Systematic Testing of Internet Protocols - First Experiences in Using TTCN-3 for SIP -

Ina Schieferdecker, Stephan Pietsch, Theofanis Vassiliou-Gioles

GMD FOKUS, Berlin, Germany
URL http://www.fokus.gmd.de/tip

Abstract

Recently, the European Telecommunications Standards Institute (ETSI) approved the third edition of the Tree and Tabular Combined Notation (TTCN-3) as a requirement to modernized and widen its application beyond pure OSI conformance testing. TTCN-3 is a text-based language for the specification of tests for reactive systems. In addition to several new concepts for defining test behavior, it supports besides textual test specifications a graphical forma as a natural and user-friendly notation for developing, documenting and visualizing test suites. This paper introduces the main concepts of TTCN-3, presents its graphical format based on Message Sequence Charts (MSC) and shows the application of TTCN-3 for testing text-based protocols – a new area of application for systematic testing.

1 Motivation

Internet protocols are getting more and more complex. They are described textually only. This encompasses the risk of misinterpretation and wrong implementation. Therefore, testing a final implementation within its target environment is essential to assure the correctness and interoperability of an implementation. The need for testing approaches arose already within the IETF community: so-called bake-off events are used to launch new technologies, to validate the specifications and to check various implementations for their functionality and performance. However, the tests used at bake-offs are not uniquely defined, so that one has to question on what basis implementations are evaluated.

In contrast to that, there are test-engineering methods and a conformance testing methodology [6] within telecommunication, which has evolved over years, is widely spread and successfully applied to assess the correctness of protocol implementations. The standardized test specification language TTCN-3 [1] with its advanced features for test specification is expected to be applied to many testing applications that were not previously open to TTCN. TTCN-3 has been defined to be applicable for protocol testing, (including mobile and Internet protocols), service testing (including supplementary services), module testing, testing of CORBA based platforms, API testing etc. TTCN-3 is not restricted to conformance testing and can be used for many other kinds of testing including interoperability, robustness, regression, system and integration testing. This paper reports first experiences on the use of TTCN-3 for the specification of (conformance) tests for Internet-related protocols, in particular for the Session Initiation Protocol (SIP [3]). SIP is a control protocol to establish, modify and terminate multimedia conferences (also called sessions) or Internet telephony calls. SIP is used to invite participants to unicast and multicast sessions. Media and participants can be added to an existing session without disruption.

The study has been undertaken on the basis of the protocol specification [3] and message flow definitions [4] explaining the use of SIP entities. Both tests provided for bake-off events and test purpose definitions according to the conformance test methodology are considered. In fact, SIP bake-off tests are from their philosophy near to conformance tests: message formats and message sequences are tested. In addition, robustness tests are defined to check for unforeseen use of protocol entities (also called torture tests).

The first question before investigating testing of Internet protocols with TTCN-3 has been, whether this is just an exercise in validating the applicability and practicability of the new test specification language TTCN-3. Or, whether there are also specifics to Internet protocols, which require new concepts in testing. Previous works [9][10] on systematic testing for Internet protocols argue that this is not the case as testing of Internet protocols can be based on the well-known black-box approach of issuing stimuli at system interfaces to the system under test, expecting responses, and comparing the results with the expected ones. However, two aspects deserve further consideration:

(1) Internet protocols use various components, which can be distributed in a network. For example, SIP entities are user agents for caller and callees, proxy and redirect server, and a registrar.

(2) Internet protocols may operate in dynamic configurations. In the case of SIP, in course of a session further participants can be invited as well as participants may leave a session without disruption to the session.

(3) Internet protocols use various kinds of interfaces such as bitstring, octetstring, and text-based interfaces.

Although those points are not a principal problem to testing as (1) and (2) can be tackled with using components also in the test system with predefined static configurations (by loosing flexibility in adapting and scaling tests to different and/or larger configurations) and (3) can be addressed with specific coder/decoder for the various interfaces, they become questions to the engineering methods for test development. Is it e.g. possible to specify test cases without having a predefined number of test components, i.e. to allow dynamic configurations? How to use data definitions of the test system directly within the test definitions? How to adapt the executable tests to the various kinds of interfaces? While component-based and dynamic test configurations in TTCN-3 are new language features, the data part is not open. TTCN-3 uses an own data type system, which is near to ASN.1 and enables therefore the direct use of ASN.1 modules within test suite specifications. External data can be used only via a mapping from external data to the TTCN-3 data type system.

The paper describes the use of TTCN-3 for testing SIP implementations. It is structured as follows: The next section gives an overview on TTCN-3 by discussing the basic concepts and giving some insights into the core notation and the graphical format of TTCN-3. This is explained by giving examples from a SIP test case. Afterwards, the graphical format is further elaborated by discussing the various possibilities of its use. In the summary section, the test requirements for SIP are discussed and compared with the abilities of TTCN-3 to express such tests.

2 Overview on TTCN-3

Testing a system is performed in order to assess its quality and to find errors. An error is considered to be a discrepancy between observed or measured values provided by the system under test and the specified or theoretically correct values. Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements. It approves a quality level of a tested system

Conformance testing in particular is the process of testing the extent to which implementations of OSI protocol entities adhere to the requirements stated in the relevant standard or specification. Conformance testing is functional black box testing. The term functional refers to the correct functional behavior of an Implementation Under Test (IUT), i.e. the correct input/output behavior in each state. Black box testing means that the internal structure of the IUT remains hidden, i.e. it is a black box for the test developer. It is an active test method as opposed to the passive, monitoring only based test methods: stimuli are issued to the system under test and responses are observed.

The OSI conformance testing procedure is defined in the international ISO/IEC standard 9646 Conformance Testing Methodology and Framework (CTMF) [5]. Test cases developed according to the principles of CTMF are abstract. Executable test cases are derived from abstract ones by compilation and adaptation to the Means of Testing (MoT). The MoT is the combination of equipment and procedures that can perform the derivation, selection, parameterization and execution of test cases. It consists typically of dedicated test devices and facilities for the coordination of test devices and the observation of the IUT.

