A DOD MANDATED RESOURCE ADAPTER INTERFACE (RAI) STANDARD IN REACH

Hugh A. Pritchett
Analysis, Integration & Design Inc. (AIDI)
1600 Sarno Rd Suite #208
Melbourne, Florida 32935
321-253-9919
HPritchett@AIDInc-usa.com

Abstract - A realistic recommendation for Resource Adapter Interface (RAI), complete with a path for evolving to more comprehensive incarnations, is emerging from an industry supported working group. The RAI working group was formed in July 2005. The group of industry experts convened to attain consensus on what RAI is, to review existing candidate standards, and possibly to recommend one or more candidates for requirement in DoD procurement policy. The RAI topic has always been a controversial one that tests the metal of many existing standards, products and implementations. In achieving an industry consensus, barriers arise from vestments and mindsets surrounding existing products, standards, and ideas. The RAI working group moved forward and passed these obstacles by defining a set of terminology, goals and requirements that ensured consistent meaning in discussions. The recommendation that is materializing promotes a technology and standard that accomplishes two objectives. First it will provide a means of providing an RAI that can be realized with existing technologies. Secondly it will enable a path to providing maximized test application portability by recommending a standard that provides the evolutionary path to a maximized RAI. To achieve this, the working group is soliciting standards committees to embrace and incorporate RAI requirements. These new requirements will facilitate the maximized test application portability when future product developments embrace the complete standard. Recently published research findings that discuss what has hindered industry from achieving a maximized RAI are presented. The paper will discuss three abstractions that apply to RAI and which one has been selected in the recommendation and why. Various software technologies applicable to RAI implementation are also discussed. The paper also discusses the consensus definition for RAI, some history, applicability, and shortfall associated with some of the major existing products, paradigms and implementations. Finally, the paper will provide insights into the future vision for what RAI can be moving forward.

Understanding What RAI Is

Original RAI goals were developed from a DoD sponsored forum know as the Critical Interfaces Working Group (CIWG). The ATS Framework Working Group has maintained and refined RAI related information since the CIWG work was completed. The RAI working group adjusted and refined the goals maintained by the ATS Framework working group in their efforts to identify standards that might satisfy the RAI need.

RAI goals sound simple from the outside:

1. Provide complete test program interoperability upon capability sufficient systems.
2. Provide flexibility to accommodate new and potentially unseen advances in test needs.
3. Reduce obsolescence costs related to Automatic Test Equipment (ATE) systems replacement and test program rehost.
4. Reduce obsolescence costs driven by test system asset replacement.

A technology that successfully minimizes the costs inherent in test program interoperability scenarios continues to elude the ATS community. In addressing RAI, various challenges immediately arise. The RAI working group quickly realized that terminology regularly used in the industry could also mean different things in other contexts and in different architectures. In particular, it was seen that a set of terminology needed to be developed and agreed upon before a definition and defined set of RAI requirements could be solidified. The agreed upon terminology and requirements are available from the author or other RAI working group members. The industry consensus RAI definition follows:

RAI is the software interface that lies between test programs and instrument access layers which ensures ATE independence. The interface is meant to provide a conduit for test instrument related information only. Other test platform assets such as monitors, printers, and others are not within the scope of RAI.

This definition is a major step toward DoD acceptance of a standard or set of standards that can be mandated in acquisition policy. The intent of the terminology definition is to eliminate the ambiguity that immediately arises when RAI discussion is pursued. The Working Group found that once the terminology was established, the goals and requirements were readily produced and agreed to. One factor that contributes to the difficulty related to agreement on terminology is the existence of several architectures and software abstractions that address RAI issues. Some of these exist on legacy test platforms and some are emerging in the industry. The abstractions and their underlying architectures use terminology and information in different ways that influence mindsets in alternative directions. Often the terminology is used with alternate meaning when the contexts of discussion on topics within the same architecture change. The next section discusses test industries' predominant software architectural paradigms that have standards or emerging standards with an interchangeability focus.

System Architectural Paradigms

Abstractions derived from basic underlying test system concepts are central to understanding the industry's predominant architectural paradigms. Each of the abstractions constrains the test developer's interactions in an attempt to achieve a level of commonality at the test program source code level. The abstractions also allow the architecture to work at other user transparent levels to achieve interchangeability in more subtle ways.

