

# Implementation of TTCN-3 Test Systems using the TRI

S. Schulz and T. Vassiliou-Gioles

*Nokia Research Center, P.O. Box 407, FIN-00045 NOKIA GROUP, Stephan.Schulz@nokia.com*

*Testing Technologies IST GmbH, Bernburger Str. 24-25, D-10963 Berlin,  
vassiliou@testingtech.de*

**Abstract:** This paper discusses the implementation of test systems, which are specified based on TTCN version 3 (TTCN-3). TTCN-3 has recently been standardized by [4] European Telecommunications Standards Institute (ETSI) as a new test specification language beyond protocol testing. TTCN-3 source code however only constitutes the first step in a TTCN-3 test system implementation. A complete test system implementation also requires the specification of a test and platform specific adaptation layer. This issue is addressed in the TTCN-3 Runtime Interface (TRI) draft standard also developed at ETSI. The TRI is expected to become the future standard interface for all TTCN-3 test system implementations. We present different kinds of test systems which can be built from the conceptual model provided by the TRI. Next to a discussion on the application of these concepts for classic conformance testing, we also propose a new approach, which significantly simplifies the specification and implementation of TTCN test systems for IP-based applications. We illustrate this approach with the presentation of an application example.

**Key words:** TTCN Standards, Conformance Testing, Test System Implementation, Communication Protocols, Testing of IP/CORBA-based Applications

## 1. INTRODUCTION

TTCN (Tree and Tabular Combined Notation) was first published as a standard in 1992 by the International Organization for Standardization (ISO). Since then the notation has grown into wide spread use and popularity, especially for conformance testing for telecommunication protocols. Nevertheless, the tight integration of previous versions of TTCN [2] with the classical OSI (Open Systems Interconnection) conformance architecture [1] and their lack of flexibility [6] have made test specification cumbersome for applications which use text-based protocols, e.g., IP-based applications. Another shortcoming of this testing language so far has been its tabular specification format.

These shortcomings have now been addressed in the third version of this testing language called TTCN-3 [4]. Arguably the most visible change compared to its predecessor TTCN-2 [2] is the extension of test specification to a wider range of presentation formats. TTCN-3 is based on a textual core notation, which has a similar look and feel of a regular programming language. It offers a number of additional new features that make it more powerful and capable of testing in a wider range of application areas, e.g., procedure-based communication, and dynamic testing configurations during the execution of test cases [7].

TTCN-3 enables the abstract specification of test cases. In a second step though this test specification needs to be integrated with some other test specific parts and a TTCN-3 execution environment to arrive at a complete TTCN-3 test system implementation. Concepts for the implementation of such test systems have recently been addressed in the context of the TTCN-3 Runtime Interface (TRI) draft standard [8] which has been accepted as a technical report by ETSI in October 2001. A revision of the document is expected to upgrade it to an ETSI technical standard by the end of the year. The purpose of the TRI is to specify a small and well-defined runtime interface for all future TTCN-3 test system implementations. The TRI defines this interface as a set of operations which are either to be offered by a TTCN-3 execution environment or implemented by the user in a test specific adaptation layer.

A similar run-time interface has previously also been introduced for TTCN-2 test system implementations, i.e., the Generic Compiler/Interpreter Interface (GCI) [3,5]. The lack of acceptance of the GCI within the TTCN community has shown that the GCI had been formulated inadequately as it forced developers to use one test system implementation style. The definition of the TRI attempts to address the shortcomings of its TTCN-2 counterpart by excluding test execution management, test component

communication and a value interface from the standard, and by specifying the runtime interface independent of a target implementation language.

In this paper we will first show the user perspective of a TTCN-3 test system and follow with a discussion of the implementation perspective, i.e., the conceptual model for TTCN-3 test system implementations on which the TRI is based. We follow with the presentation of two possible test system implementations, i.e., a conformance and a network tester, which arise from the TRI, and illustrate the construction of TTCN-3 Network Testers for a SIP proxy server. Finally, we emphasize the likely impact of this standard for future TTCN-based testing in our conclusions.