The third edition of TTCN (TTCN-3) is a textual test specification language and has been developed to address the needs of testing modern technologies in the telecom and datacom domain. It looks somehow like a common programming language, e.g., C or C++ or Java. Recently the European Telecommunication Standards Institute (ETSI) funded a team to evolve TTCN into a language whose look and feel is of a modern programming language with test specific extensions, called the TTCN third edition (TTCN-3) [5]. These extensions consist of: test verdicts, matching mechanisms to compare the reactions of the IUT with the expected range of values, timer handling, distributed test processes, ability to specify encoding information, synchronous and asynchronous communication, and monitoring. In providing these extensions it is expected that test specifiers and engineers will find this general-purpose language, more flexible and user-friendly and easier to use than its predecessor. As illustrated in Figure 5 TTCN-3 also encompasses different presentation formats: tabular format [6] and a Message Sequence Chart (MSC) like format [7].

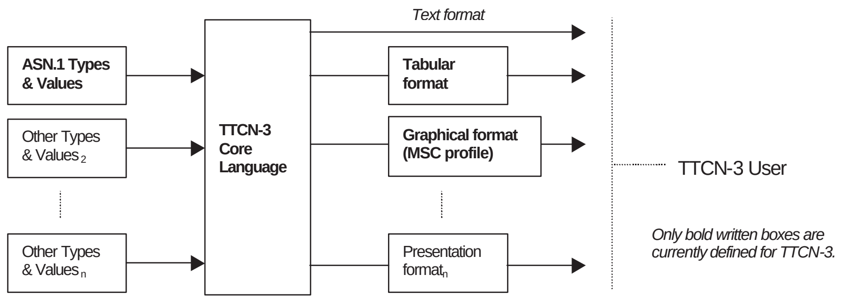


Figure 1. Users view of TTCN-3 and the various presentation formats

TTCN-3 is intended for various application areas like protocol testing (e.g. mobile and internet protocols), supplementary service testing, module testing, testing of CORBA based platforms, testing of API's etc.

With TTCN-3 the existing concepts for test specifications have been consolidated. TTCN-3 contains basic test concepts, but is not a "hardcore" language as the syntax has been defined such that it is readable like common programming languages. Besides consolidation, TTCN-3 defines new concepts to widen the scope of applicability of TTCN-3.

Retaining and improving basic concepts of predecessors of TTCN-and adding new concepts increase the expressive power and applicability of TTCN-3. New concepts are, e.g., a test execution control to describe relations between test cases such as sequences, repetitions and dependencies on test outcomes, dynamic concurrent test configurations and test behavior in asynchronous and synchronous communication environments. Improved concepts are, e.g., the integration of ASN.1 [12], the module and grouping concepts to improve the test suite structure, and the test component concepts to describe concurrent and dynamic test setups.

In the subsequent sections, TTCN-3 will be explained by a shortened version of the test suite for the sake of simplicity and ease of understanding.

2.1 The Core Language

The top-level unit of a TTCN-3 test suite is the module, which can import definitions from other modules. A module consists of a declarations part and a control part.

```
module SIP {
  import SIPdeclarations;
  control {
        execute(SIP_UA_REC_V_028());
  } // end control
} // end module SIP
```

The declarations part of a module covers definitions, e.g., for test components, their communication interfaces (so called ports), type definitions, test data templates, functions, and test cases. The module for the `SIP` test suite imports all declarations from another module called `SIPdeclarations`.

The control part of a module calls the test cases and describes the test campaign. For this, control statements similar to statements in other programming languages (e.g., if-then-else and while loops) are supported. They can be used to specify the selection and execution order of individual test cases. The imported module contains also the definition for the test case `SIP_UA_REC_V_028`, which is performed in the control part with the `execute` statement.

Test suite parameters for information on the protocol conformance statements and the extra information for testing are denoted as external constants and can be accessed within the module from any scope. Groups can be used to structure the declarations of a module to ease the readability.

```
group TestSuiteParameters {
  /** the callee user */
  external const charstring CALLEE_USER;

  /** the caller user */
  external const charstring CALLER_USER;
…
    } // end group TestSuiteParameters
```

As in other programming languages, constants are used for single declarations of static data. In the case of SIP, constants are used to denote predefined strings of the text-based messages.

```
group ConstDefinitions {
  /** SIP version */
  const charstring SIP_VERSION := "SIP/2.0";
  /** @ (at) sign */
  const charstring AT := "@";
  …
} // end group ConstDefinitions
```

Types of protocol messages are defined as structured types, which can be constructed as record, set, enumerated types, etc. from basic types or other structured types. TTCN-3 supports a number of predefined basic types. They include programming-typical basic types such as integer, boolean and string types, as well as testing-specific types such as verdict type, port and component type. The latter two are used to define the configuration of a test system. An example for a SIP message type with optional information elements is given below.

```
group TypeDefinitions {
  // Request (4) */
  type record Request {
      RequestLine           requestLine,
      ReqMessageHeader      reqMessageHeader optional,
      charstring            crlf,
      MessageBody           messageBody optional
  }
  …
} // end group TypeDefinitions
```

Test data to be sent or received over ports is defined as templates. Templates support matching mechanism for the denotation of a variety expected data, which will be received. In addition, parameterization of templates is used for flexibility in using templates in a current testing context. The template `Invite_s_1` given below is of type `Request`. It is a sending template and defines completely the data to be sent. The `messageBody` element is omitted.

```
group TemplateDeclarations {
  /** INVITE (4.2.1) */
  template Request Invite_s_1 := {
      requestLine           := Request_Line_s_1,
      reqMessageHeader      := Req_Message_Header_s_1,
      crlf                  := CRLF,
      messageBody           := omit
  }
  …
} // end group TemplateDeclarations
```

Test cases describe the probes during the test campaign, i.e., they specify the test behavior. Test cases are executed by a main test component, which is automatically created when a test case is started. Further parallel test components can be created and terminated during the execution of a test case. The test components (both the main test component and the parallel test components) are connected via well-defined communication interfaces, so-called port, to the system under test or to other test components. The set of interfaces of the system under test is the border of the test system. By default, it is assumed to be identical to the interfaces of the main test component. If not, it has to be explicitly defined.

