ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

Virtualized Tool for Service Performance Validation of Middleware Components Used In Telco Network Elements

Likhith Kumar .T.R[#1], Manoj Thomas[*2], Roja Reddy B[#3]

M. Tech Student [DC], Dept. of Digital Communication, R.V College of Engineering, Bangalore, India[#1]

Line Manager, Nokia Networks, R&D Technology Centre, Bangalore, India[*2]

Assistant Professor, Dept. of Digital Communication, R.V College of Engineering, Bangalore, India[#3]

ABSTRACT*:* Middleware is currently used in all the customized software systems in order to felicitate the applications to majorly work on their functionality not on to much of process, memory management, load Analysis, counter, Timers, Database [DB], fault detection & correction and other such constraints which are very important in building the network machine software's like what we use in 3G and 4G networks. (i.e. in IMS, IP Multimedia Subsystems). The Middleware of IMS will provide API's [Application Program Interfaces] which will be used by the applications to perform their set of functionality. The middleware also provides platform for running 3rd party application software that support the functionality of the other applications on the upper layer of the stack. So this middleware APIs have to be checked for their functionality in the maintenance mode after every new feature addition, deletion or modification of the existing ones. So Regression Testing has to be done to validate the working of the APIs. Since these APIs are very important for application to work fine without any malfunctioning. This paper presents the variant of the testing technique which was followed previously in manual terms by replacing it with automated software tool set which reduces the time, error factor prior to every software release, making the network software application error free and work in a much effective manner.

KEYWORDS*:* Middleware platform, feature, service, virtualization, APIs [Application Program Interfaces], SUT [System under Test].

I. INTRODUCTION

The software or any application to work, should have a platform which can provide services for the applications by providing basic fault control, error handling. The OS [Operating System] itself is a platform many times for more general applications which we use on our daily basic but for some robust heavily loaded applications running over internet [1]; the OS alone can't handle the software working properly which is having its own limitation. In the robust telecommunication networks the platform should be well configured to detect the faults caused by the applications running on it and provide fast recovery from those with a limited manual intervention. The Telco platform should have the capability to handle the hardware and the software resources much efficiently in order to support the applications running on it.

Limitations of OS

1. OS can't support customized timers, counters and access to DB for some standalone special applications which requires a layer between the Application and OS to provide the above requirements.
2. The error, fault handling capability of the OS is very limited for the applications.
3. The automatic alerting mechanism is not provided when there is any fault with the hardware or software part of the application.
4. The OS can monitor the memory and DB Utilization only to some extent above which it requires another powerful mechanism over that to perform the function much effectively [1].

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

5. High Availability of the Hardware and software cannot be achieved just by an OS, added software intelligence should be there to detect the malfunctioning of the machine and switch over to a redundant system until the primary system comes back to operation mode [2].
6. 3rd party applications running on the Telco machines will perform huge data traffic communication which may lead to lot of resource overhead, leading to system lockup. The OS alone doesn't have that intelligence for effective resource allocation in improving the system performance.

The Figure 1 gives the information about the layer distribution of the network machine stack compared with respect to the OSI reference model [3]. The layers of the middleware stack are basically in sync with the different layers of the OSI model. There will be interfaces between these layers to communicate within those layers and in between the two network machines using this layered architecture.

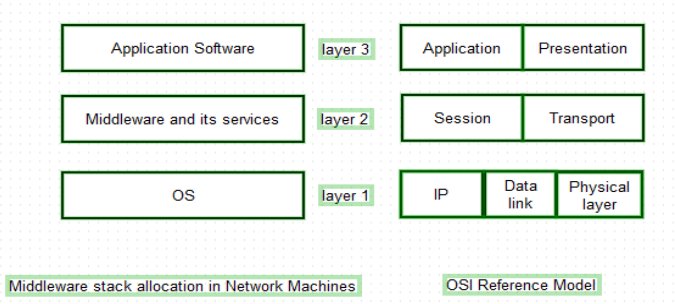

Figure 1: Middleware stack allocation in the network machines compared with respect to OSI reference model.

The Table 1 gives the information about the protocols which are used at the different layers of the network machine stack which is compared with OSI reference model as a standard [3]. The OS itself handles till the extent of network layer by bundling the different types of packets coming from upper layers and routing these packet streams over the network to the proper destination. The middleware will support the applications in properly handling these packets by the application softwares.

