A Simplified and Efficient LTE RRC Conformance Testing Adapter

Teng Huang[1], Zhizhong Ding[1*], Younan Duan[1], Yin Chen[1], Lu Ding[2]

[1] Institute of Communications and Information Systems, Department of Communication Engineering, Hefei University of Technology, Hefei, 230009, China.
[2] Western Digital Corporation, Irvine, California 92612, USA.

* Corresponding author. Email: zzding@hfut.edu.cn
Manuscript submitted January 26, 2014; accepted October 28, 2014.

Abstract: RRC (Radio Resource Control) protocol belongs to the LTE protocol stack and handles the control plane signaling of Layer 3 between the User Equipment and E-UTRAN. The specifications of RRC conformance testing are released by 3GPP in the 36-series documents. RRC test suites are normally developed by use of the testing language TTCN-3 which has the characteristics of platform independence and abstract description. As a key component in testing architecture, the adapter propagates the request and reply operations between tester and SUT (System under Test). In order to fulfill data transmission within a limited time according to the corresponding protocol, developers have to complete a complicated configuration of adapter. This paper focuses on the design of the adapter. First it analyzes the main framework of test system and constructs the concrete structure. Then the mechanism and workflow are illustrated. Finally, we elaborate how the adapter can achieve better performance to acquire accurate timing control and to implement multi-protocol communications.

Key words: Adapter, LTE, protocol conformance test, RRC, TTCN-3, TRI.

1. Introduction

Radio Resource Control Test has been one of the significant LTE Protocol Conformance Tests since the 3GPP 36 series specifications were released. Usually the test projects need a reasonable model to build a testing system. The general structure of the testing system based on TTCN-3 was published by the European Telecommunications Standards Institute (ETSI). According to the testing system model, the testing developer needs several basic devices including a Host-PC, a System Simulator (SS) and the User Equipment (UE) which is regarded as System under Test (SUT). In this solution, the Host-PC and the User Equipment (UE) are connected with special interfaces of the testing entities. The UE and SS transmit data through the standard internal interfaces and the LTE air interfaces, but the Host-PC and SS exchange data packets through the wired channels without defined regulations in the specification. Therefore the testing developers need an effective adapter as a connection between the TTCN-3 Executable (TE) and SUT. The adapter should have high efficiency and reasonable mechanism to meet the requirements of the released specifications.

This paper presents an adapter implementation which is able to invoke numbers of interfaces effectively and select the appropriate protocol. It also needs to support parallel processing. The proposed adapter can achieve better performance due to that fewer communicating ports and threads are used. Compared with some commercial tools, the proposed mechanism can avoid to run too many threads so that it can acquire

better performance. For developers, it can be used to identify the messages to complete the configuration expediently.

The rest of the paper is structured as follows: In Section 2, we give an analysis of the TTCN-3 general structure, TTCN-3 Runtime Interfaces specifications and the mechanism of the adapter. In Section 3, we propose a structure of the adapter, compare it with other solution, and elaborate how to implement it to meet the qualification of the test procedure. Conclusions and future work are summarized in Section 4.

2. Using TTCN-3 for the Test Project

TTCN-3, shorted for Testing and Test Control Notation version 3, is developed by the ETSI & ITU-T and widely used in the development of telecommunication test including the LTE protocol conformance test. We can easily use the TTCN-3 to describe the procedure of the communication test. Both of the message-based and procedure-based mechanisms are supported since it has some features different from the normal programming languages. Besides, it has built-in data matching, distributed architecture and a larger type system than normal language including the verdicts type. The verdicts type returns the result of testing steps in the procedure. It also has the excellent support for the timers as well as the parallel testing system [1].

There are a few open source tools which can be downloaded from the websites of TTCN-3 developing organizations. Several commercial development kits as well as the compliers have been released. For example, TTworkbench developed by the Testing Technology is a commercial integrated development environment (IDE), which is based on Eclipse framework and provides effective ways to analyze and execute test suites. The adapter can be loaded into it when executing the test suites released by 3GPP. LoongTesting, a free IDE developed by USTC TTCN Lab，integrates the TTCN-3 interpreter, complier and basic functions and leaves to developers more free space and workload to build a testing system due to the lack of ready-made plug-in units.

2.1. General Structure of Testing Solution

Fig. 1 shows a general structure of a test system that includes several essential parts: TE, adapter and SUT according to the ESTI specifications. TE communicates with TM, TL, Codec and CH through TCI, and exchanges data simultaneously with the Adapters through via TRI.

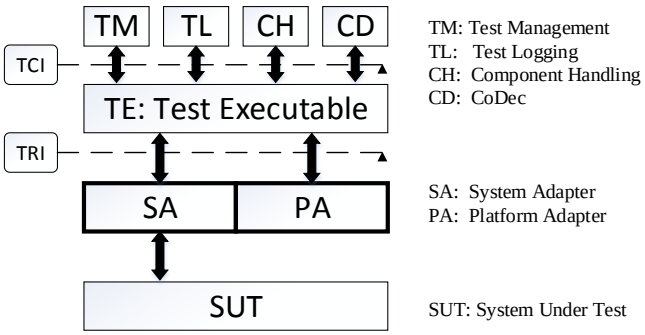

Fig. 1. General structure of TTCN-3 test system.

