Automation Hooks Architecture Trade Study for Flexible Test Orchestration

Chatwin A. Lansdowne, John R. Maclean, *Members, IEEE*
Frank J. Graffagnino, Patrick A. McCartney

***Abstract*—We describe the conclusions of a technology and communities survey supported by concurrent and follow-on proof-of-concept prototyping to evaluate feasibility of defining a durable, versatile, reliable, visible software interface to support strategic modularization of test software development. The objective is that test sets and support software with diverse origins, ages, and abilities can be reliably integrated into test configurations that assemble and tear down and reassemble with scalable complexity in order to conduct both parametric tests and monitored trial runs. The resulting approach is based on integration of three recognized technologies that are currently gaining acceptance within the test industry and when combined provide a simple, open and scalable test orchestration architecture that addresses the objectives of the Automation Hooks task. The technologies are automated discovery using multicast DNS Zero Configuration Networking (zeroconf), commanding and data retrieval using resource-oriented Restful Web Services, and XML data transfer formats based on Automatic Test Markup Language (ATML). This open-source standards-based approach provides direct integration with existing commercial off-the-shelf (COTS) analysis software tools.**

***Index Terms*— Software standards, Test equipment, Test facilities, Testing, Software management, Software reusability**

I. Introduction

NASA's Constellation Program identified an opportunity to reduce out-year operating costs for system and sub-system integration test operations through automation-assisted test choreography and data orchestration. There are complimentary opportunities to improve scientific research and engineering development workflows.

Essentially, the opportunity is that even for run-once and investigative testing, COTS and even custom hardware is configured and monitored by users through keyboard-and-mouse software packages. If data from these heterogeneous modules could be harvested through a robust, open standard based infrastructure, the data products could be formed more quickly, accurately, and thoroughly, and results correlated more powerfully—by comparison with, say, having users type data from screens into spreadsheets or collect and transfer data files in an ad-hoc fashion.

The application scenarios anticipated are not high volume or highly repetitive. Automating development of the test procedures themselves from requirements is not a significant area of interest. Much of the potential for saving is related to discovery of module data requirements and aggregation of test data in integrated scenarios containing a changing assortment of highly complex and coupled modules.

There are many technical challenges to address, but first one must confront the organizational challenge in that the software modules available at a system integration test, for example, are of diverse origins and developed on heterogeneous platforms.

Having accomplished the aggregation of coincident observations, one can further imagine storing and restoring the configuration of the test bed using read/write interfaces, and ultimately it could be possible to repeat test sequences and overlay data.

Quiescent, continuous, and event-driven test cycles are anticipated. Scripted flows are presumed built on less structured fault-isolation or experimentation test flows.

The concept of operations imposes some constraints required to enable data correlation. These include time synchronization mechanisms and resources, data indexing, labeling data with metadata, and encouraging the use of widely understood self-describing data formats.

II. Criteria and Field of Choices

A concept of operations was proposed, and then distilled down to a set of "guiding principles" which could be used for evaluating different approaches. These principles included:

Non-proprietary, with multiple vendors. A proprietary or single-vendor interface could not achieve universal penetration into varied developments and could present a single-point-of-failure risk to the Program.

Widespread usage, with active user communities. Our intention was not to reinvent the interface and associated toolsets, but rather to find and adopt (adapt) already widely supported technologies.

Supported in the Test industry. Interfaces with existing support in NASA, DoD, and consumer communities and test COTS products were given affirmative weight.

Multiple sources of ready development tools. Software

Manuscript received June 1, 2010. This work was performed in NASA Johnson Space Center's Avionics Systems Division in support of the Constellation Program.

C. A. Lansdowne is with the National Aeronautics and Space Administration, Houston, TX 77058 USA (phone: 281-483-1265; fax: 281-483-6297; e-mail: chatwin.lansdowne@ nasa.gov).

J. R. Maclean is with METECS, Houston, TX 77058 USA (phone 281-483-3265; e-mail: john.r.maclean@nasa.gov).

978-1-4244-7961-0/10/$26.00 ©2010 IEEE

interfaces supported by a family of open source tools provide rapid deployment.

Language and OS independent. Interfaces that are tied to specific operating systems or development environments only solve part of the problem, and are vulnerable to accelerated obsolescence.

