THE XAL INFRASTRUCTURE FOR HIGH LEVEL CONTROL ROOM APPLICATIONS

A. Shishlo#, C. K. Allen, J. Galambos, T. Pelaia , ORNL, Oak Ridge, TN 37831, U.S.A.
C. P. Chu, SLAC, Menlo Park CA.

Abstract

XAL is a Java programming framework for building high-level control applications related to accelerator physics. The structure, details of implementation, and interaction between components, auxiliary XAL packages, and the latest modifications are discussed. A general overview of XAL applications created for the SNS project is presented.

INTRODUCTION

The development of XAL [1] was started in 2001 at the SNS project as a framework for high level accelerator physics applications. The Java programming language was chosen because it addresses the need for a GUI interface, database services, plotting, and numerical simulations. When XAL development first began there was a lack of free mathematical and plotting packages, but the situation has since improved. EPICS has been chosen as a communication protocol. Today the XAL framework consists of the following parts:

- A hardware representation of the machine for connectivity and control.
- A beam simulation model termed the "online model" for model reference and comparison to the hardware operation.
- An application framework to provide a common “look and feel” and functionality for all XAL applications.
- Services that run continuously in the background (24/7), and which can communicate with several XAL applications simultaneously.
- A set of auxiliary mathematics, graphics, and plotting packages.
- The channel access communication library.

In this paper we present descriptions of these parts of XAL and an overview of applications implemented on the base of this framework for the SNS project.

ACCELERATOR MODEL

An accelerator model represents a structural view of an accelerator. According to this model the accelerator consists of ordered accelerator sequences which usually represent accelerator beam lines, and they can have other ordered sub-sequences or nodes corresponding to physical devices. An instance of such a structure is shown on Fig. 1. The lowest level of the accelerator model hierarchy is represented by such components as magnets, BPMs, wire scanners, RF gaps, position markers etc. Usually accelerator nodes correspond to real physical devices, but it is not necessarily a one-to-one mapping.. For instance, at SNS there are single devices consisting of a quadrupole + dipole windings + BPM strip-lines. We consider these functionalities as three separate accelerator nodes (quad + dipole corrector + BPM), all at the same position.

XAL uses an XML file called an “optics_source” as a natural way to initialize this accelerator hierarchy. This XML file includes all information about sequences, components, positions, parameters, and necessary EPICS’s PV names for device signals.

There are two ways to prepare such files. First, it can be done manually from scratch or by modification of an existing file if you are interested in only a relatively small accelerator model for testing XAL features. Second, you can prepare an application that will generate the file for you by using a relational database. Of course, this application will be specific for each accelerator, because accelerator database structures are usually different.

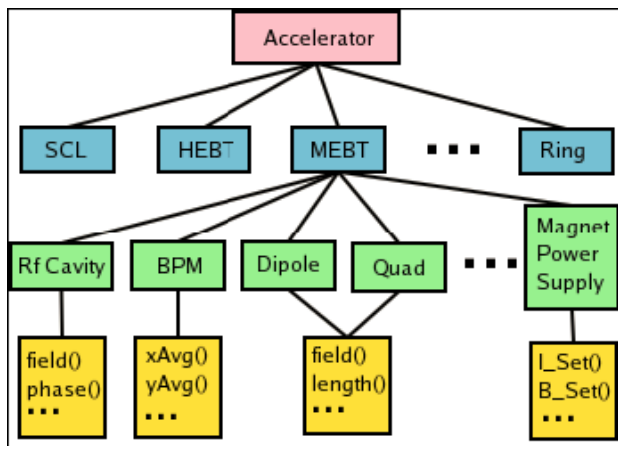

Figure 1: An example of the XAL accelerator model structure.