## 2. THE USER PERSPECTIVE OF A TTCN-3 TEST SYSTEM

Prior to discussing the TRI and a implementation perspective for test system implementation it is worthwhile to briefly review the user perspective of a TTCN-3 test system. Figure 1 shows an abstract view of a TTCN-3 test case as it is presented in the TTCN-3 standard [4]. It shows a test configuration during the execution of a test case consisting of a main (MTC) and a parallel test component (PTC). Notice that the figure depicts only a snapshot of a test system during the execution of a test case as the creation of test components as well as connections between components are dynamic in TTCN-3.

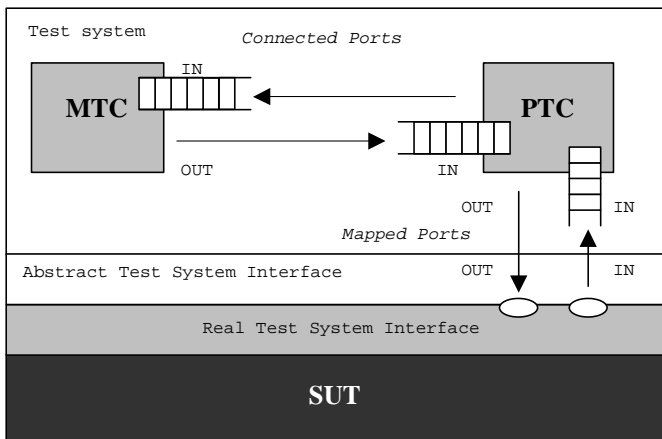


Figure 1. Conceptual view of a TTCN-3 test configuration

A TTCN-3 test case is specified by the user in the form of TTCN-3 source code. A collection of such test cases is also known as the Abstract

Test Suite (ATS). A test case specifies the instantiation of test components and their behavior. Test components can interact with each other after they connect via communication ports. Notice that new TTCN-3 feature is the possibility to specify one-to-many test component connections.

Secondly, a **system** component is generally used in each test case to specify an abstract test system interface. This abstract test system interface provides a means for test components to communicate with the System Under Test (SUT) via the real test system interface. The real test system interface can be understood as a test specific adaptation layer which establishes and handles the interaction between an executable form of the TTCN-3 ATS and the SUT [4].

In the TTCN-3 source code, the abstract test system interface is specified as a collection of communication ports, also called test system interface (TSI) ports, which are declared for the **system** component. During the execution of a test case ports of test components can be mapped dynamically to test system interface ports to establish either message or procedure based communication channel to the real test system interface.

Notice that Figure 1 (or the TTCN-3 standard) does not make any statement about the implementation of communication with the SUT, TTCN-3 external functions, timers, or SUT operations. These issues are nevertheless of relevance in an actual test system implementation or the implementation perspective of a TTCN-3 test system.

### 3. A CONCEPTUAL MODEL FOR TTCN-3 TEST SYSTEM IMPLEMENTATIONS

Concepts for a general TTCN-3 test system implementation, also commonly referred to as "tester", have been developed recently to define the TTCN-3 Runtime Interface (TRI). Figure 2 shows the conceptual model of a TTCN-3 tester proposed in the TRI standard which also serves as the basis for discussion in this paper.

In this figure, the TRI can be considered to be the most important interface in a test system implementation as it is located between a given or general part, i.e., the TTCN-3 test execution environment, and test specific parts, which implement communication with the system under test and aspects specific to the underlying processing platform of the test system. From this figure, Test Management, TTCN-3 Executable, and even parts of the Platform Adapter entities would be provided by vendors of TTCN-3 development tools whereas SUT Adapter and the remaining Platform Adapter functionality are specified by a test system developer.

