# The Testing and Test Control Notation TTCN-3 and its Use

Ina Schieferdecker, Alain-Georges Vouffo-Feudjio

**Abstract.** In system engineering, testing is a generally accepted approach to validate systems and system components. Assured quality of system and system components is particularly important, as the time-to-market becomes even shorter. A systematic approach to testing distributed systems is essential, so that the requirements of the market can be fulfilled. The Testing and Test Control Notation (TTCN-3) has been developed to address testing needs of modern telecom and datacom technologies. It is a test specification and implementation language for the development of tests for reactive and/or distributed systems. Typical areas of application are protocol and interface testing (including mobile and Internet protocols), service testing (including Web services), software component, integration and system testing and testing of component platforms (including SOA, CORBA, and EJB). The TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system, and also performance testing.

## 1. Introduction

The TTCN-3 language was created due to the imperative necessity to have universally understood language syntax able to describe test behavior specifications. 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 is the successor of the Tree and Tabular Combined Notation (TTCN) that was originally developed by ISO (the International Standards Organisation) from 1984 to 1997 as part [22] of the widely-used OSI (Open Systems Interconnection) Conformance Testing Standard and Methodology. TTCN has been a rather static test specification language in tabular format that emphasized the tree structure in testing of pairs of stimulus and several responses. TTCN was further evolved into TTCN-2 and TTCN-2++ [17] by ETSI (the European Telecommunication Standards Institute) adding modularization, concurrent tests, and support for ASN.1 (Abstract Syntax Notation One [23], [8]). The development towards TTCN-3 started at ETSI 1998. The first version of TTCN-3 was published 2000. Since then, TTCN-3 was continuously updated and extended. The most recent version is v4.2.1 [2]-[14] from 2010.

TTCN-3 enables systematic, specification-based testing for various kinds of tests including e.g. conformance, interoperability, regression, robustness, performance, and scalability testing. The 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 simple centralized, but also of complex distributed test behaviors 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. The TTCN-3 language is characterized by a well-defined syntax and operational semantics, which provide a precise understanding of TTCN-3 test specifications and their execution.

For testers, the task of specifying tests consisting of test data, test configurations, and test behaviors is easy to perform. A test system can communicate with an SUT (System under Test) synchronously or asynchronously. The data transmitted between the test system and the SUT can be defined by templates, which can make use powerful matching mechanisms to define the precise set of used or expected data. A verdict handling mechanism is provided to assess the reactions of the SUT. The types and test data can be either described directly in TTCN-3 or can be imported from other data type languages. Moreover, various forms of parameterization for templates, testcases or functions and even for TTCN-3 modules are supported to enable the flexible reuse and adaptation of test specifications within different test campaigns. The selection of test cases to be executed can be either controlled directly by the user during a test run or can be specified in the control part of a module.

Figure 1 shows an overview of the TTCN-3 language. The TTCN-3 metamodel defines the various concepts of TTCN-3 and their relations [43]. The concepts can be presented in textual [2], tabular [3], graphical [4] or another proprietary presentation format. In addition, the TTCN-3 supports the import of data types and values specified in ASN.1, IDL (Interface Definition Language [20], [9]) and XML (Extended Markup Language [24] [10]). Also other data formats such as SQL (Structured Query Language [20]) or WSDL (Web Services Description Language [32]) can be supported [33].

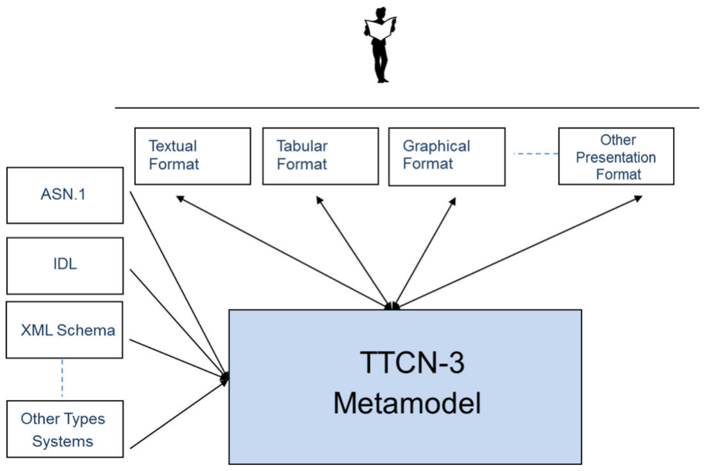


**Figure 1.** Basic structure of the TTCN-3 technology

The ETSI standard for TTCN-3 comprises ten parts which are grouped in the "Methods for Testing and Specification; The Testing and Test Control Notation version 3" set of documents:

1. The TTCN-3 Core Language [2]. This document specifies the syntax of the TTCN-3 language and how test suites are defined by a set of TTCN-3 modules.
2. The Tabular Presentation Format (TFT [3]). TTCN-3 offers other presentation formats than the textual one. The tabular format is similar in appearance and functionality to earlier versions of TTCN-2 (Tree and Tabular Combined Notation [24]). It was designed for users that prefer the previous TTCN-2 style of writing test suites. A TTCN-3 module is presented in TFT as a collection of tables. This format however did not find many users and has its latest release in 2007.
3. The Graphical Presentation Format (GFT [4]). It is the second presentation format of TTCN-3 which is based on MSCs (Message Sequence Charts [20]). In GFT, TTCN-3 behavior definitions are represented by sequence diagrams. GFT has been used a lot up until the UML Testing Profile [28] became more and more in use as another frontend for TTCN-3.
4. The operational semantics [5]. Part 4 describes the meaning of TTCN-3 behavior constructs and provides a state-oriented view of the execution of a TTCN-3 module.
5. The TTCN-3 Runtime Interfaces (TRI [6]). A TTCN-3-based test system requires a platform-specific adaptation to the SUT to handle communication and timing matters. TRI defines a set of interfaces to adapt a test system to an SUT, which is realized by a system adapter.
6. The TTCN-3 Control Interfaces (TCI [7]). In addition, a TTCN-3-based test system requires an adaptation to the test platform addressing test management, test component handling, external data encoding/decoding and logging. The set of interfaces for that defined by TCI is realized by a test

adapter.

Additional parts of the standard concerning the language mappings and other language features have been developed and encompass the following:

7. The language mapping from ASN.1 modules to TTCN-3 [8].
8. The language mapping from IDL specifications to TTCN-3 [9].
9. The language mapping from XML schemata to TTCN-3 [10].
10. The support for documentation tags in TTCN-3 modules [11].

In 2010, extension packages have been added to TTCN-3. An extension package defines an additional, but optional set of concepts for TTCN-3 useful in dedicated application domains. For example, real-time concepts are useful to test embedded or time-critical systems, but are not needed for testing in general. The extension packages for TTCN-3 include:

11. The support for static configuration and deployment [12].
12. The support for advanced parameterization including for example type parameterization [13].
13. The support for behavior types to allow for example dynamic assignment of functions [14].
14. The support for TTCN-3 performance and real-time testing [15].

The TTCN-3 is a continuously maintained test technology to which new standard parts are added when needed by the industrial users. Up-to-date information is available at the TTCN-3 home page hosted by ETSI [1].

