May 2001

WHITE PAPER

HUGHES™
SOFTWARE SYSTEMS

STATE

Software Tool for Automated Test Environment.

E-mail: info@hssworld.com

COPYRIGHT INFORMATION

Hughes Software Systems
Plot 31, Electronic City,
Sector 18, Gurgaon 122 015, INDIA
Tel: +91-124-6346666, 6455555
Fax: +91-124-6455150, 6455155
Website: www.hssworld.com
E-mail: info@hssworld.com

TRADEMARKS

DISCLAIMER

Contents

Introduction

Software testing is an integral, costly and time-consuming activity in the software development life cycle. Since testing involves running the system under test under variety of configuration and circumstances, automation of testing activities is a potential source of saving in testing process.

Testing a complex system (like UMTS, GPRS, SoftSwitch) calls for sophisticated test tools that are required at almost every phase of the testing. Development of such tools requires time & effort. This calls for a need to have a generic test framework, which can be adapted to perform in diverse scenarios, test the item to the maximum possible extent and provide performance and capability comparable to the standard test tools (e.g. K1297, MGTS (Tekelec), EAST (IPNetFusion) etc). Requirement of a generic test framework becomes more crucial if the interfaces/protocols are non-standards.

The purpose of this paper is to describe Software Tool for Automated Test Environment (STATE) - HSS Interface for Testing Systems. STATE is a Test and Emulation platform for testing protocol implementations. Intended audience of this paper include Test Managers, Technical Leads, Developers and Testers, who want to understand what STATE is and how it can help them to simplify testing of protocol implementations.

STATE provides a user-friendly test environment for developing and executing Test Scripts to test protocol implementations. The user can execute Test Suites for various protocols on STATE platform and verify protocol implementations. STATE is a cost effective Solution that caters to a user's protocol testing needs whether it is Unit Testing, Integration Testing or System Testing.

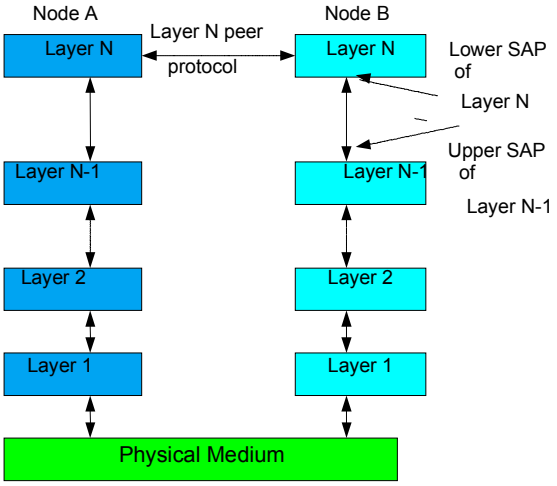
The first section in this paper is a refresher on the key concepts and Terminology of Protocol testing. STATE Solution and how it can be used to meet user protocol testing needs forms the subject of the second section. Benefits of STATE are also explained in this section.

Protocol / Stack Basics

A communication protocol defines the rules for sending blocks of messages (each known as a Protocol Data Unit (PDU)) from one node in a network to another node. In order to reduce the complexity of the system, the rules are usually partitioned into hierarchical structure of protocol layers. A protocol specification defines the operation of the protocol and its interfaces with the adjacent layers. It may also suggest how the protocol should be implemented. Protocol specifications are usually governed by standards international bodies like European Telecommunication Standards Institute (ETSI) and International Telecommunication Union (ITU).

Protocol for each layer is concerned with providing peer-to-peer service with the corresponding layer at the other end of the path. Each layer uses the services of the layers below it, by communicating via a service interface as shown in Figure 1. During peer-to-peer communication, information flows down through lower layers in the same node, across the communications path and up through layers in the other node until it reaches the peer layer.

Boundaries between adjacent layers in the same system are called Service interfaces. The service interface is used to access services provided by a lower layer to a higher layer (or vice versa) through the use of service primitives. The point at which a service is provided is called the Service Access Point (SAP).



Protocol Layering

Point of Control and observation (PCO) is a point at which the inputs of an Implementation under Test (IUT) can be controlled and outputs observed.

Protocol Conformance

Protocol Conformance Testing is essentially a Black Box Testing activity. In Black Box Testing, the Tester is not aware of the internal details of the implementation under Test (IUT) i.e. it is a Black Box to the user. The response of the Black Box to various kinds of stimuli are observed and verified against the specifications.

Purpose of this testing is to rule out all protocol issues. Protocol layers are tested thoroughly as per the standard specification. Both positive and negative scenarios are covered. Following scenarios are covered as a part of conformance testing.

