WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
*Wirel. Commun. Mob. Comput.* 2008; **8**:673–686
Published online 29 January 2007 in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/wcm.496

# NTP-PoCT: A conformance test tool for push-to-talk over cellular network

Yi-Bing Lin[1,2*,†], Chun-Chieh Wang[3], Chih-Hung Lu[3] and Miao-Ru Hsu[3]
[1]*Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan*
[2]*Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan*
[3]*Information and Communications Research Laboratories, Industrial Technology Research Institute, Taiwan*

## Summary

This paper describes a conformance test tool for push-to-talk over cellular network developed on an Open Mobile Alliance Service Interoperability Test Platform. Based on the TTCN-3 specifications, we show how PoC test cases can be efficiently implemented on the test platform. Copyright © 2007 John Wiley & Sons, Ltd.

KEY WORDS: conformance test; Open Mobile Alliance (OMA); Push-to-talk over Cellular (PoC); Testing and Test Control Notation version 3 (TTCN-3)

## 1. Introduction

Push-to-talk over Cellular (PoC) provides walkie–talkie like service in the cellular telecommunications network [9]. In this service, several predefined PoC group members participate in one PoC session. Since the PoC session is half-duplex, only one group member speaks at a time, and the others listen. Therefore, a user must ask for the permission to speak by pressing the push-to-talk button. In the PoC architecture, the PoC Server (Figure 1 (5)) provides the PoC session handling, media distribution, and talk burst control negotiation functionalities. The Presence Server (Figure 1 (3)) accepts, stores, and distributes presence information about the PoC Clients (Figure 1 (1)). The XML Document Management (XDM) Server (Figure 1 (2)) manages the databases to store PoC Clients' contact lists, pre-arranged groups, and other personal information. A PoC Client connects to the PoC Server and the Presence Server through Session Initiation Protocol (SIP)-based Core Network (Figure 1 (4)) such as IP Multimedia Core Network Subsystem (IMS) [4]. The PoC Client utilizes the XML Configuration Access Protocol (XCAP) to interact with the XDM Servers.

*Correspondence to: Yi-Bing Lin, Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan.
†E-mail: liny@csie.nctu.edu.tw
**Abbreviations used:** CD, Coding and Decoding; CH, Component Handling; EDS, Encoding/Decoding System; ETS, Executable Test Suite; IMPS, Instant Message and Presence Service; IOT, Interoperability Test; MMS, Multimedia Messaging Service; OMA, Open Mobile Alliance; PA, Platform Adapter; PoC, Push to talk over Cellular; SA, SUT Adapter; SIP, Session Initiation Protocol; SUT, Systems Under Test; TE, TTCN-3 Executable; TM, Test Management; TMC, Test Management and Control; TL, Test Logging; T3RTS, TTCN-3 Runtime System; TCI, TTCN-3 Control Interface; TRI, TTCN-3 Runtime Interface; TSI, Test System Interface; TTCN-3, Testing and Test Control Notation version 3; XCAP, XML Configuration Access Protocol; XDMS, XML Document Management Server; XML, Extensible Mark-up Language.

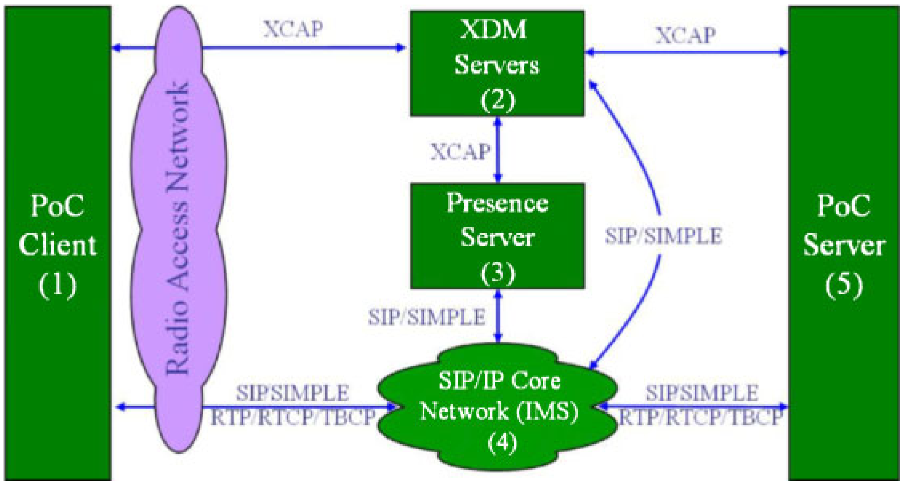


Fig. 1. The PoC architecture.

The Real Time Protocol (RTP) is utilized to deliver voice media between the PoC Client and Server.

Before the PoC application can be launched for service, it is essential to conduct testing to ensure that the PoC mechanism is correctly implemented. We have developed an Open Mobile Alliance (OMA) service interoperability test platform [8] based on the TTCN-3 specifications [1,2,5]. Several OMA Multimedia Messaging Service (MMS) [3,4] and Instant Message and Presence Service (IMPS) [6] test cases have been deployed in this platform.

Under Taiwan's National Telecommunication Program (NTP), we have implemented a PoC conformance test tool (called NTP–PoCT) on this platform. This test tool verifies the adherence to normative requirements described in the OMA PoC technical specifications [7]. The PoC service conformance tests include three types of tests. The Control Plane (CP) test cases verify SIP signals for PoC session control. The User Plane (UP) test cases verify the Talk Burst Control Protocol (TBCP) and the media distribution. The XDM test cases verify whether the XCAP messages used to access the XDM Server are correct. Figure 2 shows the conformance test environment for PoC. In this figure, NTP-PoCT (Figure 2 (4)) acts as the PoC network entities (including XDM Server, Presence Server and PoC Server) in all conformance test procedures. It connects to the handset under test (i.e., the PoC Client; see Figure 2 (2)) through the real mobile network or a network emulator such as Anritsu MD8470A (Figure 2 (3)) [10]. In the PoC CP and UP test procedures for the origination cases (where the PoC Client is the calling party), NTP-PoCT waits for the tested handset to initiate a PoC session. For the termination cases (where the PoC Client is the called party), NTP-PoCT initiates a PoC session to the tested handset, and waits for the responses form the handset. NTP-PoCT verifies if the sequence and the formats of the received messages are correct. In the XDM test procedure, NTP-PoCT receives and verifies the XCAP messages sent from the handset.

