DEVELOPMENT OF HIGH-LEVEL APPLICATION FRAMEWORK WITH A SCRIPT LANGUAGE JCE FOR ACCLERATOR BEAM COMMISSIONING

Hiroyuki Sako#, Japan Atomic Energy Agency, Tokai, Japan
Hiroshi Ikeda, Visible Information Center, Inc., Tokai, Japan

Abstract

For accelerator beam commissioning, script language is indispensable, especially in the early stage of commissioning, to create and modify applications quickly and iteratively. A high-level application framework based on script language, J-PARC Commissioning Environment (JCE), has been developed in Java. It is capable of device control via EPICS, a beam transport simulation, GUI components, mathematical functions, and so on, which are flexibly and seamlessly combined in the script. A Mathematica style of language ("SAD script") is adopted. A special care is taken to clearly separate the parser part from actual function parts, and to document the codes. Thus modularity of the architecture, code understandability, and extensibility are dramatically improved. JCE has been utilized successfully for beam commissioning of J-PARC linac.

INTRODUCTION

A scripting language is a very convenient and powerful tool to quickly develop beam commissioning and tuning applications, especially in the early stage of beam commissioning. Strategic Accelerator Design code (SAD) which works in a script language "SAD script" similar to the language of Mathematica has been developed and used in KEK successfully for accelerator design and commissioning [1]. However, it is implemented in rather old language FORTRAN77 consisting of different layers which have been developed over years. Moreover, because the script interpreter and the actual functions are tightly coupled, the software is difficult to understand and modularize and, consequently, hard to maintain and upgrade.

Therefore, we decided to adopt the SAD script as the User Interface (UI) language, but construct a new high-level application framework architecture in Java, named JCE (J-PARC Commissioning Environment) [2]. By doing so, ease of code maintenance and extensibility of the commands is improved dramatically. A schematic architecture of JCE is drawn in Fig. 1. A special care is taken to separate codes implementing the parser from actual command functions simplifying the maintenance and the procedure of adding new commands. JCE adopts also the XAL online model, which can be initialized with the XAL configuration file. Additional information such as device geometry and EPICS channel names in the configuration file can be utilized in applications. Therefore, JCE applications and XAL applications share common online model and common device information, and thus the two frameworks reflect updates of the configuration file and developments of XAL online model at the same time. JCE also imports various useful Java libraries such as Java channel access libraries JCA, RDB interface library JDBC, and other open source Java libraries. JCE also has its own graphics and optimization packages, and has a good extensibility to support a third party simulation code such as the TRACE3D model [3] via JNI (Java Native Interface).

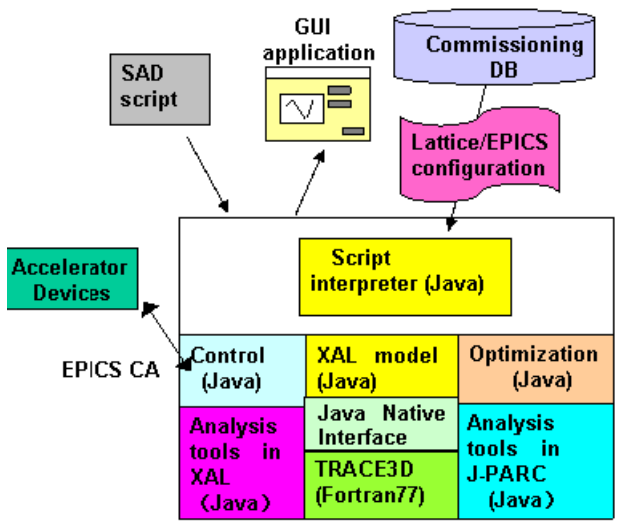

Figure 1: JCE architecture.

IMPLEMENTATION

As shown in Fig. 2, the script interpreter consists of four components; a parser, a builder, an evaluation engine, and commands. The parser is an interpreter engine, which interprets the syntax of the script and converts it into a tree structure called the syntax tree. The parser is implemented with open source software JavaCC (Java Compiler Compiler). The builder further converts the syntax tree into an evaluable tree structure (evaluation tree). The builder has several command factories each of which derives a command from a string of a script element and then embeds the command into the evaluation tree.

The evaluation engine actually "evaluates" the evaluation tree. The engine calls the commands which are embedded in the evaluation tree. Furthermore, symbols and expressions in the tree are transformed to other values and expressions by the engine according to transformation rules. This transformation procedure is called rewriting.

#hiroyuki.sako@j-parc.jp

For instance, rewriting realizes substitution of a value to a variable. Rewriting dynamically changes the content of the evaluation tree. On the other hand, transformation within a command is predefined and static. Java exception mechanism is utilized for run time errors of a script which have occurred during evaluation, and for commands that force flow changes by jumping over multiple cross-calls.

