The Test Technology TTCN-3

Ina Schieferdecker[1], Jens Grabowski[2], Theofanis Vassiliou-Gioles[3], and George Din[4]

[1] Technical University Berlin/Fraunhofer FOKUS
ina@cs.tu-berlin.de
[2] University of Goettingen
grabowski@informatik.uni-goettingen.de
[3] Testing Technologies
vassiliou@testingtech.de
[4] Fraunhofer FOKUS
din@fokus.fraunhofer.de

Abstract. The Testing and Test Control Notation (TTCN-3) is a widely established test technology traditionally used in the telecommunication domain. In its new version, TTCN-3 has a wider scope and applicability. It can be applied not only for testing the conformance and interoperability of communication protocols but also for testing the functionality, interoperation and performance of software-based systems in general. Therefore, TTCN-3 is nowadays used in other domains such as automotive, railways, avionics, or security systems. This chapter introduces the concepts of the TTCN-3 language and provides examples of its practical use.

1 Overview

Despite of automated on-the fly test generation as, for example, advocated in [3], the explicit specification of tests is needed as the majority of tests is still developed manually [16,20,22]. For that, the use of a standardized and well-defined test notation is recommended as the tests can be defined in a precise, well-understood and widely accepted format. Furthermore, automatically generated tests are often extended and adapted manually as current test generation techniques still bear several limitations. In order to keep track of those adaptations, the resulting tests should be explicitly denoted as well. Another advantage of standardized test specifications is the ability to provide test platforms for automated test execution, which could be used across domains – provided that there are domain-specific adapters for the different target technologies.

TTCN-3 is the successor language of the Tree and Tabular Combined Notation (TTCN) [14,17] which was developed due to the imperative necessity to have a universally understood test (specification and implementation) language able to describe test data and test behaviours with sophisticated test concepts. TTCN-3 [15] is a powerful test technology which allows to specify and execute

R.M. Hierons et al. (Eds.): Formal Methods and Testing, LNCS 4949, pp. 292–319, 2008.

detailed test descriptions for several kinds of testing on different levels of abstraction including, for example, component level, integration level, and system level testing.

TTCN-3 has been specifically developed for the design and definition of test systems. The syntax and operational semantics of TTCN-3 tests is commonly understood and not related to a particular programming language or technology of systems to be tested. TTCN-3 tests concentrate on the purpose of the test and abstract from particular test system details. Off the shelf tools for TTCN-3 and TTCN-3-based test systems are readily available [34]. Many successful test solutions and applications have been realised with TTCN-3 also beyond the traditional telecommunication domain [2,13,31].

The development of TTCN-3 was driven by industry and academia with the objective to obtain one test notation for black-box and grey-box testing. 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 increases their reuse. When using TTCN-3, a test designer develops test cases at an abstract level. She can focus on the test logic for checking a system against given test purposes. She does not have to deal with test system specifics and test execution details. A standardized test notation provides a lot of advantages to test solution providers, testers and other users of test specifications and test results: the use of a standardized test notation reduces the costs for education and training as a large amount of documentation, examples, and predefined test suites is available. In 2007, a TTCN-3 certification procedures has been established, so that people's TTCN-3 knowledge can be examined and certified along internationally agreed rules [4]. It is obviously preferred to use wherever possible the same notation for testing than learning different technologies for different test projects. The constant use and collaboration between TTCN-3 vendors and users ensure a continuous maintenance and further development of the base technology.

TTCN-3 enables systematic, specification-based testing for functional, interoperability, robustness, regression, scalability, and load testing on component, integration and system level. It supports the definition of test procedures for local and distributed *test configurations* via homogeneous or heterogeneous interfaces (so called *ports*) to the *system under test* (the SUT). It allows the definition of simple and complex *test behaviours* in terms of sequences, alternatives, and loops of system stimuli and observations. Test behaviours can be executed in parallel on a number of *test components*. Configurations of test components can be fixed or can vary during test execution, i.e. test components may be created dynamically depending on the conditions along a test run.

The interaction with the SUT can be realised either by *asynchronous, message-based* or *synchronous, procedure-based* communication. The test system provides stimuli to the SUT and receives and checks the SUT reactions. The observed SUT reactions are checked against expected reactions by using *templates*. Templates are defined in terms of *matching mechanisms* which allow to define expected responses in a detailed manner by concrete values, by ranges or sets of values, or by logical properties of value sets. Based on the comparison

between expected and observed reaction, the SUT is assessed and *test verdicts* are assigned. Basically, all test data and test behaviours can be *parameterized*, even parameters for complete test suites can be defined.

The concepts of TTCN-3 outlined above are defined by a well-defined syntax and operational semantics, which provide a precise meaning and execution semantics to TTCN-3. Figure 1 presents the architecture of the TTCN-3 technology. TTCN-3 is based on a *core language* which has a look and feel similar to a general purpose programming language like C, C++ or Java. The predefined tabular and graphical presentation formats address users preferring a tabular or Message Sequence Chart (MSC) [19] like presentation of test specifications. TTCN-3 allows to import data types and data values specified in the Abstract Syntax Notation One (ASN.1) [18], the Interface Definition Language (IDL) [23] or the Extended Markup Language (XML) [38]. Further presentation formats like, e.g. a state machine-based presentation format, and interfaces to data type and data value notations like, e.g. defining the import of C and C++ data types and data values are under discussion or even under development. They may be standardized in future version of the TTCN-3 standard.

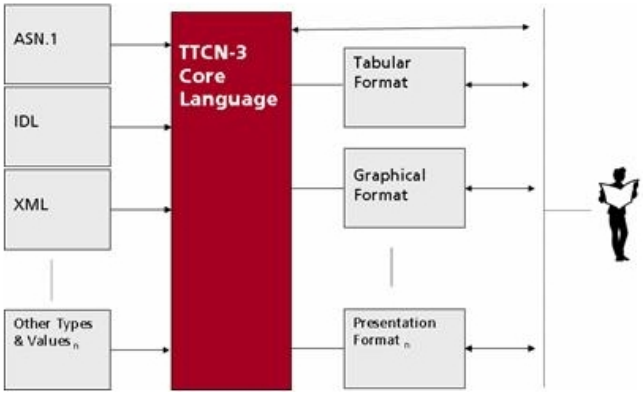

Fig. 1. The TTCN-3 Language Architecture

The rest of the chapter is structured as follows: Section 2 reviews the history of TTCN-3 and enumerates the different parts of the TTCN-3 standard series. In Section 3, the main concepts of TTCN-3 such as test case, test behaviour, verdict, or alt statement are described. The presentation formats are explained in Section 4. Section 5 introduces a web-server test example to demonstrate the application of the TTCN-3 concepts. Section 6 presents the execution interfaces of TTCN-3 and explains how TTCN-3 test systems can be realized by use of these interfaces. The import and usage of external data into TTCN-3 test suites is explained in Section 7. A comparison with the UML Testing Profile [24] is given in Section 8. Finally, a summary and an outlook conclude the chapter.

2 The Standard and Its History

TTCN-3 has been developed by the European Telecommunication Standards Institute (ETSI). The TTCN-3 standard comprises currently seven parts and two additional technical reports. The technical reports will become the parts 7 and 9 of the TTCN-3 standard and specify the use of ASN.1 and XML in TTCN-3. The already standardized seven parts of the TTCN-3 standard [6,7,8,9,10,11,12] contain the following information:

Part 1: TTCN-3 Core Language. This document specifies the textual syntax of TTCN-3.