The Instrument Paradigm

The instrument paradigm is a conceptual mindset where instrument drivers or lower level instrument access layers are communicated to through higher levels of abstraction software. The abstraction in this paradigm is one that generalizes function calls or classes of instrument functionality to make them common among disparate instrumentation supplied by differing vendors. The abstraction represents a more standard way of communication to instrumentation than the ad hoc driver interfaces often supplied with instrumentation. There are several examples of the instrument paradigm in use today, they include:

- VXI Plug and Play (VXI PnP),
- Interchangeable Virtual Instruments (IVI)

In the VXI PnP implementation, certain functionality in the instrument Application Programming Interfaces (APIs) is commonized so that calls between different vendor's manifestations have a similar look and feel. However, when rehost or instrument interchange is required, test program source code must be altered as the look and feel is only partial. Since TPS source code must be altered and reintegrated, it can be seen that VXI PnP does not approach 100% independence.

In the IVI implementation, categories of instrumentation are commonized using standardized APIs. The APIs support functionality that is general among the specific category of instrumentation for which they are defined. In addition, Common Object Model (COM) technology is employed to provide a level of indirection that removes the user from having to know the specific instrument that is being accessed. Using the IVI standards a user can achieve some advanced level of test program and instrument interchangeability. The level of interchangeability attained is dependent on several factors including:

- Ability of the instrument vendor to provide functionality consistently within the predefined IVI generic instrument class APIs.
- The ability of the generic instrument class APIs to achieve stimuli and measurements demanded by test requirements for specific Units Under Test (UUTs).
- The users adherence to the generic instrument class APIs.

When the IVI architecture is employed, instruments are identified using logical names when reference is made in source code. The logical names are tied to instrument specific information found in an eXtensible Markup Language (XML) test platform configuration file called the IVI configuration store. The logical name arrangement allows static system configuration files to be altered to point to any compliant instruments that can provide the same generic instrument class API vendor offerings. This implementation allows users instrument interchangeability without source code recompilation. But when actually employed, users find that IVI does not provide complete interchangeability. The difficulty arises because adherence to the generic instrument class APIs is often difficult or impossible when complex test requirements need to be satisfied. The IVI implementation provides a means to directly access instrument functionality and bypass the generic instrument class APIs. Interchangeability is immediately lost when functionality is accessed in this fashion. There are significant numbers of instruments and functionalities that are not covered by IVI generic instrument classes at all. TPSs that require functionality from non-IVI supported instruments must access them in an instrument dependent way. Typical DoD users have robust requirements that demand the full functionality of the instrument and have found that IVI does not achieve the maximized interchangeability demanded by their domain. IVI misses the interchangeability target in highly demanding arenas. Figure 1 below is a depiction of how IVI misses interchangeability needs. In the Figure, instruments are depicted as rectangular collections of functionality represented by various shapes. Drivers are shown as focal points for instrument capability. Generic Instrument Classes (GICs) are shown as groupings of common capabilities.

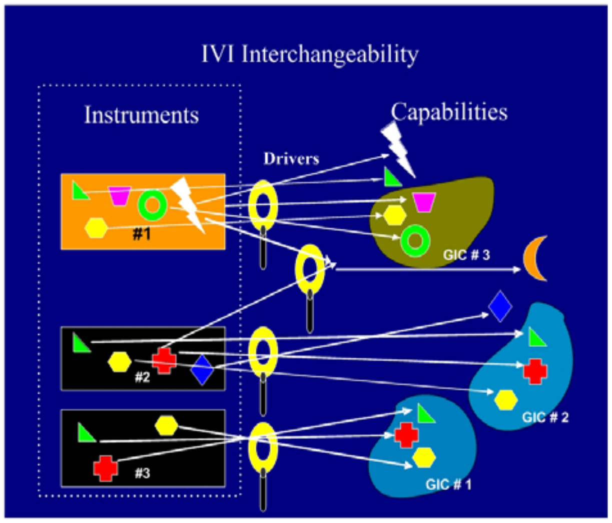
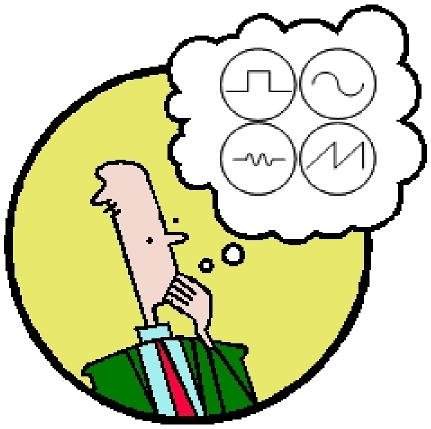


