LHC SOFTWARE ARCHITECTURE [LSA] – EVOLUTION TOWARD LHC BEAM COMMISSIONING

G. Kruk, S. Deghaye, M. Lamont, M. Misiowiec, W. Sliwinski

CERN, Geneva, Switzerland

Abstract

The LHC *Software Architecture (LSA)* project will provide homogenous application software to operate the *Super Proton Synchrotron (SPS)* accelerator, its transfer lines, and the *Large Hadron Collider (LHC)*. It has been already successfully used in 2005 and 2006 to operate the *Low Energy Ion Ring accelerator (LEIR)*, SPS and LHC transfer lines, replacing the existing old software. This paper presents an overview of the architecture, the status of current development and future plans. The system is entirely written in Java and it is using the *Spring* framework, an open-source lightweight container for Java platform, taking advantage of *dependency injection (DI), aspect oriented programming (AOP)* and provided services like transactions or remote access. Additionally, all LSA applications can run in 2-tier mode as well as in 3-tier mode; thus the system joins benefits of 3-tier architecture with ease of development and testability of 2-tier applications. Today, the architecture of the system is very stable. Nevertheless, there are still several areas where the current domain model needs to be extended in order to satisfy requirements of LHC operation.

SYSTEM OVERVIEW

Scope

The LSA system covers all of the most important aspects of accelerator controls: optics (twiss, machine layout), parameters space, settings generation and management (generation of functions based on optics, functions and scalar values for all parameters), trim (coherent modifications of settings, translation from physics to hardware parameters), operational exploitation, hardware exploitation (equipment control, measurements) and beam based measurements [1].

One of the main goals of LSA is to provide a clean and generic API to all core functionality, to be used by all operational applications.

Basic Concepts

The whole LSA core functionality is based on a few fundamental concepts among which the most important are *parameter*, *setting* and *context* [1].

Parameter

Parameters are organized in hierarchies, whose roots are usually physics-oriented, high-level parameters (e.g. tune, chromaticity, momentum, …). Leaves are typically hardware parameters such as currents. Each hierarchy describes the relationship between parameters i.e. change of one parameter always affects all its dependant parameters. Operators typically intervene on the root parameters and let the LSA system calculate the appropriate changes in the derived parameters.

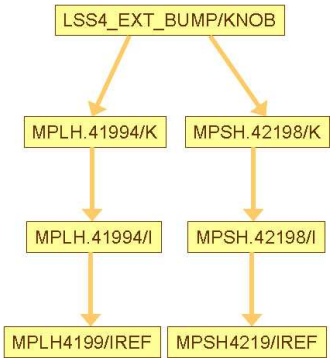

Figure 1 Example of parameters hierarchy where K, I are magnet parameters and IREF is the power converter current

Each parameter is of a specified type which in turn has a value type - a function (change of value in time) or a discrete type (scalar or array of scalars).

Context

A *context* defines a time span in which a parameter can have a value. We define three types of contexts: *super cycle, cycle* and *beam process*. *Super cycle* contains a set of cycles, which produce beams of different types and for different clients. In cycling machines like the SPS, super cycles are played repeatedly.

A *beam process* defines a specific process of a beam (e.g. injection, ramp, extraction) for a given accelerator or transfer line.

Setting

A setting represents the value of a parameter in a given context. Every time a setting of a given parameter is modified (trimmed), the change is propagated to all its dependants, in the parameter hierarchy. Settings of dependant parameters are calculated using so-called *make rules*.

Architecture

From the very beginning it was decided that the system should have 3-tier architecture [2]. There were several reasons for it: central access to the database and hardware, central security and caching, reduced network traffic and load on client consoles, and scalability.

Initially LSA architecture was based on *Java 2 Enterprise Edition* (J2EE), together with *Enterprise Java Beans* (EJB). However, after having disappointing

experiences with EJB standard, LSA team decided to replace it with a new solution for enterprise systems labelled as lightweight container. As the actual implementation of the concept, we have chosen a leading container - the *Spring* framework [3].

LSA Architecture

The LSA architecture is based on three main principles: it is modular (each module has high cohesion providing a clear API to its functionality), layered (with three isolated logical layers – database and hardware access layer, business layer, user applications) and distributed (when deployed in 3-tier configuration).

