JDS Optimisation And SDM Test Automation

Submitted By

Harshit Bhojak

15MCEI07

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

JDS Optimisation And SDM Test Automation

Thesis

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology in Computer Science and Engineering(INS)

Submitted By

Harshit Bhojak

(15MCEI07)

Guided By

Prof. Malaram Kumhar

Nirma University, Ahmedabad.

Mr. Srinivas Davuluri

ARRIS India Pvt. Ltd.

DEPARTMENT OF COMPUTER ENGINEERING

INSTITUTE OF TECHNOLOGY

NIRMA UNIVERSITY

AHMEDABAD-382481

May 2017

Certificate

This is to certify that the thesis entitled **"JDS Optimisation And SDM Test Automation"** submitted by **Harshit Bhojak (Roll No: 15MCEI07)**, towards the partial fulfillment of the requirements for the award of degree of Master of Technology in Computer Science and Engineering(INS) of Nirma University, Ahmedabad, is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this thesis, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Prof. Malaram Kumhar
Guide & Assistant Professor,
Information Technology Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Sharada Valiveti
PG Coordinator-INS,
Associate Professor,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
Computer Engineering Department,
Institute of Technology,
Nirma University, Ahmedabad.

Dr. Alka Mahajan
Director,
Institute of Technology,
Nirma University, Ahmedabad

Statement of Originality

I, **Harshit Bhojak**, Roll. No. **15MCEI07**, give undertaking that the Thesis entitled "**JDS Optimisation And SDM Test Automation**" submitted by me, towards the partial fulfillment of the requirements for the degree of Master of Technology in **Computer Science and Engineering(INS)** of Institute of Technology, Nirma University, Ahmedabad, contains no material that has been awarded for any degree or diploma in any university or school in any territory to the best of my knowledge. It is the original work carried out by me and I give assurance that no attempt of plagiarism has been made.It contains no material that is previously published or written, except where reference has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

_____

Signature of Student

Date:

Place:

Endorsed by

Prof. Malaram Kumhar

(Signature of Guide)

Acknowledgements

First and foremost, sincere thanks to Mr. RaviKiran, Manager, ARRIS India Private Limited, Bangalore.

I would like to thank my Mentor, Mr. Srinivas Davuluri, ARRIS India Private Limited, Bangalore for his valuable guidance.He has given me much valuable advice on this project work.

It gives me immense pleasure in expressing thanks and profound gratitude to **Prof. Malaram Kumhar**, Assistant Professor, Information Technology Department, Institute of Technology, Nirma University, Ahmedabad for his valuable guidance and continual encouragement throughout this work. The appreciation and continual support he has imparted has been a great motivation to me in reaching a higher goal. His guidance has triggered and nourished my intellectual maturity that I will benefit from, for a long time to come.

It gives me an immense pleasure to thank **Dr. Sanjay Garg**, Head of Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his kind support and providing basic infrastructure and healthy research environment.

A special thank you is expressed wholeheartedly to **Dr Alka Mahajan**,Director, Institute of Technology, Nirma University, Ahmedabad for the unmentionable motivation she has extended throughout course of this work.

I would also thank the Institution, all faculty members of Computer Engineering Department, Nirma University, Ahmedabad for their special attention and suggestions towards the project work.

- Harshit Bhojak

15MCEI07

Abstract

Automation is one of the key area on which todays IT companies are focusing on. In the field of software product testing companies prefer automated testing instead of manual testing . The reason behind using automated tasks is, it requires less human efforts and provides more efficiency. The main aim of this project is to design and develop test cases for SDM and automate the testing process. Before doing test automation for SDM, there is a need of creating virtual environment which can provide a set of virtual devices for SDM testing.A simulated network environment can be very useful for testing purpose. As it reduces the use of real devices in testing, the cost spend by company on testing of a device manager is also reduced. In simulated network, device is not a physical entity but it is only the database of real device. So this virtual device can function as a real device without being physically present. JDS was developed with the goal of providing a virtual setup of devices to SDM. It supports all the headend devices with smaller database. But for larger device it becomes non-responsive. It has some major issues with larger devices APEX1000 and APEX3000. Even a single APEX 3000 was not running in JDS with full configuration. JDS optimisation and improvement is much needed in order to test SDM for all the headend devices. Along with this issue there is one more issue associated with JDS, It is not organized as a project. JDS is having a single folder with shell scripts, java classes, proto files and database. This project is focusing on providing support for apex device , reduce the insertion time of apex device to database,create a single JAR for JDS, design and automate test cases for SDM.As a result of optimisation, the setup time of test environment is reduced and JDS can run more than one APEX 3000 device at a time. And we can create simulated environment using single JAR file.

Conventions

Typesetting

This thesis is typeset using Latex software.
Font used in this thesis are of Times new roman family.

Referencing

Referencing and citation style adopted in this thesis is ieee transaction(ieeetr).
For electronic references, Last publication date is shown here.

Spelling

The Unites States English Spelling is adopted here.

Units

The Units used in This thesis are based in the International System of Units(SI Units), unless specified.

Abbreviations

ASN	Abstract Syntax Notation
GUI	Graphical User Interface
JAR	Java Archive
JDS	Java Device Simulator
JVM	Java Virtual Machine
MIB	Management Information Base
OID	Object identifier
RF	Radio Frequency
SDM	SmartStream Device Manager
SNMP	Simple Network Management Protocol

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

The Cable Delivery System for SET-TOP box includes many components. From the origin of signal to delivering at the consumer's TV, many devices are arranged in between to perform a specific task.The signals are Up-link to the Satellite and through Satellite,they reach at the setup of devices(Headend Facility) for further processing.These devices are Known as Signal Processing Equipments.

Headend facility is a setup of devices,which is present at service provider's side.It includes a number of devices which are used to process the signals received from satellite and forward them to customer premises.

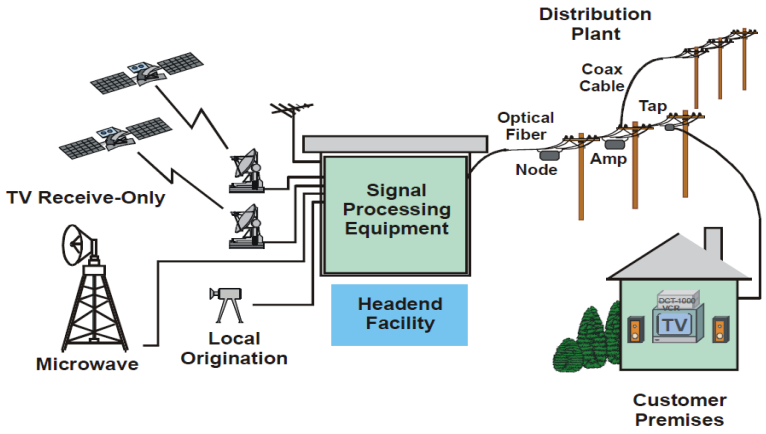

Figure 1.1: Cable TV Delivery System

1.2 SmartStream Device Manager

To manage all these devices of headend, ARRIS provides a device manager named as SDM. It is installed on a server (computer) dedicated to element management.It communicates through Simple Network Management Protocol

The SmartStream Device Manager (SDM):

- Is used to manage ARRIS digital video network elements.
- It can automatically discover devices on the network.
- It automatically adds discovered devices to its database
- Provides both management of configuration and monitoring.
- Provides a single configuration point for digital video headend products, simplified fault detection, smart alarm management, and an interface to integrated network management systems.[1]

Continuous development is going on in SDM to provide support for new devices and full support for the older devices.In order to test the performance of SDM after doing any modification, We need to communicate with real devices again and again.

