

Module Test Environment (MTE)

A SIPTECH CASE STUDY

Version: 1.00

Created Date: 29. Dec. 2008

Inside this document ...

01. Background
02. Problem / Challenges
03. Method / Intervention / Solution
04. Results Achieved
05. Lessons Learned / Conclusion

© 2005 All rights reserved. SIP Technologies and Exports Ltd. The information contained in this document is CONFIDENTIAL and PROPRIETARY in nature, and subject to the rights and ownership of SIPTECH. Any and all unauthorized copying or use of the contents hereof is prohibited.

1. Background

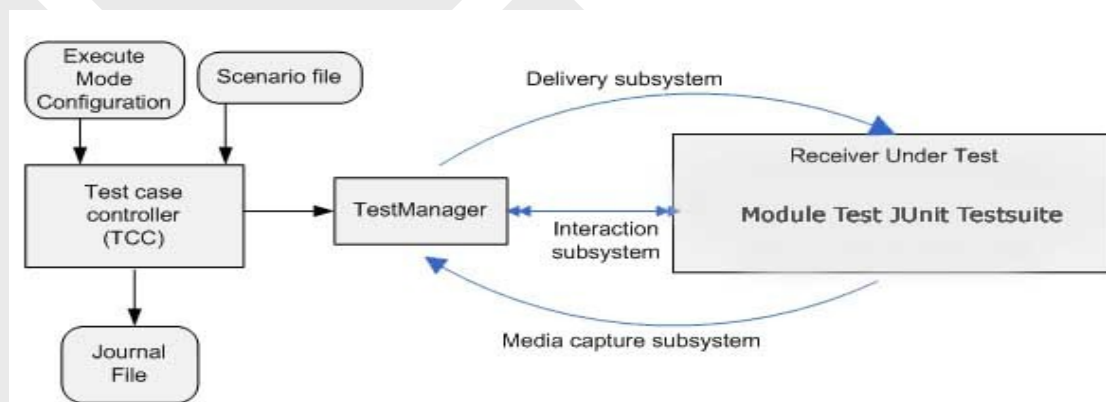
The client is a major set-top box vendor; they requested SIPTech to develop a framework for comprehensive unit testing of various modules of the OCAP middleware implementation. Unit testing covers a broad range of testing at both the API level as well as functional level. Unit testing verifies the implementation of the APIs both public and private. The testing of individual modules provides more capability for locating and isolating issues to the module level. Additionally, it helps in the verification of any modules that are modified or reused for other middleware implementations in the future.

2. Problem / Challenge

As there is no existing framework to stream the Junit test cases and the relevant test data to the Set-Top-Box (STB) to unit test the individual OCAP modules and also to communicate the results back to the user from the STB.

3. Method / Intervention / Solution used

To overcome this problem SIPTech developed a new test framework called Module Test Environment (MTE). MTE is built as an extension to CableLabs Automated test environment (ATE). It adds the capability to run JUNIT and native test suites.



This framework manages the lifecycle of the test case on the OCAP/MHP receiver. The Application

framework is sent along with the test case in the object carousel. In the manifest file for each test case, the initial class has been defined.

System Overview

The following sections describe the modules proposed for streaming data in simulation environment.

- ❖ TestInitiator (Tester's Machine)
- ❖ TestSuiteReceiver (Part of Test Manager)
- ❖ IPServer (Part of Test Manager)
- ❖ IPClient (Part of Simulation Environment)

TestInitiator (Tester's Machine)

TestInitiator starts the TestInitiator application by passing the test suite as a .jar file to the TestManager (Sony ATE). It also sends the test request details to the TestManager. The request contains information such as IP Address of the Initiator and the ModuleName to be tested.

TestSuiteReceiver (Part of Test Manager)

The TestSuiteReceiver receives the .jar file and saves it to the Test compilation directory of the ATE and makes an entry of the IP address, port of the TestInitiator, and the modulename of the test in the StreamFileInfo.properties file and set the status as NO.

After making an entry in the SystemFileInfo.properties it will invoke the tcc, softoc, tp2sec sequentially. When the dat file is created by the tp2sec file TestSuiteReceiver sets the STATUS in the StreamFileInfo.properties file as YES. Then it calls the IPServer.

System Design

The IP interface design has two modules in it.

- ❖ IPServer (Part of Test Manager)
- ❖ IPClient (A part of simulation environment)

IPServer (Part of Test Manager)

It resides as a part of the Test Manager. The IPServer executes continuously irrespective of state changes in TP2SEC. The IPServer is split into two modules to increase the efficiency of the system.

- ❖ PacketSender

❖ JournalWriter

PacketSender (Multicasting application)

PacketSender is invoked with the information such as ModuleName, IP and Port needed to stream a .DAT file. The application reads the StreamFileInfo.properties which contains the status of the .DAT file to be streamed.

If the value of STATUS field in the property file is set to YES then application reads the content of the .DAT file located at a particular path and breaks it into packets. Before sending the stream, the application sends a StartPacket which identifies that the packet following it contains the Test Data.

Once the .DAT file to be streamed is fully send the application sends a EndPacket which tells that the streaming of data is completed. So that Multicasting for the .DAT files is made based on the request.

JournalWriter (ResultReceiver)

The JournalWriter is a separate stand alone module listens in a particular port, which will be started by the IPSEver. Where all the results send by ResultSender (IPClient) after execution of the test cases are received. Once the RUT completes the test case execution and returns the results the JournalWriter receives the results and writes into a journal file. The journal file will have the information such as moduleName and Test result. Thus the results can be viewed at any time from the journal file for a particular module.

The IPSEver is solely a part of the TestManager.

Implementation

The implementation talks about how the proposed design is implemented to meet the requirements.