Part 2: Tabular Presentation Format. The tabular presentation format presents a TTCN-3 specification within a collection of tables.

Part 3: Graphical Presentation Format. The graphical presentation format is used to represent TTCN-3 tests as interactions between the SUT and the test system. This presentation format is based on the Message Sequence Chart (MSC) [19] language.

Part 4: Operational semantics. The operational semantics describes the meaning of TTCN-3 behaviour by providing a state oriented view on the execution of TTCN-3 tests.

Part 5: TTCN-3 Runtime Interfaces (TRI). A complete test system implementation requires a platform specific adaptation layer. TRI contains the specification of a common Application Programming Interface (API) interface to adapt TTCN-3 test systems to an SUT.

Part 6: TTCN-3 Control Interfaces (TCI). This part of the TTCN-3 standard contains the specification of the APIs, which a TTCN-3 execution environment should implement for the encoding and decoding of test data, test management, component handling, external data control and logging.

Part 8: Use of IDL in TTCN-3. This document provides guidelines and mappings rules for the combined use of IDL and TTCN-3.

TTCN-3 evolved from the Tree and Tabular Combined Notation (TTCN) which was developed for conformance testing of telecommunication protocols. TTCN was first published in 1992 as part 3 of the international ISO/IEC standard 9646 "OSI Conformance Testing Methodology and Framework" [17]. Since then, TTCN has been intensively used to specify tests for different technologies like Global System for Mobile Communication (GSM), Digital Enhanced Cordless Technologies (DECT), Intelligent Network Application Protocol (INAP), and Integrated Services Digital Network (N-ISDN, B-ISDN). Small extensions of TTCN [14] addressed test modules, parallel test setups and the use of ASN.1 in TTCN. Although TTCN was improved, it implements the concepts of OSI conformance testing and is therefore only of limited usability for other kinds of testing such as interoperability, robustness, regression, or system testing. Since TTCN was designed for testing OSI protocols, it is also difficult to apply TTCN to other technologies like for mobile systems or CORBA-based applications.

In 1998, ETSI was asked by its members to develop a new test language, namely TTCN-3, addressing current and upcoming test requirements. The development of TTCN-3 was encouraged by key players of the telecommunication industry and by researchers to overcome the limitations of TTCN. The standardization process, lead by the ETSI Protocol and Testing Competence Center (PTCC), completed in 2000 the first version of TTCN-3. Since then TTCN-3 is a continuously maintained test technology. For this, ETSI provides a well-defined change request procedure (see http://www.ttcn-3.org/TTCN3cr.htm), to which everybody can contribute. This allows corrections and extensions to TTCN-3 resulting in revised versions of the standard. In February 2007 TTCN-3 v3.2.1 has been approved by ETSI. From its first version in 2000 until its latest version in 2007, TTCN-3 has evolved to a powerful basis for test development and can serve as a target for test generation and other means of efficient test development [27,37].

3 The Concepts of TTCN-3

The TTCN-3 core language is a modular language which has a similar look and feel to a typical programming language like, e.g. C or C++. In addition to the typical programming language constructs, it contains all important features necessary to specify test procedures and campaigns like test verdicts to assess test runs, matching mechanisms to compare the reactions of the SUT with the expected outputs, timer handling to specify time restrictions, the handling of test components to support distributed testing, the ability to specify encoding information, support for different kinds of communication (i.e. synchronous and asynchronous communication) and the possibility to log test information during a test run.

3.1 TTCN-3 Module

A TTCN-3 test specification is defined by a set of modules. As shown in Figure 2, a module typically contains imports from other modules; data type definitions, test data descriptions, definitions for test configuration, test behaviour specifications and a module control part to specify the ordering, selection and execution of test cases.

The top-level building-block of TTCN-3 is the module. A module cannot contain sub-modules, but may import partially or completely definitions from other modules and contains further definitions necessary for a test. A module definition starts with the keyword **module**. A module can be parameterized; a parameter is a data value that is supplied by the test environment at runtime. It is possible to initialize a parameter with a default value.

A TTCN-3 module has two parts: the module definitions part and the module control part. The module definitions part contains definitions specified by that module. These definitions can be used everywhere in the module and may be imported from other modules. The module control part is the main program

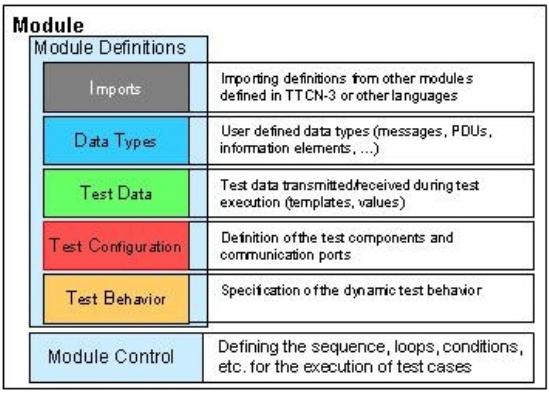

Fig. 2. The TTCN-3 Module Structure

of a module. It describes the execution sequence of the test cases. The control part can use the verdicts delivered by test cases to select the next test case to be executed. The control part of a module can call any test case known in the module, i.e. locally defined in the module or imported from another module.

3.2 TTCN-3 Test System and Test Cases

A test case (TTCN-3 keyword **testcase**) is executed by a test system. TTCN-3 allows the specification of local and distributed test systems with static and dynamic test configurations. A test system may consist of a single test component or of a set of interconnected test components. It has well-defined communication ports and an explicit test system interface, which defines the boundaries to the test system. The set of test components together with their connections to the SUT and to other test components constitute the test configuration.

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. The behaviour of the MTC is specified in the body of the test case definition. A test case terminates when the MTC terminates. This implies also the termination of all PTCs. 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 communication 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 which can be another test component or the SUT. This means that that 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. In TTCN-3, connections and mappings can be created and destroyed dynamically at runtime. There are no restrictions on the number of connections and mappings a component may have. A component may be connected to itself. One-to-many connections are allowed. 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. The principle of procedure-based communication is to call procedures in remote entities. This allows a test component to emulate the client or server side during a test. Furthermore, unicast, multicast and broadcast communication are also supported by TTCN-3.

Test cases define test behaviours which can be executed to check whether the SUT passes the test or not. 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 apply. The overwriting rules only allow that a test verdict becomes worse. For example, a **pass** may change to **inconc** or **fail**, but a **fail** cannot change to a **pass** or **inconc**. All local test verdicts contribute to the final global test verdict of the test case. For this, the overwriting rules explained above are also used, i.e. the worst local verdict will become the final global verdict of the test case.

3.3 Test Behaviour

TTCN-3 allows an easy and efficient description of simple and complex, sequential and parallel test behaviours in terms of sequences, alternatives, loops, stimuli and responses. Stimuli and responses are exchanged at the interfaces of the SUT, which are defined as a collection of ports. The test system can use a single test

component for sequential test procedures or a number of test components to perform test procedures in parallel. Likewise to the interfaces of the SUT, the interfaces of the test components are described as ports.

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 as a loop until one of the guards will permit entering an alternative.