To avoid the problem of communicating with real devices, a device simulator is developed which provides a virtual network environment. This simulator is termed as JDS(Java Device Simulator).It is mainly used for testing of SDM.Instead of communicating with real devices, SDM deals with virtual devices present in JDS.

1.3 Thesis Outline

The rest of the thesis is organized as follows:

Chapter 2 (*Literature Survey*) In this chapter, fundamentals of SmartStream Device Manager are described.Significance of SDM and JDS in headend facility is discussed.Issues and challenges present in JDS are discussed. Protocols used by SDM for communication with device are described in detail.

Chapter 3 (*Technical Specifications*) This chapter contains technical requirements of SDM and JDS.

Chapter 4 (*Headend Devices*) This chapter describes the headend devices. It contains a brief introduction about each device and how they are connected with each other.

Chapter 5 (*Objectives*) This chapter explains the main objective of this thesis work. It contains a detailed explanation about all the issues present in JDS and SDM testing.

Chapter 6 (*Implementation*) This chapter describes all the techniques and approaches which are used to resolve JDS issues. Each technique is described in detail along with its outcome. SDM test automation process is explained in this chapter.

Chapter 7 (*GUI For Simulator*) Working of JDS GUI is explained in this chapter. It shows the process of creating virtual headend device using GUI.

Chapter 8 (*Conclusion And Future Scope*) This chapter includes the major conclusion of this project work.Future directions of work in this project is also outlined in this chapter.

Chapter 2

Literature Survey

2.1 Motivation :-

SDM communicates with a devices using SNMP in order to manage that device.It interacts with the database of that device to fetch some values and to set new values in the device.So the key point is, SDM does not require the hardware part of the device.It only needs the database of device.We can fetch the database of each device using SNMP commands and this database can be stored in a text format. After that we can use this database as a virtual device,so SDM can directly communicate with this database instead of communicating with real device.We can combine all these database to a single file and create an application in Java,termed as device simulator, which will present the data of all devices to SDM on behalf to real devices.

Although JDS provides a virtual copy of each device to SDM, it does not provide full support for all devices.For some devices,it crashes the JVM and not giving the desired output.The aim is to optimise JDS to make it fully functional for SDM. Apart from this, proper packaging of JDS is required to use it as a project.

SDM testing is required to check the functionality of SDM in real headend environment. Manual testing is performed in order to test all features of SDM.Manual testing requires lot of time and man power. These tasks can be automated to save the time.

2.2 Significance :-

SDM requires headend devices in order to test its performance and functionality.There are very few real devices present in the testing lab.So if one person is using a device in order to test SDM,other person has to wait for the test to complete. After that he can use the same device.This problem may lead to longer testing time of SDM.JDS overcomes this problem. It provides a virtual device to SDM to test its performance and other updates. So no real device is required and we can have as many virtual devices of single type as we want to test the SDM.This reduces the cost of SDM testing.

After optimization,JDS supports all the devices including APEX 3000(device with large database).Now SDM can communicate with virtual APEX device without any delay and can access the complete database of APEX 3000.

JDS GUI desgin provides ease of access to the tester. By using GUI , tester can easily set up virtual headend environment. They need not to remember the shell script names for each and every operation. This GUI is not dependent on any shell scripts or java classes, It is a stand alone JAR file which can invoke JDS without using any external resource.

SDM test automation is intended to save time for doing repetitive tasks for testing purpose. For SDM feature's testing we can provide inputs from a text file instead of manually entering them.

2.3 Protocols Used By SDM :-

There should be some medium or language, by which SDM can talk to network devices.There are few network protocols which are used by SDM to communicate with devices.Both SDM and headend device should be configured with those protocols for successful communication.In order to understand the basic functionality of SDM and JDS, knowledge of these protocols is required.

2.3.1 SNMP

Simple Network Management Protocol (SNMP) is an Internet-standard protocol which collects and organise information about managed devices on IP networks.

In general, a network being profiled by SNMP will mainly consist of devices containing SNMP agents. An agent is a program that can gather information about a piece of hardware, organize it into predefined entries, and respond to queries using the SNMP protocol.The component of this model that queries agents for information is called an SNMP manager. These machines generally have data about all of the SNMP-enabled devices in their network and can issue requests to gather information and set certain properties.[2]

- Client listens on port 161.
- Server listens on port 162.
- An SNMP-managed network consists of three key components:
  * Managed device
  * Agent software which runs on managed devices
  * Network management station (NMS) software which runs on the manager
- Every information that can be queried through SNMP is looked in terms of an object. Example- IP address and MAC address of a device.
- Every object has an object ID or OID which is unique for every object.
- Collection of managed objects is known as Management Information Base (MIB). MIB works as databse for managed device.
- A management station controls the network by setting values in the MIBs of the various agents

2.3.1.1 MIB

Management Information base is the collection of objects present in a device.These objects are managed by Snmp manager.In other words MIB is the database of SNMP enabled devices.

2.3.1.2 SNMP Commands[3]

1. snmpget- To retrieve data from remote device using snmp agent using IP address, authentication information and OID.

snmpget -v2c -c public 10.10.10.1 .1.3.6.1.2.1.1.1.0

2. snmpset- This command is used to set value of a specific parameter present in the remote host. We can set IP address,MAC address any many other parameters by using this command.

snmpset -v2c -c public 10.10.10.1 .1.3.6.1.2.1.1.1.0 a "192.168.70.1"

3. snmpwalk- Using this command we can get value of all the parameters present in the MIB of remote host.

snmpwalk -v2c -c public 10.10.10.1

2.3.2 BOOTP

When a device is turned up, it needs to know its IP address in order to effectively communicate in the network. Some small devices are not having a database to store the IP address,they need to get the IP address dynamically from the server.
BOOTP is a protocol used to assign IP address to network devices.It is also used by devices to download configuration files from the server.

- Port number 67 is used by the server to receive client requests
- Port number 68 is used by the client to receive server responses.
- Client prepares a request message
- Client broadcasts the request to 255.255.255.255
- Reply Message is created by the server
- Server sends the reply.
- Client processes reply.

