped dynamically. A test component may stop itself or can be stopped by another test component.

For communication purposes, each test component owns a set of local ports. Each port has an in- and an out-direction. The in-direction is modelled as an infinite FIFO queue, which stores the incoming information until it is processed by the test component owning the port. The out-direction is directly linked to the communication partner (another test component or the system under test (SUT)), i.e., outgoing information is not buffered.

During test execution, TTCN-3 distinguishes between connected and mapped ports. Connected ports are used for the communication with other test components. If two ports are connected, the in-direction of one port is linked to the out-direction of the other, and vice versa. A mapped port is used for the communication with the SUT. The mapping of a port owned by a test component to a port in the abstract test system interface can be seen as pure name translation defining how communication streams should be referenced. TTCN-3 distinguishes between the abstract and the real test system interface. The abstract test system interface is modelled as a collection of ports that defines the abstract interface to the SUT. The real test system interface is the application specific part of a TTCN-3-based test environment. It implements the real interface of the SUT and is defined in the TTCN-3 runtime interface (TRI, part 5 of TTCN-3).

In TTCN-3, connections and mappings are created and destroyed dynamically at runtime. There are no restrictions on the number of connections and mappings a component may have. A component (and even a port) may be connected to itself. One-to-many connections are allowed, but TTCN-3 only supports one-to-one communication, i.e., during test execution the communication partner has to be specified uniquely. For the communication among test components and between test components and the SUT, TTCN-3 supports message-based and procedure-based communication. Message-based communication is based on an asynchronous message exchange and the principle of procedure-based communication is to call procedures in remote entities.

2.3 Test Cases and Test Verdicts

Test cases define test behaviours which have to be executed to check whether the SUT passes the test or not. Like a module, a test case is considered to be a self-contained and complete specification of a test procedure that checks a given test purpose. The result of a test case execution is a test verdict.

TTCN-3 provides a special test verdict mechanism for the interpretation of test runs. This mechanism is implemented by a set of predefined verdicts, local and global test verdicts and operations for reading and setting local test verdicts. The predefined verdicts are pass, inconc, fail, error and none. They can be used for the judgment of complete and partial test runs. A pass verdict denotes that the SUT behaves according to the test purpose, a fail indicates that the SUT violates its specification. An inconc (inconclusive) describes a situation where neither a pass nor a fail can be assigned. The verdict error indicates an error in the test devices. The verdict none is the initial value for local and global test verdicts, i.e., no other verdict has been assigned yet. During test execution, each test component maintains its own local test verdict. A local test verdict is an object that is instantiated automatically for each test component at the time of component creation. A test component can retrieve and set its local verdict. The verdict error is not allowed to be set by a test component. It is set automatically by the TTCN-3 run-time environment, if an error in the test equipment occurs. When changing the value of a local test verdict, special overwriting rules are applied. The overwriting rules only allow that a test verdict becomes worse, e.g., a pass may change to inconc or fail, but a fail cannot change to a pass or inconc.

In addition to the local test verdicts, the TTCN-3 run-time environment maintains a global test verdict for each test case. The global test verdict is not accessible for the test components. It is updated according to the overwriting rules when a test component terminates. The final global test verdict is returned to the module control part when the test case terminates.

2.4 Alternatives and Snapshots

A special feature of the TTCN-3 semantics is the snapshot. Snapshots are related to the behaviour of components. They are needed for the branching of behaviour due to the occurrence of timeouts, the termination of test components and the reception of messages, procedure calls, procedure replies or exceptions. In TTCN-3, this branching is defined by means of alt statements.

An alt statement describes an ordered set of alternatives, i.e., an ordered set of alternative branches of behaviour. Each alternative has a guard. A guard consists of several preconditions, which may refer to the values of variables, the status of timers, the contents of port queues and the identifiers of components, ports and timers. The same precondition can be used in different guards. An alternative becomes executable, if the corresponding guard is fulfilled. If several alternatives are executable, the first executable alternative in the list of alternatives will be executed. If no alternative becomes executable, the alt statement will be executed again.

