# Development of a SAD script interpreter with Java

Hiroshi Ikeda[1,A)], Hiroyuki Sako[B)], Hironao Sakaki[B)], Hiroki Takahashi[B)], Hiroshi Yoshikawa[B)], Yuichi Itoh[B)]
[A)] Visible Information Center, Inc.
440 Muramatsu, Tokai, Naka, Ibaraki 319-1112, Japan
[B)] Japan Atomic Energy Research Institute
2-4 Shirakata-Shirane, Tokai, Naka, Ibaraki 319-1195, Japan

*Abstract*

The SAD script is a language for commissioning, operation and simulation of an accelerator, which has been developed and used in KEK. We plan to use it in J-PARC. For maintenance, management and extension, we are reconstructing the interpreter of the SAD script with Java. We implement main engines to parse and evaluate the script, fundamental commands, handlings of DB, and methods to call the orbit calculation code, etc. We report the progress.

# JavaによるSADスクリプトインタプリタの開発

## １．はじめに

SADスクリプト[1][2]は、KEKで開発・使用されてきたスクリプト言語である。SADスクリプトは、単にシミュレーションを行うためだけでなく、その記述と同じ文法で加速器も運転制御可能という特長を有するため、特に制御シーケンスを模索するコミッショニングにおいて威力を発揮する。J-PARCにおいても、この機動性を活かし、さらには、SADスクリプトで書かれたKEKBのアプリケーション資産を利用することを考え、コミッショニング時からSADスクリプトを利用する予定である。

現在のSADのシステムは、SADシミュレーションコアとスクリプトインタプリタ部の両方が混在したソフトウェア構造となっている。このため、変更や拡張を行うには、SADの開発者と同程度の深い内部構造の知識が求められる。そこで、我々は、J-PARC単独での保守を確保するために、SADからスクリプトインタプリタ部を切り出し、保守・管理や機能拡張性をさらに高めた新しいSADスクリプトインタプリタの開発を行うことにした。

この開発には、将来の拡張性を考えオブジェクト指向言語であるJavaを用いている。これは、Javaはプラットフォームを選ばない標準的なオブジェクト指向言語であり保守管理が容易であること、また、XAL[3]の各機能がJavaで実装されているため、組み込みが容易であることが挙げられる。ここで、XALとは、米国で現在建設中のJ-PARCと同規模の陽子加速器 Spallation Neutron Source (SNS)で開発されている軌道計算・制御用ソフトウェアフレームワークであり、J-PARCで利用できるツールが多く存在する。SADスクリプトインタプリタをJavaで記述することで、KEKBのソフトウェア資産に加え、これらのSNSのソフトウェア資産が利用できるというメリットは大きい。

本発表では、開発中のインタプリタについて、その特徴と現在の実装の概要、および、既存のインタプリタとの差異について説明を行う。

## ２．インタプリタの特徴

本インタプリタの特徴としては、以下が挙げられる。

- Javaによる実装
- 既存のSADインタプリタとの互換性
- コマンドの追加の容易さ
- ドキュメントの充実

Javaは、プラットフォームを選ばない標準的なオブジェクト指向であり、保守管理が容易である。学術的な方面にも使われることが少なくなく、それらの最新の技術を取り入れることも容易である。制御・計算に限らずJava周辺の技術の発達も目覚しく、これらの技術を利用できる可能性があるのは大きな利点である。

また、既存のSADスクリプトインタプリタと互換性があるため、KEKで蓄積された資産を利用することができる。

コマンドの追加の容易さ、ドキュメントの充実は、インタプリタの保守・管理や拡張性に直結するものであり、これらは、本インタプリタの開発の大きな動機の一つである。ドキュメントの充実化については、APIを閲覧するWebページの自動生成を行うJavaの標準的支援ツールJavadocが利用できる。