## 2. The Concepts of TTCN-3

The TTCN-3 core language is a test specification and test implementation language, which has the look and feel of a modern programming language. Besides typical programming constructs, the TTCN-3 contains all the important features necessary to specify test data and test procedures for various kinds of tests including test verdicts, data templates, timer handling, test components, synchronous and asynchronous communication, logging, and alike. A TTCN-3 test specification can import definitions from other modules; can define types and test data; specify test behavior in functions, altsteps and test cases; and define the selection, parameterization and execution of test cases in a control part (see Figure 2).

### Modules

The top-level building-block of a TTCN-3-based test specification is a `module`. A module contains all other TTCN-3 constructs, but cannot contain sub-modules. It can import completely or partially definitions of other modules. The modules can be parameterized with values that are supplied by the test environment at runtime. For parameters, default values can be specified in addition.

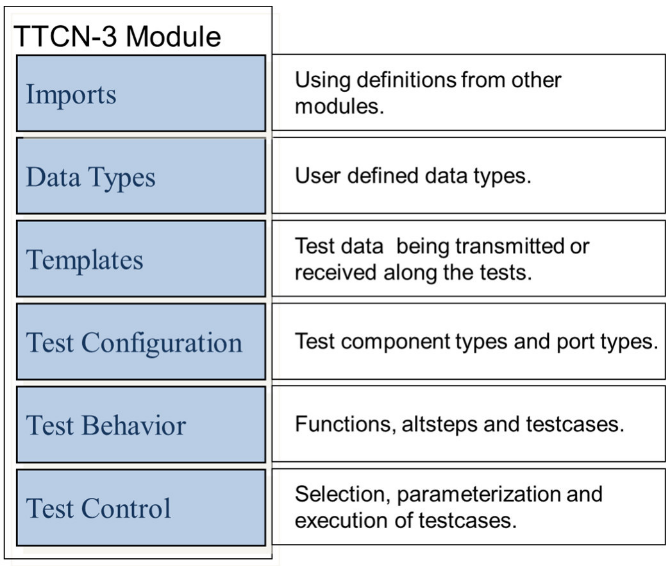

TTCN-3 Module
Imports
Using definitions from other modules.
Data Types
User defined data types.
Templates
Test data being transmitted or received along the tests.
Test Configuration
Test component types and port types.
Test Behavior
Functions, altsteps and testcases.
Test Control
Selection, parameterization and execution of testcases.

**Figure 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 throughout the module and can be imported from other modules. The control part is the main program of a module and describes the execution sequence of the test cases or functions. It can access the verdicts delivered by test cases and, according to them, it can decide the next steps of test execution. The test behaviors in the TTCN-3 are defined within functions, altsteps and testcases. The control part of a module may call any testcase or function defined in or imported by the module to which it belongs.

## Test System

A testcase is executed by a test system. The TTCN-3 allows the specification of static non-distributed and of dynamic 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 to the SUT.

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 behavior of the MTC is specified by the body of a test case. 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 can have a set of local ports. Each port has an in- and an out-direction. The in-direction is modeled as an infinite FIFO queue, which stores the incoming information until it is processed by the test component owning that port. The out-direction is directly linked to the communication partner being another test component or the SUT, i.e. outgoing information is not buffered.

During test execution, the 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. The TTCN-3 distinguishes between the abstract and the real test system interface. The abstract test system interface is modeled 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 the TTCN-3, connections and mappings are created and destroyed dynamically at runtime. There are no logical restrictions on the number of connections and mappings a component may have, but only limitations of the used test device(s). A component (and even a port) may be connected to itself. One-to-many connections are allowed. These can be used for one-to-one communication, where during test execution the communication partner has to be specified uniquely and for one-to-many communication by means of multicasting and broadcasting. 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 where sender and receiver proceed independently of each other (except that a message can only be received after being sent). Procedure-based communication is based principally on calling procedures in remote entities synchronously. In procedure-based communication, the calling component is principally blocked up until receiving a response or exception to its call. In TTCN-3, both the client side, i.e. calling procedures and awaiting responses or exceptions, and the server side, i.e. accepting calls and issuing responses or raising exceptions, can be specified for testing purposes.

## Test Cases and Test Verdicts

Test cases define test behaviors 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.

The 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 runtime 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 a test verdict to become 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 runtime environment maintains a global test verdict for each test case. The global test verdict is not accessible by 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.

## Alternatives and Snapshots

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

## Default Handling

In the TTCN-3, defaults are used to recurring sets of alternatives, which are often used to handle unexpected responses or absent responses. Default behavior can be specified by `altsteps` and activated as defaults. For each test component, the defaults, i.e., activated altsteps, are stored in a default list in the order of their activation. The TTCN-3 operations `activate` and `deactivate` operate on that default list. An activate operation appends a new default to the end of the list and a deactivate operation removes a default from that list.

The activated defaults are invoked at the end of each alt statement, if due to the current snapshot none of the alternatives in the alt matches. Like the alternatives in an alt, activated defaults and their alternatives are evaluated one by one. If one of the alternatives matches, it is being executed. If none of the alternatives of an activated default matches, a new snapshot is taken and the alt statement is being executed again.

In case of a successful match in a default, the default may stop the test component by means of a `stop` statement, it may `repeat` the alt statement to be executed once more or it may continue the main control flow of the test component immediately after the alt statement from which the default mechanism.

The TTCN-3 uses altsteps to specify default behaviors or to structure sets of alternatives. The semantics of altsteps is closely related to alternatives and snapshots. Like an alt statement, an altstep defines an ordered set of alternatives. The difference is that no snapshot is taken when entering an altstep. The evaluation of the altstep alternatives is based on the current snapshot from the alt statement, from which it is invoked. Conceptually, the alternatives of the altstep are inserted at the end into the set of alternatives of the alt statement. It is also possible to invoke an altstep like a function. In this case, the altstep is interpreted as an alt statement which only invokes the altstep, i.e., the altstep alternatives are the only alternatives of that alt statement.

## Communication Operations

Communication operations are important for the specification of test behaviors. The 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.

Message-based communication in the TTCN-3 is asynchronous communication meaning that the sending of messages is non-blocking; after sending the message, the system does not wait for a response. A receive operation blocks the execution until a matching message is received at the specified port. A receive operation defines a matching part for valid messages to be received. Optionally, an address can be specified to identify the connection from which the message is being expected. The following operations are defined for message-based communication:

- `send` to direct a message to the SUT or another PTC,
- `receive` to accept a message from the SUT or another PTC, and
- `trigger` to ignore incoming messages up until the expected one arrives.

Procedure-based communication in the TTCN-3 is synchronous communication meaning that the invocation of calls is basically blocking. For synchronous communication the following operations are defined:

- `call` to invoke a remote procedure,
- `getcall` to accept a call from,
- `reply` to provide a reply to a call,
- `getreply` to accept a reply to a call,
- `raise` to provide an exception to a call, and
- `catch` to accept an exception to a call.

**Test Data Specification**

A test system needs to exchange data with the SUT and between PTCs. The communication with the SUT can be either asynchronous, by sending/receiving messages to/from the SUT or synchronous, by calling procedures of the SUT or accepting procedure calls from the SUT. In these cases, the test data has to be defined by the test system according to the SUT specification. The TTCN-3 offers different constructs to describe the test data: types, templates, variables, procedure signatures, etc. They can be used to express protocol message, service primitives, procedure invocations, exception handling, and alike. Besides this, the TTCN-3 offers also the possibility to import data described in other languages such as ASN.1, IDL, or XML.