In this paper, we describe the NTP-PoCT architecture. Then we use examples to show how the PoC test cases are implemented in this test tool.

## 2. TTCN-3 Test System

NTP-PoCT is a Testing and Test Control Notation version 3 (TTCN-3) test system. This system manages PoC test execution, interprets or executes compiled TTCN-3 code, and implements proper communication with the systems under test (SUT). As illustrated in Figure 3, NTP-PoCT consists of the following parts.

The Test Management and Control (TMC; Figure 3 (1)) is responsible for test execution control and test event logging. The TTCN-3 Executable (TE; Figure 3 (2)) is responsible for the interpretation or execution of the PoC modules (to be described in Figure 6). The SUT Adapter (SA; Figure 3 (3)) adapts the TTCN-3 communication operations (of the TE) with the SUT (Figure 3 (5)). The Platform Adapter (PA; Figure 3 (4)) adapts the TE to a particular execution platform (i.e., Windows OS) by creating a single notion of time for a TTCN-3 test system (i.e., NTP-PoCT), and implementing external functions as well as timers.

Two interfaces are defined in a TTCN-3 test system. The TTCN-3 Control Interface (TCI; Figure 3 (a))

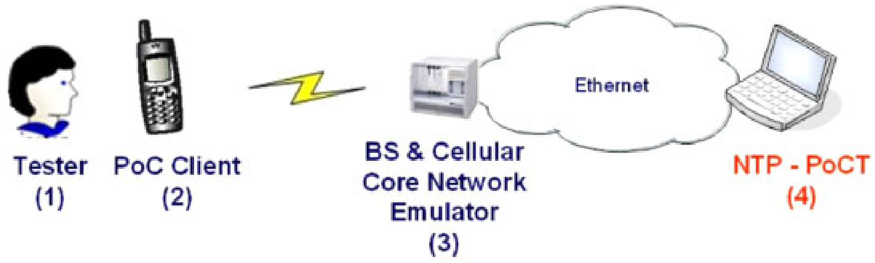


Fig. 2. Conformance test environment for PoC.

specifies the interface between TMC and TE. The TTCN-3 Runtime Interface (TRI; Figure 3 (b)) defines the interface between TE and SA/PA.

## 2.1. PoC Test Management and Control (TMC)

We briefly re-iterate the TMC descriptions in Reference [8] for the reader's benefit. The TMC consists of four entities [2]. The Test Management entity (TM; Figure 3 (6)) is responsible for overall management of the test system. After initiation the TM will invoke PoC TTCN-3 modules (e.g., `tc_CPOrg` module and PoC Control Plane Originating test cases).

The Test Logging entity (TL; Figure 3 (7)) is responsible for test component creation, start and termination, and data delivery to/from the SUT. The logging requests to the TL are posted externally from the TE or internally from the TM. Figure 4 shows a PoC graphical test log where the MTC (Figure 4 (1)) and the SUT (SYSTEM; Figure 4 (2)) is executing the 3GPP IMS registration test case. The PoC registration procedure is illustrated in Figure 5. The SUT first sends the SIP REGISTER (Figures 4 (3) and 5 (1)). NTP-PoCT verifies the REGISTER message (Figure 4 (4)), and replies 200 OK (Figure 4 (5)). The SUT then sends SIP PUBLISH (Figure 5 (2) and Figure 4 (6)) to update its PoC service setting. NTP-PoCT verifies the PUBLISH message (Figure 4 (7)), and replies 200 OK (Figure 4 (8)). Every 'match' box in Figure 4 indicates that the received SIP message matches a pass criteria described in the conformance test specification [7] and the final 'pass' box (Figure 4 (9)) indicates that this test case is passed for the SUT.

The External CoDecs (ECD; Figure 3 (8)) are invoked by the TE for encoding and decoding of TTCN-3 values into bitstrings to be sent to the SUT. The TE passes the TTCN-3 data to an appropriate encoder to produce the encoded data. The messages received from the SUT are passed to an appropriate decoder that translates the received data into TTCN-3 values. In NTP-PoCT, there are four external codecs: SIP, XDM, RTP, and RTCP. These codecs are implemented in the JAVA language, which can be easily ported to different test systems.

The Component Handling entity (CH; Figure 3 (9)) is responsible for distributing parallel test components. This entity is not implemented in the current version of NTP-PoCT.

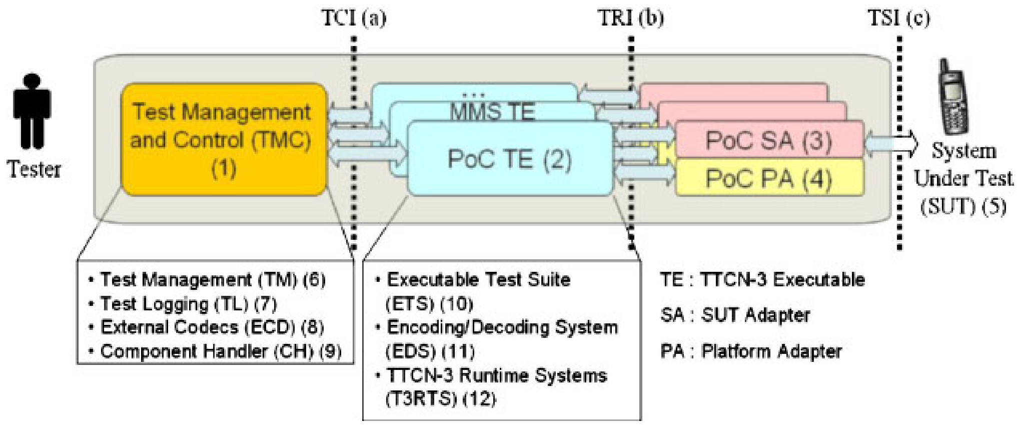

TCI (a)
TRI (b)
TSI (c)
Tester
Test Management and Control (TMC) (1)
MMS TE
PoC TE (2)
PoC SA (3)
PoC PA (4)
System Under Test (SUT) (5)
• Test Management (TM) (6)
• Test Logging (TL) (7)
• External Codecs (ECD) (8)
• Component Handler (CH) (9)
• Executable Test Suite (ETS) (10)
• Encoding/Decoding System (EDS) (11)
• TTCN-3 Runtime Systems (T3RTS) (12)
TE : TTCN-3 Executable
SA : SUT Adapter
PA : Platform Adapter

Fig. 3. The TTCN-3 system.

