Architectural considerations for implementation of the ATML standards in an Open Systems Architecture Runtime environment (OSA-RTS) using a graphical environment

Chris Gorringe
Cassidian Test Engineering Services
Ferndown, UK
chris.gorringe@eads-ts.com

Anand Jain
National Instruments
Austin, TX
anand.jain@ni.com

***Abstract*— This paper identifies the architectural components necessary for implementing the MOD's Open System Architecture Runtime System (OSA-RTS) using the LabVIEW graphical environment for communication with the test equipment. The OSA-RTS provides common shared elements to provide ATE components to implement ATML solutions for translating ATML's Test Descriptions and Test Equipment Descriptions into run able test programs, utilizing a "Carrier Language" approach. This paper identifies the additional components needed to create runable test programs utilizing "Graphical Test Program" using a LabVIEW-based runtime environment. This architecture allows ATML's Test Descriptions to be directly translated into ruanable code, extending the supported platforms for rehosting standards based test descriptions. The architecture reuses and builds on the existing OSA-RTS elements including Test Signal Frameworks (TSFs), ATML Test Equipment Descriptions and IEEE Std 1641 signals, but test programs to be implemented using VIs integrated with a overarching test executive (TestStand). Finally the paper considers the use of IVI Switch to provide path level switching and identifies alterative Switch implementations that adhere to the IVI standard, and shows how these can be preconfigured unitizing the ATML hardware standards to identify the required routing. This architect design provides a design template for ATE designers and Test programs for rapidly implementing Standards based OSA-RTS on their existing ATE platforms.**

Keywords—Open System Archetecure (OSA; MOD Policy; ATML Implementation; Graphic Environments; TestStand; LabVIEW.

I. INTRODUCTION

Adoption of test standards, especially associated with architectural design to promote reuse, rehosting and adhere to governmental policy, has always been a hard sell. Any 'stick' approach, requiring change of practices is generally met with claims of extra risks and outcries of "we do not do it that way" resulting is inflated costs.

The first realization should be that existing solutions are not providing the through-life-cycle support required by the end user. Therefore what is typical today, is based not on the customers requirement, but what industry is prepared to provide. In some cases these offered solutions are designed to lock in the support solution over the life time of the product. This is not a technical challenge; it is driven solely from the market. It will only take a few players to offer open architecture solutions for all solutions to adopt the OSA handle.

What if anything is the justification for inflated costs for using standards? One area accepted as a legitimate cost driver is initial development and being first to use a new process. Being first is a cost driver; there are always new challenges, when developing new systems. Reusing existing processes, is always seen as a safer and less risky approach, even if they do not provide what is being asked for.

The MOD's Open Source Software for OSA run-time[1], was an initiative to address the natural resistance of adopting new processes, and to reduce the cost of initial ownership, by providing a series of open source software that converting standard based test information, into run able test programs. The initial suite of tools described in "An Open Source Software Framework for the Implementation of an Open Systems Architecture, Run-Time System"[2] details the process, COTS tools and open source software to take ATML test information and convert it into 'Carrier Language' based test programs. This project was available to MOD (& DoD) contractors, wanting to adopt an Open System Architecture to their test solutions and looking to jump start any development they needed for their specific test solutions.

There has been considerable initial interest in the OSA-RTS "open source code" and several companies and organizations are already investing in a standards based test solutions, to reduce their future support costs and reliance on proprietary systems.

This paper builds on the architecture model presented in that paper, to show how a similar architecture for graphical programming environments could be used to provide an OSA Runtime compatible solution using ATML, 1641 signals with TestStand, LabVIEW and NI Switch Executive.

