

Simplifying TPS Development and Execution Using a PC, Web-Based Environment

David Rolince

Teradyne, Inc

Boston, MA 02111

Phone 617-422-3197 Fax 617-422-3440 Email david_rolince@teradyne.com

Abstract: Test Program Set (TPS) development and test execution for modern electronic modules involves the integration of a myriad of software and hardware tools which are inherently difficult to integrate. It is not uncommon to have a combination of digital, analog, and mixed-signal test routines in the same test program, all of which use different tools for development and execution. While data exchange and software language standards have helped provide a common ground for reducing the barriers to integration, there has been no progress in providing a user environment that can leverage these advances.

This paper describes a PC-based TPS development, documentation, and execution environment that uses Microsoft COM interfaces, high-level scripting and World Wide Web technologies to integrate development processes and point tools from multiple sources. The result is an open, easily re-configurable TPS operating environment that uniquely combines flexibility with a process-oriented framework.

I. BACKGROUND

No one can dispute the impact that PC-based graphical programming has had on test program development applications over the past several years. Since its introduction, Microsoft Windows has revolutionized the way engineers think about and implement programming on Automatic Test Equipment (ATE). Hundreds of commercial tools have been developed that leverage Windows technology. Graphical programming tools enable engineers to do in minutes with clicks of the mouse what used to take hours hammering away in C code. In spite of the productivity gains graphics-based tools offer, there remains the challenge of integrating the output of these tools into an organized, documented process, where each step can be traced to supporting development and documentation sources. One of the most promising solutions for integration exists in the combination of Windows and World Wide Web technologies [1].

For the purpose of this discussion, the test program development process can be divided into four phases:

1. Test requirements definition
2. Test strategy definition

3. Test design
4. Test implementation

Test design and implementation starts with a distillation of test requirements for the UUT (unit under test) be it a single device or an entire system. These requirements are generally stated in a document of some type, and are used to determine:

- an overall test strategy
- the test design in terms of the series of steps to be implemented and their sequence
- the test implementation with instruments that will be used to apply the tests to the UUT and measure the responses, and
- the tools that will be used to develop the program code to drive the instruments

Once the program code has been developed for each test using graphical programming tools or programming languages such as C /C++, it is included as a module in the execution sequence. A higher-level test executive organizes the flow of the individual program modules and provides the means for steering the UUT test program based on the responses from the individual modules. Most test executives are custom built, can take months to develop and debug, and leave little opportunity for reuse. The opportunity for reuse can significantly reduce the cost of test program development by virtually eliminating the non-recurring engineering required to redesign and redevelop the test executive under the old paradigm.

Teradyne recently announced TestStudio, a test operating environment that combines Windows and World Wide Web technologies to simplify the way test programs are developed, documented and executed. One of the key innovations of TestStudio is the use of Web tools and Microsoft COM (Common Object Model) interfaces to provide an effective means of information

