Integrating VISA, IVI and ATEasy to Migrate Legacy Test Systems

By Loofie Gutterman
Geotest-Marvin Test Systems, Inc.
17570 Cartwright Road, Irvine, CA 92614
(949) 263-2222
loofieg@geotestinc.com

New software technologies such as VISA and IVI continue to bring the industry towards greater standardization and interchangeability. The benefit to the integrator is reduced costs through reuse of the same hardware and software. The benefit to the customer end-user is lower costs by reducing modification and support throughout the life-cycle to the test set. However, while we position ourselves for the future with PXI and these software technologies, we must still provide support for legacy VXI, GPIB and instrument drivers that use current software technologies.

Using a number of additional tools, we can have the power of these tools today while waiting for wider acceptance and support of the newer VISA and IVI technologies. We are just now seeing the development of IVI drivers and the ink is still wet on the VISA specification for PXI. ATEasy provided the structure necessary to use these technologies with the legacy technologies using an innovative mechanism called Instrument Interchangeability Technology (I^2T). This paper discusses the evolution of test programming and instrument drivers and offers implementation methods for these new technologies.

Interchangeable Virtual Instrument or IVI is a result of many years of frustration and many millions of dollars spent while maintaining Automatic Test Equipment (ATE).

In the beginning of the automated test industry, programs were used to remotely control test instrument as well as to perform the necessary test functions. This paper does not discuss the very early controllable instruments that used Binary Coded Decimal (BCD) interface for control and starts with the General-Purpose Interface Bus (GPIB) that started in 1965 as a Hewlett Packard protocol called HPIB (Hewlett Packard Interface Bus) and was later adopted as an industry standard in 1975 (IEEE-488).

Early GPIB test programs had statements such as:

- Output @701, “F1”
- Output @702, “VS28.0”
- Output @702, “AS1.0”
- Output @704, “C15”
- Output @701, “?”
- Input @701, %R

To the ATE programmer, this code was legible. The first statement send the command “F1” to the GPIB instrument in address 1 which in turn, sets the instrument (in this case a DMM) to measure Volts DC. The second and third statements sets a GPIB power supply at address 2 to 28VDC with a current limit of 1A. The forth statement directs a switch box at GPIB address 4 to close relay #15 and the last two statements trigger the DMM to take a measurement (“?”) and then read the result and assign it to the variable %R.

Figure 1 depicts the software configuration of a tester using the early development methodology.
While this software architecture seemed as a great solution for that time, it later proved to be a problem of enormous proportions. First, each programmer had a unique method of writing programs so a simple program for programmer X proved to be difficult to maintain by programmer Y. The bigger problem was that when an instrument became obsolete, so did all the test programs using it and they had to be rewritten to accommodate a different GPIB command set of the new instrument.

Then came the instrument drivers. An innovative concept, the instrument driver provided a layer between the program and the instrument so that cryptic statements were no longer needed (for the most part). Figure 2 depicts the software configuration of a tester using instrument drivers.

Unfortunately, while the instrument driver did simplify software development and maintenance, they didn’t address hardware obsolescence as each manufacturer had their own GPIB command set and none were compatible. Further, even when the obsolete instrument was replaced by one from the same manufacturer, the GPIB command set was somewhat different which required software re-write.

The next step in the evolution of instrument control and programming was SCPI (Standard Commands for Programmable Instrumentation) which defines a standard set of commands to control programmable test and measurement devices in instrumentation systems. SCPI (pronounced as “Skippy”) started when a group of Test & Measurement companies met at AutoTestCon 1989 and later became a standard. SCPI was based on a language first used by HP (TMSL) and while it doesn’t completely solve instrument interchangeability issues, it does provide a high degree of consistency among instruments.

The GPIB program example used earlier in this paper would look as follows using SCPI commands:

Output @701, “FUNCTION:VOLTage:DC”
Output @702, “VOLTage:28.0”
Output @702, “CURRENT:1.0”
Output @704, “ROUTE:CLOSE:15”
Output @701, “MEASURE:VOLTage:DC?”

Obviously, the above syntax is easier to read than the earlier example however, the syntax is still cryptic and as stated earlier, does not provide a complete solution to instrument interchangeability.

The next step in the instrument programming evolution is the VXI Plug&Play Drivers and VISA standard. This standard was formed to increase the use of VXI technology but then later on was enhanced to provide a framework for instrument drivers for other type of instruments such as GPIB, RS2323, TCP/IP, PC cards and more. The standard includes a library called VISA (Virtual Instrument Software Architecture) used by drivers to access hardware. Many VXI Plug&Play drivers are available today and they all have similar functions for instrument initialization, Reset and more. The VXI Plug&Play drivers are available as a DLL with interface for many programming language such as C, LabView, Lab Windows and more.