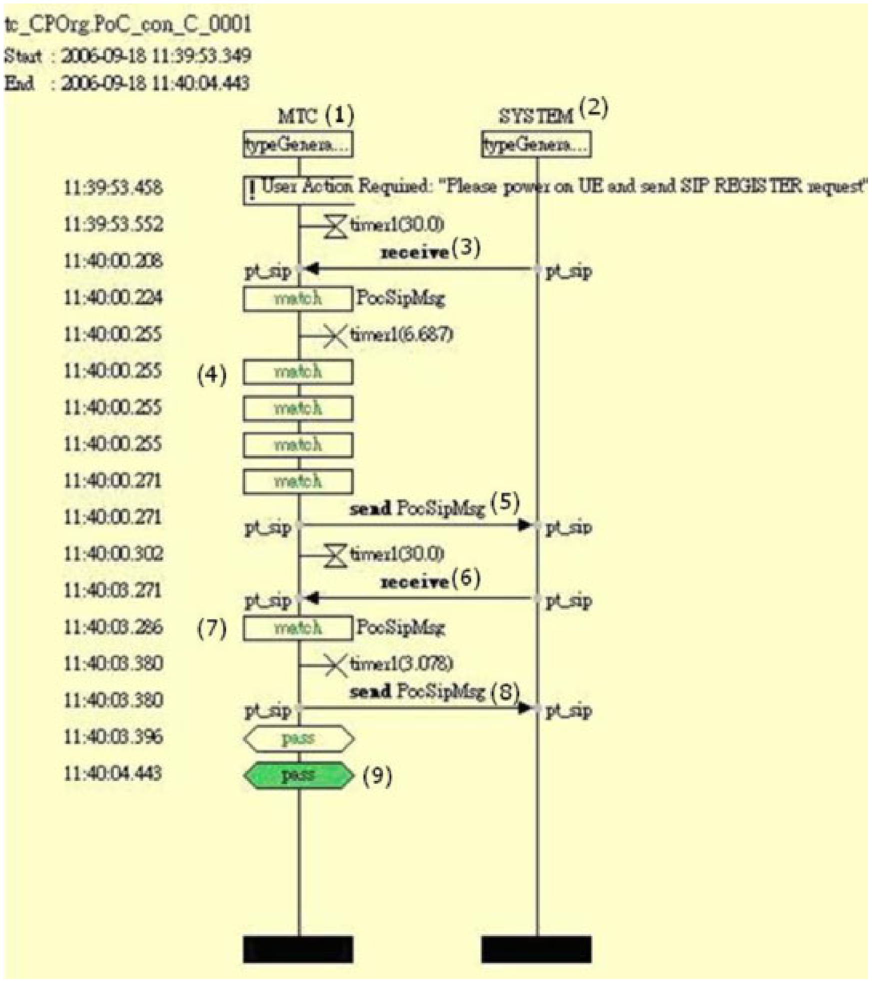


Fig. 4. PoC graphical test log for PoC registration.

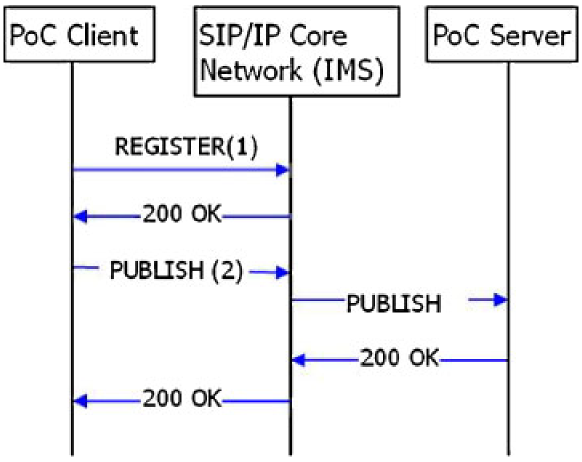


Fig. 5. PoC registration procedure.

## 2.2. PoC TTCN-3 Executable (TE)

The TE consists of three interacting entities to execute the TTCN3 test cases. The Encoding/Decoding System (EDS; Figure 3 (11)) is responsible for encoding and decoding of test data. NTP-PoCT does not utilize this built-in EDS. The TTCN-3 Runtime System (T3RTS; Figure 3 (12)) interacts with TM, SA and PA, and manages the Executable Test Suite (ETS; Figure 3 (10)) and the EDS entities. The T3RTS starts the execution of test cases in the ETS entity. Figure 6 illustrates the PoC ETS structure that classifies the PoC test cases into three groups: CP, UP, and XDM. For example, the Registration per 3GPP IMS test case `PoC_con_C_0001` is implemented in the `tc_CPOrg` module. This test case is invoked by the T3RTS when NTP-PoCT receives a REGISTER message.

| poc_Main | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| CP (Control Plane) | | UP (User Plane) | | | | | XDM | | |
| **tc_CPOrg** Originating Procedures Test Cases | **tc_CPTrm** Termination Procedures Test Cases | **tc_UPTrp** Transport Test Cases | **tc_UPTbc** Talk Burst Control Test Cases | **tc_UPMda** Media Control Test Cases | **tc_UPTmr** Timers Test Cases | **tc_UPTbcm** Talk Burst Control Messages Test Cases | **tc_XDMCore** XDM Core Procedures Test Cases | **tc_XDMPoC** PoC XDM Application Usages Test Cases | **tc_XDMShared** Shared XDM Application Usages Test Cases |

Fig. 6. PoC ETS structure.

## 2.3. PoC SUT Adapter (SA)

The SA adapts the communication between the TE and the SUT.

