A NOVEL APPROACH FOR BEAM COMMISSIONING SOFTWARE USING SERVICE ORIENTED ARCHITECTURE*

G. Shen, BNL, Upton, NY 11973, U.S.A.
P. Chu, J. Wu, SLAC, Menlo Park, CA 94025, U.S.A.

Abstract

A novel software framework is under development, which is for accelerator beam commissioning and operation. It adopts a client/server based architecture to replace the more traditional monolithic high level application approach. A minimum set of commissioning and operational services has been defined such as simulation server service, directory service, magnet service, and bpm service, etc. Most of them have been prototyped. Services can use EPICS pvData as its data container and pvAccess as communication protocol. This paper describes conceptual design and latest progress for some services.

INTRODUCTION

Traditionally, an accelerator application needs to deal with many functions such as connection to various signals, data from physics modelling, data plotting, complicated program flow and error handling. If all such computation is built in a single standalone program, the complexity level of the program may result poor performance, unreliability and code maintenance difficulty. Also, if any application needs a new feature which is not provided by an easy interface, it is hard to implement the feature without major restructure of the existing program.

On the other hand, if heavy computation functions can be distributed as running modules residing on various servers and serving up data via proper service protocol, the Graphical User Interface (GUI) application itself can be a simple thin client receiving the data from the servers. This service oriented architecture (SOA) approach can in general improve both performance and reliability of applications.

In this paper, some preliminary result for simulation or model service, Linac energy management (LEM) service and possible communication protocols such as EPICS pvAccess are reported. Work plan for the SOA is also described.

SERVICE ORIENTED ARCHITECTURE

One can identify some essential services for accelerator operation by surveying the functionalities of existing applications. The granularity of services depends on functionality shared by clients, performance, robustness coding complexity, and maintenance. On one hand, too narrow of a service means many more services in total and could cause maintenance trouble. On the other hand, a single service providing too many functions could reduce its performance and reliability. Figure 1 shows a typical top level SOA diagram with a few services.

Furthermore, services can be distributed to multiple servers with virtual machines technology. A distributed system can avoid one service bringing down others. One can also add a redundant server for any critical services.

Advantages for SOA approach are described in detail below.

Easy Application Development

Coding an application with many functions can be tedious. On the other hand, some functions can be shared by several applications. A well-designed SOA approach can greatly reduce the burden on end developers. Applications can then become "thin" clients without much inline computation. Only simple "get/set" data communication with the service providers will be needed. Coding up a complicated application such as controlling an experiment will require much less time and effort. Yet, all the high quality of supporting functionality is fulfilled because the complication is maintained on the server side. This means that even a program written in scripting language such as Matlab script can still have the same high quality of error handling and message logging without additional coding efforts.

Data Control

Because the services are centralized control, i.e. typically only one particular service instance running at a time. This approach can avoid conflict among multiple clients accessing the same device; for instance, feedback and Linac Energy Management (LEM) program might change the same corrector at the same time but magnet server can shedule the two requests properly.

Better Application Memory Management

For individual applications, SOA can avoid large memory and CPU consumption due to heavy computation and data process. Therefore, it can also reduce the chance of client application program crashing.

Service Swappable

It is not necessary to replace all traditional functions with services overnight. One can implement a service at a time. If an old service is replaced by a new one, the application programming interface (API) should remain the same so the client application can pick up the service seeminglessly. This also means the SOA work is highly scalable depending on the available resources. Furthermore, a new service should go through rigorus test before any client application in production can actually use it.