The evaluation of several guards needs some time. During that time, preconditions may change dynamically. This will lead to inconsistent guard evaluations, if a precondition is verified several times in different guards. TTCN-3 avoids this problem by using snapshots. Snapshots are partial module states, which include all information necessary for the evaluation of alt statements. A snapshot is taken, i.e., recorded, when entering an alternative. For the verification of preconditions, only the information in the current snapshot is used. Thus, dynamic changes of preconditions do not influence the evaluation of guards.

2.5 Default Handling

In TTCN-3, defaults are used to handle communication events which may occur, but which do not contribute to the test objective. Default behaviour can be specified by altsteps and then activated as defaults. For each test component, the defaults, i.e., activated altsteps, are stored as a list. The defaults are listed in the order of their activation. The TTCN-3 operations activate and deactivate operate on the list of defaults. An activate operation appends a new default to the end of the list and a deactivate operation removes a default from that list.

The default mechanism is invoked at the end of each alt statement, if the default list is not empty and if due to the current snapshot none of the alternatives is executable. The default mechanism invokes the first altstep in the list of defaults and waits for the result of its termination. The termination can be successful or unsuccessful. Unsuccessful means that none of the top alternatives of the altstep defining the default behaviour is executable, successful means that one of the top alternatives has been executed.

In case of an unsuccessful termination, the default mechanism invokes the next default in the list. If the last default in the list has terminated unsuccessfully, the default mechanism will return to the alt statement, and indicates an unsuccessful default execution. Unsuccessful default execution causes the alt statement to be executed again.

In case of a successful termination, the default may either stop the test component by means of a stop statement, or the main control flow of the test component will continue immediately after the alt statement from which the default mechanism was called or the test component will execute the alt statement again. The latter has to be specified explicitly by means of a repeat statement. If the selected top alternative of the default ends without a repeat statement the control flow of the test component will continue immediately after the alt statement.

Altsteps are function-like descriptions for structuring component behaviour. TTCN-3 uses altsteps to specify default behaviour or to structure the alternatives of an alt statement. The precise semantics of altsteps is closely related to alternatives and snapshots. Like an alt statement, an altstep defines an ordered set of alternatives, the so-called top alternatives. The difference is that no snapshot is taken when entering an altstep. The evaluation of the top alternatives is based on an existing snapshot. An altstep is always called within an alt statement, which provides the required snapshot. Conceptually, the top alternatives of the altstep are inserted into the alternatives of the alt statement. Within the core language, the user can specify where the top alternatives shall be placed into the list of alternatives. It is also possible to call an altstep like a function. In this case, the altstep is interpreted like an alt statement which only invokes the altstep, i.e., the top alternatives are the only alternatives of the alt statement.

2.6 Communication Operations

Communication operations are important for the specification of test behaviours. TTCN-3 supports message-based and procedure-based communication. The communication operations can be grouped in two parts: stimuli, which send information to SUT and responses, which are used to describe the reaction of the SUT.

Procedure-based communication is synchronous communication. Procedure-based operations defined in TTCN-3 are:

- call: to invoke a remote procedure;
- getcall: to specify that a test component accepts a call from the SUT;
- reply: to reply value when an own procedure is called;
- getreply: specifies that a method is invoked;
- raise: to report an exception when an own procedure is called an something is wrong in the procedure call;
- catch: to collect an exception reported at remote procedure invocation.

Message-based communication is asynchronous communication. The sending operations are non-blocking; after sending the data, the system does not wait for response. The receive operations block the execution until a matching value is received. A receiving operation specifies a port, at which the operation takes place, defines a matching part for selection of valid receiving values and optionally specifies an address to identify the connection if the port is connected to many ports.

Message-based communication operations defined in TTCN-3 are:

- send: send a message to SUT;
- receive: receive a message from SUT;
- trigger: specifies a message that shall receive in order to go to the next statements.