Table 1: Layer wise protocol distribution in the Network Machines with OSI as the reference model

Layer name	Protocols used	Location on the Network machine stack
Physical	Ethernet, SLAN	Redhat Linux OS
Data link	MAC,LLC	Redhat Linux OS
IP	TCP-IP, UDP, SCTP	Redhat Linux OS
Transport	SCCP, ISUP	Middleware Services
Session	SS7, SIP	Middleware Services
Presentation & Application	MAP, FTP,SSH,CAMEL	Application Software

Advantages of the middleware over the OS:

The middleware overcomes the disadvantages of the OS in following ways [4]:

- Middleware speeds up the OS communication.
- It provides interfaces between the processes for IPC.
- It helps in handling the events and alarms if there is any change in the other middleware services and also the changes from the application level.
- It helps in fault detection and correction.
- It provides a Man Machine interface to check for the internal communication of the subsystems and their processes.
- It works as a runtime environment where making the applications to work smoother on their functionality by handling the load of process and load management.
- It helps in DB management by updating the traces of the process level and the hardware level changes and their impact on the Application Software performance.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

- It provides an interface to monitor the processes and their connectivity.
- It performs better memory and CPU Load management through saving the status of each and every process and prioritizing these processes.
- It has features for automatic fault detection and control mechanism.
- It has the feature for automatic alarm handling mechanism.
- Better system failure avoidance by providing the high availability of the system resources.

As the middleware provides the support and coordination along with the OS in supporting the applications for their proper functioning, these middleware services have to be tested for their correctness and performance in their operation and handling the robust load conditions. To do that a sophisticated tool is required which can do the testing on the site and provide the end results of the key performance index related to each and every service handled by the middleware platform in the Telco network elements. The paper focus on the elimination of the manual intervention and development of a complete automated tool which can test the performance of these services in a well advanced manner by reducing the manual risk and errors incurred. The next section emphasizes on the traditional middleware test mechanism and its disadvantages and the motivation to overcome that problems faced in the previous testing mechanism.

II. LITERATURE SURVEY

Middleware Test Mechanism

The middleware has a predefined set of interfaces to the applications running over it and also to the subsystem process of the middleware to perform the required functionality. These interfaces can be modified, added or deleted according to the application requirements and these changes needs to be tested for any dependencies and the API functionalities which will be done manually after every change. The process will be taken on the live network machine and checked for its behaviour, which is a tedious and time consuming technique and requires lot of manual efforts for testing and to monitor those tests. The manual approach is error proven in certain cases where there is dependency with other subsystems of the middleware [5]. The middleware subsystems are basically the timers, counters, event and alarm managers, listeners, loggers, DB managers, tracers, context managers etc… all these processes are interlinked with each other through IPC [Inter Process Communication] where they exchange the messages between them.

Previous Testing Mechanism

The Platform was tested in a manual terms in the previous testing approach which involved lot of delay and testing overhead leading to the limitations on the platform product quality. The Figure 2 shows the flowchart of the testing process used in the conventional service API testing, where any changes in the services or any new services added or deleted will be tested for its correctness by creating a Linux executable and deploying the patch with the latest feature changes on to a live machine, if any error in the service API performance found the entire process will be repeated once again. This testing schema remains same for all the service modules present in the middleware platform.

