XAL STATUS REPORT - FALL 2009*

Thomas Pelaia II, Christopher Allen, Andrei Shishlo, ORNL, Oak Ridge, TN 37831 U.S.A.

Abstract

XAL is a collection of frameworks for building accelerator applications at the Spallation Neutron Source. We discuss progress in easing the adoption of XAL for use at other facilities by providing improved documentation, eliminating several third party jars and deprecating and removing obsolete code. New XAL features will be introduced as well as recent application additions and enhancements.

INTRODUCTION

XAL[1] is the software infrastructure for accelerator physics applications and services at the Spallation Neutron Source (SNS) in Oak Ridge. XAL contains an accelerator device hierarchy, an application framework [2], a control system adaptor, an accelerator physics model [3] and general application tools.

The number of XAL applications has grown from a handful of applications for commissioning in 2002 to over five dozen applications, today.

While we continue to build new applications and introduce new technology, recent efforts have focused on code maintenance and easing adoption. Realization of this last goal has made XAL more feasible for use at other facilities.

This paper reviews XAL Status highlights since the most recent status reports [4,5].

XAL ADOPTION

As more developers both inside SNS and at other facilities adopt XAL, we are working to improve their experience. The biggest roadblock for adoption at other facilities has been the difficulty in obtaining third party packages on which XAL depends, so we have made a significant effort to address this issue. Furthermore, we have cleaned up code, fixed and added documentation, reduced initial configuration and enhanced communication.

Third Party Packages

Do to licensing restrictions, we don't have permission to redistribute all requisite third party jars with XAL which is distributed under the friendlier BSD license. Furthermore, the plotting packages were commercial. Since this had been a major obstacle for adoption of XAL at other facilities, we made efforts to address it.

We eliminated the dependence on all commercial packages by removing some obsolete applications, and updating others to use the XAL plotting package [6]. Likewise, we have removed obsolete code which had dependence on other third party packages. We began with **twenty-nine** third party jars in July 2008, and have reduced the count to **eleven**. We also obtained explicit permission from some vendors to redistribute their packages with XAL. Finally, we clearly documented the version and source of each third party jar file used in XAL.

This effort not only makes it easier for others to adopt XAL, but generally makes it easier for us to maintain XAL. For example, we determined that one of the third party jar files had caused XAL applications to crash upon startup under Java 6 on Linux. That jar file has since been removed.

Code Maintenance

XAL has been in use and under development for over seven years. Over this time, many applications, packages and classes have become obsolete, and in some cases the original developers of these constructs have left the project. Keeping these obsolete items in the project carries overhead such as dependence on third party packages that aren't used elsewhere, dependence on other obsolete classes and packages within XAL, maintenance to keep them in compliance with the current build and overall debris. XAL is now leaner due to our efforts to remove obsolete constructs.

We have removed eleven obsolete applications which has given us flexibility to remove other obsolete packages. UML files had cluttered the project yet were never maintained or generally adopted, so they have been removed. XML DOM traversal classes have been removed since their use was limited and had become obsolete and they required a third party package which was incompatible with Java 6 on Linux. Many other obsolete classes and packages have been removed.

We have an ongoing effort to reduce and cleanup code.

Documentation and Communication

When one is investigating or just getting started with a new development platform, it is desirable to read documentation about the technology and be kept in the loop about the future directions of the platform. While Java has a convenient mechanism (Java Doc) for generating API documentation from specially marked comments in the code, the developer must maintain consistency with the API and avoid introducing text which corrupts the generated HTML documentation. The former is easily identifiable from Java Doc warnings, but the latter can go unnoticed until one attempts to read the documentation and finds that pages are broken or entire sections of documentation are missing.

XAL had over 300 Java Doc warnings, and we eliminated all of them. That addressed the consistency issue at least in its simplest form. Furthermore, we fixed numerous errors resulting in corrupted HTML. Additionally, we publish and maintain the Java Doc on our XAL website. We also generate an indexed documentation set from the Java Doc which is then

*ORNL/SNS is managed by UT-Battelle, LLC, for the U.S. Department of Energy under contract DE-AC05-00OR22725

published via a RSS feed. Developers can subscribe to this feed to pull into their IDE the latest documentation which supports rapid API lookup.

