國立交通大學

資訊學院資訊科技（IT）產業研發碩士班

碩 士 論 文

NTP-DRMT：測試行動設備中 OMA 數位使用權管理功能的符合性與互通性之測試工具

NTP-DRMT：A Conformance and Interoperability Test Tool for OMA Digital Rights Management of Mobile Devices

研 究 生：黃亭愷

指導教授：林一平　教授

陳懷恩　教授

中 華 民 國 九 十 七 年 七 月

NTP-DRMT：測試行動設備中 OMA 數位使用權管理功能的符合性與互通性之測試工具

NTP-DRMT：A Conformance and Interoperability Test Tool for OMA Digital Rights Management of Mobile Devices

研 究 生：黃亭愷	Student：Ting-Kai Huang
指導教授：林一平	Advisor：Yi-Bing Lin
陳懷恩	Whai-En Chen

國 立 交 通 大 學
資訊學院資訊科技（IT）產業研發碩士班
碩 士 論 文

A Thesis
Submitted to College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Industrial Technology R & D Master Program on
Computer Science and Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

NTP-DRMT：測試行動設備中 OMA 數位使用權管理功能的符合性與互通性之測試工具

學生：黃亭愷　　　　　　　　　　　　　　　　　　指導教授：林一平
陳懷恩

國立交通大學資訊學院產業研發碩士班

中文摘要

本論文開發一個數位使用權管理(Digital Rights Management)的符合性與互通性之測試工具，測試待測物(如:手機)是否符合開放行動通訊聯盟(Open Mobile Alliance)的服務互通性測試之標準。數位使用權管理是一種防護(或保護)的機制，發行者利用使用權來管理被使用者取得的多媒體資料。使用者必須依照使用權物件(Rights Objects)的規則來存取被數位使用權所保護的文件(DRM Content)。當行動數位裝置使用 DRM 的應用服務之前，我們必須測試此 DRM 應用服務的開發是正確的依照 OMA 所訂定的規格。我們將展示在 Testing and Test Control Notation version 3 (TTCN-3)下如何有效率的開發 DRM 的測試例子。

NTP-DRMT：A Conformance and Interoperability Test Tool for OMA Digital Rights Management of Mobile Devices

student：Ting-Kai Huang

Advisors：Dr. Yi-Bing Lin
Dr. Whai-En Chen

Industrial Technology R & D Master Program of
Computer Science College
National Chiao Tung University

Abstract

This thesis describes a conformance and interoperability test tool for Digital Rights Management (DRM) developed on an Open Mobile Alliance (OMA) service interoperability test platform. DRM is a security (protection) mechanism that allows a content issuer to manage the media objects to be delivered to the users with copyright protection. In DRM, the users then access DRM Content (i.e., the protected media objects) according to the Rights Objects. Before a DRM application is launched for a mobile device, it is essential to conduct testing to ensure that the DRM mechanism is correctly implemented. Based on the Testing and Test Control Notation version 3 (TTCN-3) specifications, we show how DRM test cases can be efficiently implemented.

Acknowledgment

I would especially like to thank my advisors, Prof. Yi-Bing Lin and Prof. Whai-En Chen. Without their supervision and perspicacious advices, I cannot complete this thesis. I would also like to thank my colleagues in Laboratory 117.

I would also like to thank Chun-Chieh Wang and Pai-Tsan Huang for their help in developing NTP-DRMT.

I would also like to express my thanks to my friends. They encourage me and make me feel confidence to complete this thesie.

Finally, I want to thank my dear parents and my sister for their unfailing love and firmly support in these years.

Contents

List of Figures

Chapter 1 Introduction

Open Mobile Alliance (OMA) *Digital Rights Management* (DRM) distributes the media objects (e.g., a movie, an MP3 music file, and so on) with secured business models that can control the usage of the content and manage the content lifecycle [1, 5, 6]. When a user attempts to access the content, he/she may choose a free preview version or a complete version with charge.

1.1 OMA DRM

OMA DRM allows a content issuer to manage DRM Content [2] to be delivered to the users with copyright protection. In the DRM architecture [3], the user can access DRM Content through a DRM agent (Figure 1 ①), where the DRM agent is installed in the mobile device to manage the usage of DRM Content. The content issuer (Figure 1 ③) is responsible for packaging the media objects into DRM Content and delivering DRM content to the users. A rights issuer (RI; Figure 1 ④) is responsible for producing the Rights Objects [4]. Each Rights Object is associated with a piece of DRM Content.

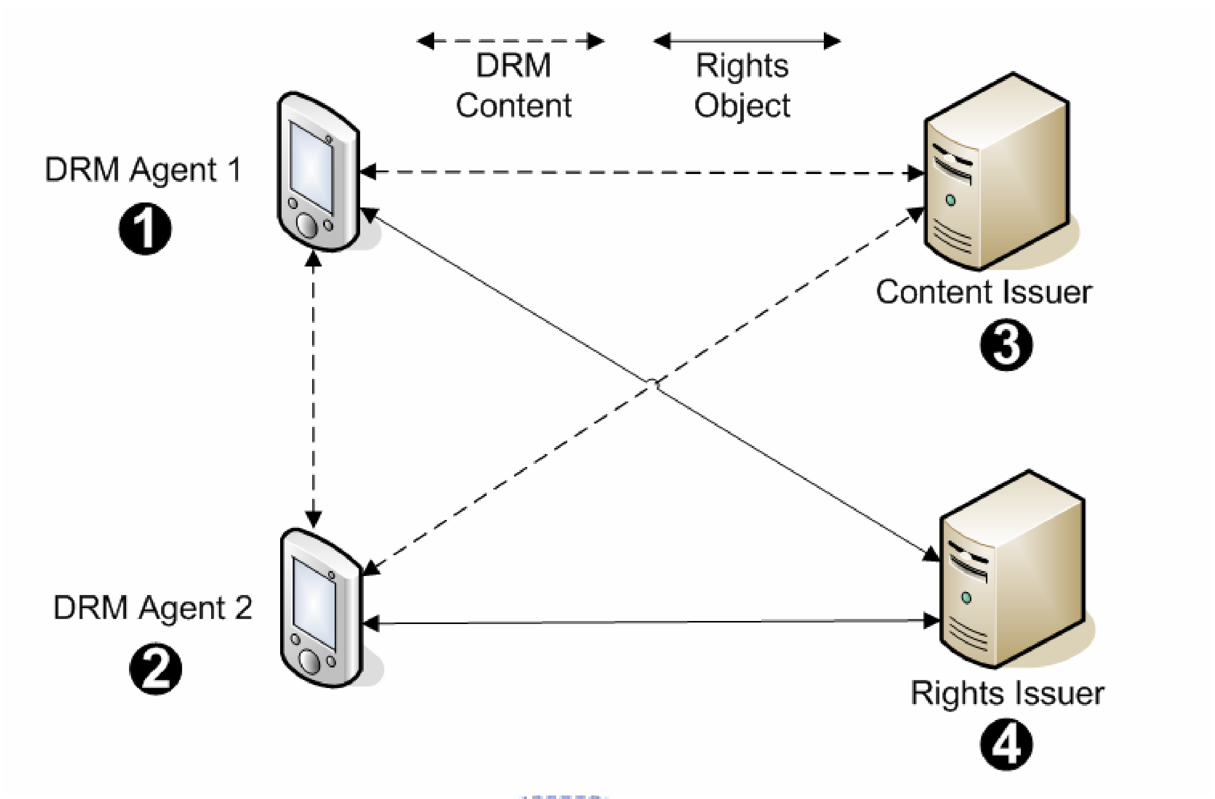


Figure 1. The DRM architecture.

DRM Content is accessed by the DRM agent under the control of the Rights Objects. A Rights Object specifies permissions, constraints and other features, which indicates how DRM Content can be used. Consider an example where DRM Content consists of a movie and two pictures, and DRM Agent 2 (Figure 1 ②) attempts to play the movie for two times and display the pictures for unlimited times. DRM Agent 2 may download DRM Content from the content issuer or another DRM agent (e.g., DRM Agent 1 in Figure 1 ①) who previously downloaded DRM Content through the DRM mechanism. DRM Agent 2 must utilize the *Rights Object Acquisition Protocol* (ROAP) to acquire a Rights Object from the rights issuer, and then accesses DRM Content according to the Rights Object. The format of the Rights Object in this example is illustrated in Figure 2. The asset field (Figure 2 ①) specifies the identity, the hash value and the key information of DRM Content. The hash value ensures the integrity of DRM Content. The key information identifies the encryption method used to encrypt the *content encryption key* (CEK), and contains the Base64-encoded value of the encrypted CEK. The permission field (Figure 2 ②) specifies the permissions (e.g., to play the movie) and the constraints (e.g., to play the movie two times). If the permission field only specifies the permissions without any constraint (Figure 2 ③), it means that the DRM agent is granted unlimited display rights.

