

IMPLEMENTATION ARCHITECTURE FOR IEEE STANDARD P1641 - SIGNAL AND TEST DEFINITION USING IVI TECHNOLOGIES

Ion A. Neag
TYX Corporation
1910 Association Dr., Ste. 200
Reston, VA 20191
703-264-1080
ion@tyx.com

Matt Cornish
Racal Instruments Group Ltd
29-31 Cobham Road
Wimborne, UK
+44 (0)1202 872800
matt.cornish@racalinst.co.uk

Abstract - The implementation of the emerging IEEE P1641 standard requires software architectures that enable the development and execution of instrument-independent TPSs. The Signal Interface standard developed by the IVI Foundation provides advanced instrument interchangeability capabilities, while using a signal-oriented instrument model. These characteristics make it an excellent choice for software solutions that implement the new IEEE standard.

This paper describes a software architecture that integrates multiple software products through standard interfaces. The integration relies on modern software technologies such as XML and COM. The proposed software solution offers unique and powerful features for TPS development. In addition, its advanced instrument interchangeability capabilities can provide significant cost savings to organizations that must maintain test equipment over long periods of time

INTRODUCTION

The IEEE Standards Coordinating Committee on Test and Diagnosis for Electronic Systems (SCC20) has finalized the development of the next release of the ATLAS standard, formally known as the IEEE P1641 - Standard for Signal and Test Definition [1]. This constitutes a radical departure from previous ATLAS standards, replacing the specialized language with a software Application Programming Interface (API), while preserving the principle of signal-based, UUT-oriented test definition.

Implementations of IEEE P1641 must address a set of architectural and algorithmic problems similar to those of "traditional" ATLAS implementations, including resource allocation, switch path calculation and conversion of signal and switching operations into instrument commands. In addition, implementation architectures must enable the replacement of instruments with minimal changes in the code of the Test Program Set (TPS).

Modern software technologies such as the Component Object Model (COM) and the Extensible Markup Language (XML) support the development of effective solutions for the problems identified before. The standards developed by the IVI Foundation use these technologies in support of instrument interchangeability [2].

This paper describes an implementation architecture for IEEE P1641 using IVI technologies, which builds on a solution originally described in [3]. The proposed architecture integrates commercial software from Racal Instruments and TYX Corporation with a prototype Signal Object Library developed by TYX. The conversion of signal and switching operations into instrument commands is achieved via IVI Signal Drivers.

STANDARDS

The architecture presented in this paper relies on two emerging standards, described briefly in the following sub-sections.

IEEE P1641 Standard for Signal and Test Definition

IEEE P1641 [1] introduces a new way to define signal types, which is more accurate and less ambiguous than that of previous ATLAS standards. The new standard defines a set of Basic Signal Components (BSCs), which provide the most basic signal, event and measurement functions that might be required in a TPS. BSCs such as **Sine**, **Square**, **Sum**, **Clock**, **AM**, **RMS**, and **FFT** are included. The standard defines the functional behavior of these BSCs through the Signal Modeling Language (SML). SML allows signal definitions to be unambiguously interpreted for the purpose of TPS execution, resource allocation, signal synthesis, simulation, etc. To create more complex signals, BSCs can be interconnected through links, attached to the BSC terminals **In**, **Carrier**, **Gate**, etc.

P1641 contains a Test Signal Framework (TSF) to describe reusable collections of BSCs which form a distinct functional block. TSFs are a convenient way to encapsulate the behavior of "traditional" ATLAS nouns. In fact, many examples in IEEE P1641 are based on these nouns.

IEEE P1641 includes Interface Definition language (IDL) and XML mappings for signal definitions, allowing these definitions to be used from a wide variety of programming languages and test environments. This capability, coupled with the signal-oriented nature of BSCs, makes the IEEE P1641 standard truly implementation-independent.

IVI Signal Interface

The IVI Signal Interface specification, currently in its final development stage within the IVI Foundation, defines a set of interfaces that enable the signal-oriented control of instruments [4] [5] [6]. By using a signal-oriented instrument model, the IVI Signal Interface design offers advanced instrument interchangeability features, such as: replacement with an instrument from a different class, or with an instrument that does not belong to an IVI class, or with a combination of instruments with similar capabilities. IVI Signal Drivers make available to their client applications a signal-oriented description of instrument capabilities, in terms of range, resolution and precision.

