XAL APPLICATION PROGRAMMING FRAMEWORK*

J. Galambos, C. M. Chu, W.-D. Klotz, T. Pelaia, A. Shishlo
SNS, Oak Ridge National Laboratory, C.K. Allen, C. McChesney
Los Alamos National Laboratory, I.Kriznar, M. Plesko, A. Pucelj
Cosylab, Ljubljana Slovenia

Abstract

The Spallation Neutron Source (SNS) is using a Java based framework for application program development. The framework, called XAL, is designed to provide an accelerator physics programming interface to the accelerator. Much of the underlying connections to the EPICS control system are hidden from the user. Use of this framework allows writing general-purpose applications that can be applied to various parts of the accelerator. Also the accelerator structure is initiated from a database, so introduction of new beamline devices or signal modifications are immediately available for all XAL applications. An on-line model is included in this framework for quick beam tracking. The overall framework is described, and example applications are shown.

1 INTRODUCTION

A part of the SNS accelerator physics activities include preparation of an application programming interface to the machine. This interface augments the EPICS [1] control system, and is aimed at applications that are more complicated than simply setting, displaying or monitoring values. Some primary goals of XAL [2] are to provide a hierarchal framework, much like the physical view of the machine: i.e. an accelerator composed of sequences; and sequences composed of components such as magnets, diagnostics and RF cavities. Another objective is to write general purpose software that can be applied to various parts of the machine. To accomplish these goals, we have written a Java based system, and constructed an Oracle database from which to initialize the accelerator objects. In section 2 the class structure we use is discussed, in section 3 the database connection is described and in section 4 some example applications are presented.

2 XAL STRUCTURE AND FEATURES

2.1 Accelerator Model

At the heart of XAL is a set of classes describing an accelerator hierarchy, shown schematically in figure 1. The accelerator is composed of a set of sequences, which in turn are composed of sets of ordered nodes. The node structure includes classes for common beamline components such as magnets, RF cavities and diagnostic types. The magnet class is further sub-classed into the usual multi-pole types (e.g. dipole, quad) as well as permanent and electromagnet classes, etc.

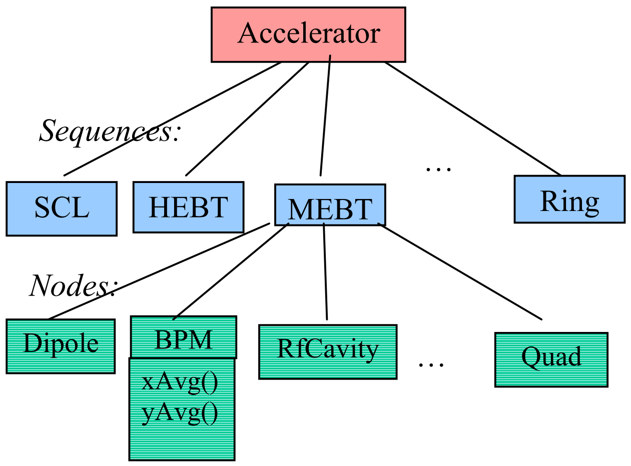


Fig.1: Schematic of the XAL accelerator class hierarchy.

Methods are provided throughout the accelerator class structure to easily perform common tasks such as: 1) selecting nodes of a certain type from a sequence, 2) getting or setting magnetic fields in a magnet, or 3) finding the beam position from a diagnostic Beam Position Monitor (BPM). Importantly, the details of the control system connection are hidden from the user. The Accelerator node objects generally correspond to the physical devices actually in the accelerator beamline, but are not necessarily one-to-one mappings. For example, at SNS there are single hardware devices consisting of a quadrupole + dipole windings + BPM strip-lines. We treat these functions as three separate nodes (quad + dipole corrector + BPM), all at the same position.

2.2 The Control System Connection

XAL uses EPICS as the underlying control system to communicate with accelerator hardware (see Fig. 2). EPICS communication uses a single "Process Variable" (PV) as the fundamental unit for communication with higher level programs via a protocol called Channel Access. XAL has a Channel class that encapsulates the communication with a process variable. There are typically many Channel objects associated with a particular accelerator node type, and these are held in a ChannelSuite (which is simply a collection of channels for that device). For example the BPM device class has ChannelSuite that holds Channel objects for providing the horizontal beam position, vertical beam position etc.

*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.

