Robot Automation Framework for ETME Card Bring Up.

Chandana H P, Mahalinga V Mandi
PG Scholar, Department of E&C, DR. AIT, Bangalore, Karnataka, India
Professor, Department of E&C, Dr.AIT, Bangalore, Karnataka, India

***Abstract*—**
This paper presents a Robot based Automation Framework for ETME card bring up (Microcontroller Exchange Terminal for Packet Transport over Ethernet). By using Robot, signal generation function is invoked. Its arguments are based on the test input. Test input data is read from user test case input text file corresponding to the test case and corresponding signals are created in a similar reference text file offline. When test case is executed, Robot will instruct the signal generator to use which file based on the test case id, the signal generator will read the intermediate reference file and form a UDP(User Datagram Protocol) packet and send to the SUT(System Under Test) configured. SGF(Signal Generating Function) will wait for the response from the SUT and dump the output in a text file corresponding to respective test case. Robot will perform validation of the expected output from the reference output file and the dumped output and generate the report.

Index Terms — **Software Testing; Robot Framework; Automated Testing; System Under Test**

I. INTRODUCTION

Robot Framework is one of the open source software developed by Nokia Siemens. It is based on the Python language keyword driven automated test framework. HTML or TSV file organizes test cases, while Setting, Variable, Test Case and Keyword are combined to form data file. Automated testing is introduced by framework, which reduces software regression testing overhead, and also is easy to use. It also provides Python or Java test library and other functions.

II. LITRATURE SURVEY

Robot Framework is one of the tool developed by Nokia Siemens communication technology limited company. But now it's open source software and based on the Python language keyword driven automated test framework. The Framework introduces the automated testing, not only can it improve the testing efficiency, reduce software regression testing overhead, but also is easy to use. At last, it provides Python test library and other functions. Software testing refers to the use of manual or automatic means to run a test system or process. Its purpose is to test whether it meets the specified requirements or find out the difference between the prediction results and the actual results. Traditional manual testing is a non-technical, inefficient, repetitive, and time-consuming labor work. Automation test uses strategies, tools and output, reducing manual intervention to non-technical (repetitive), so as to achieve unmanned guard completion test, and automatically generate test report, analysis of test results of a series of activities. Obviously, the automated test technology can improve the software testing efficiency and reduce the testing personnel repetitive work[1].

System-level test automation has gone through multiple generations, where each new generation has raised the level of abstraction used in the test design. The state of the art test automation, the keyword-driven testing process, abstracts the implementation of tests behind high-level actions, i.e. keywords. In GUI testing, keywords typically depict

basic user actions, such as pressing keys, typing or reading text. The tests are built as sequences of keywords, and keywords are automatically translated into concrete low-level scripts. The abstraction can be increased further by dividing keywords into different hierarchical levels. For instance, low-level concrete scripts may be abstracted under lower-level keywords, and lower-level keywords under higher-level keywords. The two main benefits of keyword abstraction are the reduced amount of maintenance work related to tests and the fact that the tests are easier to build and understand. When a detailed implementation of an action is in one script, the maintenance work has to be performed only to that script. Understanding and creating tests do not require programming expertise since keywords are easy to understand, because they are not low-level system commands, but rather general commands familiar from everyday usage. A keyword script also is much shorter than a more traditional test script[2].

Model-based testing (MBT) is a way of automating test design. It is defined as an approach that uses a model of the SUT in testing tasks. Model-based testing can be seen as a specification-based test generation approach in which the model is the specification. The term model-based testing is a generic term used for several test generation techniques. Utting and Legeard present four test generation approaches in model-based testing:

- Generation of test input data from a domain model.
- Generation of test cases from an environment model.
- Generation of test cases with oracles from a behavior model.
- Generation of test scripts from abstract tests.