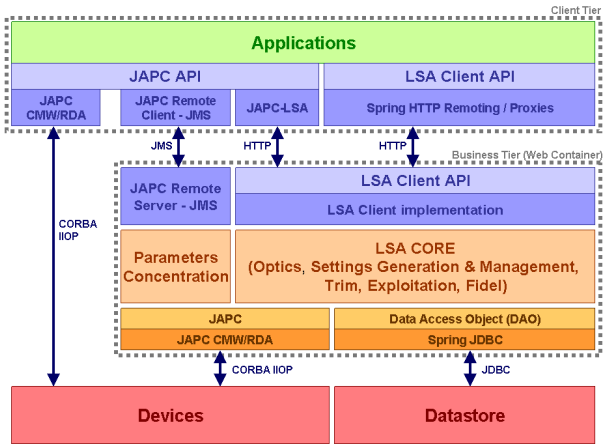

Figure 2 LSA architecture

All applications communicate with the underlying tiers in two standard ways, using the *Java API for Parameter Control (JAPC)* [4] for equipment access and the LSA *Client API* to call the business services available in the LSA core.

The current LSA architecture heavily uses standard services provided by the *Spring* framework:

- Database access is implemented using *Spring JDBC* abstraction layer.
- Transactions are managed by *Spring AOP* based transactions abstraction.
- Synchronous remote communication is done through *Spring HTTP* remoting.
- Testing framework, especially for unit testing of *Data Access Objects (DAO)*.

We have also implemented a caching mechanism with annotation-driven configuration, using the method interception facility provided by *Spring AOP,* and in the near future we plan to use asynchronous communication via *Java Messaging Service (JMS),* which is also nicely integrated in the framework

2- and 3-tier deployment

One of the great assets of the LSA system is that all its applications can seamlessly run both in 2 and 3-tier mode. The 2-tier mode is vital for development and debugging while the 3-tier mode is used for operations.

Two design principles facilitate this solution: (1) all applications do not use directly services provided by LSA core modules, but always go through a set of client controllers (façades) defined in LSA Client module; (2) all these controllers are retrieved using a special service registry based on Service Locator pattern. The registry returns either an actual implementation of the controller (in 2-tier mode) or a dynamically generated HTTP proxy which makes a remote call to the server (in 3-tier mode).

EVOLUTION TOWARD THE LHC

The initial design of the architecture and the domain model was based on requirements for the SPS and its transfer lines and on experience with the *Large Electron-Positron* (LEP) collider. While the architecture did not change significantly for the last 3 years, the data model, domain model and functionality provided by LSA core modules have been considerably updated and extended to address new requirements.

Parameters space

In the beginning, LSA was used only to manage function parameters, mainly for SPS power supplies. However, in view of LHC, LSA shall be more generic and handle settings of all devices in the machine. Therefore the LSA parameter space has been extended to support all types of parameters, including scalars and scalar arrays of any type. While function parameters specify the change of a value over time, scalar (and array) parameters describe time-independent, punctual value (or an array of values) such as the strength of a kick for a LHC injection kicker (scalar) or a thresholds table for different levels of energy for a LHC beam loss monitor (array of scalars).

Context and Settings Management

Unlike SPS or LEIR, the LHC is a non cycling machine. The machine run will be composed of a sequence of processes such as injection, ramp, squeeze that will be executed in an asynchronous way. Some of these processes, like ramp, will have a known length and settings for them will be managed using regular super cycles. The length of other processes, like injection, will depend on several conditions (e.g. beam quality) and will not be known in advance. In order to support settings management for such processes of unknown length, two additional concepts have been introduced to describe a context: an *actual super cycle* and a *hyper cycle*. An *actual super cycle*, as opposed to a regular super cycle, does not have a length and it is used only to manage scalar settings for such steady state processes.

A *hyper cycle* defines a sequence of regular and actual super cycles which are used in one run of the LHC.

Finally, we introduced settings for context-independent (called also cycle-independent) parameters whose value can be changed independently on the beam type in the machine. Cycle-independent parameters usually represent thresholds or limits of various devices.