A test configuration is a set of test components. The set of interfaces (i.e. ports) belonging to a component is defined in the respective component type. In addition, a component type may define local variables and timers, which are instantiated whenever a component of that type is created.

```
group TestConfigurationDefinition {
  /** communication port type definitions */
  type port SipPortType message {
      inout Request, Response;
  }
  /** component type definitions */
  type component SipTestComponent {
      timer T1 := 0.5; // T1 = 500 ms (10.2.1)
      timer T2 := 4.0; // T2 = 4s     (10.2.1)
      port SipPortType SIP_PCO
  }
} // end group ConfigurationDefinition
```

The `SipPortType` is a message based port (as opposed to a signature-based port, at which operation invocations, replies and exceptions are exchanged). It is bi-directional and supports in both directions `Request` and `Response` messages. The component type `SipTestComponent` defines two local timers `T1` and `T2` (TTCN-3 assumes the unit of timer values to be seconds) and one port of type `SipPortType`.

One can express a variety of test relevant behavior within a test case such as the alternative reception of communication events, their interleaving and default behavior to cover, e.g., unexpected reactions from the tested systems. In addition to the automatic test verdict assignment, more powerful logging mechanisms, e.g., for a detailed tracing, are provided.

The test case `SIP_UA_REC_V_028` shown below has a very simple behavior. After activating a default behavior for handling unexpected or irrelevant responses during test execution, the `Invite_s_1` is sent, the timer `T1` is started to avoid infinite waiting for the expected response `Response_r_1`. If this is received successfully (i.e. the response of the system under test has matched the template), the timer is cancelled, a `pass` verdict is assigned and the test terminates.

```
group TestCases {
  group UA {
  …
        /** Ensure that the IUT, in the initial state, on receipt of an
         * INVITE sip massage, sends a 4xx response. */
        testcase SIP_UA_REC_V_028() runs on SipTestComponent {
                  activate (Default_1); // Default activation
                  // the CSeg number in the request header is an error
                  SIP_PCO.send(Invite_s_1);
                  T1.start;
                  // the CSeq value has been not understood
                  SIP_PCO.receive(Response_r_1);
                  T1.cancel;
                  verdict.set(pass);
                  stop;
        } // end testcase SIP_UA_REC_V_028
        …
  } // end group UA
} // end group TestCases
```

In TTCN-3 defaults are used to handle communication events which are unexpected or irrelevant, i.e. which do not contribute to the test objective. Defaults in TTCN-3 are handled as macro expansion: when activated, they are expands automatically as last alternatives to receiving events. Several defaults can be activated and/or deactivated.

```
group Defaults {
  named alt Default_1
  {
        [] any timer.timeout {
             verdict.set(fail);
             stop;
           }
        [] any port.receive {
             verdict.set(fail);
             stop;
           }
      }
    } // end group Defaults
```

The default `Default_1` handles all unforeseen timeout events and messages receive events.

2.2 The Graphical Format

Though TTCN is capable to describe complex distributed test behavior, practice has shown that it is difficult to keep an overview about the complete test description when reading the textual form alone without any visualization. The visualization by means of the tabular format of TTCN-2, however, has turned out to be not very intuitive for behavior descriptions even if tools are used. Within the TTCN-2 tree and tabular notation, statements are written on successive lines with either successively incremented indentations to indicate subsequent statements or with equal indentations to indicate alternatives. In case of highly nested alternatives, such a notation is not very user friendly.

Graphical specification techniques are getting more acceptance and wide spread use as they are easier to develop, understand, and maintain. Prominent examples are MSC [7], SDL and UML. With the work on graphical test specifications the attempt was taken to make use of graphics also for the definition and description of test behavior. Not every detail of a test case can be graphically represented, as it is not feasible. For example, for data declarations and their use within the test case it is more adequate to use normal text. However, aspects like the behavior, structure, and configuration of a test case should be graphically represented.

The core behavior of a test case is the interaction between test system and system under test and, on a lower level of detail, between the test components. MSC instances can be used to represent the constituents and MSC messages to represent their interaction. In addition, it would be advantageous to support a graphical differentiation between test components and the communication links to the system under test and between the test components. This would visualize better the exchange of messages to the system under test and within the test system.

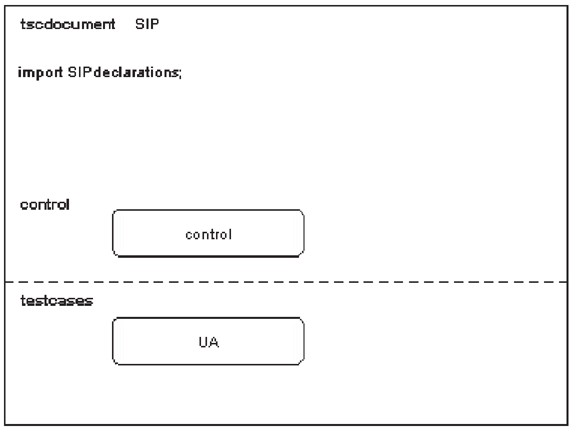
In order to support the structuring of the test behavior instead of supporting plain test behavior specifications only (like it is done in TTCN-3 with functions and function calls), graphical means to structure also graphical test case specifications are needed. MSC structuring concepts like MSC references and High-Level MSC with slight extensions are well suited for that purpose.

A critical point is the graphical representation of test configurations consisting of dynamically created and terminated components and the connections to the system under test and to other test components. Test configurations allow not only point-to-point but also point-to-multipoint connections. In addition, they are dynamic. This would require the graphical representation classes of test components (and not the individual test components), which destroys the ability to represent connections, which are always between concrete test components. In MSC, there is no such kind of diagrams or graphical symbols. In addition, the MSC gate concept of MSC turned out to be not adequate as it is based on the concept of gating singular events to the environment but does not enables the accumulation of gate events in terms of channels for example. Due to the problems in representing dynamically created test components and their communication links (this is problematic even with static configurations), this aspect is not yet addressed graphically within GFT. The textual TTCN-3 operations in order to set-up configurations are used within action boxes instead.