Through TRI, the TTCN-3 test component ports `pt_sip` (Figure 7 (5)), `pt_rtp` (Figure 7 (6)), `pt_rtcp` (Figure 7 (7)), and `pt_xdm` (Figure 7 (8)) are mapped to the following sockets in SA: `SIPSocket` (Figure 7 (9)), `RtpSocket` (Figure 7 (10)), `RtcpSocket` (Figure 7 (11)), and `XdmSocket` (Figure 7 (12)). The SA binds these sockets to Test System Interface (TSI; Figure 3 (c)) *Ports* `5060, 9000, 9001,` and `8080`, which are the default port numbers of SIP, RTP, RTCP, and XDM in NTP-PoCT. To correctly

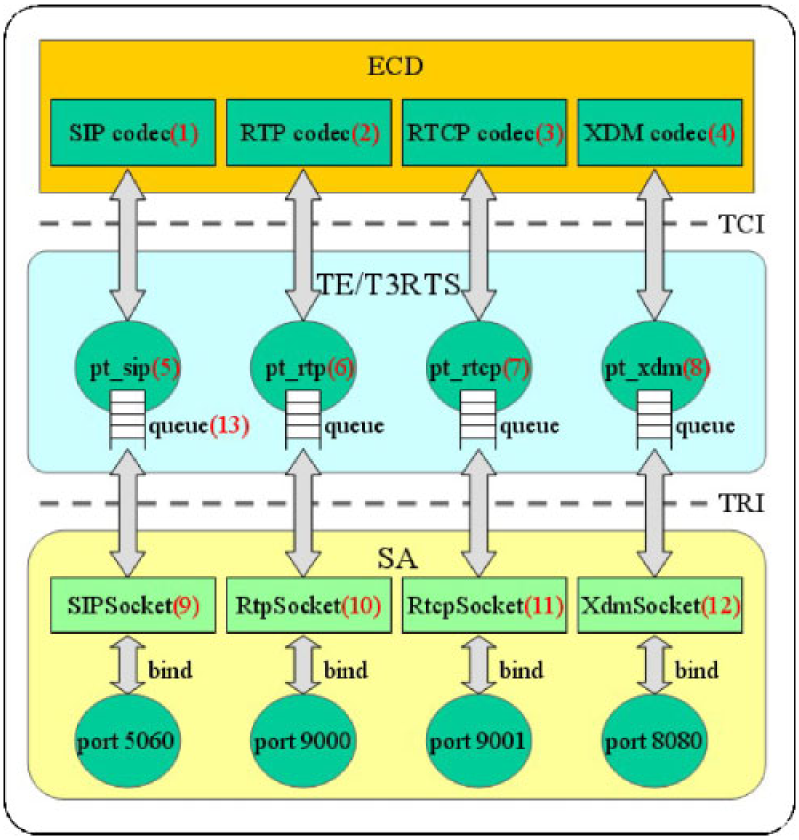


Fig. 7. Interaction between the SA and the TE in NTP-PoCT.

deliver SIP, RTP, RTCP, and XDM messages from and to the SUT, the TE calls the TRI functions that associate the component ports to the TSI ports, and uses the ECDs (Figure 7 (1)–(4)) for packet encoding/decoding.

The SA is responsible for propagating the PoC requests (e.g., sending a SIP response to the SUT through the `pt_sip`.send function) and the SUT action operations from the TE to the SUT, and notifying the TE of any received test events form the SUT (e.g., receiving a SIP request from the SUT through the `pt_sip`.receive function) by buffering them in the TE's port queues (Figure 7 (13)).

## 3. TTCN-3 Interfaces for PoC

This section describes the TTCN-3 control and runtime interfaces for PoC.

### 3.1. The TTCN-3 Control Interface (TCI) for PoC

The TCI interfaces the TE with the TMC through four sub-interfaces [2]. The TCI Test Management Interface (TCI-TM) supports operations for managing test execution, providing module parameters and external constants, and offering test event logs. The following TCI-TM program segment illustrates some NTP-PoCT parameters such as the user identity (Line 1) and the domain (Line 2) for the SUT.

```
group SysParameters  {
    modulepar  {
1.    charstring SIP_UID    :="Poc-User1";
2.    charstring DN_HOME    :="networkA.com";
      charstring DN_OTHER   :="networkB.com";
      float POC_SYS_WAIT    :=30.0;
    }
    with   {
3.      extension (SIP_UID)
          "Description: SIP public user ID for default user of the SUT";
4.      extension (DN_HOME)
          "Description: Domain Name of the home network";
5.      extension (DN_OTHER)
          "Description: Domain Name of another PoC Service provider";
6.      extension (POC_SYS_WAIT)
          "Description: Maximum time between two received packets";
    }
}
```

The extension function (Lines 3–6) shows the descriptions of the parameters to the tester and reminds the tester that these parameters should be set. They can be set dynamically by the tester and are provided to the test system as module parameters through the TCI-TM Interface.

The TCI Test Logging Interface (TCI-TL) includes operations for retrieving test execution information. The following TCI-TL program segment checks whether the SIP User-Agent header exists and logs the error if the header is missed in the received SIP request.

```
1.  if   ( ispresent(p_req.req.userAgent)  ) {
        ... //take appropriate action
    } else   {
2.       log("no User-Agent header is found in the SIP request");
         return RC_FAIL ;
    }
```

By invoking the `log` function in Line 2, the TL retrieves the error information through the TCI-TL interface and shows the error message in the test log.

The TCI Component Handling Interface (TCI-CH) consists of operations that implement the management and communication between the PoC test components in a distributed system. Since NTP-PoCT is a centralized system, this interface is not used.

The TCI Coding/Decoding Interface (TCI-CD) provides operations to access codecs. In NTP-PoCT, TCI-CD is implemented in a JAVA program `PoC_Codec.java`. In this program, the PoC `encode` operation is invoked by the TE to encode a TTCN value into a binary packet data unit based on the encoding rules. Parts of the program are listed in Figure 8. In this operation, if no encoding rule in Figure 8 (2)–(5) is matched, then Figure 8 (6) is executed for exception handling. In Figure 8 (3), an RTP message

```
public TriMessage encode(Value value) {
1.  try {
2.    if(value.getType().getName().equals("PocSipMsg")) {
        .... //SIP message encoding
      }
3.    else if(value.getType().getName().equals("PocRtpMsg")) {
3.1.     RecordValue rv = (RecordValue)value;
3.2.     RtpPacket pkt = new RtpPacket();
3.3.     int cc = ttcn.IntegerEncode(rv, "cc");
3.4.     pkt.setCc(cc);
3.5.     pkt.setMarker(ttcn.BooleanEncode(rv, "marker"));
3.6.     pkt.setPt(ttcn.IntegerEncode(rv, "pt"));
3.7.     .... //fill other parts of the RTP packet
3.8.     RtpProducer pdr = new RtpProducer(pkt);
3.9.     byte[] ret = new byte[pdr.getRtpBytes().length];
3.10.    System.arraycopy(pdr.getBytes(),0,ret,0,pdr.getRtpBytes().length);
3.11.    return new TriMessageImpl(ret);
     }
4.    else if(value.getType().getName().equals("PocRtcpMsg")) {
        .... //RTCP message encoding
     }
5.    else if(value.getType().getName().equals("PocXcapMsg")) {
        .... //XCAP message encoding
     }
6.  } catch(IOException e) {
     RB.tciTMProvided.tciError("Encoding error : "+e.getMessage());
   }
   return new TriMessageImpl(tmpbuf.toString().getBytes());
}
```

Fig. 8. PoC encode operation.

is encoded as follows. Figure 8 (3.1)–(3.7) constructs a `RtpPacket` structure from the given TTCN-3 value. Figure 8 (3.8)–(3.10) then generate a byte string from this `RtpPacket` structure.