II. The Architecture

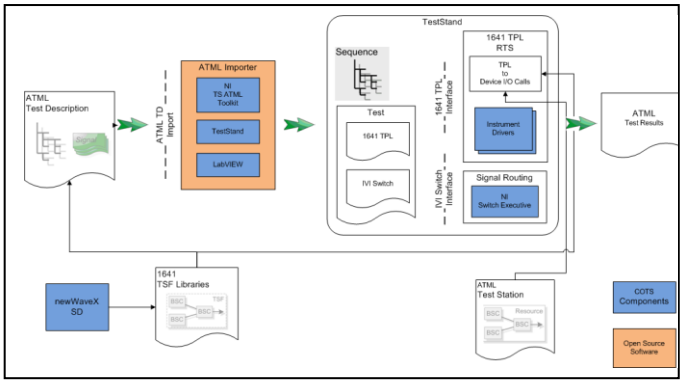

Fig. 1. OSA RTS Overview using TPL & LabVIEW Drivers

Figure 1 shows the architecture for migrating ATML test descriptions (IEEE Std 1671.1) utilizing IEEE Std 1641 signals, into run able code using TestStand sequences and implemented with LabVIEW drivers, and outputting test results (IEEE Std 1636.1)

During development the architecture supports the importing of ATML Test Descriptions (including test operations based on the IEEE Std 1641 Signal descriptions). The tool translates ATML Test Description into TestStand sequences and LabVIEW VIs, which contain the ATML Test Description Operations translated into IEEE Std 1641 TPL LabVIEW VI calls.

Each resource's available signal is defined within the ATML Test Station Description, using an IEEE Std 1641 TSF signals libraries. The actual runtime implementation exposes its functionality through a set of signal-based LabVIEW VIs whose behavior and property interface are defined by TSF (signal) libraries. TestStand sequences therefore call the TPL VIs that in turn invoke the associated signal-based VIs.

The benefits of this architecture is that many ATE implementers already use a system interface, therefore all that any future implementer would required is to ensure that their system interface is signal (rather than instrument) focus and provide a description of the interface as a suite of TSF libraries, and map these onto ATE resources through the use of the ATML Test Station Description, activities that are typically already done, but using a proprietary method.

Path level switching is used to map the resource input/output ports to the UUT pins and ports using the IVI Switch interface of NI Switch executive.

The use of Path Level switching makes the runtime development process easier by deferring until runtime the actual switch paths (and relays) to be used. The decision on which relay's to use, is typically made when identifying with resources to use, which has usually been a core design process in developing the interface adaptor. It should be noted that automatic allocation tools could still be used, because the complete diagnostic and test function path is known from the ATML Test Description. Although feasible this initial architecture relies on the user manually addressing any resource allocation or alternative path ambiguities.

For many users the architecture outlined is not that different from existing runtime implementations already deployed. This is a good thing, and one of the key objectives of the original OSA-RTS project, to allow existing implementations to be adapted to be conformant with the MOD's test policy. In many cases the peripheral test information, rather than being maintained by proprietary design documents or held within databases is now defined as one of the SCC20's test standards. It's not that the test information required is changing; it's just being formatted as a standard. As a risk assessment on any existing project changing the 'documentation' format has a low technical risk, and should have little or no impact on future programs.

III. The MoD's Open System Architecture Diagram

In order to ensure that all future test solutions procured by the UK MOD's adhere to Open Standard[4], there is now a UK Defence Standard "Def Stan 66-31 – Part 8 – Requirements for Automatic Test Systems utilizing an Open System Architecture"[3], that describes what is needed for any future test system to be considered using open standards, and therefore not locking the MOD into proprietary test solutions.

The core to this Def Stan is the use of test standards, so that test information is available to the MOD for future support or rehost or upgrades. The implementation or tools to use these standards is still the preserve of the test solution provider, but what information is required and the format by which it should be exchanged is described by the Def Stan.

A similar diagram also exists for the DoD ATS Framework group, such that the OSA-RTS is seen as a sharable component for any future joint platform between defense coalition partners on reducing test support solutions.