*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

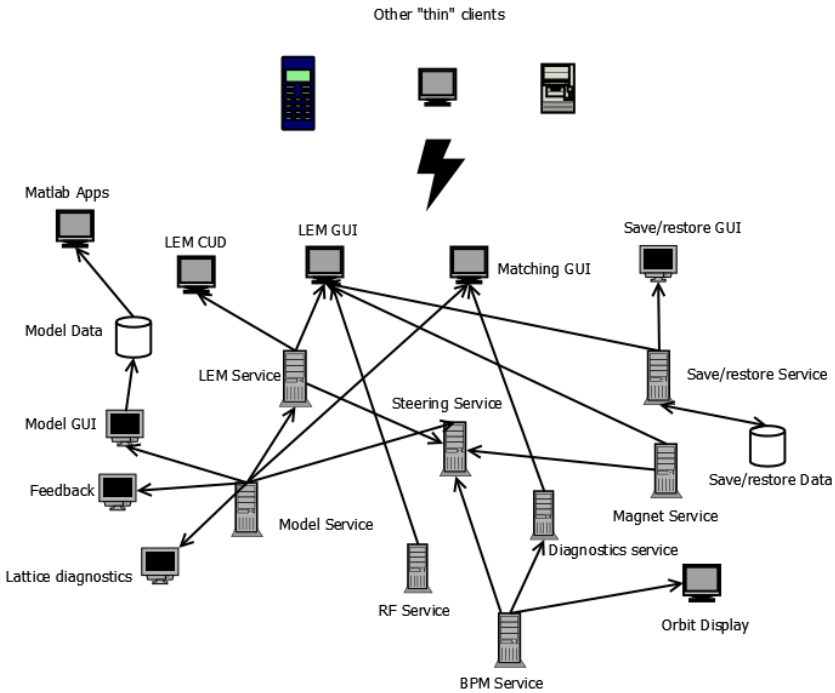

Figure 1: Top level SOA functional diagram. The arrow direction shown in the figure indicates the data flow direction. For instance, Model Service can provide model data to Linac Energy Management (LEM) Service.

SERVICE EXAMPLES

Simulation (Model) Service

Running model within an application is one of the most expensive operations in terms of CPU and memory use. Simulation or model service runs physics model periodically and makes up-to-date model data available for any subscribed clients.

The model server can be expended to cover not only online modelling but also other beam dynamics modelling such as start-to-end simulation, which can provide more detailed beam dynamics simulation information, with a set of uniform APIs. Various simulation codes can be run continuously to supply data to the model server with extant hardware set values.

Figure 2 shows a schematic diagram for the Simulation Service. The core part of the service is a model run control program which manages input data and file preparation, job submission, run status monitoring, run forced quit and output data management.

A prototyped run control program with Fortran based IMPACT-T [1] modelling code using Java and Python has been written and under test. Java part of the program is mainly for data display such as tables and plots while Python is excellent for file I/O and communication with the modelling code and the underneath operating system. The run control program dynamically generates a set of IMPACT-T input files based on user's input via GUI. For each run, a new directory named with the run start time is created and all files are saved under the directory.

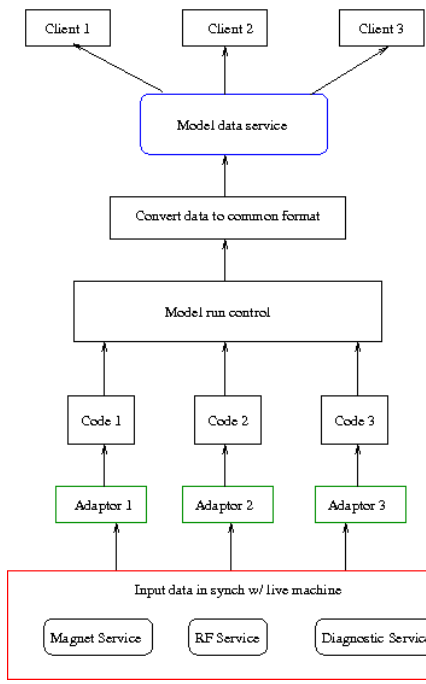

Figure 2: Data flow for model engine and service.

Linac Energy Management (LEM) Service

Any linear accelerator can change its energy from time to time. In order to maintain the same lattice all the time, a program so called LEM which continuously updates the energy information has to run regularly. LEM requires RF data and model tracking; therefore, it is most efficient that it is running periodically on a server and updating all data for clients such as LEM application and control room continuous update display (CUD).

A prototype LEM Service posting calculation result on EPICS Process Variables (PV) has been implemented. Preliminary result shows that the service has been running for over a month even with the accelerator itself being up and down. In contrast, the standalone version of the LEM program crashes easily due to various causes such as data acquisition failure, memory management issue and so on.