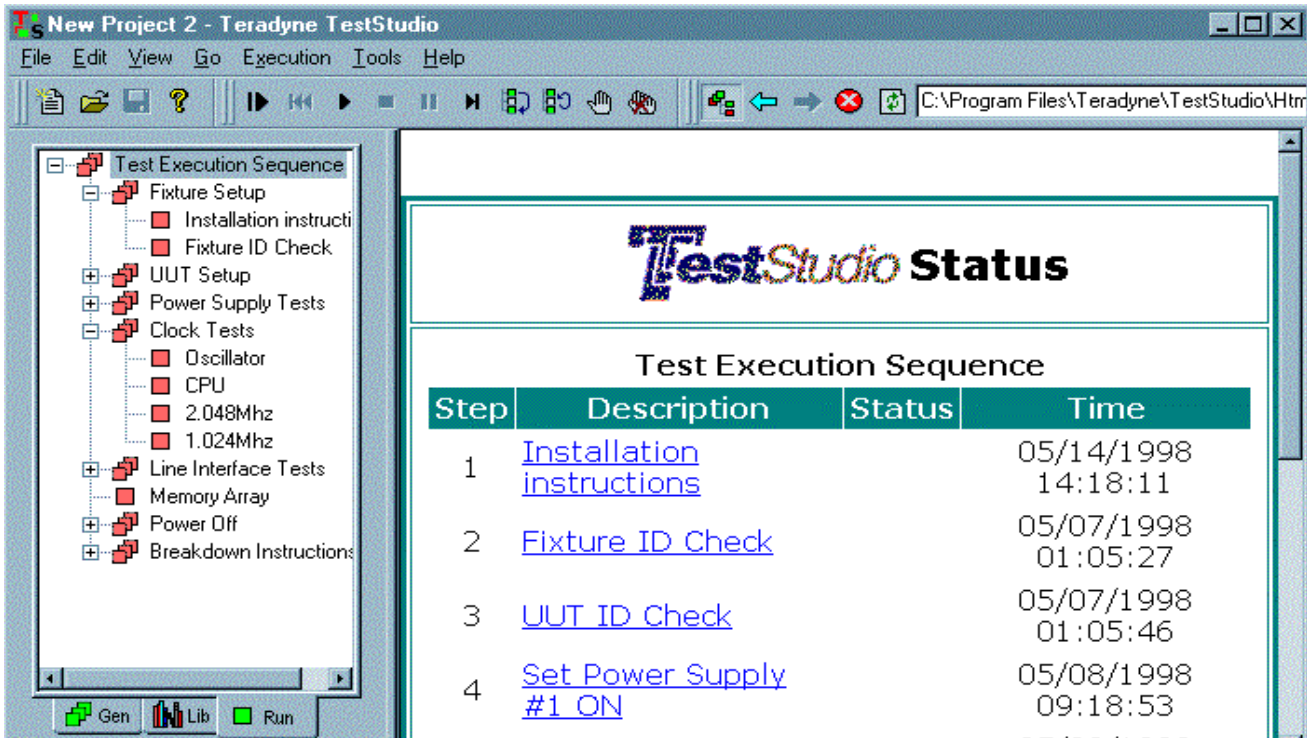


Fig. 1. User Interface in TestStudio showing the Windows Explorer tree control (left) and Web Browser (right)

access and data utilization at every stage of the test programming process.

II. HIGH LEVEL DESCRIPTION OF TEST STUDIO

TestStudio is a modular, flexible ATE operating environment that runs under Windows95, 98 and Windows NT (Fig. 1). TestStudio integrates the activities and processes involved in generating, executing and documenting a test program.

Test processes take the form of a Windows "tree" structure that supports hierarchy and sequence flow. A typical TestStudio process tree contains a series of nodes that may or may not be hierarchical. Each end node on a tree is called a leaf. It is at the leaf where TestStudio communicates with other programs and processes. The communication is achieved through COM interface objects called leaflets. Leaflets enable communication between TestStudio and other programs by passing the values of mutually recognizable "properties" back and forth. TestStudio supports inheritance so that properties defined in TestStudio can be passed down through the hierarchy of a parent branch of the tree. In this way data can be captured, stored and managed within a project.

The principal user interface in TestStudio is a Web browser. The browser displays status of each selected end node in a process, and acts as the entry point for user input. It supports static and dynamic text, graphics and video displays.

III. SIMPLIFYING TPS PROCESSES

Test Documentation

In a documentation tree, all relevant test program documents can be catalogued and referenced. Using URL (Universal Resource Locator) addresses (Fig. 2), documents can be accessed and displayed locally from any network the PC running TestStudio has access to. This could range from a file server on the local network to a geographically remote location accessible over the Internet. This offers the ability to provide broad access to important documentation sources and, at the same time, maintain strict configuration control that is critical to eliminating the ubiquitous problems associated with using the wrong information to generate a test program.

Use of Web browser technology also has the advantage of being able to display Hyperlinked Text Mark-up Language (HTML) documents as well as active graphics such as video. For example, an execution step in a