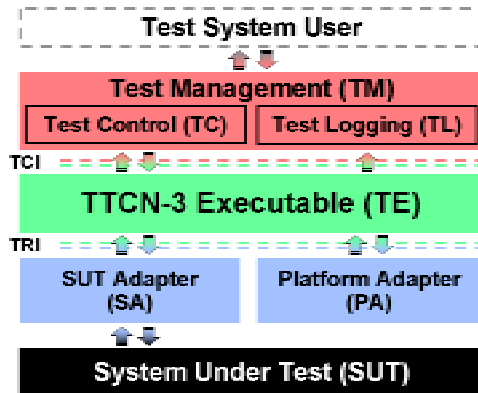


Figure 2. General Structure of a TTCN-3 Test System

### 3.1 TTCN-3 Test System Overview

A TTCN-3 test system can be conceptually thought of as a set of interacting entities where each entity corresponds to a particular aspect of functionality required in a test system implementation. The part of the test system which implements interpretation or execution of a TTCN-3 test specification is represented by the TTCN-3 Executable (TE) entity. In a TTCN-3 tester this entity corresponds either to an interpreter or an executable test suite (ETS) produced by a compiler from a TTCN-3 ATS.

The remaining part of the TTCN-3 test system, which deals with any aspects that can not be concluded from information being present in the original ATS alone, can be decomposed into Test Management (TM), SUT Adapter (SA), and Platform Adapter (PA) entities.

As depicted in Figure 2, a TTCN-3 test system has two major interfaces, the TTCN-3 Control Interface (TCI) and the TTCN-3 Runtime Interface (TRI), which specify the interface between Test Management (TM) and TTCN-3 Executable (TE) entities, and TE and Adapter entities, respectively. Notice that current standardization efforts only focus on the definition of a TRI to allow TTCN-3 tool vendors a greater flexibility in their implementation of test execution environments. The real test system interface shown in previously Figure 1 is implemented by TRI operations which implement the interface between TE and SA.

## **3.2 Test System Entities and the TRI**

This section briefly outlines the functionality associated with the entities depicted previously in Figure 2 and shows their relation to the TRI. The focus here is on a general understanding of these entities. For a more detailed discussion we refer to [8].

### **3.2.1 The Test Management Entity**

Within the Test Management (TM) entity, we can distinguish between test execution control and logging related functionality. The TM entity is responsible for general tester management and includes the implementation of a user interface and general test execution management, e.g., the preparation of the test system for a test execution, the collection of final verdicts, and propagation of module parameters to the TTCN-3 Executable (TE) [4]. In addition, the TM maintains a log of test execution. This test system entity does not interact via the TRI.

### **3.2.2 The TTCN-3 Executable Entity**

The TTCN-3 Executable (TE) is responsible for the interpretation or execution of the TTCN-3 ETS. We can think of the TE as implementing three separate tasks or aspects in the execution of a TTCN-3 test specifications: control of test case execution, proper execution of TTCN-3 behavior, and TTCN snapshot semantics [2].

The control aspect mainly implements the control part specified in a TTCN-3 ATS. It addresses, e.g., the proper sequencing of test cases or functions, collecting and resolving intermediate verdicts, and interacting with the TM to obtain module parameters. Other tasks include the initialization SUT and Platform Adapters prior to the execution of a test case as well as possibly the invocation of TRI operations for SUT operations, TTCN-3 external functions, and timer implementations.

The behavioral aspect of the TE implements the execution or interpretation of test cases and functions as defined in the corresponding TTCN-3 modules [4]. It also includes propagation and matching of test events, and management of TTCN-3 test configurations. Other tasks include the invocation of TRI operations for mappings of test component ports to test system interface ports, message or procedure based communication with the SUT, TTCN-3 external functions, SUT operations, and timer operations.

Notice that the TE entity is responsible for encoding test data prior to the invocation of TRI communication operations and decoding test data when receiving test data from the SUT Adapter according to the encoding rules

specified in the TTCN-3 ATS. The implementation of the behavioral aspect also addresses message and procedure based communication operations between TTCN-3 test components. In addition, only TTCN-3 semantics of procedure based communication with the SUT need to be implemented in the TE. All remaining parts of procedure based communication operations with the SUT are implemented outside of the TE, i.e., in the SUT Adapter [8]. Similarly, all timers are implemented in the Platform Adapter.

Finally, a TTCN-3 Executable (TE) needs to maintain its own port queues for receiving operations and a list for timeouts to properly implement TTCN snapshot and timer semantics [2,4]. These queues store test events that the TE has been notified of by the SUT Adapter but which have not yet been processed. Similarly, the timer list stores timeouts that the TE has been notified of by the Platform Adapter.

