SNS APPLICATION PROGRAMMING ENVIRONMENT*

J. Galambos, C. M. Chu, T. A.Pelaia, A. Shishlo, ORNL, Oak Ridge TN USA
C.K. Allen, N. Pattengale, LANL, Los Alamos NM USA

Abstract

The Spallation Neutron Source (SNS) Application Programming effort provides software support for physics related applications. At the core of this programming framework is a Java class library (XAL) that provides an accelerator based hierarchal view to the programmer. The accelerator hierarchy represents familiar beamline device components such as quadrupoles, corrector elements, BPMs, etc. The direct interfaces to the machine control system (EPICS) are hidden by general methods for each beamline device type. Static configuration information is stored in a database, and read into XAL via an XML interface. Initial applications are described.

1 SNS APPLICATION PROGRAMMING SCOPE

The SNS accelerator is being constructed at a greenfield site in Oak Ridge TN. The Application Programming effort covers high-level software, typically involving some sort of physics modeling or interaction between a collection of beamline devices. Given the situation of having no pre-existing legacy software, the possibility of having a relatively uniform software interface to the entire accelerator exists. To take advantage of this situation, we are creating a Java based programming hierarchy, with several goals in mind, including: 1) providing an accelerator physics based hierarchal programming framework of the accelerator, 2) offer a programming interface to accelerator physicists that hides the connection details to the underlying control system, and 3) offer reusable components for site-wide use.

Section 2 describes the overall programming framework including interfaces with other systems. Section 3 provides an overview of the XAL class structure, and Section 4 describes some prototype applications.

2 XAL PROGRAMMING INFRASTRUCTURE

SNS has adopted a Java based programming infrastructure called XAL [1]. This class structure started as the UAL model [2], and is also similar to the COSYLab Databush/Abeans Java system [3]. Some XAL documentation including the Java-Doc class description may be found in Ref. [4].

2.1 Relationship with the Control System

The relationship between the XAL programming infrastructure and other SNS accelerator and software components is shown schematically in Figure 1. XAL communicates with hardware (magnets, RF, diagnostics, etc.) with the EPICS control system, in particular using the EPICS channel access.

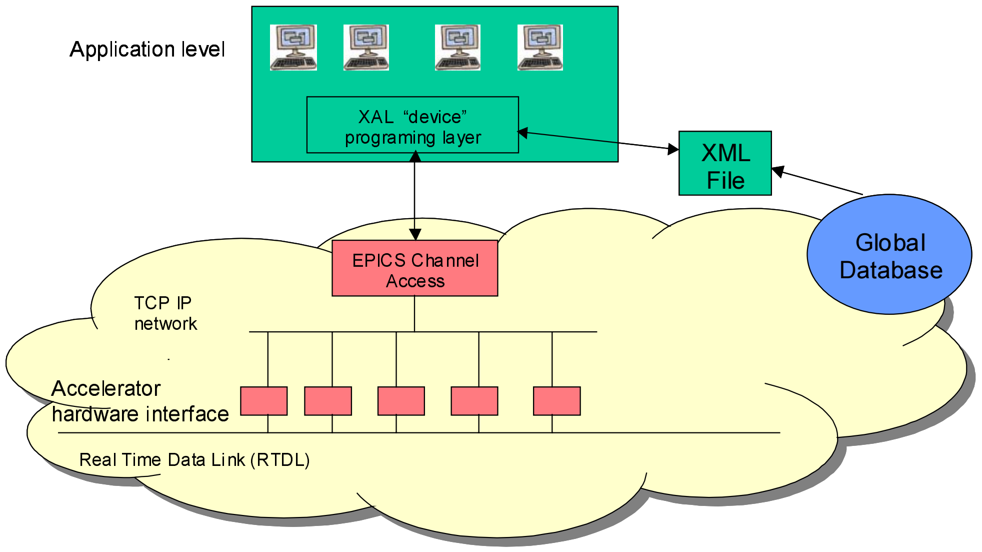
Application level
XAL "device" programing layer
XML File
Global Database
EPICS Channel Access
TCP IP network
Accelerator hardware interface
Real Time Data Link (RTDL)

Figure 1: Relationship of the XAL application-programming layer with other SNS components.

* SNS is managed by UT-Battelle, LLC, under contract DE-AC05-00OR22725 for the U.S. Department of Energy. SNS is a partnership of six national laboratories: Argonne, Brookhaven, Jefferson, Lawrence Berkeley, Los Alamos, and Oak Ridge.

Two classes are provided to assist the connection between XAL and the control system. One is a Channel class, which provides an object view of the individual EPICS process variable. The Channel class uses Java Channel Access (jca) [5] to communicate with EPICS. Another helper class is the ChannelManager, which helps manage the relationship between particular Channel objects and tasks for Accelerator Nodes. This is a key mechanism in the abstraction of actions away from the individual signal names used by EPICS and toward a common interface for all node actions of a similar type.