As an important part of the test system, the adapter's main functionality is to complete the so-called port mapping which connects the abstract interfaces of TE with interfaces of SUT. Considering the relationship between the entities and the abstract ones, we need to implement TRI specified by the specifications and the interfaces of the SUT. Since we aim to design an adapter of message-based RRC test, all the implementations should be completed only in SA.

2.2. Test Runtime Interfaces

The interaction between TE and SA is defined in the interface triCommunication in ETSI structure. The triCommunication initializes the Test System Interface (TSI) and completes the message transportation. To sum up, the adapter should have the ability to send data from TE to SUT, to enable TE to receive the feedback correctly and to check the timeout with the user-defined timer. Some essential interfaces for the RRC message-based communication test have to be implemented which are shown in the Table 1 [2].

Table 1. Some Essential Interfaces of TRI for RRC

TTCN3 instructions	TRI instructions	Explanation
Map	triMap	Connect the abstract ports and real ones
Unmap	triUnmap	Disconnect the abstract ports and real ones
Send	triSend	Send the messages from TE to SUT or in the opposite direction
	triSendBC	Broadcast commnication
	triSendMC	Multicast commnication
Execute	triExecute	Run the testcase
	triStartTimer	Start the timer
	triEndTestcase	End the timer
Timer stop	triStopTimer	Stop the timer

2.3. Mechanism of the Adapter

The adapter can only process the data encoded by Codec. The abstract data types of test case are converted into the binary code stream by the Codec. Adapter helps users to map the ports connection and ensure the data transmission with corresponding communication protocols.

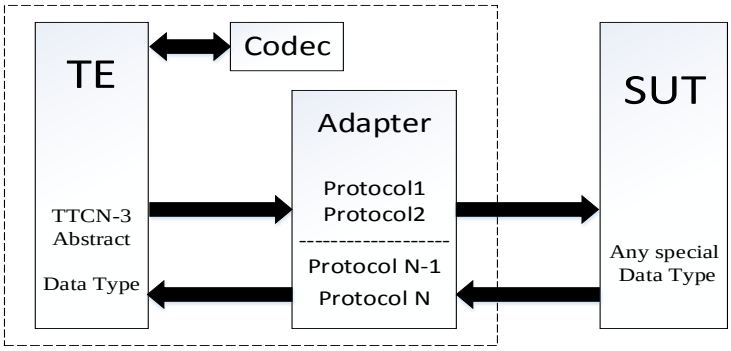

Fig. 2. Mechanism of adapter.

The specific mechanism of adapter is shown in the Fig. 2. No matter the transmission is from TE to SUT or the inverse direction, the data of messages have to be sent to the Codec for encoding, and the Codec returns the corresponding binary stream to the sender before delivering to the adapter. The binary streams of different Abstract Data Types or different abstract interfaces are transmitted over different protocols according to their bit rate and security requirement. In LTE, for instance, the two different interfaces DRB (Data Radio Bearer) and SRB (Signaling Radio Bearer) are used to transmit the data and signaling between TE and SUT over UDP and TCP respectively.

3. Design and Implementation of the Adapter

We choose the Eclipse-based tool, TTworkbench, to develop the adapter, so the language mapping from IDL (Interface Description Language) to high-level programming language is TTCN-3 to Java. All the special

methods and data types have been defined in the TRI specification of ETSI.

3.1. Design of the Structure of the System Adapter

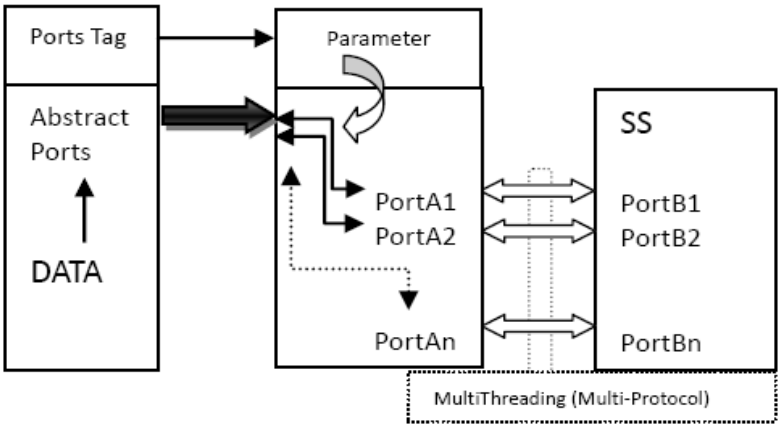

Fig. 3. Design of our system adapter.

In this paper, we concentrate on the design of Adapter to satisfy the requirements of LTE conformance test application.

In order to implement the mechanism shown in Fig. 2, we design a model shown in Fig. 3 to support the message sending and requests.

The TE has many different ports in 3GPP LTE code, e.g. mtc, ut. These abstract ports belong to different components. The adapter has many parameters that are mapped with the parameter in the TTCN-3 Test Suite. Obviously each real port has specific port number.

The messages in LTE test case are sent to the adapter through one of the abstract ports defined in the project. These messages can be identified by tags after defining and initializing mapping parameters in TTCN-3 level. Different tag corresponds to different protocol. The parameters in TTCN-3 level and in adapter are mapped by TRI. The adapter decides which specific port and corresponding protocol will be used.

