DNA Enterprises, Inc.

DNA Enterprises, Inc.
269 W. Renner Parkway
Richardson, TX 75080
(972) 644-3301

Model Driven Testing

By: Brian Berger, Majdi Abuelbassal, and Mohammad Hossain

March 1997

Introduction

Much of the focus of advanced services deployment is pointed toward application development. Look at the February 1997 issue of *Computer Telephony* and you will see multiple advertisements selling application tool kits. Application tool kits have allowed developers to rapidly create advanced services for the telephony market. But developing these services is only part of the process in getting a product ready for market.

Once the product development cycle is well underway, developers have to deal with deployment issues. A major hurdle in application deployment is integration and test. Unfortunately, the same amount of effort spent on development environments has not been applied to test environments. Test environments are often created in-house and are highly specialized, lending themselves to one particular product and, in some cases, one particular application. Often tests require operator intervention, and are difficult to reproduce when platforms and software are modified.

This paper discusses a test environment philosophy and platform developed at DNA Enterprises to test an intelligent switching product. This particular test environment was developed in parallel with the product. The goal of the test environment was to capture the test requirements, while providing the flexibility needed to respond to product changes.

DNA Test Philosophy

DNA’s testing philosophy is to describe a system, function, or feature in terms of a model. The initial model should be abstracted to a level that is independent of implementation details, or equipment required to test the system, function, or feature. The important first step in developing a test models is to describe, in some form, how the application under test should and should not work. Once accurately captured, these test models become the centralized building blocks for an application or tested item. The second step is to tie models together to describe how features or functions should and should not interact. This hierarchical development approach produces a testing environment that, at a high level, is independent of application and platform implementation issues, and produces test building blocks that lend themselves to reusability. Reusability, can in turn, expedite the development of future tests.

During the test development cycle, application and platform implementation should be addressed with low level models. These model perform similarly to driver level software used in application development. In fact, using this approach, test model development closely matches an application development process. Again, the idea in this type of testing effort is to separate the issues of what needs to be tested from the issues of how the tests are implemented. The “whats” and the “hows” are equally important, but to develop tests that can easily change in response to product or application revision(s), it is essential that the "whats" and the "hows" are clearly separated.

System Test Environment

The goal of DNA's system test group was to create an environment that was geared toward reproducible, automated testing. This goal was realized by combining a group of automated tools with remotely controlled test equipment. Effort was placed on interworking various automated tools into a cohesive bundle that enable automated testing across various types of tests.

Tests covered various aspects of the product, such as load testing, feature testing, alarm testing and Host GUI testing. Additionally, the environment verified the interaction between various tests.

For example, the test system is capable of performing the same feature testing when the system is under heavy load as when the system is under no load. Test interaction is controlled by the high level models that link feature and load tests together.

While the initial system test environment was not expected to test future product functionality, every effort was made to create a test environment that will easily incorporate testing future product enhancements. As additional test hardware or software is required, it should fit within the overall system test environment with minimal additional effort.

The basic system test architecture is divided into two major components: test case generation component and a test case execution component.

The test case generation component is a GUI based model development system. Instead of developing textual based test cases based on system feature implementation, graphical models were developed that described each feature's function and interaction with other system components. The model development tool let the test group address the issues of "what" is to be tested, before addressing the issues of "how" things are tested.

Once the high level feature models are developed, low level "driver" models are created. Driver models deal with the "how" aspect of testing. In dividing test development in this manner, only the low level models change as the product is modified. The high level tests remain intact, and often are the building blocks for more advanced tests.

The second component in the system test architecture is the test case execution environment. The execution environment is often referred to a test harness. It is the responsibility of the test harness to receive command files from the test generation component and translate these files into executable events. Additionally, the test harness is responsible for interpreting responses to these events, collecting feature metrics, and developing a report that is either sent to file, and/or displayed to a console.

Test Case Generation

TestMaster™ from Teradyne's Software and System Test Group was chosen as the test case generation component. TestMaster is an automated test program generator that includes a GUI development environment that is ideal for building models at both abstract and implementation dependent levels. TestMaster's Model Reference Test (MRT) technology fits accordingly with DNA's test philosophy and provides a loosely coupled interface between the test generation component and the test case execution component.

This product allowed DNA's system test group to develop models at various levels and to tie them together to create feature models that, in turn, drive low level models. In this manner, feature models are bound together to create complex and multifunctional tests.

TestMaster can take a detailed model as an input and quickly generate a series of test cases. As a result, the TestMaster model is maintained for each individual test case. The number of test cases generated is determined by the amount of test coverage required. A full coverage test of the model would generate a test case for every possible path through the model. The output from TestMaster for each test case, when run through the test harness, is capable of configuring the switch and the test equipment, execute a series of tests, and develop reports.