2.1 Relationship with Global Database

One purpose of XAL is to add a richer programming framework than that provided by the underlying EPICS control system. The XAL framework of accelerator objects (or object-graph) contains a representation of the actual machine elements, and is instantiated from a global database. The global database also includes information used to set up the EPICS control system. Use of a common database facilitates mapping of the individual EPICS process variable names to the higher-level XAL constructs. Although it is possible to directly query the global database from within XAL, an intermediate XML file containing the accelerator hierarchy is created from the database. This step avoids the penalty of large database queries as each application starts up. There are plans for a server to provide the object-graph if reading the fully populated XML configuration file proves to be too slow. An object-graph server would also present an authoritative view of the "current" accelerator.

3 XAL CLASS STRUCTURE

3.1 The Accelerator Framework

A primary purpose of XAL is to provide a programming hierarchy similar to that with which the accelerator physicist views the accelerator. Rather than interact with the machine via a long list of independent signals, XAL provides familiar accelerator objects to work with. This framework is shown schematically in Figure 2. For example, the SNS accelerator is composed of sequences, which are in turn composed of nodes. Nodes are typically common beamline elements such as magnets (dipoles, quadrupoles, horizontal correctors etc.), RF cavities, or diagnostics (Beam Position Monitors (BPMs), current monitors, wire scanners, etc.). Each of these node types include methods to do commonly associated activities. For example a magnet has methods to get and set fields. A BPM node has methods to get horizontal or vertical beam positions.

3.2 Additional Features

Node classes have "Attribute Bucket" members that contain static data typically initialized from the database, e.g. design values. There are generalized methods for parsing Attribute Bucket information from the XML input, which eases the introduction of new Node types. Also, Node types have qualifiers to permit easy collections of specified types of Nodes (e.g. getting all magnets, getting all horizontal dipole correctors, or getting all BPMs from a sequence). In order to separate the static information, related to the setup of the Accelerator hierarchy, and the dynamic data input from the machine, an "Edit-Context" view is provided, which contains only the dynamic information. Multiple Edit-Context views of the same Accelerator object-graph are possible, with each view containing a pointer to the originating object-graph. Separating the static and dynamic storage eliminates storing redundant static data for each "snapshot" view of a machine.

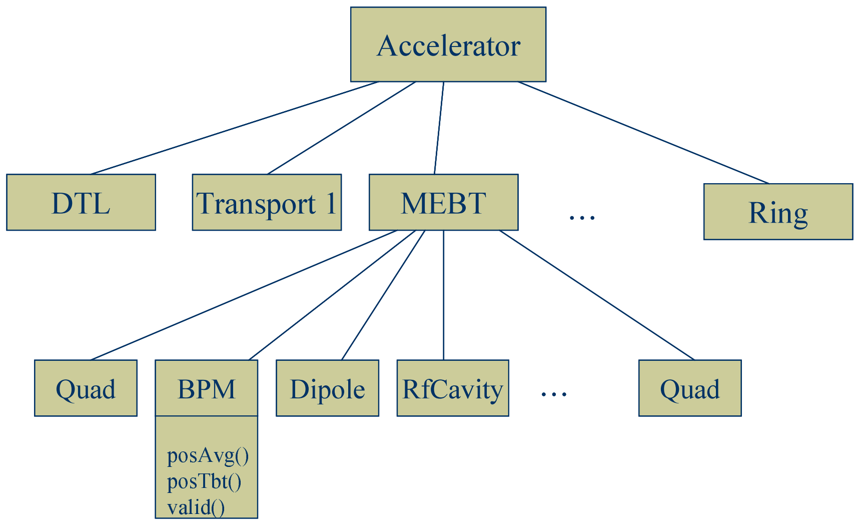

Figure 2: Schematic view of the accelerator hierarchy provided by XAL.

For a scripting interface to XAL components, we use Jython [6]. Jython allows direct use of Java classes within the python scripting language, without having to provide any "interface-glue" code. This is useful for quick prototyping and testing of new XAL components, and we also anticipate using it for on-the-fly algorithm writing during beam commissioning. Some example script interfaces to XAL components are in Ref [4].

4 EXAMPLE APPLICATIONS

First beam commissioning at the SNS Oak Ridge site is scheduled for November 2002. To prepare for this, applications have been tested in two ways: 1) using a virtual accelerator, and 2) remotely running the applications with the ongoing Medium Energy Beam Transport (MEBT) beam commissioning taking place at the Lawrence Berkeley Lab (LBNL) SNS collaboration site. The virtual accelerator is a standalone model driven program using the EPICS Portable Channel Access Server to provide the same interface as will be seen with the real machine. The virtual accelerator has presently been set up for the MEBT, and plans are to similarly provide an artificial machine interface for application testing for the other accelerator sequences, as they approach commissioning.