In TTCN-3, a **default** mechanism can be used to handle communication events which may occur, but which do not contribute to the test objective. A **default** behaviour can be specified by an **altstep** which then can be activated as a default behaviour. It is possible to define complex default behaviour by having activated several altsteps at the same time. For each test component, the defaults, i.e. activated altsteps, are stored in a list of defaults in the order of their activation. The default behaviours can be activated or deactivated through the TTCN-3 operations **activate** and **deactivate**. These operations operate on the list of defaults. An **activate** operation appends a new default to the beginning 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 state none of the alternatives is executable. The default mechanism invokes the first altstep in the list of defaults, i.e. the altstep which has been lastly activated, and waits for the result of its termination. The termination can be successful or unsuccessful. Unsuccessful means that none of the alternatives of the altstep defining the default behaviour is executable, successful means that one of the 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 indicate an unsuccessful default execution. An 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, 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.

3.4 TTCN-3 Communication and Test Data

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 the exchange of messages via buffers. This kind of communication is often called asynchronous communication, because the sending and receiving of a message are decoupled. A sender continues its execution after sending the message without waiting for an answer. Procedure-based communication uses remote procedure calls for communication. This kind of communication is often also called synchronous communication, because the caller of a remote procedure is normally blocked during the treatment of the call, i.e. caller and callee are synchronized via the call.

TTCN-3 offers the following operations for procedure-based communication:

- **call**: to invoke a remote procedure;
- **getcall**: to accept a call from remote;
- **reply**: to reply to a previously received call;
- **getreply**: to accept a reply;
- **raise**: to report an exception to a previously received call;
- **catch**: to collect an exception reported by a remote procedure invocation.

For message-based communication, TTCN-3 offers the following operations:

- **send**: to send a message;
- **receive**: to receive a message;
- **trigger**: to discard all messages until the specified message is received.

TTCN-3 offers different possibilities to specify test data. The structure of test data can be described by means of pre- and user-defined data types and signatures (for procedure-based communication. Data values, value sets and value ranges can be specified by means of constants, variables, data templates and signature templates. Besides this, TTCN-3 offers also the possibility to import data described in other languages like, for example, ASN.1, IDL or XML.

Most of the predefined data types of TTCN-3 are similar to the basic data types known from programming languages like C, C++ or Java. However, some of them are special to TTCN-3:

- Port types define the characteristics communication ports, i.e. kind of communication, communication direction, data to be exchanged via a port of this type.
- Component types define the properties of test components, i.e. ports and local variables, timers and constants owned by a component of that type.
- The **verdicttype** is an enumeration type which defines the possible test verdicts, i.e. **pass**, **fail**, **inconc**, **error** and **none**.
- The **anytype** is a union of all known TTCN-3 types of a TTCN-3 module; an instance of anytype is used as a generic object which is evaluated when the value is known.
- The **default** type is used for default handling. A value of this type is a reference to a default.

For the definition of structured data types, TTCN-3 supports ordered and unordered structured types such as **record** (ordered structure), **record of** (ordered list), **set** (unordered structure), **set of** (unordered list), **enumerated** and **union**. Furthermore, for procedure-based communication, TTCN-3 offers the possibility to define procedure signatures. Signatures are characterized by their name, an optional list of parameters, an optional return value and an optional list of exceptions.

Templates are the main means to represent test data in TTCN-3. A template is a data structures used to define a pattern for a data item sent or received over a port. A template may describe a distinct value, a range of values or a set of values. Distinct values may be transmitted over a port, whereas templates describing ranges or sets of values may be matched with data received from the SUT or other test components. Such a match is successful, if the received data is an element of the data values described by the template. Templates can be specified for arbitrary data types (data templates) and for signatures (signature templates). Furthermore, they can be parameterized, extended and in other template definitions.

4 TTCN-3 Presentation Formats

TTCN-3 offers its core language and two standardized presentation formats to serve the needs of different application domains and users. The textual core language suits best to persons familiar with a general purpose programming language. A core language based test development allows using a text editor of the users' choice and, thus, enables an easy integration into a test environment.

The tabular presentation format for TTCN-3 (TFT) is defined in part 2 of the TTCN-3 standard series. It is designed for users that prefer to use tables for test specification. TFT presents a TTCN-3 module as a collection of tables. TFT highlights the structural aspects of a TTCN-3 module – in particular type and template structures.

Part 3 of the TTCN-3 standard series defines the graphical presentation format for TTCN-3 (GFT) [29]. GFT provides a visualization of TTCN-3 behaviour definitions in an MSC-like manner [19]. It eases the reading, documentation and discussion of test procedures. It is also well suited to describe of test traces and for analyzing of the test results. For each kind of TTCN-3 behaviour definition, GFT provides a special diagram type, i.e. function diagrams for representing functions, altstep diagrams for the visualization of altsteps, test case diagrams for showing test cases and control diagrams for showing the control part of a module.

However, the work on presentation formats is not finished. Where needed, additional presentation formats to represent specific aspects of TTCN-3 test suites can be defined and seamlessly integrated into a TTCN-3 development environment.

5 TTCN-3 Example

The use of TTCN-3 is demonstrated by a small example to test a web server's functionality. A single request represented by a Uniform Resource Locator (URL) is sent to the web server. After receiving the request, the web server should respond by sending an XML file back containing a list of `dinosaurs`. We expect that the list must contain a dinosaur whose name is `Brachiosaurus`. If there is such a dinosaur in the list, the test verdict is `pass`.

5.1 Core Language Example

In the following, we present the TTCN-3 module for testing the Web server described above. The module is split into several parts. We first present a part and describe its contents afterwards.

Listing 1.

```
module dinolistTest {                                          1
 modulepar integer NUMBER_OF_PTCS := 1
 with {
 extension (NUMBER_OF_PTCS)                                    4
 "Description: Default number of PTCs";
 }
 type record urlType {                                         7
  charstring protocol,
  charstring host,
  charstring file                                              10
 }
 template urlType urlTemplate := {
  protocol := "http://",                                       13
  host := "www.testingtech.de",
  file := "/TTCN-3_Example/dinolist.xml"
 }                                                             16
 :
```

In the lines 7 to 11 of Listing 1, we specify the structure of a URL by defining the **record** `urlType`. The `urlType` consists of fields for a `protocol` (line 8), a `host` (line 9), and a `file` (line 10). All fields are of type **charstring** since the request to a Web server also is a string. Until now, we specified only which fields a URL is composed of. We did not specify the values of the fields. This is done within the template declaration with name `urlTemplate` (lines 12 to 16). In this template, the protocol field is set to `http://`, the host field is set to `www.testingtech.de` and the file is set `/TTCN-3_Example/dinolist.xml`. This means, the complete URL sent to the Web server is URL `http://www.testingtech.de/TTCN-3_Example/dinolist.xml`.

Listing 2.

```
:
type set of dinosaurType dinolistType;                          2
type record dinosaurType {
 charstring name,
 charstring len,                                                5
 charstring mass,
 charstring time,
 charstring place                                               8
}
template dinosaurType BrachiosaurusTemplate := {
 name := "Brachiosaurus",                                       11
 len := ?,
 mass := ?,
 time := ?,                                                     14
 place := ?
}
template dinolistType DinoListTemplate := {                     17
 ?,
 ?,
 BrachiosaurusTemplate,                                         20
 ?,
 ?,
 ?,                                                             23
 ?
}
:                                                               26
```

In the next step, we define the answer that we expect from the Web server during the test. As already mentioned, we expect a list of dinosaurs, where one of them must have the name `Brachiosaurus`. As we don't know at which position of the list our expected dinosaur is, we define a type `dinolistType` (line 2), which is a **set of** dinosaurs.