The first three approaches are all based on test generation from a model. The domain model describes the domains of input data that can be given to the system and the environment model the expected environment, such as operation frequencies, of the SUT. Both models can be used to generate input for the SUT, but either does not include the expected output of the system, thus requiring manual work for verification. The third model, behavioral model, includes oracle information about the expected behavior of the system and can thus be used to detect any irregularities in the output of the SUT automatically. The forth approach does not include models as such, but rather test cases that are described in a high level of abstraction, without the low-level implementation details. Basically a script defined with high-level keywords could be categorized as the fourth approach. This paper uses the term model-based testing in the third meaning. The test execution in MBT can be divided into *online* and *off-line* testing. In online testing the tests are generated at the same time as they are executed with an adapter tool. Online testing is good for testing a nondeterministic SUT and for long-running test sessions. Offline testing refers to an approach where the test execution and generation are carried out separately[1].

Fig.1 shows Robot Framework architecture. Robot Framework's application and technique is independent of each other, so it is called as versatile. At the same time, it is a set of automated testing tool as shown in Fig.1.

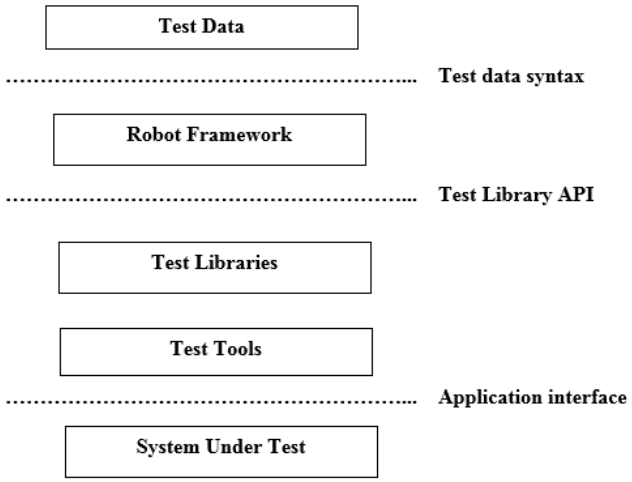

Figure 1: Robot Framework architecture.

- Test Data:

It is easy to edit, which is in the form of table. Robot Framework uses test data to run test cases, which generates logs and reports. For the different type measure systems, the core of the framework is unpredictable and is interactive with the measure

systems through the Test Libraries. Application program interface can be directly used by test library or it can use lower levels of testing tools as driving.

- Robot Framework:

Python or Jython is used to run Robot Framework, because Python language is used as source code for Robot Framework. To start framework, test data file, and to execute test cases, we can use commands.

- Test Libraries:

R&D personnel according to the test requirements as well as Built in Library Robot framework are composed to form Test Libraries.

- System under Test:

Product which is to be tested[1].

IV. PROPOSED SYSTEM

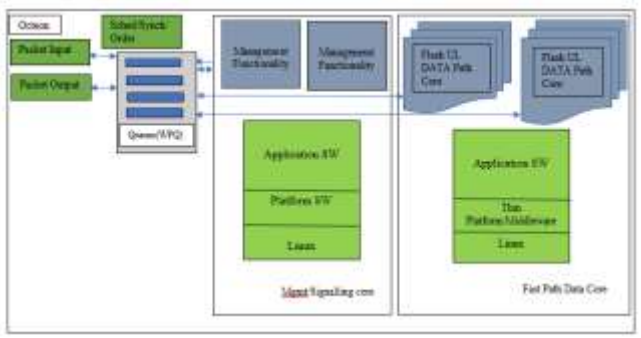

Figure 2: Software Architecture.

ETME Application Software keeps running on Octeon (Multi Controller equipment). Octeon has 12-centers. One center is Management center and other 11 centers are Fast Path (SE-S) centers.

Fig.2 shows a software architecture of the proposed system. The Management center keeps running on Linux working framework and handles O&M usefulness, ETPSIG messages from DX, manages fast path cores and switchover activities.

Fast path cores run in SE-S environment. There is a single task running on these cores and the core handles U-plane traffic forwarding.