3.2. Brief Introduction of LTE RRC Test Code

We can get the source code of RRC conformance test (3GPP LTE - Formal Delivery 36.523-3v10.1.0) from the 3GPP official website. It includes several LTE test cases, for instance the 6-1 Cellreselection.ttcn. In this huge message-based project, MTC (Main Test Component), PTCs (Parallel Test Components) and more than twenty ports from different components have been defined well. We regard these ports as abstract ports. Different type of messages are sent to or received from different abstract ports. For example, when setting the message of cell power, the TTCN-3 codes of the procedure are as follow:

```
var template (value)
CellPowerList_Type v_CellPowerList_AtT1 := {
cs_CellPower(eutra_Cell1, tsc_Suitable_NeighbourIntraFreq_CellRS_EPRE),
cs_CellPower(eutra_Cell2, tsc_ServingCellRS_EPRE)
};
f_EUTRA_CellInfo_SetSysInfo_Q_Rxlevmin
(eutra_Cell1, v_Q_Rxlevmin);
f_EUTRA_ModifySysinfoUE_Off (eutra_Cell1);
f_EUTRA_SetCellPowerList (v_CellPowerList_AtT1);
----------------------------------------------------------------------
SYS.send( cas_CellConfig_Power_REQ () );
if (v_CnfFlag) {SYS.receive(car_CellConfig_Power_CNF );}
```

In the above code segment, template defines two cells, eutra_Cell1 and eutra_Cell2, which are configured with cs_CellPower. The function f_EUTRA_SetCellPowerList that is a primary TS function in the main procedure of the LTE RRC test suite named as Cellreslection will change the power value of cells configured by cs_CellPower. The actual action of above configuring under low level is sending the message by the TE to the System Simulator that simulates the EPC and E-nodeB. If SS modifies the power to the specified value in the message, then the SS will reply a message like car_CellConfig_Power_CNF to let TE know the verdicts.

The method Sys.send sent the message cas_Cell Config_Power_REQ(...) to MTC via the SYS port of MTC, which will be transmitted further with the modified value of v_CnfFlag through the real channel. If the value of v_CnfFlag is true, the abstract port SYS will receive a message of template car_CellConfig_Power_CNF. The Sys.receive compares the coming message with the message template to check whether the type of received message is correct or not. That is so-called built-in data matching mechanism in TTCN-3. The adapter should have the ability to transmit the message sent by SYS that links with a real port and deliver it to SUT.

3.3. Adapter in Commercial IDE

In the latest version of TTworkbench (v15) released by Testing Technologies, TTplugins TCP and UDP used as TE adapter are pre-installed in the IDE, and TRI is also implemented in the IDE. Developers should load and configure the plugins before running the Test Suite. To use TTplugins TCP and UDP, one need to map the abstract ports of components to the real ports with the same unused port number first, and then select the transmit protocol. In the adapter, each abstract port, e.g. SYS, is connected with one real port of the operating system and each real port is mapped with one port of System Simulator.

As shown in Fig. 4, different links are used to transmit data of different abstract port although different links may use the same communication protocol. If we try to run the whole project 3GPP 36523-3, at least 28 real ports will be used to complete the mapping. TTworkbench allows 25 links. Normally, more than ten abstract system ports might be used in one RRC test project, that is, more than ten threads will be created. As the number of links and threads increase, the processing burden and the difficulty to control transmitting delay will increase while the stability will decrease. In addition, this solution results in inconvenience when configuring too many port numbers in the plugin.

On the contrary, in our design of the adapter, only a few of threads are necessary since the needed threads depend on how many protocols should be implemented.

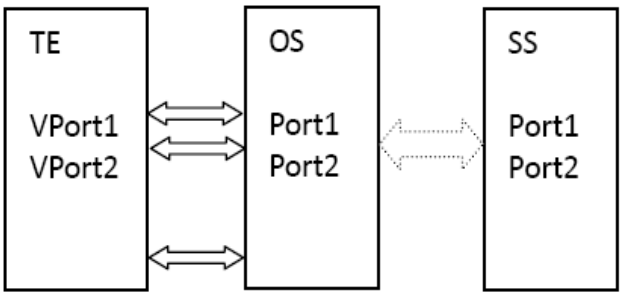

Fig. 4. Workflow of TTplugin.

3.4. Main Parts of the Testing Environment

To realize our adapter of RRC test, we will concentrate on the implementation of the TRI defined by the specifications, the transaction protocol for message exchange and the selection of multi-protocols. At start, the main framework is created in the TTworkbench that internally implements TM, TL and CH, while the Codec and TRI need to be implemented with Java language for user's specific project.

TTCN-3 Executable needs to complete the mapping. After the procedure finishes, it begins to process the encoded data as well as the data to be decoded, through different TRI according to the encoding rules. Some interfaces that have to be implemented shown as Table 2 [2].