Having described what we were seeking, we surveyed test communities at NASA, DoD, and in industry, and also considered plug-and-play consumer interfaces. We considered that our software elements could use simulation interfaces, or instrumentation interfaces, or web services interfaces. Fundamentally the difference between requirements for a test software interface and a simulation software interface is that the modules do not need to interact.

III. Summary of Study Observations

A. Existing End-to-End Infrastructures

Several existing end-to-end simulation and test infrastructures were investigated in an attempt to find an out-of-the box capability that could be used to meet the trade study criteria.

High Level Architecture (HLA) has been used in the Constellation program as an architecture for distributed dynamics simulations. It was examined briefly but it was quickly decided that the overhead associated with its run time infrastructure and simulation federate organization made it unattractive as a test orchestration infrastructure.

The Test and Training Enabling Architecture (TENA) is a DoD initiative aimed at distributed simulation and test applications. It is geared to supporting test ranges and facilities. TENA seemed to require middleware that appeared to be single-source. In addition, it is based on CORBA, an object-based messaging protocol that has been declining in popularity because of its complexity and historical difficulty penetrating firewalls. Interest in the wider community has shifted from CORBA and its competitor DCOM to Web Services which are discussed later.

B. Established Test Software Interfaces

A promising early candidate was the LXI interface, and ultimately we adopted several features of this interface. The interface was discarded because tools for development of LXI hosts were not readily available.

Investigation of the DoD Automatic Test Systems (ATS) Open Systems approach lead to interest in the Automatic Test Markup Language as a data format. Many of the approaches of the ATS Open Systems approach were compatible with our "guiding principles". Virtual Instrument Software Architecture (VISA) and Interchangeable Virtual Instruments (IVI) technologies were determined to be too low-level for our goal and available drivers appeared to be limited to the Windows OS. In addition, these technologies did not appear to be widely used outside the Automatic Test Equipment industry.

NASA's Constellation program was also developing an interface for avionics test orchestration, Software and Avionics Test Orchestration Command and Messaging (SATOCM). We did exchange observations with this group, and although there remain differences in emphasis both teams believed it would be possible to achieve convergence. This interface was designed to simplify test script-writing using Python, and its current incarnation was rejected by our study because it violates many of our guiding principles.

C. Discovery Protocols

Universal Plug and Play (UPnP) was evaluated against Zeroconf. Both were strong candidates, but we perceive the framework provided by Zeroconf to be more applicable to our technical challenge of discovery, and Zeroconf has existing heritage in the test community through its use in LXI [1].

D. Messaging Protocols

Several message oriented protocols and middleware APIs at several different levels of complexity were considered. Some, like Java Message Service (JMS) were not language neutral. Some like Advanced Message Queuing Protocol (AMQP) introduced complexity by solving problems we did not have. The Simple Object Access Protocol (SOAP) web services protocol was chosen for initial prototyping because it satisfied our evaluation criteria and fit well with ATML which was also of interest. There is a wide variety of tools and implementations available including many open-source packages. It is also widely used and accepted in many industries.

A functional prototype was implemented using SOAP. Many parts of the SOAP implementation, however, were found to be complex in the face of limited prototyping resources. For example, Web Services Description Language (WSDL) files were found to be complex to create and maintain. Different implementations of SOAP were found to be incompatible without detailed attention to configurations and options. No insurmountable problems were encountered but eventually a decision was made to prototype an alternative resource oriented or Representational State Transfer (RESTful) [2], [3] style of web services. The level of simplification, elegance and increased ease of implementation was so striking that ultimately when faced with building a prototype using limited resources we opted for the RESTful approach. This choice affected not just the messaging protocol but the overall architecture and division of responsibilities between test orchestration software and individual test set interfaces.

Many of RESTful features such as Uniform Interface, stateless server restrictions, and cacheable responses contributed to robustness and enhanced visibility of the test protocol. Also, the perspective on commanding test equipment changed from remote-procedure-call (RPC) based to resource based which was found to result in gains in elegance and simplicity.

E. Command Sets

Test execution interfaces have a long history of using verb-based command sets, including HP BASIC, ATLAS, and SCPI. NASA's SATOCM command set was intended to simplify script-writing, and initially we planned to implement a subset of SATOCM commands.

The RESTful style architecture primarily uses a small subset of standard HTTP commands such as GET, PUT, POST, and DELETE directly. The richness of the interface is then

captured as resources that are manipulated using these standard HTTP commands. This approach replaces the requirement to create a traditional RPC-based set of commands with the requirement to design appropriate resources to represent required test concepts. In prototyping, the resource-based approach was found to result in a simpler and more transparent infrastructure.