The TTCN-3 provides a number of basic and predefined types, based on which user-defined types can be specified. Most of types are similar to types of well-known programming languages such as Java or C. Some of them however are test domain specific:

- `port` types define types of messages or procedures that can be communicated via port (instances) of a given port type in a communication between PTCs or with the SUT.
- `component` types define the local ports, variables, constants, and timers of MTCs or PTCs of a given component type.
- The `verdicttype` is an enumeration of the possible verdicts that can be the result of a test case.
- The `anytype` is a union of all known TTCN-3 types and is used to specify generic data that are evaluated when the concrete data is known.
- The `default` type is used for handling default alternatives.

The TTCN-3 also supports ordered structured types such as `record`, `record of`, `set`, `set of`, `enumerated` and `union`. For procedure-based communication, procedure signatures can be defined. Signatures are characterized by their name, optional list of parameters, optional return values and optional list of exceptions.

A `template` represents test data of a given type used to define messages to be sent or received. Templates define either distinct values that are to be transmitted or sets of values (by use of wildcards or other matching mechanisms) against which received messages are evaluated to match or not. When comparing a received message with a template, the template matches if the message is one of the values in the value set defined by the template. Templates can be specified for basic types, user-defined types or for procedure signatures. They can be parameterized and reused or modified in other template definitions.

As not all details of the TTCN-3 can be given here, please refer to the standard series [2]-[14], further papers [44][45][48] and tools [33] to get a deeper insight into the test technology.

## 3. An Introductory Example

In the following, a small test specification in TTCN-3 is given in order to give an insight into this technology. The well-known triangle testing example by G.J. Myers [41] for the classification of triangles is used. A triangle is defined by its side lengths und has to be classified into scalene, isosceles, and equilateral triangles. In addition, the data used to characterize a triangle can be incorrect, or the data is correct but does not define a triangle.

When testing software that classifies triangles, the question of typing of the SUT has to be posed. In TTCN-3, you are explicitly defining the SUT type system – in this case by defining type `Triangle` to consist of three `integer` values of unlimited size and by defining type `Classification` to specify the possible triangle classifiers:

```
type integer Triangle[3] (0..infinity);
// whole numbers to characterize a triangle
type enumerated Classification {
// properties of a triangle
     syntacticallyIncorrect,
     noTriangle,
     scaleneTriangle,
     isoscelesTriangle,
     equilateralTriangle
  }
```

Subsequently, the interface `DetermineTriangle` to the SUT can be defined in terms of a message-based port that transmits values of `Triangle` in the in direction and values of `Classification` in the out direction. A test component for the SUT uses that port as defined by the test component type `TriangleTester`.

```
type port DetermineTriangle message
   { out Triangle; in Classification }
// Interface to SUT
type component TriangleTester
   { port DetermineTriangle b }
// Test Component
```

These definitions are sufficient to define concrete tests for the SUT as demonstrated by the testcase `Simple` that cheks for a concrete equilateral triangle:

```
testcase Simple() runs on TriangleTester {
// a simple test case
      b.send(Triangle: {2,2,2});
      // the triangle to be checked
      b.receive(Classification: equilateralTriangle);
      // the expected response
      setverdict(pass); // a successful test
}
```

This test specification given above is wiring the test data firmly with the testing activity, which is neither useful nor easy to maintain. Another method would be to separate the test data from the test behaviour. In order to so, we define an additional type `TestVector` that pairs triangle integer triples and their classifications. Concrete triangle triples and classifiers are then defined in form of templates `valid1`, `valid2`. etc.

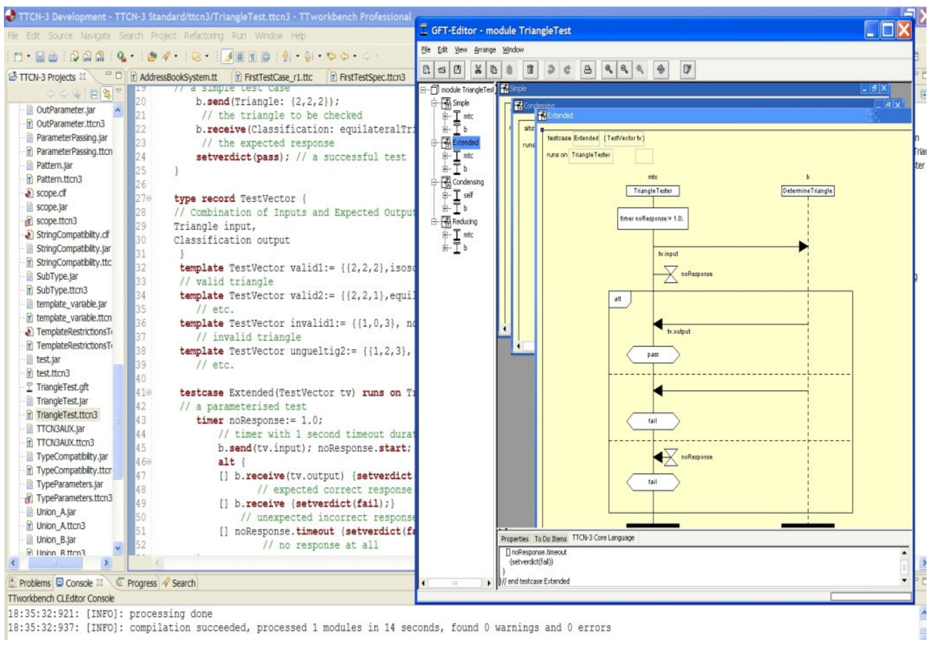
```
type record TestVector {
// combination of inputs and expected outputs
    Triangle input,
    Classification output
}
template TestVector valid1:= {{2,2,2}, isoscelesTriangle};
// valid triangle
template TestVector valid2:= {{2,2,1}, equilateralTriangle};
// etc.
template TestVector invalid1:= {{1,0,3}, noTriangle };
// invalid triangle
template TestVector invalid2:= {{1,2,3}, noTriangle };
// etc.
```

In addition, a test specification should define how unexpected or no responses from the SUT are to be evaluated. Unexpected responses are handled by a receive any statement (2nd alternative in the alt statement), which is evaluated only after the alternative for the correct response (1st alternative in the alt statement). In order to identify no responses from the SUT, the local timer `noResponse` is defined, started and observed for timeout whenever a triangle classification query is given to the SUT.

```
testcase Extended(TestVector tv) runs on TriangleTester {
// a parameterised test
    timer noResponse:= 1.0;
    // timer with 1 second timeout duration
    b.send(tv.input); noResponse.start;
    alt {
    [] b.receive(tv.output) {setverdict(pass);}
          // expected correct response
    [] b.receive {setverdict(fail);}
          // unexpected incorrect response
       [] noResponse.timeout {setverdict(fail);}
          // no response at all
   }
}
```

We could further extend the examples by e.g. using parameterization to have additional flexibility in the test data or by giving queries with two parallel testing components to the SUT to check concurrent usages.

