SERVICE ORIENTED ARCHITECTURE FOR HIGH LEVEL APPLICATIONS*

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

Abstract

Standalone high level applications often suffer from poor performance and reliability due to lengthy initialization, heavy computation and rapid graphical update. Service-oriented architecture (SOA) is trying to separate the initialization and computation from applications and to distribute such work to various service providers. Heavy computation such as beam tracking will be done periodically on a dedicated server and data will be available to client applications at all time. Industrial standard service architecture can help to improve the performance, reliability and maintainability of the service. Robustness will also be improved by reducing the complexity of individual client applications.

INTRODUCTION

At several accelerator facilities, some Java based standalone applications have suffered from poor performance and reliability. The major reasons for the setback are that the applications have to prepare large amount of data initialization, perform heavy computation and communication with underneath control system in real time, and update high rep rate Graphical User Interface (GUI). Also, if any application needs a new function provided by a module not currently called by the app, it may have to implement the function and, therefore, further complicate the program flow.

Inspired by the previous control system at SLAC, SLC (Stanford Linear Collider) Control Program or SCP, one way to solve the performance and reliability problems is to implement a Service-Oriented Architecture (SOA) [1-2] for high level applications. Because SCP contains all data in a monolithic memory, applications are able to react to any requests quickly. Different from the SCP architecture, the new control system is highly distributed and, therefore, supporting functions should be residing on various servers. A highly distributed system can also reduce the risk of potential single point of failure.

SERVICE ORIENTED ARCHITECTURE

Fig.1 shows a top level SOA diagram with a few services as an example.

Services should be distributed to multiple servers which can be virtual machines on a physical server. A distributed system can avoid single point of failure. One can also add a redundant service provider for any critical services.

As mentioned above, coding an application with many functionalities can be tedious. On the other hand, a well-designed SOA design can greatly reduce the burden on end developers. Applications then become "thin" clients without much computation involved. Application development can, therefore, be greatly facilitated. Coding up a complicated experiment application 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 Matlab script accessing services can have the same high quality of error handling and message logging.

Other advantages of the SOA approach include:

- 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.
- 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.
- It is not necessary to replace everything 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 need not to be changed. This also means the SOA work is highly scalable depending on the available resources. Furthermore, a new service should go through stress test before any client application in production can actually use it.
- If the service protocols chosen are complied with industrial standard, web and mobile support may be included.

SERVICE PROVIDERS

As shown in Fig.1, some services are identified as examples. 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.

Below are a few services we would like to implement or study.

*Work supported in part by the DOE Contract DE-AC02-76SF00515.
#pchu@slac.stanford.edu

