IEEE STD 1641 AND THE DOD AUTOMATIC TEST SYSTEM FRAMEWORK ELEMENTS

Chris Gorringe BSc (Phys), MSc, MIEEE
EADS Test & Services (UK) Ltd.
29-31 Cobham Road
Wimborne, Dorset BH21 7PF
+44 (0)1202 872800
chris.gorringe@eads-ts.com

Abstract - The paper highlights where IEEE Std 1641-2004 fits into the existing DoD ATS framework elements. It identifies criteria for categorizing the affected elements and shows how the IEEE 1641 Signal & Test Definition standard augments other standards such as ATML to enable an architecture that helps fulfill the ATS framework.

The IEEE Std. 1641 is a standard specifically for dealing with signals including sources, conditioning, signal based measurement, events, timing and location. Historically IEEE Std. 1641 came out of the ATLAS2000 development to isolate and use the 'useful' parts of the ATLAS language. This led to the concept of signal modeling to provide a mechanism for describing the properties that made up the nouns and noun modifier components. Signal modeling provides the structures to support signal component usage within a hierarchical signal classification system. By so doing, the signal models provide the basis for reuse that is essential to the evolution of a standardized capabilities vocabulary without a corresponding explosion in nomenclature and complexity. This implies IEEE Std. 1641 is a far reaching standard as it defines several facets to support a wide range of signal applications that cross ATS framework elements.

The paper reiterates what is meant by a Signal Layer in this context and considers an ATS set of abstractions used by a test system and associated resource managers as a method of classification within the ATS framework. The use of signals to define both measurement requirement and capability is a key element in any system looking to provide TPS portability across different ATS platforms, and when coupled with the information content of ATML has an impact on several signal and UUT related ATS framework elements, which helps us demonstrate an architecture conforming to the ATS framework using available standard interfaces.

A FEELING

Do you recall that feeling when you see something for the first time and it brings together those aspects of design, capability and technology and it just fits. I remember seeing my first Apple Mac (Macintosh is those days) whilst at an interview at a research establishment and it just made sense, unfortunately it was so easy to use It allowed me to dropped the clock in the bin, which on that particular model was a bad thing, but I left the interview knowing I'd seen the future. More recently when Google Earth came out, instantly here was something that just made sense, I'm convinced that its approach will change the way we navigate and find information using our computers.

I get the same feeling with Signal Modeling. "It's of its time". The technology, capability, design and desire are all there for Signal Modeling to change the way we do things within the test and measurement industry. This will not be an overnight revolution, there will be no "out with the new and in with the old", but it will be evolutionary, pervasive and relentless. Today we just have to look at the different areas that are looking to embrace signals: Synthetic Instruments, ATML

Test Description and Capability, RAI, IVI Signal, IEEE Signal & Test definition to get a glimpse of where the future is going.

SIGNAL MODELING

The term signal modeling is quite simply defining signals through models. For this discussion a signal includes signal sources, signal conditioning, signal based measurement, digital, communications & busses, events, timing and even locations (connections). The models provide a means to define both behavior and interface parameters, and utilize both composition and hierarchy to allow building increasingly complex signal definitions. Regardless of the complexity of the signal, because they define behavior, signal models represent an ideal component of test requirement interchange.

As such the use of signal modeling has a positive impact on several DoD (and MoD) objectives

- Improve instrument Interchange
- Make ATE more adaptable with no penalty to requirements
- Faster technology insertion
- Improve TPS re-host
- Improve TPS interoperability
- Use model based programming techniques
- Modernize test programming environment
- Capture 'design to test' data

Improve instrument Interchange

Signal modeling provides for the signal description to be transmitted to the instrument, with a "do this" semantics. This allows instruments and subsystems to provide a far wider range of signals, using a common or standardized communication format. Synthetic Instruments (SI) are considering the form of programming at the 'measurement science' level to allow SI to be configured into specific instruments, and therefore the signal description becomes the item that is transportable rather than the instrument being configured. This then leads to better instrument interchange because we can program different instruments with the same signal descriptions.

Note: the term instrument refers to both the instrument driver software and hardware.

Make ATE more adaptable with no penalty to requirements

Adaptability means getting the most form the available assets as well as the ATE's ability to be upgraded.

Configurable instrumentation has a major impact on ATE adaptability.