As a consequence, the visualization of TTCN by means of a graphical Message Sequence Chart (MSC) format has been investigated. A main advantage of the MSC language is its clear graphical layout, which immediately gives an intuitive understanding of the described behavior. Within the area of conformance testing, MSC is already well established for the specification of test purposes and as such for the automatic generation of TTCN test cases. Though MSC has been used for test specifications in the past only in a fairly restricted manner, the powerful language constructs for MSC composition, object oriented modeling as well as data descriptions, contained in the present version of the MSC language, even make a comprehensive MSC specification of test cases feasible. It should be pointed out, however, that the graphical format for TTCN is not intended as a standalone language but as a basis for the generation of TTCN-3 descriptions. Therefore, hybrid representations where only the main parts of the test behavior are visualized by means of MSCs while the remaining parts are provided in form of TTCN descriptions may turn out to be most effective.

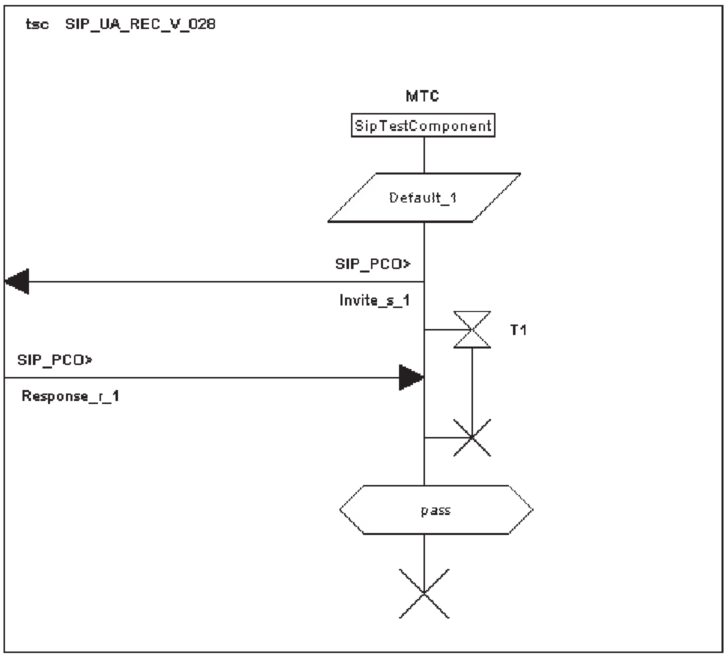
The graphical format of TTCN-3 uses a subset of Message Sequence Charts 2000 (MSC-2000) with test specific extensions and extensions of general nature. This will be in the following referred to as test specific MSC profile or as Graphical Format for TTCN-3 (GFT[1]). The majority of extensions are textual extensions only. Graphical extensions are defined to ease the readability of GFT documents, which are based on MSC documents. All graphical extensions have either substitutes within standard MSC-2000 or are optional (i.e. need not to be used). This allows using established MSC tools (with some additions to the textual syntax) for the graphical definition of TTCN-3 test cases. The graphical format provides an alternative way of displaying the core language of TTCN-3.

The GFT document defines a test suite and the associated collection of message sequence charts, which again define traces of test events.



The GFT document contains a collection of test cases. Test cases are defined as scenarios of interaction between the instances contained in the GFT document. They are defined within the test cases part of a GFT document. The control part contains an MSC with identifier control, which may refer to other control MSCs or to MSCs that again refer to MSCs representing test cases. The functions parts contain test behavior patterns, which are reused by the test case part. The heading of the GFT document contains the test suite identifier, test suite parameters and further declarations such as component and port instances, messages, timers, types and templates etc. Subsequently in separate parts, GFT references to control, test case, and function definitions in GFT are given. Dashed lines separate those parts. They start with one of the keywords control, testcase, and functions, resp. Aanyof the parts might be empty. Only one of each part can be defined in a GFT document.

The control part is represented by GFT Test Sequence Charts that describes the execution order (and possibly repetitions) of actual test cases.



Test cases and functions are represented by MSCs. A MSC represents the flow of test events between test component instances, port instances and/or the environment. The test case identifier is used as the identifier for the MSC. The component type identifier given after the runs on keyword is represented as the type of the MTC test component instance inside the instance head symbol. The type declaration for the test system interface is put into the MSC header with the system keyword.

The test component instance has an instance head symbol, on top of which the identifier of the test component and inside of which optionally the type of the test component is given. The head symbol is followed by the instance line (represented as a solid line). The events on this instance line are totally ordered

[1] When writing this paper it has been decided to have the acronym GFT for the Graphical Format of TTCN. Before it was called Test Sequence Charts to denote its closeness to MSC and to indicate that it is in certain points different to MSC. As the tooling could not be modified fast enough, some of the figures still contain tsc as or within keywords. This has to be changed in the near future to msc as keyword or part of a keyword.

from top to bottom. The end of a behavior description for a test component is indicated by the instance stop symbol.

Ports facilitate communication between test components, and between test components and the test system interface. A port instance can support message-based events, procedure-based events or a mixed of both. Test components then use these ports to communicate with other test components or the test system interface. They can be represented explicitly or implicitly. In the first case, graphical symbols comparable to the test component instance symbols are used, but with dashed lines. In the latter case a prefix to messages and calls is used indicating the port name.

3 Different Use of the Graphical Format of TTCN-3

GFT can be used in different forms to enable focusing on different aspects of TTCN-3 test cases.

3.1 An Example

An example taken directly from the SIP specification [3] is used to express the various forms of GFT specifications. By using a number of simple examples for this test case, each illustrating a different form explains those forms.

In testing the redirect functionality (i.e. interaction (1) to (4)), the test uses two test components for the caller and the callee. Coordination messages between the test components are used to coordinate the redirection. The system under test has two ports to be connected to caller and callee.

