DESIGN OF ACCELERATOR ONLINE SIMULATOR SERVER USING STRUCTURED DATA*

G. Shen #, BNL, Upton, NY 11973, U.S.A.
M. Kraimer, ANL, Argonne, IL 60439, U.S.A.
P. Chu, J. Wu, SLAC, Menlo Park, CA 94025, U.S.A.

Abstract

Model based control plays an important role for a modern accelerator during beam commissioning, beam study, and even daily operation. With a realistic model, beam behaviour can be predicted and therefore effectively controlled. The approach used by most current high level application environments is to use a built-in simulation engine and feed a realistic model into that simulation engine. Instead of this traditional monolithic structure, a new approach using a client-server architecture is under development. An on-line simulator server is accessed via network accessible structured data. With this approach, a user can easily access multiple simulation codes. This paper describes the design, implementation, and current status of PVData, which defines the structured data, and PVAccess, which provides network access to the structured data.

INTRODUCTION

For a modern accelerator, model based control (MBC) plays an important role during beam commissioning, beam study, and daily operation. The MBC provides an efficient approach for establishing a bridge between a real accelerator and a theoretical machine. Model codes are the engine for beam computation. A Model Engine provides the support infrastructure. It prepares model input parameters, sets up a model run and packages the run results. By feeding it a realistic model, a Model Engine can predict beam behaviour. With the predicted behaviour, an analysis can then be performed, and a control algorithm can be applied to effectively control the beam behaviour. With an on-line model, the beam can be controlled dynamically, and misbehaved beam can be corrected in real-time.

The approach used by most current high level application environments is to use a built-in simulation engine and feed a realistic model into that simulation engine. The data transfer between application and simulation engine is through either an in-memory data structure or an internal file as shown in Fig. 1. With this architecture, an application is tightly coupled with its simulation code making it difficult to call other simulation codes which are not supported by the applications High-level Application (HLA) environment. However, there are many accelerator modelling simulation codes;each one has strengths but none solves all problems. Thus it is not sufficient to use only one simulation code.

*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

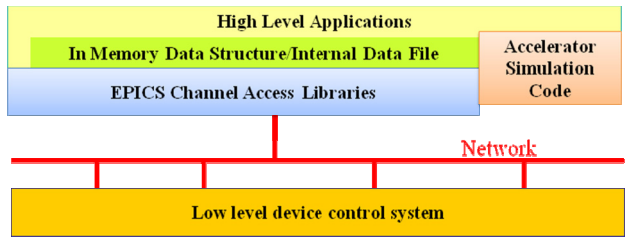

Figure 1: Architecture of traditional HLA environment.

A solution to this problem is to provide a platform to host multiple modelling tools so that one can easily switch among the codes. In order to achieve such a platform, a set of common physics data structures is required. Additionally, the software infrastructure for the model platform should be extremely robust. This paper describes the design and implementation, and the latest status.

MODEL SERVICE DESIGN

System Architecture

The client-server architecture [1, 2] is shown in Fig. 2.

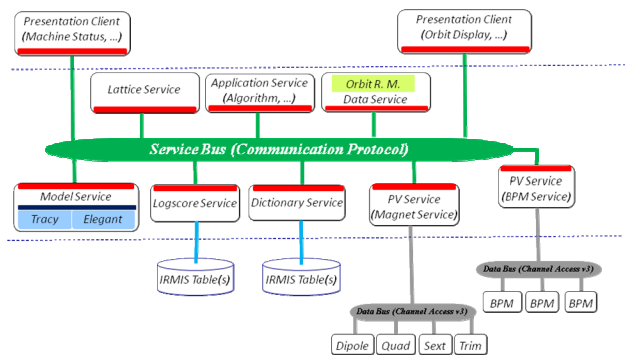

Figure 2: Client-server based architecture.

The environment is a 3-tier architecture:

- Data source layer. It accesses live data from hardware control system (EPICS system for example), or provides data which is stored in relational database;
- Service layer. It provides an extensible set of services mostly for easy data communication;
- Presentation layer. It provides a user graphic interface.

Model Service Architecture

As shown in Fig. 2, instead of coupling with the application, the on-line simulation code is hosted by a model service, which runs as a standalone server. Data transfer is via the network using a pre-defined data

Work supported by US Department of Energy contract DE-AC02-76SF00515.

structure as shown in Fig. 3. A client application can switch from one simulation code to another seamlessly and transparently. Detailed design can be found in [8].

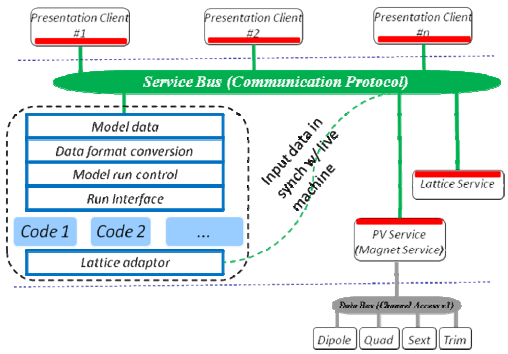

Figure 3: Model service architecture.