The cores have been divided into UL and DL cores. This gives a better usage of Instruction/Data cache, compared to when both UL and DL functionality is run on a single core. Octeon platform is configured such that packet for the any Call from a BCF will be ATOMIC tagged i.e., only one packet from a BCF will be processed at a time by any of the core. UL cores handle the user plane traffic coming from BCF and DL cores handle the user plane traffic from ETMA and PCUM.

The Application Software on the ETME implements the control, management and user plane requirements for providing CS and PS services. The following requirements are handled by ETME software.

- Resource allocation/deallocation for CS and PS calls
- Multiplexing/Demultiplexing of DL/UL CS traffic respectively.
- Routing of UL and DL U-plane traffic to destination TLA of ETMA for CS Traffic and destination TLA of PCUM for PS Traffic.
- Maintains DB to replicate on spare unit in case of controlled switchover.
- Spare unit warming up: Incase of controlled switchover, telecom data for ongoing CS calls will be warmed up on spare unit to support ongoing CS calls.

V. METHODOLOGY

Fig.3 shows the block diagram of robot framework. Robot framework is mainly used for triggering test case, generation of intermediate input signal, validation and report generation. By using Robot, trying to control the invocation of signal generation function and data generation function. Its arguments are based on the test input. User has to create test input data for each test case and the expected output in a pre-defined template. i.e currently it is in sack structure filled manually (future action is integrating sack template generator so that it would be user friendly).

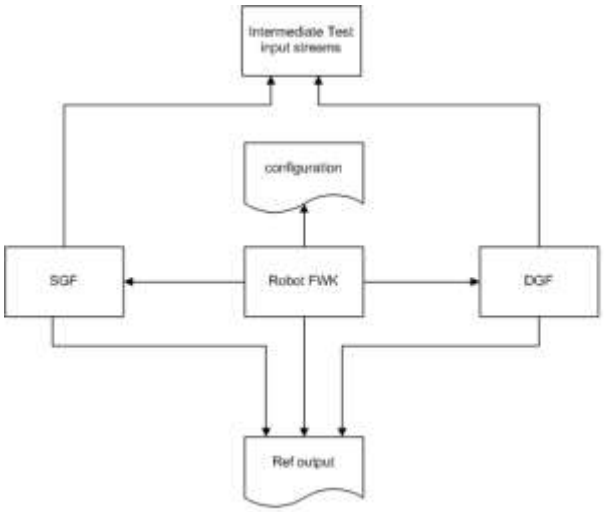

Figure 3: Block diagram of Robot Framework.

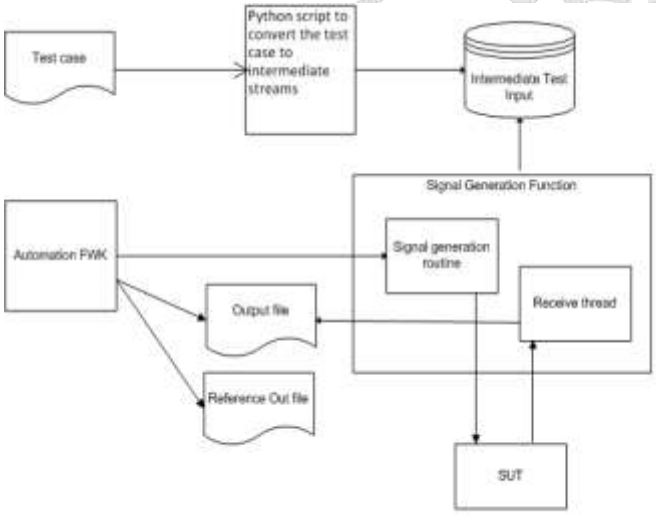

Figure 4: Block diagram of Signal Generation Function.

Fig. 4 shows the block diagram of signal generation block. Test input data is read from user test case input text file corresponding to the test case and corresponding signals are created in a similar reference text file offline. When test case is executed, Robot will instruct the signal generator to use which file based on the test case id, the signal/data generator will read the intermediate reference file and form a UDP packet and send to the SUT configured. SGF will wait for the response from the SUT and dump the output in a text file corresponding to that test case. Robot will perform validation of the expected output from the Reference output file and the dumped output and generate the report. If the response is success, then based on the test case input, Robot will invoke the data generation function for sending the user plane data to SUT.