### **3.2.3 The SUT and Platform Adapter Entities**

The purpose of the SUT Adapter is to implement all message and procedure based communication of the TTCN-3 test components with the SUT. This entity serves as a means to properly propagate send requests to the SUT as well as SUT operations invoked by the TTCN-3 executable (TE) and to notify the TE of any received test data. It also realizes TTCN-3 SUT operations. Notice that implementation of procedure based communication in the SUT Adapter is responsible to distinguish between the different messages types (i.e., call, reply, and exception).

The Platform Adapter (PA) implements TTCN-3 external functions and provides a TTCN-3 test system also with one notion of time. Notice that although timers can be instantiated in the TE they are implemented in the PA [8].

## **4. TTCN-3 TESTER CONSTRUCTION**

The specification of TTCN-3 ATS is only the first step in the construction of a TTCN-3 tester. A complete tester requires also the implementation of SUT and Platform Adapters. We focus here in our discussion on the implementation of the SUT Adapter as it always needs to be specified by a TTCN-3 user. As stated before, it can be assumed that Platform Adapters are for the most part (i.e., platform specific timer implementations) provided by vendors of TTCN-3 development tools.

We identify and describe in the following sections two types of TTCN-3 testers, which can be derived from the conceptual model presented in the previous section. The classical application of TTCN so far has been protocol

conformance testing. This type of tester specification and implementation is still possible in TTCN-3 and will be outlined in the next section. In the following section we then introduce concepts for a second type of tester, called a network tester. Network testers promise a much simpler specification of TTCN-3 testers for, e.g., IP-based applications, by abstracting away the details of protocol based conformance testing.

#### 4.1 Classic Conformance Testers

The construction of TTCN-3 conformance testers follows straight from the methodology provided by ISO standard 9646 [1,2]. Communication of the TTCN-3 test suite with the real test system interface, i.e., the protocol stack, is achieved via points of control and observation (PCOs). Here, test components communicate with the top layer of the protocol stack by sending and receiving Abstract Service Primitives (ASPs), which may contain Protocol Data Units (PDUs), via PCOs. In a TTCN-3 tester, all protocol stacks are implemented in the SUT Adapter by following, e.g., the well known OSI protocol layer model [9]. This approach is illustrated in Figure 3.

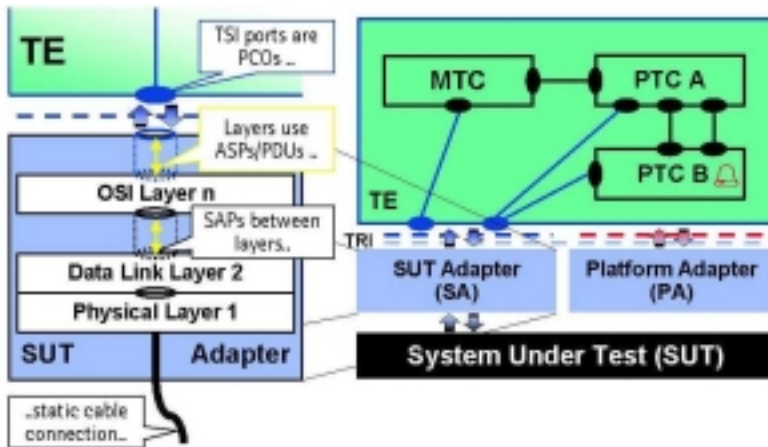


Figure 3. A Classic TTCN-3 Conformance Tester Example

In TTCN-2, PCOs represented component ports and access points to the protocol stack at the same time. In TTCN-3, they are specified slightly differently. Here, Test System Interface (TSI) ports take the role of PCOs. TSI ports still be represent access points to an underlying protocol but they are no longer directly associated with a specific test component. Notice that many-to-one test component mappings to a PCO, i.e., TSI port, allow now multiple test components to share one protocol stack implementation. A TTCN-2 like tester structure, however, can still be achieved by only

specifying dedicated (one-to-one) mappings of test component ports to TSI ports in the TTCN-3 ATS.