Figure 1

Notice that in Figure 1, GIC #1 and GIC #2 are identical because they expose the same functionality, where GIC #3 is a separate class type that exposes a different collection of functionality. Since GICs #1 and #2 are the same, instruments #2 and #3 can be utilized interchangeably. It does not matter if the instruments are physically the same type or not as long as the GICs that support them are the same. From a user perspective, as long as a common GIC provides the needed functionality, instruments #2 and #3 are the same. The Figure also shows that instrument capabilities often fall outside GIC definition. When these non-GIC capabilities are needed, the functionality must be accessed through the instrument driver itself and not through the GIC where interchangeability can be obtained. Again referring to Figure 1, if a user required the functionality represented by the diamond then only one instrument (instrument #2) could perform the needed function and instrument #3 would no longer be interchangeable with instrument #2. In the light of test program and instrument interchangeability, the instrument paradigm breaks down because it does not completely address significant test platform resources that must be utilized in complex scenarios.

The Signal Paradigm



The Signal paradigm embodies a mindset where tests are conceived as signals that are applied and detected without the concept of instrumentation being introduced. Two real world examples that utilize the signal paradigm are:

- ATLAS
- Institute of Electrical and Electronics Engineers (IEEE)-1641

Both of these examples are similar in their use and application. ATLAS provides its own procedural language for achieving the signal application, measurement, and sequencing. IEEE-1641 provides these features in conventional procedural programming languages. Both examples embody a similar signal model. However, the IEEE-1641 signal model provides more information in terms of instrumentation, path connection, and inherent test system signal conditioning than its ATLAS predecessor. The intent of both ATLAS and IEEE-1641 are to allow the developer to envision the electrical manifestations required at UUT pins and allow the user to procedurally sequence and control those signals programmatically. The concepts were developed specifically to achieve test program and instrument interchangeability. ATLAS has been embraced for more than 20 years by the DoD and commercial airlines. ATLAS has provided some level of test language standardization which helps when test programs need to be migrated. But over the years the ATLAS and ATLAS based languages like IEEE-1641 have not proven to deliver complete test program or instrument interchangeability. There are several reasons why ATLAS isn't 100% interchangeable. These include:

- ATLAS provides the ability to call Non-ATLAS Modules (NAMs)
- Nearly all instances of real world ATLAS are variants of the standard

Several lesser known, but no less important factors that impair ATLAS interchangeability are:

- Restricted ability to determine UUT to signal source connection from ATLAS source code
- Implied information in key test requirement areas including:
    - Timing
    - Location
    - Path bound signal conditioning
    - Others

Figure 2 below depicts a typical ATLAS TPS rehost effort

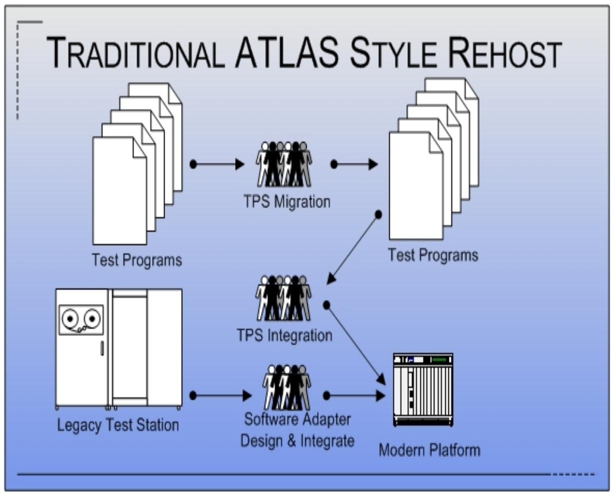
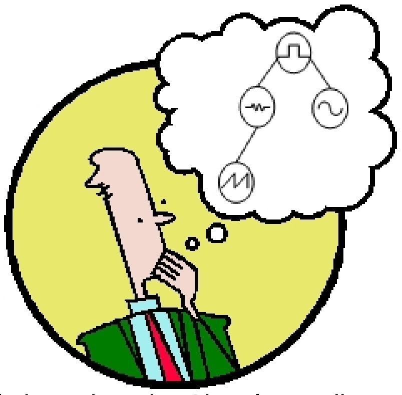


