PROTOTYPE OF BEAM COMMISSIONING ENVIRONMENT AND ITS APPLICATIONS FOR NSLS-II*

G. Shen#, L. Yang, BNL, Upton, NY 11973, U.S.A.
M. Kraimer, ANL, Argonne, IL 60439, U.S.A.

Abstract

A software framework for commissioning the NSLS-II storage ring beam is in development. It adopts a client/server model, and consists of various servers for data acquisition, communication and management. Based on this structure, physics applications can be developed to satisfy the requirements of both day-1 beam commissioning and future beam study. This paper describes the status of the infrastructure development and its applications.

INTRODUCTION

NSLS-II (National Synchrotron Light Source II) is a 3rd generation synchrotron light source which is currently under construction [1]. It consists of a 200MeV linear accelerator, a 3 GeV booster, and a 3 GeV storage ring. Although based on well known techniques, its design has several novel features. For example, it uses DWs (Damping Wigglers) to reduce horizontal emittance and thus provide high flux X-rays. The design goals present some challenging control issues especially non-linear dynamics related issues [2] such as control of the Touschek lifetime and momentum aperture, control of tune footprint, control of impact of DWs and IDs (Insertion Device) including leading order nonlinear effects from DWs in the DA (Dynamic Aperture) optimizations, control of high order chromaticity, and so on. All such issues were addressed during the theoretical lattice design, but now the challenge is to provide a convenient and flexible software framework to address these issues during beam commissioning.

The high level applications developed by accelerator physicists should be able to achieve their goals by focusing on algorithms while being released from tedious data acquisition and manipulation issues. This is the design strategy for the software architecture. With a clean and carefully designed interface, collaborators, who have different areas of expertise such as GUI design, numerical analysis, accelerator physics, data acquisition, hardware control, and so on, can work together effectively and productively.

Many existing software frameworks, for beam commissioning, are available in our community but each exchanges data between layers via in-memory data structures, files, or methods. None of them can easily share data with others or directly include any portions of the other frameworks. For example, beam behaviour can be predicated by feeding a realistic on-line model to a simulation code. A method known as model based control can control beam behaviour effectively and dynamically by comparing real beam status with the predication. A traditional application interfaces directly to its build-in simulation code to predicate behaviour, and can only communicate with its build-in code. However, it is not sufficient to use one simulation code to solve all problems.

There have been many discussions about how to solve the problem of sharing code. Some frameworks such as MML [8] (Matlab Middle Layer) provide a set of APIs which makes it easier to development new high level applications. However, these APIs are not well-defined, or widely to use. For the NSLS-II project, a use case approach is being studied [3]. Instead of a traditional monolithic approach, a client-server based architecture has been designed and prototyped. The software environment for physics applications is based on EPICS, which is used for our hardware control [4]. Detailed design can be found in [5, 6], and the architecture is shown as Fig. 1.

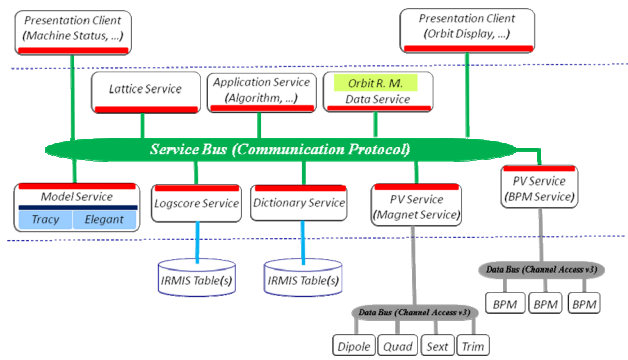

Figure 1: Architecture of Client/server based HLA.

Briefly, the system consists of: (1) the data source layer, which can be low level hardware control system, or a relational database; (2) a service layer, which provides services to gather data from the data source layer, and perform data manipulations such as constructing an orbit using BPM data; (3) the presentation layer, which present machine status to operators, and provides an interface for machine control

At NSLS-II, the following services have been prototyped: an on-line model service [7], a directory service, and a gather service. This paper discusses the high level application requirements for NSLS-II, describes the implementation of the directory and gather services, shows system integration with a use case, and gives a summary.

*Work performed under auspices of the U.S. Department of Energy under Contract No. DE-AC02-98CH10886 with Brookhaven Science Associates, LLC.
#shengb@bnl.gov

SOFTWARE REQUIREMENTS

The high level applications must satisfy all the requirements for beam commissioning and tuning. The applications are mainly involved with accelerator physics, mechanics, magnets, and vacuum. The required applications for the use case approach have been listed in [3].