IVI Signal Drivers expose signal-oriented methods such as: **Reset**, **Setup**, **Initiate**, **Fetch**, **Connect**, etc. The implementation of these methods converts the signal and switching operations into instrument commands or instrument driver calls. This functionality, along with the advanced instrument interchangeability features, make IVI Signal Drivers an excellent solution for test environments based on the IEEE P1641 standard.

SOFTWARE PRODUCTS

The solution presented in this paper integrates three distinct software products, described briefly in the following sub-sections.

Racal Instruments *newWave*

Racal Instruments *newWave* [7] encapsulates IEEE P1641 in a graphically based signal and measurement development and modeling tool. In addition, *newWave* provides on-the-fly, bi-directional editing and translation between its graphical format and XML or Visual Basic (as supported by the IDL mappings of IEEE P1641).

newWave incorporates a set of Streaming Filters utilizing Microsoft's DirectX[®] technology. These filters, derived from the SML of IEEE P1641, use XML signal definitions to provide live simulation of signal models. Because the Streaming Filters are encapsulated as ActiveX[®] components, they can be deployed as part of a third-party application.

Microsoft's OLE document interface, described in detail in a following section of the paper, is also supported, enabling *newWave* to be embedded in its entirety into a third-party application.

TYX TestBase

TestBase is a Commercial Off-The-Shelf (COTS) test executive that enables the visual development and run-time execution of "fault tree" test strategies [8].

The architecture of TestBase enables system integrators to customize and extend the functionality of the product. This capability includes the extension of the data type set, via plug-in COM components called "Custom Data Type (CDT) Editors" [9]. CDT Editors typically implement three interfaces:

1. "Edit" graphical interface: displayed during development, enables the assignment of CDT values to parameters
2. Programmatic interface: allows test procedure code and report macros to access the CDT values
3. "Display" graphical interface: can be embedded as an ActiveX® control in soft front panels of test procedures and in reports to display graphical representations of CDT values

TYX Signal Object Library

The Signal Object Library (SOL) enables the development of signal-based, UUT-oriented test procedures using general-purpose programming languages such as C++ and Visual Basic. The SOL consists of two main parts:

1. A library of COM components, implementing the signal types defined in the IEEE P1641 TSF. Instances of these components can be used in the test procedure code to perform signal operations. The component interfaces contain properties corresponding to signal parameters (equivalent to the ATLAS nouns), as well as methods such as **Reset**, **Setup** and **Measure**, similar to the ATLAS verbs.
2. A run-time engine, supporting the execution of test procedures through automatic resource allocation and switch path calculation services. The run-time engine uses IVI Signal

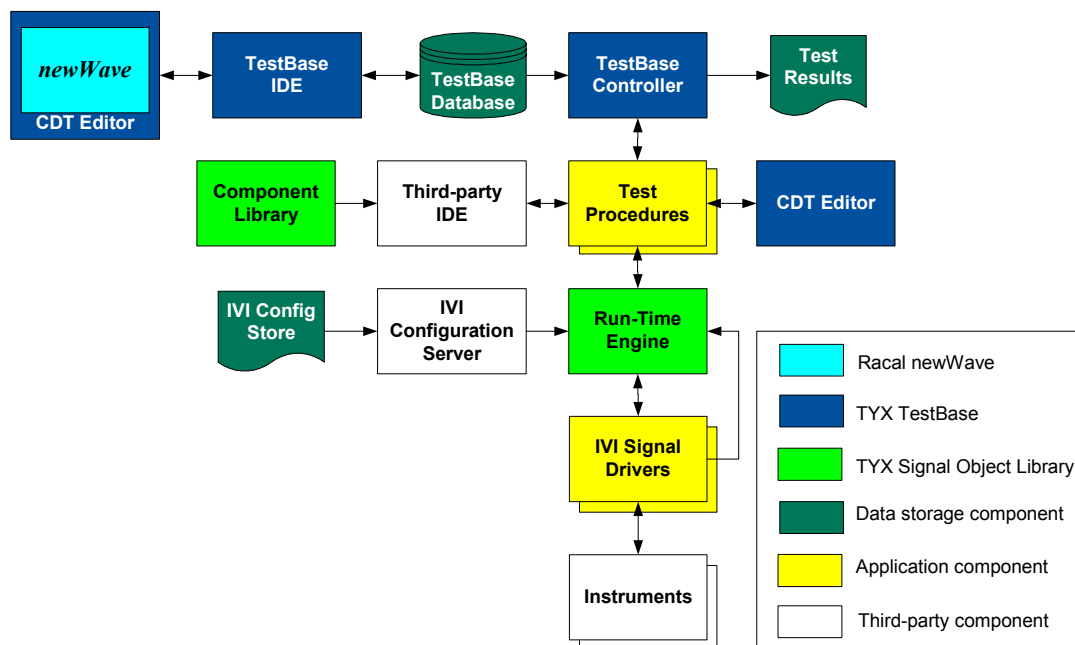
Drivers for converting signal operations into instrument and switch commands.