By use of TTCN-3 tools [33], TTCN-3 test specifications as those defined for the triangle example can be developed, implemented and executed efficiently.



```
    // a simple test case
        b.send(Triangle: {2,2,2});
         // the triangle to be checked
        b.receive(Classification: equilateralTri
         // the expected response
        setverdict(pass); // a successful test
    }

    type record TestVector {
    // Combination of Inputs and Expected Output
    Triangle input,
    Classification output
     }
     template TestVector valid1:= {{2,2,2},isosc
     // valid triangle
     template TestVector valid2:= {{2,2,1},equil
        // etc.
     template TestVector invalid1:= {{1,0,3}, no
        // invalid triangle
     template TestVector ungueltig2:= {{1,2,3},
        // etc.

     testcase Extended(TestVector tv) runs on Tr
     // a parameterised test
        timer noResponse:= 1.0;
            // timer with 1 second timeout durat
            b.send(tv.input); noResponse.start;
            alt {
            [] b.receive(tv.output) {setverdict
                  // expected correct response
            [] b.receive {setverdict(fail);}
                // unexpected incorrect response
            [] noResponse.timeout {setverdict(fa
                   // no response at all
```

```
[] noResponse.timeout
  {setverdict(fail)}
}
}// end testcase Extended
```

```
18:35:32:921: [INFO]: processing done
18:35:32:937: [INFO]: compilation succeeded, processed 1 modules in 14 seconds, found 0 warnings and 0 errors
```

**Figure 3.** A TTCN-3 Integrated Development Environment based on Eclipse

## 4. TTCN-3 Semantics and its Application

The semantics of TTCN-3 is defined by an operational semantics [5], which defines the meaning of TTCN-3 behaviours in an intuitive and unambiguous manner. It is not in itself a formal semantics, so that the abilities for formal verification and validation are limited. In fact, the operational semantics of TTCN-3 provides a state oriented view on the execution of TTCN-3 modules. It defines soley the meaning of behaviour in TTCN-3, i.e. functions, altsteps, test cases, module control and behavioral statements like communication operations or control flow statements.

The operational semantics uses flow graphs of the TTCN-3 module to be executed. A flow graph is a directed graph that consists of labelled nodes and labelled edges, which describe the possible flow of control during the execution of the behaviour represented by the flow graph. The operational semantics is used to construct runtime environments (see the following section), test solutions (see e.g. [50]), and provides a basis for formal semantics definitions for additional validation and verification purposes.

A major research direction is on analyzing and improving the quality of TTCN-3 test suites. Herein, the operational semantics of TTCN-3 has been extended with constraints that further precise data and behavior in TTCN-3 test suites. An overview on optimizing test quality for maintenance purposes is given in [54]: metrics, patterns and anti-patterns have been defined, which constitute the basis to identify code smells and propose refactorings for an assessment and automated restructuring of test suites [55]. The use of guidelines for improved test suite design and maintenance and their automated verification is presented in [56]. [57] presents an approach to analyze and optimize the coverage of test data in TTCN-3 based test suites.

Furthermore, verification of test behavior in TTCN-3 has been subject to research. In particular, behavioral anomalies like inconsistencies in test behavior within and across test cases have been identified. For example, response inconsistencies have been defined in [58] and analyzed by model checking.

Another major research direction along formalized testing based on TTCN-3 is on testing real-time, embedded systems: In [51], the authors provide a semantics for host-based testing with simulated time and a simulated-time solution for distributed testing with TTCN-3. [52] defines a formal semantics based on timed automaton for testing of real-time systems with an extended version of TTCN-3. Some of these concepts became later part of the real-time concepts for TTCN-3 in [15]. [53] provides concepts for testing continuous behavior of embedded systems and a formal semantics based on stream processing functions as used by the time partition (TPT) test method. This was further extended into a semantics definition based on hybrid automata and is now being considered for addition to real-time concepts for TTCN-3 in [15] too.

## 5. A Distributed Test Platform for the TTCN-3

The full potential of the TTCN-3 can be realised by a test platform that supports test execution across multiple, potentially heterogeneous test interfaces and devices. By that, not only aFehler potentially distributed and heterogeneous configuration of an SUT can be reflected by the test system by use of interconnected test devices, but also performance and scalability issues of the test system can be addressed.

We have developed such a distributed test platform for the TTCN-3 (depicted in [45]) by use of a CORBA (Common Object Request Broker Architecture [27]) middleware. We use this platform in particular for performance, scalability, load and stress tests, but also for non-distributed functional tests or for cases were single test devices and/or interface are needed only. We do not intend to make an exhaustive presentation of all implementation details; rather, we provide an overview of the most important aspects.

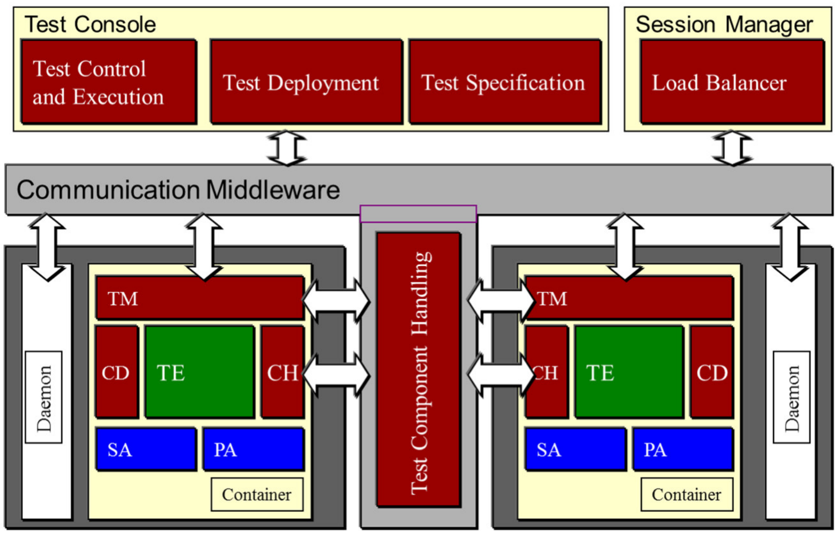


**Figure 4.** A distributed test platform architecture for the TTCN-3

The Test Console is the control point of our platform and is an IDE (Integrated Development Environment), which provides support to specify TTCN-3 test cases, create test sessions, deploy test suites into containers and control the test execution.

Daemons are standalone processes installed on any test device. They manage the containers which belong to different sessions. Containers intercede between the test console and components, providing services transparently to both, including transaction support and resource pooling. From the test view point, containers are the target operational environment and comply with the TCI (TTCN-3 Control Interfaces) standard for a TTCN-3 test execution environment. Within a container, the following specific test system entities exist:

- `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 realizing the TTCN-3 semantics.
- `CH` (Component Handler) handles the communication between test components. The CH interface contains operations to create, 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.
- `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.
- `CD` (Encoding/Decoding) encodes and decodes values according to assumed or given types. Every TTCN-3 value is encoded into bitstrings for communication with the SUT. The received data is decoded into abstract TTCN-3 types and values.