Figure 2

Even with these limitations, ATLAS and the signal based paradigm is the only existing, viable, and well supported interchangeability mechanism that encompasses the gamut of test system capability. ATLAS also provides the highest level of interchangeability available in existing tools because its underlying paradigm does not ignore any test capability as does the Instrument paradigm. At this time, ATLAS and signal based architectures are the only choice when the highest level of industry standardized interchangeability is required.

The Signal Requirement Paradigm (SRP)



Partially based on the Signal Paradigm, the Signal Requirements Paradigm directly addresses three impediments present in Signal Paradigm architectures that cause transportability difficulty. The SRP suggests that Test Requirements are actually concerts of signals that occur at defined locations and with timing relation. The paradigm was presented to the Automatic Test Markup Language (ATML) Working Group in a paper titled the SSAI RAI Standard [1]. The paper was produced using results from a study on TPS rehost efforts. The focus was targeted where rehost practitioners spent their efforts. It was found that three areas became primary:

1. Reverse engineering path information to determine provisioning of legacy signals to and from UUT connection points
2. Reverse engineering and experimenting with code to adjust implied test needs, such as timing and conditioning.
3. Reengineering signal definitions to accommodate additional and also previously unnecessary information that is needed in the new platform. These items are a result of an incomplete signal model in ATLAS, and also the implementations of ATLAS that allows signal definition without requiring complete modifier sets.

The SRP addresses each of these observed areas where deficiencies found in legacy test program architectures and paradigms are found. A fundamental assertion was utilized in the development of the SRP. The assertion is that test programs must define what is needed and not how to accomplish what is needed. Traditional test programs all define how UUT related test actions must be accomplished. SRP based test programs will define what must be provided and sampled at the UUT. This assertion is applied based on the observation that many of the observed deficiencies in legacy rehost efforts were related to redeveloping the procedure and that the specification of what is needed will not require TPS procedure redevelopment. In this context, when “what” is mentioned, it can be thought of as data. When “how” is mentioned, it can be thought of as procedure. The contention here is that in scenarios that require maximized test program interoperability, the procedure would need to be rewritten to accommodate the new systems where only data needs to be reread.

First, test requirements must specify at which locations signals must occur and not how to get the signals to the desired location. The SRP consistently applies the need to specify what is needed and not define how to implement.

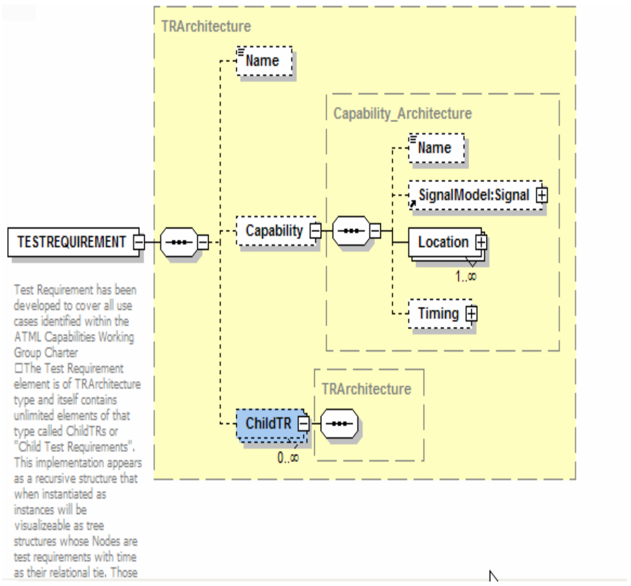
Second, the relationship between signals must be expressed explicitly. The definition of what is needed in time is required, not how to achieve it in time. Legacy test programs all have implied timing that impact their ability to interoperate on more than just the platforms for which they were developed.

Third, the paradigm defines what is needed at UUT pins in the form of signals. This is fundamentally the same as the Signal Paradigm, except that explicit timing and location definitions are associated along with one or more signals to define complete test requirements/ signal requirements. These requirements never need to be redeveloped because they are definitions of what is needed, not how to achieve what is needed. How to achieve what is needed will be different on every test system variant and will always require redevelopment during system development.