DATA STRUCTURE

The server is prototyped using structured data defined by PVData (Process Variable Data) [3] which was initially part of an Eclipse project named JavaIOC [4]. It is now a separate open-source project named epics-pvdata [3] which is hosted by SourceForge. Epics-pvdata consists of many modules. The model server uses modules PVData, PVAccess, and JavaIOC.

PVData defines and implements an efficient way to store, access, and transmit memory resident structured data. The basic data types defined by PVData are:

- Scalar. A scalar can be one of the followings: boolean, byte, short, integer, long, float, double, and string;
- Structure. A structure is an ordered set of fields where each field has a name and a corresponding type.
- Array. An array is a one-dimensional array with the element type being either a scalar or a structure;

Since a field can have type structure, complex structures are supported. No other types are needed since structures can be defined for any particular simulation data types.

PVData is memory resident data stored in a PVDatabase (Process Variable Database). A PVDatabase has the following features:

- A database has records (PVRecords). Each PVRecord has a unique record name, and holds a top level PVStructure.
- A PVStructure is a structured set of PVFields.
- A PVField contains data, and the data type can be one of the above 3 types. Each PVField has support code for accessing individual piece of data and for introspection.

Fig. 4 shows an example of creating records via the XML format defined by PVData. Two PVRecords are shown in Fig. 4, "simulationResult" and "SH1:INDX:0004". The model data in Fig. 3 is contained in the "simulationResult" record in an array format. Each PVField can be accessed by connecting the record name and a field with a ".". For example, to get a tune value, user can fetch it as "simulationResult.tune".

```
<database>
 <import name="org.epics.ioc.*"/>
 <import name="org.epics.pvData.*"/>
 <record recordName="simulationResult"
          extends="org.epics.pvService.modelService.modelServerBase">
  <structure extends="alarm" name="alarm"/>
  <structure extends="timeStamp" name="timsStamp"/>
  <scalar name="model" scalarType="string" immutable="false">elegant</scalar>
  <array name="elemName" scalarType="string" immutable="false"/>
  <array name="s" scalarType="double" immutable="false"/>
  <array name="alphax" scalarType="double" immutable="false"/>
  <array name="betax" scalarType="double" immutable="false"/>
  <array name="nux" scalarType="double" immutable="false"/>
  <array name="etax" scalarType="double" immutable="false"/>
  <array name="etapx" scalarType="double" immutable="false"/>
  <array name="alphay" scalarType="double" immutable="false"/>
  <array name="betay" scalarType="double" immutable="false"/>
  <array name="nuy" scalarType="double" immutable="false"/>
  <array name="etay" scalarType="double" immutable="false"/>
  <array name="etapy" scalarType="double" immutable="false"/>
  <array name="orbitx" scalarType="double" immutable="false"/>
  <array name="orbitpx" scalarType="double" immutable="false"/>
  <array name="orbity" scalarType="double" immutable="false"/>
  <array name="orbitpy" scalarType="double" immutable="false"/>
  <array name="tune" scalarType="double" immutable="false"/>
 </record>
 <record recordName="SH1:INDX:0004"
          extends="org.epics.pvService.modelService.element">
  <structure extends="alarm" name="alarm"/>
  <structure extends="timeStamp" name="timsStamp"/>
  <structure name="attribute">
   <scalar name="index" scalarType="int" immutable="true">4</scalar>
   <scalar name="type" scalarType="string" immutable="true">MULT</scalar>
   <scalar name="order" scalarType="int" immutable="true">3</scalar>
   <scalar name="b3" scalarType="double" immutable="false">9.9954</scalar>
  </structure>
 </record>
 .....
</database>
```

Figure 4: Design a PVDatabase using PVData.

The supporting code for "simulationResult" extends to "org.epics.pvService.modelService.modelServerBase" structure, which is defined in Fig. 5. The user processing code is attached for defining this structure.

```
<database>
 <package name = "org.epics.pvService.modelService" />
 <structure structureName = "modelServerBaseFactory">
  <scalar name = "supportFactory" scalarType = "string">
     org.epics.pvService.modelService.impl.ModelServerFactory
  </scalar>
 </structure>
 <structure structureName = "modelServerBase">
  <auxInfo name = "supportFactory" scalarType = "string">
     org.epics.pvService.modelService.modelServerBaseFactory
  </auxInfo>
 </structure>
 <structure structureName = "elementFactory">
  <scalar name = "supportFactory" scalarType = "string">
     org.epics.pvService.modelService.impl.ElementFactory
  </scalar>
 </structure>
 <structure structureName = "element">
  <auxInfo name = "supportFactory" scalarType = "string">
     org.epics.pvService.modelService.elementFactory
  </auxInfo>
 </structure>
 </database>
```

Figure 5: Structure supporting code.