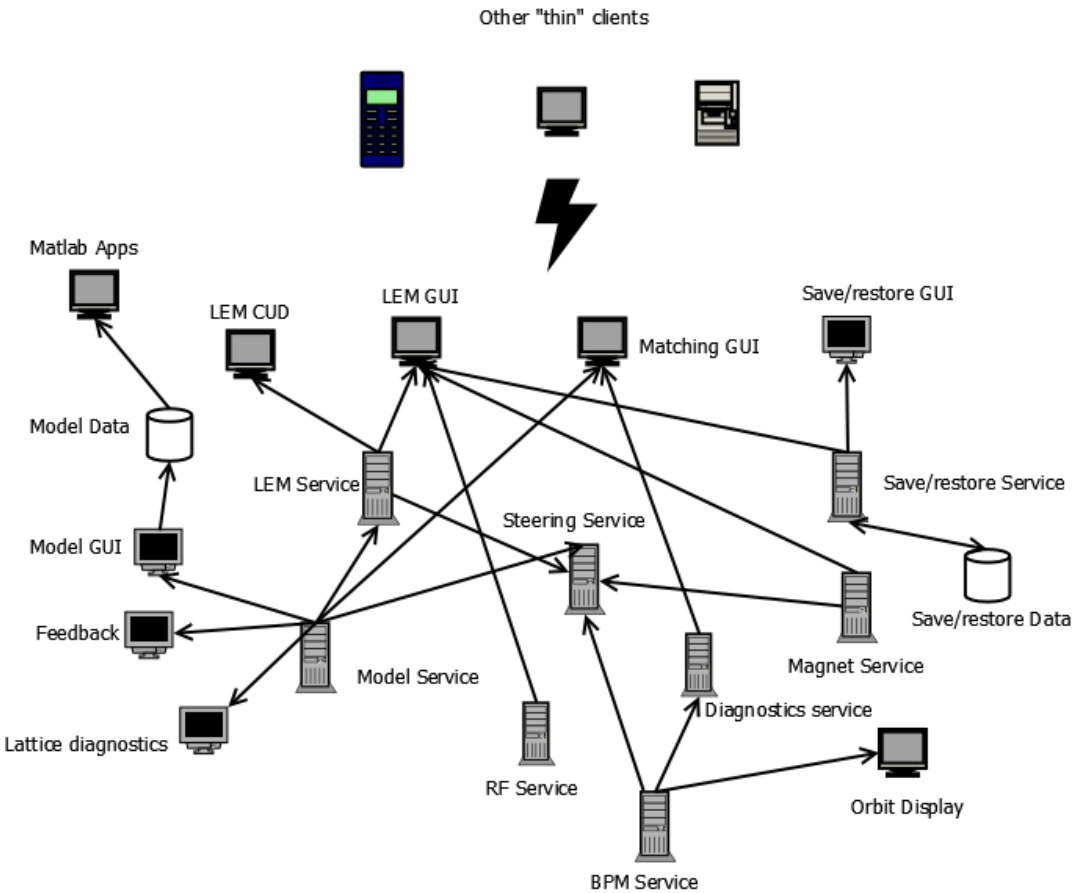

Figure 1: Top level SOA functional diagram. The arrow direction shown in the figure indicates the data flow direction. For instance, Model Service can provide model data to Linac Energy Management (LEM) Service.

Magnet Service

Magnet service handles all magnets' EPICS Process Variables (PV) for reading, setting and monitoring. Client applications do not have to worry about EPICS Channel nor any other control system underneath. Features for the magnet service include:

- Updating all prescribed magnets' *set* values and *read-back* values with monitoring.
- Providing magnet statuses such as active, various warnings, offline or inactive. Setting magnets should follow the status for proper action.
- Unit and magnet name conversion between physics model and control system.
- Handling out-of-tolerance exception when trying to set a magnet.
  - Knowing how to gracefully roll back magnet settings if any one fails to set.
  - Reporting problems to messaging and logging systems for offline debugging and program diagnosis.
- Providing other magnet attributes such as location, polarity, multipoles, aperture size and any other magnet measurement properties.
- Handing many independent signals in parallel for best performance.
- Providing software time correlation if hardware time correlation not available for a pulsed machine.

RF Service

RF Service is very similar to Magnet Service but dedicated to Klystron/RF signals. For Linac Coherent Light Source (LCLS), part of the klystron control is still relying on SLC control system. The hybrid RF system creates complication for applications which would like to access the RF signals. Furthermore, actual RF field amplitude and phase are calculated against beam reference energies which should be updated regularly.

Model Service

Model service runs online model periodically and makes up-to-date model data available for any subscribed clients. This model server can be expended to cover not only online model but also other beam dynamics

modelling codes such as start-to-end simulation with a set of uniform API. Details for the model server will be described in another paper [3].

Linac Energy Management Service

For example, LCLS Linac klystron complement can change constantly; in order to maintain the same lattice all the time, a LEM program 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).

Save/Compare/Restore Service

Save, compare and restore (SCORE) utility has to acquire data from many signals. It is very expensive to connect to all those channels. If many SCORE instances are running concurrently, the performance would suffer from such overwhelmingly high volume of network traffic. A service can connect to all necessary signals and monitor them all the time. Whenever a SCORE snapshot is requested, all monitored data can be saved immediately. Some detail requirements for the SCORE Service are:

- Signals can be grouped in SCORE and set different update rate.
- The backend storage, typically a relational database (RDB), connection and access can be optimized.
- Use cases for this service: reference orbit snapshot, online model replay.

Orbit or BPM Service

Orbit service collects all the Beam Position Monitors (BPM) data and can be buffered for clients to consume. The data from Orbit Service can be used for orbit display and beam steering.

Steering or Orbit Correction Service

Steering Service should execute orbit optimization algorithm continuously to provide orbit correction solution. The Steering Service is also a consumer of the Model Service which takes beam tracking data for orbit calculation. The service can also be a backend for transverse feedback system.

Start-to-end Simulation Service

A Start-to-end Simulation Service can provide convenient solution for any applications which need such detail beam dynamics computation without going through tedious model runs. Various simulation codes can be run continuously to supply data to the model server. The run-control program residing on the server should be robust and easy to use for physicists.

POSSIBLE SOA TECHNOLOGIES

There are many commercially available SOA solutions. Communication protocols such as XAL-RPC, JSON-RPC, Java Message Service (JMS) and EPICS PVData [5] are under consideration. Commercial packages such as Oracle Application Server are powerful but expensive. We would like to prototype a few of the mentioned technologies. The final choice might not be limited to a single technology. For the level of data complexity, there might be different optimal solutions.

CONCLUSION

Presently, several services such as Model Service, LEM Service and SCORE Service are under prototype. As the first services are stable enough, they will be deployed to production. Once we gain some experience, new service development will be facilitated.

REFERENCES

[1] http://en.wikipedia.org/wiki/Service-oriented_architecture
[2] Nicolai M. Josuttis, "SOA in Practice: The Art of Distributed System Design", O'Reilly Media, Inc., 2007.
[3] P. Chu *et al*, "Generic Model Host System Design", these proceedings.
[4] P. Chu *et al*, "Linac Energy Management for LCLS", these proceedings.
[5] EPICS PVData reference.