ACCELERATOR LATTICE AND MODEL SERVICES*

P. Chu#, MSU, East Lansing, MI 48824, USA
H. Lv, F. Guo, C. Wang, Z. Zhao, IHEP, Beijing, China
G. Shen, BNL, Upton, NY 11973, USA

Abstract

Physics model based beam tuning applications are essential for complex accelerators. Traditionally, such applications acquire lattice data directly from a persistent data source and then carry out model computation within the applications. However, this approach often suffers from poor performance and modelling tool limitation. A better architecture is to offload heavy database query and model computation from the application instances. A database has been designed for hosting lattice and physics modelling data while a set of web based services then provide lattice and model data for the beam tuning applications to consume. Preliminary lattice and model services are based on standard J2EE Glassfish platform with MySQL database as backend data storage. Such lattice and model services can greatly improve the performance and reliability of physics applications.

INTRODUCTION

Most beam tuning applications are based on beam simulations; therefore, it is essential for these applications to get the simulation data. It is straightforward to embed beam simulation operation within the applications while during prototyping stage. However, online beam simulation typically requires heavy computing and other supporting utilities such as lattice settings from database and real-time data from control systems. As the program development progressed with more features, its complexity level can be dramatically increased. On the other hand, the same model simulation result may be applied to multiple application instances or even across different applications.

To simplify the application codes and to increase data reusability, Service-Oriented Architecture (SOA) is introduced. With lattice and model services providing needed data as opposed to the application carrying out the computation, applications as clients can consume the data and perform their functionalities more efficiently. Furthermore, lattice and model services shield clients from various data sources such as multiple databases or other services. An existing model server example is AIDA [1] implemented at SLAC which uses CORBA (Common Object Request Broker Architecture) technology.

Additionally, standardized data structure provides the possibility of modelling tool independent Application Programming Interface (API) for applications, i.e. an application can easily switch among various modelling tools without changing its code. The architecture for accelerator model service can also be extended to beamline instrumentation modelling for an integrated start-to-end simulation platform.

APPLICATION ARCHITECTURE

Typical applications are either desktop-based or web-based. The application architecture should be compatible to both types of applications. As shown in Fig.1, the application architecture diagram is composed by three parts: database, business layer, and client applications. The database is a general data storage container for lattice and model related data; the business layer provides computing power for beam simulation or any other needed computation; and the client applications provide user interface with minimal computation.

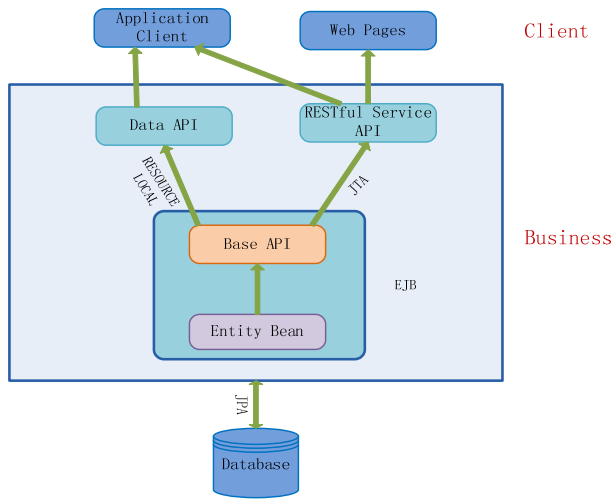

Figure 1: Lattice and Model Service architecture diagram.

Database

A MySQL based database was designed to hold lattice and model data for most accelerators [2]. This database is part of the standardization effort to accommodate various modeling tools, as well as API between database and data services. The technology for mapping Java objects to relational database is through Java Persistence API (JPA). This lattice and model database is part of overall accelerator database collaboration [3].

Business Layer

The business layer in the application architecture contains the business logic, interface between the service and the database, the interface between the service and client applications, and a running engine to provide either on-demand or periodic updating data services. Also, any

*Work supported by the U.S. Department of Energy Office of Science under Cooperative Agreement DE-SC0000661 and China Spallation Neutron Source Project.
#chu@frib.msu.edu

ISBN 978-3-95450-139-7

computation shared among applications such as beam simulation is handled in the business layer. For the Lattice and Model Service, lattice and model data are typically saved in a relational database; however, for performance reason, it is possible to run an inline beam simulation within the service's memory or to acquire the data from another model data provider. It is totally transparent to the client applications where the data actually come from. In order to satisfy popular web service requirements, the Java classes are following the Java Bean standards. Details for each component are described below.