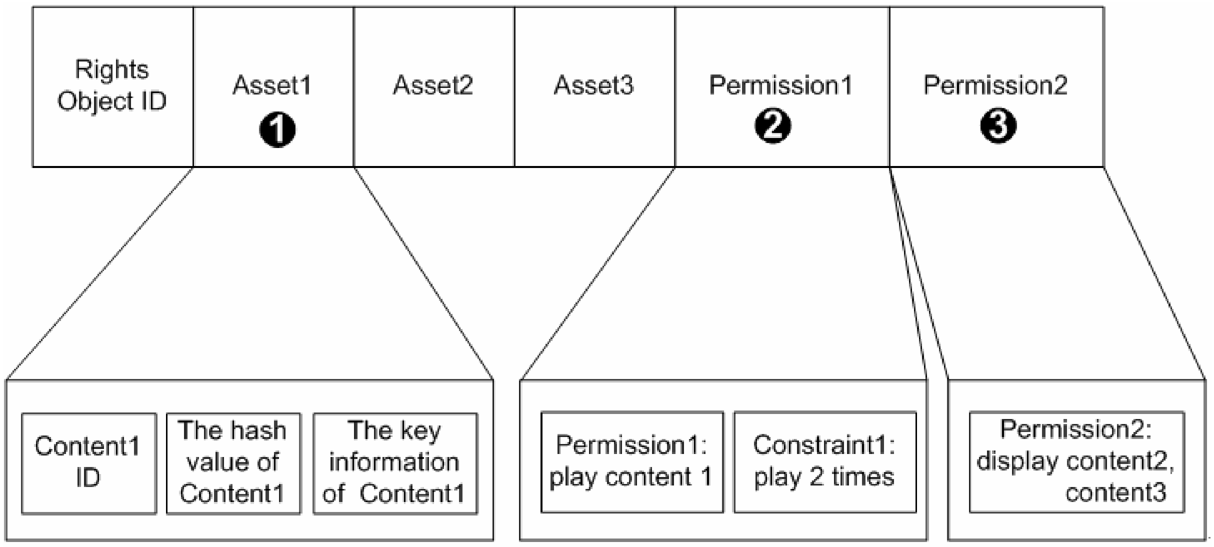

Rights Object ID
Asset1 ❶
Asset2
Asset3
Permission1 ❷
Permission2 ❸
Content1 ID
The hash value of Content1
The key information of Content1
Permission1: play content 1
Constraint1: play 2 times
Permission2: display content2, content3

Figure 2. The Rights Object format.

1.2 NTP-DRMT

Before a DRM application can be launched for service, it is essential to conduct testing to ensure that the DRM mechanism is correctly implemented in a mobile device. Although such tests can be done manually, it is more appropriate to automatically validate the DRM mechanism through a test tool. Under Taiwan's *National Telecommunications Program* (NTP), we have developed an OMA service interoperability test platform [16] based on the *Testing and Test Control Notation version 3* (TTCN-3) specifications [9, 10, 11, 12, 13, 14, 15]. Several OMA *Push-to-talk over Cellular* (PoC) test cases were deployed on this platform [17].

In this thesis, we implement a *DRM conformance and interoperability test tool* (called NTP–DRMT) on this platform. NTP-DRMT verifies the adherence to normative requirements described in the OMA DRM technical specifications [7, 8]. The conformance test cases verify whether the DRM agent follows the standard regulations described in [7]. The interoperability test cases verify whether the *system under test* (SUT; Figure 3 ②), which is implemented based on the specifications, works satisfactorily in various commercial mobile telecommunication networks [8]. Figure 3 shows the conformance and interoperability test environment for DRM. In this figure, NTP-DRMT (Figure 3 ④) acts as the DRM network entities (including the content issuer and the rights issuer) in all test procedures. It communicates with the SUT through a real cellular network or a cellular network emulator (Figure 3 ③) such as Anritsu MD8470A [18]. In the DRM conformance test procedures, NTP-DRMT waits for the SUT to request a Rights Object or DRM Content. This request is issued by a tester (Figure 3 ①). After the SUT receives the response, the tester verifies if the results reported by the SUT are correct. The tester then reports the verdict (e.g., pass or fail) to NTP-DRMT. In the DRM interoperability test procedures, NTP-DRMT waits for the SUT to request DRM Content and the associated Rights Objects through different commercial mobile networks. The tester then verifies if the SUT is capable of executing the interoperability test cases and reports the verdict to NTP-DRMT.

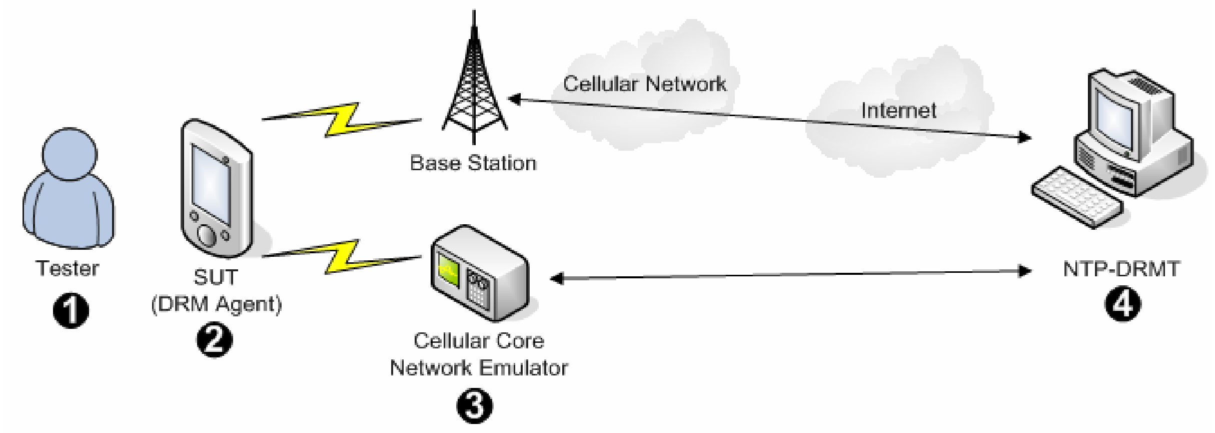


Figure 3. Conformance and interoperability test environment for OMA DRM.

1.3 Test Procedure for the DRM Registration

The DRM registration procedure is illustrated in Figure 4. When a user attempts to access new DRM Content, the DRM agent first sends an HTTP_GET (Figure 4 ①) message to the rights issuer to request the Rights Object. The rights issuer verifies the HTTP_GET message. Then the rights issuer replies a ROAP Trigger message (Figure 4 ②). This message is used to trigger the ROAP message exchange. The DRM agent then sends a Device Hello message (Figure 4 ③) to provide device information (such as the Device ID and the ROAP version) and initiate the registration procedure. The rights issuer verifies the Device Hello message and replies a RI Hello message (Figure 4 ④). The RI Hello message provides RI preferences and decisions according to the values provided by the DRM agent. The DRM agent then sends a Register Request message (Figure 4 ⑤). The rights issuer verifies the Register Request message and checks if the session ID, the device nonce, the request time and the signature in the Register Request message are correct. Then the rights issuer replies a Register Respose message (Figure 4 ⑥). Upon the receipt of this message, the registration procedure is complete, and the DRM agent can acquire the Rights Object.

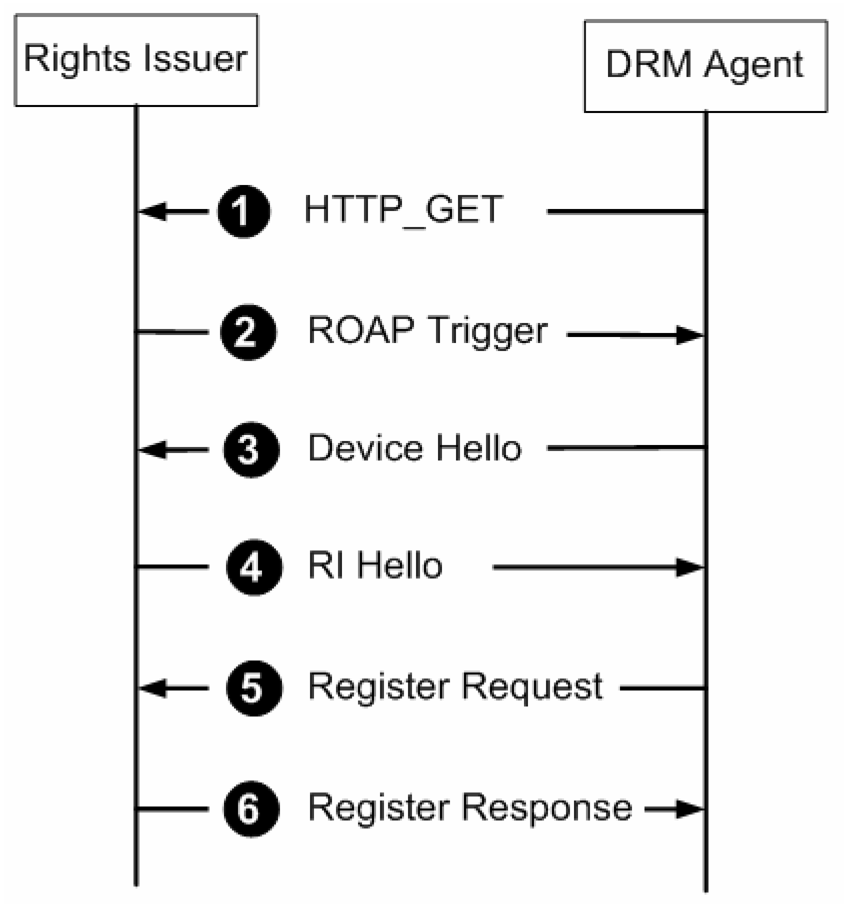


Figure 4. The test case for DRM registration procedure.

In the remainder of this thesis, we will describe the design and implementation of NTP-DRMT. Then we use examples to show how the DRM test cases are developed in this test tool.

Chapter 2 TTCN-3 Based Test System

The test system NTP-DRMT is implemented based on TTCN-3. This system manages DRM test execution, interprets/executes compiled TTCN-3 code, and implements proper communications with the SUT. As illustrated in Figure 5, NTP-DRMT consists of the following parts.