The container sub-entities are functionally bound by the TRI and TCI interfaces and communicate with each other via the CORBA platform. The session manager coordinates test components (and their behaviors) chosen at test deployment (and not in the test specification). The TRI system adapter realizes SUT-specific communication and timing. It is typically provided by the test developer. The TCI test adapter realizes the adaptation to the test management environment and to the test devices. It is typically provided by the test vendor.

The whole execution platform is implemented in Java JDK1.4 [34]. For the communication between containers we used the CORBA implementation provided with JDK1.4. Following the standard specification helped us to optimize our execution environment for common operations across the test execution. The advantage of adopting a pure Java solution for our TTCN-3 platform gave us hardware and operating systems independence, so that we are able to run tests in various scenarios and deployment configurations for different SUTs.

## 6. Case Study I: Testing of OSA/Parlay Services

In the following, we describe different case studies that demonstrate the application of the TTCN-3 to test various target systems. The first case study is on testing OSA/Parlay (Open Service Architecture [16]) services. OSA/Parlay defines a set of standard APIs (Application Programming Interfaces) jointly developed by 3GPP (the Mobile Systems 3rd Generation Partnership Project), ETSI and the Parlay Group to ease the creation of telecom services. The APIs provide a common interface to the underlying network, offering software developers major support to implement their telecom applications.

The TTCN-3 is particularly suited for testing client-server applications when the communication interfaces between client and server are well specified, i.e. an interface definition language like IDL or WSDL is given. Based on these specifications, functional tests for operations the server provides to its clients as well as of functional tests for client-side applications that use the server features can be generated.

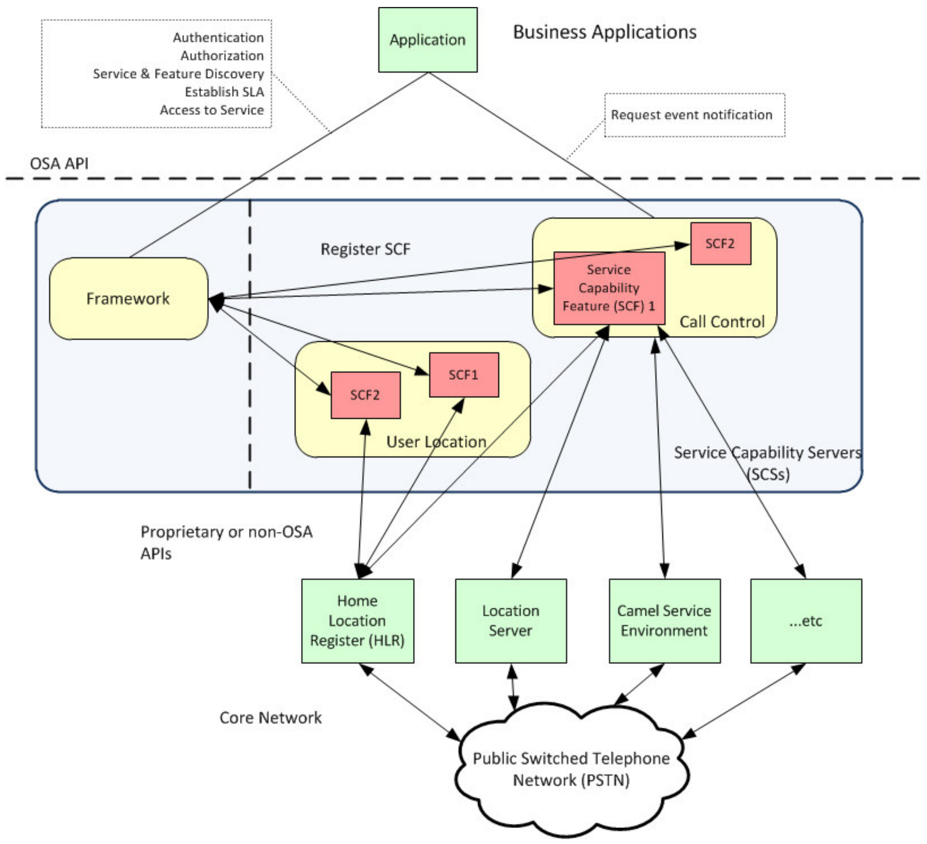

Authentication
Authorization
Service & Feature Discovery
Establish SLA
Access to Service
Application
Business Applications
Request event notification
OSA API
Register SCF
Framework
SCF2
Service Capability Feature (SCF) 1
Call Control
SCF1
SCF2
User Location
Service Capability Servers (SCSs)
Proprietary or non-OSA APIs
Home Location Register (HLR)
Location Server
Camel Service Environment
...etc
Core Network
Public Switched Telephone Network (PSTN)

**Figure 5.** The OSA/Parlay architecture

The architecture of OSA/Parlay is depicted in Figure 5. OSA/Parlay defines three types of components: applications, service capability servers, and a framework.

- `Applications` represent any business application which uses the core network. The application connects and authenticates to the framework. After successful authentication, the application uses the framework for service discovery.
- The `services` that the network can provide and the applications can request are called service capability features (SCFs). An application can use

an SCF via calls to its interfaces, which are described in the OSA technical specifications.

- A `framework` provides applications with directory services and connection factories, and mediates between the SCFs and the applications. The only thing the application needs to know is where to find the factory, and what features it is looking for.

The aim of this case study was to realize a TTCN-3-based test framework for the ctesting of OSA/Parlay-based services and applications. The TTCN-3 is used to specify tests for the operations of the framework, the services and/or the application APIs. The test system can be in two different roles:

- the role of a service consumer that communicates via the OSA/Parlay API with the framework or with the services, or
- the role of the framework or of a service that validates the behavior of the application.

The application tested in the case study was a small application that uses OSA/Parlay to establish a call between two users connected to the network. The test system emulates the behavior of the SCFs (Service Capability Features), which are used to establish calls between two users.

The tool environment for testing the OSA/Parlay application with TTCN-3 uses the TTCN-3 to Java compiler TTthree provided by Testing Technologies [33] and an IDL-to-TTCN-3 plugin and a test adaptor for CORBA provided by Fraunhofer FOKUS [36].

In a first step, the OSA/Parlay IDL specifications have been compiled to TTCN-3 using the IDL-to-TTCN-3 plugin based on the standardized mapping rules [9]. As a result of this step, the data types and structural information of the OSA/Parlay interfaces are available in TTCN-3. In a second step, the functional behavior of the application along selected scenarios has been specified in TTCN-3. The scenarios contain operations to be called at the OSA/Parlay interfaces, timeouts, default behaviors, etc. Each test case consists of a preamble, a test body, a postamble and the assignment of a final verdict. In a last step, the concrete test data for each of the method calls and messages to be sent and received has been specified.

A hybrid approach for the test tools has been selected due to the complex behavior of the test preamble: To emulate the complex behavior of the entire OSA/Parlay framework with the test tools (in TTCN-3) would not have been feasible for this case study. Thus, a Java implementation of the OSA/Parlay framework has been integrated into the test tool chain for handling the test preamble (in particular authentication and service discovery). The actual test bodies reflecting the main concerns of the test scenarios have been implemented in TTCN-3.