The Channel class is an abstract class that has interfaces most control system would provide. For EPICS PV communication, we extend it to a concrete class that wraps to Java Channel Access (JCA) [3]. This class presently has interfaces to native C routines, and is the only non-Java library we use. We hope in the future to provide a native Java interface to the EPICS channel access.

The Channel class also provides convenience capability, by hiding some of 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 stamp, units, display limits, etc.). Also capabilities to switch between synchronous and asynchronous communication and PV monitoring are provided. 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. In this sense, it is possible to assign EPICS PVs to more than one XAL Channel, with some Channels representing the raw PV value, and others with value added transformations.

Typically applications dealing with the accelerator classes never actually use Channel objects directly, but rather use methods that directly provide the information of interest. For example, with a magnet the user may use a getField() or setField() method to get or set the magnetic field. For a BPM, the user can use a getXAvg() to get the average horizontal beam position. The actual Channel and control system connection details are hidden. The programmer can however provide local exception handling, in-case the underlying calls to the control system fail to connect.

2.3 Tools

Several general purpose tools are provided that are used in applications and classes. Some of the more extensively used tools are described here.

SNS is a pulsed device, and will operate at 60 Hz eventually. EPICS will provide a timestamp to each PV, from a common timing system. It is important to be able to gather many signals from a common pulse, when analyzing beam behavior. To this end a correlator package is provided. A list of requested PVs, time window for correlation and fraction of PVs gathered to consider success are input. The correlator can be used to provide a steady stream of correlated PVs from a common pulse to a listener, or to provide results at a prescribed rate. Another correlator feature is the use of customizable filters, which can be used to selectively gather data sets. For example, a minimum threshold on one PV can be used as a trigger mechanism for acquiring other data.

A DataTable set of classes provides a simple database like capability. These classes are more powerful than the Java collection classes, and are much easier to use than constructing an external data base. An example of their use is in the Save/Restore/Compare application, to provide a query-able structure that contains a custom hierarchy of system and device types.

We also use widgets from the Cosylab ABeans project [4] to augment XAL tools. These include value selection and display tools such as a convenient thumbwheel selection tool. We use the Cosylab synoptic beamline device display on charts to easily identify the portion of the accelerator being analyzed.

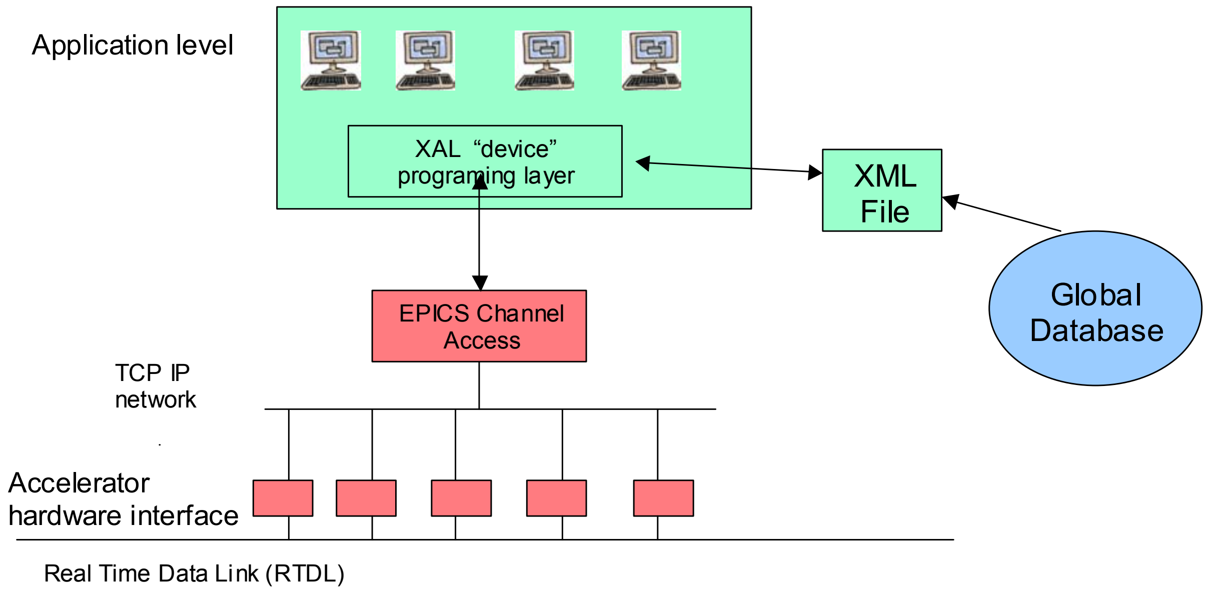

Application level
XAL "device" programing layer
XML File
Global Database
EPICS Channel Access
TCP IP network
Accelerator hardware interface
Real Time Data Link (RTDL)

Fig. 2: Overall relationship of the XAL application framework, and the SNS Control system.