2.7 Test Data Specification

A test system needs to exchange data with the SUT. The communication with the SUT can be either asynchronous, by sending/receiving messages to/from SUT or synchronous, by calling procedures of the SUT or accepting procedure calls from the SUT. In these cases, the test data must be described within the test system, according to the SUT specification. TTCN-3 offers different constructs to describe the test data: types, templates, variables, procedure signatures etc. They can be used to express any type of protocol message, service primitive, procedure invocation or exception handling. Besides this, TTCN-3 offers also the possibility to import data described in other languages (e.g. ASN.1, IDL, or XML schema).

In order to describe basic data types, TTCN-3 provides a number of predefined types. Most of these types are similar to basic types of well-known programming languages (Java, C). Some of them are only testing domain specific:

- Port types define the characteristics (message or procedure based, allowed data in the in and out direction) of ports used in the communication between test components and to the SUT.
- Component types define the properties of test components such as their ports and local variables and timers.
- The verdicttype is an enumeration which defines the possible verdicts that can be associated to a test case: pass, fail, inconc, error, none.
- The anytype is a union of all known TTCN-3 types; the instances of anytype are used as a generic object which is evaluated when the value is known.
- The default type is used for default handling and represents a reference to a default operation.

TTCN-3 also supports ordered structured types such as record, record of, set, set of, enumerated and union. Furthermore, for procedure-based communication, TTCN-3 offers the possibility to define procedure signatures. Signatures are characterized by name, optional list of parameters, optional return value and optional list of exceptions.

Templates represent test data and are data structures used to define message patterns for the data sent or received over ports. They are used either to describe distinct values that are to be transmitted over ports or to evaluate if a received value matches a template specification. Templates can be specified for any type or procedure signature. They can be parameterized, extended or reused in other template definitions. The declaration of a template contains a set of possible values. When comparing a received message with a template, the message data shall match one of the possible values defined by the template.

As not all details of TTCN-3 can be given here, please refer to the standard, further papers and tools to get a broader overview on the technology.

2.8 TTCN-3 based test systems

TTCN-3 defines not only a specification technology for tests, but provides also with the TTCN-3 runtime interfaces (TRI) in part 5 of the standard series and with the TTCN-3 control interfaces (TCI) in part 6 an common architecture for the realization of executable tests with TTCN-3. This architecture enables the adaptation of TTCN-3 tests to test environments and systems under tests in a well-defined manner. It also enables the reuse of test platform components.

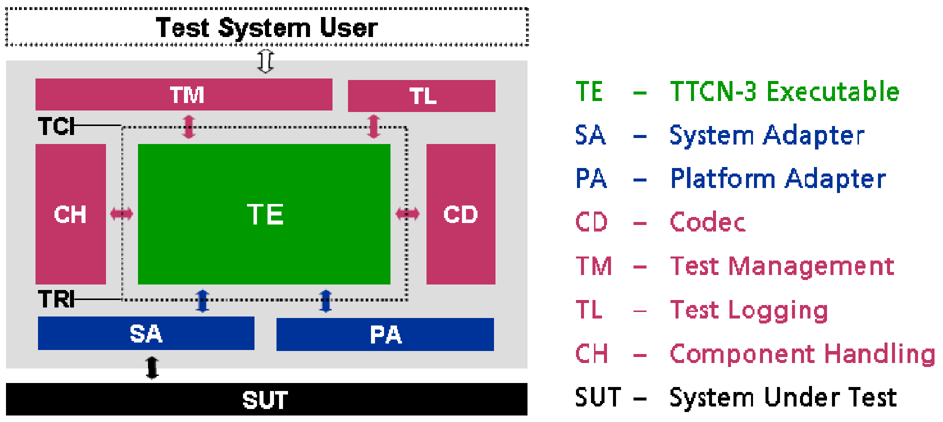

Fig. 3: *TTCN-3 Test System Architecture*

The following entities are part of a TTCN-3 test system:

- TE (TTCN-3 Engine) executes the compiled TTCN-3 code. This entity manages different sub-entities for test control, behavior, components, types, values and queues entities which realize the TTCN-3 semantics.
- TM (Test Management) manages the test execution. This entity implements operations to execute tests, provide and set module parameters and external constants. The test logging is also tracked by this component.