INTEGRATION ARCHITECTURE

The software architecture of the integrated solution is shown in Figure 1. *newWave* is integrated with the **TestBase Integrated Development Environment (IDE)** via a **CDT Editor** component, which uses the OLE document interface exposed by *newWave*. This enables the storage in the **TestBase Database** of complete signal definitions, including the XML representation of signal parameters, as well as the signal diagrams displayed by *newWave*.

The **TestBase Controller** executes the test strategies developed with the IDE by triggering the execution of external **Test Procedures**. These Test Procedures receive signal definitions embedded in their input parameters. The above CDT Editor is used again, this time via its programmatic interface, to extract from the parameters the signal definitions in XML format. The Test Procedure code requests from the **Run-Time Engine** the allocation of signals, passing as requirements the XML signal definitions.

Figure 1. Software Architecture



The Run-Time Engine compares these requirements against the capabilities of the available instruments, capabilities provided by the **IVI Signal Drivers**. When a match is found, the Run-Time Engine instantiates a **Component Library** object for the requested signal type, connects it to the appropriate Signal object from the IVI Signal Driver [4], then returns to the Test Procedure a reference to the newly created object.

The Test Procedure code uses the above reference to invoke various signal-oriented methods exposed by Component Library object. For example, to configure a source signal the code can call the **Setup** method, passing as argument the XML signal definition received in an input parameter. When receiving such method calls, the Component Library object invokes the corresponding method or methods from the IVI Signal Driver, which in turn performs the appropriate instrument control actions.

OLE Document Interface

The integration between *newWave* and the TestBase CDT Editor is greatly facilitated by the OLE document interface exposed by *newWave*.

OLE document interfaces are used for embedding a document within another document, for example a Microsoft Excel spreadsheet within a Microsoft Word document. Through this technology, the capabilities of multiple software products can be seamlessly integrated into a single application. An embedded document can carry out certain actions, such as exchanging its menu for the host application's menu and serializing its document data, to be combined with the host application's own data.

This technology enables the TestBase CDT Editor to edit IEEE P1641 signal diagrams seamlessly by embedding a *newWave* document. *newWave* presents all the required editing facilities and passes the changes back to the CDT Editor, which in turn transfers them to the TestBase IDE, for persistent storage.

XML Signal Definition Format

The modules of the integrated software architecture represented in Figure 1 use various XML-based formats for exchanging and storing

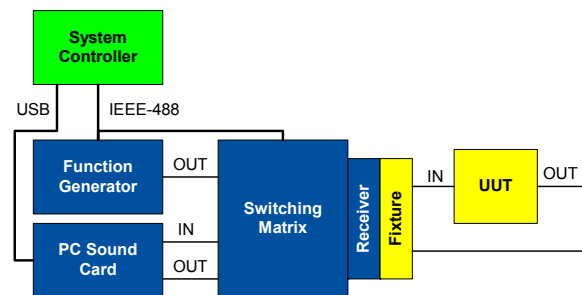
information. This paper focuses on the XML signal definition format specified by the IEEE P1641 standard. As mentioned before, the IEEE standard contains mappings of BSC and TSF signal definitions to XML. These mappings also include the physical types of signal parameters, such as **Voltage**, **Acceleration**, **Time**, etc. These mappings enable any IEEE P1641 signal definition to be exchanged in an easy-to-parse, human-readable format.