The XmlDataAdaptor class provides a way to read and write data in XML format, from a generalized structure and is used extensively. For example it is used to parse the overall accelerator structure from an xml initialization file (see Section 3). Also, specific applications have open and save capabilities. The application data initializations (open) and saving are typically done with this class (most of the file data storage in XAL is done in XML format).

2.4 Online Model

An important XAL feature is an online accelerator model [5], which allows for on-the fly calculation of beam parameters, based on machine settings. The main components are a lattice (constructed from the XAL accelerator nodes) and a probe (describing the beam, and how it is to be modeled). The relationship between the lattice and the XAL sequence structure is shown schematically in Fig. 3. The lattice is generated via a set of rules, from the accelerator node device information. In the transformation to the lattice view, devices may be split into more than one piece, and drift spaces are added (note - no drift information is stored in the XAL initialiazation database, only actual device information). Also a visitor pattern scheme is used to facilitate synchronization of the lattice view parameters, with updates from different sources.

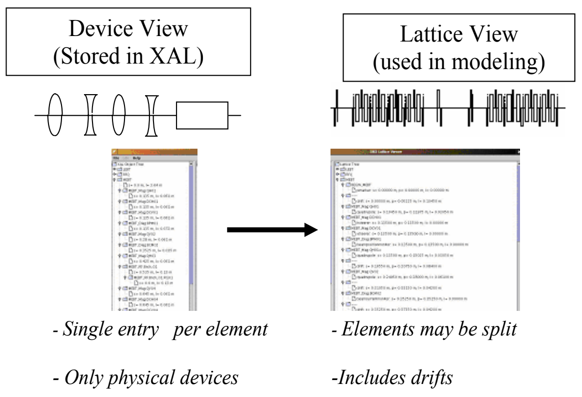


Fig. 3: Relationship between the XAL device view and the online model lattice view of the machine

2.5 Scripting Interfaces

In addition to creating pure Java applications to exercise XAL capabilities, scripting interfaces are available. These include Jython[6] and Matlab™ interfaces. In both these cases, no glue code, special wrapper code or build steps are needed; rather the Java classes can be directly imported into the scripting level and used seamlessly along with the scripting language. Although we have written complete applications in Jython, the dominant use of these scripting interfaces is for quick testing and providing easily digested examples. The scripting interface makes it possible to exercise code features with only a few lines of code, facilitating testing and easily showing others how to use features via examples.

3 THE DATABASE CONNECTION

XAL provides a hierarchal framework, but the actual objects within this structure need to be initialized. We initialize the accelerator structure using information from “beamline component” tables in the global SNS relational database [7]. The global database is an extensive database containing information about both abstract devices, and specific hardware equipment. For XAL, the abstract device information is of interest. In particular, there are tables defining the sequences within SNS, and the devices within each sequence. A beamline device could be a magnet, diagnostic instrument or RF cavity. A SQL query produces either the XAL structure directly, or an intermediate XML representation.

We primarily use the intermediate XML file for XAL initialization for several reasons. First, it is faster to do the query once, and have individual applications read the intermediate XML file, rather than having each application perform the query on startup. Also, using the XML file, allows the possibility of easily overwriting database values. During commissioning activities this has proved quite useful. For example, if a BPM’s electronics are swapped with another, the switch of the PVs associated with the swapped electronics can be accommodated in the XML file. This change in a single XML initialization file, is sufficient to update all the XAL applications, since XAL uses a uniform initialization method. We also add Channel transformations (section 2.2) in the XML initialization file to quickly make temporary correction quick across the board .

An important link between the independent PV oriented EPICS system and the accelerator hierarchy oriented XAL structure takes place in this initialization step. I.e., the mapping from many individual EPICS PVs to a single beamline device channel-suite (see section 2.1) is done in this step. After this point, the EPICS PVs are assigned to a strict class structure. Presently, we use a naming convention to facilitate this mapping.

4 APPLICATIONS

XAL tools have been used to create about 10 applications. These applications use Java Swing GUI components. An important feature of the applications is the use of a common framework. The framework is discussed first, and some specific application examples are presented next.