Chapter 3

Technical Specifications

3.1 Java Device Simulator

Front-end	Java
Database	Apache Derby 10.11.1.1
ORM Tool	Hibernate 5.2.8
Operating System	Linux,Ubuntu
Scripting Language	Bash(shell)

Table 3.1: Jds Software Specifications

3.2 SmartStream Device Manager

Hardware Specifications: The SDM software shipped on an HP DL360 G8 server

Intel Xeon E5-2620 Processor (2.0Ghz/6-core/15MB/95W)
8GB DDR3-1600 memory
300GB HDD
DVD RW

Table 3.2: SDM Specifications

Chapter 4

Headend Devices

4.1 List of Devices

Here are few headend devices which are managed by SDM:

- **INTEGRATED RECEIVER TRANSCODER(IRT)- Converts 950-1450 MHz signals to 6MHz signals for transmission.**
- **RF Combiner- Combines multiple input streams from headend devices onto a single RF output.**
- **Digital Addressable Controller (DAC) - It controls the headend equipments and DCTs. It authorizes DCT for services and collects purchases from two-way set-top terminals.**
- **Network Controller- Acts as an IP gateway between two networks. It is used for interactive communication such as Video-On-Demand.**
- **Billing/Business System -A computer system that has a database of subscribers, their services and their set-tops.It generates subscriber bills and it is connected to DAC.It sends instructions to the DAC for it to execute.**
- **Digital Consumer Terminal(DCT)- It decompresses and decrypts digital signals and presents them for viewing. It can be one-way or two-way.**
- **Out-of-Band Modulator -It converts the control data digital input signal to an RF output signal that is transmitted to customer DCTs over an**

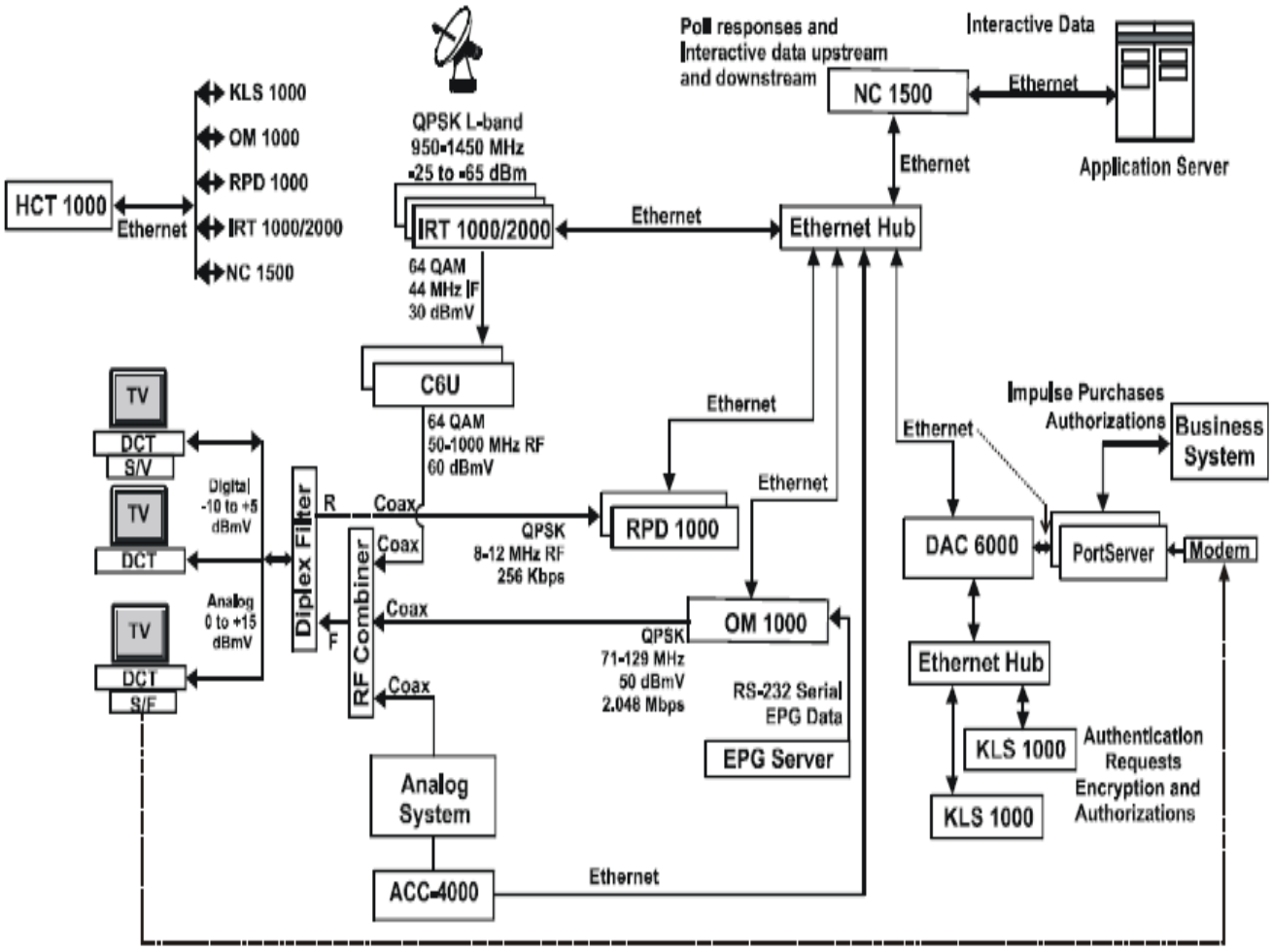

Figure 4.1: Headend signal flow

out-of-band cable(OOB). The OOB data stream can also include other type of information, such as downloadable objects.

- **APEX 3000 - This device is used to encrypt the signals received from IRT and send them to Set Top Box.**

Chapter 5

Objectives

5.1 Objectives

- **Providing support for APEX 3000 in JDS**
  -JDS becomes non responsive after adding one or more APEX 3000 devices.When SDM sends an SNMP request to the JDS,it is not able to process that request and after few seconds it stops working.

- **Reducing database insertion time in JDS**
  -When we add a device to JDS, we need to add its firmware to database and after that we add the proto file in database for each device.To create a test environment,we need to add these devices many times based on the test cases.JDS takes 14-15 minutes to add single APEX 3000 device.The aim is to lower down the insertion time to less than 2 minutes for single APEX device.

- **Designing GUI for JDS**
  -There is no proper packaging of JDS code.JDS uses shell scripts to invoke java class file to add devices in DB and to create virtual interface for each device.There is a need of creating GUI of JDS to provide a user friendly setup procedure to testers.

- **Implementing Hibernate in JDS**
  -Hibernate is an ORM (Object relational mapping) tool.By using hibernate, we can write database independent queries.In future ,if we want

to change underlying database then we can do it very easily. We need to change few lines of code in configuration file and we can use the new database without any problem.

- **Design and automate SDM test cases -Instead of testing SDM with dedicated people,we can automate the task of testing using some bash scripts.This will make the testing easier and faster.**

Chapter 6

Implementation

6.1 Providing Support for APEX device

Every Java application starts in a separate JVM. This JVM is allocated some RAM space to store objects and class files generated by java application. When we start JDS it fetches the device database from derby(Jds DB) and creates objects for all the parameters present in the database. These objects are stored in the heap area of JVM.

Initially JDS is allocated 640 Mb of RAM. When we have devices with large database, heap space can not accommodate all the objects of that device and it gives exception. The exception thrown by JDS is:

```
Exception in thread "DefaultUDPTransportMapping_192.168.81.84/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
        at java.net.PlainDatagramSocketImpl.receive0(Native Method)
        at java.net.PlainDatagramSocketImpl.receive(PlainDatagramSocketImpl.java:136)
        at java.net.DatagramSocket.receive(DatagramSocket.java:725)
        at org.snmp4j.transport.DefaultUdpTransportMapping$ListenThread.run(Unknown Source)
        at java.lang.Thread.run(Thread.java:619)
        at org.snmp4j.util.DefaultThreadFactory$WorkerThread.run(Unknown Source)
Exception in thread "DefaultUDPTransportMapping_192.168.79.33/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.78.52/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.79.237/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "DefaultUDPTransportMapping_192.168.81.108/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.80.62/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "DefaultUDPTransportMapping_192.168.78.197/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.78.253/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "DefaultUDPTransportMapping_192.168.78.58/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.79.86/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.80.72/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "DefaultUDPTransportMapping_192.168.80.131/161" java.lang.OutOfMemoryError: Java heap space
Exception in thread "DefaultUDPTransportMapping_192.168.78.162/161" java.lang.OutOfMemoryError: GC overhead limit exceeded
Exception in thread "DefaultUDPTransportMapping_192.168.78.89/161" java.lang.OutOfMemoryError: Java heap space
```

Figure 6.1: Exception Thrown By JDS

As soon as we start JDS, It starts creating objects and because of this configuration(640 Mb RAM for JVM), heap area overflows and it crashes the JVM. When SDM sends SNMP requests to JDS, no response is given by JDS.There is no space available to create transport mapping object for device and without transport mapping,snmp requests can not be processed by the device.