The official XAL website, http://www.ornl.gov/~t6p/Main/XAL.html, contains the documentation discussed above as well as a news feed to keep up with the latest XAL news, an introduction to XAL, a quick start guide and links to resources including tutorials. The quick start guide has been valuable in rapidly getting XAL installed, built and running for new developers.

CORE MODIFICATIONS

Key-Value Table Model

Creating table models from scratch in Java can be cumbersome. It is common to display a table representing an array of objects of a common class where each row is mapped to an object in the array and each column corresponds to a getter method to be evaluated on the objects. We have developed a mechanism to generate table models dynamically for this common pattern.

KeyValueTableModel is a table model class we created to represent a table model from a list of objects and a (variable argument) array of named properties. Each object in the supplied list represents a row in the table model. The array of named properties are Strings each of which names a path of getter methods to be called on each object and mapped to a table column. A path is represented as a string of keys separated by dots and corresponds to a calling chain on an object. Each key corresponds to the name of a method with "get" prepended if necessary to make a match. For example, suppose a hypothetical class called "Bounds" has a method to get the size, *size()*, as an object which in turn has methods to get the width and height called *getWidth()* and *getHeight()*. The property, "size.width", applied to an object would correspond to a call on that object like: *object.size().getWidth()*. In this case, the table model column title corresponding to the property is automatically generated to be "Size Width", but can be specified manually if desired.

Table models matching the common pattern described can be generated with as little as one line of code rather than a page of code. Furthermore, the subclass, *KeyValueFilteredTableModel*, supports dynamic filtering of displayed objects based on matches of property values against entered text in a text component.

Database Configuration

Many of our applications and services require access to the SNS global database. The mechanism to support a database connection was a connection dictionary properties file which supplied the URL of the database server, the user name and password. However, recent changes to our database requires support for multiple accounts. Furthermore, it would be convenient for development to offer the option to specify a development or production database server.

Significant changes have been made to the XAL database infrastructure to support multiple accounts and multiple servers. The connection dictionary properties file has been replaced with a database configuration XML file. A connection dictionary can be generated from the database configuration by requesting a server and an account each of which have defaults.

This new database configuration architecture allows applications more flexibility while meeting the requirements for multiple accounts.

Software Device Types

Devices in the accelerator hierarchy each have an associated device type (e.g. Horizontal Corrector) which indicates the role of the device and the hardware type. However, a device of a given hardware type may have different software and hence a different control interface than other devices of the same hardware type. For this reason, we have added a property to the accelerator node (and correspondingly to the database) which indicates the software type of a device. Currently this property is being used for wire scanners and it allows new wire scanner software to be tested on a subset of devices before being applied to all of them.

NEW AND MODIFIED APPLICATIONS

Several applications have been modified with new features and to reduce configuration and maintenance.

Launcher

The Launcher is an application which displays and launches applications and scripts on demand. The previous launcher required the developer to maintain a bash script for each executable (application, jython script or jruby script) to be launched and to enter the bash script path and description of the executable to be launched. Furthermore, as the number of applications and scripts had grown, it became difficult to navigate through the long list of applications and scripts to find the desired one. The Launcher has been significantly modified to address these issues.

Since all XAL application products are built to the same directory and the scripts reside in just a few directories, the new Launcher simply records these directories as search paths. By default the Launcher determines the application search path based on the current location of itself thus requiring zero configuration for launching applications on the current host. Furthermore, the Launcher looks into each application's jar file to locate the "About.properties" file that every XAL application has and parses it to get the description and full application name. For scripts, it simply parses the header and attempt to extract the description from the comments.

All applications and scripts are displayed in a table showing the executable name, type, description and file path. A filter text box filters the displayed executables against each of these fields allowing for quick access to the desired application or script.

Custom launch commands may be applied to applications and scripts based on file name pattern matching. For example, you could specify different launch command templates for jar, jython and jruby files. Just as with the original version, you can specify a list of

hosts on which to run the executables in a round robin pattern for load balancing.