4.1 The Application Framework

An application Framework has been created, to be used as a common starting point for all applications. The framework is a set of classes that actual applications extend. This approach has several advantages. First all applications have a common look and feel, which is important for operators and users to more easily get acquainted with new applications. Also, common application features are provided one time up front, and shared by all – this also helps jump start application development. Finally, it allows easy extension of all applications in a common place. The application framework template is shown in Fig. 4. A standard “windows application” menu bar, tool bar and empty panel is provided as a starting point, that the user can easily augment or cull. The application framework uses a document-view architecture, i.e. a single application can have multiple documents associated with it, each with its own window view. This is useful for general approach applications. The user can selectively pick the portion of accelerator and setup details and store the setup as a document. The standard menubar shown in Fig. 4 has typical functionality that a user familiar with windows applications would expect. The Accelerator menu items shown in Fig. 4 come with an Accelerator extension of the framework which provides a browsing capability of the accelerator hierarchy. Users can customize the menu items and their action handling as desired.

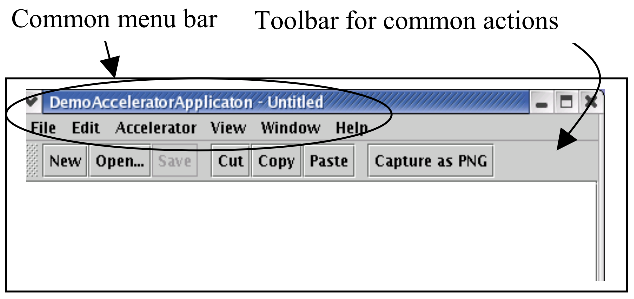
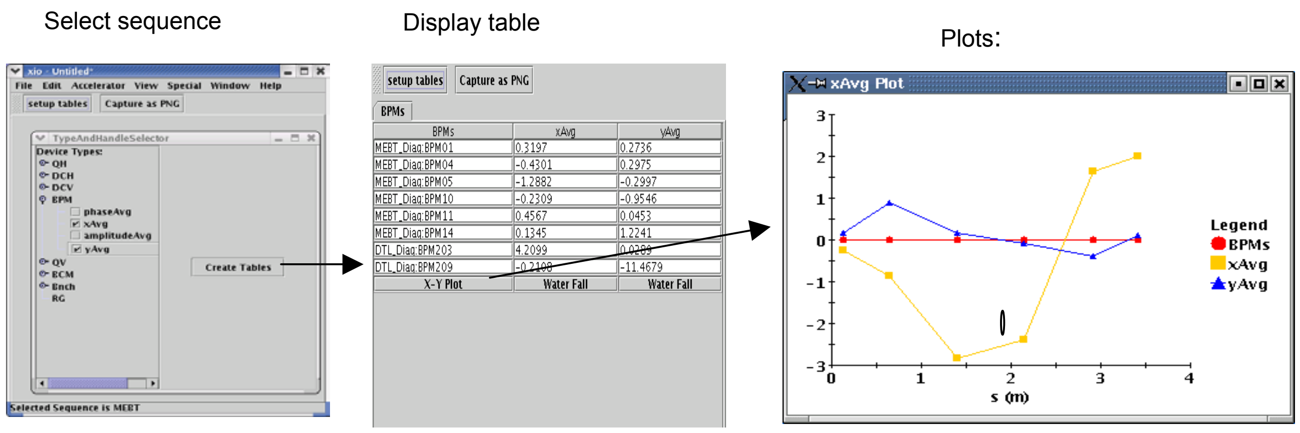


Fig. 4: The accelerator application framework template.



BPMs	xAvg	yAvg
MEBT_Diag:BPM01	0.3197	0.2736
MEBT_Diag:BPM04	-0.4301	0.2975
MEBT_Diag:BPM05	-1.2882	-0.2997
MEBT_Diag:BPM10	-0.2309	-0.9546
MEBT_Diag:BPM11	0.4567	0.0453
MEBT_Diag:BPM14	0.1345	1.2241
DTL_Diag:BPM203	4.2099	0.0289
DTL_Diag:BPM209	-0.2108	-11.4679

Fig. 5: The XIO application, progression from accelerator sequence selection t device type selection to signal type selection to data monitoring (tabular + plots).