This case study demonstrated how TTCN-3 can be used to develop test automation for systems and services that use a synchronous communication scheme following request-response scenarios. In addition, it showed that the language mappings (in this case for IDL) provide good means to generate the interface related test structures efficiently. Last but not least, the case study demonstrated that TTCN-3 only solutions are not always practical, but that the open system architecture of TTCN-3 enables an easy integration with non-TTCN-3 components. This case study has been further elaborated and lead to the definition of test patterns often used in test solutions for communication services infrastructures [49].

## 7. Case Study II: Testing of IMS Equipment

The purpose of this case study is to apply TTCN-3 for the definition of performance tests for IMS (IP Multimedia Subsystem [35]) equipment. The IMS is an open, standardized, operator-friendly, multimedia network architecture for mobile and fixed IP (Internet Protocol) services. IMS supports a rich set of services available to end users on either wireless or wired user equipment, which are provided via a uniform interface.

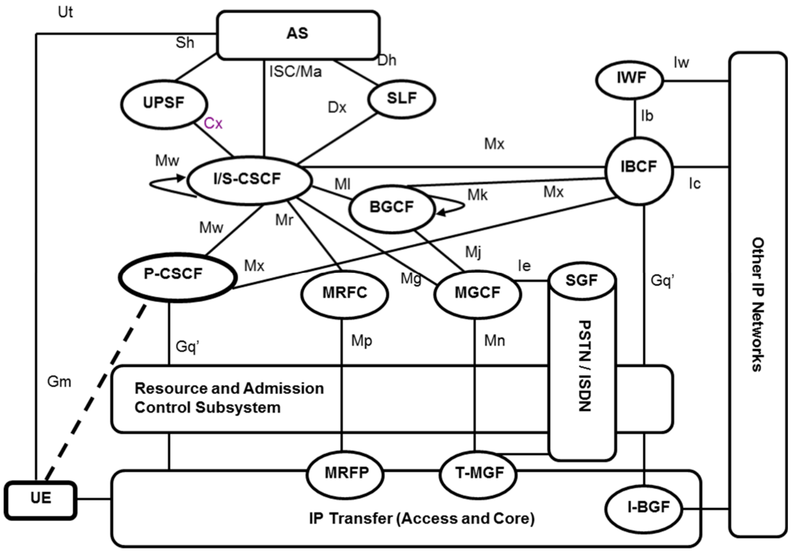


**Figure 6.** IMS Reference Architecture (from 3GPP)

The IMS reference architecture is presented in Figure 6. IMS defines that traffic between a UE (User Equipment) and the P-CSCF (Proxy-Call/Session Control Functions) is carried by an IPSec tunnel (i.e. via secure IP) corresponding to the UE. A P-CSCF is a SIP proxy (the Session Initiation Protocol, being a protocol e.g. for the control of multimedia sessions in the Internet [21]) that is the first point of contact for an IMS terminal. It is assigned to an IMS terminal during registration, and does not change for the duration of the registration. The I-CSCF (Interrogating Call/Session Control Functions) queries the HSS (Home Subscriber Server) to retrieve the user location, and then routes the SIP request to its assigned S-CSCF (Serving Call/Session Control Functions). S-CSCF is the central node of the signalling plane, which decides to which application server(s) the SIP message will be forwarded to, in order to provide their services.

The aim of this case study was to develop IMS performance tests in TTCN-3 that define the input stimulus to the SUT along precise transaction types and contents, statistical distributions for transaction types, arrival rates, and other relevant parameters. The tests specify how the traffic is to be provided to the SUT, define the measurements that have to be taken, and how the results are to be reported.

The performance tests measure the capacity of the control plane of an IMS system. This consists of the components that perform SIP-based signalling to each other and the database accessed by those components.

A call is a relationship between one or more subscribers, signalled and managed through the IMS system. The intent of a subscriber when using IMS is to make calls, so the obvious way to obtain realistic behavior in an IMS test is to model the behavior of calls. We use a voice call model describing the behavior of a call established for a conventional audio or multimedia conversation. 0 depicts the sequence of messages through which a voice call passes.

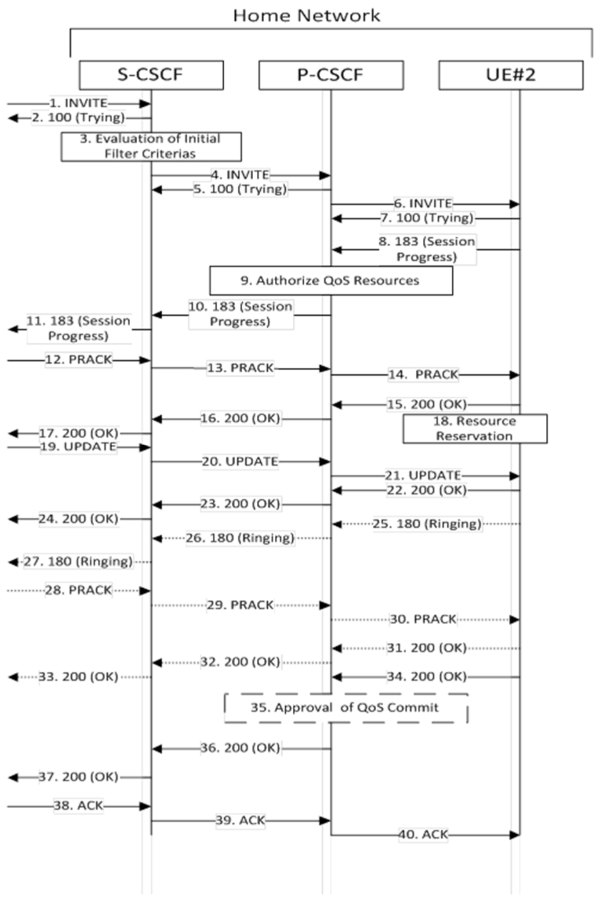


**Figure 7.** Sequence diagram of an IMS call

The realization of the call scenario is based on the SIP conformance test suite by ETSI [18] and on the SIP specification by IETF [21]. The type system and template definitions of the conformance tests are reused in the performance test specification for creation of inputs to the IMS system.

The call scenario uses two UEs (user equipments) which establish a call session. One of them plays the role of CALLER by sending the INVITE message to the IMS network and another one plays the role of CALLEE by accepting the call invitation and responding accordingly.

The performance tests scale by a parallel run of the call establishment procedure. The UEs (callers and callees) are implemented in TTCN-3 as PTCs which interact with the SUT via the system component. The case study implements two techniques to realize load test, where the SUT is put under increasing load while its responsiveness is being measured and analysed:

1. Use of parallel test component. This technique is the obvious method to implement parallel activities in TTCN-3. The callers and callees are implemented as PTCs and executed in parallel. Unfortunately, this technique has the disadvantage that the creation of too many test components may consume too many hardware resources (memory, CPU) which may lead to an overloading of the test system.
2. Reuse of test components after call establishment. This technique is used together with the first one implying that once a test component finished a call establishment procedure it is reused for a next call (of course with new identifiers and sets of data). Thus, only one test component may establish several calls in a second increasing the overall workload.