First install the robot framework by following the steps in confluence page. Create a folder ETP_TC and copy 'robot_fwk' to ETP_TC in windows from the SVN link. A python script - etm_ssi_framework.py with keywords is written in the folder 'robot_modules'. A robot script - 'Login_script_vetm_ssi.txt' is written in folder 'robot_tc'. Test cases are executed using the keywords in the above mentioned script. Run below mentioned command on command prompt set

PYTHONPATH=%PYTHONPATH%;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\TestCases\robot_modules;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\comm\communication\connections;in windows to set python path for Robot :

Before executing the testcases generate DX_SIM by below command

make clean; make

tc<ID>_in.txt is generated by tc<ID>_msg.txt by below command.

pybot -P `pwd` --include robo_test_gen_input ./robot_tc/etme_card_bring_up.txt

If range of tc<ID>_in.txt's need to be generated below is the command.

pybot -P `pwd` --include robo_test_gen_input_all ./robot_tc/etme_card_bring_up.txt

Robot invokes DX Simulator by sending a test case ID. According to the test case ID sent, the arguments for DX_SIM are read from an input file in the same folder as DX_SIM. For test case ID: 1, the file is tc1.txt. The file contains the message IDs to be called and the parameters to be sent along for the messages to vETME. When acknowledgement for particular message is received, it is stored in the same

folder in a tc1_out.txt file. Robot validates the result with the reference file created by the user. tc1_expected.txt We have written User writes data into the file tc<ID>_msg.txt where ID is the testcase ID and this file contains message parameters to be sent to ETM units for signalling/data. From this file Robot Framework fetches parameter values and write it into a format understood by the DX_SIM into file tc<ID>_in.txt. When DX Sim is executed it reads message parameters from tc<ID>_in.txt and sends to ETM unit. DX_SIM gets response from ETM unit and saves in the file tc<ID>_out.txt. Robot compares the result received with the reference result saved by user and test cases passes/fails accordingly.

To run all the testcase in a regression suite by robot framework etme_card_bring_up.txt is written. This will have all testcases list with tag mentioned. Here I have used card_bring_up_etme tag. Below commands are run from the command prompt

set PYTHONPATH=%PYTHONPATH%;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\TestCases\robot_modules;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\comm\communication\connections;

pybot -P `pwd` --include card_bring_up_etme ./robot_tc/etme_card_bring_up.txt

VI. RESULT AND DISCUSSION

By executing the below commands, the robot framework will look for all the testcases with the tag “card_ bring_up_etme” and executes them in suite. These testcases is to bring up etme card. Below are the results. Fig. 5 shows 12 testcases status to written to bring up the etme card and to create bcf from bcf configuration testcase. Fig. 6 shows the time elapsed to run all the 12 testcases. This will be generated as html report.

set PYTHONPATH=%PYTHONPATH%;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\TestCases\robot_modules;D:\userdata\chhp\Desktop\RF_files\ETP_TC\IPV6_ROBOT_FWK\comm\communication\connections;

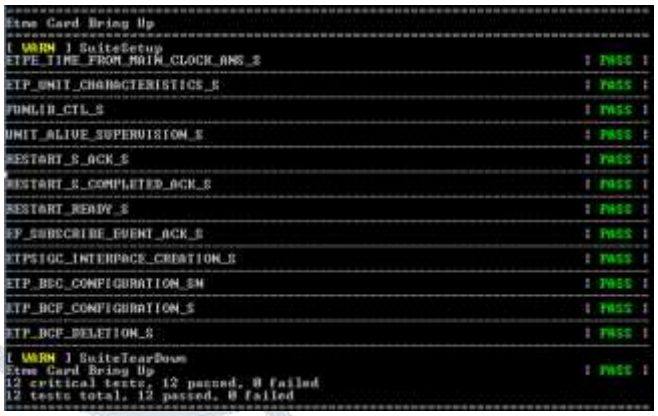

Figure 5: Output of running etme card bring up test cases from robo.