In the beginning of XAL development, the optics XML file was the only source for the accelerator model initialization, but later several new XML files were added to provide the model with the necessary information. First, there was an XML file with hardware node status information. This is a small file describing availability of certain diagnostics nodes, because they frequently go from the“online” to the “offline” state, and the model should know about a validity of the diagnostic signals. The second new XML file maps an accelerator node type with a particular implementation of this type in the model. This file was introduced to generalize XAL and to use it for different accelerators where the similar devices (i.e. BPM in SNS or J-PARC) can have different functionality. The third one includes information about signals from

#shishlo@ornl.gov

systems like a timing system that does not belong to any particular beam line. Finally, there is a XML file with the online model parameters. Today the combination of all these XML files is used to initialize the XAL accelerator model.

After initialization, the model is ready for usage. Examples of common tasks for the model and nodes are:

- Creating a new combo-sequence from existing sequences.
- Getting a position of a node or a sequence inside a parent sequence.
- Selecting nodes of a certain type and properties from a sequence.
- Getting a list of possible EPICS signals from a node.
- Getting default values of parameters of a particular node.

None of the actions listed above need live EPICS connections and therefore they do not require a real machine or virtual accelerator. There is another group of actions which require live EPICS communications. Some of these actions include getting or setting live hardware parameters such as magnetic fields or RF cavity phase and amplitude or reading diagnostic data such as BPM signals. This group provides live interfacing with the accelerator. The details of such EPICS connections and communications are hidden from the user by the simple interface of the model.

The XAL accelerator model is a very useful and convenient control system tool, but by itself it is has nothing to do with accelerator physics. To perform meaningful operations with the accelerator or beam lines we need a physical model. We call this model the “online model”.

XAL ONLINE MODEL

The XAL online accelerator model [2] performs on-the fly calculations of beam parameters based on machine settings. These settings can be extracted from a live accelerator, from design values, from a combination of these two sources, or they can be modified by the user.

The three main components of the model are an accelerator lattice which is constructed from the accelerator nodes, a probe which describes the beam and how it is to be modeled, and a set of algorithms for probe tracking through different elements of the lattice. The online model implements the Element-Algorithm-Probe design pattern introduced by Malitsky and Talman [3]. This design strategy separates the machine representation from the beam model and the dynamics calculations.

A lattice can be generated for any sequence of the accelerator model described in the previous section. In the transformation to the online model lattice view, devices may be split into more than one piece, and drift spaces are added (note – the XAL initialization database does not have drifts as accelerator nodes, only actual device information).

The different probes in the online model represent different physics aspects of charged particles beam. There are three probe types commonly used:

- The envelope probe: This is a correlation matrix of moments in a 6D phase space up to second order. By using this probe we can simulate the beam emittance transformation along the lattice.
- The transfer map probe: This represents the transformation matrix of 6D coordinates from the beginning of the sequence to a particular point at the lattice. This probe is usually used for ring modeling.
- The particle probe: This represents the center of the beam. It is frequently used for phase scan analysis and orbit predictions.

Each probe has one or several corresponding algorithms describing the tracking of the probe through the lattice.

The XAL online model went through a series of verifications and benchmarks [2]. The most important feature of the online model is the speed of calculations. The simulation time for SNS sequences is usually much less then one second, and that enables use of the online model in the control room even if a problem includes multiple runs of the model (optimization procedures).

XAL APPLICATIONS FRAMEWORK

In the early days of XAL development, each application was created with its own JFrame menus, toolbars, and standard functionalities like open, save etc. This approach meant duplicated efforts, a different look and feel for each application, and maintenance difficulties. To avoid this the XAL Application Framework was created [4].

An application framework is a set of classes that actual applications extend, and it is used as a common starting point for all XAL applications. There are several advantages of using this framework.

The framework provides all applications with the same look and feel, which helps operators and users to more easily get acquainted with new applications. An example of the application framework template is shown in Fig. 2. A standard “windows application” menu bar, tool bar and empty panel is provided as a starting point, modeled after the familiar windows application format. All menu and toolbar items and actions within them can be customized by changing a simple configuration file which is unique for each application.