One general observation that has been made is that ATLAS developers and those that rehost ATLAS programs are forced to tinker with the code, sequence, timing, and signal definition in order to make it work. In ordinary procedural languages like C++, BASIC, etc., code is more deterministic. A developer knows what the results of his development tools are. But with ATLAS, developers must tweak, observe, adjust, delay, and iterate until functionality and repeatability are achieved. This can be observed in both new development using ATLAS and in program rehost and migration efforts where ATLAS is involved.

The Signal model found in IEEE-1641 appears to be the model of choice for the SRP as it is the most robust model in terms of explicitness. The IEEE-1641 signal model takes into account conditioning and other features that are imposed to create the final UUT signals that are not available in other models. For instance, consider a particular signal to be measured from a particular UUT pin with an instrument whose port has 50 ohm impedance. In ATLAS, that port information may be completely implied or not implied. There are many of these undeclared facets that can adjust signal quality that are not explicit in ATLAS. In IEEE-1641, a user is allowed to explicitly declare these types of information. Without these previously unmentioned facets being programmatically available and explicit, no signal could be deterministically and completely made transportable. It is this deficiency in content and explicitness that forces ATLAS users into the tweak, observe, adjust, delay, and iterate scenario.

Since the SRP always defines what is needed and not how it should be achieved, it should be clear that data definition is all that is required in Signal Requirement Definition. XML is a prolific modern software data definition technology that is a perfect match for defining signal requirements as required in the SRP. Figure 3 below shows a graphical representation of a schema developed to represent the SRP signal requirement intent.



Test Requirement has been developed to cover all use cases identified within the ATML Capabilities Working Group Charter
□The Test Requirement element is of TRArchitecture type and itself contains unlimited elements of that type called ChildTRs or "Child Test Requirements". This implementation appears as a recursive structure that when instantiated as instances will be visualizeable as tree structures whose Nodes are test requirements with time as their relational tie. Those

Figure 3

In the Figure, signal, location, and timing are unexpanded for brevity. For location, only a way to define a UUT location is required. For timing, a starting time from a fixed reference or another signal's timing along with accuracy and a lifetime are needed. Although any signal model that can be referenced in a schema can be utilized, it turns out that in IEEE-1641 a signal definition schema exists that is applicable to the robust signal model that IEEE-1641 also defines. Some adjustments to this XML based schema will bring it completely in line with the SRP concept while still meeting the legacy signal requirements paradigm.

The IEEE-1641 committee has agreed to consider these changes. Interestingly, the two paradigms attempt to accomplish the same thing with the same signal model. The signal abstraction uses procedures to define how things need to be accomplished at run time. The signal requirement abstraction uses data to declare what is needed and leaves the procedural decisions to the test platform at runtime.

In the SRP, test programs become simple diagnostic sequencing and control mechanisms that assert and acquire test requirements as they are needed and then fulfilled. Another advantage is that no domain specific language is required in the SRP. The processor in this scenario is utilized only for computation and sequence outside test requirement definition from the test program point of view. This is to say the procedure used in the test program is only for sequencing and diagnostic control issues that do not affect test asset control or instrument interoperability. The burden of allocation, connection and invocation, in SRP, falls on the test system. This means that procedural test system function is now controlled exclusively by the test system. In legacy test stations procedural functionalities are defined and integrated over and over again in test programs. In SRP, procedural functions are defined and integrated only once by a system integrator and with only a single cost associated. Once defined, a particular kind of procedure can be utilized again and again with little or no integration cost. The burden falls on the test system to extract SRP test requirement definitions and to provide the needed procedure to accomplish them using the test system assets. The test program developer no longer modifies procedural language that attempts to define how to achieve test requirements. The test system knows how to produce deterministically defined test requirements. The developer now deterministically specifies what is needed in the form of signal requirements. Figure 4 below depicts the vision for SRP TPS rehost efforts

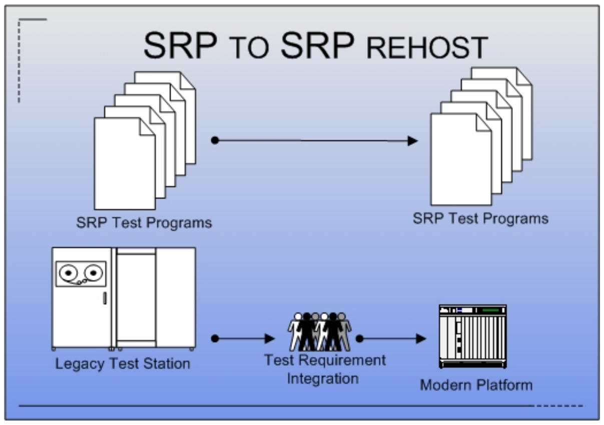


Figure 4.