ATML's Capabilities uses signal model definitions to help describe resource capabilities and characteristics

The use of signal modeling as described in ATML's Capabilities provides for instrument and test station capabilities to be defined such that new uses and signals can be create using existing capability descriptions. I.e. the system can create signals that were not defined or thought of as part of its original requirement.

Similarly, ATML's capabilities are a sum of components so new system capabilities can be derived by adding new instrument capabilities, rather than having to expand out all possible uses, when the systems are first created.

Faster technology insertion

Being able to add single components (instruments or resources) and have them integrate into the whole test system, naturally provides for faster technology insertion.

Improve TPS re-host & interoperability

Re-host relates to the ability of executing an existing TPS on an ATE, which the TPS has never been executed on. Re-host is also necessary when the ATE that currently executes the TPS is updated with additional capability[1].

Interoperability is the ability of systems to provide and accept data and services from other systems and enable them to operate effectively together. *Interoperability* includes systems, processes, procedures, organizations and missions over the life cycle and must be balanced with information assurance. For the purposes of the RAI Working Group, interoperability defines the ability of the RAI element to operate with the other elements in the DoD's ATS Framework[2].

The use of signal modeling with ATML's Test Description, or directly through using IEEE 1641 or RAI directly supports these definitions of re-host and interoperability
This support can be provided at two key levels

1. Data level where ATML Test Descriptions are 'compiled' into optimized runtime code
2. Runtime calls where the signal models are directly implemented by the system. (RAI)

Use model based programming techniques

Answers on a post card please.

The use of model based techniques, allows the model information to be used in a variety of different phases throughout the product life cycle. This is one of the key objectives for reusing test information from design through factory and out into the field support.

In the case of the signal modeling the models can be used:

1. For simulation, at product design.
2. Reused from design through to field support.
3. Transported across systems to aid portability.

Modernize test programming environment

"*Would a change of programming language or design philosophy reduce life cycle costs for test?*"

Within the professional S/W industry there is little doubt that Object Orientated (OO) techniques lead to reduced life cycle costs although there are many critics of the OO approach over the procedural approach, and the method chosen probably has far less impact than say using the best people, or having fast and powerful development tools,

One of the key issues therefore comes down to "*How many test programs fail because of the code and data problems rather than ITA and instrumentation issues?*"

Where test programming has failed or been difficult there have been some very creative programming just because the language chosen was not up to the programming task; however this should have been solved by adding additional functionality into the system, DLLs or ActiveX objects, rather than being confined to a one language solution for all.

In the OO vs. Procedural debate there appears to be established trade-offs and these trade-offs seem to have little impact in the traditional test programming arena:

1. OO reduces use of global data - large programs that share and manipulate global data are difficult to debug - but how many test programs use a lot of global data
2. Abstractions provide for reuse and portability but may have a speed impact - speed can be a big issue for us.
3. Re-use - For test programs our reuse issues are more related to the instrumentation and test systems than for code reuse.

Where users wish to maintain a sequence of signal requirement actions, the choice of OO or procedural language is not going to make a big difference. When test programs become more orientated towards signal simulation, end-to-end to 'Functional' distinct from parametric then OO will offer a definite advantage.

The positive point is that signal modeling, and specifically the IEEE 1641 IDL control interface offers an Object model that can be used by both paradigms for both sequential and parallel testing.

As a program management activity if it really makes little impact as to which method is used, then as an industry the best approach would be to use what is being taught in schools and commercially. This will reduce re- training needs and maximize the pool of programmers who could be available. For existing programs this will have to be balanced against the re-training needs of the existing 'experienced' programmers.

Other tools which would make a major impact and that signal modeling may help support such as self/automatic documentation, requirement tractability, good debug tools that provide the true state of the system, and can convert 'accurately' from requirement "what you want" into design "what you get", in a twinkling of an eye would be a good start.

Note: Using COM and ActiveX objects provided in items such as IEEE 1641, IVI or even RAI, is a good start to changing the languages traditionally used.

Capture ‘design to test’ data

The ability for signal models to be simulated makes them ideal for signal modeling to be integrated within the system design phase. As such the signals used to manipulate and test the designs can be carried forward onto the ATE test platforms. Providing for same data at design to be captured and used in the maintenance and test phase.