## ３．インタプリタの開発手順

本インタプリタの開発を以下の手順で行った。

(1) 既存のインタプリタの挙動及びドキュメントから、SADスクリプトの仕様を調査する。
(2) データ構造およびモジュールとデータの受け

[1] E-mail: ikeda@vic.co.jp

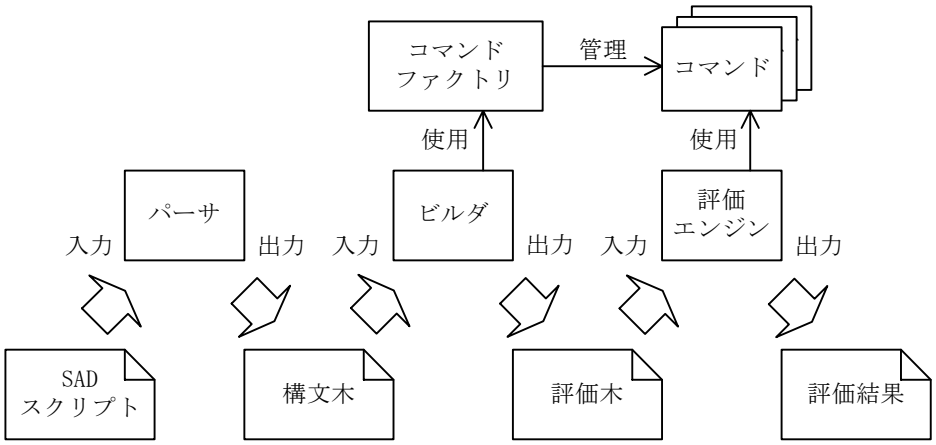


図1 インタプリタの構成

渡しの手順について、全体構成を設計・実装する。全体構成については次章で述べる。

(3) 各コマンドの挙動の詳細調査・実装と、それに伴うメインモジュールの改良・拡張を行う。

## ４．インタプリタの構成

SADインタプリタは、パーサ、ビルダ、評価エンジン、コマンドの４つのモジュールに分けて開発している（図1）。

### 4.1 パーサ

パーサ（解釈エンジン）は、与えられたSADスクリプトから、それが表現する構文木を作成する。パースに失敗すると、すなわち構文エラーとなり、評価は行われない。パーサの構築にはJavaCC[4]を利用している。

構文木は、SADスクリプトと密接に関係する単なる構造を表すものとして位置付けられ、ビルダによって評価可能な木構造に変換される。

### 4.2 ビルダ

ビルダは、構文木からさらに変換を掛け、評価可能な木構造（以下、評価木）を作成する。ビルダは、名称を表す文字列からコマンドを導き出すコマンドファクトリを複数所持し、コマンドを評価木に組み込む機能がある。

### 4.3 評価エンジン

評価エンジンは、実際に、評価木の評価（すなわち、副作用を伴う変換）を行う。評価エンジンは、評価木に組み込まれたコマンドを呼び出す他に、上方値や下方値と称される式の変換ルール（以下、リライティング）に従って式を変形していく。SADスクリプトにおける変数の代入などは、変数-値の組み合わせではなく、リライティングによって実現されている。

リライティングは、式の照合と代入の２つの機能によって構成される。式の照合に関しては、SADスクリプトでは表現力の強い記述が可能であり、これ自体一つのコマンドとして捕らえることもできる。照合や代入の機構は、コマンドから呼び出される。

これらのリライティングは、評価エンジンに付随するデータとして扱われ、代入操作のコマンドの実行などにより、追加・削除される。評価エンジンは、これらコマンドやリライティングを、どのように実行するかを制御するモジュールという位置付けとして捕らえることができる。

評価中に発生したスクリプトの実行エラーや、複数の相互呼び出しを跨って評価の流れを強制的に変更するコマンドの実装は、Javaの例外の機構を用いている。

### 4.4 コマンド

コマンドは、評価エンジンから呼び出され、特定の動作を行う。コマンドの存在数・拡張性がインタプリタの表現力・可能性であり、インタプリタを構築する上で重要な点となる。

コマンドは評価時の動作の他、評価木の構築や評価手順にも影響を与える属性等をもち、これらを利用することで大きな表現力を持たせることができる。これらのコマンドは基本的にコマンドファクトリと称されるものによって管理され、ビルダによって評価木が作られる際にそれにコマンドが組み込まれる。

現在実装されているコマンドは、約250個になる。EPICSとの通信を含む既存のSADスクリプトにおけるコマンドをはじめ、新規に追加されたコマンドとして、DBへのアクセス、Trace3D・XALのメソッドを呼び出すコマンドを実装してきている（表1）。

EPICSとの通信、DBへのアクセス、Trace3Dの初期化ファイル生成に関しては、KEKのLINAC MEBT1の実験にてスクリプトのテストを行い、成功している。