Table 2. Interfaces and Parameters of TRI

Interface	Parameters	Explanation
TriStatusType triMap	in TriPortIdType compPortId, in TriPortIdType tsiPortId	**compPortId** identifier of component port **tsiPortId** identifier of system interface port
TriStatusType triUnmap	in TriPortIdType compPortId	**compPortId** identifier of the test component port to be unmapped **tsiPortId** identifier of the test system interface port to be unmapped
TriStatusType triSend	in TriComponentIdType componentId, in TriPortIdType tsiPortId, in TriAddressType SUTaddress, in TriMessageType sendMessage	**componentId** identifier of sending component **tsiPortId** identifier of system interface port via which message is sent to SUT Adaptor **SUTaddress** (optional) destination address within SUT **sendMessage** the encoded message to be sent
TriStatusType triExecuteTestCase	in TriTestCaseIdType testCaseId, in TriPortIdListType tsiPortList	**testCaseId** identifier of the test case going to be executed **tsiPortList** a list of test system interface ports defined for the test system
triStatusType triSAReset()	void	
void triEnqueueMsg	in TriPortIdType tsiPortId, in TriAddressType SUTaddress , inTriComponentIdType componentId , in TriMessageType receivedMessage	**tsiPortId** identifier of system interface port via which SUT Adaptor enqueues messages **SUTaddress** (optional) source address within SUT **componentId** identifier of receiving component **receivedMessage** received encoded msg.
TriStatusType triStartTimer	in TriTimerIdType timerId , in TriTimerDurationType timerDuration	**timerId** identifier of timer instance **timerDuration** duration of timer in seconds
TriStatusType triStopTimer	in TriTimerIdType timerId	**timerId** identifier of timer
TriStatusType triReadTimer	in TriTimerIdType timerId, out TriTimerDurationType elapsedTime	**timerId** identifier of timer **elapsedTime** value of elapsed time in seconds since the timer started
void triTimeout	in TriTimerIdType timerId	**timerId** identifier of timer

The workflow of TRI is shown in Fig. 5 in the next page. Since we only concentrate on the adapter, the implementation of TTCN-3 interpreter and complier are not in our consideration. It should be noted that we

can only observe the outputs of SUT interfaces and do not know the details of the components inside SUT. Therefore, we view SUT as a black box as we implement our adapter [3]. From this point of view, the test suit has the following structure:

```
testcase TC_6_1_2_2() runs on MTC_LTE system SYSTEM_LTE {
      timer t_GuardTimer := int2float(600);
      v_EUTRA := EUTRA_PTC.create alive;
      f_MTC_ConnectPTCs_LTE
(v_EUTRA, v_UTRAN, v_GERAN, v_CDMA2000, v_IMS1, v_IMS2);
      v_EUTRA.start(f_TC_6_1_2_2_EUTRA());
      t_GuardTimer.start;
      f_MTC_MainLoop(t_GuardTimer);
 }
```

The entrance of the adapter is triExecuteTestCase that starts the procedure, as shown in the following:

```
public TriStatus triExecuteTestcase
(final TriTestCaseId testcase,final TriPortIdList tsiList) {
TAParameterServer parameterServer =
(TAParameterServer) RB.TestAdapter;
remoteIPAddress = getTAParameter(pluginIdentifier, "mtcPort");
remotePortNumber = getTAParameter(pluginIdentifier, "mtcPort");
localPortNumber = getTAParameter(pluginIdentifier, "mtcPort");
            rxSocket = null;
            txSocket = null;
            return TriStatus;
      }
```

It can be seen from the above code description, the first step is connect operation that connects the ports of two test components. The port connections in the LTE testing are as follow [4].

```
connect(p_Eutra:IP, v_IP_PTC:EUTRA_CTRL);
connect(v_ImsPdn1:IMS_CTRL,          v_IP_PTC:IMS_CTRL[tsc_Index_PDN1]);
connect(v_ImsPdn1:IMS_Server,       v_IP_PTC:IMS_Server[tsc_Index_PDN1]);
      connect(v_ImsPdn1:IMS_Client,       v_IP_PTC:IMS_Client[tsc_Index_PDN1]);
connect(v_ImsPdn1:IPCAN,p_Eutra:IMS[tsc_Index_PDN1]);
if(p_Eutra!=null)
{connect(mtc:PTC_Ut[tsc_MTC_PortIndex_EUTRA],       p_Eutra:UT);}
```

The connect in TTCN-3 is an internal procedure. Suppose there are M component ports, a queue could be created to accommodate the M ports. The TE accesses each element of the queue and maps the ports between the component and the test system. Some key operations for the ports are defined in the following codes.

```
// TRI IDL TriPortIdType
package org.etsi.ttcn.tri;
public interface TriPortId {
public String getPortName();
public String getPortTypeName();
public TriComponentId getComponent();
public int getPortIndex();
```

```
}
public TriStatus triMap(final TriPortId compPortId,
                        final TriPortId tsiPortId) {
  //Check the linklist is whether exist
       if (v_PortIndex_exist==false)
       {return TRI_ERROR;
       }
       String ComPortName=compPortID.getPortname();
       String TsiPortType=tsiPort.getPortTypeName();
      //Map the TsiPort to the ComponentPort
       MapPort PortGroup=MapTsiPortId(tsiPortId,compPortId);
              //Add the Group of tsiPort and ComPort    into the PortIndex
                                                        which is a LinkList of the Test Execut     able
       PortIndex.addlast(PortGroup);
       return TRI_OK;}
```