IEEE 1641 SIGNAL & TEST DEFINITION

The IEEE 1641 Std. Signal & Test Definition is an ideal platform to realize signal modeling within a standard framework.

This standard provides the following key items:

- Data model interface in the form of XML
- API Interface in form of Object Model interface (IDL)
- Library definitions that allow reuse and structural composition

These description formats allows signal definitions to be described in XML and used in emerging information standards such as ATML, and also provides for runtime interfaces for use with traditional test programming environments, which in both cases describe the same signal and test requirement. The library definitions allow users to build up complex signal requirements and tests for reuse as domain specific tests.

IEEE 1641 is a standard specifically for dealing with signals. Historically IEEE Std. 1641 came out of the Atlas2000 initiative to isolate and use the ‘useful’ signal parts of the C/ATLAS-716 language. The result however is far more reaching than just replacing C/ATLAS, it provides a framework that defines signal abstractions across all its uses; from design through test, capabilities and even new technologies such as synthetic instrument programming.

The concept of signal modeling provides a mechanism for describing the properties that made up the nouns and noun modifier components of the ATLAS language. Signal modeling provides for the structures to support signal component usage within a hierarchical signal classification system. By so doing, the signal models provide the basis for reuse that is essential to the evolution of a standardized capabilities vocabulary without a corresponding explosion in nomenclature and complexity

IEEE Std. 1641 is quite a far reaching standard as it defines several facets to support a wide range of signal applications, including:

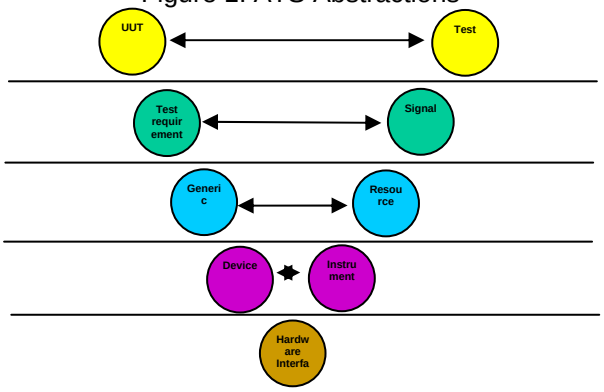
- Defining Physical Types, Signals and Measurements
- Defining and Creating Signal library components for re-use (in the form of TSFs)
- Defines a Resource Manager interface for signals for test programs or applications to create and request signals from a system or resource
- Defines a Signal Interface for test programs or applications to control signals
- Provides a interface (through signal based measurement) to test the result of measurements against limits and retrieve a PASS/FAIL/GO/NOGO/HIGH/LOW condition
- Defines a default Test Signal Framework library that extracts the commonly used nouns and noun-modifiers into a reusable library
- Provides a signal XML schema for defining static signals and TSFs in XML for re-host and porting
- Defines a macro language that can be used to capture Test Requirements derived from Signals
- Provide a mathematically supportable basis for signal definitions and simulations
- Defines the physical point(s) on the UUT where the signal is to be applied or measured.

ATS FRAMEWORK

When considering signal modeling within the ATS Framework, it may be worth reiterating what is meant by the signal layer in this context. If we consider an ATS set of abstractions used by a test system and associated resource managers

- UUT Test
- Signal and test requirement
- Resource
- Instrument

Figure 1. ATS Abstractions



Test

The **Test** abstraction deals with the notion of performing an item to get a result, This is most prevalent in the Diagnostics standards such as AI-ESTATE and where the test generally offers a convenient (but not the only) abstraction to manage for the purposes of making a diagnosis of a unit or situation prior to a repair. In this instant what the test does, is not really of interest the real information is the test result and the corresponding failures/repairs the result identifies or eliminates.

Signal and Test Requirement

The **Signal** abstraction is a high level description of what the test is doing, without needing to understand how the test is performed on the incidental hardware and system. It is also the recognized abstraction at which to define test requirements and describe the behavior of the UUT, since it allows the "what is required" to be captured independent of the system that will implement it.

Signals are foundation elements for test operations. A parametric signal can be defined by:

- Its *waveform function*.
- Its *location* on the UUT.
- Its *lifetime*.

The *waveform function* is the physical signal that could be represented by a mathematical function that describes the signal as a function of time (generally).