Figure 6: Time taken to run etme card bring up testcases from robo.

CONCLUSION

Since Robot Framework has rich libraries, it is easier for users to extend based on its framework. Extended Robot Framework is an external test library based on Robot Framework and is a scalable keyword-driven test automation framework that can be used to test distributed complex applications. Manual testing is laborious and time consuming. Automation framework reduces manual effort and makes the work faster and efficient. Added advantage of automation framework is can reused. Hence reduces time, cost and manual effort.

REFERENCES

[1] Liu Jian-Ping, Liu Juan-Juan, Wang Dong-Long, “Application Analysis of Automated Testing Framework Based on Robot”, 2012

[2] Tuomas Pajunen, Tommi Takala, and Mika Katara, “Model-Based Testing with a General Purpose Keyword-Driven Test Automation Framework” 2011

[3] Wu JianJie, Chen ChuanBo, Xiao LaiYuan. Software testing technology base. Huazhong university of science and technology press. 2008. pp.10-15.

[4] Author: Louise Tamres, Translator: Bao XiaoLu, Wang XiaoJuan, Zhu GuoPing. Software Testing. Posts & Telecom press. 2004. pp.20-26.

[5] Deng Bo, Huang LiJuan, Cao QingChun, etc. Software test automation. Mechanical industry press. 2003. pp.6-8.

[6] Cai JianPing. Software testing university tutorials. Tsinghua university press. 2009. pp.42-46

[7] Nokia Siemens Networks. Robot Framework User Guide. 2008-2009. pp.6-74.

[8]J. Wu, C. K. Tse, F. C. M. Lau, and I. W. H. Ho, ―Analysis of communication network performance from a complex network perspective,‖ IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 60, no. 12, pp. 3303–3316, 2013.

[9] L. C. Freeman, ―A set of measures of centrality based on betweenness,‖ Sociometry, vol. 40, no. 1, pp. 35–41, 1977.

[10] J. Wu, C. K. Tse, and F. C. M. Lau, ―Concept of node usage probability from complex networks and its applications to communication network design,‖ IEEE Trans. Circuits Syst. I, Reg. Pap., vol. 62, no. 4, pp. 1195–1204, 2015.

[11]J. Liu, R. Love, K. Stewart and M.E. Buckley, ―Design and Analysis of LTE Physical Downlink Control Channel‖, IEEE VTC-Spring 2009, pp.1-5, Apr. 2009.

[12] 3GPP TR36.921, ―Evolved Universal Terrestrial Radio Access (EUTRA);FDD Home eNode B (HeNB) Radio Frequency (RF) requirements analysis‖, V11.0.0.

[13] V. Garcia, Y. Zhou and J.L. Shi, ―Coordinated Multipoint Transmission in Dense Cellular Networks with User-Centric Adaptive Clustering‖, accepted by IEEE Trans. Wireless Comm., Apr. 2014.

[12] 3GPP TR36.921, ―Evolved Universal Terrestrial Radio Access (EUTRA);FDD Home eNode B (HeNB) Radio Frequency (RF) requirements analysis‖, V11.0.0.

[13] R. Kantola, J. Llorente Santos, and N. Beijar, Policy-based communications for 5G mobile with customer edge switching, Security andCommunication Networks, 2015.

[14] Wu JianJie, Chen ChuanBo, Xiao LaiYuan. Software testing technology base. Huazhong university of science and technology press. 2008. pp.10-15.

[15] Author: Louise Tamres, Translator: Bao XiaoLu, Wang XiaoJuan, Zhu GuoPing. Software Testing. Posts & Telecom press. 2004. pp.20- 26.

[16] Deng Bo, Huang LiJuan, Cao QingChun, etc. Software test automation. Mechanical industry press. 2003. pp.6-8.