A dinosaur represented by the **record** type `dinosaurType` (lines 3 to 9). It consists of the fields `name` (line 4) which specifies the name of the dinosaur, `len` (line 5) which specifies the length (length is a keyword in TTCN-3 and can therefore not used as field name), `mass` specifying the weight (line 6), `time` describing when the dinosaur lived (line 7) and `place` specifying the place where the dinosaur lived (line 8).

A template for dinosaurs with the name `BrachiosaurusTemplate` is defined in the lines 10 to 16. As it is not important which length and mass the dinosaur had or when and where it lived, these fields were filled with a `?` wildcard meaning any value is accepted as valid, i.e. these fields must be filled with a **charstring** but the concrete values do not matter.

For the list of dinosaurs, the template `DinoListTemplate` is defined (lines 17 to 25). During the test we expect to receive a list of seven dinosaurs from the Web

server. The third dinosaurs in the list must be a `Brachiosaurus`. For this, the template `BrachiosaurusTemplate` is reference in the `DinoListTemplate` template (line 20). The other fields of `DinoListTemplate` must be valid dinosaurs, but their kind is is irrelevant. This is specified again by using the matching for any value symbol `?`.

Listing 3.

```
:
type component ptcType {                                    2
 port httpPortType httpPort;
 timer localTimer := 3.0;
                                                            5
}
type port httpPortType message {
 out urlType;                                               8
 in dinolistType;
}
type component mtcType {}                                   11
type component systemType {
 port httpPortType httpPortArray[NUMBER_OF_PTCS];
                                                            14
}
:
```

After defining the messages that will be exchanged, test components have to be specified (Listing 3). We start with the type for the PTCs that will send requests to the Web server and check the received messages (lines 2 to 6). This component type has the name `ptcType` and owns a port and a timer. The port is named `httpPort` (line 3) and of type `httpPortType`. The timer has the name `localTimer` and its default expiration time is set to `3.0` seconds (line 4). The port type definition `httpPortType` of `httpPort` can be found in the lines 7 to 10. It is a port for message-based communication (keyword **message**) and it allowed to send messages of type `urlType` (line 8) and to receive messages of type `dinolistType` (line 9).

In a next step, we define the component type `mtcType` (line 11). This component type describes the type of the MTC. The MTC is created by the test system automatically at the start of each test case execution. The behaviour of an MTC is the body of a test case definition. As in the example test case, presented afterwards, the whole communication with the SUT is done only via PTCs, there is no need to define ports, variables or timers for the MTC. Hence, the component type `mtcType` is empty.

Finally, a test component is needed that represents the interfaces to the SUT (lines 12 to 15). It is necessary to define the interfaces where the ports of the PTCs can be mapped to, so that communication can take place. Therefore, an array of ports is created with a size of the given module parameter `NUMBER_OF_PTCS` (Listing 1, lines 2 to 5). Module parameters give the possibility to change

parameter settings during a test campaign, without the need to change the TTCN-3 module definitions and to recompile it. Every module parameter can have a default value; in our case it is set to `1`.

Listing 4.

```
:
testcase DinoListTest_1()
runs on mtcType system systemType {                                    3
 var ptcType ptcArray[NUMBER_OF_PTCS];
 var integer i := 0;
 for (i := 0; i < NUMBER_OF_PTCS; i := i + 1) {                        6
  ptcArray[i] := ptcType.create;
  map (ptcArray[i]: httpPort, system: httpPortArray[i]);
 }                                                                     9
 for (i := 0; i < NUMBER_OF_PTCS; i := i + 1) {
  ptcArray[i].start (ptcBehaviour());
 }                                                                     12
 all component.done;
}
:                                                                      15
```

Now, the **testcase** itself and by that the behaviour of the MTC has to be specified (Listing 4, lines 2 to 14). The name of the test case is `DinoListTest_1`. The test case body starts with the definition of an array of components of type `ptcType` (line 4). The size of this array is defined by the module parameter `NUMBER_OF_PTCS`.

Then, the PTCs are created within a **for** loop (lines 6 to 9). In each cycle of the loop, a test component is created and its reference is stored in the array of PTCs (line 7). Furthermore, the port of the newly created test component is mapped to the system port in the system array of ports `httpPortArray`.

Once the test configuration is set up in a second **for** statement (lines 10 to 12), the behaviour of each PTC is started with the function `ptcBehaviour()`.

With the **all component.done** statement (line 13), the termination of all parallel test components is awaited. This ensures that every test component contributes to the overall test case verdict.

Listing 5.

```
:                                                                      1
function ptcBehaviour() runs on ptcType {
 httpPort.send (urlTemplate);
 localTimer.start;                                                     4
 alt {
  [] httpPort.receive (DinoListTemplate) {
   localTimer.stop;                                                    7
   setverdict (pass);
  }
```

```
    [] httpPort.receive {                                    10
     localTimer.stop;
     setverdict (fail);
    }                                                        13
    [] localTimer.timeout {
     setverdict (fail);
    }                                                        16
   }
  }
  :                                                          19
```

The behaviour of a PTC is described by the function `ptcBehaviour()` shown in the lines 2 to 18 of Listing 5. Firstly, a request (the template `urlTemplate` defined in Listing 1) is sent via `httpPort` (line 3). The `httpPort` is the port of a PTC and has been mapped to a port of the SUT by the MTC in the test case definition. This means that messages send on port `httpPort` are forwarded to the SUT.

Immediately after sending the request, a `localTimer` is started (line 4) as a watch dog to avoid infinite waiting for responses. The timer will run `3.0` seconds as this is the default value of `localTimer` in the `ptcType` type definition (Listing 3).

After the start of the timer, an **alt** statement is used to describe the potential reactions of the SUT:

1. The expected dinosaur list that matches the `DinoListTemplate` is received (line 6). Then the timer is stopped (line 7) and the verdict is set to **pass** (line 8).
2. Something else is received that does not match the expected response. This is specified by using a **receive** operation without a parameter (line 10). In this case, the timer will also be stopped (line 11), but the verdict will be set to **fail** (line 12) as the SUT responded incorrectly.
3. If nothing is received within `3.0` seconds, a **timeout** message from the timer `localTimer` occurs (line 14). Then, the verdict will also be set to fail (line 15).

After that, the PTC terminates. If during test case execution no further PTCs are running, the MTC terminates also. The final verdict of the test case is the accumulated test verdicts of all PTCs.

Listing 6.

```
 :
 control {
  execute (DinoListTest_1());                                3
 }
}
```

In the lines 2 to 4 of Listing 6, the module control part for our example module is specified. If the control part is called from the test management, the testcase `DinoListTest_1` will be executed (line 3).

5.2 GFT Example

The test behaviour definitions of the TTCN-3 module described in the previous section can be visualized by means of the Graphical Presentation Format for TTCN-3 (GFT). Figure 3 visualizes the test case `DinoListTest_1`. A comparison with the textual description provided in Listing 4 shows that the information in both presentations is identical.