The next (and more recent) step in the instrument programming evolution is the Interchangeable Virtual Instruments (IVI) standard that was built on top of the VXI...
Plug & Play standard. Again, it was a result of several Test & Measurement companies trying to resolve the interchangeability issues that have been haunting test engineers for the past few decades. IVI drivers define improved usability, functionality, and interchangeability that reduce the cost of test system development and ownership. IVI drivers further simplify upgrading or replacing instruments in test systems by providing better interchangeability.

The IVI standards define open driver architecture and a set of instrument classes with each class addressing the driver requirements for a specific family of test instruments. The currently defined Instrument Classes include:

- DC power supply
- Digital multimeter (DMM)
- Function generator & Arb
- Oscilloscope
- Power meter
- RF signal generator
- Spectrum analyzer
- Switch

With many of today's test instruments being Instruments On a Card (IOC), the software controlling these products is in many cases a Dynamic Link Library (DLL) driver as opposed to GPIB commands. DLL's, like any other software products can have a different look and feel when coming from different vendors. However, an IVI driver will have the same look and feel provided that the instruments have similar functionality. While some of the instruments may still use a DLL, the IVI driver acts as an additional layer, providing the necessary interchangeability. Figure 3 depicts the software configuration of a test system that uses IVI drivers.

**Figure 3: ATE Software Configuration Using IVI Drivers**

As shown here the application can use Class complaint driver (e.g. IviDmm class) or the vendor driver (class complaint or not). The IVI driver comes in two interfaces IVI-C for languages (i.e. Visual C, ATEasy, Lab Windows/CVI) that can use DLL functions or IVI-COM for COM based applications (i.e. Visual Basic, ATEasy).

The statements below represent the IVI equivalent of the instrument control code used earlier in this paper:

```c
IviDMM_SetAttributeLong(m_viDmm, IVIDMM_ATTR_FUNCTION, IVIDMM_VAL_DC_VOLT);
IviDCPwr_ConfigureVoltageLevel(m_viDcPwr, "0", 28.0)
IviDCPwr_ConfigureCurrentLimit(m_viDcPwr, 1, 1.0) ! trip limit
IviSwitch_Connect(m_viSwitch, "1", "10");
IviSwitch_WaitForDebounce (m_viSwitch, 1000);
IviDMM_Read(m_viDcPwr, 2000, &dReading);
```

The above code may seem more complex than the previous examples although it is more legible. Further, assuming IVI drivers are available, if the hardware being used (i.e. DMM, Power Supply & Switch Matrix) is to be replaced with instruments with similar capabilities, the programs can remain unchanged.

In 2002, the SCPI Consortium, the controlling body of the SCPI specifications voted to become part of the IVI Foundation, the controlling body of the IVI specifications and in 2003, the board of directors of the IVI Foundation voted to accept SCPI so these two standards are now under one umbrella and controlled by a single group.

While IVI provide a major breakthrough in instrument interchangeability, the availability of Instrument Classes is still the limiting factor. For example, a basic test and measurement instrument such as a Frequency Counter doesn't have a class defined yet. The availability of
Instrument Classes, combined with the complexity of the programming code limits the wide-scale deployment of IVI.

Recognizing the issues associated with instrument interchangeability, ATEasy software provides a unique vehicle to eliminate instrument obsolescence problems. ATEasy is an integrated framework for the development and execution of test applications. While ATEasy fully supports IVI and is provided with IVI drivers for the released Instrument Classes, it further provides the mechanism to support Instruments without classes through Instrument Interchangeability Technology (I2T). The I2T allows the development of two-tier instrument drivers that provide for full interchangeability. While a typical non-IVI instrument driver uses a single function for the user interface (API) and for the instrument interface (the low-level code used to control the instrument), I2T drivers utilize two separate tiers with the first being the API and the second being the instrument interface as shown in Figure 4.

![Figure 4: Driver Architecture Comparison](image)

In the I2T driver, the API and the Instrument Code are two separate the interface (ATEasy commands) that the application uses to program the instruments from the implementation code. In the event that an instrument needs to be replaced, the driver needs to be modified but ONLY the Instrument Code sub-module needs to be replaced, maintaining the API sub-module intact and as a result; all test programs utilizing this driver can remain fully operational without a single modification.

For reference, the code below is an example of ATEasy IVI code which provides the same functionality and interchangeability while simplifying the programming and increasing the legibility of the programs.

```plaintext
DMM Set Function DCVolts()
DCPWR Configure VoltageLevel("0", 28.0)
DCPWR Configure CurrentLimit LimitTrip ("0", 1)
DCPWR Configure OutputEnabled On()
SWITCH Connect("1", "10")
SWITCH WaitForDebounce(1000)
DMM Read (2000, TestResult)
```

How does this affect your next tester design? If the target tester has to operate for many years to come and requires support for 10+ years, do not even start without first considering interchangeability issues. While your task may be simpler today without these considerations, it may cost you and your employer dearly when the state-of-the-art hardware you selected today becomes obsolete tomorrow.