In the proposed solution, the above format is used for exchanging signal definitions between *newWave*, TestBase, the test procedure code and the SOL components.

PROTOTYPE IMPLEMENTATION

This section describes the operation of the prototype implementation for a typical TPS development and execution scenario. Figure 2 shows the experimental system used to verify the operation of the prototype.

Figure 2. Experimental System



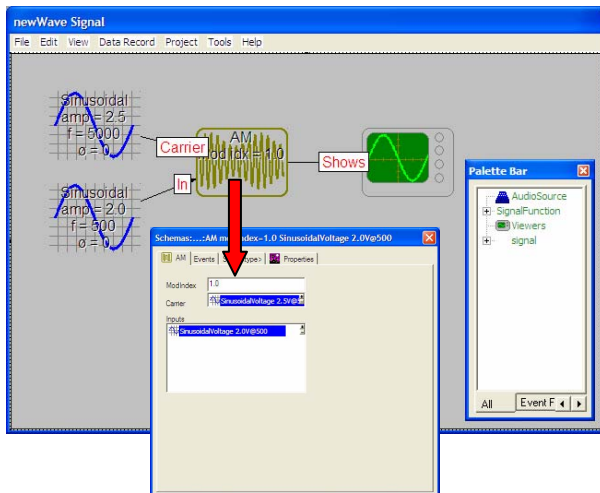
The experimental system includes a “traditional” **Function Generator** and an external **PC Sound Card** controlled via USB and acting as a synthetic instrument. The Unit Under Test (UUT) is a passive circuit containing a voltage limiter and a low-pass filter.

Visual Definition of Signals

IEEE P1641 signals can be assigned to input parameters of TestBase test procedures. To edit the value of such a parameter for a particular test procedure call, the TestBase IDE user opens the CDT Editor, which in turn displays the *newWave* user interface shown in Figure 3. This interface is used for the visual definition of signals through IEEE P1641 diagrams.

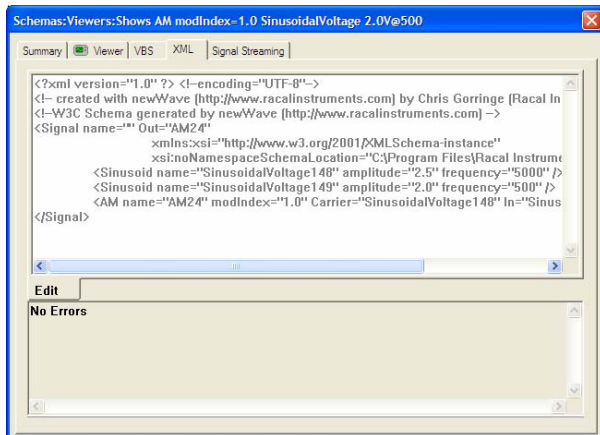
Note: Figure 3 exemplifies the definition of a signal using multiple BSCs. Alternatively, signals can be defined using a single TSF component, for example “AM_SIGNAL” [1].

Figure 3. Definition of a P1641 Signal with *newWave*



After the signal design is completed, *newWave* converts the graphical signal definition into an XML representation, exemplified in Figure 4, which it returns to TestBase.

Figure 4. XML Signal Definition Generated by *newWave*

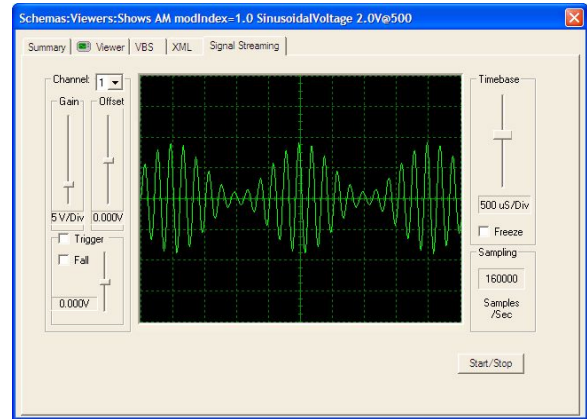


Signal Simulation

The waveform of the previously defined signal can be displayed live, using the Streaming

Filters built into *newWave*, as shown in Figure 5 [10]. This capability enables test developers to adjust signal parameters until the desired signal waveform is obtained.