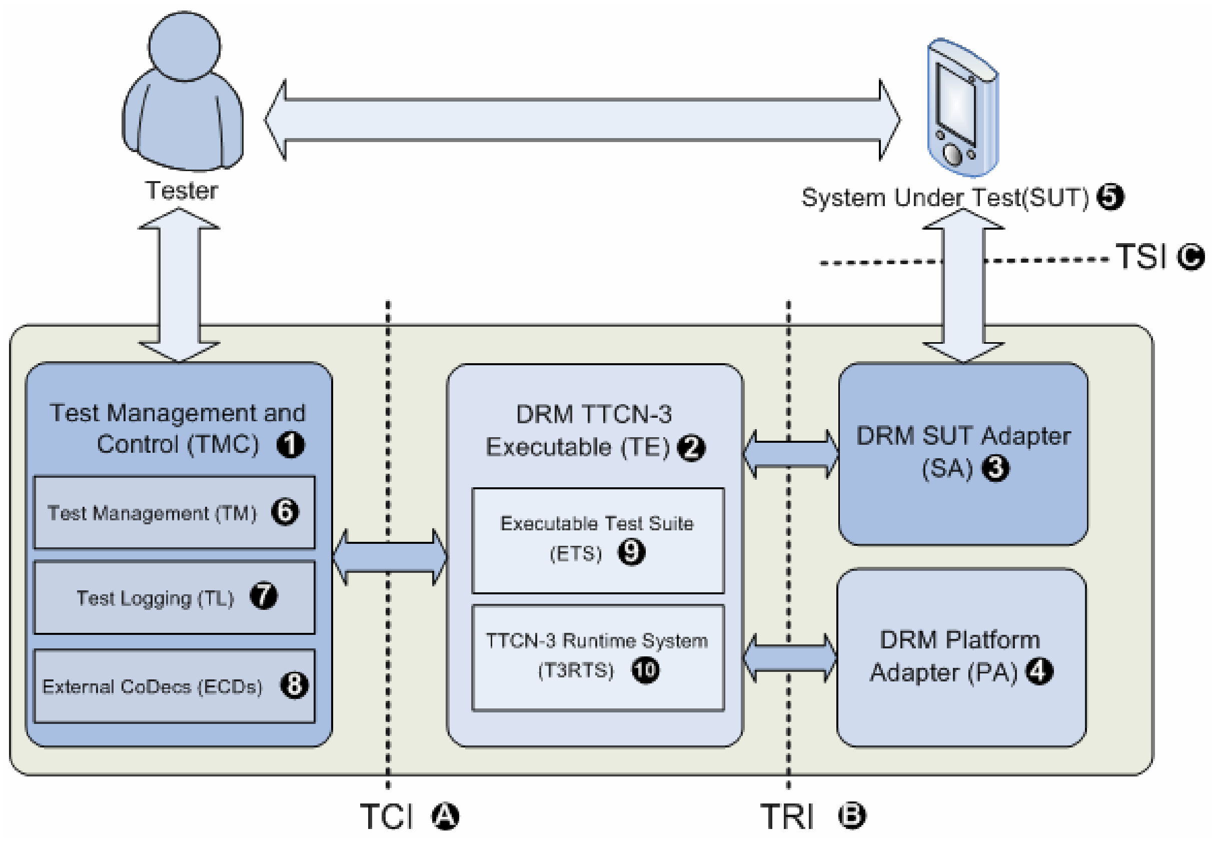

Tester
System Under Test(SUT) ❺
TSI Ⓒ
Test Management and Control (TMC) ❶
Test Management (TM) ❻
Test Logging (TL) ❼
External CoDecs (ECDs) ❽
DRM TTCN-3 Executable (TE) ❷
Executable Test Suite (ETS) ❾
TTCN-3 Runtime System (T3RTS) ❿
DRM SUT Adapter (SA) ❸
DRM Platform Adapter (PA) ❹
TCI Ⓐ
TRI Ⓑ

Figure 5. TTCN-3 based NTP-DRMT.

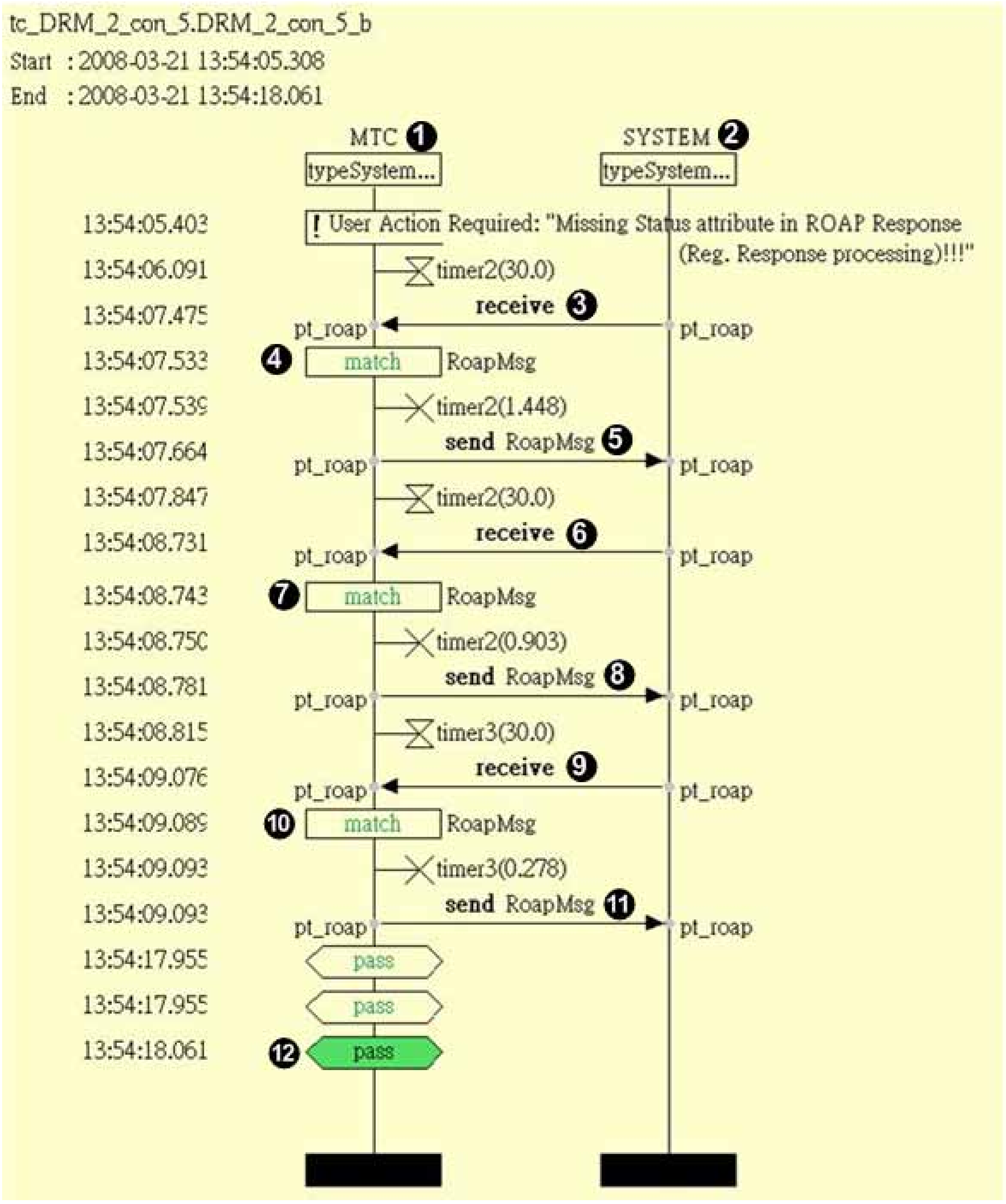
The *Test Management and Control* (TMC; Figure 5 ①) is responsible for the test execution control and the test event logging. The *TTCN-3 Executable* (TE; Figure 5 ②) is responsible for the interpretation or the execution of the DRM modules (i.e., abstract test suites). The *SUT*

Adapter (SA; Figure 5 ③) adapts the TTCN-3 communication operations between the TE and the SUT (Figure 5 ⑤). The *Platform Adapter* (PA; Figure 5 ④) adapts the TE to an operating system (e.g., Microsoft Windows XP) by creating a single notion of time for NTP-DRMT and implementing the external functions as well as timers.

Two interfaces are defined inside NTP-DRMT. The *TTCN-3 Control Interface* (TCI; Figure 5 Ⓐ) specifies the interface between the TMC and the TE. The *TTCN-3 Runtime Interface* (TRI; Figure 5 Ⓑ) defines the interface between the TE and the SA/PA. A third interface *Test System Interface* (TSI; Figure 5 Ⓒ) specifies the interface of the test system towards the SUT, which includes HTTP and ROAP.

2.1 DRM Test Management and Control (TMC)

The TMC consists of three entities: *Test Management* (TM; Figures 5 ⑥), *Test Logging* (TL; Figure 5 ⑦) and *External CoDecs* (ECDs; Figure 5 ⑧). The TM is responsible for controlling the creation and the execution of tests. In DRM test execution, the TM invokes the DRM modules (e.g., tc_ConRoap module which will be described in Figure 7) to propagate the module parameters and/or extra testing information to the TE.



tc_DRM_2_con_5.DRM_2_con_5_b
Start : 2008-03-21 13:54:05.308
End : 2008-03-21 13:54:18.061

! User Action Required: "Missing Status attribute in ROAP Response (Reg. Response processing)!!!"

Figure 6. Graphical test log for DRM registration.