- Valid behavior tests: Pre-defined state transitions are considered as valid. The test purposes in the valid behavior test sub-group covers, as far as reasonable, the verification of the normal and exceptional procedures of the various Finite State Machines (FSMs) of stack entities.

- Invalid behavior tests: This test sub-group is intended to verify that the IUT (Implementation Under Test) is able to react properly on receiving an invalid Protocol Data Unit (PDU). This category includes transfer syntax errors, abstract syntax errors & logical errors.

- Inopportune behavior tests: This test group is intended to verify that the IUT is able to react properly in the case an inopportune protocol event occurs. Such an event is syntactically correct but occurs when it is not expected. An example for this is when a correctly coded operation is received in a wrong state (the IUT may respond by sending error "unexpected component sequence").

- Timer tests: Different timers and counters are defined to supervise the various state transitions. This test subgroup is intended to verify that the Finite State Machine (FSM) is reacting properly to an expiry of one of the timers or counter mismatch.

Challenges before the Test Manager

- Cost and learning: : It is very essential that Unit Testing, Integration Testing and System Testing be performed before releasing the Protocol Stack to the field. Considerable amount of time and money is spent in procuring different types of test equipment/tools for each one of the above-mentioned testing. Training the test personnel to handle these different types of test equipment is also a significant effort. One of the challenges that a test manager faces is to optimize on the investment in terms of resources and have a

single cost effective tool for doing Unit, Integration and System testing.

- Testing on non-target platform: Another challenge faced by Test/Development team is to do some amount of testing of the Protocol Implementation before porting on to the Target Hardware so that we get an early feel of the implementation. In addition, in some cases it is desirable to do testing of the protocol implementation on the standard operating system (Owing to the variety of tools available for debugging etc) before it is cross-compiled for a particular embedded platform. In these cases, the test team adopts a Variant of normal System testing. This involves testing of IUT on development platform before porting it on the Target Hardware.

- Changes in protocol specification: Protocols get refined over a period of time and their specifications undergo changes to cater to wide variety of applications. Protocol testers also need to adapt to these changes and perform testing accordingly. For the System Testing team, this means that the Test Suite needs to be upgraded to cater to the changes in protocol specifications. Vendors who had earlier supplied the test equipment would now have to supply the Test Suite upgrade for the next version of Protocol Specifications. This could be an expensive and time-consuming proposition for the test Team. In this scenario, a simple, cost effective migration path for the Test Suites to the next version of the protocol specifications is something that has been a challenge to the testers and developers. A Testing Platform that aids in adding new Test Cases easily and also allows a user develop them in Open Scripting Language like TCL would be a perfect combination to enhance the unit testing effectiveness.

- Automation: Testing is a repetitive, time consuming and tedious activity. To reduce the testing time and keep the testers motivated, it is necessary that testing activities are automated.

STATE

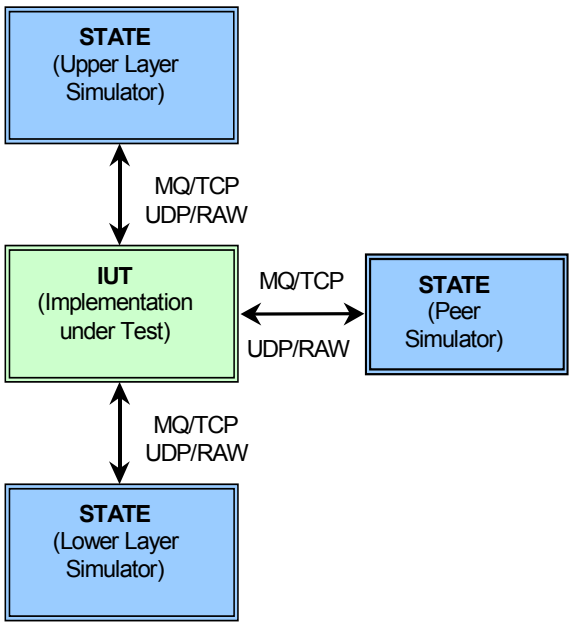
Overview

Software Tool for Automated Test Environment (STATE) is a generic test tool designed to test any protocol stack or product even at the stage of development. It can simulate all the IPC mechanisms of the Implementation Under Test (IUT) at upper, lower, or peer layers.

STATE can test the behavior of IUT according to pre-written Tool Command Language (TCL) based test scripts based on the test plan. Inter-Process Communication (IPC) between STATE and IUT is preformed through message queues, Transmission Control Protocol (TCP) Sockets, User Datagram Packet (UDP) Sockets or Raw Sockets.