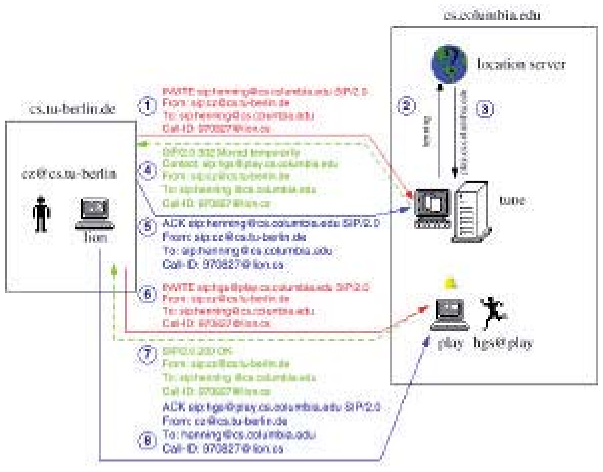

cs.columbia.edu
location server
cs.tu-berlin.de
cz@cs.tu-berlin
lion
tune
play
hgs@play
1
2
3
4
5
6
7
8
ACK sip:henning@cs.columbia.edu SIP/2.0
From: sip:cz@cs.tu-berlin.de
To: sip:henning@cs.columbia.edu
Call-ID: 970827@lion.cs
ACK sip:hgs@play.cs.columbia.edu SIP/2.0
From: cz@cs.tu-berlin.de
To: henning@cs.columbia.edu
Call-ID: 970827@lion.cs

Figure 1: Small example taken from [3]

The parts of the textual test case that are essential to discuss the various GFT forms, are given below:

```
type component UA_type {
  timer T1 := 0.5; timer T2 := 4.0;
  port SipPortType SIP_PCO; port CoordPortType COORD
}

type component TSI_Type {
//test system interface has a SIP and a location port
  port SipPortType CALLER_PCO; port SipPortType CALLEE_PCO
}

testcase example() runs on UA_type system TSI_Type {
  var UA_type Caller:= UA_type.create;
  map(mtc:SIP_PCO,system:CALLER_PCO);
  map(Caller:SIP_PCO,system:CALLEE_PCO);
  connect(mtc:COORD,Caller:COORD);
  verdict.set(pass);
  activate (Default_1);
  Caller.start(caller_behaviour);
  SIP_PCO.send(Redirect_s_1);  //send redirect henning@cs.columbia.edu
```

```
    SIP_PCO.receive(OK_Redirect_r_1); //OK for redirect
    COORD.send(redirected_1);
    Caller.done;
  } // end of testcase example

  function caller_behaviour() runs on UA_type {
    verdict.set(pass);
    activate (Default_1);
    COORD.receive(redirected_1); //coordination with callee
    SIP_PCO.send(Invite_s_2);   //send invite henning@cs.columbia.edu
    alt {
          [] SIP_PCO.receive(Redirection_r_1);  //receive moved temporarily
          [] SIP_PCO.receive(Response_r_2)  //receive response incorrectly
                { verdict.set(fail); }
    }
  } // end function caller_behaviour
```

3.2 Form Aspect: Vertical Split vs. No Vertical Split

In a vertical split form, there are MSCs per test component, which represent the behavior of the test components. For the example, there is a MSC `vertical_split_example` for the main test component (indicated by the keyword **mtc**), a MSC `Caller` for test component `Caller`. Per MSC, there is a MSC instance representing the test component. The MSC instance head contains the identifier of the test component (i.e. for the main test component the keyword **mtc**) and its type. In MSC `Caller`, there is an unnamed instance of type `UA_type`, which acquires the test component identifier from the create message of the mtc. A test component is started with the special start message (indicated by the dashed arrow symbol). Afterwards, the behavior of the test component is represented. Finally, the test component terminates.

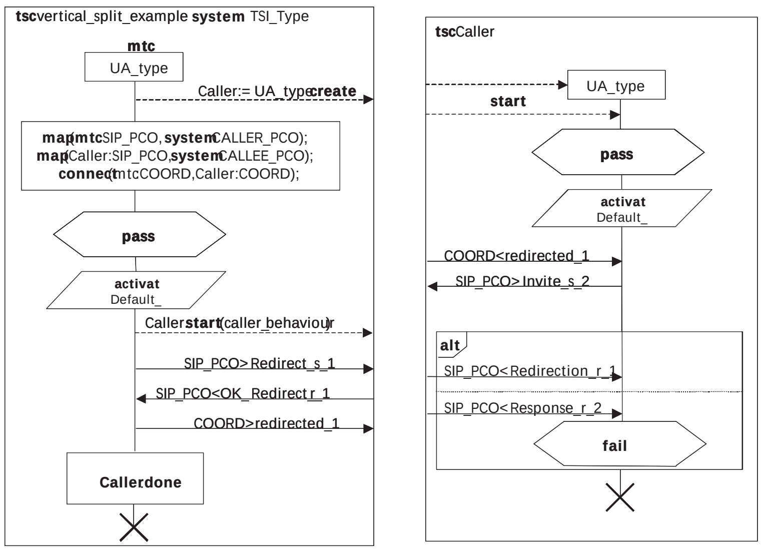


Figure 2: Vertical split

Without a vertical split, one MSC contains all test components constituting the test case behavior. In this view, the communication between **mtc** and `Caller` with message `redirect_1` transferred via port `COORD` becomes apparent. In addition, the order between test events at different test components is visible. The identifier for the behavior function of a test component instance is contained in its start operation (i.e. in this case `caller_behaviour`).