4.2 XIO

XIO is a general purpose application to monitor sets of signals. The user first uses the accelerator sequence selection feature (part of the framework) to select the portion of the accelerator to work with. Then device types of interest within the selected accelerator sequence are picked (e.g. BPMs ...). Finally the signal types for the selected device types are picked (e.g. horizontal beam position signal type for the BPMs). A separate table is created for each selected device type within in a tabbed panel and within each table there is a column for each selected signal type. The table cells display the values at a prescribed update rate (typically 1-2 Hz). Options exist to display each column of signals as a live X-Y plot, where the horizontal axis is the distance along the selected accelerator sequence. Also possible is a color “waterfall” display of a signal type’s value (rendered via a color mapping) along the beamline (x axis) and versus time (y axis).

4.3 Scope

As SNS is a pulsed device, understanding time dependencies of quantities during a pulse is important. Also, there are minimal provisions for analog display of waveforms in the control room, rather waveforms are digitized and made available as EPICS PVs. A “digital scope” application is available for viewing the waveforms. An important requirement is to be able to overlay different waveforms from different sources (e.g. RF, BPMs, loss monitors, …). Since each waveform source uses different digitization methods, we require additional information for each waveform PV, describing the time offset of the first element from a common time point, and the time bin size that each array element corresponds to. With this information, waveforms from various sources can be displayed together, with real time as the x coordinate. The time correlator engine described in 2.3 is used to ensure that simultaneously displayed waveforms are from the same machine pulse.

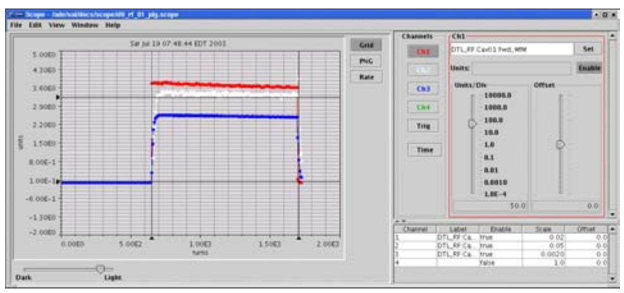


Fig. 6: The scope application display.

The scope trigger mechanism permits waveform acquisition based on the filtered behavior of another PV.

Also, a programmable math function is available to display functions of various waveforms. Preliminary testing shows good performance up to 10 Hz (test limits).

4.4 Scan application

A commonly used application is a general purpose 1-D scan. This allows the user to prescribe a scan of one quantity and monitor the behavior of other quantities. Capabilities include averaging over multiple pulses, delays between steps, analysis and exporting of results. This general purpose tool provides an easy, flexible way to perform quick unanticipated experiments in an automated fashion.

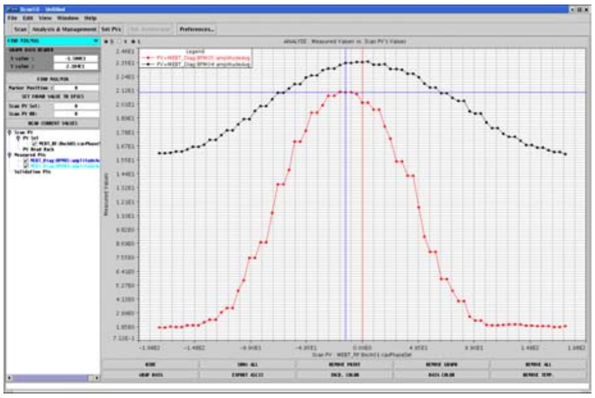


Fig. 7: The one-dimensional scan application display.

4.5 Online Model

The online model application provides mechanisms for user selection of the accelerator sequence, probe (i.e. model) type, and model data source. The data source can be the machine, the design values, or user supplied (useful for quick what-if analysis). Views of both the selected XAL device structure and the lattice structure (see Sect. 2.4) details are available, as well as the calculated beam trajectory and Twiss parameters. A screen shot of this application is shown in Fig. 8. The Twiss plot tab is visible in this image, but clicking on the other tabs presents views of different aspects of the modeling, from input to output.

4.6 Save-Compare-Restore (SCORE)

The Score application is an operator tool to aid in "snap-shotting" the state of the machine, comparing the machine state to a previous state, and restoring conditions to a previous state. This set of requirements is broader than for just beamline device parameters. This application is not limited to only settable quantities and also saves readback quantities. It uses a hierarchy of system / device types / and related settable and readback quantities. We use the DataTable classes described in section 2.3 to facilitate querying using this hierarchy. The screen image shown in Fig, 9 shows the live values displayed by the side of the saved values. Each system is displayed in a separate tab. It is possible to selectively pick combinations of systems and device types for restoring values.