The waveform function is defined by a set of parameters that fully determine the characteristics of that function and of the resulting waveform, over its lifetime. The waveform function can be continuous or piecewise continuous. In the latter case, discontinuities are permitted between the continuous (in time) pieces. A waveform can be referred to as a Signal, even though no specific location or lifetime is specified. Location and lifetime are test-specific parameters and are in addition to the waveforms attributes, methods and properties.

Waveforms are considered to be instances of one of several signal classes which have been defined for the test. Any signal object has one or more attributes that contain information concerning various parameters of interest. In the case of a signal, examples of these attributes may include:

- Amplitude (Voltage, Current, Power)
- Periodicity (Period, Frequency)
- Phase

Test Requirement

Requirements are concerts of signals where each signal must appear at specific locations and at some defined time period. These signals may or may not have to be related to each other in time. Test Requirements only allow us to observe electrical and physical manifestations that the UUT exposes as the test requirement is applied.

Resource

A **Resource** is some logical entity that fulfils a common role, at the time of asking. Resources can be dynamic, i.e. they can be created and destroyed on demand, and do not have to exist for the lifetime of the test system, or for that matter the test program. An example would be creating an ILS sub-system out of two other devices, e.g. RFSynth and Arb. Another example is that we may have one SI device supporting multiple IVI class drivers. Each class driver instance is a resource based on one SI device, since it provides the IVI class necessary on demand.

Resource interfaces are represent by standards such as IVI classes, MSS IVI-10 and the IEEE 1641 *ResourceManager* Interface such as the ***Require*** method..

A resource is the top level layer or wrapper that a signal interface can communicate with. A resource can be composed of devices and or instruments and their drivers. A resource in general has its own interface software. The interface of a

resource's function must be known and its definition traceable to a standard (e.g. 1641) for interoperability and resource management. In the simplest configuration a device or instrument could be a resource.

Examples of simple resources would also include DMMs RFSynthesisers, AnalysisLibraries, class interfaces, where they represent generic function. At this point note tat resources have both signal and resource class interfaces, providing an interface above and. at the resource layer

Instrument or Device

A **Device** represents a more physical configuration role. A device represents a piece of hardware or software that performs a role which is indivisible, with a specific interface.

An instrument is the specific piece of hardware (with driver, model number, S/N, etc.) that is configured into the ATS system, that supports one (or more) Devices. Examples of instruments would be HP4060, or RI3251.

We group Devices and Instruments because their control interfaces are the same.

Resource Managers

The traditional notion of a **Resource Manager** is to provide information on instruments, devices and to create & destroy dynamic resources. From the previous discussion, the resources could be used in anything from a signal-programming device-class interface or even a raw instrument programming device, these may all be necessary support features of a resource manager. There may be multiple resource manager roles depending on which abstractions are used.

So in an ATE system that utilizes UUT signal descriptions instead of device control commands, a resource management service is required to perform the following typical functions:

- Map the signal requirements to equivalent resources / instruments.
- Allocate the appropriate resources.
- Configure the ATE to accept the appropriate information.
- Maintain status of the dynamic / static ATE configuration.

1641 AND THE ATS FRAMEWORK ELEMENTS

The ATS Framework defines several elements each of which fore fills a unique roll, and where the goal is to identify standards to satisfy that roll. The range of capability offered by 1641 means that it has an impact on several of these elements, as it address the complete use of signals. The following identifies the ATS elements and the potential use of 1641 to help satisfy their requirements

UDI (UUT Device Interfaces)

The use of TSF libraries allow standard signals tests to be defined for UUT technologies. Full demonstration has yet to be given. It would be envisaged that the UDI element would require (for a particular UUT technology) a set of parameterized tests/signals that could be used to test a UUT of that type, for example, for Radio they might be Signal To Noise, Sensitivity, Selectivity, Gain Flatness, Spur search etc..

IEEE 1641 Annex J, XML for TSFs[3] and Annex F IDL for TSFs[4] defines how such tests and signals can be defined in XML, and once defined provides the interface (IDL) that test programs can use to program such tests.

IEEE Std. 1641 TSFs libraries and the standardized 716 TSF fulfill most of the software interfaces required for this element

UTR (UUT Test requirements)

Defines UUT requirements in terms of signals or signal properties to be measured based on the requirements of the UUT and not on the properties of the test resources.