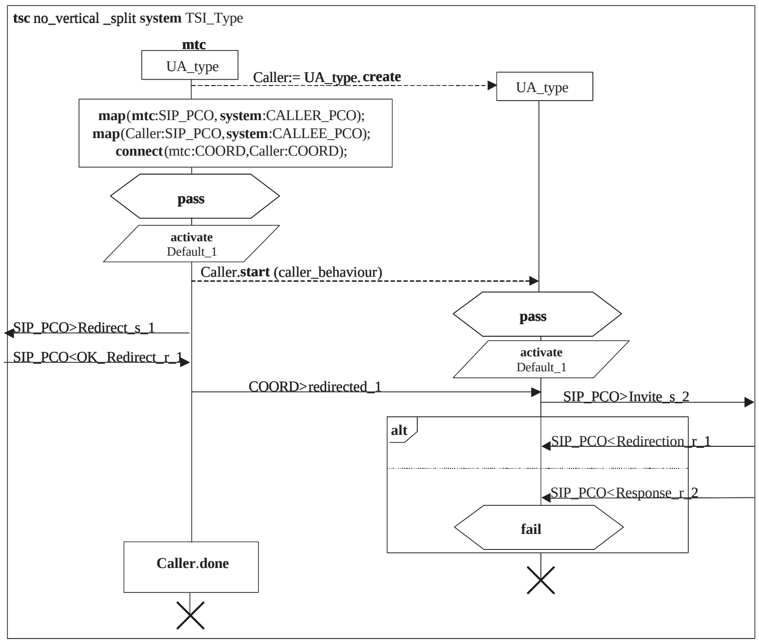


Figure 3: No vertical split

3.3 Form Aspect: Horizontal Split vs. No Horizontal Split

In the horizontal split form, separate MSCs per function performed by a test component are defined. In this example, there is an additional MSC `caller_behaviour` to the MSC representing the `Caller` for the definition its test behavior.

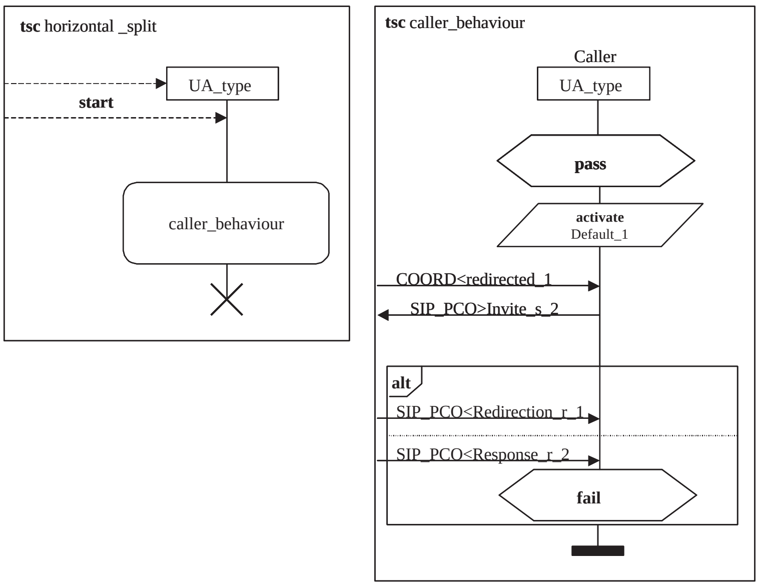


Figure 4: Horizontal split

Without horizontal split, the test behavior of functions is represented inline at the MSC instance, which represents the test component that performs a given function. Please see also right hand side of Figure 2, which shows the same definition as above but without a horizontal split.

3.4 Form Aspect: Explicit vs. Implicit Port Representation

Communication ports can be represented in GFT differently. Either, a port is represented by a special port instance (indicated by dashed instance head, line and end symbols) or it is represented as a prefix to the message.

Figure 5 contains an explicit port representation for the ports `COORD` of type `CoordPortType` and `SIP_PCO` of type `SipPortType`. The instance head of port instances contains the port type. Messages, which are sent or received at a port, are represented by incoming or, respectively, outgoing messages of the port instance. For example, message `redirect_1` that is received by `Caller` from port `COORD` is represented by a message from instance `COORD` to instance `Caller`. Explicit port representations allow following the sequence of interaction at individual ports.

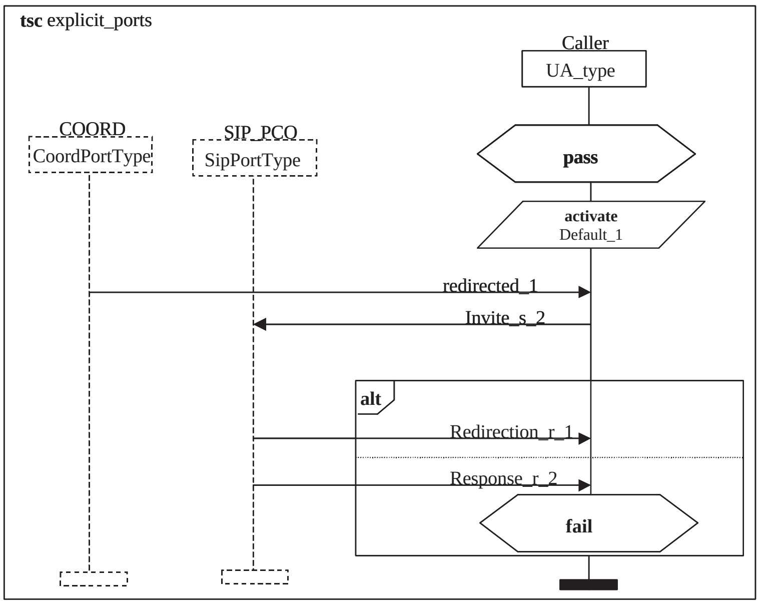


Figure 5: Explicit ports

Figure 1 to 4 the ports have been represented implicitly. The prefix in front of a message indicates the port to which a message is sent or from which a message is received. The prefix is separated from the message with a '>' symbol. For example, `SIP_PCO>Invite_s_2` indicates that message `Invite_s_2` is sent to port `SIP_PCO` Optionally, the port type can be given in addition such as it is done with `SIP_PCO: SipPortType>Invite_s_2`. MSCs with implicit port representations put the emphasis on the sequence of interaction of test components and make less apparent the interactions at the individual ports.

A combination of different port representation can also be used. Often, ports connected to other test components are shown implicitly, while ports connected to the system under test are given explicitely. In Figure 7, port `SIP_PCO` is represented explicitly and port `COORD` is given implicitly.