SUT Adapters for classic conformance testers are implemented similar to TTCN-2 protocol stacks. Whereas protocol stacks have usually been tightly integrated in a tool dependent manner with test components in TTCN-2 testers, they are implemented separately from each other in TTCN-3 testers. A set of TRI operations provides the required interface for test components in the TE to access stacks in the SA and vice versa. Main reasons for this new approach are the new TTCN-3 feature of an abstract test system interface (i.e., the **system** component) and greater flexibility for implementation of the TE in TTCN-3 execution environments.

TRI operations which need to be implemented by the test system developer for conformance testers as the bare minimum include the **triExecuteTestCase()** to initialize protocol stacks or establish required connections to the SUT, **triSend()** to implement ASPs or process and send PDUs to the SUT which have been received from the TTCN-3 ATS via a TSI port. Messages, which have been received by the tester from the SUT, are processed in the SA, eventually packaged within an ASP, and then sent from the SA via a TSI port to the test suite by invoking the **triEnqueueMessage()** operation. The information provided by **triMap()** and **triUnMap()** operations can be used by the SA to specify the appropriate destination test component for test data received from the SUT when many test components have been mapped to one TSI port.

## 4.2 Network Testers

The main motivation in the design of TTCN-3 has been to relieve this testing language from the peculiarities specific to conformance testing and to bring TTCN-3 to a larger audience, e.g., developers interested in a general approach to testing with no previous background in conformance testing. The new features of the TTCN-3 language as well as the TRI draft standard allow now also the specification a second type of tester, which we call a network tester. Network testers are intended for the testing of protocols or applications which operate at an abstraction level above the network layer, i.e., Layer 3 in the OSI model.

### 4.2.1 An Abstraction for Protocol Stack Interaction

In protocol-based conformance testing PCOs have been defined as access points for test components to protocol stack implementations. ASPs have enabled users to access information via PCOs or influence the propagation of information in the protocol stack as explained in the previous section.

This approach introduces unnecessary overhead and implementation complexity to testers where test components only use a minimal set of ASPs to interact with the protocol stack: establishing and closing a connection, and sending or receiving information via this connection. Testers for applications which use computer network protocols, most importantly internet protocols, fall into this category. Similarly, testers for applications in early stages of development (e.g., for which no protocol has been selected yet) such an interaction abstraction would suffice in many situations.

Since the TTCN-3 language already offers constructs to establish and close connections to a TSI port, and to send or receive data via a TSI port, the specification and implementation of testers for applications which use such protocols can be significantly simplified. Notice that the SA then establishes connections to the SUT dynamically based on TTCN-3 components *mapping* to a TSI port instead of a static association of each TSI port (or PCO) with a single SUT connection.

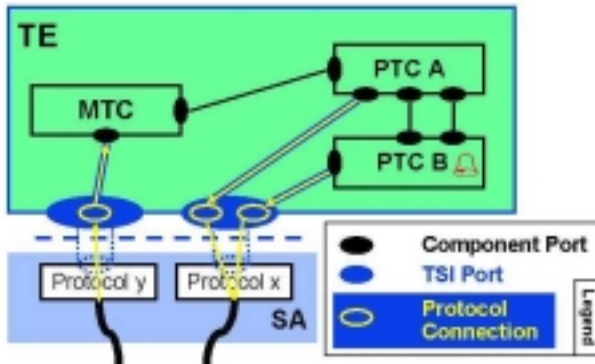


Figure 4. TTCN-3 Network Tester Example

#### 4.2.2 Implementation of TTCN-3 Network Testers

The result of the interaction abstraction is that TSI ports can be interpreted as complete protocol implementations. Mapping a test component port to a TSI port here implies the opening of a new connection with this protocol. Similarly, unmapping from this TSI port corresponds to closing that connection. Since the invocation of the TTCN-3 operations `map()` and `unmap()` has been directly correlated in the TRI standard with invocations of `triMap()` and `triUnmap()` operations the execution of such TTCN-3 operations serves in essence as a signal for the SA to take an action. The mapping of multiple components to the same TSI port simply results in several connections to the SUT using one specific protocol. Notice

that in the specification of our TTCN-3 ATS we can ignore the details of how exactly this connection is established. An example network tester is shown in Figure 4.