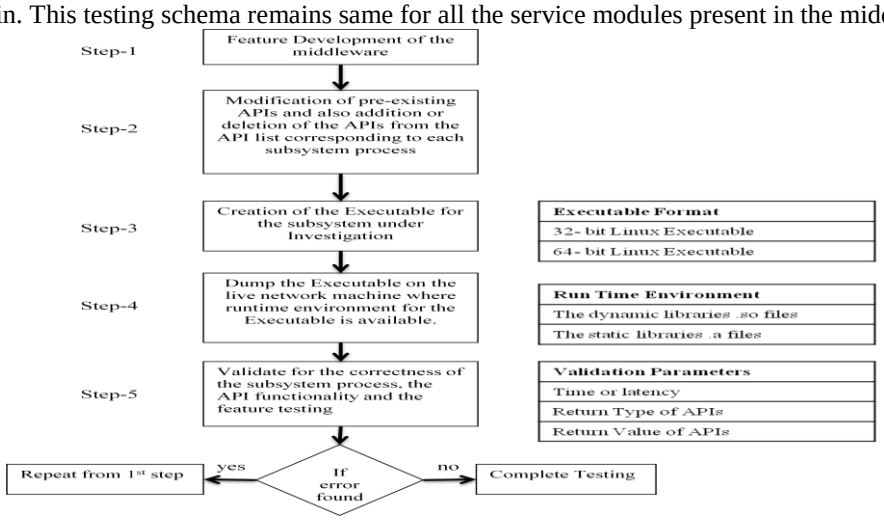

Figure 2: Flowchart for the previous testing mechanism

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

Limitations of the Previous testing Mechanism

The previous testing strategy was had lot of drawbacks in the analysis of the issues with the API leading to the below limitations:

- The previous testing mechanism requires a standalone live machine to test the changes each and every time, which will not be affordable.
- The process created with changes made on the subsystem has to be taken on the live site for testing where dependencies will be high and much of them will not be essential to test that component.
- The running of subsystem process under test on live site may lead to abnormal side effects on other subsystem processes which many lead to non revertible errors, leading to complete stack reinstallation on live site.
- The manual efforts may lead to a failure of system many times.
- More hardware dependency, leads to more expensive test mechanism which is many times unaffordable.
- It will be difficult to automate because the development will be done one site and the testing on the other site which takes pretty much of time to transfer the process onto the live site and test the required features and validated results have to be transmitted back on to the development site for further processing. To overcome all these limitations the paper presents an automated testing mechanism which can also be called as a well equipped tool rather than a simple automation framework and this tool has the following features:

- Standalone development node testing.
- Virtualized test bed simulation.
- GUI based interface with the test bed.
- GUI based validation procedure.

The Automation frameworks can just change the manual effort to the automated terms which doesn't satisfy the needs of our testing mechanism. The Emphasis on the design of the tool will be made in next section.

III. DESIGN METHODOLOGY

The GUI based test tool utility has overcome the majority of limitations faced in the previous testing mechanism as highlighted in the previous section. The proposed test tool utility performs the testing of the APIs by creating a virtual test bed. The virtual test bed is simulated for each and every middleware services separately by virtualizing all the necessary dependencies needed to run that service. The term virtualization here is to make the service independent of the other services. Because of which there is no need for any live setup to run the test cases to validate the middleware services under change. The virtualization is done at the backend as shown in the Figure 3; which will be linked with the GUI developed to view from front end. The virtual bed constitutes of dynamic libraries which will be used by the service under test during runtime [6]. The services are made to work standalone by minimizing the dependencies by the test bed.

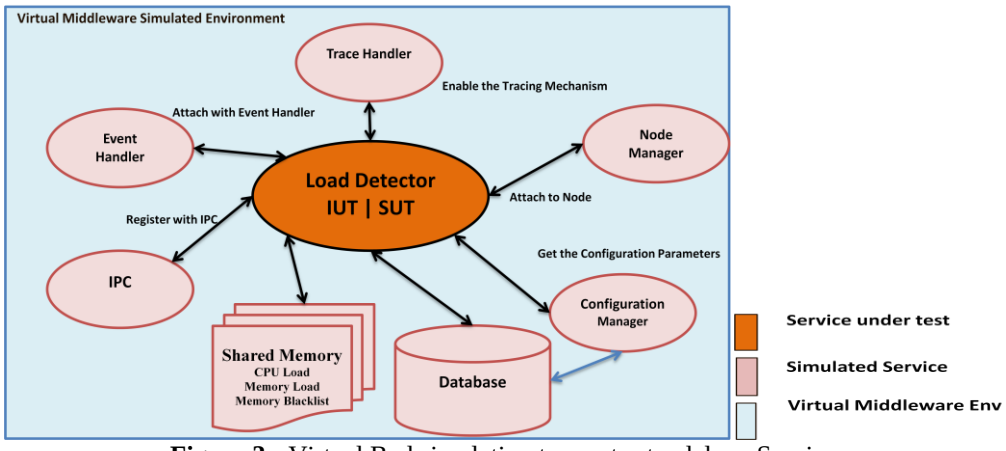

Figure 3: Virtual Bed simulation to create standalone Services

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

The features of the Virtualized test bed are:

- The virtual environment constitutes the service under test and dynamic libraries used during runtime of that service.
- The service is made standalone by minimizing dependencies.
- The service will be a daemon running at backend.