Commands are called by the evaluation engine and they execute particular functions. The commands not only invoke functions in evaluation but also have attributes that can affect creation of the evaluation tree or evaluation procedures. By utilizing such attributes, a script can have a variety of functionalities. These commands are managed by the command factory class, which embeds the commands into the evaluation tree. A variety of commands should be large and wide enough to be able to create various applications for beam commissioning. The current number of commands is about 300 and still increasing steadily. If a new command is required by a user, it must be added easily. A user can add a command easily by implementing a command class based upon a template; the procedure is documented. The commands are categorized as in Table 1.

Table 1: Command Category and Commands

Category	Command examples
Pattern matching	Pattern(:),PatternTest(?), Alternatvies(|)
Mathematical calculations	Plus(+),Equal(==),And(&&),Cos, Fourier
Flow control	If, Do, For, Throw
Optimization	SimplexMinimize, ResponseMatrixMiniize
EPICS	CaRead, CaWrite, CaMonitor
Online model	XalLatticeInfo,XalProbeInfo, XalCalc
Waveform	WaveArchiveReader,CaWaveArchiver
Graphics	Window, Frame, Button, TkWait
Plots	FastXYPlot, OpticsPlot
List operation	Table, Length, Map, Scan
String operation	StringJoin(//), StringLength
File I/O	Get, OpenRead, Read, Write

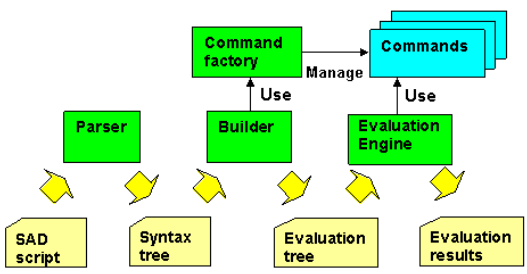

Figure 2: JCE data flow.

Special Features of JCE

There are some special features in JCE worth noting.

1. Optimization of parameters

Optimization of arbitrary number of variables with any kind of linear or non-linear functions is realized. The choice of optimization method is possible among Simplex method, Powell method, and Newton-Raphson method. The last one is the most rapidly converging method referred as the "Singular-Value Decomposition (SVD) method", since it uses a response matrix and calculates a pseudo-inverse matrix using the SVD technique. This method is very powerful and utilized in J-PARC commissioning applications. For optimization of model parameters, special optimization commands combined with the online model have been developed.

2. Event loop

Evaluations in JCE are carried out in a single thread (the main thread) for simplicity and efficiency. Therefore, events from external system must be delegated to the main thread. To realize this, an event queue and an event loop mechanism to receive an event are implemented. Since the event queue is implemented thread safe, any events from external sources using the queue can be notified safely.

3. Graphics User Interface

Each graphic component of JCE is represented as a single expression. Thus, a complex graphic application can be flexibly constructed by combining these components. An example script for a graphic application is given in Fig. 3. SAD uses Tcl/Tk to implement graphic components, while JCE uses the standard AWT/Swing library. Since elementary units of graphics components are different, several basic components are combined to implement JCE graphic components. Tcl/Tk is executed in the main thread in SAD, while UI thread is used for Swing execution in JCE. To absorb the difference the event loop mechanism described above is used.

```
Add-> {KBFComponentFrame[
        Add-> {KBFGroup[Text->"Wire Scanners X for emittance fit"]},
        Add-> {KBFCheckButton[Width->xwid,Variable:>awsx[1],Text->ws[1],WidgetVariable:>wawsx[1]]},
        Add-> {KBFCheckButton[Width->xwid,Variable:>awsx[2],Text->ws[2],WidgetVariable:>wawsx[2]]},
        Add-> {KBFCheckButton[Width->xwid,Variable:>awsx[3],Text->ws[3],WidgetVariable:>wawsx[3]]},
        Add-> {KBFCheckButton[Width->xwid,Variable:>awsx[4],Text->ws[4],WidgetVariable:>wawsx[4]]},
```

Figure 3: An excerpt of a JCE script to create a graphic application.

4. Online Models

We use XAL online model as a core model. We can also use the TRACE3D model [3] via Java Native Interface (JNI). With JCE commands, model parameters of all elements in the lattice and initial beam parameters can be set. The calculated beam properties can be taken as a set of parameters in the script, which can be analyzed and visualized further. Therefore JCE can control the model flexibly and combine the model with other commands.

5. Configuration File

The XAL configuration file is used to initialize the model. The whole tree structure of the XML file format is mapped to the JCE list and can be accessed via commands.