The key performance parameters evaluated within the case study have been

- the number of invitations (INVITE messages) per second, and
- delays of the reaction of the SUT measured under load

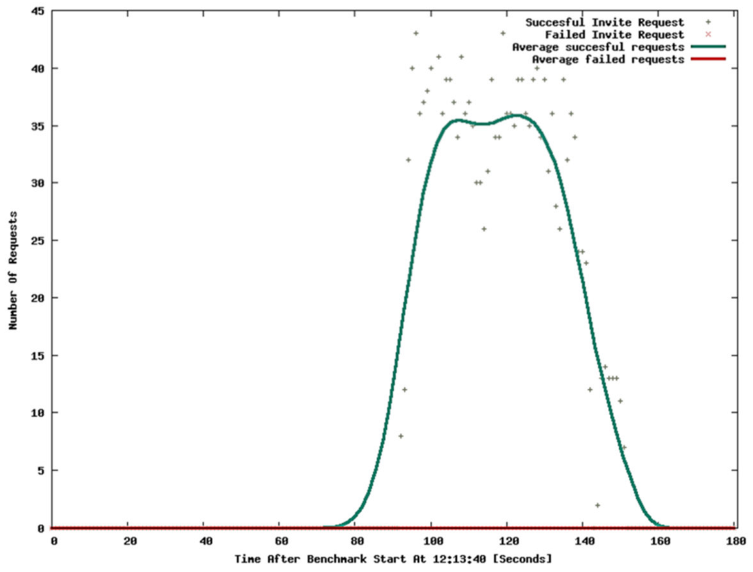


**Figure 8.** Number of requests per second

Figure 8 displays the number of INVITES per second applied for a period of 80 seconds.

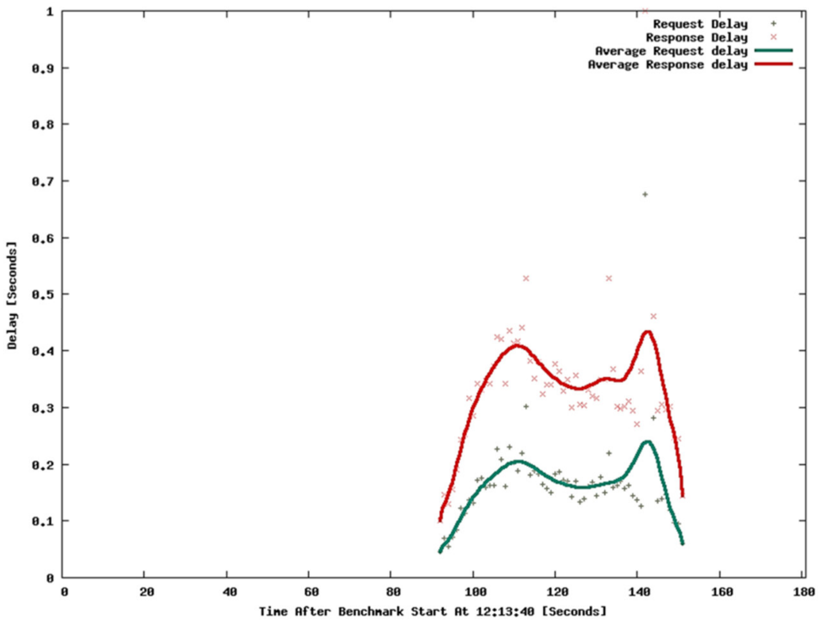


**Figure 9.** Delays of the IMS system reactions under load

Figure 9 shows the evolution of the delays in the reactions of the SUT when applying a load of 35 Invites/second. Reading the two graphs in parallel one can identify when the IMS system needs more time for internal computations.

This case study has demonstrated the power of TTCN-3 to not only specify and execute functional tests, but also performance and load tests. It uses the capabilities of TTCN-3 to specify distributed and concurrent test setups by use of parallel test components, which can be assigned different test behaviors. In addition, the case study shows the abilities of the TTCN-3 logging format defined in [7] to enable test reports that contain graph-based evaluations of the test execution and SUT timing.

This case study has been further extended and led to the evaluation of concrete IMS equipment provided by a hardware vendor. Additional results are described in [37].

## 8. Conclusion

TTCN-3 is the only standardized test specification and implementation technology being applicable to a wide range of test kinds for various system technologies. It has progressed a lot in the past: A plethora of tools is available that support TTCN-3 in different scale. Various test solutions are based on TTCN-3. There are plans to apply TTCN-3 in additional domains such as scientific computing or cloud computing.

Still, more needs to be done. In particular people need to be trained in order to enable an efficient use and adoption of this test technology. The established TTCN-3 Certificate [19] provides a good basis – however more reading and training material and training courses should be provided.

Also, TTCN-3 is rarely lectured at universities – although free tools help in this. In particular, it is not enough to spread the knowledge about TTCN-3 and its success stories, but also a thorough methodology including guidelines and best practices for the application of TTCN-3 should be provided.

A strong basis for the further adoption of TTCN-3 will be an ongoing maintenance and evolution of TTCN-3. Although already a quite exhaustive and powerful set of testing concepts is supported by TTCN-3, still more requirements appear – for example to have native support for object-oriented data types when interacting with the SUT. However, like in the past every new concept is very critically reviewed and discussed before being added to the core language and its execution interfaces or to one – if not to a new – extension package of TTCN-3.

## References

[1] ETSI TTCN-3 Home Page, http://www.ttcn-3.org.

[2] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", July 2010.

[3] ETSI ES 201 873-2: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: Tabular Presentation Format", Febr. 2007.
[4] ETSI ES 201 873-3: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: Graphical Presentation Format", Febr. 2007.
[5] ETSI ES 201 873-4: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Operational Semantics", July 2010.
[6] ETSI ES 201 873-5: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI)" , July 2010.
[7] ETSI ES 201 873-6: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI)" , July 2010.
[8] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3", July 2010.
[9] ETSI ES 201 873-8: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping", July 2010.
[10] ETSI ES 201 873-9: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 9: Use XML with TTCN-3", July 2010.
[11] ETSI ES 201 873-10: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 10: TTCN-3 Documentation Comment Specification", July 2010.
[12] ETSI ES 202 781: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Configuration and Deployment Support", July 2010.
[13] ETSI ES 202 784: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Advanced Parameterization", January 2010.
[14] ETSI ES 202 785: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: Behavior Types", January 2010.
[15] ETSI ES 202 782: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; TTCN-3 Language Extensions: TTCN-3 Performance and Real Time Testing", July 2010.
[16] ETSI ES 203 915 V1.1.1: OSA/Parlay Specification, April 2005.
[17] ETSI TR 101 666: Information technology Open Systems Interconnection Conformance testing methodology and framework; The Tree and Tabular Combined Notation (TTCN) (Ed. 2++), May 1999.
[18] ETSI TS 102 027-3 Ver. 3.2.1: Methods for Testing and Specification (MTS), Conformance Test Specification for SIP (IETF RFC 3261), Part 3: Abstract Test Suite (ATS) and partial Protocol Implementation eXtra Information for Testing (PIXIT) proforma, 2005.
[19] GTB TTCN-3 Certificate, http://german-testing-board.info/de/ttcn3_certificate.shtm, 2008.
[20] IETF RFC 2616: Hypertext Transfer Protocol (HTTP/1.1), June 1999.
[21] IETF RFC 3261: Session Initiation Protocol (SIP), June 2002.
[22] ISO/IEC 9646-3: Information technology -- Open Systems Interconnection -- Conformance testing methodology and framework -- Part 3: The Tree and Tabular Combined Notation (TTCN), 1998.
[23] ISO/IEC 9075: Database Language SQL (Structure Query Language), 30 July 1992.
[24] ITU-T X.680: Data Networks and Open System Communications. OSI networking and system aspects – Abstract Syntax Notation One (ASN.1), July 2002.
[25] ITU-T Z.120: Languages and General Software Aspects for Telecommuncation Systems. Formal description techniques (FDT) – Message Sequence Chart (MSC), April 2004.
[26] OASIS: Universal Description, Discovery and Integration (UDDI) Spec Technical Committee Draft, http://uddi.org/pubs/uddi_v3.htm, 19 October 2004.