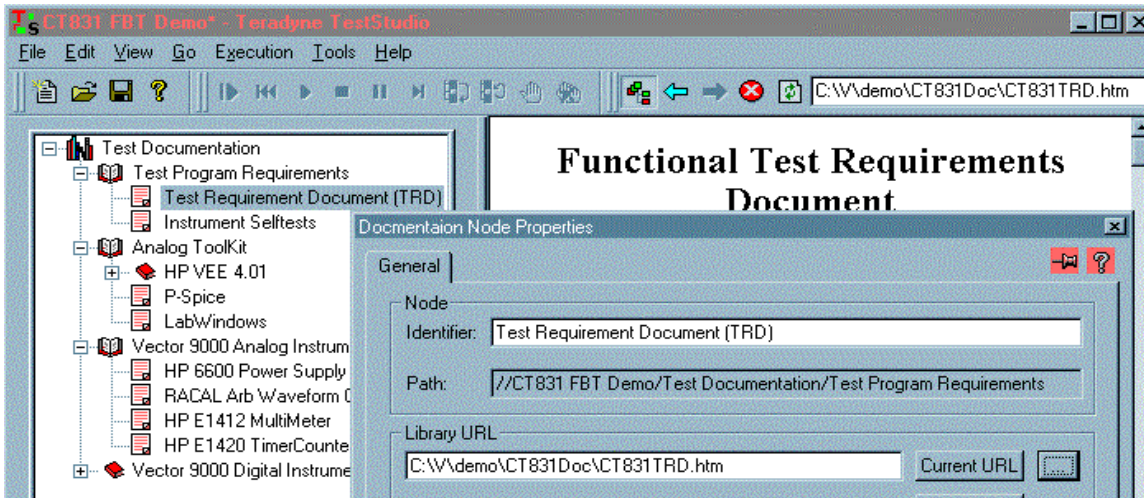


Fig. 2. Documents and other such files are referenced in TestStudio by their URLs.

production test program could navigate to an URL that runs a video clip to illustrate how to attach a fixture to the tester, or how to execute a probing sequence to isolate a failure on the UUT. This can be significantly more effective than describing a procedure in text or in a static diagram.

Test Strategy Implementation

Once a test strategy has been decided upon, a test developer will define and lay out the test steps to implement the strategy. This typically takes the form of a flow diagram that is identical in structure to the Windows Explorer tree used in TestStudio. Once the details of the flow have been defined, it can be saved and archived for reuse in a test program employing the same or similar flow. Existing process tree architectures can be imported into a test project by simply copying the archived process into the new project. This can save significant redevelopment time and provide a means for standardizing test processes at a high level.

Test Design and Execution

One of the most time consuming efforts in test program integration is generating the high-level executive process that sequences and manages the application of each coded test module. While integrating modules that use the same programming language helps reduce the effort, this is not always feasible. Integrated UUT test solutions frequently employ a variety of tools from different vendors, or a combination of commercial products and custom C/C++ code. This can result in com-

plex test executives that are difficult to create and nearly impossible to reuse.

Through the use of leaflet technology, TestStudio simplifies the process of test module access and control. TestStudio includes a leaflet library that enables communication with Application Development Environments (ADEs) such as National Instruments LabVIEW and LabWindows/CVI, and Hewlett Packard's HP VEE (Fig. 3). These ADEs support the Microsoft COM interface standard, which is the foundation of TestStudio leaflet technology. There are also leaflets for interfacing to specialized in-house programs written in C, C++ and Visual Basic.

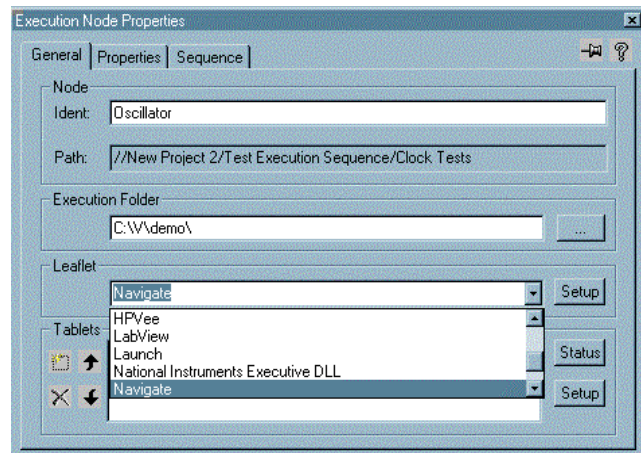


Fig 3. COM interface programs called leaflets, are selectable in the property sheet of a node in TestStudio.