Directory Service

This service is prototyped under a sourceforge project so-called epics-pvdata [3,4]. The epics-pvdata consists of 4 modules: (1) pvData, which defines and implements an efficient way to store, access, and transmit memory resident structured data; (2) pvAccess, which is a new generation of EPICS Channel Access protocol. It is used to deliver data over the network and fully supports pvData, and depends only on module pvData; (3) javaIOC, which is a processing engine. All behaviours are defined by JavaIOC engine, and user has only to develop his own support for all desired behaviours. It depends on the pvData and pvAccess; (4) pvService, which is a combination of all services under this project. All generic services or facility specified services should locate here.

The Directory Service, so-called itemFinder, is one particular example under pvService module. It provides a basic function to get a list of physics elements and its associated properties such as EPICS PV names for read-back, set-point, temperature, and so on if they apply. It is designed and prototyped against MySQL relational database (RDB). The RDB schema consists of two (2) tables: (1) *item* table, which stores the physics names for all elements installed in a facility; (2) *property* table, which stores all properties associated with each element.

A client application gives search criteria by calling a client API. The search command is passed to a daemon record and the record is processed inside the JavaIOC, and a RDB query is performed to get an item name list with properties, which satisfied the search constrains. The value is returned back to the client through a dynamically created pvRecord.

One use case of this service is to get a list of EPICS channel names. Since a channel name is an entry of properties for an element, by getting the list back to client, user can retrieve the element's channel names easily.

Gather Service

The Gather Service is another service under pvService module. Basic idea of this service is that a client sends a PV list with a string to this service; the service then creates a pvRecord dynamically with the string name given by the client.

Here we have to mention that the type of each PV in the PV list should have same data type, and pvService does not check it. Also the client has to make sure that name string is unique and did not exist in the Gather Service. Otherwise, it will use existing pvRecord instead of creating a new one. This has to be improved later.

After a client ships a PV list to the gather service, the gather service creates a pvRecord as mentioned above, and connects to low level hardware IOCs for example BPM IOCs, and update its value every time a PV in a low level IOC changes.

Client can customize the Gather as desired service such as a BPM orbit server, or a magnet server.

COMMUNICATION PROTOCOL

An adequate communication protocol is indispensable for SOA architecture. There are many protocols available such as HTTP, XML-RPC and so on. A new generation of EPICS Channel Access protocol, pvAccess, is a better option to deliver accelerator data over the network. The main advantages are as below:

- It fully supports pvData, and depends only on project pvData. We can integrate our servers seamlessly with pvData.
- It is developed against current Channel Access, and inherits the advantages of EPICS Channel Access. For example, it is data stream oriented protocol, and can be expected to have good performance for an accelerator control system.

The performance benchmarking is undergoing, and a preliminary result shows a good performance. For example, on a local office network, when we feed 1000 PVs to the Gather Service, it can update the 1000 PVs' value with a frequency large than 100Hz.

PLAN

Some service such as Simulation Service, itemFinder, and gather service are being prototyped. They all are in the stage of choosing a good communication protocol for production and EPICS pvAccess shows a good performance as communication protocol. Some more development and benchmarking are necessary for a production server.

ACKNOWLEDGEMENT

The authors would like to thank Matej Sekoranja at COSYLAB and Marty Kraimer for their contributions on epics-pvdata development. They also want express their thanks to Ji Qiang at LBNL for providing IMPACT-T code. They want to give their thanks to Leo Dalesio at BNL for his continuous support and encouragement.

REFERENCES

[1] J. Qiang, S. Lidia, R. D. Ryne, and C. Limborg-Deprey, "A Three-Dimensional Quasi-Static Model for High Brightness Beam Dynamics simulation", Phys. Rev. ST Accel. Beams **9**, 044204 (2006).

[2] P. Chu *et al*, "Generic Model Host System Design", Proc. of IPAC10, TUPEC071

[3] G. Shen *et al*, "Design of Accelerator Online Simulator Server Using Structured Data", Proc. of IPAC10, WEPEB024

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