Traditionally, this element may have been fulfilled by ATLAS. Now we would expect the Test Requirements to be met utilizing IEEE 1641 specifically either using Test programs incorporating signals or using the specific IEEE 1641 Annex H (Test Procedure Layer)[5] and Annex G (Carrier Language Requirements)[6]. Further these are encoded into the ATML Test description that uses 1641 to capture its 'signal requirements'.

The expectation would be not only use the IEEE 1641 TPL to capture and define the requirement

but to utilize the UDI XML TSFs to provide the extensible signals and tests required to test the UUTs

The TPL (with UDI) completely fulfils this element. It does provide a 'run able test requirement'. But may want to have an XML test requirement in which case we need to look at ATML Test Description, as added extra sequencing and program information.

TPD (Test Program Documentation)

Where the Test Program Documentation is a description of the UUT signals to be applied and a description of the signal attributes ranges and errlmt. The IEEE 1641 XML signal and test description or the actual TPL English language provide a excellent vehicle to document the test programs. Also if the UUT has a special function and set of attributes a XML TSF model would allow one to unambiguously define that UUT test requirement.

The XML format provides a convenient way to process the information into a dynamic web based documentation system using style sheets and HTML documents.

The element should be fulfilled by using the 1641 XML signal descriptions (with the UDI TSFs) and augmented with the ATML Test Descriptions to define flow and special conditions.

RAI (Resource Adaptor Interface)

The use of Signal definitions allows the match between the UUT world and the ATE world to be brought together.

Signals are a key element to achieve this and the signal definitions shall utilize the XML static definitions defined in IEEE 1641 Annex I and Annex J.

There is discussion of whether the IEEE 1641 connections should be extracted into a separate location field, and whether the IEEE 1641 events should be extracted into a separate Timing requirement. In my own mind, we need to have extra information in the RAI such as the ITA information, so having important information at a higher level. I see this as a necessary feature to make the standards suitable for the problem they are trying to fix.

The key part here is to provide an addition to IEEE Std. 1641 that performs two roles

1) Mandates the use of signal, location and timing as a XML unit (perhaps extending the <Signal …/> concept) to create a *parametric signal*
2) Provides a structured hierarchy of such *parametric signals* to represent a Test Requirement

RMS (Resource Management Services)

The RMS role is to provides a minimum service for the Resource Manager, and the XML signal definition to allocate resources using signal definitions and resource descriptions.

The Resource Manager Interface (IResourceManager) defined in IEEE Std. 1641 is the minimum interface required to provide a signal resource interface.[7][8]. It fulfils the basic signal resource manager interfaces required by a test program or signal application.

RMS still needs to define services for **Resource**, **Device** and **Instrument** management.

Briefly the role of a resource manager interface needs to be scalable, at one level we want to be able to ask an individual device or resource can it achieve this signal requirement[9]. At the next level we may want to ask a group of instruments, as in a Synthetic Instrument" can it provide these coupled signal requirements, but using the same interface. Finally at a ATS or sub system level the resource manager will be expected to select resources that not only can deliver the signals but also provide them to the correct locations, correctly calibrated, and while other events are concurrent. The key item is that these are all the same interface what changes is a matter of scale, we do not want to change the interface however the requirement is to use it on different size problems.

Within IEEE Std 1641 the resource manager interface, provides a signal that will become real when the signal is **Run**. As such the actual (final) allocation can be differed, until the signal is needed. So if we allocate without all the information were not actually getting to a real resource, its just an empty wrapper to be fulfilled. Obviously the downside is if you can not do it a runtime error occurs. If the signal contains

location (Connection) information and Timing (event) information then we can jump straight to the chase and provide the real deal.

The concept of scalability and the ability for different objects to expose the IEEE 1641 ResourceManger Interface was inherent in the design of the standard e.g. a DMM would expose a ResourceManager interface, this would allow a resource manager to ask the DMM if it could for fill a part of a signal requirement-at its ports, and then ask the switch if it could connect such that the DMM is connected to the UUT; potentially changing the signal description to compensate for system effects.

A resource manager may also have to know if it can provide a resource within a certain time period else a test program might be attempting to manage that aspect and impose a test platform dependence, reducing the ability to re-host.