The command and error message sets already provided by HTTP understood by a large collection of off-the-shelf software. The command set is compact and powerful, and the error message set is rich. Security and data compression solutions are innate.

F. Data Interface Protocols

We evaluated the architecture of having software modules write directly to a designated database interface without an intermediary. Scalability and robustness were identified obstacles. To make a successful interface, a completed software module must be able to create its own tables and write data to them without further changes to the platform to accommodate different database vendors or other changes in database technology. JDBC was entertained as meeting this objective, but limits the usability of the module by requiring each software module to interface with Java. An ODBC driver approach was evaluated but required specialized software to be installed and maintained on each client. An ODBCbridge driver approach eliminates the client software issue, but introduces an issue with proprietary software and a sole-source provider. It was determined that this type of SQL-oriented middleware merely transfers the maintenance problem to another vendor who must then be required.

The solution that worked best in prototyping and met the goals of the study was to use the resource oriented interface to serve data. An unexpected side benefit was that data resources could be accessed by web-ready off-the-shelf software. For example, prototypes built on modular open source software have demonstrated that this interface is already natively accessible to web browsers and to Excel.

Data log requests are submitted by the orchestrator and each test set module makes locally buffered data accessible through a resource interface for that data log request. This approach allows data to be logged with arbitrary resolution and alignment occurs after the observations are aggregated.

G. Data Formats

Software written by hardware engineers often will write data using comma separated value (CSV) or tab separated value (TSV) formats. These formats are easy to generate, widely supported by tools, and are decades old. There are, however, many format variations including how commas are handled within a data file. This can be particularly troublesome for countries that use commas as the decimal separator. In addition there are no recommendable approaches for incorporating meta-data. There are also some operating system differences.

Binary formats are very system-dependent, although they can be supplemented with descriptive XML metadata files.

The SQL statement format was also considered, but the availability of XML-enabled databases diminishes the appeal of this option. It was further identified that different database vendors have different interpretations of the SQL standard.

XML allows sufficient metadata to be included so that database tables can be automatically created, standardizes the date-time format, and allows further information like theory of operation (help-text) for a parameter to be captured. ATML [4] provides an XML language that standardizes information exchange for many kinds of data and meta-data we are interested in capturing, and is also becoming represented in test industry products. An alternative schema, NASA Exploration Information Ontology Model (NExIOM), was discarded only because it has a limited following. The authors hope that NASA can participate in the further development of ATML.

IV. Conclusion

Combining RESTful principles with Zeroconf and ATML formed a powerful, versatile, rugged interface that met all of the study criteria. The combination was found to provide a simple, elegant, and easy to use infrastructure for test orchestration. Prototypes have already demonstrated connectivity with LabVIEW, NASA's Trick Simulation Development Environment, and the Engineering DOUG Graphics for Exploration (EDGE) software used for 3D graphics rendering in Constellation training and test facilities. Prototypes have been hosted in various distributions of the Linux operating system and in Windows XP and Vista. Distributed and co-hosted topologies have been demonstrated, and multiple copies of modules are distinguishable. The interface has been demonstrated with both simulations and hardware and has been used to orchestrate a distributed Orion abort-to-orbit test scenario using JSCs Avionics Integration Enivironment (AIE) facility and the Reconfigurable Cockpit Simulation Facility. It is being integrated with test hardware in the Kedallion avionics facility and the Electronics Systems Test Lab (ESTL) at Johnson Space Center. We currently rate this interface as Technology Readiness Level 4.

Acknowledgment

This endeavor required focusing many kinds of nonintersecting experts on a multi-faceted problem. We would like to thank Thomas Brain, Tom Smith and Chris Winton for their contributions to the software prototypes. Joan Zucha and Adam Schlesinger were sources of guidance and evaluation.

References

[1] N. Barendt, *LXI 1.2 Improves Discovery and Identification*, LXI ConneXion, February 2008, pp. 12-16.

[2] R. T. Fielding, *Architectural Styles and the Design of Network-Based Software Architectures*, Doctoral dissertation (2000), University of California, Irvine.

[3] L. Richardson, S. Ruby, *Restful Web Services*, O'Reilly Media, Inc., 2007.

[4] http://grouper.ieee.org/groups/scc20/tii/