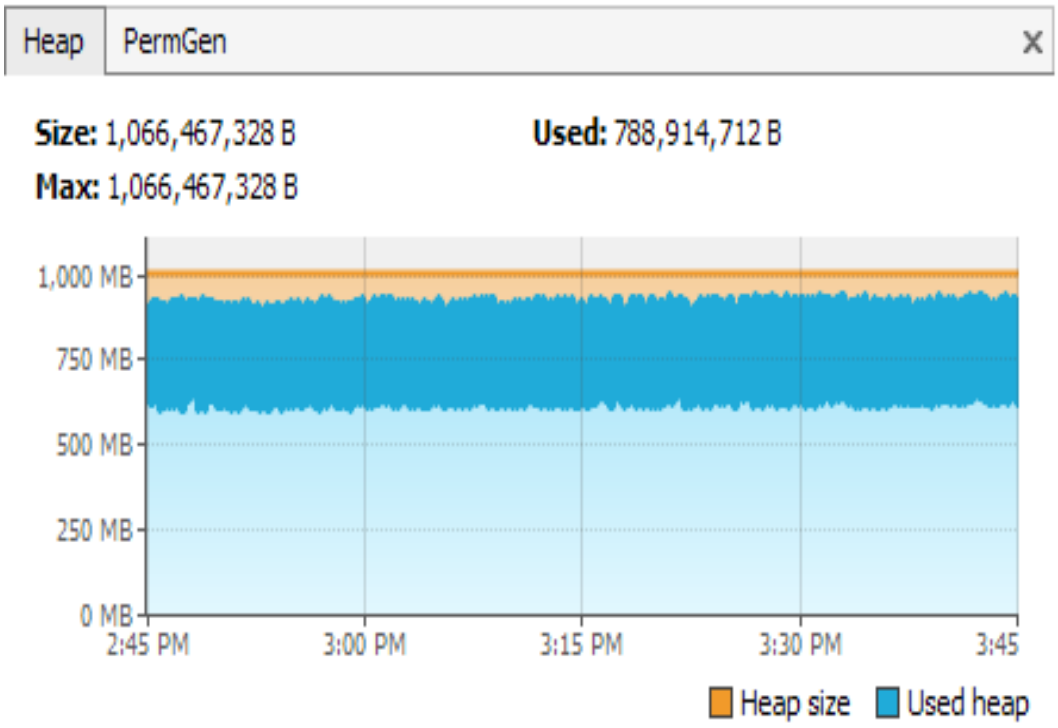

Figure 6.2: Heap Space Used By Jds

To support large devices, we need to increase the RAM allocated to JDS. Instead of 640 Mb RAM,if we give 1024Mb RAM to JDS, it can accommodate all the objects created by jds class files.
After changing RAM configuration, JDS works fine for APEX 3k devices. Now it can support more than 100 APEX 3k devices at a time.

6.2 Reducing Database insertion time for APEX 3k

The process of creating a test environment for SDM is:

- **Add proto(MIB) of device to database.This process will add all the parameter OIDs to database table.This does not store OID values.**
- **After that, Add device to database,which will use the same proto file but it will store OID values along with the protoid and ip address of the device.**
- **When all the devices are added to database,we start JDS.**

All these steps take some time to add MIB to database. Each object ID along with its type and value is treated as single row in database table.
APEX 3k is having nearly 18 lakh rows and it takes 14 minutes to add proto file to database.After that adding single APEX 3k takes 14 minutes.
For basic testing we need more than 130 apex devices in simulated environment.To add 130 devices to database tester has to wait for nearly 30 hours,then only he can start testing.Aim of this task is to reduce the time from 14 minutes to less than 1 minute for APEX 3k device.

```
AddProtoProcessor.addProto: Attempting to add APEX 3000 2.8.4 using protos/apex3000_2.8.4_proto
AddProtoProcessor.addProto: Added APEX 3000 2.8.4 using protos/apex3000_2.8.4_proto
AddProtoProcessor.addProto: Processing time 843.689 sec
```

Figure 6.3: Initially Time taken by JDS to add APEX 3k

6.2.1 Approach 1

First focus was on the java code which deals with database insertion.Each parameter of device is inserted as a separate row and each row is inserted using separate prepared statement.Before executing the prepared statement,java class file needs to acquire connection from database.In initial configuration of jds,each prepared statement commits individually as soon as it completes its execution.
The reason behind this is : auto commit mode of derby database. By default,

Auto commit mode is enabled in Derby database and because of this it commits every prepared statement individually. This leads to time taking inserts in database.[4]

To reduce the insertion time in derby database,we need to disable the auto commit mode. After disabling auto commit mode,we can commit more than one prepared statement in single commit operation.We need to commit the operation by issuing commit() method.[5]
The Approach is "Committing a batch of prepared statements instead of one prepared statement at a time". For testing of this approach,initially I have taken a batch of 100 prepared statements.Commit operation was performed after execution of these 100 prepared statements. This gives a significant reduction in insertion time.Then 500 prepared statements were taken in one batch, the insertion time was less compared to the batch of 100 statements. While we commit 1000 statements together in a single batch, insertion time is reduced to half with compared to initial time.A batch of more than 1000 prepared statements gives same result. So we can say that minimum 1000 statements need to be commit together in order to reduce the insertion time of JDS database.Now the time taken by JDS to add APEX 3k proto file is 7 minutes and to add a single APEX device,it takes 6 minutes.

```
[root@SDM25 jds-1.1]# bash ./addProto.sh apex 3000 2.8.4 protos/apex3000_2.8.4_proto
AddProtoProcessor.addProto: Attempting to add apex 3000 2.8.4 using protos/apex3000_2.8.4_proto
AddProtoProcessor.addProto: Added apex 3000 2.8.4 using protos/apex3000_2.8.4_proto
AddProtoProcessor.addProto: Processing time 422.889 sec
[root@SDM25 jds-1.1]# bash ./addDeviceBlock.sh apex 3000 2.8.4 192.168.78.1 192.168.78.4 2 protos/apex3000_2.8.4_proto
AddDeviceBlockProcessor.main: See results by opening another shell and typing: tail -f jds.log
Attempting to add apex 3000 2.8.4 192.168.78.1 -> 192.168.78.4 v2 eth1 protos/apex3000_2.8.4_proto
Attempting to add apex 3000 2.8.4 192.168.78.1 v2 eth1 protos/apex3000_2.8.4_proto
AddDeviceProcessor.addDevice: Added apex 3000 192.168.78.1 on eth1
AddDeviceProcessor.addDevice: Processing time 306.806 sec
Attempting to add apex 3000 2.8.4 192.168.78.2 v2 eth1 protos/apex3000_2.8.4_proto
AddDeviceProcessor.addDevice: Added apex 3000 192.168.78.2 on eth1
AddDeviceProcessor.addDevice: Processing time 292.151 sec
Attempting to add apex 3000 2.8.4 192.168.78.3 v2 eth1 protos/apex3000_2.8.4_proto
AddDeviceProcessor.addDevice: Added apex 3000 192.168.78.3 on eth1
AddDeviceProcessor.addDevice: Processing time 290.857 sec
Attempting to add apex 3000 2.8.4 192.168.78.4 v2 eth1 protos/apex3000_2.8.4_proto
AddDeviceProcessor.addDevice: Added apex 3000 192.168.78.4 on eth1
AddDeviceProcessor.addDevice: Processing time 291.685 sec
AddDeviceBlockProcessor.addDeviceBlock: Added 4 devices, skipped 0, Errors 0; TotalTime 1181.609 secs
```