## ５．既存のインタプリタとの差異

現在のインタプリタは開発段階ではあるが、既存

表 1 実装済コマンドの概要

| 分類 | | コマンド（抜粋） |
|---|---|---|
| 既存 | 照合 | Pattern(:), PatternTest(?), Alternatvies(\|) |
| | 評価制御 | Hold, If, Do, Throw, Module |
| | 数値・論理演算 | Plus(+), Equal(==), And(&&), Cos, Fourier |
| | リスト演算 | Table, Length, Map(/@), Scan |
| | 文字列演算 | StringJoin(//), StringLength |
| | ファイル入出力 | Get, OpenRead, Read, Write |
| | 代入 | Set(=), UpSet(^=), AddTo(+=) |
| | GUI | Window, Frame, Button, TkWait |
| | EPICS との通信 | CaOopen, CaRead, CaWrite |
| 新規 | DB との接続<br>Trace3D 入力ファイル生成 | JparcDBOpen, JparcDBGetDataSet<br>JparcCreateT3DinputFile |
| | XAL との通信 | XALBeamLine |

のインタプリタとの差異について以下に述べる。

### 5.1 FFS

SADでは、SADシェルとフレームワークFFSの２種類の文法が混在している。SADシェルでは、主にビームの光学、軌道計算を行い、フレームワークFFSでは、数学計算やGUIを扱っている。

本インタプリタでは、FFSの文法に対して処理を行う。

### 5.2 GUI

SADでは、Tcl/TkによりGUIを構築しているが、これは、JavaのGUIフレームワークとは仕組みも画面構成の単位も異なる。したがって、最終的にも互換性にある程度の制限を許容することになるだろう。

例えば、Javaでは、GUIコンポーネントを扱うために、UIスレッド（または、イベントディスパッチャー）と呼ばれる特別なスレッドが用意されている。また、GUIコンポーネントは、基本的にこのUIスレッドのみからアクセスすることを前提に作成されており、スレッドセーフではない。したがって、メインスレッドとUIスレッドの両方を使うには、マルチスレッドを扱う場合に起因する問題に対して気をつける必要がある。

一方、SADではTcl/Tkを用いており、シングルスレッドでのみGUIコンポーネントを扱う。また、イベントディスパッチループに入るためのコマンド（TkWait）の後に、ディスパッチループを抜けた後の処理を記述できる点も、JavaのGUIフレームワークと思想が異なる。

### 5.3 名前空間（コンテキスト）

現在のところ、コンテキストと称される名前空間はサポートしていない。

SADスクリプトの仕様上、名前空間の扱い方には注意が必要である。例えば、あるシンボルが与えられたとき、それがどの名前空間に属しているかというのは、（その箇所だけでは判断ができず）以前にそのシンボルが出現していたかどうかで変わる可能性がある。ここで、未評価の式に含まれていたシンボルも出現したと見なされるので、この点でも直感的にも分かりづらい。

### 5.4 実行速度

現在のところ既存のインタプリタより実行速度は低下している。実行速度は環境に依存するが、同一機器で簡単なループの実行時間を確認したところ、既存のインタプリタより実行時間は2～4倍ほどになった。

実行速度は低下したが、既存のインタプリタに見られた不安定さはなく（非常に極端な命令でもクラッシュしない）、拡張性が増している。SADスクリプトのように表現力の高い言語は実行速度が遅いのが一般的であり、そもそも、スクリプトの利点は実行速度ではなく、手軽さや柔軟性にある。

実行速度に関して言えばコンパイル型言語の方が有利であり、速度が求められるロジックについては「コマンド」として直接Javaで実装することを検討するのが良いだろう。

## ６．今後の予定

今後は以下の項目の実装を行う。これらは、J-PARCのコミッショニングが開始されるころには完成する予定である。

- 未実装であるSADスクリプトのコマンドの実装や拡張
- XALとの連携を強化する。特に、Latticeの入力、基本的optics計算機能の導入
- 任意のJavaBeanに対するインスタンスの生成や実行
- マルチスレッドの導入

## 参考文献

[1] URL: http://acc-physics.kek.jp/SAD/sad.html
[2] URL: http://www-acc-theory.kek.jp/SAD/SADTkinter.pdf
[3] URL: http://www.sns.gov/APGroup/appProg/xal/xal.htm
[4] URL: https://javacc.dev.java.net/