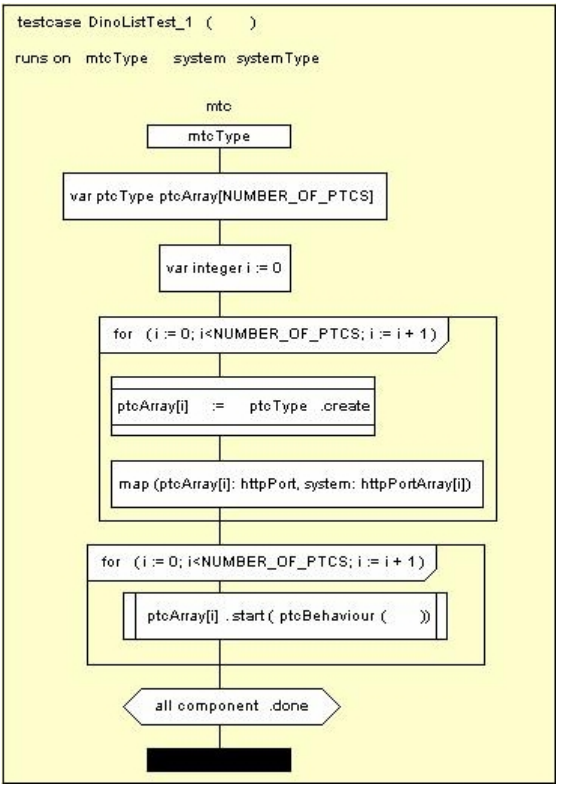

Fig. 3. A GFT Testcase

The GFT diagram in Figure 4 presents the behaviour of the PTCs. It shows that after sending the URL request the positive case when receiving the expected answer and the two negative cases when receiving a wrong or no response.

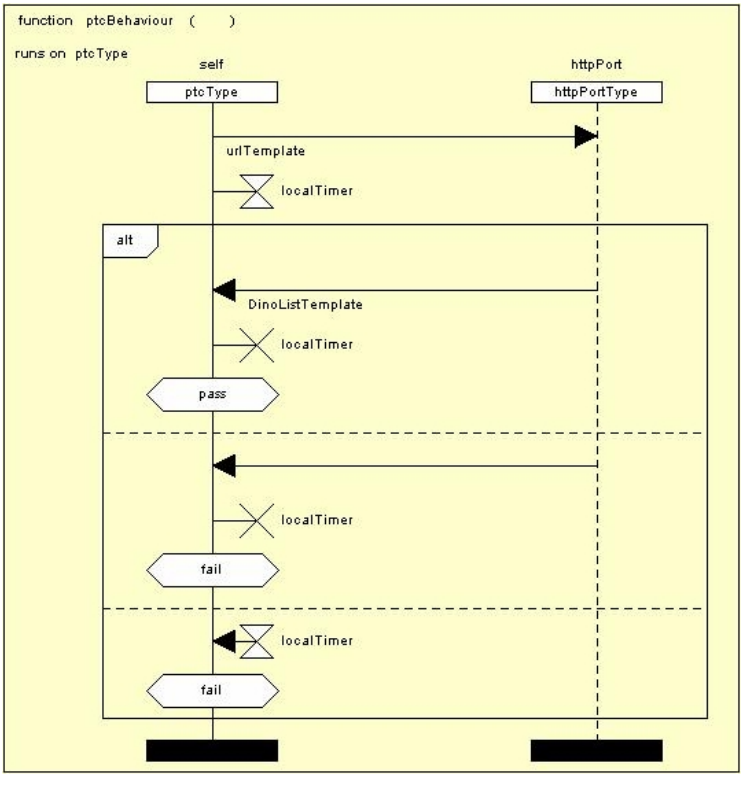

Fig. 4. A GFT Function

6 TTCN-3 Based Test Execution

TTCN-3 standardizes not only the language but also the architecture of an execution environment for TTCN-3 test suites. The standard architecture of a test system consists of several entities which communicate mutually through predefined interfaces. The ETSI specification of the test system architecture is given in two documents: the TTCN-3 Runtime Interfaces (TRI) which is the fifth part of the standard and the TTCN-3 Control Interfaces (TCI) which is the sixth part of the standard. The TRI and TCO provide with a defined set of APIs a unified model to realise the TTCN-3 based test systems. Further extension beyond a pure TTCN-3 based test system are possible [36].

The general structure of a TTCN-3 test system is depicted in Figure 5. A TTCN-3 test system is built-up of a set of interacting entities which manage the test execution (by interpreting or executing the TTCN-3 code), realise the communication with the SUT, implement external functions and handle timer operations.

The TTCN-3 Executable (TE) contains the executable code produced by the compilation of TTCN-3 modules and the TTCN-3 run-time itself. The TE

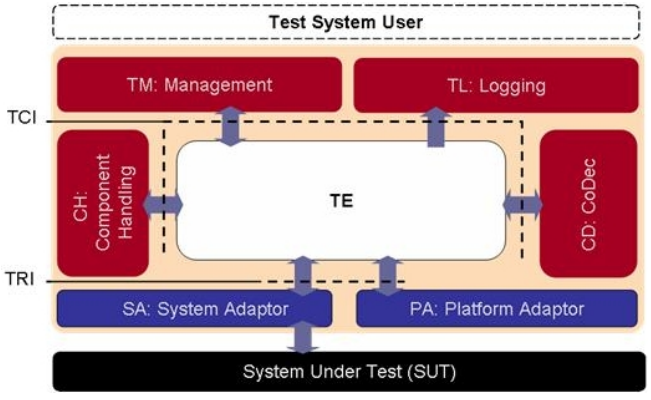

Fig. 5. TTCN-3 Test System Architecture

communicates with the Test Management (TM), the Component Handling (CH) and the Codec (CD) via TCI. The communication with the SUT is realised by using the TRI which defines the interfaces between the TE, the System Adapter (SA) and the Platform Adapter (PA). The different components of a test system have the following functions:

- The TE interprets or executes the compiled TTCN-3 code. It manages the different TTCN-3 entities like test control, test behaviour, test components, types, values and queues.
- The CH handles the communication between components. The CH API contains operations to create, start, stop test components, to establish the connection between test components, to handle the communication operations and to manage the verdicts. The information about the created components and their physical locations is stored in a repository within the test execution environment.
- The TM manages the test execution. It implements operations to execute tests, to provide and set module parameters and external constants. The TTCN-3 logging mechanismis also realised by this component.
- The CD encodes and decodes values according to their types. The TTCN-3 values are encoded into bit strings which are sent to the SUT. The received data is decoded back into TTCN-3 values.
- The SA realises the communication with the SUT. The TTCN-3 communication operations used to interact with the SUT, are implemented by the SA.
- The PA implements timers and external functions. Timers are platform specific elements and have to be implemented outside the pure TTCN-3 test system. The PA provides operations in to handle timers by means of create, start and stop operations. External functions are only declared in a TTCN-3 module. They are implemented in the PA.

The TCI and TRI operations are defined in IDL [23]. They are mapped to the test system specific technology. Particularly, TRI and TCI handle aspects like inter-component communication and timer handling which need to be implemented out-side the TE. This approach allows the use of different test system implementations, e.g. the CH may be implemented with CORBA, Remote Method Invocation (RMI) or another technology. The implementation of the CH is transparent to TE. The TE calls the operations provided by CH which finally handle the requests.