SUT Adapters for network testers consist in this case mainly of complete protocol implementations. As in TTCN-3 conformance testers the user may implement in the SA the `triExecuteTestCase()` operation to initialize these protocol implementations or to establish a physical connection to the SUT statically for the duration of a test case if necessary. In network testers, however, `triMap()` and `triUnmap()` operations are used to administer protocols and connections, e.g., they invoke protocol specific operations to open or close a connection. Notice that this connection to the SUT is here more likely to be a virtual as opposed to a physical connection. `triMap()` and `triUnmap()` operate on component and TSI port types and allow therefore a flexible dynamic test configuration, if so desired.

Similarly, the `triSend()` operation is used now to invoke in the SA the actual protocol specific send primitive with the proper parameters. Messages, which have been received by the tester from the SUT on a connection are sent from the SA to the test suite via a TSI port by invoking again the `triEnqueueMessage()` operation. In this invocation the desired destination test component in the TTCN-3 ETS is determined from the information generated in previous mappings.

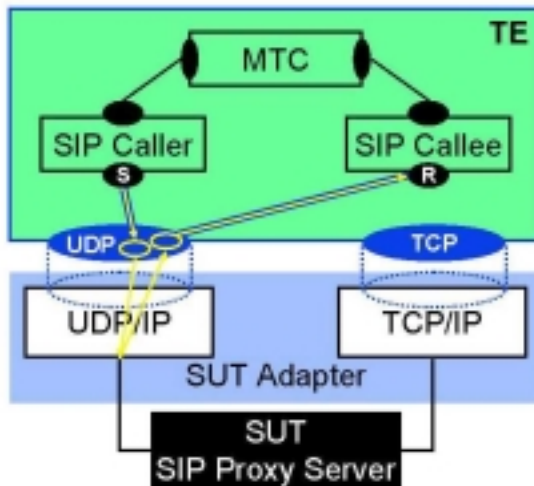


Figure 5. General SIP Proxy Test Architecture for UDP Test

## 5. A TTCN-3 NETWORK TESTER APPLICATION EXAMPLE

To illustrate the construction of a TTCN-3 network tester we present a real world IP-based application example. Our example is a tester that checks the ability of a Session Initiation Protocol (SIP) [10] proxy server to build up connections between a calling and a called party. SIP is an IETF signaling protocol for establishing multimedia connections in the internet and future telecommunication networks which may use either on top of UDP/IP or TCP/IP as its transport protocol. Figure 5 displays a general test architecture.

The figure shows that the test system consists of two parallel test components (SIP Caller and Callee) and one coordinating Master Test Component (MTC). The test system contains two TSI ports, i.e., one for the sending and receiving of SIP messages using UDP and one for using TCP.

### 5.1 TTCN-3 ATS Specification Examples

A TTCN-3 ATS fragment for such a test configuration could be specified as follows:

```

module sipTest {
  testcase UdpTcpTest() runs on MyMtcType system MyTSI {
    var MyPtcType caller = MyPtcType.create;
    var MyPtcType callee = MyPtcType.create;
    map(caller:S, system:UDP); map(callee:R, system:UDP);
    caller.start(callerBehavior(USER1));
    callee.start(calleeBehavior(USER2));
    all component.done;
    unmap(caller:S, system:UDP); unmap(callee:R, system:UDP);
    caller = MyPtcType.create;
    callee = MyPtcType.create;
    map(caller:S, system:TCP); map(callee:R, system:TCP);
    caller.start(callerBehavior(USER1));
    callee.start(calleeBehavior(USER2));
    all component.done;
  }
}

```

First, the MTC creates the SIP caller and callee test components and maps their ports to the UDP test system interface port. In this example it is assumed that the behavioral functions callerBehavior() and calleeBehavior() perform all the necessary actions to assess the SUT. For the TCP tests the

same behavior can be reused. PTCs can reuse the component structure while the MTC takes care of the proper test configuration. The SUT Adaptor handles in such a dynamic configuration the protocol and connection management. This split development allows to focus in the specification of test behavior on the coordination and message exchange between the PTCs and ignore for the details of underlying protocol implementations.