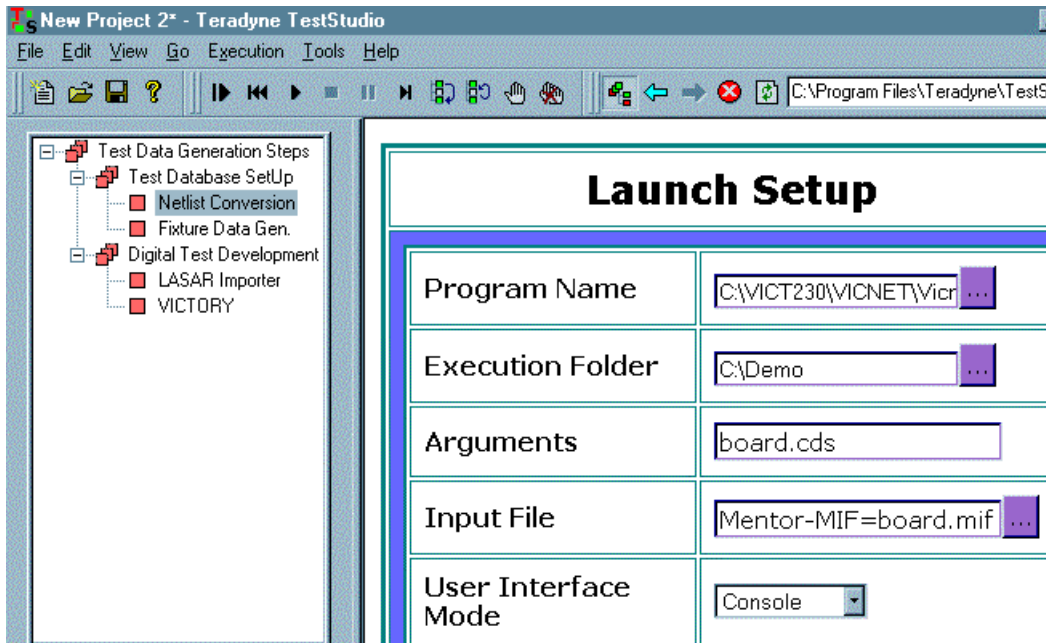


Fig. 4. The “launcher” leaflet can be used to encapsulate non-UI programs to provide a consistent, easy to use interface to the program.

In addition to basic communication with ADEs, TestStudio provides the means to execute an activity based on the status of a condition prior to or subsequent to a test step. This is accomplished through special purpose leaflets called inlets and outlets, respectively. For example, a procedure that powers down a UUT can be bound to the outlet of a test step such that if a failure were detected on that step, the power down procedure would be executed, and a diagnostic message would be displayed in the browser window. Since TestStudio supports inheritance, the fail property set in the previous example would be carried through to all the other nodes at that level of hierarchy and lower, thereby influencing their execution status.

Over the course of test program integration and debug, test program execution flow frequently needs to be modified or re-sequenced. TestStudio simplifies this task by supporting cut/copy/paste and drag-and-drop capabilities of the Windows environment. Any node modified by these actions automatically carry with them their property assignments.

Test Development

After having decided on a test strategy and implementation plan, the tools required to generate the data needed for the test program and the program development tools need to be identified. While these tools are

often test implementation specific, many of them can be a standard part of the test program generation process. These include netlist translators, digital test pattern translator and compilers, simulation data postprocessors, or any program that is used to create or modify data used in test program generation. By organizing these generation steps in the tree structure, a test generation process and predefined sequence can be developed and standardized. For example, if a successful netlist translation is a prerequisite for fixture data generation, the development tree can be organized such that netlist translation precedes fixture data generation. The status display in the browser window clearly indicates the disposition of the process.

Custom tools and utilities written in C and C++ are common in many test development groups. These are used to automatic specific tasks such as file transfer, data conversion, or other site specific tasks that make the test developers job easier. These are generally not elegant programs and often require that the user be familiar with the qualifiers and syntax rules of the program. Using a leaflet in TestStudio called the launcher (Fig. 4), custom programs can be easily encapsulated to provide a clean and consistent user interface to the custom tool and thereby eliminate the need to memorize syntax and qualifier rules.

VII. CONCLUSION

The benefits of graphical test process generation and test program code and documentation reuse are just one part of the solution for TPS development. The real benefit in complex TPS projects is the use of an open architecture operating environment that enables reconfiguration and reuse of test resources. By employing familiar World Wide Web technology and industry standard COM interfaces, easy access is provided to test data and to a broad range of best in class tools for test program development.

REFERENCES

[1] P. Hansen, "The World Wide Web Leads a Revolution in ATE Programming Environments", *Proceedings of IEEE Autotestcon, 1997*.