Algorithm with an example: Load Analyser

Need to create a standalone service for the load analysis, so the load Analyser daemon will be the service under test. To initiate this service a set of dependency services have to be virtualized to monitor the normal functionality of the service undergoing the test as shown in the Figure 3. These dependencies get registered with the service under test prior to the actual testing of the service. In the next stage the service under investigation will be tested for its actual functionality, i.e. load analyser will detect the CPU load value, storage capability, memory used, memory blacklisted etc…

Steps undergone for testing:

1. Make service under test, a daemon (Run on infinite loop in background)
2. Run the testcase to check for each and every interface related to load analyzer service.
3. Validate the results for the correctness of those APIs.
4. If any abnormalities in the results, send the request for further analysis of the service.
5. Repeat the steps again (from step 1 to 5)
6. If the test cases executed normally with expected outcome shutdown the service with graceful termination.

This procedure will be applicable for all the other services in the same way.

The features provided by GUI interface of the utility are:

1. The GUI provided interface to select the services to be tested and the APIS list under those services in a user friendly approach [7].
2. The GUI has the ability to run more than one service and its API testing done sequential on a single run and provide the stable results at the end of the test.
3. The GUI provided the slot where we can see the execution and validation result. The above features are as explained in the Figure 4.

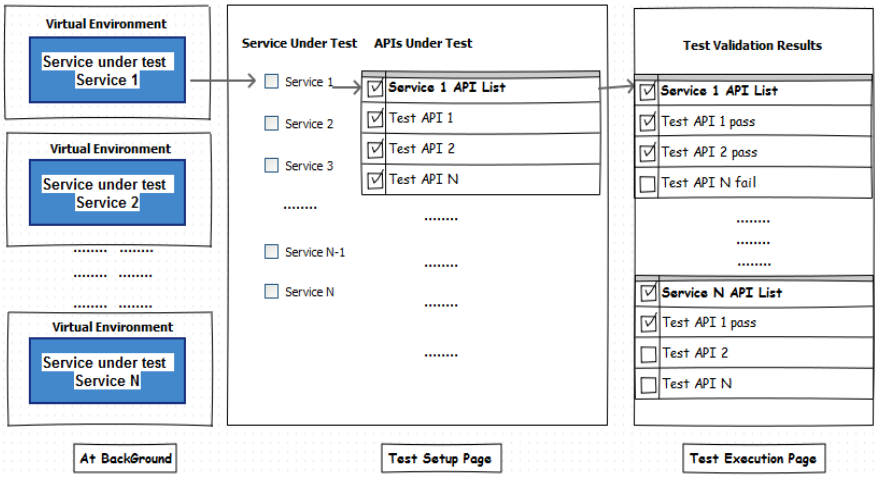

Figure 4: GUI Architecture provision to support the testing mechanism.

Advantages of the test setup use:

- One time creation of virtual environment
- One time creation of test cases.
- Provision to ADD and delete test cases from GUI page.

The test setup implementation and the results inferred from that is explained in the next section.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

IV. RESULTS AND DISCUSSION

As discussed in the previous section, the implementation is done with the help of the requirements provided in the Table 2. The description related with the tools used for the deployment is provided on the same Table. These tools or the utilities have to be driven with the help of some programming languages to build the logic which is as explained in the Table 3.

Virtual Environment Simulation

The primary step involved in the deployment is to create a virtual environment without which the daemon service cannot run on the backend to test the relative test cases. So in order to serve this situation a set of required virtual services are created to make our required daemon process to run.

Table 2: Requirements for the tool Development

Tool Utility	Software/Hardware Tool	Version
Operating System	Redhat Linux	6.3.2 [64 bit]
Test Automation Engine	C Unit	2.1.0
Documentation	Doxygen	1.8.5
Hardware platform	HP or ATCA	NA
Web Server	Apache	2.2.15

Table 3: The Programming Languages used

Language	Utility
C Programming	Used for API implementation, Service Implementation and Build mechanisms + Virtualized test bed creation.
C Unit	Used to build CUnit Automation test cases.
J Query	Used for front end GUI development.
Xml	Used to render the results and fetch the contents to and from the Backend Database
Perl	Used to develop script to access the web based modules
Shell	Used for automation purpose.