The high level control environment can be separated to two parts: (1) a set of standalone applications whose algorithms are mature and can be used in daily operation; (2) an interactive environment which is suitable for scripting and for prototyping algorithms. A set of well-defined APIs, like but more clearly defined than MML, is required to support both the standalone applications and the interactive environment. The APIs must allow users to make full use of the functionality and APIs provided by lower level servers/libraries. Conceptually, a high level application would have the following layers: (1) Client APIs provide functions with physics meaning, such as "measureChromaticity", "rampMagnet", etc.. The API operates on element names, e.g. SL1, instead of the long EPICS channel names; (2) Server APIs are used to implement client APIs. Anything at this level or below is hidden from accelerator physicists who are only interested in "operating on beam" instead of "operating on magnets". The implementation of this and lower level are described in the following sections.

The monitoring software must provide overall status of the ring, and give warnings when a read-back differs from its set point by more than some threshold. The status includes, for example, current, lifetime, closed orbit, orbit feedback status, orbit stability, single shot beam position, magnet settings, vacuum, cryogenics system and RF. The software must be able to log all necessary machine settings data, and be able to retrieve the historical data for analysis.

The control part must provide software, either stand alone applications or scripts, to perform the following tasks: BPM (including turn by turn) testing and data analysis, orbit control and optimization, tune and chromaticity measurement and correction, the response matrix (Jacobian) for orbit, tune and chromaticity. Internal notes have been composed and active development is in progress.

A set of client APIs are grouped by prefix: measure, set, get, load, save, calculate and plot. For example, the APIs with prefix measure are for something that requires changing the hardware settings of a complex set of hardware in order to achieve a single measurement, while set APIs involve a simple setting on a single hardware device or a set of similar hardware. All can be used interactively without looking up a channel dictionary. The APIs can operate on magnet names directly but also provide some low level channel access methods.

SERVICE PROTOTYPES

The services are implemented and prototyped as JavaIOCs [9], which use PVData as the data container and PVAccess for network communication [10]. Currently, 3 services have been prototyped: model service [7], directory service, and gather service. The schematic for the directory and gather services is shown in Fig 2.

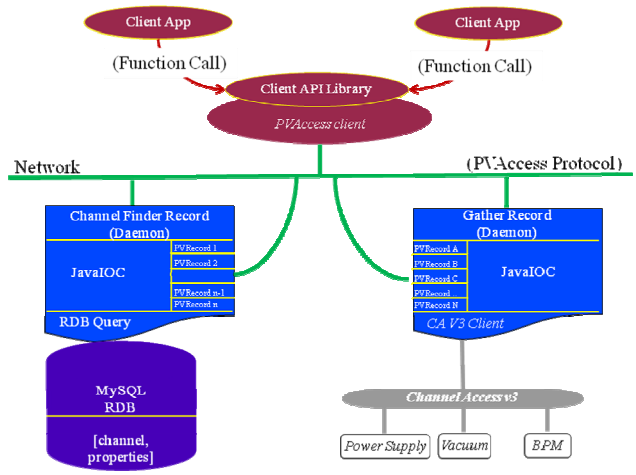

Figure 2: Architecture of 2 services.

The directory service provides an interface to get a list of PVs (Process Variable) and related properties such as position information. It uses a relational database, MySQL, to store all channels and their properties. A client application gives search constraints by calling a client API. The search command is passed to a daemon record which is shown as ChannelFinder Record in Fig 2. The record is processed inside the JavaIOC, and a RDB query is performed to get a channel name list with properties, which satisfied the search constrains. The value is shipped back to client through a dynamical created PVRecord. The detailed data flow is shown as Fig. 3.

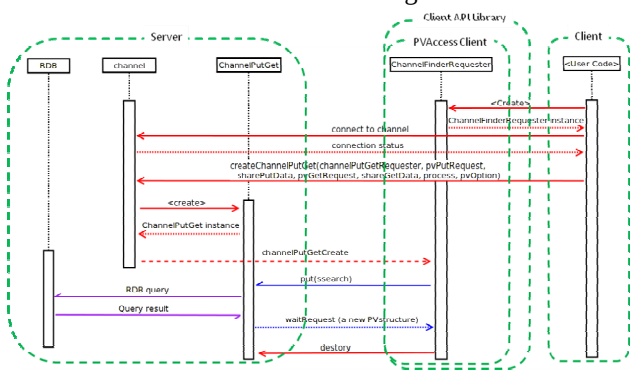

Figure 3: Data flow and API interface calling for directory service.

The interfaces to access the directory service are described as below:

- ChannelFinder. It is the interface for issuing a request. It adds a single method to what service client provides.
- sendRequest. It sends a request to server, and the argument is a search string.
- ChannelFinderRequester. It is the interface that must be implemented by the channelFinder client. It has the methods: (1) connectResult, which is called when waitConnect is called. (2) requestResult,

which is called after a sendRequest and waitRequest are called. It provides the results.