It is possible to send Protocol Data Units (PDUs) to the IUT from the Upper Layer Simulator (Upper Tester) and to receive and analyze the response from IUT at the Lower Layer Simulator or Peer Simulator (Lower Tester), and vice versa.

STATE distinguishes the Upper Tester, Lower Layer Simulator, and Lower Tester, depending on the respective interfaces with a logical name.



STATE Simulating Upper/Lower/Peer Interfaces

STATE also supports functional interface to enable calling functions through test scripts.

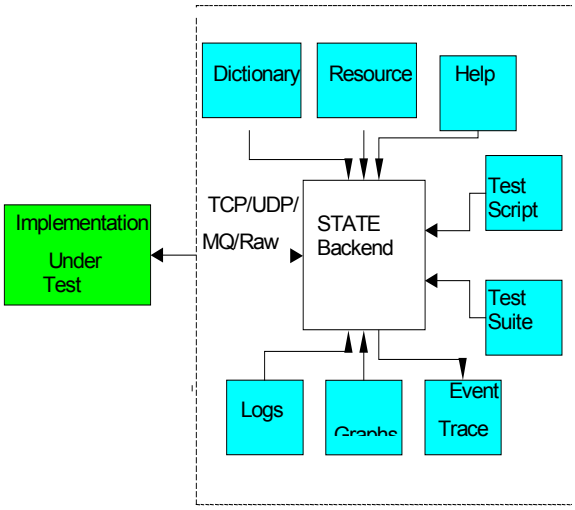
Features

STATE provides an aggregate of most of the features provided by industry standard commercial tool. Some of the important features of STATE are listed below.

- Supports any Protocol & Message/Data format
- Useful for any stage of Testing
- Simulates Upper and Lower Layer
- Portable across widely used Platforms
- GUI as well as CLI mode
- On-line help
- Support for timers & time stamping
- Automatic Script Generation
- Standard Language for Scripting (TCL)
- Test Session/suite with conditional execution
- Interpretation of Test Verdicts
- Statistics with Graphical Presentation
- Supports TCP, UDP, Raw Socket and Message Queue interfaces.
- Supports Multiple Interfaces
- Provision to integrate the stacks directly with the STATE in case of functional interface.
- Supports Octal/Binary/ASCII/Structure formats
- Supports TLV,TV,LV,V data format
- Supports decimal and hexadecimal presentation.
- Supports signed, unsigned and BCD numbers.
- Supports all the basic data types, enums, structures and unions.
- Supports little-endian and big-endian formats
- Support for default & range values
- Automatic generation of Dictionary file using Stack Header file
- Automatic calculation of message length field
- Extraction of a parameter from an incoming message and filling in an outgoing message
- Calling of a user defined function from script.
- Calling of a Test Case(s) within a Test Case
- Embedding of APIs

Architecture

STATE provides a user friendly test environment for development and execution of test scripts to test the complete functionality of a protocol. Functional overview of various components of STATE is described in the figure below.



STATE Components

Dictionary: All the protocol specific information (Headers/ APIs) is defined in the dictionary file. Dictionary can be created manually through GUI. STATE also provides a utility for automatic creation of dictionary from stack header files. This utility is extremely important when the size of the header files is very large.

Resource File: IUT interfaces for Inter Processor Communication (IPC) are defined in the resource file. STATE supports UDP, TCP, Message Queue and Raw Sockets for IPC.

Help: STATE has an extensive help utility to assist the user in efficient use of STATE. Command line help is available on-line. Complete User Manual is also available on-line.

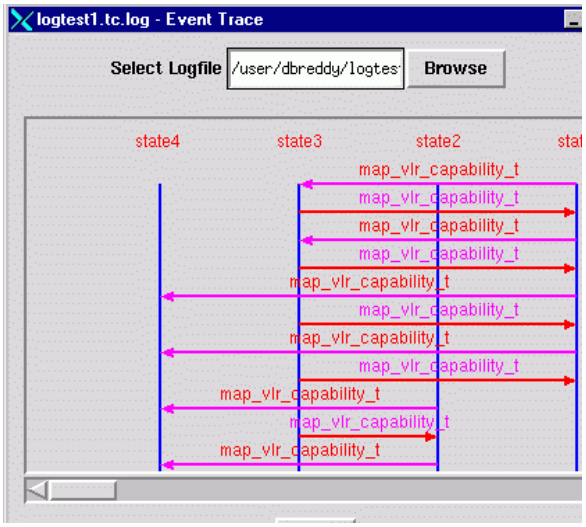
Test Script: STATE provides a simple procedure for automatic creation of Test Scripts. All the protocol information defined in the dictionary file is available in GUI at the time of constructing test script. Test scripts with default, minimum and maximum values of various parameters (defined in the dictionary) can be automatically created with a few selections in GUI. Power of TCL language can be utilized to construct complex scenarios.