The Figure 5 represents the required services virtualized to make the load analyser daemon to run in an infinite loop. Only the required functionalities are simulated to make the service under test to come up. The services virtualized are in the form of libraries which are also executables some are used as static libraries and the others may be taken during runtime for the load analyser to work. The load analyser is a Linux binary executable as represented in the Table 4. The Figure 6 represents the binaries linked with the load analyser service during runtime, without the linking of the virtualized libraries the load analyser daemon process will not come up at the dev machine site.

Figure 5: The Virtual Environment for the Load Analyser service under test

Table 4: Executable formats

File	File Type	File Format
.so	Dynamic Library	Binary ELF
.a	Static Library	Binary ELF
.out	Linux Executable	Binary ELF

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

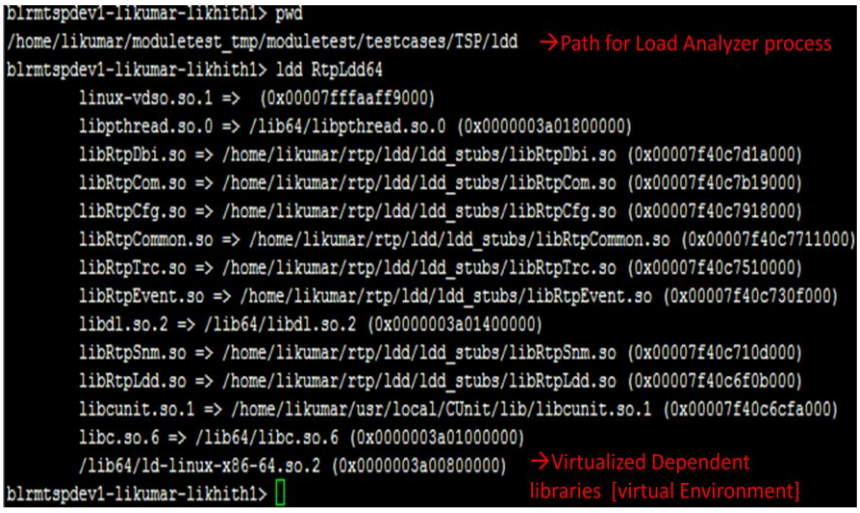

Figure 6: The Load Analyser service under test

The test cases are created using the C-Unit Automation Engine with C as the programming language. To run these CUnit test cases for the API Validation [8] there should be a daemon service running at the background. This can be controlled by the GUI from the front end. To access the GUI, we need to call the apache web server modules with the help of a Perl script which will start the web server by loading the required information provided by the processes running at the backend for the server start-up. As a part of validation to start the web server we need to provide the port number and the user credentials to login to the server, the same credentials have to be provided in the web browser to log in to the GUI home page as shown in Figure 8. The GUI home page will be linked with the other set of service pages namely,

- The test execution config page to do the configuration setup for test execution.
- The results page to check for the validated outcome.
- The basic information page which can give information related with the list of service interfaces and their default threshold value.
- The test summary page which can provide briefed out summary of test results.
- The results obtained will be reflected both on the GUI console and also logged on to the backend database.

The schema of the testing mechanism is as shown in the Figure 7 by taking a service “load Analyser” as an example to explain the test execution scenario.