- RequestResult and ChannelProperty. These 2 classes provide the result.

After getting the PV list, the client can ship it to another service, for example the gather service. The gather is designed to accept a list of PVs, dynamically create and initialize a new PVRecord if it does not exist, connect to low level hardware IOCs for example BPM IOCs, and update its value every time a PV in a low level IOC changes. The data flow is similar to Fig. 3 and implementation logic is similar to the logic for the directory service.

A use case is to fetch an orbit from the BPM IOCs. Fig. 4 shows an example of a client using the above 2 services to get and display an orbit from the NSLS-II storage ring. The BPM IOC is simulated using a virtual accelerator, and the first horizontal corrector strength is set to 1e-5.

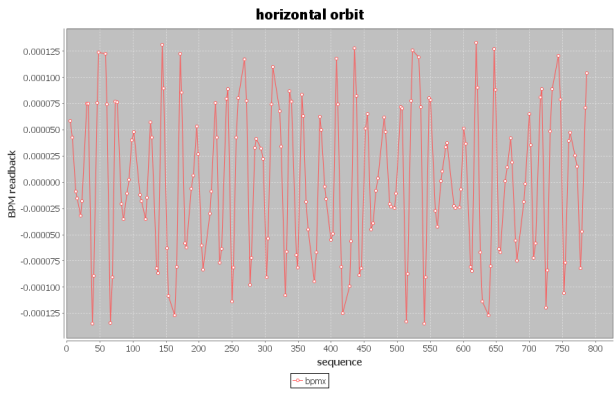

Figure 4: Orbit display using services.

The client does the following: Gets a PV list related to horizontal beam position and location along the storage ring from the directory service, feeds that list to the gather service to get horizontal orbit. Thus getting a PV list and connecting to a low level IOC is distributed to different services. The client can concentrate, for example, on how to design a GUI for the operator.

SUMMARY

To satisfy the requirement for NSLS-II beam commissioning and operation, a client/server based environment is under development. The system architecture and software requirements are briefly described. A 2 layer APIs architecture, which are for client and server respectively, is designed to satisfy the requirements. Two service servers, the directory service and the gather service are described. They are prototypes based on an open-source project, epics-pvdata.

PVData is used to store and access memory resident structured data. PVAccess is used to transfer PVData over the network. The data flow is described. Based on the directory and gather services, a client example is demonstrated that fetches an orbit and displays it on the presentation layer.

ACKNOWLEDGEMENT

The authors would like to thank Johan Bengtsson, Weiming Guo, and Donald Dohan for their helpful discussions and comments on the server development. They want to give their thanks to Ralph Lange for sharing his RDB query code to implement the directory service. They also want to express their thanks to Leo (Bob) Dalesio and Sam Krinsky for their continuous support and encouragement.

REFERENCES

[1] http://www.bnl.gov/nsls2/; NSLS-II Preliminary Design Report (2007).

[2] J. Bengtsson, "Design and Control of Ultra Low Emittance Light Sources", in the Proc. of ISCA09, San Francisco, USA, Aug, 2009

[3] J. Bengtsson, B. Dalesio, T. Shaftan, T. Tanabe, "NSLS-II: Model Based Control – A Use Case Approach", NSLS-II Tech Note 51 (2008).

[4] G. Carcassi, D. Dohan, G. B. Shen, L. R. Dalesio, N. Malitsky, Y. Tian, A. Ratti, L. R. Doolittle, "NSLS II Control System", in the Proc. of ICALEPCS09, TUP104, Kobe, Japan, Oct 2009

[5] G. Shen, "A Software Architecture for High Level Applications", Proc. of PAC09, Vancouver, Canada, 2009, FR5REP004

[6] G. Shen, "A Modular Environment for High Level Applications", Proc. of ICALEPCS 2009, Kobe, Japan, 2009, THP094

[7] G. Shen, M. Kraimer, P. Chu, J. Wu, "Design of Accelerator Online Simulator Server Using Structured Data", this proceeding, WEPEB024

[8] G. Portmann, J. Corbett, A. Terebilo, "Middle Layer Software Manual for Accelerator Physics," LSAP-302, 2005; J. Corbett, A. Terebilo, G. Portmann, "Accelerator Control Middle Layer," PAC 2003.

[9] http://epics-pvdata.sourceforge.net/; M. R. Kraimer, M. Sekoranja, "JavaIOC Status", talk on EPICS Meeting, Oct. 2009, Kobe Japan

[10] M. R. Kraimer, L. R. Dalesio, K. Zagar, M. Sekoranja, "Evolution of the EPICS Channel Access Protocol", in the Proc. of ICALEPCS 2009, Oct. 2009, Kobe, Japan, MOD005