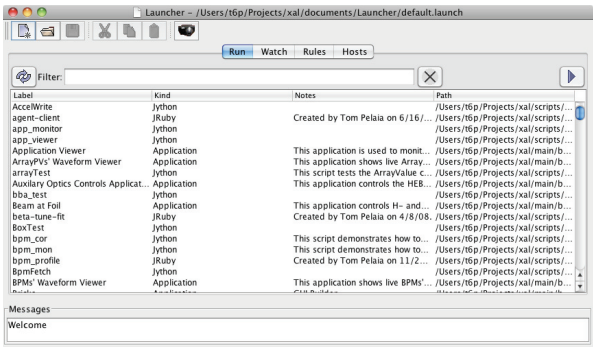

Figure 1: Launcher.

Virtual Accelerator

The Virtual Accelerator [7] has been instrumental in offline testing of our applications. It simulates a live machine by running a local channel access server and updating diagnostic values in response to magnet changes using the online model.

We have modified the Virtual Accelerator to use the new Java Channel Access Server which was recently made available in Channel Access Java by Cosylab (http://caj.cosylab.com/index.html). The virtual accelerator now runs with zero configuration since it no longer depends on an external channel access server.

XYZ Correlator

The XYZ Correlator application monitors and displays channel access events correlated by time stamp. This application has undergone significant modification. Plotting is now done using the XAL plotting package. The interface for specifying process variables to monitor is more efficient and consistent with other XAL applications. While it can display a buffer plot of correlations for two or three process variables, there is no limit on the number of process variables that can be monitored. A report of correlations over all specified process variables can be exported.

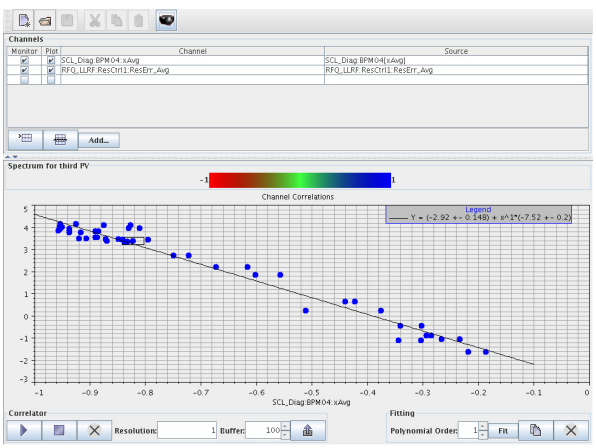

Figure 2: XYZ Correlator.

FUTURE PLANS

We continue to work on core enhancements, application development and bug fixes along with cleaning up code and documentation. The Wire Scanner and Wire Analysis applications are being merged into a single application that is being written from scratch to improve the overall user experience and provide better fits. A distributed agent system is currently under development. We are migrating XAL to Java 6. The Operations group is developing a sequencer application. Channel access for accelerator nodes will support batch channel operations offering the opportunity to dramatically improve performance. We will design a beam simulator that will operate on field maps.

REFERENCES

[1] John Galambos et. al., "SNS Application Programming Environment", EPAC 2002, Paris, France, June 2002; http://www.jacow.org/e02/PAPERS/THPLE021.pdf.

[2] Thomas Pelaia II, XAL Application Framework and Bricks GUI Builder, ICALEPCS 2007, Knoxville, TN, October 2007; http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/TPPA09.PDF

[3] C.K. Allen et. al., "A Novel Online Simulator For Applications Requiring a Model Reference", ICALEPCS 2003, Gyeongju, Korea, 2003; http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/WE116.PDF

[4] Thomas Pelaia et. al., "XAL Status", ICALEPCS 2007, Knoxville, TN, October 2007; http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/MOPB02.PDF.

[5] Thomas Pelaia et. al., "XAL Status Report Spring 2009", EPICS Collaboration Meeting, Vancouver, Canada, May 2009; http://isacwserv.triumf.ca/epics09html/FR1/XALStatus Report 2009.pdf.

[6] A. Shishlo et. al., "Java Swing-Based Plotting Package Residing Within XAL", ICALEPCS 2007, Knoxville, TN, October 2007; http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/TPPA08.PDF.

[7] A. Shishlo et. al., "The EPICS Based Virtual Accelerator – Concept and Implementation", Proceedings of the 2003 Particle Accelerator Conference, Portland, OR, May 2003; http://accelconf.web.cern.ch/AccelConf/p03/PAPERS/WPPE017.PDF.