Figure 5. Live Signal Simulation with *newWave*



Test Execution

For demonstration purposes, the authors have developed a simple Visual Basic test procedure, which applies a signal to the input pins of the UUT shown in Figure 2 and measures the response at the output pins, then displays the stimulus and response waveforms.

The source code fragment that allocates, configures and connects the input signal is shown in Figure 6. This code shows how the SOL enables the development of signal-based, UUT-oriented test procedures with a general-purpose programming language.

Figure 6. Test Procedure Code

```
' allocate stimulus signal
Dim inputSignal As Object
Set inputSignal = rc.GetSignalXML(_
    Source, m_strXML)

' setup stimulus signal
inputSignal.SetupXML m_strXML

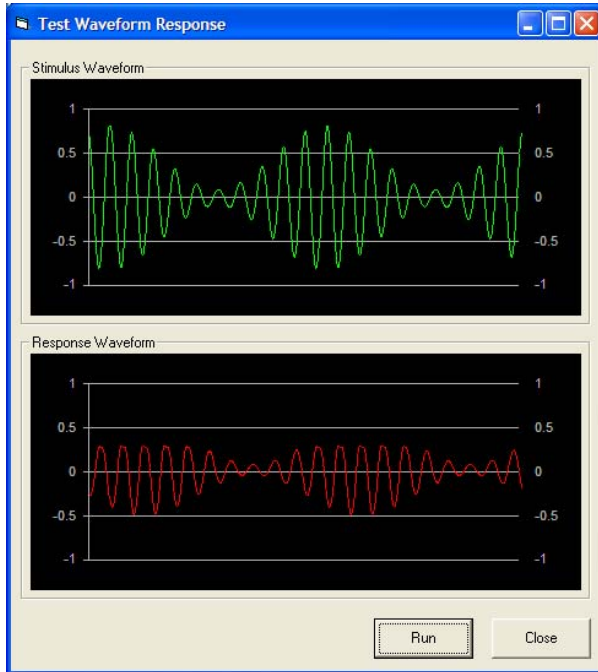
' connect stimulus
inputSignal.Connect "IN_HI", "IN_LO"
```

The object “rc” is a previously created instance of the SOL Run-Time Engine. The string stored in “m_strXML” is the XML signal definition received in an input parameter of the test procedure. This string is passed to the

Run-Time Engine as requirements for the signal allocation algorithm, then to the "inputSignal" object as signal settings.

A call to the above test procedure, with an AM Signal as input parameter, was included in a TestBase test strategy. Figure 7 shows the soft front panel of the test procedure, as displayed during the execution of the test strategy.

Figure 7. Test Procedure Soft Front Panel



Instrument Interchangeability

The execution described before was performed using the Function Generator as source for AC and AM signals and the PC Sound Card as waveform sensor (see Figure 2). Each of these instruments is controlled by an IVI Signal Driver, which supports the signal types and signal roles indicated above.

To demonstrate the interchangeability capabilities of the proposed solution, the authors have extended the functionality of the IVI Signal Driver for the PC Sound Card, adding capabilities for generating AC and AM signals. The system was then reconfigured by removing the entry for the Function Generator from the IVI Config Store (see Figure 1). At this point, the Function Generator can be physically removed from the system. As a result, the SOL Run-Time System will allocate the stimulus capabilities of

the PC Sound Card, instead of the similar capabilities of the Function Generator. During execution in the new system configuration, the waveforms displayed in the soft front panel look identical to those from Figure 7 (within the accuracy performance of the two instruments). It is important to note that the Function Generator was replaced without changing the test procedure code or the test strategy design.

EXTENSION CAPABILITIES

The prototype implementation presented in this paper can be extended as described in the following sub-sections.

Signal Synthesis

For simplicity, the signal generation capabilities of the IVI Signal Driver for the PC Sound Card are implemented in the prototype by calculating the waveform samples and transmitting them to the Sound Card through the Windows Multimedia API.

As described before, *newWave* includes a set of Streaming Filters, which take as input XML signal definitions and can communicate with the Sound Card. This is achieved using the same streaming technology that enables simulation. In this case, however, the signal stream is directed at the DirectX® driver of the PC Sound Card [10]. Future implementations of the IVI Signal Driver will use these Streaming Filters to support a larger set of signal types.