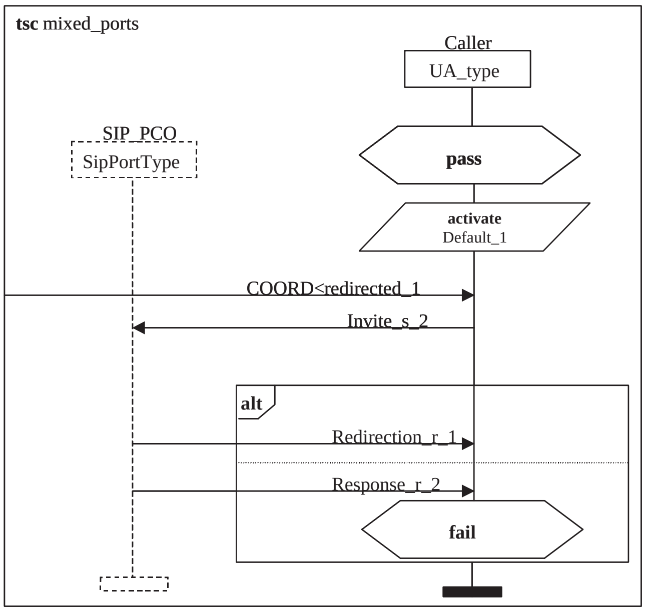
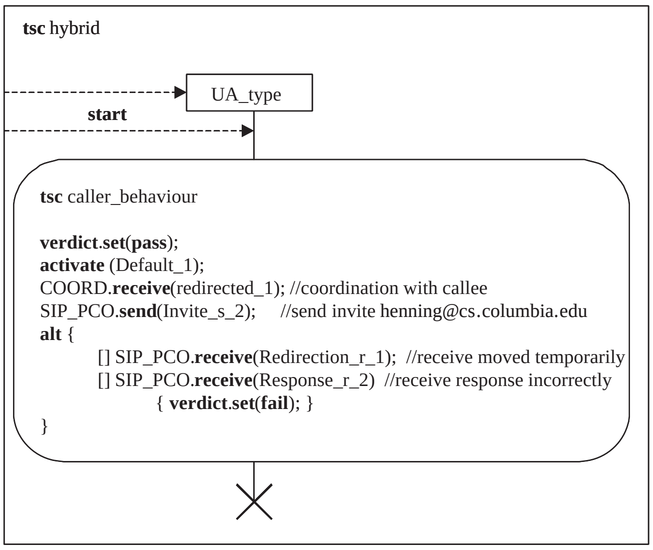


Figure 6: Combination of different port representation

3.5 Form Aspect: Hybrid vs. Not Hybrid

In hybrid MSC, TTCN-3 code can be used directly inside of MSC references. In Figure 8, the test behavior is given textually in terms of TTCN operations and not graphically. Hybrid MSCs are of particular use where is no need to define every detail graphically.



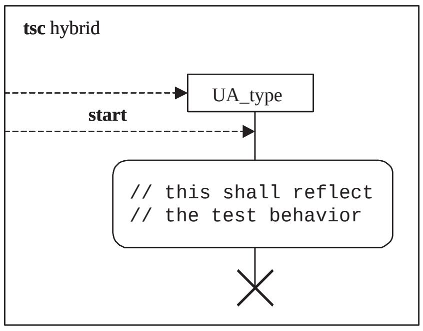
```
verdict.set(pass);
activate (Default_1);
COORD.receive(redirected_1); //coordination with callee
SIP_PCO.send(Invite_s_2);     //send invite henning@cs.columbia.edu
alt {
        [] SIP_PCO.receive(Redirection_r_1);  //receive moved temporarily
        [] SIP_PCO.receive(Response_r_2)  //receive response incorrectly
                { verdict.set(fail); }
}
```

Figure 7: Hybrid MSC

The figures given in the previous sections have not been hybrid.

3.6 Form Aspect: Partial vs. Complete

MSC specifications may be partial only. A MSC may contain inside a MSC reference a comment only (as it is done in Figure 9). The support of partial MSC specifications is of particular importance for the step-wise development of test suites within MSC.



```
// this shall reflect
// the test behavior
```

Figure 8: Partial MSC

Complete MSC specifications contain MSC identifiers or TTCN-3 code within MSC references. The figures given in the previous sections are complete ones.

3.7 HyperMSC

Practice has shown that apart from simple cases, the representation of test cases by means of standard inline expressions, MSC references and HMSCs leads to diagrams which are not easy to read and handle. In order to overcome this deficiency, in GFT an expanded form of MSC references is admitted introducing the concept of HyperMSCs. The MSC referencing mechanism is interpreted in a hypertext-like manner. The main concept is to enable the expansion of MSCs within references to an MSC. This way, a convincingly transparent representation is obtained even in case of many MSC references and alternative and loop inline expressions.

A MSC specification can make use of hyper facilities by interpreting MSC references as hyperlinks to the defining MSC and by supporting different views on a MSC in expanded (showing the MSC behavior) and collapsed form (showing the reference only). This makes the test specification more readable and more transparent.

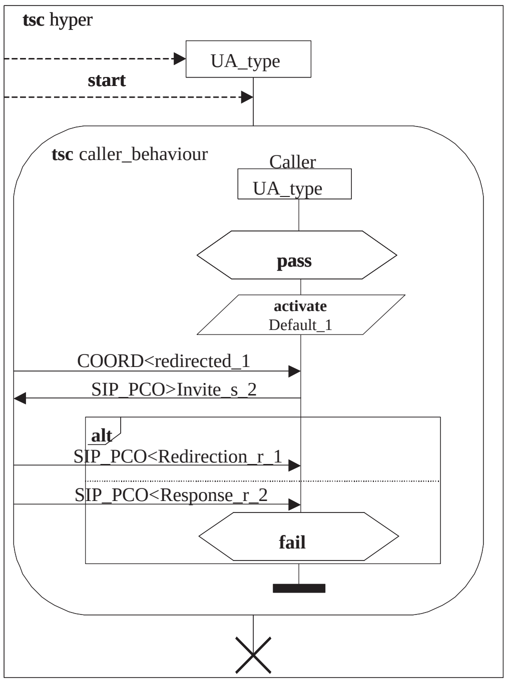


Figure 9: Hyper MSC

3.8 Summary of MSC forms

The different form aspects of MSC result in several different forms for representing test suites in MSC. The first three aspects, i.e. vertical vs. no vertical split, horizontal vs. no horizontal split, and explicit vs. implicit port representation, relate to the graphical setting of test specifications in MSC. The hybrid vs. not hybrid aspect relates to the combined use of MSC and TTCN-3. The partial vs. complete form aspect relates to the level of completeness of a MSC specification. Practice will show which forms are of primary use for the test developers and/or for test documentation. Appropriate tool support is a precondition to enable the representation of test cases in different forms in order to highlight various aspects of a test case.