The TL is responsible for maintaining the test log, such as the logging information about 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 6 shows a graphical test log that describes the interactions between MTC (Main Test

Component, i.e. NTP-DRMT; Figure 6 ①) and SYSTEM (i.e. SUT; Figure 6 ②) based on the registration test case. When NTP-DRMT receives an HTTP_GET message (Figure 4 ① and Figure 6 ③) from the SUT, NTP-DRMT verifies the HTTP_GET message and replies a ROAP Trigger message (Figure 4 ② and Figure 6 ⑤) to the SUT. Then NTP-DRMT receives the Device Hello message (Figure 4 ③ and Figure 6 ⑥), verifies this message, and replies a RI Hello message (Figure 4 ④ and Figure 6 ⑧). Upon receipt of the Register Request message (Figure 4 ⑤ and Figure 6 ⑨), NTP-DRMT verifies this message and replies a Register Response message (Figure 4 ⑥ and Figure 6 ⑪). Every "match" box (Figure 6 ④, ⑦ and ⑩) indicates that the received message matches the pass criteria described in the conformance test specification. The final "pass" box (Figure 6 ⑫) indicates that the SUT passes this test case.

The ECDs are invoked by the TE for encoding the TTCN-3 values into the bitstrings to be sent to the SUT or decoding the bitstrings sent from the SUT into the TTCN-3 values. Specifically, the TE passes the TTCN-3 values to an appropriate encoder to produce the encoded data. The received data is passed to an appropriate decoder to translate the received data into the TTCN-3 values. In NTP-DRMT, there are two ECDs in the `DRM_Codec.java` file; one for the HTTP procedure and the other for the ROAP procedure. These codecs are implemented in JAVA language so that they can be easily ported to different test systems. These two ECDs will be described in Section 4.

2.2. DRM TTCN-3 Executable (TE)

The TE consists of two interacting entities, *Executable Test Suite* (ETS; Figure 5 ⑨) and

TTCN-3 Runtime System (T3RTS; Figure 5 ⑩), to execute the DRM test cases. The T3RTS manages the ETS and interacts with the TMC, the SA and the PA. The T3RTS starts the execution of the DRM modules in the ETS. Figure 7 illustrates the ETS structure that classifies the DRM modules into two groups: conformance, and interoperability. Each module consists of a set of related test cases. For example, in the conformance group, the ROAP related conformance test case (e.g. DRM_2_con_5) is implemented in the tc_ConRoap module. This test case is invoked by the T3RTS when NTP-DRMT receives an HTTP_GET message (which is described in Section 1.3) to trigger the registration procedure.

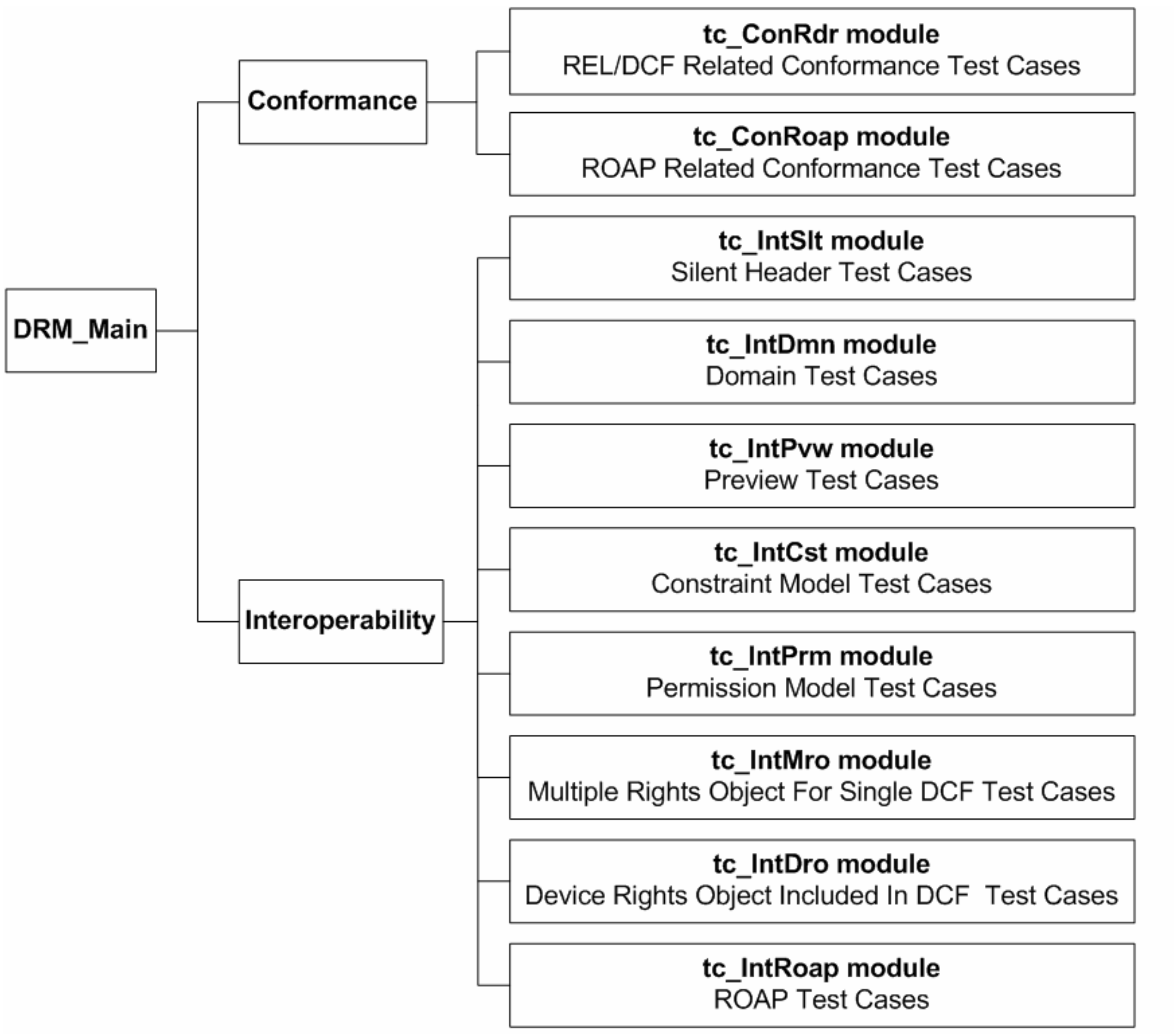


Figure 7. The structure of the DRM ETS.

2.3. DRM SUT Adapter (SA)

The SA adapts the communications between the TE and the SUT. The SA interacts with the TE through the TRI. Specifically, the test component ports pt_http (Figure 8 ③) and pt_roap (Figure 8 ④) that are mapped to the socket HTTPSocket (Figure 8 ⑤) in the SA. The SA binds the socket to the TSI port 8080 for the interaction with the SUT. To correctly deliver the HTTP and the ROAP messages to and from the SUT, the TE calls the TRI functions (to be described in Section 3.2) to associate the test component ports with the TSI port. Also, the TE invokes the ECDs (Figure 8 ① and ②) for the message encoding/decoding.

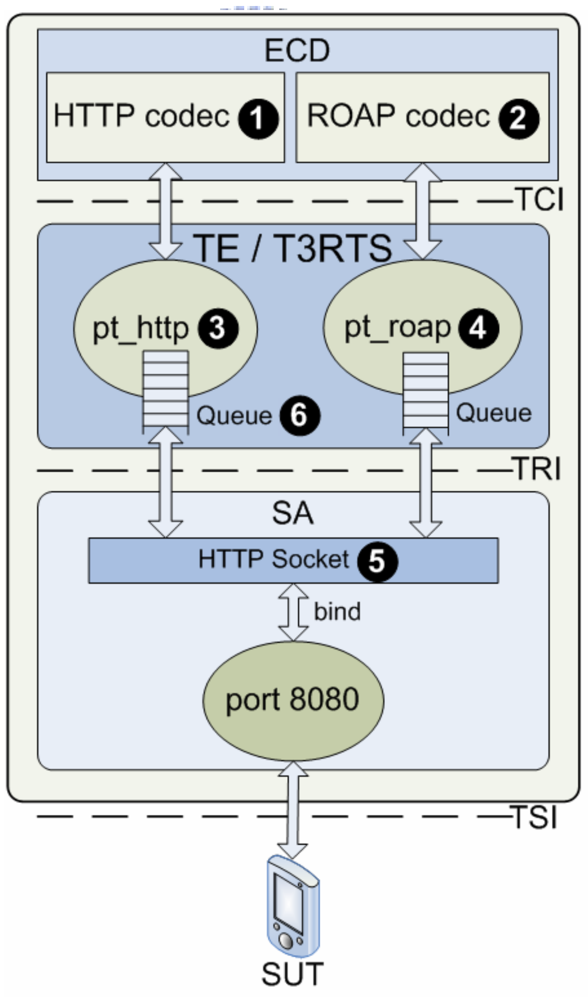


Figure 8. Interaction between the SA and the TE in NTP-DRMT.

The SA is responsible for propagating the DRM requests (e.g., sending a ROAP message through the `pt_roap.send` function) from the TE to the SUT and notifying the TE of any received test events form the SUT (e.g., receiving a ROAP message through the `pt_roap.receive` function) by buffering them in the TE's port queues (Figure 8 ⑥).

Chapter 3 TTCN-3 Interfaces for DRM

This section elaborates on the NTP-DRMT control and runtime interfaces. We first describe the *TTCN-3 Control Interface* (TCI), and then the *TTCN-3 Runtime Interface* (TRI).

3.1 The TTCN-3 Control Interface (TCI) for DRM

The TCI between the TE and the TMC has three sub-interfaces: Test Management Interface (TCI-TM), Test Logging Interface (TCI-TL) and Coding/Decoding Interface (TCI-CD). The TCI-TM supports the operations for controlling the test execution and providing module parameters. The TCI-TM program segment in Figure 9 illustrates some DRM module parameters. Figure 9 ① assigns the test system port HTTP_PORT with the value 8080. Figure 9 ② assigns the test system Uniform Resource Locator (URL). Figure 9 ③ assigns the maximum waiting time between two received messages with the value 30.0 (seconds).