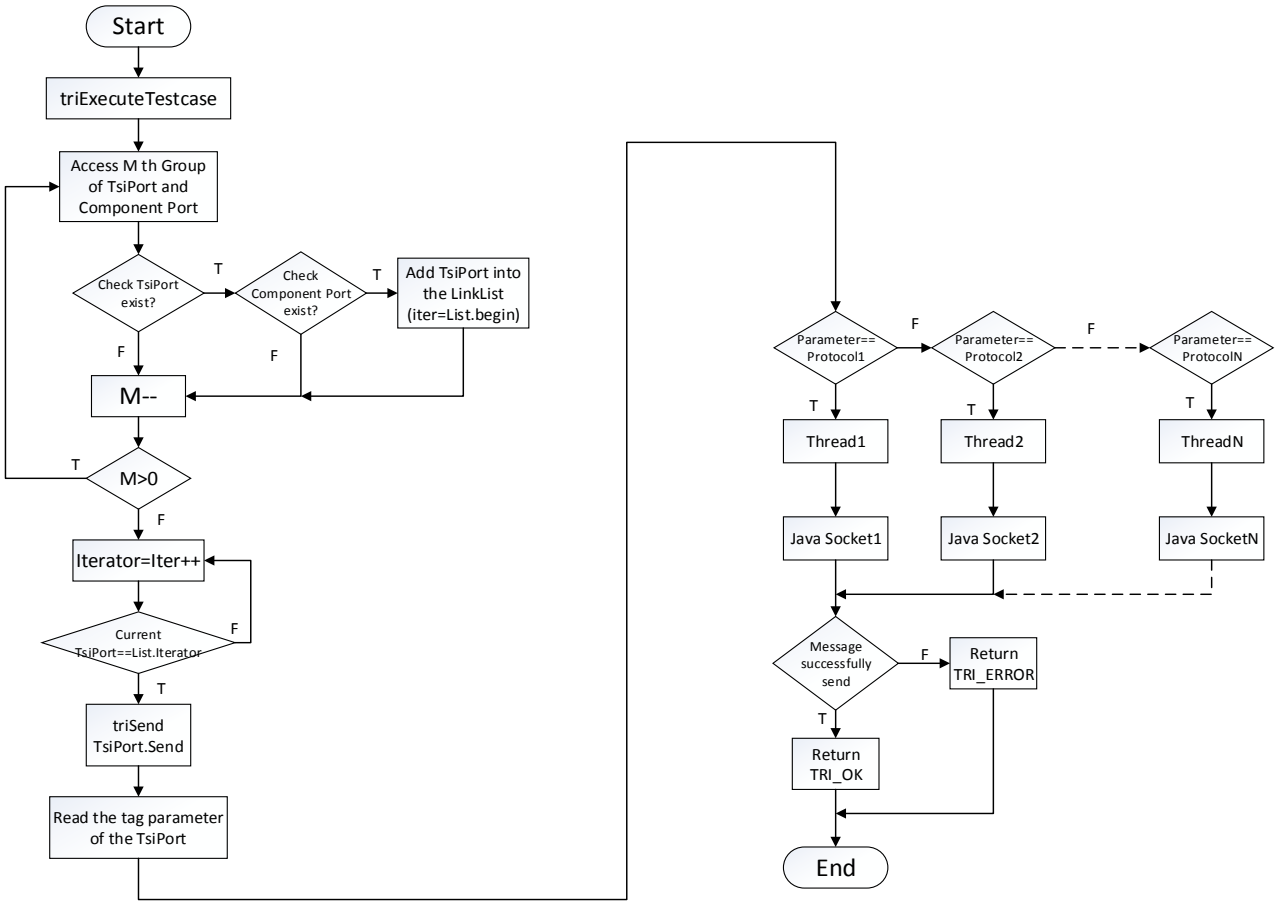

Fig. 5. Workflow of TRI loading.

After the above operations completed, all the messages in the queue wait to be sent by TsiPort. TTCN-3 test suite will send a LTE message that is encoded into binary stream. Adapter starts the traversal cycle until it finds the correct TsiPort with Java Iterator and then the related variable is set to TRUE.

Test suite executes triSend method of the corresponding TsiPort. TTworkbench provides a method getTAparameter in the package com.testingtech.ttcn.tri, which enables the adapter to get values from the TTCN project. Therefore, we define a variable 'tag' to mark each message in the TTCN project. According to variable ‘tag’ of TsiPort, the adapter is able to know which transmit protocol shall complete the action and which port is the right one assigned by operating system. The configuration of real ports should be defined

well in the adapter.

```
// TRI IDL TriMessageType
package org.etsi.ttcn.tri;
public interface TriMessage {
public byte[] getEncodedMessage();
public void setEncodedMessage(byte[] message);
public int getNumberOfBits();
public void setNumberOfBits(int amount);
public boolean equals(TriMessage message);
}
// TRI IDL TriAddressType
public interface TriAddress {
public byte[] getEncodedAddress();
public void setEncodedAddress(byte[] address);
public int getNumberOfBits();
public void setNumberOfBits(int amount);
public boolean equals(TriAddress address);
}
public TriStatus triSend(final TriComponentId componentId,
final TriPortId tsiPortId, final TriAddress address,
final TriMessage sendMessage) {
//    Get the Encoded Message from EDS
try {  //Create the sockets in this adapter
        //Transmit the Encoded message from EDS to the Sockets
                // TE may send the encoded Address to the Sockets
        } catch (IOException e) {e.printStackTrace();}
return new TRI_OK;
}
```

In the above code segment, the core function triSend have parameters componentId, tsiPortId, address and sendMessage. These four parameters contain the information of sending component, abstract ports in the 3GPP project, destination IP address and binary streams (namely the un-interpreted message data) respectively.

The specific conduit of information exchange is also established by Java sockets to process the encoded binary stream [5]. Within the range of the time delay, the Test Executable returns TRI_OK if the stream data is successfully sent to the right destination. If not, Test Executable will return TRI_ERROR, and then the verdict of this step will be set fail or inconclusive.