It is important to note that this synthesis technique is not limited to PC multimedia peripherals, since DirectX® drivers might be written to drive synthetic test instrument [10].

Instrument Capability Description

As previously suggested in [11], the capabilities of instruments can be described in terms of the BSCs that they are able to support. For example, a simple RF synthesizer might support **Sinusoid**, **Triangle** and **Square** sources, along with **AM**, **FM** and **Sum** conditioners.

The architecture described in this paper can use such an approach to support the automatic resource allocation functionality of the Run-Time Engine (see Figure 1). In the prototype implementation, the resource allocation

algorithm uses TSF-level capability descriptions provided by the IVI Signal Drivers and either TSF-level or BSC-level requirements provided by the test procedure. Using BSC-level capability descriptions would improve the ability of the algorithm to match requirements expressed at the BSC level. In addition, BSC-level capability descriptions are better suited for describing synthetic instrumentation.

ATML Support

The Automatic Test Markup language (ATML) Working Group [12] is currently developing a set of standard formats based on XML, which will facilitate the exchange of TPS and Automatic Test Equipment (ATE) data between software applications. The authors have proposed the incorporation of the XML signal definition format specified by IEEE P1641 in the ATML schemas.

The open architecture of the software products included in the proposed solution enables them to support for other ATML schemas such as Test Program, Diagnostics, Test Results and Instrument Specifications.

CONCLUSION

The solution described in this paper offers several benefits to TPS developers and to organizations that must maintain TPSs over long periods of time:

1. The ability to define signals visually and to verify the waveforms of these signals through simulation.
2. The ability to transmit signals as input parameters to test procedures, along with other parametric data.
3. The ability to develop signal-based, UUT-oriented test procedures with general-purpose programming languages.
4. The ability to port TPSs across various run-time implementations.
5. The ability to replace instruments or to perform a full re-host with minimal changes in the TPS code; this capability encompasses both traditional and synthetic instruments.

These benefits are made possible by *integrating multiple software products through software interfaces that adhere to industry standards*. In particular, the solution described in this paper

uses XML signal descriptions specified by IEEE P1641, IVI Signal Interfaces, OLE document interfaces and ActiveX[®] interfaces.

REFERENCES

- [1] *** IEEE P1641/D1, Draft Standard for Signal and Test Definition, IEEE, October 2003
- [2] *** IVI Foundation web site, <http://www.ivifoundation.org>
- [3] Ramachandran, N., Oblad, R.P., Neag, I.A., Tyler, D.F., "The Role of the IVI Signal Interface Standard in Supporting Instrument Interchangeability", Proc. AUTOTESTCON, Anaheim, CA, 2000
- [4] *** IVI-3.12: DRAFT IviSig Class Specification, Revision 0.2, IVI Foundation, January 2003, <http://www.ivifoundation.org/groups/Signal-Interfaces/default.htm>
- [5] Neag, I.A., Ramachandran, N., "ATLAS2K and the IVI Signal Interface - the Framework for an Open, Modular and Distributed ATS Architecture", Proc. AUTOTESTCON, Valley Forge, PA, 2001, pp. 23 - 37
- [6] Gal, S., Neag, I.A., "A Unified Interface for Signal-Oriented Control of Instruments and Switches", Proc. AUTOTESTCON, Huntsville, AL, 2002
- [7] *** *newWave*, <http://www.racalinst.com/newwave/newWave.htm>
- [8] *** TestBase, <http://www.tyx.com/testbase/>
- [9] Neag, I.A., Gal, S., Hartop, D., "Data Type Extensibility in Automatic Test Systems", Proc. AUTOTESTCON, Huntsville, AL, 2002
- [10] Cornish, M., Hazlewood, R., Gorringer, C., "PC Based, IEEE Signal & Test Description Standard", Proc. AUTOTESTCON, Anaheim, CA, 2003
- [11] Cornish, M., Gorringer, C., Langlois, J., "Synthesis of Complex Signals on Test Equipment", Proc. AUTOTESTCON, Huntsville, AL, 2002
- [12] *** Automatic Test Markup Language web site, <http://www.atml.org/>