The extension function (Figure 9 ④—⑥) displays the descriptions of the parameters to the tester and reminds the tester that these parameters can be changed. If the parameters are modified, the updated parameters are provided to the test system as the module parameters through the TCI-TM.

```
group SysParameters {
      modulepar {
①           integer   HTTP_PORT    := 8080;
②           charstring   DRMURL    :="http://localhost:8080";
③           float   DRM_SYS_WAIT    := 30.0;
      }
      with {
④           extension (HTTP _PORT)
                  "Description: Specify the port number of HTTP protocol";
⑤           extension (DRMURL)
                "Description: The test system URL.";
⑥           extension (DRM_SYS_WAIT)
                "Description: Maximum time between two received messages.";
      }
}
```

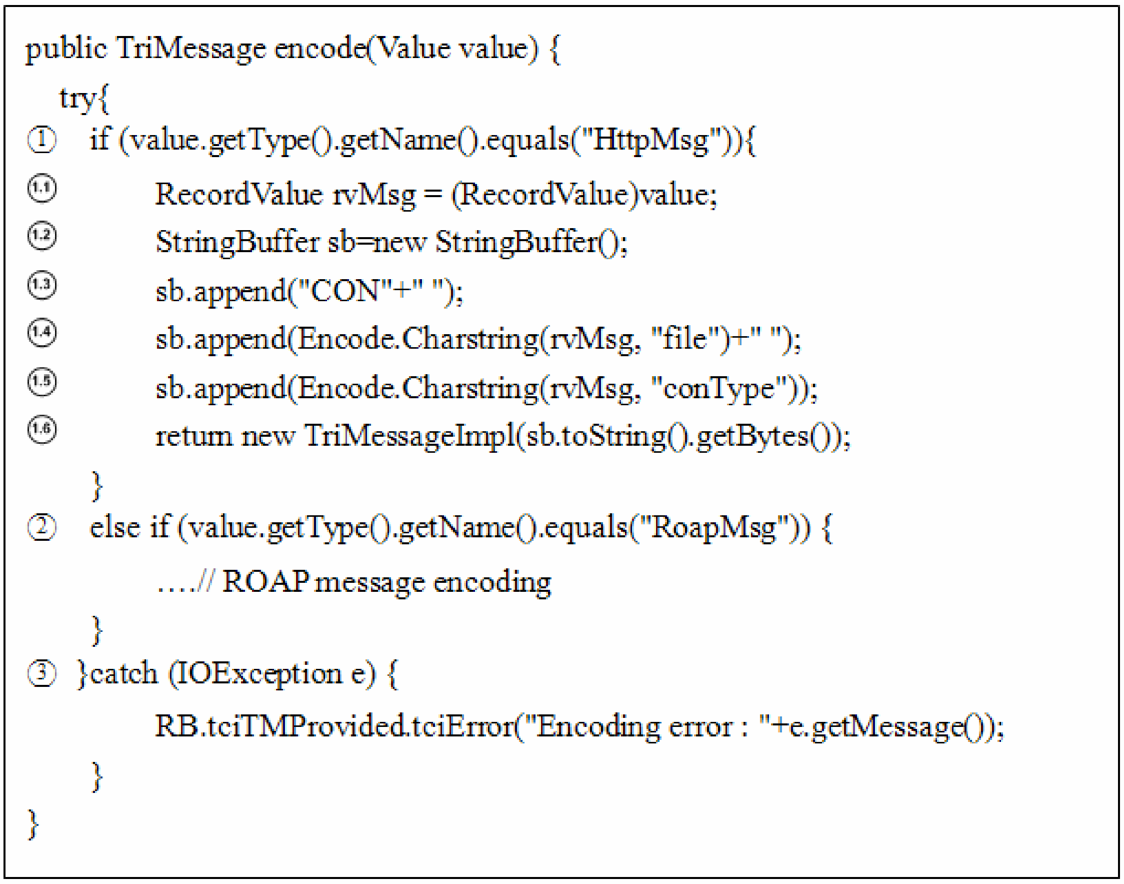
Figure 9. An example of DRM module parameter.

The TCI-TL includes the operations for retrieving the information about the test execution. Figure 10 illustrates a TCI-TL program segment that checks whether the signature is correct and logs the error if the signature is incorrect in the received ROAP message. In Figure 10 ①, NTP-DRMT checks if the signature in the received message is correct. If yes, NTP-DRMT replies a response message (Figure 10 (1.1)). Otherwise, NTP-DRMT logs the error information and shows the error message in the test log (Figure 10 ②).

```
①     if(DRM_Verify_Signature(v_recvMsg.roReq.sign)){
(1.1)       f_DRM_2_con_31_a_senRORsp(v_recvMsg);
            return SC_SUCCESS;
      } else {
②           log("Signature verify failed");
            return SC_FAIL;
      }
```

Figure 10. DRM test logging example.

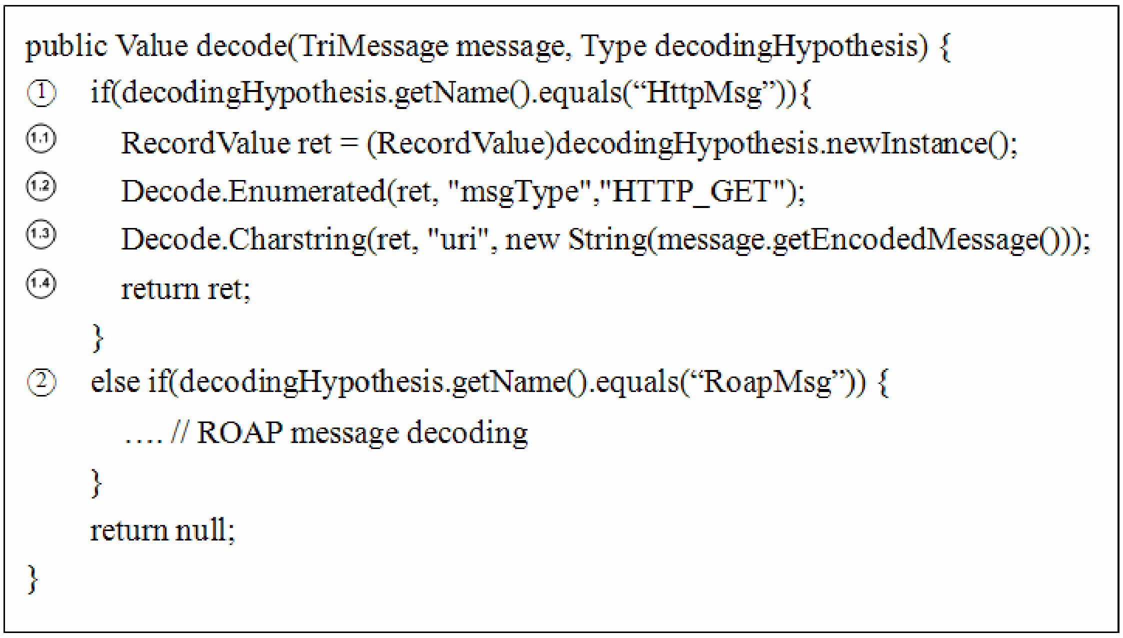
The TCI-CD provides the operations to access codecs. In NTP-DRMT, TCI-CD is implemented in `DRM_Codec.java` described in Section 2.1. Parts of the program are listed in Figure 11. In this example, if no encoding rule is matched, then Figure 11 ③ is executed for the exception handling. In Figure 11 ①, an HTTP message is encoded as follows. Figure 11 (1.1)—(1.5) construct the HTTP message structure and append content, and set the content type. Figure 11 (1.6) generates a byte string from this HTTP message structure. Figure 11 ② encodes a ROAP message, and the details are omitted.



```
public TriMessage encode(Value value) {
   try{
①    if (value.getType().getName().equals("HttpMsg")){
(1.1)        RecordValue rvMsg = (RecordValue)value;
(1.2)        StringBuffer sb=new StringBuffer();
(1.3)        sb.append("CON"+" ");
(1.4)        sb.append(Encode.Charstring(rvMsg, "file")+" ");
(1.5)        sb.append(Encode.Charstring(rvMsg, "conType"));
(1.6)        return new TriMessageImpl(sb.toString().getBytes());
      }
②    else if (value.getType().getName().equals("RoapMsg")) {
           ….// ROAP message encoding
      }
③   }catch (IOException e) {
           RB.tciTMProvided.tciError("Encoding error : "+e.getMessage());
      }
}
```

Figure 11. DRM encode operation.

The DRM 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 as follows. Figure 12 ① and ② decode the message as an HTTP or a ROAP message according to the message type which is set in the TRI (Section 3.2). For example, in Figure 12 ①, the HTTP message decodes into a structured TTCN-3 value. Figure 12 (1.2) sets the “msgType” to HTTP_GET, and Figure 12 (1.3) retrieves the *Uniform Resource Identifier* (URI).



```
public Value decode(TriMessage message, Type decodingHypothesis) {
①     if(decodingHypothesis.getName().equals(“HttpMsg”)){
(1.1)     RecordValue ret = (RecordValue)decodingHypothesis.newInstance();
(1.2)     Decode.Enumerated(ret, "msgType","HTTP_GET");
(1.3)     Decode.Charstring(ret, "uri", new String(message.getEncodedMessage()));
(1.4)     return ret;
      }
②     else if(decodingHypothesis.getName().equals(“RoapMsg”)) {
          …. // ROAP message decoding
      }
      return null;
}
```

Figure 12. DRM decode operation.