Presently, we have three applications using the XAL framework: There is an Orbit difference application which looks at changes in the beam trajectory with a magnet change for both the BPMs and a model prediction (presently we are using the Trace-3D [7] model in the linac applications). Another application is orbit correction that has a variety of algorithms to center the orbit (presently using BPM readings). Finally there is an application called XIODiag that allows a user to traverse the accelerator hierarchy to examine or modify machine parameters. A common feature of these applications is that they are compatible with future SNS beamline sequences, as they become populated in the Global Database. Figure 3a shows screen snapshots of the XIODiag application running, demonstrating the drill down capability to quickly monitor values. Figure 3b shows a screen snapshot of the orbit correction application. Both these applications are running at the Oak Ridge SNS site, using live data from the MEBT at LBNL.

Work is ongoing for providing built-in modeling capability, using the algorithm/probe architecture described in Ref. [2]. The first set of built-in algorithms will be similar to the Trace-3D methods.

5 REFERENCES

[1] C.M. Chu et.al.,"SNS Application Programming Plan",http://www.slac.stanford.edu/econf/C011127/THAP060.shtml

[2] N. Malitsky, "A Prototype of the UAL 2 Toolkit", http://www.slac.stanford.edu/econf/C011127/THAP013.shtml

[3] Igor Kriznar, Mark Plesko, "The Object Oriented Approach to Machine Physics Calculations with Java Technology",http://www.slac.stanford.edu/econf/C011127/THCI001.shtml

[4] http://www.sns.gov/APGroup/appProg/xal/xal.htm

[5]http://www.aps.anl.gov/xfd/SoftDist/swBCDA/jca/jca.html

[6] http://jython.org/

[7] K. Crandall, D.P. Rusthoi, "TRACE 3-D Documentation, Los Alamos National Laboratory report LA-UR-97-886, May 1997.

Acknowledgements: Thanks to N. Malitsky for the genesis of XAL and to the SNS controls and diagnostics groups for valuable input.

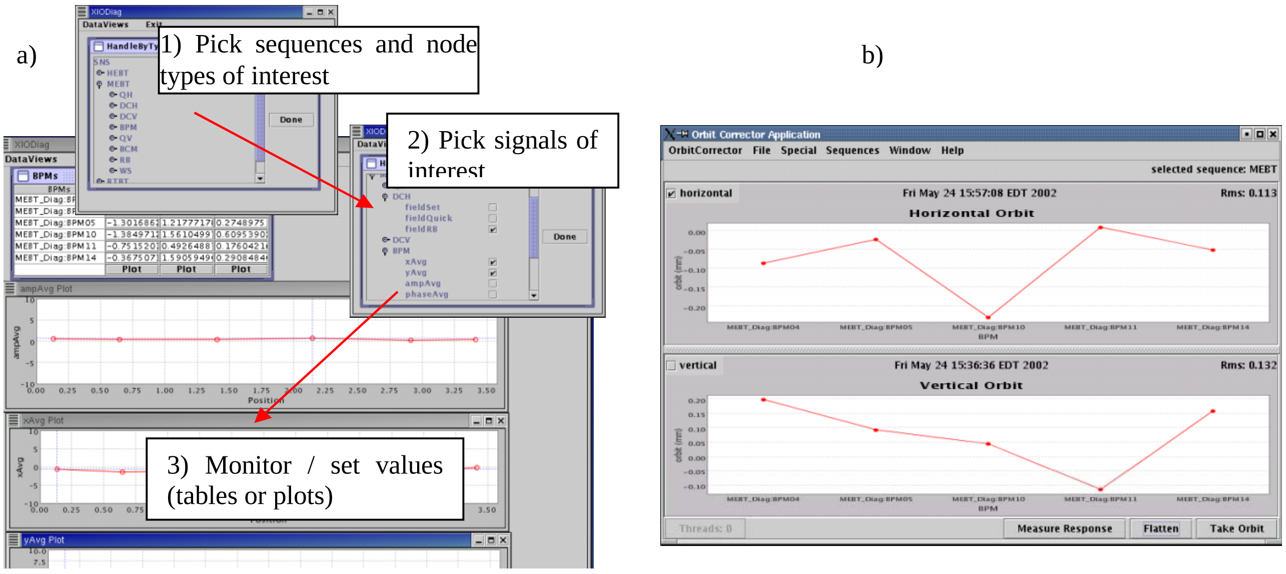
a)
1) Pick sequences and node types of interest
2) Pick signals of interest
3) Monitor / set values (tables or plots)
b)
Orbit Corrector Application
OrbitCorrector File Special Sequences Window Help
selected sequence: MEBT
horizontal
Fri May 24 15:57:08 EDT 2002
Rms: 0.113
Horizontal Orbit
vertical
Fri May 24 15:36:36 EDT 2002
Rms: 0.132
Vertical Orbit
Measure Response
Flatten
Take Orbit

Figure 3: Screen snapshots of a) the XIODiag application and b) the orbit correction application.