Test Suite: Test Scripts can be arranged in the Test Suite. These scripts can be executed any number of times. It is also possible to set delays between the execution of various scripts. Test suite becomes extremely useful in Regression Testing.

Logs: All the information corresponding to test case execution is captured in Log files. User can control the amount of information to be logged by appropriately selecting the log level. Logged information can be subjected to second level of off-line filtering for the purpose of viewing only the relevant information.

Graphs: Important information like test cases passed / failed, messages sent / received is displayed graphical for easier analysis of results.

Event Trace: All the messages exchanged between STATE and IUT is pictorially displayed through the event trace functionality. This gives a very clear graphical display of the test case execution.



Event Trace

Back End: All the modules described above are front-end modules, which assist user in using STATE in a more effective manner. These modules interact with the back-end of STATE, which in turn communicates, with IUT through any of the IPC mechanism discussed before. However, user is transparent to the functioning of back-end.

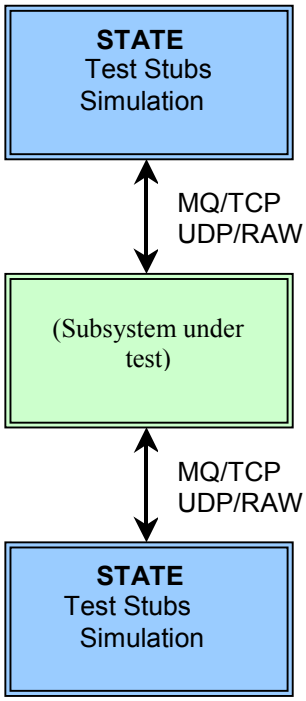
Phases of Testing

Unit Testing (UT)

In this phase, individual units are tested independently while the interfacing units are simulated. Messages are injected at the upper / lower SAPs and the responses are observed on the other side. STATE supports both socket interface as well as functional interface with IUT. In functional interface, the functional unit to be tested is integrated with STATE and a new STATE executable is created. This way, the IUT itself becomes a part of STATE. This is a unique feature of STATE and is very useful for Unit Testing.

Sub System Integration (SSI)

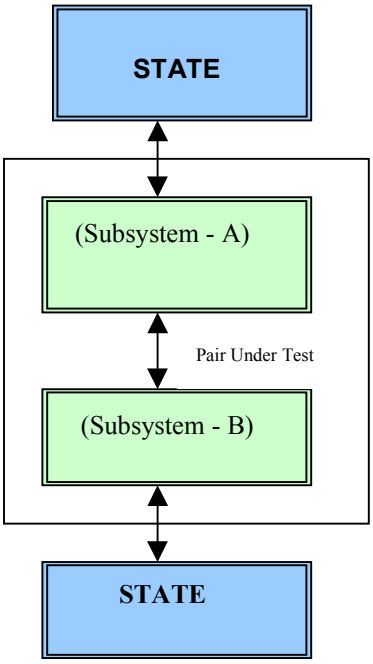
The SSI focus is on integrating the units, which are already unit tested, into a subsystem. The Overall operation and flow within the subsystem is verified along with the sub-system functions. The units are added in steps and are tested for the sub-system functionality. This phase provides additional filter against coding errors. The test cases of this phase are mapped to high-level design requirements.



Sub-System Integration Test Setup

Pair Wise Integration (PWI) Testing

The purpose of this testing is to rule out all the inter-working issues between different subsystems/elements before the complete system is put under tests. The focus of the PWI is to test the ICD between the subsystem/elements. This test phase, conducted prior to End-to-end Integration, reduces risks to future testing.



Pair-Wise Integration Test Setup

Integration Testing

Integration is putting the components together to form the system & achieving end-to-end signaling or data/voice path. Integration is one of the most costly and time-consuming activities in the system engineering process. For large and complex systems, up-to 40% of the development effort may be used in this activity. The integrated system must be tested to check that it meets its requirements.

Following are the key features of Integration testing:

- Functionality testing
- Call Scenarios
- Regression Testing

Performance/Load Testing

The aim of this phase of testing is to verify that integrated end-to-end path meets the performance criteria. This phase is the logical extension of the previous phase & the performance focus is associated to it. In addition, this phase meets the requirement of stress/soak testing.