The PoC `decode` operation invoked by the TE decodes a message according to the decoding rules and returns a TTCN-3 value. Parts of the program are listed in Figure 9. The message is decoded as a SIP, a RTP, a RTCP, or a XCAP message according to its type name, which matches the record type structure specified in the PoC TTCN-3 ETS. For example, in Figure 9 (2), a RTP message is decoded as follows. Figure 9 (2.1) retrieves an encoded byte string from the given message `message`. Figure 9 (2.2)–(2.4) uses `RtpParser` to check and parse the format of the given message. Then, Figure 9 (2.5)–(2.10) decodes the parsed RTP packet into a TTCN-3 value.

```
public Value decode(TriMessage message, Type decodingHypothesis) {
1. if (decodingHypothesis.getName().equals("PocSipMsg")) {
      .... //SIP message decoding
   }
2. else if (decodingHypothesis.getName().equals("PocRtpMsg") {
2.1.    byte[] encodedMsg = message.getEncodedMessage();
2.2.    RtpParser rtpParser = new RtpParser(encodedMsg);
2.3.    if (!rtpParser.parse())
2.4.      return null;
2.5.    RecordValue ret = (RecordValue)decodingHypothesis.newInstance();
2.6.    RtpPacket pkt = rtpParser.getRtpPacket();
2.7.    ttcn.IntegerDecode(returnValue, "cc", pkt.getCc());
2.8.    ttcn.BooleanDecode(returnValue, "marker", pkt.isMarker());
2.9.    ttcn.IntegerDecode(returnValue, "pt", pkt.getPt());
2.10.   .... //decode other parts of the RTP packet
2.11.   return returnValue
   }
3. else if (decodingHypothesis.getName().equals("PocRtcpMsg") {
       .... //RTCP message decoding
   }
4. else if (decodingHypothesis.getName().equals("PocXcapMsg")) {
       .... //XCAP message decoding
   }
   return null;
}
```

Fig. 9. The PoC `decode` operation.

## 3.2. The TTCN-3 Runtime Interface (TRI) for PoC

The TRI supports the communication of a TTCN-3 ETS with the SUT [1]. The NTP-PoCT TRI is implemented in a JAVA program `PoC_TestAdapter.java` consisting of the connection and communication operations shown in Figure 10.

Through the connection operations, the TSI ports are mapped to the test component ports. An example is `triMap` (Figure 10 (1)) called by the TE when it executes a TTCN-3 map operation. This operation instructs the SA to establish a dynamic connection to the SUT for the referenced TSI port.

TRI also supports the communication operations. An example is `triEnqueueMsg` (Figure 10 (2)) called by the SA after it has received a message from the SUT. This operation passes the message to the TE indicating the component where the TSI port is mapped. Another example is `triSend` (Figure 10 (4)) called by the TE when it executes a TTCN-3 unicast send operation on a component port mapped to a TSI port. This operation instructs the SA to send a message to the SUT. For PoC testing, four types of messages are sent by `triSend`: PocSipMsg for SIP, PocRtpMsg for RTP, PocRtcpMsg for RTCP, and PocXcapMsg for XCAP.

```
Public class PoC_TestAdapter extends TestAdapter {
1. public TriStatus triMap(final TriPortId compPortId, final TriPortId
                    tsiPortId) {
   CsaDef.triMap(compPortId, tsiPortId);
   if (tsiPortId.getPortName().equals("pt_sip")) {
     .... //bind SIPSocket to port 5060, and wait for SIP packets
2.    Cte.triEnqueueMsg(tsiPortId, new TriAddressImpl(new byte[]{}),
        compPortId.getComponent(), new TriMessageImpl(SIPdp.getData()));
   }
   .... //other map functions
   return new TriStatusImpl();
  }
3. public TriStatus triUnmap(TriPortId compPortId, TriPortId tsiPortId) {
   TriStatus mapStatus=CsaDef.triUnmap(compPortId, tsiPortId);
   .... //unmap functions
   return super.triUnmap(compPortId, tsiPortId);
  }
4. public TriStatus triSend(TriComponentId compId, TriPortId tsiPortId,
                     TriAddress address, TriMessage message) {
   if (tsiPortId.getPortName().equals("pt_sip")) {
     .... //send a SIP packet
   }
   else if (tsiPortId.getPortName().equals("pt_rtp")) {
     byte[] buf = message.getEncodedMessage();
     .... //error check
     DatagramPacket rtpPkt =
            new DatagramPacket(msg, msg.length,
                                  InetAddress.getByName(ip), port);
     RtpSocket.send(rtpPkt);
     return new TriStatusImpl();
   }
   .... //send a RTCP, XDM packet
  }
}
```

Fig. 10. PoC `Adapter` Program.

## 4. A PoC Conformance Test Scenario

We use a Control Plane (CP) test case in Figure 11 to show how PoC test suits are implemented. This test case verifies the PoC Client's SIP registration procedure through 3GPP IMS [4]. The TE first maps the four test component ports to the system server ports (Figure 11 (1); the `triMap` operation at the TRI is invoked). Then it pops up an action window (Figure 11 (2)) that instructs the tester (Figure 2 (1)) to send a SIP REGISTER message from the Client (Figure 2 (2)). The TE initializes the registration state machine at State 1 (Figure 12) and invokes function `f_mainloop` (Figure 11 (4)). In State 1 (WAITING for REGISTER), if a correct SIP REGISTER message is received from the SUT, the TE sends 200 OK to the SUT and changes to State 2 (WAITING for PUBLISH). If incorrect SIP REGISTER or other SIP messages are received at State 2, NTP-PoCT sets test result to fail and stops the state machine at State 3 (TEST FAIL). The TE sets verdict (Figure 11 (5)–(7)) according to the testing result returned from the `f_mainLoop` function. After the test result is returned, the TE removes the bindings between the four test component ports and the system server ports (Figure 11 (8)) using the `triUnmap` operation at the TRI.