- TL (Test Logging) provides test execution logs. This entity realizes trace generation for the online or offline analysis of test runs.
- CH (Component Handler) handles the communication between test components. The CH API contains operations to create, start, start, and stop test components, establish the connection between test components (map, connect), handle the communication operations (send, receive, call, reply) and manage the verdicts. The information about the created components and their physical locations is stored in a repository inside the containers.
- CD (Coding/Decoding) encodes and decodes types and values. The TTCN-3 values are encoded into bitstrings which are sent to the SUT. The received data is decoded into abstract TTCN-3 types and values.
- SA (System Adaptor) realizes mainly the sending and receiving of data over concrete physical means.
- PA (Platform Adaptor) provides means to include external functionality, measures time and determines when a timer has expired.

TTCN-3 tooling (as e.g. by Testing Technologies, Telelogic or OpenTTCN) provides typically TE, TM, TL, CH and PA as often only SA and CD are system under test dependent. This limits the implementation efforts needed for the preparation of executable tests substantially.

The definition of the above introduced TRI and TCI interfaces makes TTCN-3 not only a test specification, but also a test implementation technology. The interfaces are central for the practical relevance of TTCN-3. They define the openness of the TTCN-3 architecture and enable the customization and application of TTCN-3 in different domains. The openness is the basis to provide domain- or system-specific test frameworks with TTCN-3 as outlined in [10].

3 Continuous TTCN-3: An extension of TTCN-3 for embedded systems

In order to enable the definition of tests for hybrid systems, TTCN-3 has been extended in [7] with

- a concept of continuous time and sampling rates,
- a concept of streams and stream ports, and
- a concept of assignments to/evaluations of ports by equational definitions and by time partitions.

In Continous TTCN-3, time is considered to be global and continuous. It is sampled by means of global or local samples which can be of any precisions. The samples give access to the current point in time. Although there might be in addition to a global sample, different local samples (with different precision) they time progress for the samples is the same – i.e. time is global and shared by all samples (and by all test components).

Streams are continuous flows of data, i.e. there is a current value of the stream at any point in time. They can be of any type, i.e. both basic and structured types can be the data of a stream. Stream ports provide interfaces to streams, i.e. they are TTCN-3 ports that offer or accept streams.

Stream behaviour and stream evaluations can be provided in two forms. One is the direct definition by a set of equations, which defines how (potentially) local streams and output streams at ports behave depending on input streams at ports – typically within certain ranges of time. In addition, Continous TTCN-3 uses the **carry-**statement to denote a behaviour (a continuous, discrete or hybrid one) which is performed without any time limitation or up until a given condition matches.

4 Working with Streams in Continuous TTCN-3

In [7], we defined the evaluation of continuous behaviour in terms of equation systems. In this paper, we will further outline how to access stream values and how to calculate with and analyse streams. These considerations are influenced by the concepts proposed in [6].

Please consider the following example defining for the automated cruise control stream data including the brake, gas and velocity information:

```
:
  type record ACC{
    float phi_brake;
    float phi_gas;
    float v_act;
  }
:
 stream ACC ACCStream;
:
```

Then, we can access current stream values at time t with the @-operator to check for example certain values in the data structure (here checking if the brake is used):

```
:
if (ACCStream@t.phi_brake > 0.0) { … }
:
```

which also allows to assign certain values to the stream such as assigning acceleration values:

```
:
ACCStream@t:= {0.0,10.0,10.0};
ACCStream@(t+1.0):= {0.0,10.0,20.0}
:
```

Depending on the definition of sample t, t provides an absolute or relative point in time: it is absolute if t is the global sample for the current test suite; it is relative if it is a local sample within a function.

As the time points for which a stream has a value can be enumerated, we propose an alternative syntax for the indexed access to stream values:

```
:
ACCStream[12]:= {0.0,10.0,10.0};
ACCStream[13]:= {0.0,10.0,20.0}
:
```

assuming that the previous time point t has index 12 and that time point t+1 has index 13.