Figure 6.4: Time taken by JDS after Approach 1

6.2.2 Approach 2

Snmpwalk on APEX device gives 18 lakh object IDs.When we store the proto file in database, these 18 lakh OIDs are stored in the database as 18 lakh rows. But many of these OIDs are related to performance statistics,which are not used by SDM. After doing analysis of SDM and APEX communication,I come to know that SDM does not require all the parameters present in the proto file.It only deals with 5 lakh parameters,rest of the 12 lakh parameters are never used by SDM.

Adding these extra 12 lakh parameters does not serve any purpose.We can ignore all the extra parameters from the proto file and can store only 5 lakh essential parameters to JDS database.This reduces the database insertion time to one third of the time which was achieved by Approach 1.

```
[root@SDM25 jds-1.1]# bash ./addProto.sh apex 3000 2.8.4 protos/apex3000_new.proto
AddProtoProcessor.addProto: Attempting to add apex 3000 2.8.4 using protos/apex3000_new.proto
AddProtoProcessor.addProto: Added apex 3000 2.8.4 using protos/apex3000_new.proto
AddProtoProcessor.addProto: Processing time 125.953 sec
```

Figure 6.5: Time taken by JDS after Approach 2

6.2.3 Approach 3

After applying the modification discussed in approach 2, we are adding only 5 lakh rows to JDS database for a single apex device.So now the process will be:

- **Add proto file with 5 lakh entries.**
- **Then add device parameters which consists of 5 lakh entries.**

Here all the 5 lakh entries which are present in the proto file, are copied to device parameter table with device IP address and values for those parameters. So basically all the entries are stored multiple time with different IP address. The scenario is :

Each proto file stores 5 lakh parameter OID to database.

After this, when we add a single apex device,same parameter OIDs are getting

stored in the database with device IP as unique field for those parameters.But these parameters with values are stored in a separate table.So those 5 lakh entries are repeated for every apex device.
This was the motivation behind finding a new approach to store apex device in database. If we avoid the redundant entries, which are same for every apex device, we can achieve a significant reduction in insertion time of device.
The first step for this approach was to find out the parameters which are unique for every apex device.This task was done with the help of basic configuration files available for APEX and SDM. After processing those files I found out that only 18 parameters are device specific in the proto file, rest are common for every device.

So instead of adding all APEX devices with 5 lakh entries, we can have a template device, which will have all these entries,and for rest of the device, we can store only 18 entries. After doing this modification, JDS working will be changed:

When SDM quraies any parameter from device, if that parameter is present in the device specific parameters,it will be returned from there.Otherwise it will be returned from template device.
This approach has reduced the database size as well, because adding an APEX device is adding only 18 rows,instead of 18 lakh rows.

```
[root@SDM25 jds-1.1]# bash ./addProto.sh apex 3000 2.8.4 protos/apex3000_new.proto
AddProtoProcessor.addProto: Attempting to add apex 3000 2.8.4 using protos/apex3000_new.proto
AddProtoProcessor.addProto: Added apex 3000 2.8.4 using protos/apex3000_new.proto
AddProtoProcessor.addProto: Processing time 125.953 sec
Attempting to add apex 3000 2.8.4 Template apex3000-2.8.4 v2 eth1 protos/apex3000_new.proto
AddDeviceProcessor.addDevice: Added apex 3000 Template apex3000-2.8.4 on eth1
AddDeviceProcessor.addDevice: Processing time 91.135 sec
```

Figure 6.6: Time taken by JDS after Approach 3

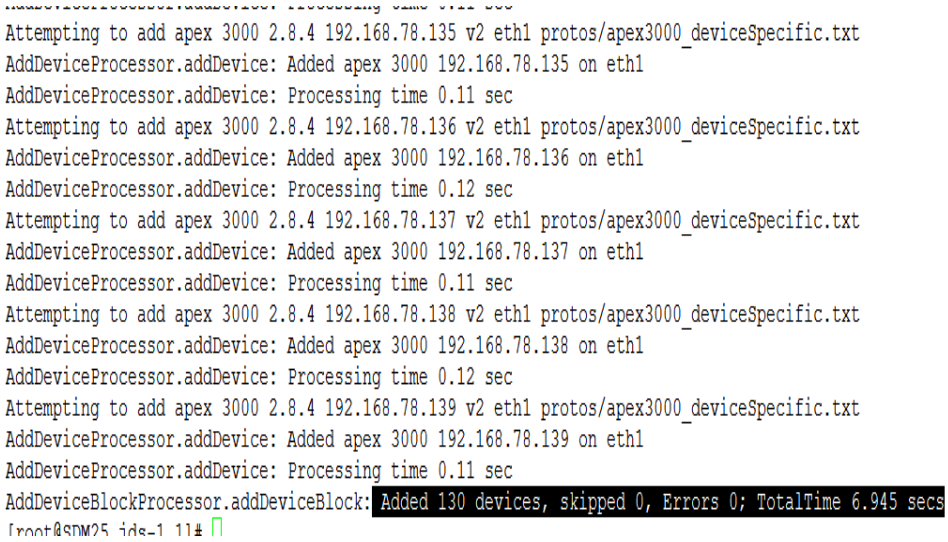

Figure 6.7: Time taken for APEX Device

6.2.4 Results

Time taken by JDS to add proto and device to database:

$$Total\ Time = Time\ required\ to\ add\ Proto\ +\ Time\ required\ to\ add\ APEX\ device \tag{6.1}$$

	For Proto	For One APEX	For 130 APEX	Total Time
Initially	14 min	14 min	1820 min	1834 (30 hour)
Approach 1	7 min	6 min	780 min	787 (13 hour)
Approach 2	2 min	2 min	260 min	262 (4 hour)
Approach 3	4 min	0.11 sec	7 sec	4 min

Table 6.1: Result after every Approach

Note:- min=minutes , sec=seconds

6.3 Implementing Hibernate in JDS

Hibernate is an ORM tool, which provides object relational mapping for java. It gives a framework which maps java classes to database tables.