Database Access

To facilitate database access in programmatic way, a set of Java entity classes for object-relational mapping (ORM) are generated following Enterprise Java Bean (EJB) standards. These EJB classes provide base APIs for database accessing. Database query through these JPA classes is much simpler than direct SQL with JDBC (Java Database Connectivity) API. For very few complicated database queries and batch insert, however, direct SQL may be used. Each database table has a corresponding entity class as direct mapping.

On top of the entity class set, there are two ways for serving up data to client applications: one is through data API and the other is through REST (Representational State Transfer) service API. Unfortunately, the data API and REST are using different transaction management, "Resource Local" and JTA (Java Transaction API), respectively, which are not quite compatible. In order to reuse the code as much as possible, commonly shared by the two sets of APIs are implemented in the Base API box shown in Fig. 1. Note that desktop applications can receive services from either data API or REST API; however, web applications can only communicate the service provider via REST API. More details for these two types of APIs are described below.

Data API

Data API is in the form of traditional programming language API which provides convenient access to the database. Applications can call the data API directly without any running service required. These data APIs are predefined database queries to facilitate application programming. With these APIs, physicists can easily program physics applications with the data saved in the database. An example for data API is *getElementByName("an_element_name")* which returns all the element related properties stored in the database as a Java object. Another example for data API can be seen as *setElementProperty("an_element_name", "property_category", "property_name", "data_type", property_value)* which sets an element's property in the database. The majority of the data APIs are derived from the JPA entity classes while the remaining few data APIs are written in direct queries.

REST Service API

There are two main stream service technologies, REST based and SOAP (Simple Object Access Protocol) bases services, with different use cases. For future extendibility to cloud computing, it is in favour of REST based services because the services can be distributed to many servers without worrying about the client request status tracking.

The data access for REST Services are through "get", "put", "delete" and "post" methods. Any data API wrapped by one of these methods then becomes a REST API.

There are over one hundred data APIs in the data API collection where a small subset of the collection is available as REST APIs. The REST API is deployed and managed by a web application server. For REST service API, it is usually specified by URI (Uniform Resource Identifier).

Client Layer

Client applications communicate with the service provider via either data API or service API. Typically, applications can be either desktop based or web based. As mobile devices getting more popular, there will be more web based accelerator applications in the future; therefore, more data API will have REST API support. Ideally, client GUI (Graphical User Interface) for both desktop and web based applications should be shared; however, historically the two types of GUI approaches have not been smoothly merged. JavaFX is the latest Java GUI technology which may provide hope for a single GUI API for both desktop and web applications. Therefore, we select JavaFX for our prototype applications to test out the feasibility for unified application GUI.

PROTOTYPE SERVICE

A REST based prototype lattice and model service has been implemented. For accelerator applications, major service technology candidates are EPICS v4 [4] and web application server. Consideration is mainly based on the complexity of data structure. EPICS v4 services support EPICS control systems seamlessly with data structure for defined in XML syntax. On the other hand, the REST services are intended for web applications on various hardware platforms. For complex data types such as Object or List, it is necessary to specify content types as application/xml or application/json with the data defined in XML or JSON format, respectively. In principle, both EPICS v4 and web services can provide the necessary functions for the lattice and model service. For prototype purpose, we chose the Jersey implementation [5] with standard J2EE Glassfish platform. It should be straightforward to implement the same service for EPICS v4 platform.

Fig.2 shows a screen snapshot of a REST service output example which performs a beamline sequence data query. The URI for this request looks like