In addition to those basic operations on streams, we propose to enable the calculation with streams in order to e.g. add or compare streams:

```
:
 stream ACC ACCStream1, ACCStream2, ACCStreamResult;
:
ACCStreamResult:= ACCStream1 + ACCStream2;
//resulting in the sum of the values at every point in time
:
if (ACCStream1 < ACCStream2) {
// yielding true if every value of ACCStream1 is less than the
// corresponding value of ACCStream2 at that point in time
:
```

This operations work individually for every point in time. Currently, we assume that streams can be added, subtracted or compared if they are defined for the same sample and are of equal length (to be determined with the sizeof operation). If not, the operation will result in a runtime error during test execution as depicted in the following pseudo code:

```
stream float NumericStream;
function add(inout NumericStream p, inout NumericStream q inout)
  return NumericStream{

  var NumericStream ret;
  var integer maxlen:= max(sizeof(p),sizeof(q));
  for (integer i:=0;i<maxlen;i++){
    if (sizeof(p)<=i && sizeof(q)<=i){
      ret[i]:=p[i]+q[i];
    }
    else{
      ret[i]:=error;
    }
  }
  return ret;
}
```

In future work, we will further analyse needed concepts for dealing with streams in concrete case studies.

5 Summary

In this paper we propose to use the standardized test technology TTCN-3 with its extension Continous TTCN-3 for the systematic testing of embedded control systems.

We review the basis concepts of TTCN-3 for testing discrete behaviour of communicating, software-based systems and the concepts of Continuous TTCN-3 for testing continuous behaviours including streams, stream-based ports, sampling, equation systems for continuous behaviours and the carry-statement for complex control flows along continuous behaviours.

Along the application of Continuous TTCN-3, we experienced needs for an easy access to streams and their values, but also for the calculation with streams. An outline of these concepts is given at the end of this paper.

In future work, the concepts will be implemented in the TTCN-3 tool set [11] and applied to more complex case studies in the context of CAN-based engine controls in a car.

6 References

[1] Utting M.: Model-Based Testing, The University of Waikato, New Zealand, Verified Software: Theories, Tools, Experiments, VSTTE 2005.

[2] M. Utting, A. Pretschner and B. Legeard: A Taxonomy of Model-Based Testing – In preparation.

[3] Broy M., Jonsson B., Katoen J.-P., Leucker M., Pretschner A. (Eds.), Model-Based Testing of Reactive Systems, Advanced Lectures, Springer 2005, ISBN 3-540-26278-4

[4] ETSI ES 201 873: Testing and Test Control Notation (TTCN-3), Sophia Anti-polis, France, v3.1.1, October 2005. http://www.etsi.org/ptcc/ptccttcn3.htm.

[5] ISO/IEC 9646-3: Information Technology - Open Systems Interconnection - Conformance Testing Methodology and Framework (CTMF) - Part 3: The Tree and Tabular Combined Notation (TTCN), edition 2, Dec. 1997.

[6] Zander-Nowicka, J.; Schieferdecker, I.; Marrero Pérez, A.: Automotive Validation Functions for on-line test evaluation of hybrid real-time systems, IEEE AutoTestCon 2006, Anaheim California, USA, Sept, 2006.

[7] Schieferdecker, I.; Bringmann, E.; Großmann, J.: Continuous TTCN-3: Testing of Embedded Control Systems, SEAS'06 at ICS 2006, May 23, 2006, Shanghai, China, ACM Press.

[8] Deiß, T.; Rennoch, A; Schieferdecker, I.; Vassiliou-Gioles, T.: Advanced Test Processes using TTCN-3, ITEA Publications, March 2006.

[9] TTCN-3 User Conference Series: 2004, 2005 at ETSI, Sophia-Antipolis, France, 2006 at Siemens, Berlin, Germany; http://www.ttcn-3.org.

[10] Schieferdecker, I.; Vassiliou-Gioles, T.: Tool Supported Test Frameworks in TTCN-3. – 8th Intern. Workshop in Formal Methods in Industrial Critical Systems, Røros, Norway, June 2003, ENTCS (80), Elsevier Science.

[11] TestingTechnologies IST GmbH : TTCN-3 integrated development environment TTworkbench. http://www.testingtech.de/