Figure 2 shows the standards covered by the Def Stan 66-31,

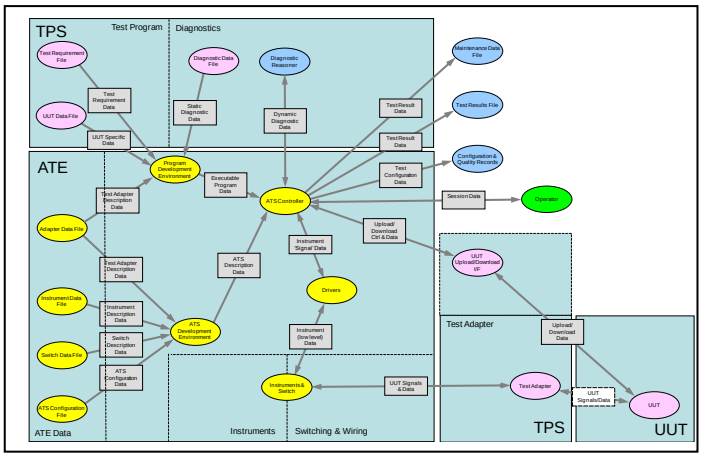

Fig. 2. Def Stan 66-31 Open System Archietcure Diagram

The OSA Runtime System [1] typically only covers a subset of these components and the diagram can be simplified

to just those components that would have an impact on the runtime. Including:

- ATML Test Description (IEEE Std. 1671.1)
- TSF Libraries (IEEE Std 1641)
- (ATML) Test Results (IEEE Std 1636.1)
- ATML Test Station Description (IEEE Std 1671.6)
- ATML Test Adaptor Description (IEEE Std 1671.5)
- ATML Instrument Description (IEEE Std 1671.2)
- ATML UUT Description (IEEE Std 1671.3)

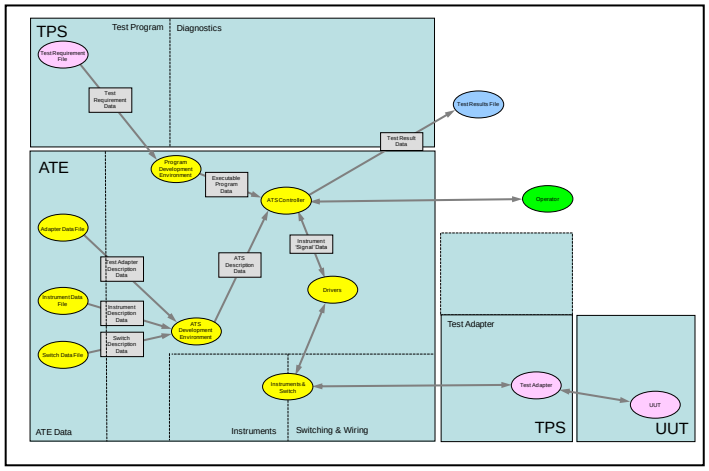

Fig. 3. Simplified Def Stan 66-31 Open System Archecture Diagram for runtime system

By mapping the solution components onto this simplified OSA RTS diagrams we can see how the architecture components fulfill an OSA solution.

A. *Test Requirement Data*

The is accomplished through the use of ATML Test Descriptions (IEEE Std 1671) and Signal Descriptions (IEEE Std 1641) in the form of TSFs.

To simplify the runtime process the initial solution uses the same set of TSF libraries for both ATML Test Description operations and ATML Test Station Capability, allowing a one-to-one mapping to be performed. Having such a limitation does not stop the solution being OSA compliant it just restricts the test descriptions which could be rehosted. However by ensuring the use of standards this could be overcome in future implementations without affecting existing programs.

B. *Program Development Environment*

The ATML Test Description translator (ATML importer), is the key front end tool. It uses NI TestStand ATML toolkit, TestStand and LabVIEW to create the test program files, because the import process is directly controlled by the ATML Test Descriptions. Once the users customized any MMI and environment (for all test programs) any changes would be at the import level and not changes to the generated test programs files. The ATML importer also has the capability to only apply changes made to the ATML Test Description file to the generated test program files.