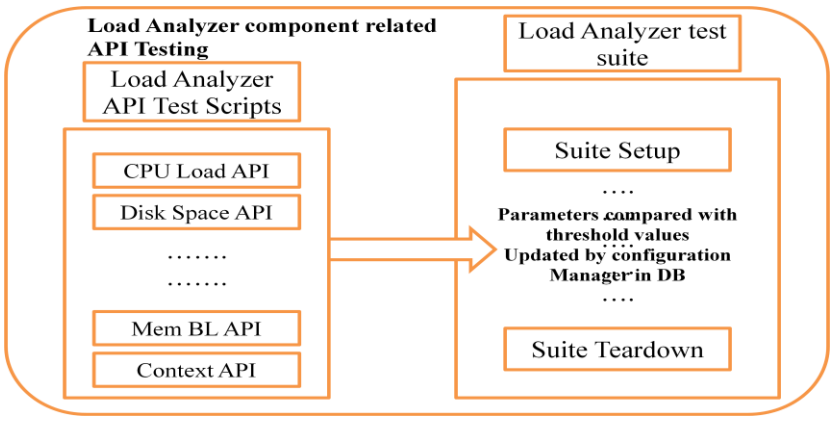

Figure 7: Test setup running at the back end

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

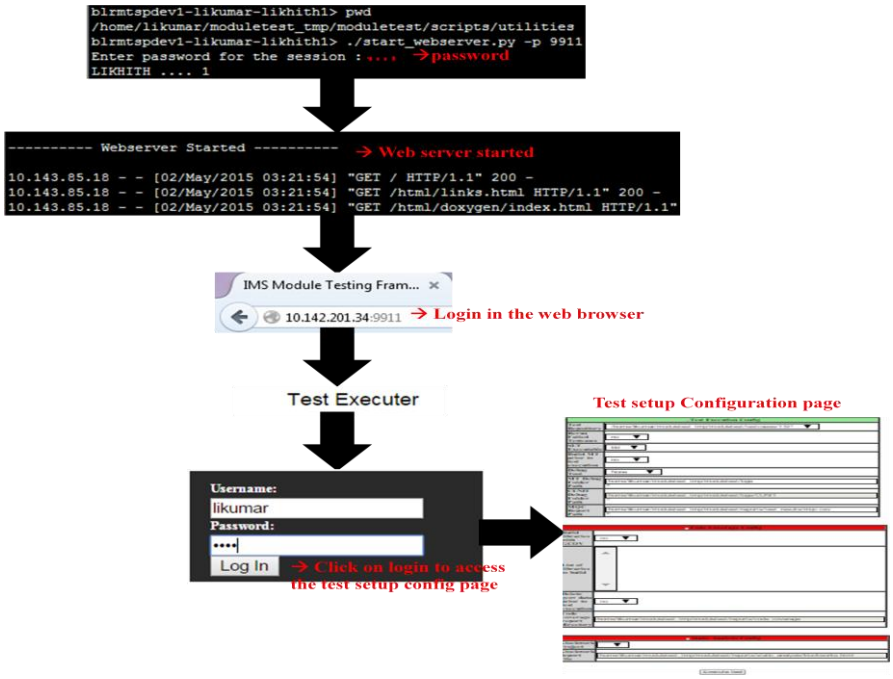

Figure 8: GUI initiation on the web browser by calling the backend web server start-up script.

The test cases once selected at the setup page will be subjected to execute as shown in the Figures [7-8]. The algorithm is as provide in the below steps:

1. Setup the Test suite for the service.
2. Execute the test cases and validate the service APIs
3. Teardown the test suite of the service
4. Repeat the same steps [1-3] for the other services under queue sequentially with their respective API list to be validated.
5. The xml file will be generated by the CUnit Validation Engine which has to be parsed to the HTML format using xml parsing.
6. The results in the HTML format are as depicted as in the Figure 9. It represents the sample results obtained for the load analyser service under test with its set of APIs which are validated when the service daemon is running at the backend [9]. The load analyser related APIs are CPU load, disk usage and memory validation. These APIs are tested in the presence of the service under test is running over which the APIs are verified. If we can get the return values within the threshold limit then the test cases are passed else failed which will be taken for further correction in the service logic.

Automated Test Run Results

Running Suite Suite_1

Running test Test to verify API RtpLddCpuUsage ...		Passed	
Running test Test to verify API RtpLddDiskSpaceUsage ...		Failed	
File Name	RtpLdd_ApiTest.c	**Line Number**	76
Condition	CU_FAIL("Disk Usage is exceeding limit")		
Running test Test to verify API RtpLddMemBlackListed ...		Passed	
Running test Test to verify API RtpLddMsgQueueUsage ...		Failed	
File Name	RtpLdd_ApiTest.c	**Line Number**	128
Condition	CU_FAIL("Message Queue Usage is exceeding limit")		
Running test Test to verify API RtpLddMemUsage ...		Passed	
Running test Test to verify API RtpLddCtxTypeUsage ...		Passed	

Cumulative Summary for Run

Type	Total	Run	Succeeded	Failed	Inactive
Suites	1	1	- NA -	0	0
Test Cases	6	6	4	2	0
Assertions	8	8	6	2	n/a

Figure 9: Result obtained in html format after xml parsing of result obtained from CUnit Test Engine.

