NSLS-II HIGH LEVEL APPLICATION INFRASTRUCTURE AND CLIENT API DESIGN *

Guobao Shen#, Lingyun Yang, Kunal Shroff, BNL, Upton, NY 11973, U.S.A.

Abstract

The beam commissioning software framework of NSLS-II project adopts a client/server based architecture to replace the more traditional monolithic high level application approach. It is an open structure platform, and we try to provide a narrow API set for client application. With this narrow API, existing applications developed in different language under different architecture could be ported to our platform with small modification. This paper describes system infrastructure design, client API and system integration, and latest progress.

INTRODUCTION

As a new 3rd generation synchrotron light source with ultra low emittance, there are new requirements and challenges to control and manipulate the beam. A use case study [1] and a theoretical analysis [2] have been performed to clarify requirements and challenges to the high level applications (HLA) software environment.

To satisfy those requirements and challenges, adequate system architecture of the software framework is critical for beam commissioning, study and operation. The existing traditional approaches are self-consistent, and monolithic. Some of them have adopted a concept of middle layer to separate low level hardware processing from numerical algorithm computing, physics modelling, data manipulating, plotting, and error handling. However, none of the existing approaches can satisfy the requirement. A new design has been proposed by introducing service oriented architecture technology [3][4][5][6][7][8].

The HLA is combination of tools for accelerator physicists and operators, which is same as traditional approach. In NSLS-II, they include monitoring applications and control routines. Scripting environment is very important for the later part of HLA and both parts are designed based on a common set of APIs. Physicists and operators are users of these APIs, while control system engineers and a few accelerator physicists are the developers of these APIs.

With our Client/Server mode based approach, we leave how to retrieve information to the developers of APIs and how to use them to form a physics application to the users. For example, how the channels are related to magnet and what the current real-time setting of a magnet is in physics unit are the internals of APIs. Measuring chromaticities are the users of APIs. All the users of APIs are working with magnet and instrument names in a physics unit. The low level communications in current or voltage unit are minimized.

In this paper, we discussed our recent progress of our infrastructure development, and client API.

* Work supported under auspices of the U.S. Department of Energy under Contract No. DE-AC02-98CH10886 with Brookhaven Science Associates, LLC, and in part by the DOE Contract DE-AC02-76SF00515
#shengb@bnl.gov

SYSTEM INFRASTRUCTURE

The system architecture is shown as Fig. 1. As described in [6], it is a 3-tier architecture:

- Distributed front-end layer. This layer talks directly with physical device including magnet power supply, vacuum, RF, diagnostics system, and so on. The communication can use either existing Channel Access protocol, or pvAccess protocol. Migrating EPICS database to pvAccess is undergoing. A virtual accelerator could also locate at this layer, and provide same access for high level application development.
- Middle layer server layer. The middle layer service collects data from front-ends, and relational database such as IRMIS, organizes the data in predefined data structure, publish to its upper layer, and/or accepts data from its upper layer, and ships data to database or front end.
- Application layer. Physics application developed locates in this layer. Each application uses either Channel Access client or pvAccess to access middle layer servers and/or front ends. The application can be a scripting, matlab middle layer application, or a control system studio application. A clear API for client to access middle layer is under development [6].

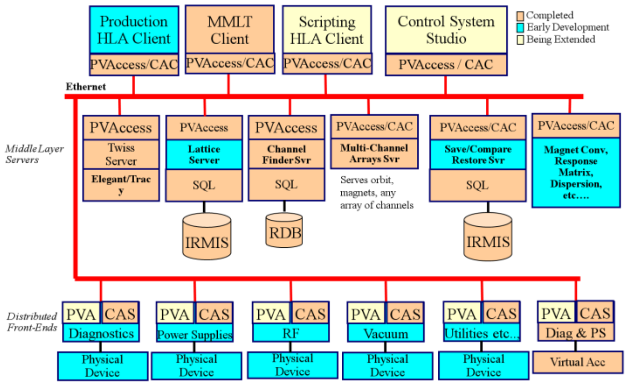

Figure 1: System Architecture.

Since EPICS control system is adopted for the NSLS II hardware control, it is nature to design the physics applications against EPICS system also. However, existing EPICS V3 cannot entirely satisfy the requirements as mentioned above. A new EPICS implementation, namely epics-pvdata [9], is under active development.

The epics-pvdata is an open source project, available from sourceforge. It consists of 4 modules:

- pvData, which defines and implements an efficient way to store, access, and transmit memory resident structured data;
- pvAccess [10], which is a new generation of EPICS communication protocol, and will be successor of existing Channel Access protocol. It is used to deliver data over the network, fully supports pvData, and depends only on module pvData;
- pvEngine/pvIOC, which is a processing engine. All behaviours are defined by pvEngine/pvIOC engine, and user has only to develop his own support for all desired behaviours. It depends on the pvData and pvAccess;
- pvService, which is a combination of all services under this project. All generic services or facility specified services should locate here.

The implementation of this project under Java is close to stable, and binding to other language such as C++ and/or Python is undergoing.

There are 3 services are implemented under epics-pvdata framework, and detailed implementation can be found in for example [6][7], and the benchmarking showed good performance [8][10][11].

Here we have to mention that for the dictionary service, there are 2 different approaches at NSLS II project. Different with the one implemented in epics-pvdata, a standalone ChannelFinder is implemented as a REST style web service [12]. Each entry consists of a channel name, an arbitrary set of properties (name-value pairs), and an arbitrary set of tags (names). An application sends an HTTP query to the service, specifying an expression that reference channel names, properties and their values, and/or tags. The service returns a list of matching channels with their properties and tags, as an XML or JSON document. It allows an authorized user to create, update, add properties or tags, and/or delete.