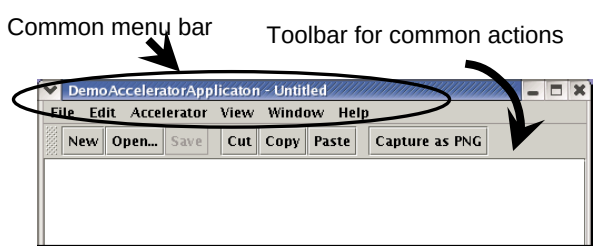

Figure 2: The accelerator application framework template.

The framework uses a document-view architecture, i.e. a single application can have multiple documents

associated with it, and each with its own window view. The application document can be stored (restored) in (from) a file. Of course, this functionality should be provided by the document.

An Accelerator extension of the XAL application framework document class provides an accelerator model with a specific set of menu items. This extension enables the application to read an accelerator file, and to choose and create accelerator sequences.

The application framework usage is not mandatory for XAL applications, but it speeds up and facilitates their development and modifications.

XAL SERVICES

XAL services are a special type of XAL applications. They run 24/7 in the background; they can communicate with any number of standard XAL applications; and they do not have a GUI interface. A Service-Application communication uses XML-RPC for inter-process data exchange and uses multicast DNS for discovering subscribers and publishers. Knowledge of the details of multicast DNS and XML-RPC are hidden from the user.

There are several advantages of using the services. First, users need not worry about starting or restarting them. They will restart automatically in the case of shutdown or crash. Second, they reduce the amount of network traffic by avoiding duplicated EPICS and database requests from different XAL applications running simultaneously. And finally, they are prototypes of a future XAL distributed agent systems that is currently under development.

In this paper we present as examples two services that are useful for each accelerator facility.

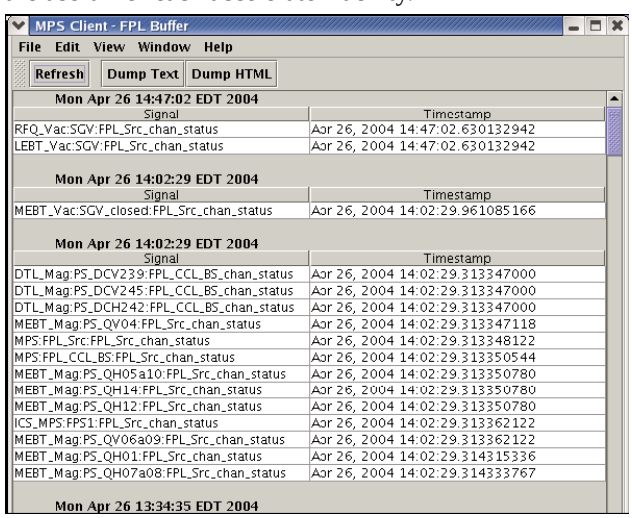

Figure 3: An example of the XAL MPS client application.

MPS Service

MPS service is our Machine Protection System (MPS) post-mortem application. Originally it was a standalone client application. Later it was migrated to the service framework. This service is always running in the background monitoring MPS events – capturing the stream of signals that emanate from each trip and sorting them to determine the root cause of the trip. It also provides statistics and views of the MPS trip history. Any number of client applications can view this data. Figure 3 shows a client view of the MPS trip.

PV Logger

Another service application is the XAL "PV Logger". This application logs predefined sets of control system signal values to a database, at specified intervals and upon requests from any XAL application. One example of use is to grab machine settings directly used by accelerator physics such as magnet, RF and BPM values. This provides complete sets of information needed to configure the online model, taken by a background process. At SNS we have several PV Loggers covering the needs of different systems like accelerator physics, SNS cooling system, beam loss monitors, etc.

XAL AUXILARY PACKAGES

XAL has numerous general purpose packages. They were developed at different times, and the list is still growing. Usually these packages are independent from the rest of XAL, and they can be easily ripped off and used elsewhere.. Below we discuss a few of the most interesting and useful of these packages .