Hibernate's primary feature is mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.[6] To use hibernate framework , we need to add jar files of hibernate in our classpath. Steps to include hibernate in JDS are as follows :

- **Create a configuration file named as hibernate.cfg.xml, this xml contains all the database related properties.Some basic properties included in this file are :**
  1. **Database connection url**
  2. **Database connection driver class**
  3. **Database username**
  4. **Database password**
  5. **Sql Dialect**
- **Create DTO(Data Transfer Object) classes**
- **Create DAO (Data Access Object) classes**
- **Use hql(Hibernate query language) instead of sql, to avoid database dependency. Hibernate will automatically convert the hql into the required format for the underlying database.**

6.3.1 Performance Improvements in JDS

Following are some performance improvements in JDS after implementing Hibernate:

- **We need not perform separate operations to create database schema , hibernate will take care of this.**

- **We can write database independent queries so that in future we can change underlying database without changing the queries.**
- **Line of code is reduced in JDS.**
- **Database insertion time is reduced for apex device.**
- **Database retrieval becomes faster.**

	For Proto	**For One APEX**	**For 130 APEX**	**Total Time**
Using Hibernate	1 min	0.1 sec	8 sec	1 min 8 sec

Table 6.2: Result after Hibernate Implementation

Note:- min=minutes , sec=seconds

6.4 SDM test automation

6.4.1 Providing support for input parameter file in runConfigTester

- For SDM performance testing, an utility is present in SDM.
- Tester can test all the functionalities by giving some input parameters to that utility.
- This provides some options to the tester, to perform a specific task.

```
[root@sdm20 scripts]# sh runSDMConfigTester

Select operation to test:
        0. Exit
        1. Set device configuration values
        2. Retrieve device configuration values
        3. Retrieve device configuration definitions
        4. Modify device firmware
        5. Add reload flag in FOF file
        6. Query parameters across devices
        7. Delete device
        8. Refresh device
        9. Add device
        10. Control log level
        11. Parse INI File
        12. Get Server Status
        13. Discover devices
        14. Discovery Status
        15. Clear Headend Device Caches
        16. Reboot device
        17. Clear Extended Data Object Cache
        18. Ops Mgr test
        19. CSV Param file to device
        20. Device to CSV Param file
        21. Generate device parameter report
        22. Get Device Info
        23. Dump and log internal queues
        24. Northbound Configuration
        25. Management Groups
        26. Authorize Entitlement Features
        27. Status Poll Operations
        28. SDM Configuration
        29. Interactive Groups Info
        30. Synchronize Interactive Groups
        31. CAST Support

        "s" Toggles in and out of silent mode.
Enter:
```

Figure 6.8: Config. Tester for SDM

Problems with this tool:

- User can perform one operation at a time.
- User needs to provide all the parameters again again even if he wants to test the same functionality more than one time.
- It does not provide any detailed report about the statistics of operations performed by sdm.

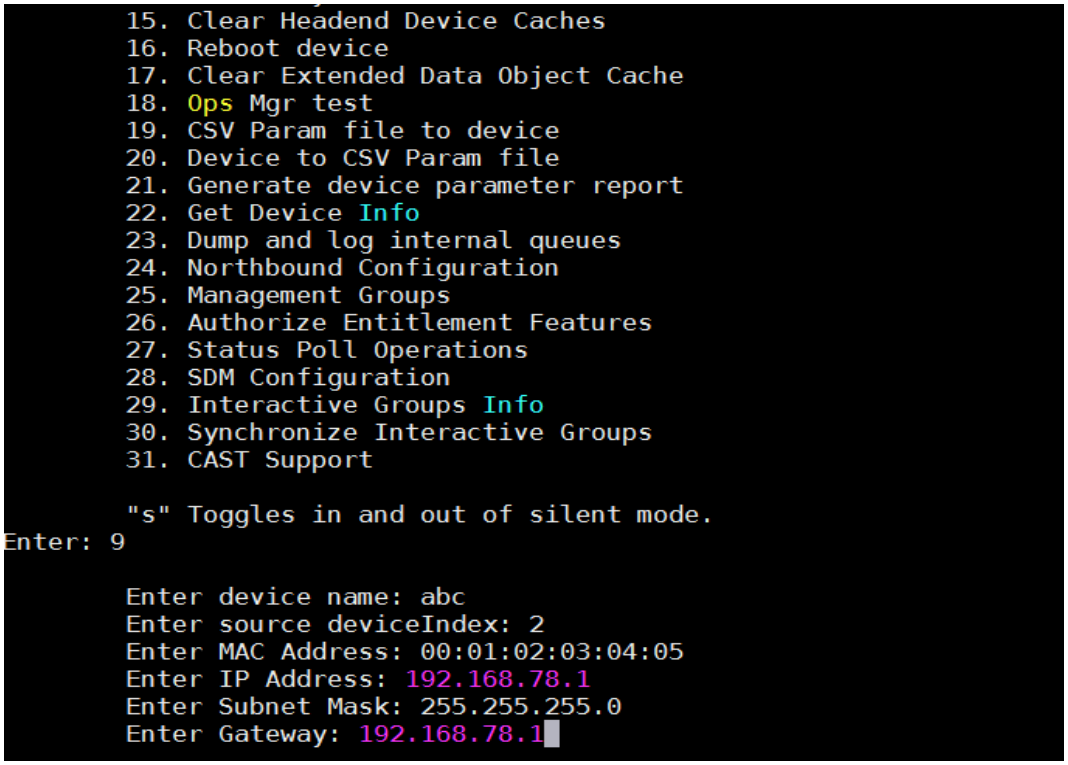

Figure 6.9: Add Device to SDM

Modifications :-

- **For each operation , created a text file which contains input parameters.**
- **Each file can conatin multiple set of input parameters.**
- **If we want to use this file as input to all operations, we just need to pass the name of file as an argument to the script.**
- **The code will parse the input file and will perform the required operation by taking parameters from input file.**
- **After completion of all the operations from text, it will call a script, which will generate a report from the SDM database.**

```
#
# Input script to test adding a device instance to the SDM.
# Date: 10/03/2017
#

#private static final String DEV_NAME = "name";
#private static final String FW_TYPE = "type";
#private static final String FW_MODEL = "model";
#private static final String FW_VERSION= "version";
#private static final String IP_ADD = "ipadd";
#private static final String MAC_ADD = "macadd";
#private static final String SUBNET = "subnet";
#private static final String GATEWAY = "gateway";


name = "tst1", type = "om", model = "2000", version = "1.3.1", macadd = "ab:23:cd:23:ef:35", ipadd = "193.1.17.100",
name = "tst2", type = "om", model = "2000", version = "1.3.1", macadd = "ab:23:cd:23:ef:34", ipadd = "193.1.17.101",
name = "tst3", type = "om", model = "2000", version = "1.3.1", macadd = "ab:23:cd:23:ef:33", ipadd = "193.1.17.102",
```

Figure 6.10: Input file for Add Device

date	operation	object_name	avg_time
2017-03-08	Create Device Backup	192.168.30.160	00:01:38.011
2017-03-08	Create Device Backup	192.168.30.161	00:01:04.029348
2017-03-07	Create Device Backup	192.168.30.160	00:01:13.469
2017-03-07	Create Device Backup	192.168.30.161	00:00:05.965667
2017-03-06	Delete Firmware	APEX 2.7.1	
2017-03-06	Load Firmware	APEX 2.7.1	00:00:45.521
2017-03-06	Load Firmware	ARPD 2.1.8	00:00:10.534
2017-03-06	Load Firmware	OM 1.2.2	00:00:12.546

(8 rows)

Figure 6.11: Report of SDM operations

Chapter 7

GUI For Simulator

JDS code is present in the from of separate class files.All the libraries required for JDS are present in a separate folder.There are some bash scripts present which are used to :

- **Create JDS database**
- **Add firmware to database**
- **Add Device to database**
- **Remove device from database**
- **Start the JDS**
- **Stop the JDS**

These scripts internally invokes the java class files in order to perform a specific operation.Testers need to run all these scripts one by one in order to create a test environment.