3.5. Transmission Protocol

In our adapter, the procedure of transmission involves two communication protocols TCP and UDP. Data transmissions over the two protocols are implemented in Java.

TCP is a connection-based Internet protocol and resides at the transport layer. It offers reliable and ordered data transmission in stream [6]. That is, it accepts data from a stream of octets, divides it into chunks and adds a TCP header to create a TCP segment [7]. The implemented Java stream operation and Java socket is as the following.

```
ServerThread(Socket pSocket) {
        socket = pSocket;
try {
        send = socket.getOutputStream();
        receive = socket.getInputStream();
```

```
        } catch (IOException e) {
        e.printStackTrace();}
byte[] buf=new byte[100];
int len=receive.read(buf);
send.write((BufferedMsg).getBytes());
```

UDP is a connectionless and transaction-oriented transport protocol. That is, there is no guarantee of delivery, ordering or duplicate protection when transmit data over UDP [8]. Java.net.Datagram includes several methods of datagram socket [9].

```
DatagramSocket server;
        try {
        server=new DatagramSocket(5050);
        byte[] sendBuf;
        byte[] addr =new byte[]{
                        (byte)192,(byte)168,(byte)1,(byte)103};
        sendBuf = sendMsgU.getBytes();
        DatagramPacket sendPacket = new DatagramPacket (
sendBuf,sendBuf.length,InetAddress.getByAddress(addr),5051);
        try {
        server.send(sendPacket);
        } catch (IOException e)
{
e.printStackTrace();
        }
        server.close();
        } catch (SocketException e1) {e1.printStackTrace();
        } catch (UnknownHostException e1) {}
```

3.6. Parallel Transmission

Different port number with corresponding protocol should establish the connections between TE and System Simulator at first. Different pipeline of data transmission works on different threads of the operating system. We divide the adapter into two parts, i.e. send and receive, to realize bi-direction communications. Each part involves different protocols according to the actual testing requirements.

After the links shown in the Fig. 6 are established, the adapter begins to read the parameter, i.e. the port tag, by the method getTAparameter, and then decides which port the stream should be delivered to.

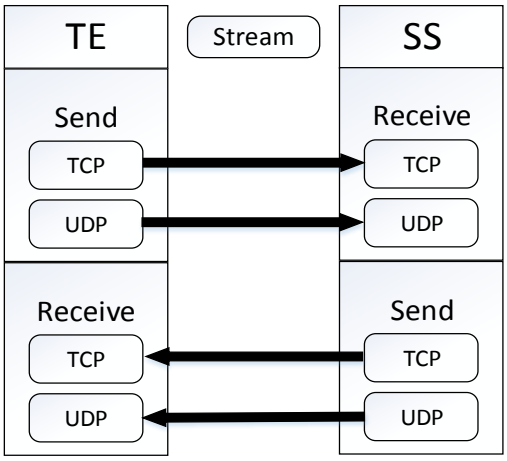

Fig. 6. Parallel transmission.

In Fig. 6, six threads in our adapter need to be created. They are

mainThread of JVM,
TCPSendThread,
UDPSendThread,
TCPReceiveThread,
UDPReceiveThread,
TimingCallBackThread.

However, only two threads run at the same time for message transmission.

TCPSend & UDPSend starts as soon as the send in the TTCN project executes. In order to ensure conformance and reliability of data reading from EDS, all the variables related to the reading status should be defined as volatile and the called methods should be defined as synchronized. We set a Boolean variable in the message struct, whose value can be accessed from the main thread. The sending thread keeps running before the sending of one message's binary stream is over. When the value of the Boolean variable becomes TRUE, the function run() of TCPSend & UDPSend will return immediately and then TCPReceive & UDPReceive will start to wait for the replying messages from SS.

Since the RRC Test is a signaling test process, only one message need to be sent to the destination at one movement. That means the adapter could work in a half-duplex way in sending message and checking the feedback from the User Equipment within a limited time. In the 3GPP specification, there are rules about the limitation of waiting time for each step of the process. For instance, the time limitations specified by the document 3GPP 36521-3 for some key steps of E-UTRAN cell re-selection in inter frequency case are shown in the Table 3 [10].

Table 3. Time Limitations of Key Steps for E-UTRAN Cell Reselection

Steps	**Timer value**	**Messages**
SS re-adjusts the cell-specific reference signal level of Cell 1	T1	-
Check: Does the UE send a ConnectionRequest (RRC) message on Cell 1 within the next 60s ?	60s	RRCConnectionRequest TP1
SS re-adjusts the cell-specific reference signal level of Cell 1 level	T2	RRCConnectionRequest TP2

The following code shows, in the test procedure, how long the timer of SS will wait for UE's response and how the SS will change the cell's power and the physical identity.

```
var float v_TimerValue := 60.0;
timer t_IdleMode_GenericTimer := tsc_IdleMode_GenericTimer;
if(f_EUTRA_RRC_RRCConnectionRequest_Check
(eutra_Cell1, v_TimerValue)) {
  //verdict fail UE sent RRCConnectionRequest message on Cell 1 within v_TimerValue
  f_EUTRA_SetVerdictFailOrInconc (__FILE__, __LINE__, "Test Case 6.1.2.2 Step 2");}
t_IdleMode_GenericTimer.start; //Step 4
  //Receive RRCConnectionRequest on Cell 1
  f_EUTRA_RRC_ConnectionRequest_Def(eutra_Cell1);
//Stop Idle Mode Geberic Timer
  t_IdleMode_GenericTimer.stop;
```