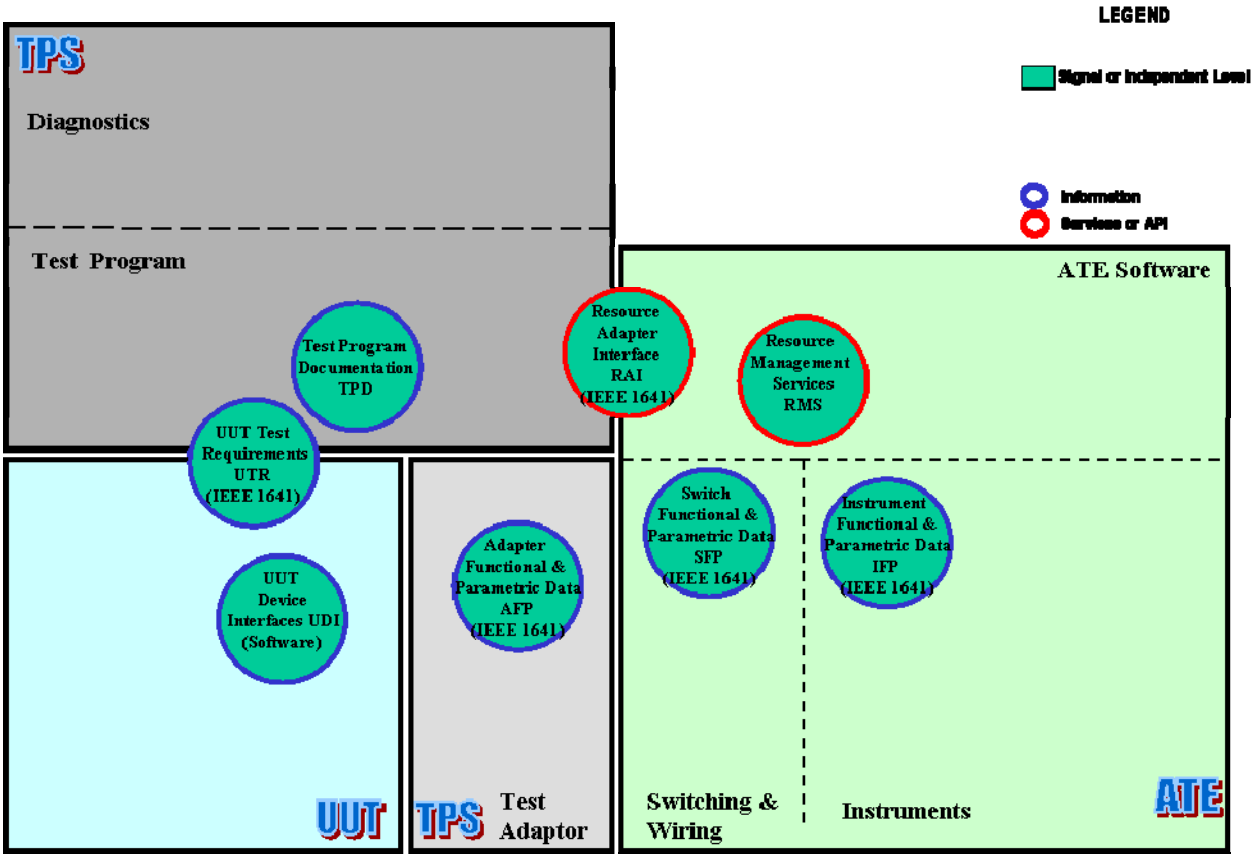
IFP (Instrument Functional & parametric Information)

Provides a signal foundation by which instrument resources can express their capabilities in terms of signals, IEEE 1641 is used within the capability descriptions to support this information. In addition the basic signal definitions and TSFs will also be used in such a capabilities description[9]

The ability to map like capabilities of instruments in a standard unambiguous way requires the application of IEEE 1641 SML/TSF/BSC signal definition framework.

This is being closely aligned with the ATML Capabilities description, which is being based on the IEEE 1641 signal descriptions. IFP will build on ATML Capabilities which will use IEEE 1641;.

Figure 2. ATS Elements utilizing Signal modeling concepts



CONCLUSION

Given the wealth of ATLAS TPS - how are we going to support these within a signal modeling framework whilst creating new TPSs?