In the `f_mainloop` function, the `t_mainLoop` timer starts at the PA (Figure 13 (1)) and the TE enters the main loop. When a SIP request from the SUT is received, the `pt_sip.receive` function is executed

```
testcase PoC_con_C_0001() runs on PoCComponent system PoCComponent
{
    //initialize state, timers, and necessary variables
1.  map(mtc:pt_sip,  system:pt_sip);
    map(mtc:pt_rtp,  system:pt_rtp);
    map(mtc:pt_rtcp, system:pt_rtcp);
    map(mtc:pt_xdm,  system:pt_xdm);
2.  f_action("Please power on Handset and send SIP REGISTER request");
3.  f_nxtStat({ST_1, omit}); // moves to State 1
4.  v_rc := f_mainLoop(1);

    if (v_rc == RC_PASS) {
5.      setverdict(pass); //moves to State 5
    } else if (v_rc == RC_TIMEOUT) {
6.      setverdict(inconc); // moves to State 4
    } else {
7.      setverdict(fail); // moves to State 3
    }
8.  unmap(mtc:pt_sip,  system:pt_sip);
    unmap(mtc:pt_rtp,  system:pt_rtp);
    unmap(mtc:pt_rtcp, system:pt_rtcp);
    unmap(mtc:pt_xdm,  system:pt_xdm);
}
```

Fig. 11. PoC CP test case.

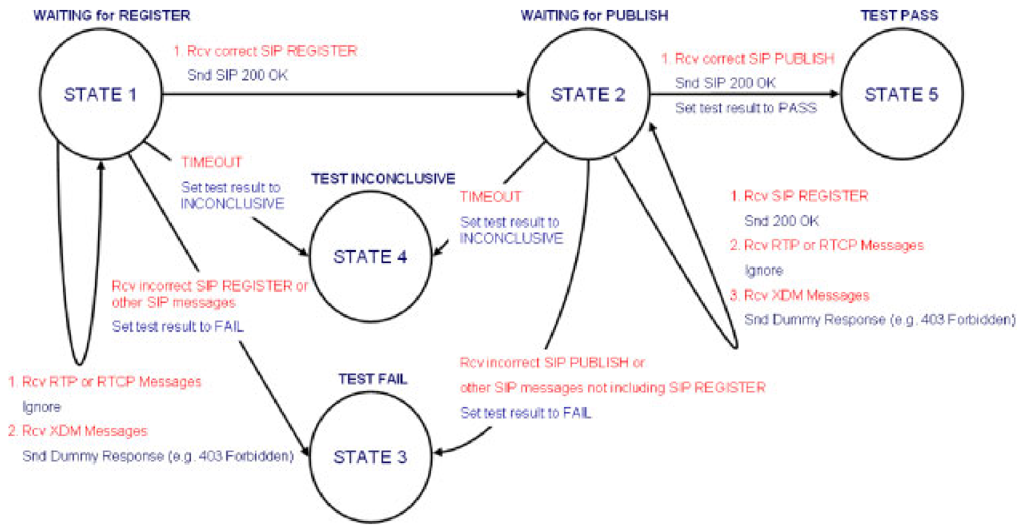
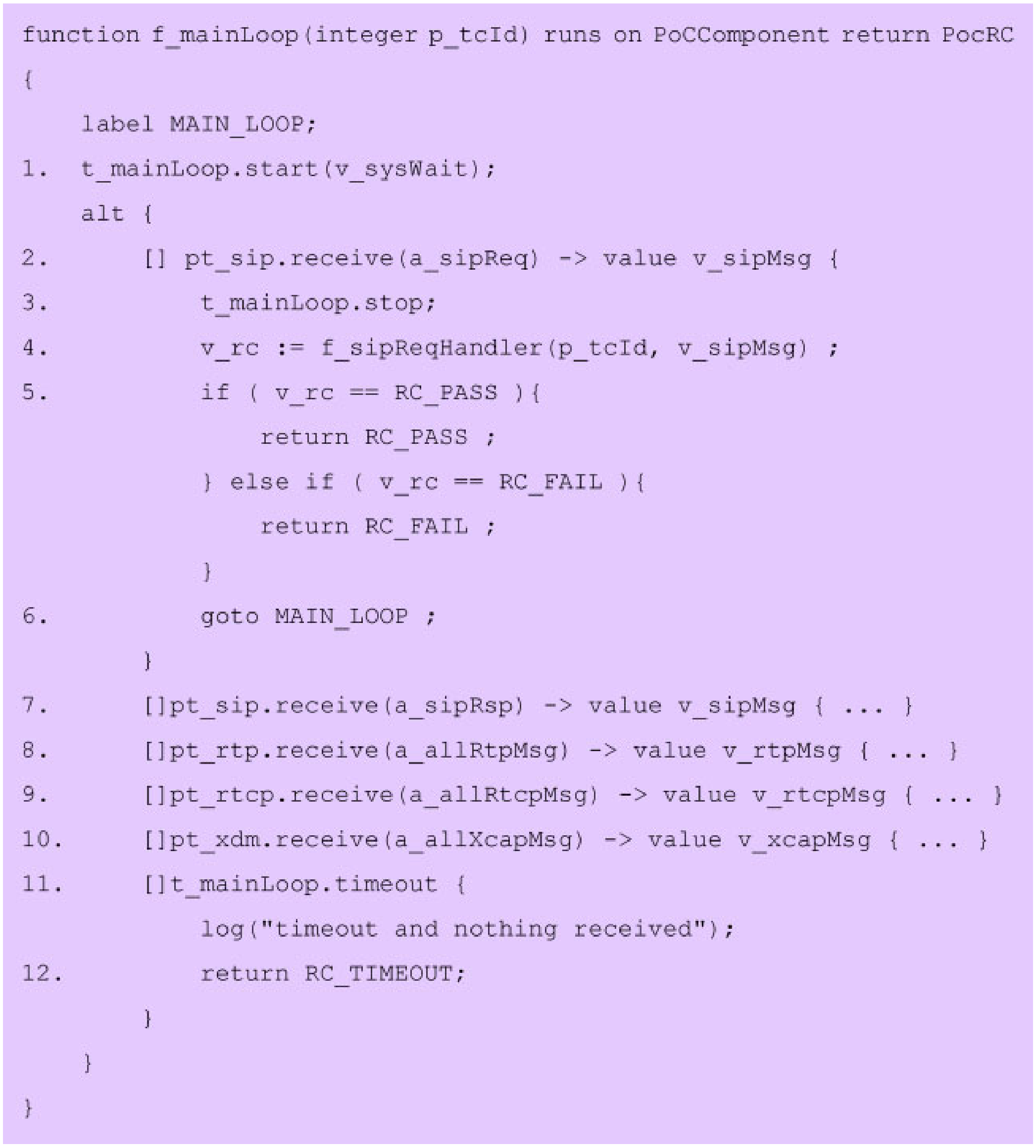