CLIENT API

As described above, the client API consists of 2 parts, internal API and user API respectively. Here we focus on the internal API development, and the user API will be formalized eventually during developing the internal API.

The 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.

ChannelFinder service creates a link between PVs and accelerator elements. A set of internal APIs are developed for ChannelFinder client in both Java and Python. With the Java API, CSS user has the capability to map an element physics name to EPICS pv name, and build his CSS application.

Since NSLS II chose Python as HLA scripting environment, the Python API enables user to benefit from the ChannelFinder service. For example, the channel finder will have two records for PV "SR:C30-MG:G02A{QDP:H1}Fld-RB" and "SR:C30-MG:G02A{QDP:H1}Fld-SP". One is for read back and the other is for setpoint of quadrupole "QH1G2C30A". The property of the read back PV would have handle="get", elementname="QH1G2C30A", cell="C30", girder="G2", elementtype="QUAD" and tagged "default". In a high level script, when calling get("QH1G2C30A"), the channel finder will be queryed to match elementname=="QH1G2C30A", handle="get" and tagged by "default". A similar group read can be achieved by calling get("QUAD") or get("C30") to read all quadrupoles or all elements in cell 30. This makes the high level scripts clear and short.

Wildcard matching on the element names can make group operation more convenient.

As an example, here is part of a script for dispersion measurement.

```
bpm = hla.getElements('P*C0[3-6]*')
s1 = hla.getLocations(bpm)
f0 = hla.getRfFrequency()
f = np.linspace(f0 - 1e-5, f0 + 1e-5, 5)

# avoid a bug in virtac
x0, y0 = hla.getOrbit(bpm)
time.sleep(4)

codx = np.zeros((len(f), len(bpm)), 'd')
cody = np.zeros((len(f), len(bpm)), 'd')
for i,f1 in enumerate(f):
    hla.putRfFrequency(f1)
    time.sleep(6)
    x1, y1 = hla.getOrbit(bpm)

    codx[i,:] = x1[:]
    cody[i,:] = y1[:]
```

From the demo code, we can see that there are no EPICS pv names used directly, and the function is realized by calling internal API such as getOrbit, getRfFrequency, and putRfFrequency.

The result is as shown in Figure 2. It was run against a virtual accelerator [3], which uses Tracy [13] simulation code as backend, and wraps the simulator into EPICS v3 control system. By changing the wildcard matching in "hla.getElements('P*C0[3-6]*')", we can measure dispersion at other location without changing more codes.

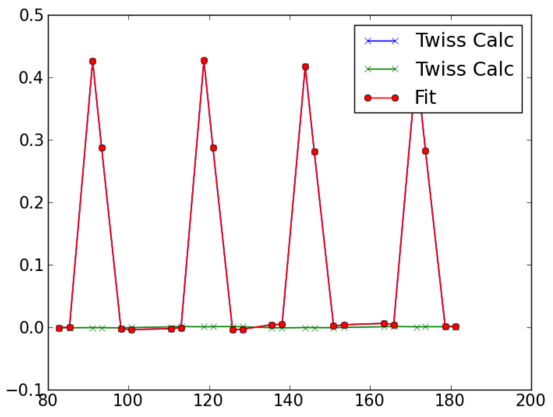

Figure 2: Dispersion Measurement Result.

SUMMARY

The proposed Client/Server mode based architecture for NSLS II HLA environment is under active development. There are some services have been prototyped. The development strategy for client API has been decided, and client application are starting to use the services.

ACKNOWLEDGEMENT

The authors would like to thank Matej Sekoranja at COSYLAB, and Marty Kraimer for their contributions on epics-pvdata development. They also want to thank Ralph Lange at Bessy for his contribution on web-service based channel finder development. They want to give their thanks to Leo Dalesio and Sam Krinsky at BNL for their continuous supports and encouragements.

REFERENCES

[1] J. Bengtsson *et al*, “NSLS-II: Model Based Control – A Use Case Approach”, NSLS-II Tech Note 51 (2008).

[2] J. Bengtsson, “Design and Control of Ultra Low Emittance Light Sources”, Proc. of ICAP09 (2009), TU3IOPK04, San Francisco, USA.

[3] G. Shen, “A Software Architecture for High Level Applications”, Proc. of PAC09 (2009), FR5REP004, Vancouver, Canada.

[4] G. Shen, “A Modular Environment for High Level Applications”, Proc. of ICALEPCS09 (2009), THP094, Kobe, Japan.

[5] P. Chu *et al*, “Service Oriented Architecture for High Level Applications”, Proc. of IPAC10 (2010), TUPEC072, Kyoto, Japan.

[6] G. Shen *et al*, “Prototype of Beam Commissioning Environment and its Applications for NSLS-II”, Proc. of IPAC10 (2010), WEPEB026, Kyoto, Japan.

[7] G. Shen *et al*, “A Novel Approach for Beam Commissioning Software using Service Oriented Architecture”, Proc. of PCaPAC10 (2010), WEPL037, Saskatoon, Canada.

[8] G. Shen *et al*, “Server Development for NSLS-II Physics Applications and Performance Analysis”, this Proc., MOP252.

[9] http://sourceforge.net/projects/epics-pvdata/

[10] G. Shen, “Performance Analysis of EPICS Channel Access and pvAccess”, NSLS-II Tech Note 082 (2010).

[11] G. Shen *et al*, “Services Development for NSLS-II Physics Application Environment using pvService”, EPICS Collaboration Meeting Fall 2010, BNL.

[12] http://channelfinder.sourceforge.net/ChannelFinder/

[13] J. Bengtsson, “TRACY-2 User’s Manual”, SLS Internal Document, February 1997; M. Böge, “Update on TRACY-2 Documentation”, SLS Internal Note, SLS-TME-TA-1999-0002, June 1999.