In conclusion with so many aspects and choices how do we move forward to take advantage of signal modeling and dependent standards such as ATML and RAI. Within the industry both UK and US its recognized that a major TPS offload will not happen so underlying it all is the question - *How does signals fit into the current systems?.*

To address this issue a couple of ground rules would be useful, to try and outline the different aspects and 'ways of use' that signals are currently perceive being needed for, or used in.

1) There is not going to be a TPS update project to convert current ATLAS programs into something else. Therefore existing systems will remain as they are. - This is also coherent with any MoD statements, where existing TPSs are not being converted or modified (in fact on programs such as DDR their legacy runtime environment is being ported onto the new platform - emulation of the original systems - one new platform - multiple legacy platform emulation support)

> *Note: This implies nothing for new TPS programs, that could utilize new development environments such as .Net but utilizing the same runtime/support tools.*

2) There is a need to support multiple approaches.

Across the user base and even within the DoD and MoD there appear to be different communities which want different things, at this point no judgment is being made on which approach is better or more important, what is identified however, is the requirement to support them all, without escalating parallel or stove-pipe solutions.

- Continue with current ATLAS and build on what is there now.

- The need for faster test programs, both to diagnose the fault quickly and reliably, but also not to have long (3 hour) Standard Serviceability Test programs.

  > *Note: As an aside in the UK the SST TPS is generally run at least twice. Once to find the initial fault (No Integrated Diagnostics) and once to show the unit is serviceable. A Net Centric approach to repair could half these times by excluding one of these runs, and could be achieved by changing the process using the same TPSs. rather than changing the programs*

  Faster test programs, save time (and therefore cost) in the weapon life cycle, removes resistance to their use and therefore improve efficiency. Faster test programs, have historically come from optimizing the TPS with knowledge of the ATE and instruments, by utilizing specific features and techniques.

- There is also a demand for runtime signal test programs (with the RAI). This may be using new development environments such as .NET., where users use modern programming languages to build test programs, and use Signals to control any test resources, directly from the test program.

  There are a variety of derivatives for this where the signal interface could be in the form of a new RAI, Interface, an IEEE Std. 1641 Interface, an IVI Signal Interface, or moving away from signals, Test programs using IVI Class Drivers, and IVI Drivers directly. However at this point the IVI Driver discussion is potentially outside what is really accepted as a supportable TPS solution.

- Augment Test Programs with ATML Test Description Information. In ATML the approach is taken not to recreate a test program in XML, but to capture the test description information and sequencing information separately. The intend is that this information could be for a variety of purposes.
  - Represents a standard TRD in *HOW to test a UUT*
  - Describes the entry points of a test program
  - Provide information that can be used to Generate a Test Program.

3) New non TPS related uses for Signals in the programming of Synthetic Instruments

This is probably jumping the gun, but if SI provides the ability to create a plethora of measurement and signal capability, Then there is required a way to program up a specific 'capability' across all SI configurations. For example using SI,

there may be a requirement to tell the SI (sub system) to become an IFF transponder and measure the pulse characteristics. That description, we would want to reuse across all SI systems that can perform that role, thus maximizing reuse and TPS re-host across such systems. The only real way to do that across systems, is to abstract the "what is required" into a signal abstraction, that the SI (sub system) can interpret and implement on behalf of the TPS.

How does Signal Modeling help?

We do not want to throw away what we do and have today, but we'd really like some step change to improve for the future. Ideally what we want is evolution rather than revolution.

The first change I'd recommend is the current practice of jumping to NAMs as soon as the signal requirement gets too complex or complicated. With only the NAM the ability to understand, maintain and re-host the TPS becomes very difficulty.

- **Step 1** - Signal Modeling can be used to formalize this current practice. rather than prevent it, when the signal requirements fall outside the ATLAS. a new signal description is developed using the IEEE 1641 Std. That describes the signal(s) needed for testing the UUT (by the Signal NAM). This is best achieved by making use of the XML TSF schema that defines the signal's interface and behavior, and ensures that the signal is documented for future re-host or support.
- **Step 2** - Having described the signal we can further describe the signal(s) interaction with either the ATML Test Description using the 1641 TPL action verbs to sequence the signal actions or use the RAI XML description (directly) to describe the test requirement sequences.
- **Step 3** - Finally we can implement the Signal NAM directly by making use of the IEEE 1641 Std IDL or the IVI Signal IDL or RAI Interface (depending on which runtime system is supported, and any speed optimization that is required) to write or translate code from our XML descriptions into runable code.