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

The TRI supports the communication of the TE with the SUT [13]. This interface is implemented in the SA to initialize the TSI and establish connections to the SUT. The TRI is implemented in a JAVA program `DRM_TestAdapter.java` consisting of the connection and the communication operations shown in Figure 13.

```
public class DRM_TestAdapter extends TestAdapter {
①   public TriStatus triMap(final TriPortId compPortId, final TriPortId tsiPortId)
{
            TriStatus mapStatus = CsaDef.triMap(compPortId, tsiPortId);
            if (tsiPortId.getPortName().equals("pt_roap")) {
            .... // bind Socket to port 8080, and wait for packets
②                Cte.triEnqueueMsg(tsiPortId, new TriAddressImpl(new byte[]{}),
                     compPortId.getComponent(), new TriMessageImpl(msg));
            }
             .... //other map functions
            return mapStatus;
      }

③   public TriStatus triUnmap(TriPortId compPortId, TriPortId tsiPortId) {
            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_roap")) {
            ...//send a ROAP message
            }
            else if (tsiPortId.getPortName().equals("pt_http")) {
                  FileInputStream fr= message.getEncodedMessage();
                  ByteArrayOutputStream os=new ByteArrayOutputStream();
                  byte data[]=new byte[1];
                  while(fr.read(data)!=-1)
                        os.write(data);
                  out.write(os.toByteArray());
                  return new TriStatusImpl();
            }
      }
}
```

Figure 13. DRM adapter operation.

The connection operations include map and unmap operations. Through the connection operations, the test component ports are mapped/unmapped to the TSI ports. An example is the `triMap` function (Figure 13 ①) called by the TE when TE executes the map operation (Figure 15 ①). This operation instructs the SA to establish a dynamic connection to the SUT.

The TRI also supports the communication operations which are used to exchange messages with the SUT. The communication operations include enqueue and send operations. An example is the `triEnqueueMsg` function (Figure 13 ②) called by the SA after SA has received a message from the SUT. When receiving an HTTP_GET message, this operation passes the message to the test component port of the TE (i.e., pt_http) and sets the message type HttpMsg. When receiving other messages, this operation passes these messages to pt_roap and sets the message type RoapMsg. Note that the test component port has been mapped to the TSI port beforehand. Another example is the `triSend` function (Figure 13 ④) called by the TE when the TE executes the send operation (Figure 16 ④). This operation instructs the SA to send a message to the SUT. For DRM testing, two types of messages are sent by the `triSend` function: DRMRoapMsg for ROAP and DRMHttpMsg for HTTP.

Chapter 4 A DRM Conformance Test Scenario

We use a conformance test case “missing States in RI Hello processing” to show how the DRM test suites are implemented. This test case verifies if the SUT correctly handles an incorrect RI Hello message without “Status" in the DRM registration procedure. The “Status” is used to indicate if the preceding request was successfully or not. If the SUT receives this message, it should display an error message and terminate the connection.

We utilize a *Finite State Machine* (FSM) to illustrate how the conformance test case “missing States in RI Hello processing” is implemented. Figure 14 shows the state diagram of the DRM conformance test case “missing States in RI Hello processing”.

- At **State 1** (Waiting for HTTP_GET), if the HTTP_GET message is received from the SUT, the TE sends a ROAP Trigger message to the SUT and changes the FSM to **State 2** (Waiting for Device Hello). If any ROAP message is received at **State 1**, the TE sets the verdict to “fail” and stops the FSM at **State 4** (Test Fail). If the timer expires and the TE does not receive any message, the TE sets the verdict to “inconc” and stops the FSM at **State 3** (Test Inconclusive).
- At **State 2**, if the Device Hello message sent from the SUT is correct, then the TE sends an incorrect RI Hello message to the SUT and changes the FSM to **State 5** (Waiting for Tester Post Result). If an HTTP_GET message is received at this state, the TE resends the ROAP Trigger message and waits for the Device Hello message. If an incorrect Device Hello message is received at this state, the TE sets the verdict to “fail” and stops the FSM at **State 4**. If the timer expires, the TE sets the verdict to “inconc” and stops the FSM at **State 3**.

- **State 3** is the "Test Inconclusive" state.
- **State 4** is the "Test Fail" state.
- At **State 5**, if the tester reports pass, the TE sets the verdict to "pass" and changes the FSM to **State 6** (Test Pass). If the tester reports fail, the TE sets the verdict to "fail" and stops the FSM at **State 4**.
- **State 6** is the "Test Pass" state.

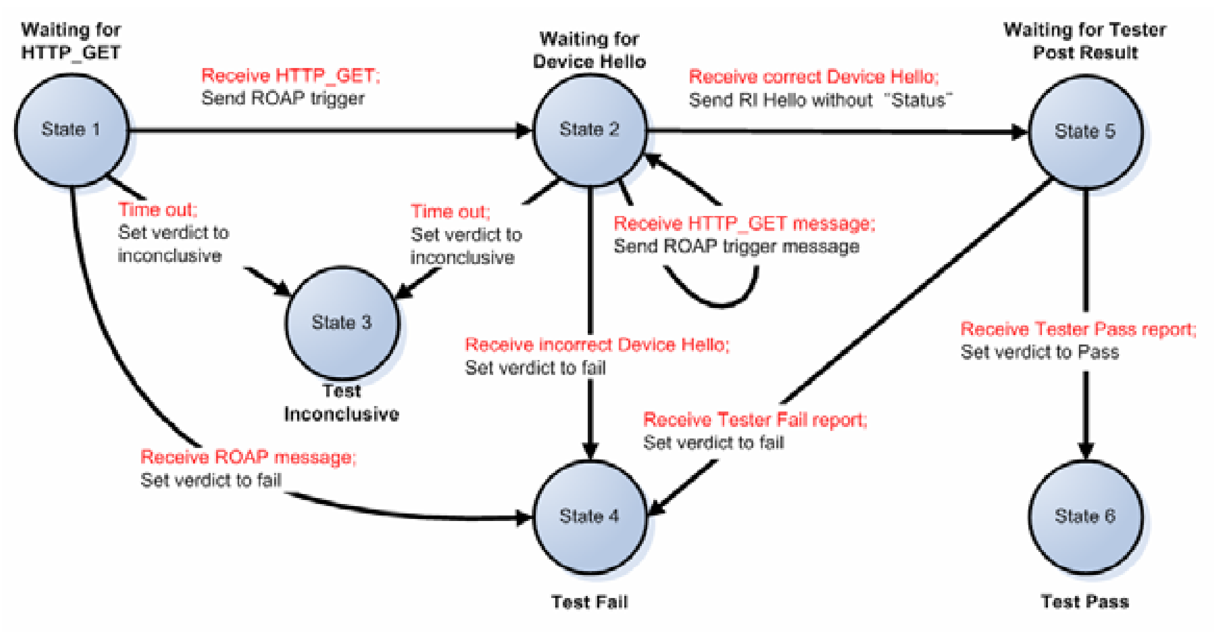
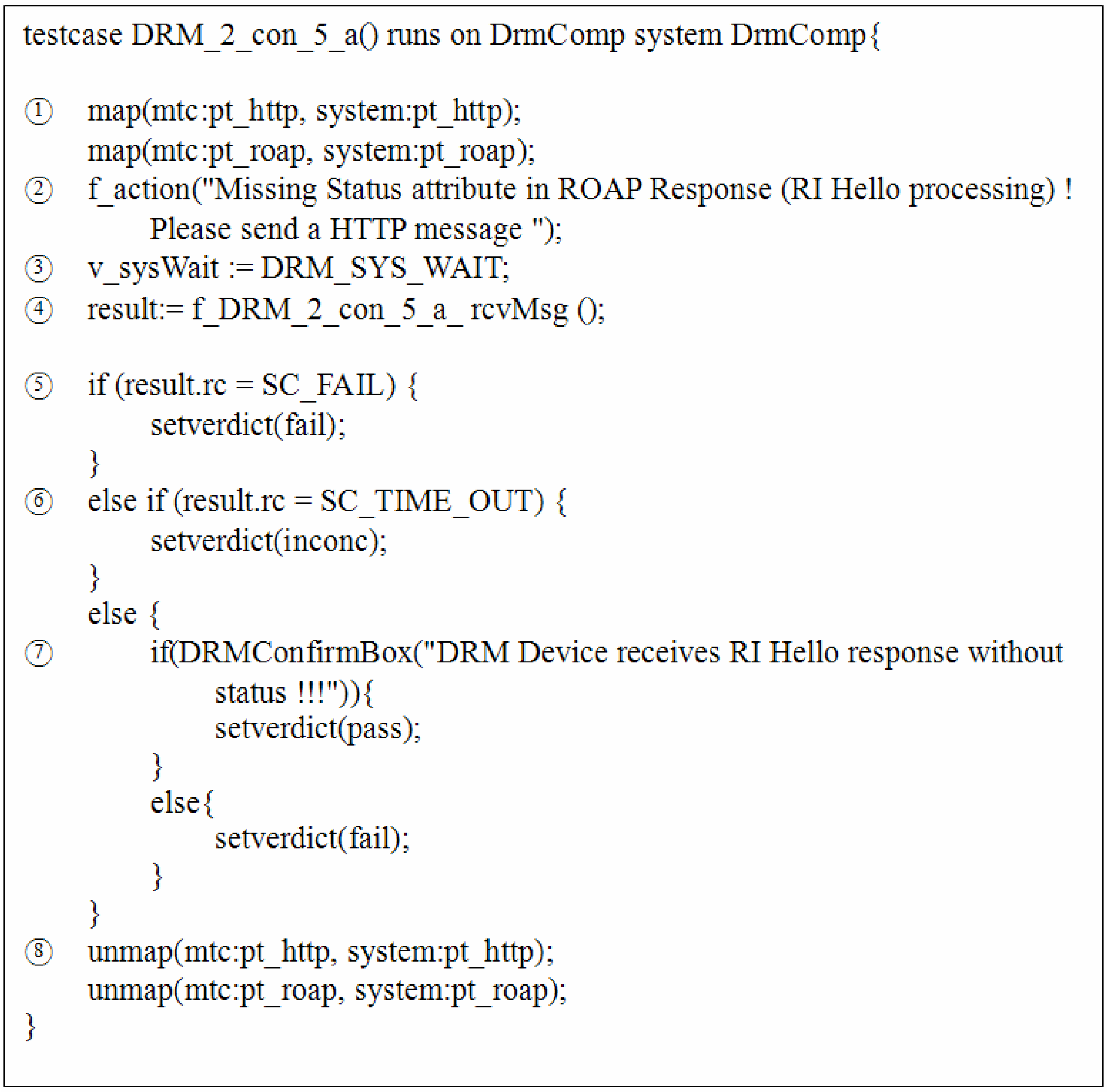