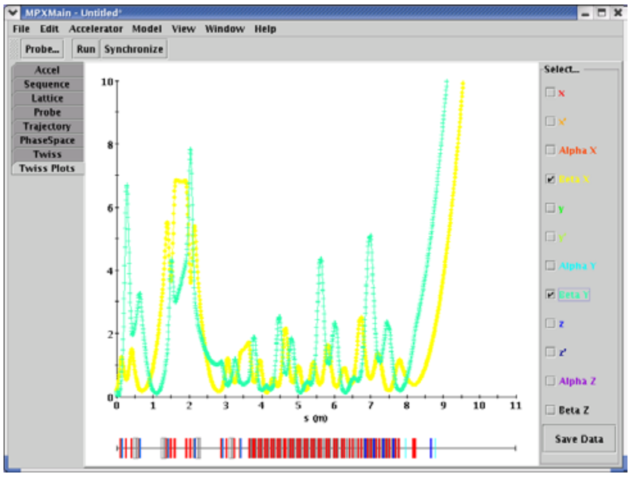
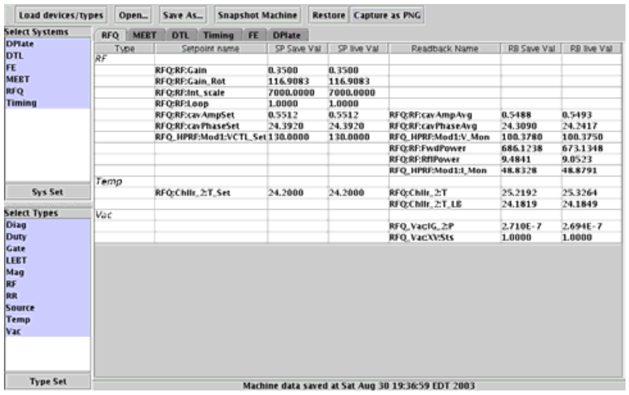


Fig. 8: Image of the online model application



Type	Setpoint name	SP Save Val	SP live Val	Readback Name	RB Save Val	RB live Val
RF						
	RFQ:RF:Gain	0.3500	0.3500			
	RFQ:RF:Gain_Rot	116.9083	116.9083			
	RFQ:RF:Int_scale	7000.0000	7000.0000			
	RFQ:RF:Loop	1.0000	1.0000			
	RFQ:RF:cavAmpSet	0.5512	0.5512	RFQ:RF:cavAmpAvg	0.5488	0.5493
	RFQ:RF:cavPhaseSet	24.3920	24.3920	RFQ:RF:cavPhaseAvg	24.3090	24.2417
	RFQ_HPRF:Mod1:VCTL_Set	130.0000	130.0000	RFQ_HPRF:Mod1:V_Mon	100.3780	100.3750
				RFQ:RF:FwdPower	686.1238	673.1348
				RFQ:RF:RflPower	9.4841	9.0523
				RFQ_HPRF:Mod1:I_Mon	48.8328	48.8791
Temp						
	RFQ:Chllr_2:T_Set	24.2000	24.2000	RFQ:Chllr_2:T	25.2192	25.3264
				RFQ:Chllr_2:T_LB	24.1819	24.1849
Vac						
				RFQ_Vac:IG_2:P	2.710E-7	2.694E-7
				RFQ_Vac:XV:Sts	1.0000	1.0000

Machine data saved at Sat Aug 30 19:36:59 EDT 2003

Fig. 9: Image of the SCORE (Save-Compare-Restore) application.

5 SUMMARY

The basic Java XAL structure is produced, and includes facilities for application programming. A set of initial applications is produced, and used at the initial SNS commissioning. The next development stages include populating the database and developing more applications specific to other beamlines that will be commissioned in the next several years. Also, in the framework area we are developing an agent based client/server capability.

6 REFERENCES

[1] http://www.aps.anl.gov/epics

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

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

[4] http://cosylab.com/

[5] C.K. Allen, et. al.," A Novel Online Simulator for High-Level Control Applications Requiring A Model Reference", these proceedings.

[6] http://www.jython.org/

[7] D. Purcell, et. all., Initial Experience with SNS Database Applications, these proceedings.