4 Summary: SIP test requirements and comparison with TTCN-3

The use of TTCN-3 for SIP test suite development has been successful. With the advantages of TTCN-3 (being programming-like language, supporting graphical test specification, enabling load tests from functional tests), we think that it will be accepted within the IETF community and could become a starting point for thorough systematic tests of Internet protocol and services.
The SIP features are comparable to the features of signaling protocols in the telecommunication area like in ISDN and B-ISDN. As such, it does not impose specifically hard test requirements for functional tests on valid, invalid and inopportune behavior. The test configurations for functional tests are rather simple by having test components on the caller and callee side.
One specific is the text-based definition of messages in SIP. An EBNF grammar is given in [3] to define the message formats. The encoding of the messages is simple. For their decoding, we make use of a lexer and parser approach that are developed from the EBNF definition.
Another aspect though became apparent: TTCN-3 uses a set of hard-coded values (in form of templates) for the definition of test data for interacting with the system under test. However, for a set of test cases the pure test behavior is identical while only parts of the test data are modified (e.g. to cover extreme and typical value). As

TTCN-3 lacks a mechanism to define templates for sending messages (i.e. in TTCN-3 sending templates have to be defined concretely for each individual bit), several test cases have to be defined for such cases. We have already started work to have extensions to TTCN-3 on flexible value definitions for sending templates and to enable a mechanism how to make use of varying data within repeated executions of test cases.
In the case of performance tests for SIP, the TTCN-3 feature for dynamic configurations becomes of use. The same test behavior is performed by several test components, which are created dynamically and impose an increasing load on the SIP server. However, the inability to define real-time and performance constraints (except of timer issues) in TTCN-3 requires to support performance testing concepts via external functions. How this will work out is a subject of future research.

References

[1] ETSI DES-00063-1 TTCN-3: Core Language.
[2] ETSI DES-00063-3 TTCN-3: Graphical Presentation Format.
[3] IETF - Internet Engineering Task Force, Network Working Group: RFC2543bis-02: Session Initiation Protocol (SIP). – Nov. 2000.
[4] IETF - Internet Engineering Task Force, Network Working Group: draft-ietf-sip-call-flows-01.txt: SIP Telephony Call Flow Examples, July 2000.
[5] ISO/IEC 9646-3 (1998): “Information technology - Open systems interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)”
[6] ITU-T Recommendation X.290 (1995): “OSI Conformance Testing Methodology and Framework – General Framework”
[7] ITU-T Recommendation Z.120 (2000): “Message Sequence Charts (MSC)”.
[8] M. Born, A. Hoffmann, I. Schieferdecker, T. Vassiliou-Gioles, and M. Winkler: Performance Testing of a TINA Platform. - TINA99, TINA Conference, Hawaii, April 1999.
[9] T. Csöndes, S. Dibusz, P. Kremer: Experiments on IPv6 Testing13th IFIP International Conference on Testing Communicating Systems’ (Testcom 2000), Ottawa (Canada), August 2000.
[10] R. Gesce, P. Kremer: Automated Test of TCP Congestion Control Algorithms. - IFIP 12th International Workshop on Testing of Communicating Systems (IWTCS'99), Budapest (Hungary), Sept. 1999.
[11] J. Grabowski, A. Wiles, C. Willcock, D. Hogrefe: On the Design of the new Testing Language TTCN-3. - 13th IFIP International Conferenc on Testing Communicating Systems’ (Testcom 2000), Ottawa (Canada), August 2000.
[12] M. Li, I. Schieferdecker, A. Rennoch: Testing the TINA Retailer Reference Point.- ISADS’99, Fourth International Symposium on Autonomous Decentralized Systems, Tokyo (Japan), 1999.
[13] E. Rudolph, I. Schieferdecker, J. Grabowski: HyperMSC - a Graphical Representation of TTCN. Proceedings of the 2nd Workshop of the SDL Forum, Society on SDL and MSC (SAM’2000), Grenoble (France), June, 26 - 28, 2000.
[14] E. Rudolph, I. Schieferdecker, J. Grabowski: Development of an MSC/UML Test Format. FBT’2000 - Formale Beschreibungstechniken für verteilte Systeme (Editors: J. Grabowski, S. Heymer), Shaker Verlag, Aachen, June 2000.
[15] I. Schieferdecker, J. Grabowski: Conformance Testing with TTCN.- Special Issue on Languages for telecommunication applications, Telektronikk, Vol. 4 - 2000.
[16] I. Schieferdecker, M. Li: Functional Tests for Component Based Distributed Systems. - First Intern. Conf. on Software Testing (1st ICSTEST 2000), Bonn (Germany), April 5-7, 2000.
[17] I. Schieferdecker; B. Stepien, B.; A. Rennoch: “PerfTTCN, a TTCN Language Extension for Performance Testing”, in: Kim, M.; Kang, S.; Hong, K. (eds.), Testing of Communication Systems, Vol. 10, , 10th International IFIP TC6/WG6.1 Workshop on Testing of Communication Systems, Cheju Island (Korea), Sept. 1997, London (UK): Chapman & Hall, 1997.
[18] T. Vassiliou-Gioles, M. Li, I. Schieferdecker, M. Born, M. Winkler: Configuration and Execution Support for Distributed Tests. - IFIP 12th International Workshop on Testing of Communicating Systems (IWTCS'99), Budapest (Hungary), Sept. 1999.