TTCN-3 tests can run on a single test device or be distributed over several test devices and executed in a parallel and coordinated manner [28,33].

Figure 6 provides a distributed view of the test system architecture. The TE is instantiated on each test device. The handling of test components, which may be created on different nodes, is realised by the CH.

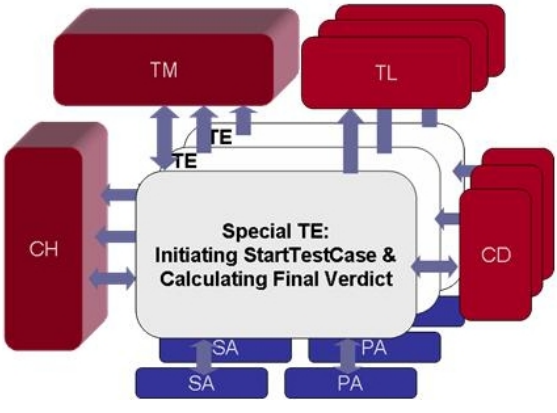

Fig. 6. Realisation of Distributed TTCN-3 Test Systems

The TM is executed the test device from which the test runs are being managed by the user. It is responsible for the user control of the test execution, for the logging of the distributed tests and for the presentation of the results to the user. The CD, SA and PA entities are instantiated on each device because their implementation may differ depending on the underlying, potentially heterogeneous test devices.

7 TTCN-3 External Data

As illustrated in Figure 1, TTCN-3 supports the import of foreign data objects, i.e. defined in other languages than TTCN-3, into TTCN-3 modules. Such foreign data objects can only be used in TTCN-3 modules, if they have a TTCN-3 meaning.

The term TTCN-3 view can be best explained by considering a case where the definition of a TTCN-3 object refers to another TTCN-3 object; the information content of the referenced object shall be available and is used for the new definition. For example, when a template is defined based on a structured type, the identifiers and types of fields of the base type shall be accessible and are used for the template definition. In a similar way, when the referenced type is a foreign object, it shall provide the same information content as if it has been declared as TTCN-3 type. A foreign object, may contain more information than required by TTCN-3. In such a case, the additional information has no meaning in TTCN-3 and is therefore not accessible.

The use of foreign objects in TTCN-3 modules is supported by two concepts: Firstly, the language allows to **import** and use them like TTCN-3 definitions. Secondly, special **attribute** strings are defined which assure that a TTCN-3 module referring to foreign objects will be portable to any tool supporting the external language.

To make declarations of foreign object visible in TTCN-3 modules, their names shall be imported just like declarations from other TTCN-3 modules. When imported, only the TTCN-3 meaning of the object will be seen in the importing TTCN-3 module. There are two main differences between importing TTCN-3 items and objects defined in other languages:

- When importing non-TTCN-3 definitions, the **import** statement shall contain an appropriate language identifier.
- Only foreign objects with a TTCN-3 meaning can be imported into a TTCN-3 module.

Importing can be done automatically using the **all** directive, in which case all importable objects are automatically imported, or done manually by listing the names of items to be imported.

In several important application domains that are amongst the first users of TTCN-3, ASN.1 is used to describe the data structure of messages. For this reason, TTCN-3 provides sophisticated support to use ASN.1 together with TTCN-3. It allows the reference to ASN.1 definitions from within TTCN-3 modules and the specification of encoding rules for imported ASN.1 definitions including the dynamic control of encoding options.

An increasing number of distributed applications use the XML to exchange data These data exchanges follow very precise rules for data format description in the form of Document Type Descriptions (DTDs) or XML Schemas. There are also XML based communication protocols like, for example, the Simple Object Access Protocol (SOAP), that are based on XML Schemas. Like any other communication system, XML based systems are also candidates for testing using TTCN-3. The XML mapping rules provide a definition for the use of XML with TTCN-3. It enables the combined use of XML types defining XML based protocols, interfaces, Web services, applications or documents with TTCN-3 based testing [32].

Last but not least, object-based technologies such as CORBA, the Distributed Component Object Model (DCOM), or the Distributed Computing Environment

(DCE) and component-based technologies such as the CORBA Component Model (CCM), the Enterprise Java Beans (EJB), or the .Net framework use interface specifications to describe the structure of object- and component-based systems including operations and capabilities to interact with the environment. These interface specifications support interoperability and reusability of objects and components. The techniques used for interface specifications are often IDL-based, for example, CORBA IDL, Microsoft IDL or DCE IDL. These languages are comparable in their abilities to define system interfaces, operations at system interfaces and system structures. They only differ in details of the object or component model. When considering the testof object- and component-based systems with TTCN-3, one is faced with the problem of accessing the systems to be tested via the system interfaces described in form of an IDL specification. TTCN-3 supports the import of IDL definitions into TTCN-3 modules by providing standardized IDL to TTCN-3 mapping rules.

8 U2TP and TTCN-3

The OMG (Object Management Group) has initiated the development of a UML 2.x testing profile (U2TP) to make the Unified Modelling Language (UML) also applicable for the design of test systems. It addresses typical testing concepts in model-based system development and for integrated system and test development processes [1]. Compared with TTCN-3, U2TP also addresses test design and can be mapped to TTCN-3 [16,25,39].

U2TP defines a language for designing, visualizing, specifying, analyzing, constructing and documenting the artefacts of test systems. It is a test modelling language that can be used with all major object and component technologies and be applied to test systems in various application domains. U2TP can be used stand alone for test artefacts only or in an integrated manner with UML for handling system and test artefacts together.

U2TP extends UML 2.x with test specific concepts like test components, verdicts, defaults, etc. These concepts are grouped into concepts for test architecture, test data, test behaviour and time. As a UML profile, U2TP seamlessly integrates into UML. It is based on the UML 2.x meta-model and reuses UML 2.x syntax. The U2TP test concepts are structured into

- *Test architecture* concepts defining concepts related to test structure and test configuration, i.e. the elements and their relationships involved in a test;
- *Test behaviour* concepts defining concepts related to the dynamic aspects of test procedures and addressing stimuli, observations and activities during a test;
- *Test data* concepts defining concepts for test data used in test procedures, i.e. the structures and meaning of values to be processed in a test;
- *Time* concepts defining concepts for a time quantified definition of test procedures, i.e. the time constraints and time observation for test execution.

Architecture concepts	Behaviour concepts	Data concepts	Time Concepts
SUT	Test objective	Wildcards	Timer
Test components	Test case	Data pools	Time zone
Test context	Defaults	Data partitions	
Test configuration	Verdicts	Data selectors	
Arbiter	Test control	Coding rules	
Scheduler			

Fig. 7. Overview of Basic Testing Profile Concepts

	U2TP	TTCN-3
Test Design	✓	–
Test Specification	✓	✓
Test Execution	(–)	✓
Test Meta Modelling	✓	(✓)
Format	Graphical	Textual and graphical
Transformation	U2TP to TTCN-3 (✓)	TTCN-3 to U2TP ✓

Fig. 8. Comparison of U2TP and TTCN-3

U2TP was an ideal opportunity to bring TTCN-3 in form of GFT to the attention of the UML world [26]. In fact, GFT is the archetype for U2TP. U2TP uses several concepts being developed in GFT. Still, TTCN-3 and U2TP differ in several respects: U2TP is based on the object oriented paradigm of UML where behaviours are bound to objects only, while TTCN-3 is based on the TTCN-3 test behaviour concept of functions and binding of functions to test components. U2TP uses additional diagrams to define, e.g. the test architecture, test configuration and test deployment. Test behaviour can be defined as interaction diagrams (as in TTCN-3) but also as state machines or activity diagrams. While TTCN-3 supports dynamic test configurations, U2TP uses static configurations where only the number of test components may vary but not the structure of the connections between test components. In addition, U2TP has only one FIFO queue per test component, while TTCN-3 uses a FIFO queue per test component port. New concepts in U2TP as compared to TTCN-3 are the arbiter, the validation action, the test trace and the data pool, data partition and data selector.