Waiting for HTTP_GET
State 1
Receive HTTP_GET; Send ROAP trigger
Waiting for Device Hello
State 2
Receive correct Device Hello; Send RI Hello without "Status"
Waiting for Tester Post Result
State 5
Time out; Set verdict to inconclusive
Time out; Set verdict to inconclusive
Receive HTTP_GET message; Send ROAP trigger message
State 3
Test Inconclusive
Receive incorrect Device Hello; Set verdict to fail
Receive Tester Pass report; Set verdict to Pass
Receive Tester Fail report; Set verdict to fail
Receive ROAP message; Set verdict to fail
State 4
Test Fail
State 6
Test Pass

Figure 14. DRM test case "missing States in RI Hello processing" state diagram.

Figure 15 illustrates the program segment for the test case. When the test starts, the TE first maps the two test component ports to the test system ports (Figure 15 ①; the `triMap` operation at the TRI is invoked). Then it pops up an action window (Figure 15 ②) that notifies the tester (Figure 3 ①) what the test case is and asks the tester to send an HTTP_GET message from the SUT (Figure 3 ②). In Figure 15 ③, the TE sets the waiting time. Then the TE invokes the `f_DRM_2_con_5_a_rcvMsg` function (Figure 15 ④) to check whether the received messages from the SUT are correct.



```
testcase DRM_2_con_5_a() runs on DrmComp system DrmComp{

①     map(mtc:pt_http, system:pt_http);
      map(mtc:pt_roap, system:pt_roap);
②     f_action("Missing Status attribute in ROAP Response (RI Hello processing) !
            Please send a HTTP message ");
③     v_sysWait := DRM_SYS_WAIT;
④     result:= f_DRM_2_con_5_a_ rcvMsg ();

⑤     if (result.rc = SC_FAIL) {
            setverdict(fail);
      }
⑥     else if (result.rc = SC_TIME_OUT) {
            setverdict(inconc);
      }
      else {
⑦           if(DRMConfirmBox("DRM Device receives RI Hello response without
                  status !!!")){
                  setverdict(pass);
            }
            else{
                  setverdict(fail);
            }
      }
⑧     unmap(mtc:pt_http, system:pt_http);
      unmap(mtc:pt_roap, system:pt_roap);
}
```

Figure 15. DRM test case “missing States in RI Hello processing”.

In the `f_DRM_2_con_5_a_rcvMsg` function, the `t_tMsg` timer starts at the PA (Figure 16 ①). When an HTTP_GET message from the SUT is received, the `pt_http.receive` function is executed (Figure 16 ②) and the TE invokes the decode operation in Figure 12. After the message is decoded, the TE stops the `t_tMsg` timer at the PA (Figure 16 ③), invokes `f_sndRegTrig` function (Figure 16 ④) to send a ROAP Trigger to the SUT, and goes to execute Figure 16 ⑦. If a ROAP message is received (Figure 16 ⑤), the test result (SC_FAIL) is returned. If the TE does not receive any message from the SUT after the t_tMsg timer expires, the PA notifies the TE of this timeout event (Figure 16 ⑥) and the f_DRM_2_con_5_a_rcvMsg function returns SC_TIME_OUT. The TE sets the verdict to “inconc” (Figure 15 ⑥) to indicate that an inconsistent exception occurs.

If the HTTP_GET message is received (Figure 16 ⑧), then the TE resends the ROAP Trigger to the SUT. If the TE receives a Device Hello message (Figure 16 ⑨), then the TE checks whether the header in the Device Hello message contains the DRM feature-tag “devhello version” (Figure 16 ⑩). If any check of the message is failed, the f_DRM_2_con_5_a_rcvMsg function returns SC_FAIL to indicate the failure. If the received Device Hello message is correct, then the TE sends an incorrect RI Hello message (without “Status") to the SUT. If the t_tMsg timer expires, the f_DRM_2_con_5_a_rcvMsg function returns SC_TIME_OUT (Figure 16 ⑪).

Finally, if the f_DRM_2_con_5_a_rcvMsg function returns SC_FAIL or SC_TIME_OUT, the TE sets the verdict to “fail” or “inconc” (Figure 15 ⑤ and ⑥). Otherwise, the TE pops up a confirm box (Figure 15 ⑦) to notify the tester to check whether the result reported by the SUT is correct and report the result. Then the TE sets the verdict to “pass” or “fail” according to the tester’s report. After the verdict is set, the TE removes the bindings between the test component ports and the test system ports (Figure 15 ⑧) using the triUnmap operation at the TRI.

```
function f_DRM_2_con_5_a_rcvMsg () runs on DrmComp return RetResult{
①    t_tMsg.start(v_sysWait);
     alt{
②        []pt_http.receive(a_httpget){
③            t_tMsg.stop;
④            f_sndRegTrig();
         }
⑤        []pt_roap.receive(){
             t_tMsg.stop;
             ret.rc:= SC_FAIL;
             return ret;
         }
⑥        []t_tMsg.timeout{
             log("timeout and nothing received");
             ret.rc:=SC_TIME_OUT;
             return ret;
         }
     }

⑦    t_tMsg.start(v_sysWait);
     alt{
⑧        []pt_http.receive(a_httpget){
             t_tMsg.stop;
             f_sndRegTrig();
             repeat;
         }
⑨        []pt_roap.receive(v_devhello) -> value v_recvMsg{
             t_tMsg.stop;
⑩            if(not ispresent(v_ devhello.version)){
                 ret.rc:=SC_FAIL;
                 return ret;
             }
             …//check other headers in the received Device Hello message
             f_DRM_2_con_5_a_sndRiHello();
             ret.rc:=SC_SUCCESS;
             return ret;
         }
⑪        []t_tMsg.timeout{
             log("timeout and nothing received");
             ret.rc:=SC_TIME_OUT;
             return ret;
         }
     }
}
```

Figure 16. The `f_DRM_2_con_5_a_rcvMsg` function.

Chapter 5 Conclusions

This thesis described the architecture and the operations of NTP-DRMT which is a DRM test system developed based on the TTCN-3 specifications. This system has been jointly developed by the *National Telecommunications Program* (NTP) and the *Industrial Technology Research Institute* (ITRI) in Taiwan. We used the DRM registration procedure to illustrate how the conformance test can be implemented in NTP-DRMT. The conformance and interoperability test cases are conformed to the OMA Enabler Test Specification (Conformance) for DRM-V2_0 [7] and the OMA Enabler Test Specification for DRM Interoperability [8]. Currently, 493 DRM tests cases have been developed in NTP-DRMT.

Reference

[1] Open Mobile Alliance, "DRM Specification", OMA-TS-DRM-DRM-V2_0- 2006 0303-A, 2006.

[2] Open Mobile Alliance, "DRM Architecture", OMA-AD-DRM-V2_0-20060303-A, 2006.

[3] Open Mobile Alliance, "DRM Content Format", OMA-TS-DRM-DCF-V2_0-20060303-A, 2006.

[4] Open Mobile Alliance, "DRM Rights Expression Language", OMA-TS-DRM-REL-V2_0-20060303-A, 2006.

[5] Open Mobile Alliance, "OMA DRM Requirements", OMA-RD-DRM-V2_0-20060303-A, 2006.

[6] Open Mobile Alliance, "Enabler Release Definition for DRM V2.0", OMA-ERELD-DRM-V2_0-20060303-A, 2006.

[7] Open Mobile Alliance, "Enabler Test Specification (Conformance) for DRM- V2_0", OMA-ETS-DRM_ CON_Test_Client-V2_0-20060615-C, 2006.

[8] Open Mobile Alliance, "Enabler Test Specification for DRM Interoperability", OMA-ETS-DRM-INT- V2_0-20060704-C, 2006.

[9] ETSI, "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language", ETSI ES 201 873-1, V3.1.1, 2005.

[10] ETSI, " Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 2: TTCN-3 Tabular presentation Format (TFT)", ETSI ES 201 873-2 V3.1.1, 2005.

[11] ETSI, " Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 3: TTCN-3 Graphical presentation Format (GFT)", ETSI ES 201 873-3 V3.1.1, 2005.

[12] ETSI, " Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 4: TTCN-3 Operational Semantics", ETSI ES 201 873-4 V3.1.1, 2005.

[13] 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.

[14] 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.

[15] ETSI, " Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3", ETSI ES 201 873-7 V3.1.1, 2005.

[16] Lin, Y.-B., Liang, C.-F., Chen, K.-H., Liao, H.-Y. "NTP-SIOT: A Test Tool for Advanced Mobile Services", IEEE Network. VOL 21; NUMB 1, pages 21-26, 2007.