COMMUNICATION PROTOCOL

A new generation of EPICS Channel Access protocol, PVAccess [5], is used to deliver data over the network. PVAccess fully supports PVData, and depends only on project PVData.

Two mechanisms are provided for accessing PVData: (1) “local”, which provides access to the local PVDatabase without using the network, and (2) “pvAccess”, which implements network access between the client and server.

Some major functions are described as below:

- An application can implement a server which provides full support for PVData by starting a local PVDatabase.
- Monitoring is defined by project PVData, which supports an extensible set of monitoring algorithms. Server developers can design their own monitor algorithms, or use the default set. PVAccess provides the data access method.
- A query facility is supported. A client can ask a network query to be made.
- A gateway between PVAccess and other systems. For example the JavaIOC implements caV3, which provides a gateway between EPICS Channel Access and PVAccess.

Fig. 6 shows an example data flow using PVAccess to get data from a PVRecord.

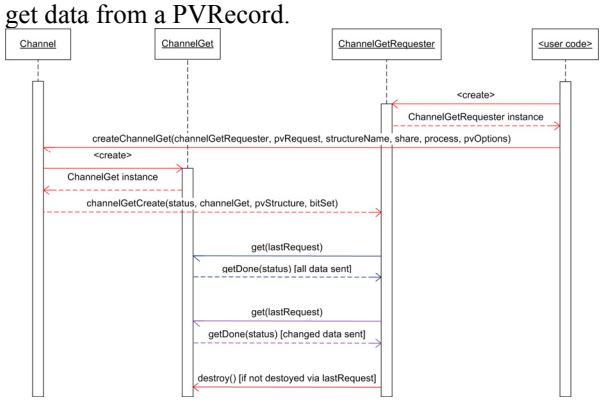

Figure 6: PVAccess Data Flow.

RESULT

Currently, the model servers are implemented to support the Tracy-3 [6] and Elegant [7] simulation codes as shown in Fig. 7.

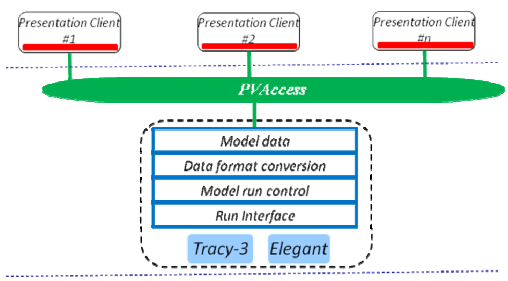

Figure 7: Implementation for model service.

A user can switch between those 2 models using setting “simulationResult.model” as shown in Fig.4, which uses Tracy-3 if the attribute for the name *model* is set to “tracy” or Elegant if it is set to “elegant”. Fig. 8 shows a horizontal betatron function plot from the model server on a presentation client.

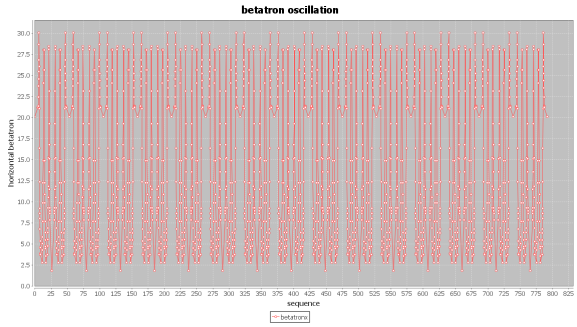

Figure 8: Beta function plotting.

DISCUSSION AND SUMMARY

An on-line model server is designed and implemented using PVData as the data container and PVAccess as the communication protocol. Work remains before the model server is ready for use. Remaining tasks include: concurrent multi-user access support, system robustness, and performance benchmark. Nevertheless, the current prototype provides a way to host multi-simulation codes, and access them transparently. Currently, Tracy-3 and Elegant codes are supported and demonstrated.

ACKNOWLEDGEMENT

The authors would like to thank Johan Bengtsson, Weiming Guo, and Donald Dohan at BNL and Ji Qiang at LBNL for their helpful discussions and comments on the model server development. They also want to thank all developers at COSYLAB, especially Matej Sekoranja, for their contributions on the epics-pvdata project. They want to express their thanks to Leo (Bob) Dalesio for his continuous support and encouragement.

REFERENCES

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

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

[3] http://epics-pvdata.sourceforge.net/

[4] M. R. Kraimer, M. Sekoranja, “JavaIOC Status”, talk on EPICS Meeting, Oct. 2009, Kobe Japan

[5] 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

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

[7] M. Borland, “elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation,” APS LS-287, 2000.

[8] “Generic Model Host System Design”, P. Chu, J. Wu, G. Shen, J. Qiang, these proceedings, TUPEC071