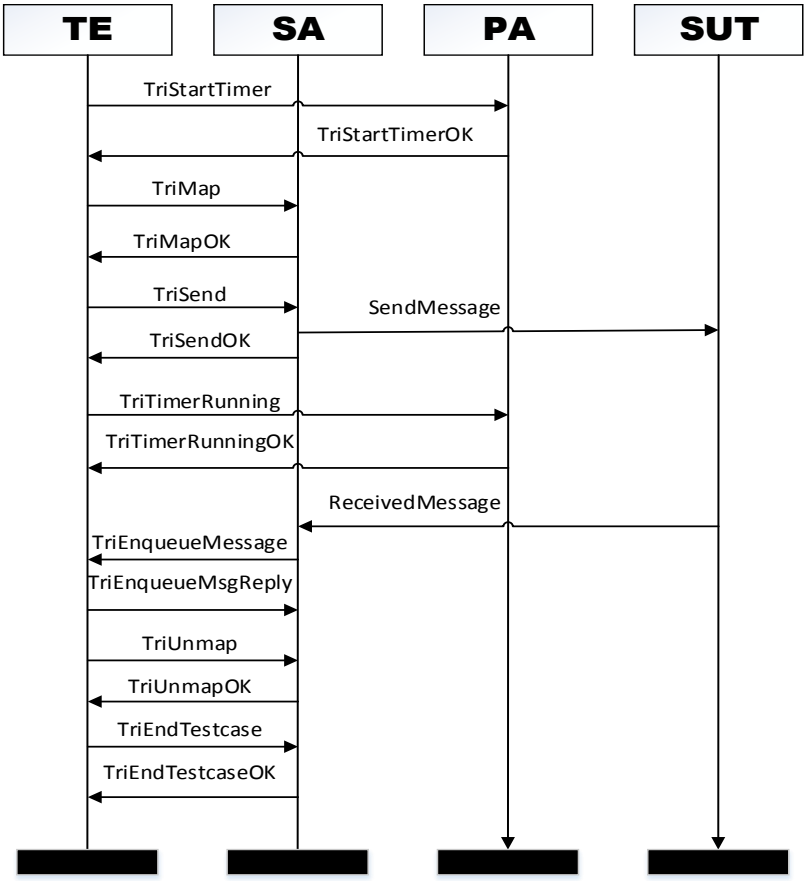

Fig. 7. Message sequence chart.

In the test suite of 3GPP 36523-3, two timers are defined at the beginning of the procedure. Among the steps in the test suite, the second step and the fourth step use the timers to set the verdict value.

The TRI mapping between TTCN-3 level and Java level make it possible for the adapter to read, start and stop the timers, as well as to know the timer's running state. According to the TRI definition, we can implement the timer mechanism, which is shown in Fig. 7.

```
package org.etsi.ttcn.tri;
public interface TriPlatformPA {
public TriStatus triPAReset();
public TriStatus triStartTimer(TriTimerId timerId,
TriTimerDuration timerDuration);
public TriStatus triStopTimer(TriTimerId timerId);
public TriStatus triReadTimer(TriTimerId timerId,
TriTimerDuration elapsedTime);
public TriStatus triTimerRunning(TriTimerId timerId,
TriBoolean running);}
TriStatus triStartTimer(TriTimerId timerId,
TriTimerDuration timerDuration){
Monitoring the feedback from SS, If PA receive the feedback frame
{       Switch(timerId){
Case 1: timer1.start(); Thread1.Sleep(timerDuration);
CaseN:timerN.strat();ThreadN.Sleep(timerDuration);}}}
public TriStatus triStopTimer(TriTimerId timerId){
Switch(timerId){
Case 1: timer1.stop();Thread1.interrupt();
```

Case 2: timer1.stop();Thread1.interrupt();
CaseN:timerN.stop();ThreadN.interrupt();}}

The adapter in the above code uses multi-threads to implement the message transmission. Before a message is sent from TE to SS, or in an inverse direction, the receiver should start the timer first and waits for the message for a few of seconds by sleeping the thread in time Duration seconds. If the message does not reach the destination during the time limitation, the timer stops and the corresponding thread executes the interrupt operation, and then the adapter immediately returns TRI_ERROR. Otherwise, it returns TRI_OK. With the time slice mechanism, the system cannot acquire very high performance in terms of real-time when many threads run at the same time, due to the characteristics of JRE (Java Runtime Environment). In our adapter, only a few threads run simultaneously and it has improved timing performance.

Fig. 8 shows four types of delay in data transmission procedure, i.e. SA delay caused by JAVA sockets, delay of network card, transmission and delay of SS action [11], [12]. Due to the delays, SS does not send data to the UE immediately when TTCN send executes at T0. In other words, they affect the accuracy of the timer. Since d4 in Fig. 8 is much larger than d1, d2 and d3, it is reduced remarkably in our adapter by monitoring the feedback of message transmission. After the adapter receives the feedback frame sent at T1, the timers start.

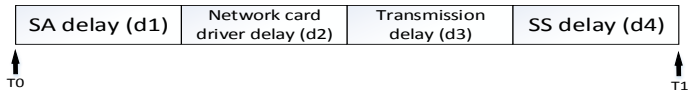

Fig. 8. Delays of the data transmission procedure.