The system is initiated with a request from the TestInitiator. The request will have information such as ModuleName to be streamed, IP address of the initiator and the port through which the .DAT file has to be streamed.

Once the request is made, The TestManager invokes the execution of tcc, softoc and tp2sec the packet sender. The PacketSender reads the StreamFileInfo.properties file and reads the status of .DAT file to be streamed. The application checks the value of STATUS to be YES. Once it finds the value of STATUS as YES it reads the .DAT that has to be streamed.

Once the check through over the property file is completed it creates Multicast Sockets for .DAT files requested with corresponding ports through which the streaming has to be made. The information on IP and Port through which the streaming has to be made is in SystemFileInfo.properties.

The contents of the file are packetized using the Datagram Packets in `java.net.DatagramPacket`. The start packet is identified in such a way that the packet content will have all 1's. On the receiver side the packet whose contents are 1's are identified as start packet.

Soon after sending the StartPacket the other packets which contain the data from .DAT file will be sending over the network. After sending all the contents of the .DAT file the application creates and sends a EndPacket whose contents will have all 0's. The receiver will identify that the EndPacket and stops receiving or listening to that particular port.

Once PacketSender sends some Datagram Packets it invokes the Journalwriter to start its execution. The JournalWriter listens in a particular port to receive the test results. Soon after receiving the test results it writes the results in a journal file from which the TestEngineers can view the results of their test.

IPClient (Part of Simulation Environment)

The IPClient which is a part of IP interface resides on the simulation environment. The IPClient has two different modules in it.

- ❖ PacketReceiver
- ❖ ResultSender

PacketReceiver

This resides as a part of the simulation environment. Once this application is invoked, it continuously checks for StartPacket. Soon after receiving the StartPacket it starts accumulating the contents of the data packets received which contains the content of .DAT file needed to perform the test. While receiving the data packets it also checks whether the packet received is EndPacket. Once it encounters with EndPacket it stops listening on the port and makes an entry for the .DAT file with a particular IP and port (frequency) in the sections. properties file. Thus the simulation environment loads the .DAT file whenever it needs for testing.

ResultSender

The result sender is invoked after executing the test cases in the RUT. The ResultSender is invoked with the test result as a string. It sends the results to the IPServer where it can receive the results and write into a journal file.

Implementation

This part talks about how we are going to implement the IPClient part of the IP interface.

PacketReceiver can listen (Tune) to a particular port (Frequency). The PacketReceiver application listens and receives data packets of .DAT file streamed by the IP Server in the TestManager. It checks for the kind of packet. If the packet is identified as StartPacket it starts receiving the data packets which are streamed after the StartPacket. The start packets are identified in such a way that the data will have all 1's in it.

As soon as the packets are received it accumulates the content till it comes across an EndPacket. The end packet can be identified in such a way that the data will have all 0's in it. As the received packet has the contents of .DAT file an entry is made in Sections.properties file stating that the file .DAT file is completely received.

Once the test cases are executed in the RUT. The ResultSender is invoked, which sends the test results to the IP Server where it writes the results in a journal file. The result is expected to be a string to the resultsender.

Sequence of Operations:

- ❖ TestInitiator requests for testing a particular module.
- ❖ TestSuiteReceiver application makes an entry in the SystemFileInfo.properties with the information's such as IP address of TestInitiator, Port, and modulename. The status of the .DAT file is set to NO.
- ❖ Tcc, softoc, tp2sec are executed sequentially. After creating the .DAT file the SystemFileInfo.properties file is updated and the STATUS field is set to YES.

IP Server

- ❖ TestManager Invokes the PacketSender to stream the .DAT file needed for testing.
- ❖ Reads the STATUS of the .DAT file from the StreamFileInfo.properties file
- ❖ Checks whether the STATUS is YES. If the status is YES goto Step 7, else goto step 5.
- ❖ Sends a startPacket whose data element has all 1's.
- ❖ Reads the contents of .DAT file and constructs packets and sends across the network through the port.
- ❖ Once the .DAT file contents are completely streamed it sends an End Packet.
- ❖ Invokes ResultListener.
- ❖ Receives the result file and writes it in Journal file and sends the stream from where the TestInitiator can read.

IPClient

- ❖ TestManager Invokes the PacketSender to stream the .DAT file needed for testing.
- ❖ Invokes Packet Receiver and it Starts Listening for the Streams or Datagram Packet.

- ❖ Check for the Start packet.
- ❖ Once start packet is identified then it starts accumulating the contents of the other packets received in that particular port else step 2 is continued.
- ❖ Checks the received packets for End Packet. If received packet is End Packet stops receiving packets from that port.
- ❖ Stores the contents accumulated into a file. Adds the name of the .DAT file in sections.properties file.
- ❖ Once the test cases are executed in the RUT. ResultSender is invoked with test results as a string.
- ❖ The test results are sent to the IP Server.
- ❖ Step 1 to 5 can be repeated when ever specific .DAT file stream is required.

4. Results Achieved

By using this MTE framework unit test execution become simple and faster. By using this testing framework the dependency on the development team by the testing team was eliminated optimally. By using this MTE framework with the Simulator, we can able to eliminate the need of the target STB's (with monolith build) for the module testing. Testers can install the simulator in a standalone Windows PC and all the testers can stream and execute their individual module level test suites using the ATE and the simulator. This intern reduced the cost of testing and increased the productivity of the entire testing team.

5. Lessons Learned / Conclusion

By developing this MTE framework, automating the test execution reduces the time spent on the test execution and increases the turn around time for capturing the bugs. Overall it also increased the productivity of the entire testing team, by eliminating the unnecessary waiting time for the individual testers. Since the entire MTE test framework was developed as reusable components, it was very useful during integration with the ATE.