C. *Executable Program Data*

The files produced are TestStand sequence files, LabVIEW VIs, that contain within them IEEE Std 1641 TPL statements, matched to the ATML Test Description operations, and IVI Switch path level statements (for an IVI driver) (See E Signal Routing).

The Executable files represent an interpretable file that can be executed by the test executive, containing the program flow, signals and signal paths. A districting between this design and the “Carrier language” OSA-RTS [2], is that the interfacing does not use the signal Interface Definition, and therefore does not provide a runtime signal interface available to other development environments capable to read type libraries, although it does use an internal signal interface.

D. *Test Executive*

The Test executive is the core component, incorporating the TestStand sequencer, LabVIEW VIs that map IEEE Std 1641 TPL onto LabVIEW signal VIs that contains the signal implementation onto instrument VI’s. These mappings could be using the IVI drivers or native LabVIEW instrument drivers. The architecture supports both.

E. *Signal Routing*

Signal Routing uses the IVI switch interface as per the Carrier language OSA-RTS, but using the NI Switch Executive to implement the path level switching.

F. *Test Results File*

ATML Test Results have long been generated by NI TestStand test executive, providing a robust interchange format with viewing and analysis tools into other systems.

G. *Test Station Description*

The ATML Test Station Description brings together the relevant ATML capabilities that make up the target automatic test system. It includes the various instrument descriptions and station wiring, and is used to help identify which resource supports with TSF Signal.

IV. Conclusion

The graphical OSA-RTS architecture is conformant with the Def Stan 66-31 Architecture Diagram as shown in the figure 4 mapping, but uses different COTS components and driver solutions, to the Carrier Language, Open source solution described in reference[2].

One outcome of this architecture is that is enforces strict adherence to ATML 1641.1 Test Description as the UUT Test requirement, since there are no other source files or carrier language to maintain or that can be edited. Additionally the use of the same TSF libraries for both UUT signals and ATE capabilities brings the requirement for a standardized set of TSF Libraries a major step closer. Having standard TSF libraries to support such an architecture will only help reduce

the resistance and envisage start up costs of adopting such a Open Architecture solution.

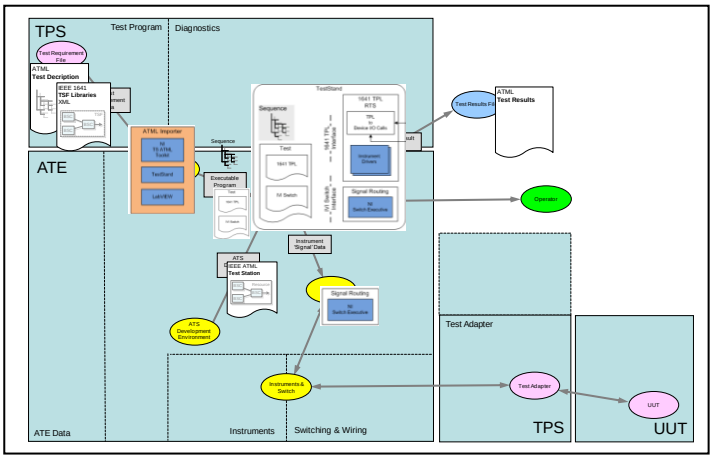

Fig. 4. Mapping onto Open System Archecture Diagram

So who wants to join in and use it?

REFERENCES

[1] Open Source Software for OSA Runtime. Online reference 2012

[2] M Cornish and M Brown, "An Open Source Software Framework for the Implementation of an Open Systems Architecture, Run-Time System" IEEE AUTOTESTCON 2012.

[3] Def Stan 66-31 Part 8 – Requirements for Automatic Test Systems utilising an Open System Architecture, MoD Defence Standards, August 2011

[4] M Brown, K Ellis, A Hulme, State of IEEE 1641 Standard, Applications and UK MOD Policy, AUTOTESTCON 2008