Unfortunately, there are yet no commercially supported tools that utilize the SRP. However, work is being performed to incorporate the paradigm in the IEEE-1641 standard so that it embodies both the signal paradigm and the SRP. Once incorporated in the standard, the hope is that industry will realize the value in the technology and move forward to implement the SRP in the future.

RAI IMPLEMENTATION CHOICES

As with any system implementation, there are choices that must be made that affect performance, accuracy, aesthetics, and other system characteristics. With RAI, performance has come to the forefront as a key industry concern. Tasks that all the paradigms must accomplish such as instrument allocation, path selection and allocation, minimizing runtimes, and others are affected by the decisions. For instance, instrument allocation in the IVI standard is achieved against virtual instruments at compile time, while real instruments are allocated at runtime by matching virtual references in the configuration store to paired real instrument assignments. The IVI standard specifies the way this allocation is performed. In most ATLAS implementations, instrument allocation happens at compile time by comparing signal characteristics against available and comparable instrument functionality descriptions. Nothing in the ATLAS standard requires this implementation. As with ATLAS, the topic of allocation is not constrained by the SRP abstraction. There is no requirement to perform allocation at any particular level of execution or definition. It is left to the implementer to design this to his requirements in SRP. Despite significant industry focus on this issue, there is no reason SRP cannot be implemented in a manner that is at least as satisfactory as its predecessor abstractions.

AN RAI WORKING GROUP PERSPECTIVE.

Each of the three paradigms that we have found to address interchangeability fit within the RAI definition. In all three cases the abstraction that they embody was developed with interchangeability as a primary objective. In order to reduce costs, the DoD would like to embrace the most robust but still commercially viable RAI candidate standard possible. Since so many DoD related test programs exist on today's test systems it would seem natural to want something as closely aligned to ATLAS as possible. This means that if ATLAS is not recommended, that we need an easy migration path or at least some level of compatibility from ATLAS to anything new. This author's opinion is that nothing else will become widely utilized. The ideal situation would be to have something that supports ATLAS but provides a pathway to the future for improving the level of TPS and instrument interchangeability. The IEEE-1641 committee is currently in a revision cycle for the standard. Real world practioners are providing inputs that are being incorporated to improve the IEEE-1641 documents. The committee has also agreed to incorporate SRP concepts so that both the Signal and Signal Requirements paradigms are supported. This means we will shortly have a standard that; embraces a pathway to the future; and at the same time is supported in today's environment. The desire is for it to become a standard that provides a vehicle to maximized interchangeability using the ATLAS like signal abstraction for today and moving towards maximization with the SRP in the future. The RAI Working Group is supporting that direction and is supporting the IEEE-1641 committee to that end.

SUMMARY

The RAI Working Group is an industry supported technical forum that was formed to achieve consensus on what RAI is and how it can be standardized for use in DoD acquisition policy. The RAI working group has produced a consensus definition for RAI. Various industry-supported emerging abstractions, focusing on test program and instrument interchangeability, have been assessed. The instrument abstraction does not meet DoD needs when complex test applications require:

1. Users to step outside the generic instrument classes and utilize instrument specific functionality
2. Use of instruments for which generic instrument classes have not been developed

The Signal abstraction and Signal Requirements abstraction are found to be the best interchangeability candidates. These abstractions and paradigms comprehensively cover the complete universe of test requirements. Taken together the Signal and Signal Requirement abstractions will both soon be embodied in the IEEE-1641 standard. Acceptance of this standard will allow DoD to select existing products that achieve today’s level of interchangeability while providing a pathway to the future for maximized interchangeability. The culminations of these efforts allow the RAI Working Group to make a realistic recommendation for RAI, complete with a path for evolving to more comprehensive incarnations.