XAL Plotting Package

Development of the XAL plotting package [5] started at the early stages of development as a research project to study how fast data plotting can be updated. Later, interactive features of the package were found useful in several applications. This package is not intended to completely replace existing and freely available powerful plotting packages, but it is sufficient for instances where simple charts and color surface plotting is required.

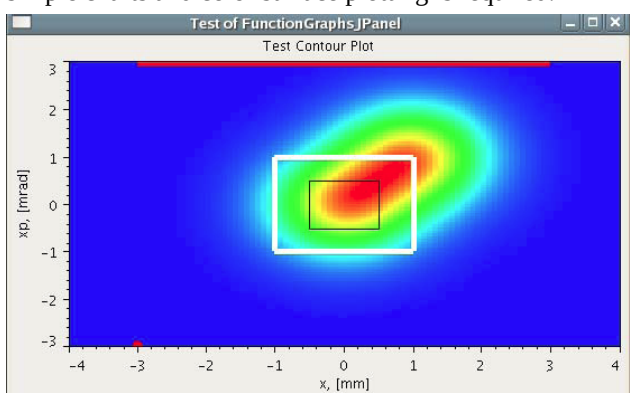

Figure 4: A color surface plot with XAL plotting package.

In terms of the Model-view-controller (MVC) pattern, the XAL internal plotting package is simplified, and it has only two components. The view and controller are combined, and they are implemented in the FunctionGraphsJPanel Java class, which does not have any subclasses. There are four major types of data (Model components). Two are related to 2D chart plotting. The third can be used for bar-charts, and the last one for color

surface 3D plotting. An example of the color surface data plotting with 100x100 point graphics area resolution is shown in Fig. 4.

XAL Channel Access Package

To communicate with accelerator hardware XAL uses the EPICS Channel Access protocol. EPICS communication uses a single "Process Variable" (PV) as the fundamental unit for communication via an EPICS protocol called Channel Access. XAL has a Channel class that encapsulates the communication with a process variable.

The Channel class is an abstract class that has the same interfaces that most control system would provide. This abstract layer insulates the rest of XAL from possible changes in the existing implementation of the EPICS protocol. For EPICS PV communication, we extend it to a concrete class that wraps Java Channel Access (JCA) [6] or Channel Access for Java (CAJ) [7] packages. JCA has interfaces to native C routines, and until recently it was the only non-Java library we used. Now there is CAJ - a 100% pure Java implementation of the EPICS Channel Access library.

The Channel class conveniently hides from users the underlying actions required to make connections to PVs. It also has member functions to provide Process Variable parameters other than the value (e.g. times stamps, units, display limits, etc.). Additionally, it has the capability to switch between synchronous and asynchronous communication and PV monitoring. Another useful feature of the Channel class is the ability to apply a specified "transform" to a Channel value. For example, one may apply a scaling transformation on a power supply current, in order to get a magnetic field level.

Typically applications dealing with the accelerator classes never actually use Channel objects directly, but rather they use methods that provide the information of interest. For example, with a magnet the user may call a getField() or setField() method to get or set the magnetic field. The actual Channel and control system connection details are hidden.

BRICS Package

XAL Bricks package (gov.sns.tools.bricks) is a tool to facilitate writing of GUI interfaces. The main idea of Bricks is to keep information about the GUI window elements and their appearances in a XML file which can be used later to restore the GUI window. The developer can get the reference to the GUI window simply by specifying the name of this XML file. Then, all effort can be concentrated on the functionality of an application instead of its appearance.

The Bricks package also has been integrated with the XAL application framework. During the runtime the bricks definition file can be loaded to create the main XAL application window. After that, all child views are available for usage inside the XAL application document.

To create XML Bricks files XAL includes a GUI builder based on the Bricks package. This builder is a XAL application and works like others GUI builders for many existing Integrated Development Environments (IDEs). The key difference between existing GUI builders and XAL Bricks is that Bricks does not create any Java code. This design for Bricks is inspired by OpenStep's™ Interface Builder™ [8]. The snapshot of the Bricks GUI builder application is shown in Fig. 5.