ISBN 978-3-95450-139-7

"http://localhost:8080/modelDBREST/webresources/beamlineSequence/name/FE" where "FE" is the short name for Front-End beamline, "name" represents the specified field for a beamline sequence, and the server is "localhost" with port number 8080. The returned query is in standard XML format. Applications can then handle the XML based data with standard XML parser and renderer for further calculation or better display.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <beamlineSequence>
    <beamlineSequenceId>45</beamlineSequenceId>
    <firstElementName>FE_START</firstElementName>
    <lastElementName>FE_END</lastElementName>
    <sequenceDescription>NULL</sequenceDescription>
    <sequenceLength>43.917797</sequenceLength>
    <sequenceName>FE</sequenceName>
  </beamlineSequence>
```

Figure 2: An example for REST API for query of a selected beamline segment.

Both data API and REST API can provide database CRUD (create, retrieve, update, and delete) operations. Particularly, the JPA API nicely handles join queries; for instance, "delete a lattice" operation cleans up all beamline sequences within the lattice, all elements within the beamline sequences and all element properties within the elements.

CLIENT APPLICATIONS

A couple of client applications using either the data API or REST API are under development. Fig 3 shows an editable tabbed panel which contains multiple tables; the GUI application allows users to view and change lattice data saved in the database. The GUI for this application is done with JavaFX technology which is a new Java graphical API intended for both desktop and web applications.

BeamlineSequence Data | Element Data | BeamParameter Data | Model Data | ParticleType Data

firstElementNa...	lastElementNa...	predecessorSeq...	sequenceDescr...	sequenceLength	sequenceName
FE_START	FE_END		NULL	43.917797	FE
LS1_START	LS1_END	FE	NULL	102.970489	LS1 TO STRIPPER
CSS1_START	CSS1_END	LS1 TO STRIPPER	NULL	8.963245	CSS1
STRP_START	CSS1-CSS2_END	CSS1	NULL	0.0	CSS1-CSS2
CSS2_START	CSS2_END	CSS1-CSS2	NULL	14.552447	CSS2
FS1_START	FS1_END	CSS2	NULL	4.103876	FS1
DUMP2_START	DUMP2_END	FS1	NULL	27.503633999999977	DUMP2
LS2_START	LS2_END	DUMP2	NULL	122.20456300000001	LS2
FS2_START	FS2_END	LS2	NULL	40.25585799999999	FS2
LS3_START	LS3_END	FS2	NULL	124.022672	LS3
BDS_START	BDS_END	LS3	NULL	27.024158	BDS

Figure 3: A JavaFX based user interface as a client application for the lattice and model service.

Another application under development is a model run and data display desktop GUI application as shown in Fig.4. This is an example of utilizing XAL [6] framework and the latest lattice/model service. The application can run XAL Online Model as well as display any saved model data from the database.

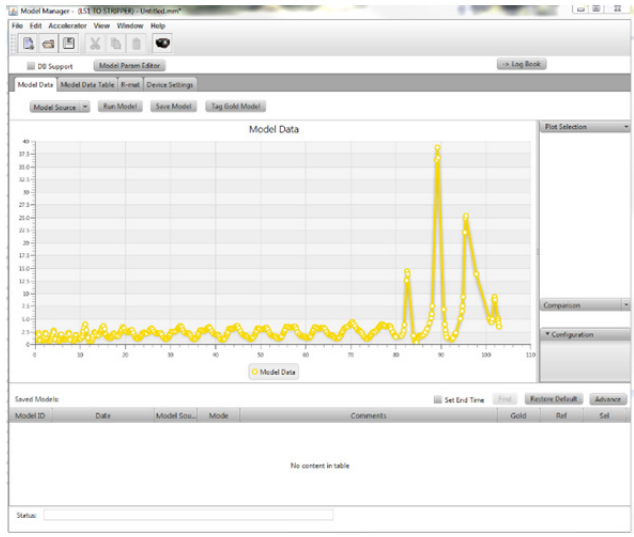

Figure 4: A desktop application for displaying model data. The data plot shown is horizontal β-function for FRIB linac segment 1.

CONCLUSION

A prototype lattice and model service has been implemented with a generic database and REST based web service. A couple of client applications are also developed. To accommodate more use cases such as physics scripts, it is necessary to extend the present service API set. The REST URI can also be simplified with proper wrapper. Furthermore, performance tuning and more applications connecting to the service are planned.

ACKNOWLEDGMENT

The authors would like to express special thanks to Dr. Don Dohan for his contribution to the database design and many valuable advices. The rest of the database collaboration team's help on database related issues as well as the compatibility of the lattice and model database with the other database domains are also greatly appreciated. Early discussion with Dr. Juhao Wu and many other physicists is greatly benefit to the project.

REFERENCES

[1] http://www.slac.stanford.edu/grp/cd/soft/aida/
[2] P. Chu et al., "Database for Accelerator Modeling", Proceedings of 2013 International Particle Accelerator Conference.
[3] V. Vuppala et al., "Distributed Information Services for Control Systems", WECOBA02, these proceedings.
[4] http://epics-pvdata.sourceforge.net/
[5] https://jersey.java.net/
[6] J. Galambos et al., "XAL Application Programming Structure," p. 79, Proceedings of 2005 Particle Accelerator Conference.

ISBN 978-3-95450-139-7