[27] OMG: Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, Jan. 2008.
[28] OMG: UML Testing Profile (UTP) Specification, Version 1.0. July 2005.
[29] W3C: Extensible Markup Language (XML) 1.0, W3C Recommendation, 6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006
[30] W3C: XML Schema Part 0,1,2: Primer, Structures, Datatypes, W3C Recommendations, 2 May 2001, http://www.w3.org/TR/2001/REC-xmlschema-{0,1,2}-20010502
[31] W3C: Simple Object Access Protocol (SOAP) 1.1: http://www.w3.org/TR/soap/, 27 April 2007.
[32] W3C: Web Services Description Language (WSDL) 1.1: http://www.w3.org/TR/wsdl, 15 March 2001.
[33] Testing Technologies: TTworkbench – a TTCN-3 tool set, www.testingtech.com
[34] Oracle: Java 4 SE Development Kit Version 4, JDK1.4, Documentation, http://www.oracle.com/technetwork/java/index.html, 2005.
[35] 3GPP TS 23.228: Technical Specification Group Services and System Aspects: IP Multimedia Subsystem (IMS), Stage 2, 2006.
[36] Z. R. Dai: An Approach to Model-Driven Testing - Functional and Real-Time Testing with UML 2.0, U2TP and TTCN-3, Promotion, TU Berlin, Fakultät Elektrotechnik und Informatik, June 2006.
[37] G. Din: A Workload Realization Methodology for Performance Testing of Telecommunication Services, PhD Thesis, TU Berlin, Fakultät Elektrotechnik und Informatik, Sept. 2008.
[38] M. Ebner, A. Yin, M. Li: Definition and Utilisation of OMG IDL to TTCN-3 Mapping. – 16th Intern. IFIP Conference on Testing Communicating Systems (TestCom 2002), Berlin, March 2002.
[39] J. Grabowski, D. Hogrefe, G. Rethy, I. Schieferdecker, A. Wiles, C. Willcock: An Introduction into the Testing and Test Control Notation (TTCN-3). Computer Networks Journal, Vol.42, Issue 3, 2003.
[40] M. Grochtmann, J. Wegener and K. Grimm: Test Case Design Using Classification Trees and the Classification-Tree Editor CTE. Proc. of 8th International Software Quality Week, SanFrancisco, California, USA, pp. 4-A-4/1-11, 1995.
[41] G.J. Myers: The Art of Software Testing. John Wiley & Sons, 2004.
[42] I. Schieferdecker, Z.R. Dai, J. Grabowski, A. Rennoch. The UML 2.0 Testing Profile and its Relation to TTCN-3. Proc. of the 15th IFIP Intern. Conf. on Testing of Communicating Systems (TestCom2003), LNCS 2644, Springer, May 2003, pp. 79-94.
[43] I. Schieferdecker, G. Din: A Metamodel for TTCN-3. 1st Intern. Workshop on Integrated Test Methodologies. Colocated with 24th Intern. Conference on Formal Description Techniques (FORTE 2004), Toledo, Spain, Sept. 2004.
[44] I. Schieferdecker, J. Grabowski: The Graphical Format of TTCN-3 and its Relation to UML and MSC. 3rd Intern. Workshop on SDL and MSC - Telecommunication and Beyond, SAM 2002, Aberystwyth, UK, June 2002.
[45] I. Schieferdecker, T. Vassiliou-Gioles: "Realizing distributed TTCN-3 test systems with TCI", IFIP 15th Intern. Conf. on Testing Communicating Systems -TestCom 2003-, Cannes, France, May 2003.
[46] I. Schieferdecker, S. Pietsch, T. Vassiliou-Gioles: Systematic Testing of Internet Protocols - First Experiences in Using TTCN-3 for SIP. 5th IFIP Africom Conference on Communication Systems, Cape Town, South Africa, May 2001.
[47] I. Schieferdecker, B. Stepien: Automated Testing of XML/SOAP-based Web Services. Proc. of the GI Fachtagung "Kommunikation in Verteilten Systemen", KIVS 2003, Leipzig, Germany, Feb. 2003.

[48] S. Schulz, T. Vassiliou-Gioles: “Implementation of TTCN-3 Test Systems using the TRI”, IFIP 14th Intern. Conf. on Testing Communicating Systems -TestCom 2002-, Berlin, Germany, March 2002.

[49] A. Vouffo-Feudjio: A Methodology for Pattern-Oriented Model-Driven Testing of Reactive Software Systems, PhD Thesis, TU Berlin, Fakultät Elektrotechnik und Informatik, Jan. 2011.

[50] P. Xiong, R. L. Probert, and B. Stepien. An efficient formal testing approach for Web service testing with TTCN-3.2005. http://www.site.oottawa.ca/ bernard/softcom paper.pdf.

[51] S.C.C. Blom, T. Deiß, N. Ioustinova, A. Kontio, J.C. van de Pol, A. Rennoch, and N. Sidorova: TTCN-3 for Distributed Testing Embedded Software. In: Perspectives of Systems Informatics, 27-30 Jun 2006, Novosibirsk, Russia, 2007.

[52] J. Grossmann, D. Serbanescu and I. Schieferdecker: Testing Embedded Real Time Systems with TTCN-3, 2nd International IEEE Conference on Software Testing, Verification, and Validation (ICST 2009), Denver, Colorado, USA, April 2009.

[53] I. Schieferdecker, E. Bringmann, and J. Grossmann: Continuous TTCN-3: testing of embedded control systems. In Proceedings of the 2006 international workshop on Software engineering for automotive systems (SEAS '06). ACM, New York, NY, USA, 29-36, 2006.

[54] H. Neukirchen, B. Zeiss, J. Grabowski, P. Baker, and D. Evans: Quality assurance for TTCN-3 test specifications. Software Testing, Verification and Reliability, 18: 71–97, 2008.

[55] B. Zeiß: Quality Assurance of Test Specifications for Reactive Systems. PhD Thesis, Universität Göttingen, June 2010.

[56] G. Din, D. Vega, and I. Schieferdecker: Automated Maintainability of TTCN-3 Test Suites based on Guideline Checking, 6th IFIP Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2008), Capri Island, Italy, Oct. 2008.

[57] D. Vega, G. Din, and I. Schieferdecker: TTCN-3 Test Data Analyser using Constraint Programming, 19th Intern. Conf. on Systems Engineering (ICSENG 2008), Las Vegas, USA, Aug. 2008

[58] B. Zeiss and J. Grabowski: Analyzing Response Inconsistencies in Test Suites. In Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop. Springer-Verlag, Berlin, Heidelberg, 2009.