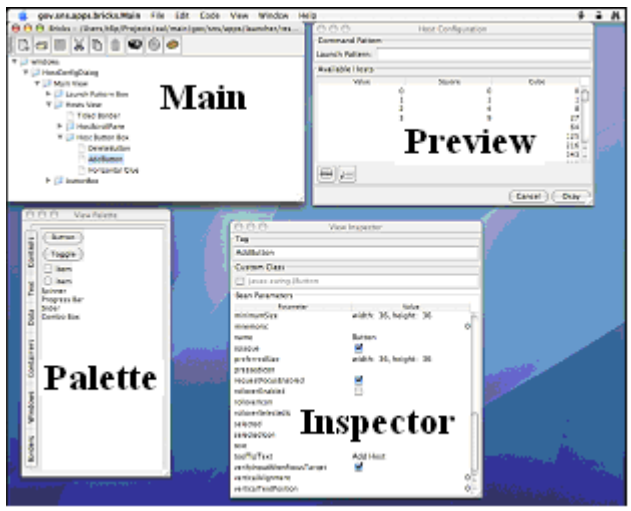

Figure 5: XAL Bricks GUI builder.

The main window for the Bricks document shown in Fig. 5 displays a hierarchy of windows and their views for a future GUI interface. A palette of views shows possible views such as buttons, tables, text fields etc. so the developer can simply select and drop views into the main window. A preview window reveals how the GUI window will look. An inspector is used for editing the properties of a selected view. Copy, Cut and Paste support along with drag and drop support make it easy to rearrange views. The built in code assistant helps developers avoid errors by generating references to views to pass directly into their code.

General Optimization Package

The XAL optimization package (gov.sns.tools.solver) is a "home grown" product of a long evolution of such packages during the XAL development. Currently it is the third generation of optimization packages in XAL. The base components of the package are:

- The Solver component: This is a primary class for setting up and running an optimization process. To create an instance of the Solver we need instances of the Algorithm Market, Stopper, and Solution Judge classes. The Problem class is needed to start the optimization process in the Solver.
- The Algorithm Market: This is a collection of algorithms that will compete during the optimization process. The user can choose arbitrary combination of algorithms from the existing ones in the package.
- The Stopper Class and Subclasses: This decides if the process should stop. The decision is made on the basis of time, number of iterations, a level of

satisfaction, etc. Users can choose which criterion or which combination of criteria will be used.

- The Solution Judge component: The implementation of the Solution Judge interface compares the new solution and the best known one at a particular moment.
- The Problem Class: This class consists of the list of variables, restriction rules, possible hints for different algorithms, and a scorer that estimates the model function.

At the moment there are four available algorithms: a random search, a random search with a shrinking rage of search, a direct gradient method, and a simplex algorithm.

The XAL optimization package has no upward connection to other XAL classes except the XAL message center, and it can be freely used outside of XAL.

Least Square Method Package

There is a special linear squares method (LSM) package (gov.sns.tools.fit.lsm) in XAL to solve optimization problems that can be reduced locally to a linear problem. Usually, we want to find an approximation of measured points (x,y) by a known function with unknown parameters. This type of problems has exact solutions, so there is no need to use the general XAL optimization. The XAL LSM package implements two algorithms: a classic LSM and a Levenberg-Marquardt Method (LMM) [9]. Both methods assume that the user will provide methods to calculate partial derivatives of the model function with respect to the model parameters.

The two types of functions most frequently used in XAL applications are a Gaussian distribution and a damped sinusoidal oscillation,

$$y_{gauss}(x) = a_0 + a_1 \cdot x + \exp\left\{a_2 (x - a_3)^2\right\},$$

$$y_{dump}(x) = a_0 + a_1 \cdot \exp(a_2 x) \cdot \sin(a_3 (x - a_4)),$$

where a_i are the unknown parameters. The Gaussian distribution function is used in wire scanner data analysis, and the damped oscillations are used in SNS ring tune calculations. To provide initial values for the model parameters, both fitting classes have a “guess” method which analyzes the initial data and suggests reasonable values. There is also a polynomial class that calculates coefficients of arbitrary power series.