However, above all, U2TP and TTCN-3 address different phases in test development as shown in Figure 8. U2TP addresses primarily test design and test specification, while TTCN-3 addresses test specification and test execution. U2TP can also support test execution, but this needs still to be worked out along the approaches towards executable UML.

Test design is out of scope for TTCN-3. U2TP has by definition a meta-model (as an extension of the UML 2.0 meta-model). For TTCN-3, proprietary meta-models exist only. Both have graphical presentation formats, while TTCN-3 has also a textual notation.

The transformation from TTCN-3 to U2TP is always possible; the other way works only, if the U2TP specifies implementable and executable tests. Mapping rules have been defined accordingly. Currently, TTCN-3 is widely supported by tools and test solutions. For U2TP, first tools e.g. [35] exist, further tools are under development.

9 Summary

In this chapter, a detailed introduction into TTCN-3 is provided. The chapter explains the concepts behind TTCN-3, the TTCN-3 core language as well as the implementation and execution of TTCN-3 test systems. It also discusses how TTCN-3 can be integrated with other technologies such as by reusing external data or mappings to and from U2TP.

TTCN-3 has been designed especially for testing purposes and provides powerful testing constructs. These make it the technology of choice for a wide variety of testing needs. Over the last years, the TTCN-3 technology has been used in different areas of testing including telecommunication or data communication as well as automotive, railways, avionics or security systems.

One of the most important characteristics of TTCN-3 is its technology and platform independence. This allows testers to concentrate on the test logic, while the complexity of the test realization on a given test device (e.g. operating system, hardware configuration, etc.) is moved to the TTCN-3 platform. Complex test behaviours which involve multiple interacting test entities are easier to specify in TTCN-3 than in other test frameworks such as JUnit [21] since technical aspects are hidden behind the abstract language artefacts.

Typical applications of TTCN-3 include functional, conformance, and interoperability testing of various systems on component, integration and system level. There is also an increasing interest in applying TTCN-3 to performance, load and scalability testing — one example is the application of TTCN-3 for IMS (IP Multimedia Subsystem) benchmarking [5]. TTCN-3 has gained also special attention in the context of testing embedded systems [30]. Future work on TTCN-3 will include specializations of TTCN-3 in further application areas such as financial or medical systems.

References

1. Baker, P., Dai, Z.R., Grabowski, J., Haugen, O., Lucio, S., Samuelsson, E., Schieferdecker, I., Williams, C.: The UML 2.0 Testing Profile. In: Proceedings by ASQF Press, Nuremberg, Germany (September 2004) (conquest 2004)
2. Burton, S., Baresel, A., Schieferdecker, I.: Automated testing of automotive telematics systems using TTCN-3. In: Proceedings by Fraunhofer IRB Verlag, 3rd Workshop on System Testing and Validation (SV 2004), Paris, France (December 2004)
3. de Vries, R.G., Tretmans, J.: On-the-fly conformance testing using SPIN. International Journal on Software Tools for Technology Transfer (STTT) 2(4), 382–393 (2000)
4. Schieferdecker, I., et al. The TTCN-3 Certificate: An ETSI/GTB Certification Scheme for TTCN-3 (2007), `http://www.german-testing-board.info`
5. ETSI TISPAN. IMS/NGN Performance Benchmark, Technical Standard (TS) 186 008, Sophia-Antipolis, France (February 2007)
6. ETSI Standard (ES) 201 873-1 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
7. ETSI Standard (ES) 201 873-2 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular presentation Format (TFT). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
8. ETSI Standard (ES) 201 873-3 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
9. ETSI Standard (ES) 201 873-4 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
10. ETSI Standard (ES) 201 873-5 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
11. ETSI Standard (ES) 201 873-6 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
12. ETSI Standard (ES) 201 873-8 V3.2.1 (2007–02): Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: Using IDL with TTCN-3. European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (February 2007)
13. TTCN-3 User Conference Series (2004-2007), `http://www.ttcn-3.org`
14. ETSI Technical Report (TR) 101 666 (1999–2005): Information Technology — Open Systems Interconnection Conformance testing methodology and framework; The Tree and Tabular Combined Notation (TTCN) (Ed. 2++). European Telecommunications Standards Institute (ETSI), Sophia-Antipolis France (May 1999)
15. Grabowski, J., Hogrefe, D., Rethy, G., Schieferdecker, I., Wiles, A., Willcock, C.: An Introduction into the Testing and Test Control Notation (TTCN-3). Computer Networks Journal (2003)

16. Gross, H.-G., Schieferdecker, I., Din, G.: Model-Based Built-In Tests. In: ITM 2004, International workshop on Model Based Testing, co-located with ETAPS 2004, Barcelona, Spain, January 2004. Electronic Notes in Theoretical Computer Science, vol. 111 (2004)
17. ISO/IEC IS 9646. Information Technology - OSI Conformance Testing Methodology and Framework. International Multipart Standard 9646, Geneva, Switzerland (February 1992-1996)
18. ITU-T Recommendations X.680-683 (2002): Information Technology — Abstract Syntax Notation One (ASN.1):
– X.680: Specification of Basic Notation
– X.681: Information Object Specification
– X.682: Constraint Specification
– X.683: Parameterization of ASN.1 Specifications.
ITU Telecommunication Standards Sector, Geneva Switzerland (2002)
19. ITU-T Recommendation Z.120: Message Sequence Chart (MSC). ITU Telecommunication Standards Sector, Geneva Switzerland (1999)
20. Kaner, C., Falk, J., Nguyen, H.Q.: Testing Computer Software, 2nd edn. John Wiley & Sons, Ltd, Chichester (1999)
21. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices, Upper Saddle River, NJ, USA. Prentice Hall PTR, Englewood Cliffs (2003)
22. Myers, G.J. (Revised by C. Sandler, T. Badgett, and T.M. Thomas): The Art of Software Testing, 2nd edn. John Wiley & Sons, Ltd, Chichester (2004)
23. Object Management Group (OMG). Common Object Request Broker Architecture (CORBA): Core Specification, Version 3.0.3 (16.08.2005) (March 2004), `http://www.omg.org/docs/formal/04-03-01.pdf`
24. Object Management Group (OMG). UML 2.0 Testing Profile (April 2004), `http://www.omg.org/cgi-bin/doc?ptc/2004-04-02`
25. Schieferdecker, I.: The UML 2.0 Test Profile as a Basis for Integrated System and Test Development. In: Proceedings by Köllen Druck+Verlag GmbH, Jahrestagung der Gesellschaft für Informatik, Bonn, Germany, vol. 35 (September 2005)
26. Schieferdecker, I., Dai, Z.R., Grabowski, J., Rennoch, A.: The UML 2.0 Testing Profile and its Relation to TTCN-3. In: Hogrefe, D., Wiles, A. (eds.) TestCom 2003. LNCS, vol. 2644, Springer, Heidelberg (2003)
27. Schieferdecker, I., Din, G.: A Metamodel for TTCN-3. In: Núñez, M., Maamar, Z., Pelayo, F.L., Pousttchi, K., Rubio, F. (eds.) FORTE 2004. LNCS, vol. 3236, Springer, Heidelberg (2004)
28. Schieferdecker, I., Din, G., Apostolidis, D.: Distributed Functional and Load tests for Web services. International Journal on Software Tools for Technology Transfer (STTT) (2004)
29. Schieferdecker, I., Grabowski, J.: The Graphical Format of TTCN-3 and its Relation to UML and MSC. In: Sherratt, E. (ed.) SAM 2002. LNCS, vol. 2599, Springer, Heidelberg (2003)
30. Schieferdecker, I., Grossmann, J.: Testing Embedded Control Systems with TTCN-3. In: Obermaisser, R., Nah, Y., Puschner, P., Rammig, F.J. (eds.) SEUS 2007. LNCS, vol. 4761, pp. 7–9. Springer, Heidelberg (2007)
31. Schieferdecker, I., Rennoch, A., Hoefig, E.: TTCN-3 — A Test Technology for the Automotive Domain. In: Proceedings by expert Verlag. Simulation und Test in der Funktions- und Softwareentwicklung für die Automobilelektronik, Berlin, Germany (March 2005)