[17] Lin, Y.-B., Wang, C.C., Lu, C.H., Hsu, M.R. "NTP-PoCT: A Conformance Test Tool for Push-to-talk over Cellular Network", Wireless Communications and Mobile Computing. VOL 8; NUMBER 5, pages 673-686, 2008.

[18] Anritsu Corporation, MD8470A Signaling Tester Product Introduction, http:// www.us.anritsu.com/products/ ARO/North/Eng/showProd.aspx?ID=659.

Appendix A The conformance and interoperability test cases

The conformance test cases

Test case ID	Test case description
DRM-2.0-con-1	ROAP trigger with expired RI context
DRM-2.0-con-3	Missing Signature in Leave Domain trigger
DRM-2.0-con-4	Invalid Signature in Leave Domain trigger
DRM-2.0-con-5	Missing Status attribute in ROAP Response
DRM-2.0-con-6	Status ≠ Success in ROAP Response
DRM-2.0-con-7	Missing Signature in ROAP Response
DRM-2.0-con-8	Invalid Signature in ROAP Response
DRM-2.0-con-9	1-pass RO Response processing reception while expired RI context
DRM-2.0-con-29	Missing Session ID in Register Response processing
DRM-2.0-con-30	Invalid Session ID in Register Response processing
DRM-2.0-con-31	Missing Device ID in ROAP response; 2 pass RO acquisition and Join Domain
DRM-2.0-con-32	Invalid Device ID in ROAP response; 2 pass RO acquisition and Join Domain
DRM-2.0-con-33	Missing Device ID in 1-pass RO Response processing
DRM-2.0-con-34	Invalid Device ID in 1-pass RO Response processing
DRM-2.0-con-35	Missing Device Nonce in ROAP response
DRM-2.0-con-35	Missing Device Nonce in Leave Domain Response processing
DRM-2.0-con-36	Invalid Device Nonce in ROAP response
DRM-2.0-con-37	Missing RI ID in ROAP response
DRM-2.0-con-38	Invalid RI ID in ROAP response
DRM-2.0-con-40	Install Device RO from RO Response processing; Invalid Signature
DRM-2.0-con-41	Install Device RO from RO Response processing; Missing MAC element
DRM-2.0-con-42	Install Device RO from RO Response processing; Invalid MAC element
DRM-2.0-con-43	Install Device RO from RO Response processing; Missing RI ID
DRM-2.0-con-44	Install Device RO from RO Response processing; Invalid RI ID
DRM-2.0-con-45	Install Device RO from RO Response processing; Missing Signature
DRM-2.0-con-46	Install Device RO from RO Response processing; Invalid Signature
DRM-2.0-con-47	Install Device RO from RO Response processing; Missing MAC element
DRM-2.0-con-48	Install Device RO from DCF; Invalid MAC element
DRM-2.0-con-49	Install Device RO from DCF; Missing RI ID
DRM-2.0-con-50	Install Device RO from DCF; Invalid RI ID
DRM-2.0-con-51	Install Device RO from DCF; RI Context Expired
DRM-2.0-con-52	Consume rights in Device RO; Invalid Hash value
DRM-2.0-con-53	Install Domain Context; Missing MAC
DRM-2.0-con-54	Install Domain Context; Invalid MAC
DRM-2.0-con-55	Install Domain Context; Missing RI ID in DomainKey
DRM-2.0-con-56	Install Domain Context; Invalid RI ID in DomainKey
DRM-2.0-con-57	Delete Domain Context
DRM-2.0-con-58	Install Domain RO; No valid RI context with corresponding RI ID

DRM-2.0-con-59	Install Domain RO; Missing Signature
DRM-2.0-con-60	Install Domain RO; Invalid Signature
DRM-2.0-con-61	Install Domain RO; Missing Domain ID
DRM-2.0-con-62	Install Domain RO; Invalid Domain Generation
DRM-2.0-con-63	Install Domain RO; Device not in the domain
DRM-2.0-con-64	Install Domain RO; Missing MAC.
DRM-2.0-con-65	Install Domain RO; Invalid MAC.
DRM-2.0-con-66	Install Domain RO; RI Context Expired
DRM-2.0-con-67	Replay protection – Stateful RO with RITS; Future RITS
DRM-2.0-con-68	Replay protection – Stateful RO with RITS; In Replay cache
DRM-2.0-con-69	Replay protection – Stateful RO with RITS; Early RITS
DRM-2.0-con-70	Replay protection – Stateful RO without RITS; In Replay cache
DRM-2.0-con-71	Parent Rights object; Invalid Rights issuer
DRM-2.0-con-72	Nonce generation on Device without system shutdown
DRM-2.0-con-73	Nonce generation on Device with system shutdown
DRM-2.0-con-74	Wrong permissions for an image object
DRM-2.0-con-75	Wrong permissions for a sound object
DRM-2.0-con-76	Wrong permissions for an application object
DRM-2.0-con-77	Unknown permissions
DRM-2.0-con-78	Export permissions ("move") for DCFs with stateless rights object
DRM-2.0-con-79	Export permissions ("copy") for DCFs with stateless rights object
DRM-2.0-con-80	Export permissions ("move") for DCFs with stateful rights object
DRM-2.0-con-81	Export permissions ("copy") for DCFs with stateful rights object
DRM-2.0-con-82	Export permissions not present for DCF
DRM-2.0-con-83	Instant Preview
DRM-2.0-con-85	Erroneous Count constraint
DRM-2.0-con-86	Erroneous Timed-Count constraint
DRM-2.0-con-87	Erroneous Datetime constraint
DRM-2.0-con-88	Erroneous Interval constraint
DRM-2.0-con-89	Erroneous Accumulated constraint

The Interoperability test cases

Test case ID	Test case description
DRM-2.0-int-1	To test “Forward Lock” DRM 1.0 functionality.
DRM-2.0-int-2	To test DRM 1.0 “Combined Delivery” functionality
DRM-2.0-int-3	To test DRM 1.0 “Separate Delivery” functionality
DRM-2.0-int-4	To test RO Registration and RO Acquisition
DRM-2.0-int-5	To test RO Registration with existing RI Context
DRM-2.0-int-6	To test RO Acquisition without existing RI Context
DRM-2.0-int-7	To test 1-pass RO Acquisition with existing RI Context
DRM-2.0-int-8	To test 1-pass RO Acquisition without existing RI Context
DRM-2.0-int-10	To test a situation where an RO is included in the DCF
DRM-2.0-int-11	To test behavior in the presence of a group RO for multiple DCFs, using the Group ID mechanism
DRM-2.0-int-12	To test behavior in the presence of an individual RO for a content item which has a Group ID
DRM-2.0-int-13	To test behavior in the presence of several rights objects for one piece of content

DRM-2.0-int-14	To test behavior in the presence of several rights objects for one piece of content
DRM-2.0-int-15	To test DRM Agent's capability to process Multipart DCFs from the RI
DRM-2.0-int-16	To test behavior in the presence of multiple ROs for a multipart DCF
DRM-2.0-int-17	To test behavior when different content items in a multipart DCF are associated with different groups
DRM-2.0-int-18	To test "Superdistribution" functionality. The protected content is sent from one DRM Agent to another. The rights object is obtained by ROAP session to the rights issuing service.
DRM-2.0-int-19	To test the TransactionID mechanism in connection with Superdistribution
DRM-2.0-int-20	To test <display> and <print> permissions
DRM-2.0-int-21	To test <play> permission
DRM-2.0-int-22	To test <execute> permission for an application object
DRM-2.0-int-23	To test <count> constraint for a DCF
DRM-2.0-int-24	To test <timed-count> constraint for a DCF
DRM-2.0-int-25	To test <datetime> constraint for a DCF
DRM-2.0-int-26	To test <interval> constraint for a DCF
DRM-2.0-int-27	To test <accumulated> constraint for a DCF
DRM-2.0-int-28	To test <individual> constraint for a DCF
DRM-2.0-int-29	To test <system> constraint for a DCF
DRM-2.0-int-30	To test the effect of having multiple constraints
DRM-2.0-int-31	To test the REL Permission Model in the case that the rights include a stateful top level constraint
DRM-2.0-int-32	Initiate ROAP from DCF Preview Header with existing RI Context & domain name NOT in Domain Name Whitelist
DRM-2.0-int-33	Initiate ROAP from DCF Preview Header with existing RI Context & domain name in the Domain Name Whitelist
DRM-2.0-int-34	To test inheritance model when stateful constraints are involved
DRM-2.0-int-35	To test a case where the Parent Rights Object
DRM-2.0-int-36	To test inheritance model when a child RO is a group RO
DRM-2.0-int-37	Trigger-initiated domain join without existing RI Context
DRM-2.0-int-38	Trigger-initiated domain join with valid RI Context and no existing Domain Context for this RI
DRM-2.0-int-39	Automatically-initiated domain upgrade with valid RI Context and existing Domain Context for this RI
DRM-2.0-int-40	Trigger-initiated domain join with valid RI Context and existing Domain Context for this RI
DRM-2.0-int-41	Domain RO Acquisition with existing RI Context
DRM-2.0-int-42	To test delivering the DomainRO inside a DCF
DRM-2.0-int-43	To test if different devices related with the same domain are able to share DCFs
DRM-2.0-int-44	Device leaves a domain after receiving a LeaveDomain trigger
DRM-2.0-int-45	Initiate ROAP from DCF Silent Header with existing RI Context and domain name NOT in Domain Name Whitelist
DRM-2.0-int-46	Initiate ROAP from DCF Silent Header with existing RI Context and domain name NOT in Domain Name Whitelist