Another advantage of the LSM package with respect to the general XAL optimization is that errors for model parameters can be estimated if the user specifies the errors in the measured data.

Formulas Evaluation Package

The formulas evaluation package (gov.sns.tools.formula) is used to evaluate a formula with a given set of variables provided by the user. The formulas are presented as text. A Formula Interpreter class compiles the text and evaluates the formula. The compilation process is performed only once, so the efficiency of following evaluations is very high. This allows use of the package for fitting problems inside the optimization package. The formulas evaluating package has a variety of operators including arithmetic operators, function operators, and logical operators. The set of internal functions can be extended.

XAL APPLICATIONS

At this moment, only the SNS project has more than 50 XAL applications. They were developed at different times by different developers, and many of them do not comply completely with all standards for correctly implementing XAL applications. In this paper we consider only few of them which could be useful everywhere.

To write a correctly implemented general XAL application the developer should follow several rules:

- The application cannot use java classes from another application. If a Java class is useful for more than one application, it should be moved into the core XAL packages.
- The source code of the application should not contain any specific information related to a particular accelerator.
- The application should use the accelerator model for hardware interaction. The model should be initialized from the default XML accelerator model file.
- The number of external packages should be minimized.

Virtual Accelerator

A virtual accelerator (VA) is a simulation program running permanently and generating sensible diagnostics signals in response to changing parameters of the model. From the point of view of a client that communicates (through EPICS or another protocol) with this VA there is no difference between the VA and a real machine. The VA is a very useful tool for developing and testing control room applications, especially during the early stages when the real accelerator is not functioning yet.

In the beginning of XAL development, the virtual accelerator was based on PARMILA or TRACE3D codes and an EPICS Portable Channel Access Server (PCAS) [10]. The structure of this VA is shown in Fig. 6. Later, the XAL online model was used as a simulation program.

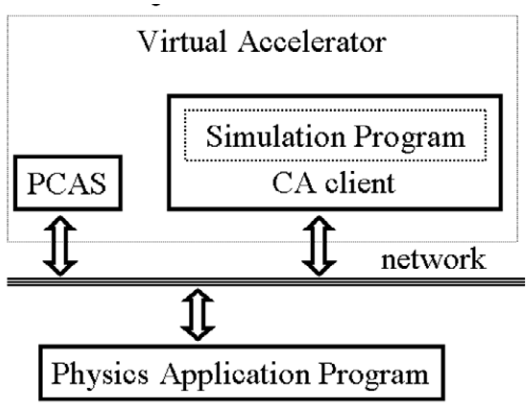

Figure 6: The structure of a virtual accelerator.

Until recently, PCAS was the only option to provide communications between an application and the simulation program. It was inconvenient because each time the user had to generate a special initialization file with a list of EPICS PV names that will be used in the VA. After replacing the JCA wrapper around the native EPICS C-library by the CAJ (pure Java EPICS communication package [7]), the PCAS was dropped from XAL. Now the XAL Virtual Accelerator Application does not need any external executables and produces all necessary EPICS signals by itself. The user needs only the XML Accelerator Model file to start VA.

Orbit Correction Application

An orbit correction application was developed to correct transverse orbit errors with dipole correctors and bend magnets in the linac and the ring. Among the features of this application, there is the ability to save and restore the existing orbit, to correct orbit errors to zero or a reference orbit, to switch on and off any particular dipole correctors from the process, and to initialize from the XML accelerator model file. This application utilizes the XAL optimization package by specifying a goal of optimization as a combination of zeroing orbit errors, by providing a smooth orbit, and by keeping the currents in the correctors within their control limits. The satisfaction function of this optimization is a nonlinear combination of all goals. The use of nonlinearity eliminates the need to introduce artificial weights, as is often necessary with many linear optimization approaches.