The major item of this phase of testing is identification of the performance criteria, which forms basis of the performance testing. System modeling is carried out and the test scenarios are run on the model to arrive at the objective figures. This model can also be used to ascertain the system behavior during load/stress for different types & quantity.

Following features of STATE would be available by Jan 2003 and would allow usage of STATE for this phase.

- Advanced load testing with different load pattern: Different load scenarios can be planned for a pre-defined duration. A consolidated report is also provided.
- Multi Threading: In case of pumping the bursty traffic, STATE can help in defining the multiple data sources, load patterns (burst, random, linear).

Benefits of STATE

STATE provides several benefits for the testers and developers. Here are a few ways how one can benefit from using STATE as a testing solution.

- STATE as a Framework: STATE provides the user with the flexibility to create new dictionaries. All the protocol specific information is captured in the dictionary. Therefore, by using STATE, one can create dictionary for any protocol and subsequently test it. Thus, STATE provides user with the power to create his own test tools for testing various protocols. For all the other commercially available tools, usage is restricted

to a particular protocol and user has to buy a license for every family of protocol suite.

- High level of automation: STATE provides a very high level of automation for its usage. Test script / suite generation can be automated. These can be executed any number of times and in any order as per the requirements of the user. Automation becomes extremely significant in case of regression testing.

- Easy Usage: STATE requires minimal instructions for installation and installation can be completed in a few minutes. STATE is very easy to learn. It provides complete functionality through GUI thus ensuring a quick ramp-up. It also provides complete capability through Command Line Interface (CLI) for more experienced user.

- Standard scripting language (TCL): All the test scripts are created in the standard scripting language, TCL. TCL functionality can be used to prepare scripts for complex scenarios. Scripts, once prepared, can be easily modified to simulate new test scenarios.

- Automatic dictionary generation: STATE provides a utility for automatic creation of dictionary from stack header files. This saves very significant time if the header files are very large in size. This is a very useful utility, especially in case of Wireless protocols.

- Functional Interface: This feature allows to carry out test cases that require functional interface. This feature is extremely useful, especially in Unit testing.

- Useful for all stages of testing: STATE can be used for all phases of testing starting from Unit Testing, SSI, and PWI to Integration Testing. This eliminates the need to buy a new license for every stage of testing. This way, it results in a better utilization of Test Tool resources across the various development/testing teams in a project. It also eliminates the need to learn several tools for different stages of testing thus reducing the project cycle time.

- Multiple Interfaces: STATE can simulate multiple interfaces. This helps in establishing an end-to-end scenario for a complete call/data flow consisting of multiple messages across multiple interfaces.

- Timer: Timer functionality is needed to test the capability of protocol to handle timing issues. STATE provides the provision of simulating as many timers as user wants for testing purpose.

- Load: By January 2003, STATE would have complete load functionality. This would help in subjecting IUT to different kinds of load patterns and monitor its performance under stress.

Conclusion

To improve the efficiency and effectiveness of the testing process, groups need to find ways to simplify and automate this process. As a solution to address these issues, STATE was developed at HSS. Since its inception, STATE has been used to create a variety of innovative testing solutions. Using STATE, HSS has achieved considerable savings with respect to people, time and hardware necessary to perform testing.

HUGHES SOFTWARE SYSTEMS

Hughes Software Systems is a key supplier of communication technologies for Voice over Packet, Intelligent Networks and High-speed Mobile Networks, and is fully focussed on the needs of its customers to build Next Generation Networks.

The comprehensive set of software building blocks from Hughes Software Systems consists of both frameworks and protocol stacks for the Voice over Packet domain.

Frameworks	Stacks
Softswitch Framework	MEGACO stack
Media Gateway Framework	MGCP stack
Gatekeeper Framework	SIP stack
SIP Server Framework	H.323 stack
Mini Gateway Framework	SIGTRAN stack

Hughes Software Systems

Plot 31, Electronic City, Sector 18, Gurgaon 122 015, India
Tel: +91-124-6346666, 6455555 Fax: +91-124-6455150, 6348931

HSS USA, East Coast
Germantown
Tel: +1-240-453-2498
Boston
Tel: +1-617-547-6377
Dallas
Tel: +1-972-517-3345

HSS USA, West Coast
San Jose
Tel: +1-408-436-4604
Los Angeles
Tel: +1-323-571-0032; 571-0114

HSS Europe
Milton Keynes, UK
Tel: +44-1908-221122
Germany
Tel: +49-6155-844-274
Finland
Tel: +358 40 8290977

HSS India
Gurgaon
Tel: +91-124-6455555; 6346666
Bangalore
Tel: +91-80-2286390

E-mail: info@hssworld.com

www.hssworld.com