TTCN-3 had been designed not only for conformance or functional testing but it should be suitable also for other types of testing like robustness, scalability, interoperability testing, etc. The second scenario is a scalability test where behavior specified for the test case `UdpTcpTest()` are reused. Here, the SUT is tested to handle an arbitrary number of caller/callee pairs which may use both, UDP and TCP, communication means at the same time.

This scenario is illustrated in the following TTCN-3 ATS fragment:

```

testcase scalabilityTest() runs on MyMtcType system MyTSI {
  var integer i;
  for(i:=1; i < MAXNUMBER; i := i +1) {
    caller[i] = MyPtcType.create;
    callee[i] = MyPtcType.create;
    if(tcpConnection() == true) {
      map(caller[i]:S, system:TCP);
      map(callee[i]:R, system:TCP);
    } else {
      map(caller[i]:S, system:UDP);
      map(callee[i]:R, system:UDP);
    }
    caller[i].start(callerBehavior(USER1 & int2char(i)));
    callee[i].start(calleeBehavior(USER2 & int2char(i)));
  }
}

```

The shown fragment uses component arrays to store references for the PTCs and a function `tcpConnection()` to decide whether to build up a TCP connection or not for a certain caller/callee pair.

## 5.2 Implementation of TRI Operations

While the above examples have shown the ability of TTCN-3 to specify dynamic test configurations this section will highlight some implementation issues with respect to TRI. TRI defines two language mappings, a Java and C language mapping [8]. In the following Java will be used as implementation language.

Before the first statement of the MTC will be executed by the TE it will call the `triExecuteTestcase()` operation within the SA, to indicate that the referenced test case with the test system interface port list will be executed. Assuming a static approach to test configuration the communication means (protocol emulation, connections) would be created and parameterized in the implementation of this operation. Since however the SA provides both, UDP and TCP protocols, the scalability scenario introduced above requires initialization of a fixed number of TCP and UDP "connections" to the SUT so that all possible send and receive statements can be implemented. Another approach is to avoid an unnecessary creation of communication means by creating connections depending on the test case to be executed. While the first approach leads to an inefficient tester implementation the later results in a test suite behavior specific test adapter. With the introduction of new test cases the SA would have to be modified.

These problems can be completely avoided by using TRI operations `triMap()` and `triUnmap()`. Connections can be created on demand based on the respective test system interface ports used, thus supporting the newly introduced dynamic test configuration of TTCN-3 in an efficient way. Next to reducing required resources an universal SUT Adapter can be implemented. The following Java code-fragment illustrates this approach:

```
public class MySA implements TriCommunicationSA {
    public TriStatus triExecuteTestcase( TriTestCaseID TCId,
                                         TriPortIdList tsiPorts) {

        // just do nothing
        return new TriStatus (TRI_OK); // Indicate nevertheless OK!
    }
    public TriStatus triMap(   TriPortId compPortId,
                              TriPortId tsiPortId) {
        String portName = tsiPortId.getPortName();
        if(portName.equals("UDP"))
            mapUdpPortAndCreateSocket(compPortId, tsiPortId);
        else
            mapTcpPortAndCreateSocket(compPortId, tsiPortId);
        return new TriStatus (TRI_OK);
    }
    public TriStatus triUnmap( TriPortId compPortId,
                               TriPortId tsiPortId) {
        String portName = tsiPortId.getPortName();
        if(portName.equals("UDP"))
            unmapUdpPortAndFreeSocket(compPortId, tsiPortId);
        else
```

```
// etc ...
}
public TriStatus triSend( TriComponentId componentId,
                        TriPortId      tsiPortId,
                        TriAddress      address,
                        TriMessage      sendMessage) {
    String portName = tsiPortId.getPortName();
    if(portName.equals("UDP")) {
        fetchUdpSocket(componentId).send(sendMessage);
    } else if(portName.equals("TCP")) {
        // etc ...
    }
}
```

This code fragment shows how a test suite behavior independent SA might be implemented. It is by no means complete and should merely illustrate the building and maintaining of communication means (here sockets) solely based on TRI type information, e.g., that port names are obtained in the implementation of the TRI send operation from the source component type. Furthermore it does not address issues with respect to the physical distribution of test components that might be necessary to handle limitation of resources of a single test device.