This process is time taking and it requires more efforts from testing team.To avoid this long process of setting up test environment, I developed JDS GUI.This provides a very easy way to add devices in database and start the JDS.

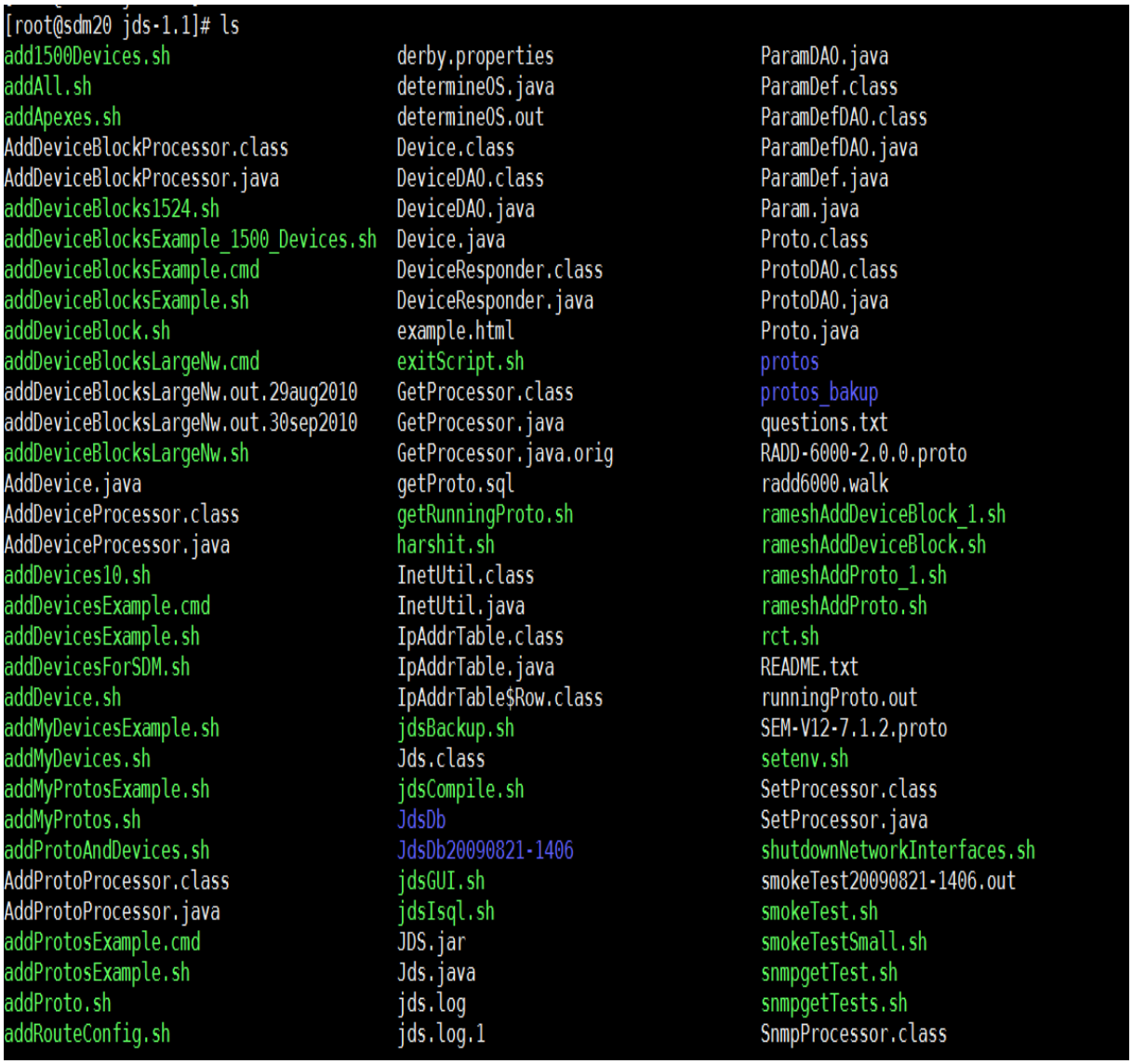

Figure 7.1: JDS Without GUI

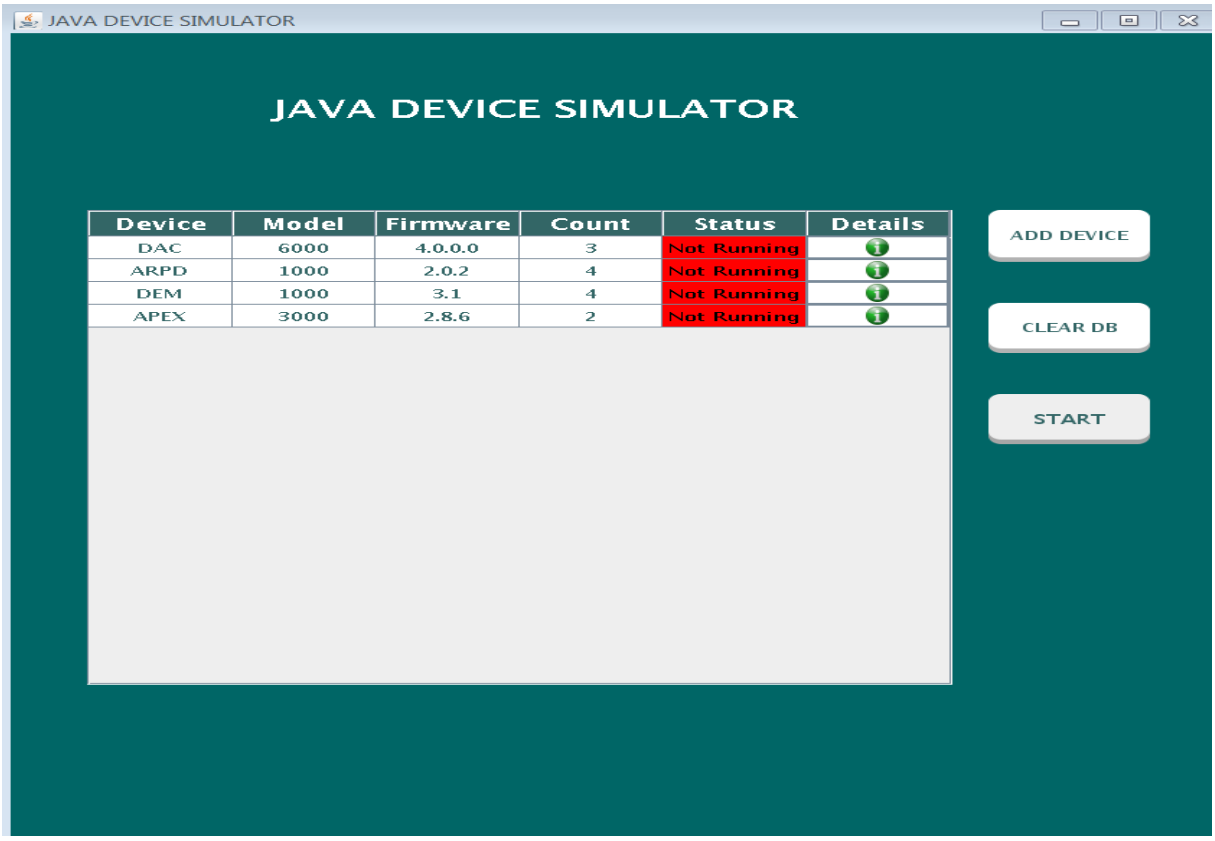

Figure 7.2: JDS GUI

Details about each section of GUI :-

- **Add Device :- This button will open a new frame which contains options for adding a device.**
- **Clear DB :- This button will clear the JDS Database and recreates the schema.**
- **Start/Stop :- This button starts and stops the simulator.**