32. Schieferdecker, I., Stepien, B.: Automated Testing of XML/SOAP based Web Services. In: Informatik Aktuell, Fachkonferenz der Gesellschaft für Informatik (GI) Fachgruppe Kommunikation in verteilten Systemen (KiVS), Leipzig, vol. 13 (February 2003)
33. Schieferdecker, I., Vassiliou-Gioles, T.: Realizing distributed TTCN-3 test systems with TCI. In: Hogrefe, D., Wiles, A. (eds.) TestCom 2003. LNCS, vol. 2644, Springer, Heidelberg (2003)
34. Schieferdecker, I., Vassiliou-Gioles, T.: Tool Supported Test Frameworks in TTCN-3. In: ENTCS (80). 8th Intern. Workshop in Formal Methods in Industrial Critical Systems, Røros, Norway (June 2003)
35. Eclipse Test & Performance Tools Platform Project (2004-2007), `http://www.eclipse.org/tptp/`
36. Vassiliou-Gioles, T., Din, G., Schieferdecker, I.: Execution of External Applications using TTCN-3. In: Groz, R., Hierons, R.M. (eds.) TestCom 2004. LNCS, vol. 2978, Springer, Heidelberg (2004)
37. Vouffo-Feudjio, A., Schieferdecker, I.: Test Pattern with TTCN-3. In: Grabowski, J., Nielsen, B. (eds.) FATES 2004. LNCS, vol. 3395, Springer, Heidelberg (2005)
38. World Wide Web Consortium (W3C) Recommendation: Extensible Markup Language (XML) 1.1 (2004), `http://www.w3.org/TR/2004/REC-xml11-20040204/`
39. Zander, J., Dai, Z.R., Schieferdecker, I., Din, G.: From U2TP Models to Executable Tests with TTCN-3 — An Approach to Model Driven Testing. In: Khendek, F., Dssouli, R. (eds.) TestCom 2005. LNCS, vol. 3502, Springer, Heidelberg (2005)

Glossary

Acronym	Explanation
ASN.1	http://www.asn1.org/) Abstract Syntax Notation One: an ITU standardized data type specification and coding language, which is used particularly in telecommunications
ATS	Abstract Test Suite: a collection of abstractly defined, thus system-, test system-, and implementation-independent test cases, typically described in TTCN-3, U2TP or other proprietary test notations
ETS	Executable Test Suite: a collection of executable test cases, which are typically generated automatically from abstract test cases
ETSI	http://www.etsi.org) European Telecommunication Standards Institute: an independent, non-profit organization, whose mission is to produce telecommunications standards
FIFO	Queuing Discipline First-In-First-Out
GFT	Graphical Presentation Format of TTCN-3: the graphical format of TTCN-3, especially for the visualization, development and documentation of test behaviours
IDL	http://www.omg.org/gettingstarted/omg_idl.htm) Interface Definition Language: IDL is an OMG standardized specification language for object interfaces
ISO	http://www.iso.org International Organization for Standardization: the world's largest developer of standards with its principal activity being the development of technical standards
ITU	http://www.itu.ch International Telecommunication Union: an international organization within the United Nations System where governments and the private sector coordinate global telecom networks and services
MSC	http://www.sdl-forum.org/MSC/ Message Sequence Charts: a language standardized by ITU for the description and specification of the interactions between system components based on sequence diagrams, which is adopted to a big extend in UML 2.0
MTC	Main Test Component: the main test component of a TTCN-3 test case, which steers and controls the test configuration and test run
OMG	http://www.omg.org Object Management Group: an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications
PTC	Parallel Test Component: a parallel test component of a TTCN-3 test case, which performs test behaviour in parallel to other test components and which determines its own, local verdict about the correctness of the tested system
SUT	System Under Test: the system to be tested - in dependence of the testing level a system component, a set of system components, a subsystem, a system or a composition of systems, which is/are to be tested
TCI	TTCN-3 Control Interfaces: the control interfaces of TTCN-3, which support the test management, the handling of test components and the coding/decoding of test data

Acronym	Explanation
TFT	Graphical Presentation Format of TTCN-3: the graphical format of TTCN-3, especially for the visualization, development and documentation of test data and type structures
TRI	TTCN-3 Runtime Interfaces: the run time interfaces of TTCN-3, which support the communication with the SUT, the time handling during test execution and the integration of external functionalities
TSI	Test System Interface: the interface to the SUT, which is taken as black box or grey box when testing with TTCN-3, and over that the interaction with the test system for the evaluation of functionality, the efficiency, the scaling, etc. of the SUT is performed
TTCN-3	http://www.ttcn-3.org) Testing and Test Control Notation: standardized test specification and implementation technology by ETSI (ES 201 873 series) and by ITU (Z.140 series). TTCN-3 is a technology for the development, specification, visualization and documentation of detailed test specifications
UML	http://www.uml.org/ Unified Modelling Language: UML is a non-proprietary modelling and specification language. The use of UML is not restricted to software modelling. It can, for example, be used for modelling hardware and is commonly used for business process modelling and organizational structure modelling. The UML is an open method used to specify, visualize, construct and document the system artefacts. The current version is UML 2.0
U2TP	http://www.fokus.fraunhofer.de/u2tp UML 2.0 Testing Profile: the standardized testing profile of UML 2.0 by OMG. U2TP defines a language for designing, visualizing, specifying, analyzing, constructing and documenting the artefacts of test systems. It is a test modelling language that can be used with all major object and component technologies and applied to testing systems in various application domains. U2TP can be used stand alone for the handling of test artefacts or in an integrated manner with UML for a handling of system and test artefacts together
XML	http://www.w3.org/XML/ Extended Markup Language: XML is a standardized markup language for documents containing structured information by the World Wide Web Consortium