WAITING for REGISTER
STATE 1
1. Rcv correct SIP REGISTER
Snd SIP 200 OK
WAITING for PUBLISH
STATE 2
1. Rcv correct SIP PUBLISH
Snd SIP 200 OK
Set test result to PASS
TEST PASS
STATE 5
TIMEOUT
Set test result to INCONCLUSIVE
TEST INCONCLUSIVE
STATE 4
TIMEOUT
Set test result to INCONCLUSIVE
1. Rcv SIP REGISTER
Snd 200 OK
2. Rcv RTP or RTCP Messages
Ignore
3. Rcv XDM Messages
Snd Dummy Response (e.g. 403 Forbidden)
Rcv incorrect SIP REGISTER or other SIP messages
Set test result to FAIL
Rcv incorrect SIP PUBLISH or other SIP messages not including SIP REGISTER
Set test result to FAIL
1. Rcv RTP or RTCP Messages
Ignore
2. Rcv XDM Messages
Snd Dummy Response (e.g. 403 Forbidden)
TEST FAIL
STATE 3

Fig. 12. PoC registration test case state diagram.

(Figure 13 (2)), and the TE invokes the `decode` operation in Figure 9. After the message is decoded, the TE stops the `t_mainLoop` timer at the PA (Figure 13 (3)) and the `f_sipReqHandler` function (Figure 13 (4)) verifies whether the received message is correct. When the test result (`RC_PASS` or `RC_FAIL`)is returned (Figure 13 (5)), `f_mainloop` exits. Otherwise, the PA restarts the `t_mainLoop` timer again (Figure 13 (6)) and waits for another message from the SUT. If the TE does not receive any message from the Client after the `t_mainLoop` timer expires, the PA notifies the TE of this timeout event (Figure 13 (11)), and `f_mainloop` returns `RC_TIMEOUT`. The test case sets verdict to `inconc` (Figure 11 (6); indicating that an inconsistent exception occurs). In the `f_mainloop` function, `f_sipRspHandler`, `f_rtpHandler`, `f_rtcpHandler`, and `f_xdmHandler` handle the received SIP response messages (Figure 13 (7)), RTP messages (Figure 13 (8)), RTCP messages (Figure 13 (9)), and XDM messages (Figure 13 (10)), respectively.

Function `f_sipReqHandler` (Figure 13 (4)) invokes `f_sipRegHandler0001` (Figure 14) to handle the incoming SIP REGISTER request for the test case `PoC_con_C_0001`. This function first checks whether the SIP headers of the received REGISTER are correct according to the pass criteria [7]. For example, the `f_sipHdrContactChk` function (Figure 14 (2)) checks if the Contact header in the REGISTER contains the correct PoC feature-tag '`+g.poc.talkburst`.' If any pass criteria is not satisfied, the `f_sipRegHandler` function returns `RC_FAIL` to indicate the failure. If the received REGISTER message is correct, SIP 200 OK response is sent (Figure 14 (4)) to the PoC Client. This action triggers the Client to send the PUBLISH message (Figure 5 (2)). The SIP 200 OK response is generated by the `a_sipRspCmn` template illustrated below.

```
1.  template PocSipMsg a_sipRspCmn (PocSipMsg p_req, integer p_sc)  :=
    {
2.    srcIp   := p_req.dstIp,  // source address is the destination
      srcPort := p_req.dstPort, // address of the request message
      dstIp   := p_req.srcIp,  // destination address is the source
      dstPort := p_req.srcPort, // address of the request message
3.    msgType := PSMT_RSP, // SIP message type: SIP response
      rsp     :=  {
4.        sc       := p_sc,
5.        callId   := p_req.req.callId,
6.        fromHdr  := p_req.req.fromHdr,
7.        toHdr    := p_req.req.toHdr,
8.        cseq     := p_req.req.cseq,
9.        via      := p_req.req.via
      }
};
```

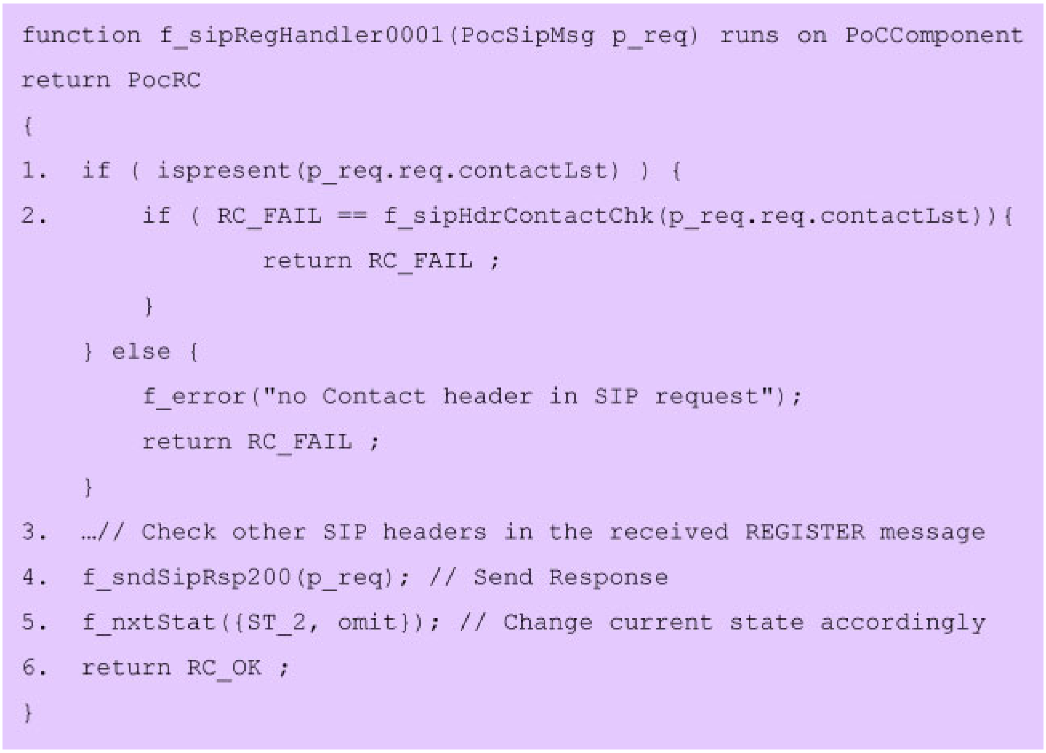


```
function f_mainLoop(integer p_tcId) runs on PoCComponent return PocRC
{
    label MAIN_LOOP;
1.  t_mainLoop.start(v_sysWait);
    alt {
2.      [] pt_sip.receive(a_sipReq) -> value v_sipMsg {
3.          t_mainLoop.stop;
4.          v_rc := f_sipReqHandler(p_tcId, v_sipMsg) ;
5.          if ( v_rc == RC_PASS ){
                return RC_PASS ;
            } else if ( v_rc == RC_FAIL ){
                return RC_FAIL ;
            }
6.          goto MAIN_LOOP ;
        }
7.      []pt_sip.receive(a_sipRsp) -> value v_sipMsg { ... }
8.      []pt_rtp.receive(a_allRtpMsg) -> value v_rtpMsg { ... }
9.      []pt_rtcp.receive(a_allRtcpMsg) -> value v_rtcpMsg { ... }
10.     []pt_xdm.receive(a_allXcapMsg) -> value v_xcapMsg { ... }
11.     []t_mainLoop.timeout {
            log("timeout and nothing received");
12.         return RC_TIMEOUT;
        }
    }
}
```