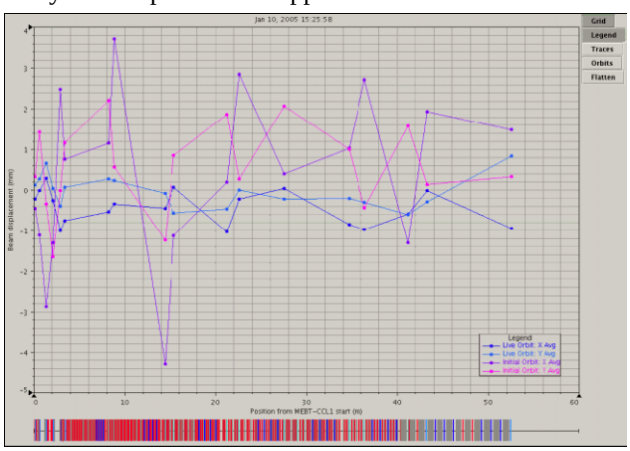

Figure 7: Orbit correction for the MEBT-DTL-CCL1 SNS's linac sections.

When correcting an orbit, the user can choose whether to correct the orbit based on the XAL online model or through empirical measurement. For the first case, the orbit response coefficients of the dipole correctors for the downstream beam position monitors (BPMs) come from the online model. If the model is incorrect the orbit correction will fail. In this situation the empirically measured coefficients can be used. The empirical approach will work even in a case where there are dipole correctors and BPM polarity errors. The disadvantage of this method is the long period of time needed to measure the orbit response to each dipole corrector.

The result of the orbit correction for part of the SNS linac is shown in Fig. 7. The four trajectories are the initial horizontal and vertical trajectories (purple and pink, respectively), and the horizontal and vertical trajectories after correction (blue and light blue, respectively). The initial trajectory excursion is about 4 mm horizontally and 2 mm vertically. After correction, trajectory oscillation in both planes is within 1 mm.

The XAL orbit correction application can also serve as a live orbit display.

CONCLUSION

The XAL framework has three basic interconnected components: the hierarchical accelerator model, the online model, and the XAL application framework. Additionally, XAL includes a collection of independent Java packages that can be used anywhere.

Correctly implemented XAL applications can be easily ported to other accelerators, and there are persistent efforts to make the XAL more portable.

ORNL/SNS is managed by UT-Battelle, LLC, for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] J Galambos et al. "XAL Application Programming Framework", ICALEPCS'03, Gyeongju, Korea, October 2003, WE204 , http://www.JACoW.org.

[2] C.K. Allen et al. "A novel online simulator for applications requiring a model reference", ICALEPCS'03, Gyeongju, Korea, October 2003, WE116, http://www.JACoW.org.

[3] N. Malitsky and R. Talman, "The Framework of the Unified Accelerator Libraries", Monterey, CA, USA, September, 1998, ICAP 1998.

[4] T. Pelaia "XAL Application Framework and Bricks GUI Builder", ICALEPCS'07, Knoxville, TN USA, October 15-19, 2007, TPPA09, http://www.JACoW.org.

[5] A. Shishlo et al. "Java Swing-Based Plotting Package Residing Within XAL", ICALEPCS'07, Knoxville, TN USA, October 15-19, 2007, TPPA08, http://www.JACoW.org.

[6] http://www.aps.anl.gov/xfd/SoftDist/swBCDA/jca2/index.html.

[7] http://caj.cosylab.com.

[8] Nancy Craighill, OpenStep for Enterprises, John Wiley & Sons, Inc., New York, NY, 1997.

[9] Numerical recipes in C: The art of scientific computing. Cambridge University Press., 1992.

[10] A. Shishlo et al. "The EPICS Based Virtual Accelerator - Concept and Implementation", PAC'03, May 12-16, Portland, OR, 2003, WPPE017, http://www.JACoW.org.