In addition, it can be concluded from this examples that the resulting SUT Adaptor can not only be used for a SIP test suite but for a multitude of test suites that use UDP and TCP connections as long as the test system interface ports are referenced in the same manner.

## 6. CONCLUSIONS

We have discussed in this paper the use of the latest version of the TTCN testing language (TTCN-3) and test system implementation concepts defined in the TTCN-3 Runtime Interface (TRI) standard for conventional conformance testing as well as the testing of IP-based applications. We have presented the construction of two different types of TTCN-3 testers, a "classic" conformance tester and a newly defined network tester. For both types of testers we focused on the implementation of TRI operations which manage communication with the SUT, i.e., the SUT Adapter. Our SIP application example has shown that a much more compact and simpler TTCN-3 test system implementation can be achieved for IP-based protocols or applications by applying our network tester approach. Finally, this paper

has pointed out the importance and impact of the TRI in the construction of future TTCN-3 testers in general.

## ACKNOWLEDGEMENTS

The authors would like to thank all the contributors at the numerous TRI meetings and discussions on the TRI mailing list which have inspired the writing of this paper. At Nokia Stephan thanks Dr. Thomas Deiß for his thorough review and Dr. Colin Willcock, Dr. Stephan Tobies, Jan Spudich, and Martti Söderlund for many discussions about TTCN-3 and tester implementation. Theo thanks at FOKUS Dr. Ina Schieferdecker for pushing the TRI, and at TU Berlin Jacob Wieland, Baltasar Trancon-y-Widemann, and Markus Lepper for intense discussions on TTCN-3 and the TRI.

## REFERENCES

- [1] International Organization for Standardization, "Information technology - Open Systems Interconnection - Conformance testing methodology and framework - Part 1: General Concepts", ISO/IEC 9646-1, 2<sup>nd</sup> ed., Geneva, December 1994.
- [2] International Organization for Standardization, "Information technology - Open systems interconnection - Conformance testing methodology and framework - Part 3: The Tree and Tabular combined Notation (TTCN)", ISO/IEC 9646-3, 2<sup>nd</sup> ed., Geneva, November 1998.
- [3] P. Cousin, "The European initiative for the development of Infrastructural Tools: the INTOOL programme.", *Proceedings of 10th International Workshop on Testing of Communications Systems*, Cheju Island, Korea, 8-10, September 1997.
- [4] European Telecommunications Standards Institute, "Methods for Testing and Specification (MTS); The Tree and Tabular Combined Notation version 3; Part 1: TTCN-3 Core Language", ETSI ES 201 873-1 (v1.0.11), Sophia Antipolis, May 2001.
- [5] F. Brady and R.M. Baker, "INTOOL/GCI; Generic Compiler/Interpreter interface; GCI Interface Specification", INTOOL CGI/NPL038 (V2.2), Infrastructural Tools for Information Technology and Telecommunications Conformance Testing, December 1996.
- [6] J. Grabowski and D. Hogrefe, "Towards the third edition of TTCN", *Proceedings of 12th International Workshop on Testing of Communication Systems*, Budapest, Hungary, 19-29, September 1999.
- [7] J. Grabowski, A. Wiles, C. Willcock, and D. Hogrefe, "On the design of the new testing language TTCN-3", *Proceedings of 13th International Conference on Testing of Communicating Systems (TestCom)*, Ottawa, Canada, 161-76, August 2000.
- [8] European Telecommunications Standards Institute, "The TTCN-3 Runtime Interface (TRI); Concepts and Definition of the TRI", ETR, v4.2, September 2001.
- [9] G. Holzmann, *Design and validation of computer protocols*, Prentice-Hall, 1991.
- [10] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol", Draft IETF SIP RFC 2543bis-04, July, 2001.