ISSN (Print) : 2320 – 3765
ISSN (Online): 2278 – 8875

International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 5, May 2015

The tool developed provides better outcome compared to the previous manual testing approach in terms of time consumption, probability of error, system failure rate, the node testing time, the latency in the execution cycle etc…, which is as shown in the Figure 10. We have taken 5 units as the standard unit of comparison between the two testing mechanisms. But the only difficulty is lot of effort is required to develop the test setup.

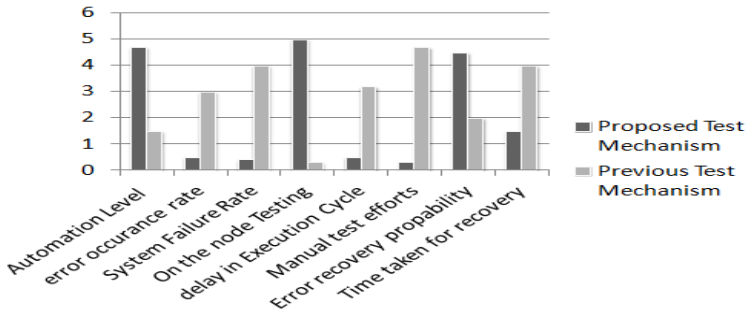

Figure 10: Comparison of performance impact between the proposed testing tool and the pre existing test procedure.

V. CONCLUSION

The results obtained from proposed test tool infers that is provides better performance compared to the previous testing mechanism with respect to key performance parameters like,

- Delay or latency involved in execution.
- Manual efforts involved.
- Error occurrence probability.
- On the node [single node] deployment and testing.

The proposed testing tool has provided better performance than the previous existing testing mechanism which proves better reliability of the proposed tool, which reduces the implementation cost and is only a one time implementation. Effort exists in virtualizing the test bed and creates test cases; the rest is simple by monitoring the test cases and addition or deletion of test cases if required.

VI. ACKNOWLEDGEMENTS

Sincere regards to Mr. Manoj Thomas and Mr. Dharani Basavaraju of Nokia Networks & RV College of Engineering for providing all the necessary guidance, support and facilities for carrying out this project successfully.

REFERENCES

[1] Prashant Shenoy, Purushottam Kulkarni, Krithi Ramamritham, "Middleware versus Native OS Support: Architectural Considerations for Supporting Multimedia Applications," International Conference of Computer Networking, vol. 46, pp. 23 - 32, 2011.
[2] Coluccio R, Ghidini G, Reale A, Levine D, "Online stream processing of machine-to-machine communications traffic: A platform comparison," IEEE Symposium on Computers and Communication (ISCC), vol.25, pp. 1-7, 2014.
[3] Zimmermann H, "OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection," IEEE Transactions on Communications, vol 28, pp. 425 - 432, 2013.
[4] Steegmans F, Mercouroff N, Ceccaldi B, "Mu3S: a middleware platform for telecommunications information networking," Telecommunications Information Networking Architecture Conference Proceedings, vol 36, pp. 131 - 133, 1999.
[5] Pitt J.V, Mamdani E.H, Hadingham R.G, Tunnicliffe A.J, "Agent-oriented middleware for telecommunications network and service management," IEEE Colloquium AI for Network Management Systems, vol 49, pp. 3-5, 2011.
[6] J.Archana, Senthil Raja Chermapandian, Saravanan Palanive, "Automation framework for localizability testing of internationalized software," International Conference on Human Computer Interactions (ICHCI), vol 24 , pp. 1-6, 2013.
[7] Jun Li, Moore K, "A Runtime and Analysis Framework Support for Unit Component Testing in Distributed Systems," 40th Annual Hawaii International Conference on System Sciences, vol 31, pp. 261c, Jan 2007.
[8] Wawrzyniak P. Korbel P, Borowska-Terka A, "Student information delivery platform using telecommunications open middleware APIs," Computer Science and Information Systems (FedCSIS), vol 23, pp. 871 – 874, 2013.
[9] Li Feng, Sheng Zhuang, "Action-driven automation test framework for Graphical User Interface (GUI) software testing", IEEE Autotestcon, vol 8, pp. 22 – 27, Aug 2007.