Security and Access Control

Due to the very high energy stored in the LHC beam and potential damage which could be caused by that, the LHC machine has strong security requirements.

In order to satisfy these requirements, LSA has been integrated with two access control infrastructures: *Role Based Access Control* and *Management of Critical Settings*.

Role Based Access Control (RBAC)

The RBAC [5, 6] infrastructure has been developed in the frame of *LHC at FermiLab Software (LAFS)* collaboration. Its main goal is to prevent accidental and unauthorized access to the hardware. It is based on access maps which describe access rules for specified user roles and hardware properties, taking into account machine mode or location of the user sending settings. Verification of user credentials against these access maps is performed by the *Controls Middleware (CMW)* [6] at the moment when settings are sent to the hardware.

Management of Critical Settings (MCS)

The MCS [7] infrastructure has been developed to complement RBAC with an additional layer of security. It uses digital signatures to protect data integrity of settings of the most critical hardware. Usually critical settings will be modified by experts and immediately sent to hardware, however in some cases settings might be prepared in advance by experts, and later on used by machine operators.

The whole MCS infrastructure is composed of three components:

- LSA which enforces authentication when trimming critical settings, requests RBAC to sign new values, stores new settings together with a digital signature in the database and sends them to the hardware (if requested).
- RBAC system, which allows user authentication, issues a secure token and signs settings with appropriate private key.
- *Front-end Software Architecture (FESA)* [8] which verifies signature using a public key issued for given hardware type.

LHC Timing

All processes in the LHC (injection, ramp, squeeze, etc) will be synchronized and triggered by timing events sent by the LHC timing system [9]. The LSA core provides a timing service to send timing events and access the timing real-time data channel. The events can be sent individually as asynchronous events or structured in tables. The latter are persisted in the database and can be loaded/unloaded from the timing system. The system supports up to 8 concurrent tables containing a maximum of 256 events. In addition, the LSA timing service will be used for LHC injection requests by high-level applications such as the sequencer.

Hardware Transactions

One of the key issues when sending settings to many devices installed in the LHC ring is the atomicity of that operation – either all or none succeeds. The lack of transactional behaviour could lead to serious problems or even damage caused by the beam. Furthermore all devices must be synchronized i.e. start to play loaded settings exactly at the same moment.

To address this requirement a support for hardware transactions has been recently implemented in the FESA [8] framework. The basic idea behind is that settings sent from LSA to the hardware will contain an additional field representing a transaction identifier. All settings sent with such an identifier will not be played immediately, but will wait for a commit. If sending settings to all devices succeeds, LSA will request a timing event (sent via the LHC timing system) containing the same transaction identifier, which will be treated as a commit action.

CONCLUSIONS

As the domain is very complex, the project team started with a base model which was iteratively extended to cover newly coming requirements. After a successful deployment of the system for control of transfer lines, SPS and LEIR machines two years ago, the LSA team has began to work on LHC requirements. Today most of the crucial functionality, required for the first beam in LHC, is in place and is being tested by machine operators. Nevertheless, there are still few areas where current domain model can be improved. In addition, several specialized applications need to be written and the data model has to be completed, therefore the coming months will be certainly challenging.

REFERENCES

[1] M. Lamont, L. Mestre et al, “LHC Era Core Control Application Software”, ICALEPCS’2005, Geneva, October 2005

[2] L. Mestre et al, “A Pragmatic and Versatile Architecture for LHC Controls Software”, ICALEPCS’2005, Geneva, October 2005.

[3] http://www.springframework.org

[4] V. Baggiolini et al, “JAPC - the Java API for Parameter Control”, ICALEPCS’2005, Geneva, October 2005.

[5] S. Gysin “Role-Based Access Control for the Accelerator Control System at CERN”, ICALEPCS’2007.

[6] W. Gajewski “Role-Based Authorization in Equipment Access at CERN”, ICALEPCS’2007.

[7] V. Kain “Management of Critical Settings and Parameters for LHC Machine Protection Equipment”, Functional Specification, CERN, 2006.

[8] M. Arruat “Front-End Software Architecture”, ICALEPCS’2007.

[9] J.H. Lewis “The CERN LHC Central Timing, a Vertical Slice”, ICALEPCS’2007.