In this way we build on a established practice of packaging code in NAMs but using the new and emerging standards, that allows us to support changes, improvements and migrate gracefully into a signal modeling concept.

I should point out that not all the steps need to be implemented to achieve a benefit, step 3 may be done by hand but using step 1 & 2 as a requirement, this approach would be suitable on today's system that do not have the new interfaces implemented yet.

Adding runtime interfaces (IVI, RAI or IEEE 1641) can be a managed program, after steps 1 & 2 have been introduced. The simulation ability of the TSFs means that the signals can be used as requirements and compared against the test program's signal NAM to validate the requirement against the test program, whether or not step 3 has been implemented.

Now we have a portable process for creating signal NAMs we can look to what development languages and environments to develop the signal NAMs within. For Windows based systems, systems that support DLLs or ActiveX or COM provide ideal interfaces through which these signal NAMs can communicate. This allows the signal NAM code to be developed in a range of languages, and start to introduce the OO techniques as a modernized test program environment.

A key benefit comes with the automated step 3, where we have a signal runtime system, and/or automated translation process that takes our ATML & TSF description and produces code - targeted at the specific ATE for fast test programs. Since we continue to be signal based this could be back tracked into the ATLAS tool set such that the TSF signals and ATML test descriptions become new ATLAS nouns that extend the station's sub-set. This approach could be attractive to users wishing to maintain a single configuration item (ATLAS file) for the TPS, but as we see later there are benefits by keeping the two approaches separate for the time being.

The speed of testing can be a major problem, not only do we have the runtime of the test program, but also the number of times we need to run that program on a repair cycle is significant. Using NET centric repair, the previous test results and failure information is provided at the start of the repair stage, this allows the system to dispense with the initial SST, and jump straight into an appropriate diagnostic entry point. For an existing ATLAS program the entry-points are fixed, however returning to our NAM discussion the signal NAMs provide alternative entry points, that by rebuilding with different wrappers or using a general purpose wrapper may be called out of turn

as a separate entry point by a test executive, this allows us to use signal NAMs based on ATML and TSFs to help increase the entry points used for diagnostics.

We recognize that at this point our signal NAMs have become reusable tests packages in some executable form that can be used by both test executives and the traditional ATLAS systems, with a Signal and ATML description. So for new test programs it is only a small step, to make more of these signal test packages, defined from ATML, that support both SST and smart and integrated diagnostics programs. By rebuilding the ATML and TSF information we can re-host these signal test packages across ATEs (O, I and D) to reuse test procedures on different ATE platforms running different instrumentation, but optimized for that environment.

Finally creating test descriptions by hand and developing diagnostic is really something we want to automate. The use of simulation has been very successful in the digital field, but never quite taken off with mixed or analogue signals. By extending this approach into a UUT modeling design and test, and utilizing the simulation capability of the signals and thus the signal test packages, we can combine test procedures and UUT modeling to provide path from UUT design into automatic test program generation though simulation to provide both SST and fault diagnostics as an initial start into integrated diagnostic solution.

A vision too far or something within our grasp

REFERENCES

[1] *TPS Red Team Package*, 2004 DoD
[2] *Directive 4630.5, Directive 5000.1* , DoD
[3] *IEEE Std. 1641 Annex J, XML for TSF for ATLAS*, 2004 IEEE
[4] *IEEE Std 1641 Annex F IDL for TSF for ATLAS*, 2004 IEEE
[5] *IEEE Std 1641 Annex H Test Procedure Language (TPL),* 2004 IEEE
[6] *IEEE Std 1641* Annex G Carrier Language Requirements, 2004 IEEE
[7] *IEEE Std 1641 Annex B Basic Signal Component (BSC*), 2004 IEEE
[8] *IEEE Std 1641* Annex D IDL Basic Components, 2004 IEEE
[9] *ATML Capabilities Requirements*, 2005 SCC20 TII,
[10] *Instrument Description Capabilities Versus Ports*, SCC20 TII ATML Capabilities Group