Figure 7.3: JDS GUI : Adding Device

This screen contains 3 panels:

- **Device Details : Here we need to enter type ,model ,firmware ,total number of device and starting IP of device block. It will automatically calculate ending IP address.**
- **Network Interface : Here we need to select the interface on which we want to run the virtual device.**
- **Firmware file : We need to select the appropriate firmware for the device.**

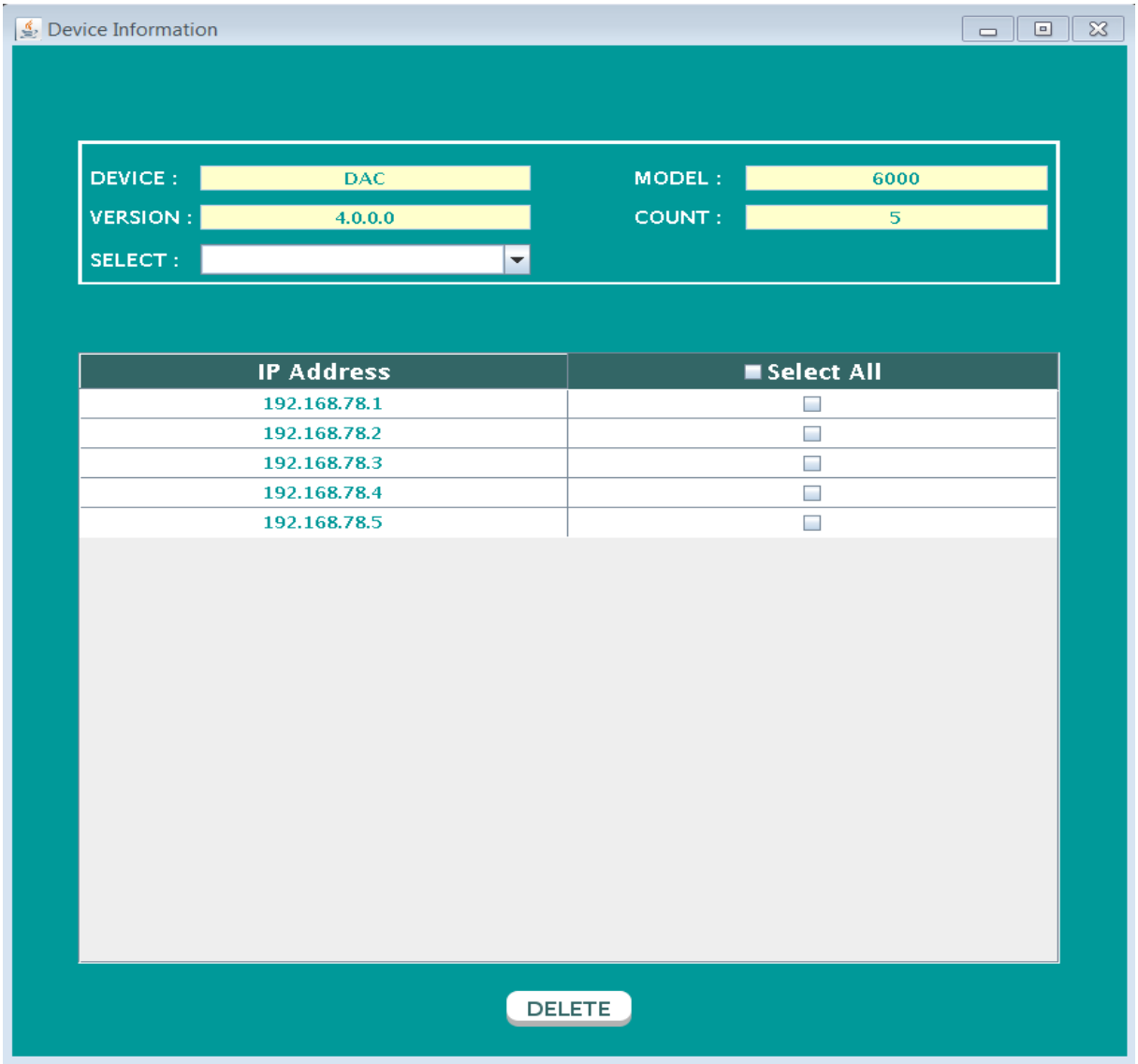

Figure 7.4: JDS GUI : Device Details

When we click on the details button on the main screen of GUI, it will redirect us to this frame. This frame gives the list of all IP address associated with a particular device type. We can delete device using delete button.

Chapter 8

Conclusion And Future Scope

8.1 Conclusion

After completion of this project, JDS is able to provide full support for all the headend devices. Insertion time for JDS database is also reduced by a significant amount. Because of all these improvements, JDS can provide a fully functional virtual network setup for SDM testing.

By the use of hibernate, JDS DAO classes became database independent.We can switch to a new database with minimal changes in the existing code.

Now we can run JDS with a single JAR file. Overhead of shell scripts and external java classes is removed by this JAR.

JDS is tested with 1500 virtual devices after these modifications and it is working perfectly. It responds to all the snmp requests send by SDM.

Some of the SDM test cases are automated, so we can perform testing in automated way.

Because of the modifications in runConfigTester, testers can trigger more than one operation at a time in SDM.A detailed report will be available after completion of testing, which can be useful in performance measurement of SDM.

8.2 Future Scope

SDM testing has many areas which can be automated.In future we can design more number of test cases for SDM testing. JDS can simulate all the headend

devices, but simulated device does not provide all the features which can be achieved by a real device.We can not send traps from a simulated device.JDS can be improved further to provide trap processing and alarms to SDM.

Bibliography

[1] **M. G. B. Information, "Smartstream device manager(sdm)," 2014.**

[2] **J. Ellingwood, "An Introduction to SNMP (Simple Network Management Protocol)."** https://www.digitalocean.com/community/tutorials/an-introduction-to-snmp-simple-network-management-protocol**, 2014.**

[3] **W. Stallings, "Snmp and snmpv2: the infrastructure for network management,"** ***IEEE Communications Magazine*****, vol. 36, no. 3, pp. 37 – 43, 2002.**

[4] **H. Group, "Performance."** http://www.h2database.com/html/performance.html**, 2014.**

[5] **A. S. Foundation, "Tuning Derby."** https://db.apache.org/derby/docs/10.5/tuning/tuningderby.pdf**, 2009.**

[6] **J. Developer, "Hibernate."** http://hibernate.org/**, 2003.**