As mentioned above, the TimingCallBackThread starts after one message sent from the TE. Considering the transmission rate, the callback of timing are based on UDP. At T1, SS receives the message and then returns the actual receiving time instantaneously. In this way, the timer can start at the exact moment.

In actual testing, host PC is connected by an Ethernet cable with SS that is an Integrated Testing Simulation Instrument for LTE. GFT (Graphical Format of TTCN-3) of signaling message flows in the TTworkbench shows that designed adapter can satisfy the requirements of the test procedure. It transmits each message successfully and TE has accurate timing by the use of the timing callback mechanism.

4. Conclusion

In this paper, the proposal is concerned with the design of a reasonable adapter model for the RRC test based on TTCN-3, which satisfies the requirements of ETSI specification. After elaborating its mechanism and main functionality in the general structure, we implement the model as running parts of the RRC test case.

Compared with the adapter plugins of some IDEs, the proposed adapter not only implements the functionalities that user needs, but also provides convenient usage for the testing developer. By using the real ports as few as possible, it has better performance because of system load reduction and has better control of the transmission timing. Furthermore the user can easily identify the signaling messages in the TTCN-3 level to enable the multi-protocol communication between TE and SS.

The further work may involves improving the performance and functionality of the adapter and using Realtime TTCN-3 [13] released in TTCN extensions to enhance the capability of synchronization of the multiple logical channels for the LTE RRC Test Suites.

Acknowledgment

The authors thank Dingliang Wang and Rui Huang for their assistance and appreciate the reviewers' valuable comments.

References

[1] Willcock, C., Deib, T., Tobies, S., Keil, S., Engler, F., & Schulz, S. (2005). An Introduction to TTCN-3. *West Sussex PO19 8SQ* (pp. 1-4). England: John Wiley & Sons Ltd.

[2] ETSI. (2011). Methods for Testing and Specification (MTS); the Testing and Test Control Notation Version 3, Part 5: TTCN-3 Runtime Interface (TRI). ETSI Standard, ETSI ES 201 873-5 V4.2.1.

[3] Cao, Y. Q., Nie, S., & Sun, Y. Y. (2012, Feb). The application of TTCN-3 in LTE terminal test. China Telecommunication Technology Labs, the Ministry of Industry and Information Technology, China.

[4] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); and Evolved Packet Core (EPC); User Equipment (UE) conformance specification; Part 3: Test Suites: 3GPP TS 36.523-3 V10.1.0.

[5] ETSI. (2010). Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3-Part 6: TTCN-3 Control Interface (TCI). *ETSI Standard*, ETSI ES 201 873-6 V4.2.1.

[6] Wikipedia encyclopedia. Transmission control protocol. Retrieved 20 January, 2014 from the Wikipedia website: http://en.wikipedia.org/wiki/Transmission_Control_Protocol

[7] Tan, J., Chen, X. S., & Du, M. (2012, Jan). An internet traffic identification approach based on GA and PSO-SVM. *Journal of Computers*, *7(1)*.

[8] Wikipedia encyclopedia. User datagram protocol. Retrieved 3 January, 2014 from the Wikipedia website: http://en.wikipedia.org/wiki/User_Datagram_Protocol

[9] He, K. F., Zhang, Z. J., Chen, J., & Li, Q. (2012, Dec). Ethernet solutions for communication of twin-arc high speed submerged arc welding equipment. *Journal of Computers*, *7(12)*.

[10] 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA);User Equipment (UE) conformance specification Radio; transmission and reception; Part 1: Conformance Testing: 3GPP TS 36.521-1 V10.2.0.

[11] Wang, M. C., Duan, J. T., & Zhang, B. (2013, Jan). TRI and TCI design of HINoc MAC testing software based on TTCN-3. The State Key Lab of Integrated Services Networks, Xidian University, The Northwest Institute of Nuclear Technology, China.

[12] Hu, W. M., Lu, Z. H., Liu, H. Z., & Jantsch, A. (2012, July). TPSS: A flexible hardware support for unicast and multicast on networks-on-chip. *Journal of Computers*, *7(7)*.

[13] ETSI. (2013). 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. ETSI Standard, ETSI ES 202 782 V1.1.1

Teng Huang received his B.E degree in communication engineering from Anqing Normal University, Anhui province, China. He is currently working toward the master's degree at Hefei University of Technology. His research interests include protocol conformance testing based on TTCN-3 and OFDM system.

Zhizhong Ding was born in Wuhu, China, in 1961. He received his B.E degree in radio communications from Nanjing University of Aeronautics and Astronautics, Nanjing, China, the master's degree in circuit and system from Hefei University of Technology (HFUT), Hefei, China, and the Ph.D. degree in information and communication engineering from University of Science and Technology of China. He currently is a professor with the Dept. of Communication Engineering and with the Institute of Communications and

Information Systems, HFUT. His research interests include wireless communications, network communications and information theory.

Younan Duan received his B.E degree in communication engineering from Huaibei Normal University, Anhui province, China. He is currently working toward the master's degree at Hefei University of Technology. His research interests include protocol conformance testing based on TTCN-3, and hardware implementation of 802.11p protocol.

Yin Chen received his B.S degree in information and computing science from Hefei University of Technology, Anhui province, China. He is currently working toward master's degree at Hefei University of Technology. His research interests include protocol conformance testing and wireless networks.