Test Harness

The Test Harness is responsible for taking input test cases from the TestMaster/parser software, running the test cases to exercise a switch under test, and creating a test case report.

The Test Harness is built around a UNIX-based PC with Ethernet and serial communication links to the switch, the switch host, and telecom test equipment boxes. Custom software executing on the UNIX platform coordinates downloading and executing test cases on the test boxes or the system and host. Results are collected and formatted into test case reports for analysis and quality assurance purposes.

Part of the process in defining the Test Harness was to create the language constructs used by TestMaster to generate test cases. Special care was taken to develop a language construct that would let the Test Harness act as a test director. Additional consideration was given to future Test Harness expansion.

At the lowest level, the Test Harness consists of a series of hardware telecom test boxes and switch interfaces, each being remotely controlled by an independent software process. Each of these software processes (termed a communication server) is responsible for controlling the flow of information to and from an external interface during the execution of a test case. Laying on top of these individual communications servers is an additional layer of software that is responsible for controlling the sequence of events and the flow of information between the individual communication servers. This controlling software is referred to as the Test Execution Engine. This software is required to coordinate events and information between various test boxes, and the product under test. The Test Execution Engine software resides on a PC referred to as the System Test Controller.

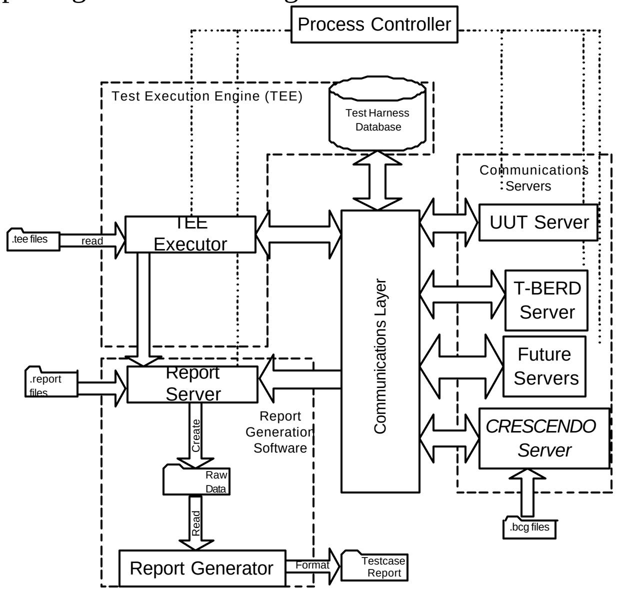
External devices used in the test harness fall into two primary categories - Telecom Test boxes like Bulk Call generators and interfaces to the switch under test like a remote Maintenance and Provisioning (MAP) Terminal or an Alarm Panel interface. The System Test Controller and controlling software is connected to remote devices either through an Ethernet connection, an RS232 serial link, or a GPIB connection. On the trunk side, the telecom test boxes are cross-connected to the switch under test via a jackpanel.

Test Harness Software

The test harness software is broken into major subsections:

- Test Execution Engine (TEE) - The TEE is responsible for executing test cases within the environment. It is driven by a sequential test execution script file written in Perl 5.0. This script file is generated by the TestMaster/Parser software and executed by the TEE executor. Information regarding the system configuration is located in the Test Harness database.
- Communication (Comm) Servers - Communication Servers handle all communications between external devices within the test harness. The test execution engine communicates with any external device via a communication server for that device using a Comm Server API.
- Report Generation Server - The Report Generation Server develops test case reports based upon raw data from the test execution engine and the Comm Servers. It is composed of a report server for raw data acquisition and a report generator for report formatting and generation.

Figure 1 Illustrates a model of the Test Harness subsections and their interactions. Adding equipment or other interfaces to the Test Harness is a matter of developing another communications server for that particular device. This gives the Test Harness the power to add additional test devices or switching interfaces without impacting the current configuration.



Conclusion

The combination of TestMaster with DNA’s Test Harness has given DNA’s system test group a powerful and flexible tool set. This tool set has given our system test group the ability to generate and maintain complex tests suites with minimal staff.

About DNA Enterprises

DNA Enterprises has established a unique position within the telecommunications industry. Since 1981, DNA Enterprises has offered a broad array of services, ranging from concept evaluation to full system development. Areas of expertise include software technology, hardware technology, systems architecture, and digital signal processing systems. We specialize in telecommunications and multimedia systems and applications.

For additional copies of this paper, or additional information about DNA Enterprises, visit our Web site at http://www.dnaent.com, or send e-mail to info@dnaent.com.

References

TestMaster™ is a trademark of Teradyne, Inc.