6. Online Accelerator Map

Recently, an online accelerator map has been added as shown in Fig. 4. This GUI component displays a lattice structure in one-dimensional and two-dimensional representations. The latter is especially useful for a ring. The map can be automatically created from the XAL configuration file.

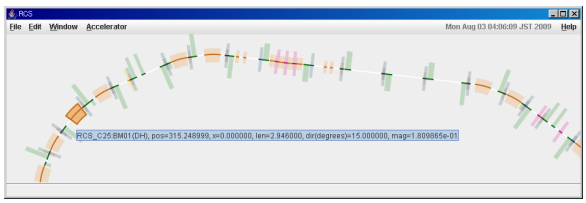

Figure 4: The online accelerator map GUI component.

APPLICATIONS AND BEAM TUNINGS

Here we show important beam commissioning applications implemented in JCE.

Transverse Matching Application

The "matcher" application was implemented for transverse beam matching corrections for J-PARC linac as shown in Fig. 5. There are several matching sections in the linac where beam profiles are measured with 4 wire scanners placed at periodic lattice positions. There are also 4 or more knob quadrupole magnets upstream the wire scanners. The beam profiles are measured and beam widths are calculated with "wsm" application developed in JCE shown in Fig. 6. In the matching procedure, first, at an upstream position before quadruple magnets, transverse Twiss parameters and emittance of the online model are fit to the measured beam widths. Then, in the online model, corrections of quadruple magnet fields are calculated requiring Twiss parameters agree with each other at the wire scanners. For these calculations, SVD method is used. Corrected beam envelopes are shown in Fig. 5.

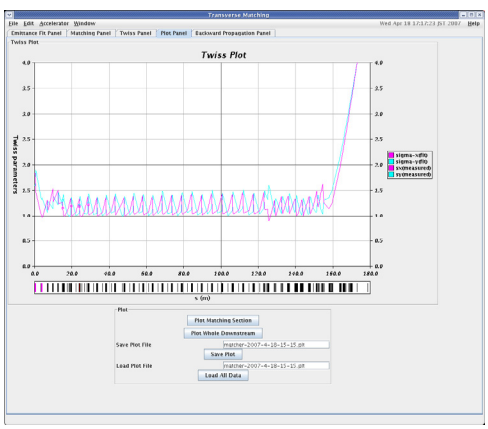

Figure 5: The transverse matching application.

Beam Operation Applications

For beam operations, we have developed beam monitor displays for beam position monitors, fast current transformer for energy measurements, and beam loss monitors. Also applications to manage magnetic field settings for quadrupole and dipole magnets have been developed.

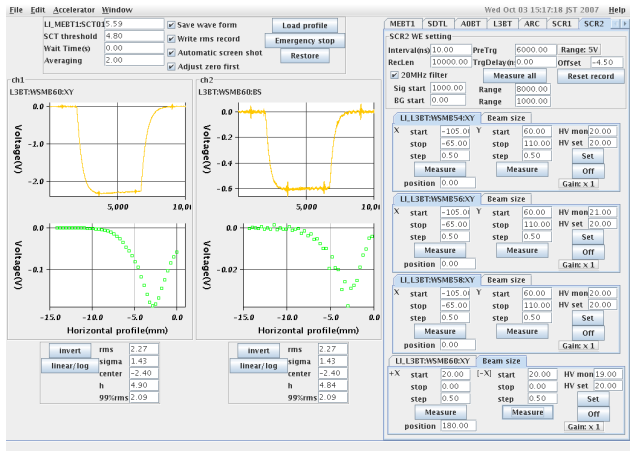

Figure 6: The wire scanner measurement application.

SUMMARY AND OUTLOOK

A high-level framework JCE based on a script language has been developed for J-PARC beam commissioning and operation. Based on the framework, high-level applications have been developed and utilized successfully for J-PARC linac. In a near future, environment tools for JCE script development are going to be developed.

REFERENCES

[1] K. Oide and H. Koiso, “Anomalous Equilibrium Emittance due to Chromaticity in Electron Storage Rings”, Phys. Rev. E49, 4474 (1994).

[2] H. Ikeda, H. Sako, *et al.*, “Development of a SAD Script Interpreter with Java”, Proceedings of the 2nd Annual Meeting of Particle Accelerator Society of Japan, 2005, Tosu, Japan; H. Ikeda and H. Sako, "Development of Script Interpreter for Beam Commissioning at J-PARC LINAC", Proceedings of the 4th Annual Meeting of Particle Accelerator Society of Japan, 2007, Wako, Japan.

[3] K. R. Crandall and D. P. Rusthoi, “Trace 3-D Documentation”, LANL Report LA-UR-97-887 (1997).