Fig. 13. The `f_mainloop` function.

For 200 OK, the input parameter `p_sc` of the `a_sipRspCmn` template is set to 200 (Lines 1 and 4) and some SIP headers (Call-ID, From, To, CSeq, and Via headers) should be set to the identical values as those headers in the corresponding SIP request (Lines 5–9).

After the SIP 200 OK response is sent in Figure 14 (4), the registration state machine moves to State 2 (WAITING for PUBLISH; Figures 12 and 14 (5)). When NTP-PoCT receives the correct PUBLISH message in State 2, the test is passed (Figure 12), and the state machine moves to State 5 (TEST PASS).



```
function f_sipRegHandler0001(PocSipMsg p_req) runs on PoCComponent
return PocRC
{
1.  if ( ispresent(p_req.req.contactLst) ) {
2.      if ( RC_FAIL == f_sipHdrContactChk(p_req.req.contactLst)){
                return RC_FAIL ;
        }
    } else {
        f_error("no Contact header in SIP request");
        return RC_FAIL ;
    }
3.  …// Check other SIP headers in the received REGISTER message
4.  f_sndSipRsp200(p_req); // Send Response
5.  f_nxtStat({ST_2, omit}); // Change current state accordingly
6.  return RC_OK ;
}
```

Fig. 14. The `f_sipRegHandler0001` function.

## 5. Conclusions

This paper described the architecture and operations of NTP-PoCT, a PoC test system developed based on the TTCN-3 specifications. This system has been jointly developed by the National Telecommunication Program (NTP) and the Industrial Technology Research Institute (ITRI) in Taiwan. We used the PoC Control Plane procedures to illustrate how conformance tests can be implemented and conducted in NTP-PoCT. These test cases are conformed to the OMA Enabler Test Specification (Conformance) for PoC [7]. Currently, 52 PoC tests cases have been developed in NTP-PoCT.

## References

1. ETSI. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 5: TTCN-3 Runtime Interface (TRI), ETSI ES 201 873–5 V3.1.1, 2005.
2. ETSI. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 6: TTCN-3 Control Interface (TCI), ETSI ES 201 873–6, V3.1.1, 2005.
3. Open Mobile Alliance, Enabler Test Specification for MMS 1.2, OMA-IOP-ETS-MMS-V1_2-20040409-A, 2004.
4. Lin Y-B, Pang A-C. *Wireless and Mobile All-IP Networks*, Wiley, 2005.
5. ETSI. Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language, ES 201 873–1, V3.1.1, 2005.
6. Open Mobile Alliance, Enabler Test Specification for OMA IMPS CSP, OMA-ETS-IMPS_CSP-V1_2_1-20051115-A, 2005.
7. Open Mobile Alliance, Enabler Test Specification (Conformance) for PoC, OMA-ETS-PoC_CON-V1_0-20051020-A, 2005.
8. Lin Y-B, Liang C-F, Chen K-H, Liao H-Y. NTP-SIOT: A Test Tool for Advanced Mobile Services, IEEE Network, November/December 2006, pp. 2–7.
9. Open Mobile Alliance, Push to talk Over Cellular Architecture, OMA-AD-PoC -V1_0-20060609-A, 2006.
10. Anritsu Corporation, MD8470A Signaling Tester Product Introduction, http://www.us.anritsu.com/products/ARO/North/Eng/showProd.aspx?ID=659

## Authors' Biographies

**Yi-Bing Lin** is Chair Professor and Vice President of Research and Development, National Chiao Tung University. His current research interests include mobile computing and cellular telecommunications services. Dr. Lin has published over 200 journal articles and more than 200 conference papers. He is the co-author of the books *Wireless and Mobile Network*

*Architecture* (with Imrich Chlamtac; published by Wiley, 2001) and *Wireless and Mobile All-IP Networks* (with Ai-Chun Pang; published by Wiley, 2005).

He is an IEEE Fellow, ACM Fellow, AAAS Fellow, and IEE Fellow.

**Chun-Chieh Wang** is currently a Software Engineer at the Department of Mobile Internet, Information and Communications Research Laboratories, Industrial Technology Research Institute, R.O.C. He received his B.S. and M.S. degrees in Computer Science and Information Engineering from National Chiao Tung University in 2000 and 2002, respectively. His current research interests include wireless Internet protocols and wireless Internet applications.

**Chih-Hung Lu** is a Software Engineer of Information and Communications Research Labs, Industrial Technology Research Institute. His specialties are TTCN3 and JAVA language. His current major job is writing OMA conformance testcases including OMA IMPS, PoC, and DRM.

**Miao-Ru Hsu** is a Section Manager at the Department of Mobile Internet, Information and Communications Research Laboratories, Industrial Technology Research Institute. She got her M.S. degree in Computer Science and Engineering from Yuan Ze University. Her current research interests include mobile services and wireless Internet protocols.