Development of Java libraries for accelerator control applications in J-PARC

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 Agency
2-4 Shirakata-Shirane, Tokai, Naka, Ibaraki 319-1195, Japan

Abstract

For efficiently management of several thousands of equipments in J-PARC, we need the control applications with united UI (User Interface). Moreover, for efficiently development of such applications, we need to make libraries to share basic modules of them. On the other hand, we are developing such control applications with a programming language Java because of the following advantages: (1)Java is a standard object-oriented language and independent from platforms, so applications written with Java have advantages to maintain, (2)There are many general and good quality libraries developed by Java, moreover there are libraries specialized to accelerators, such like JCA[1] and XAL[2], (3)The development of the technologies around Java are remarkable in these days, so there are potentialities to apply such high technologies to the applications. So we have been developing the libraries for the control applications with Java. The libraries provide the basic parts of the function such as I/O or UI, etc. which will be often appeared in the control applications, and moreover modulate concepts of devices and channels. The one of final goals is to use the libraries in the high-level applications, which work on a variety of kinds of equipments for many purposes of commissioning, and also have the functions of the simulations. In this paper we will describe the basic concepts, structures, problems, and directions to the future of the libraries.

J-PARC加速器制御アプリケーション用Javaライブラリの開発

１．背景

加速器は複数のデバイスから構築されており、それぞれのデバイスを制御するために、チャネルという単位で信号の送受信を行なう。本ライブラリは、このチャネルにより受け取った信号をそのままあるいはデバイスの単位でまとめ、UIで表示し、あるいはUIから制御のためのデータを送信する。

デバイスの形態は多岐多様に渡るため、本ライブラリでは、アダプタパターン[4]を用いることを前提として高い抽象度でモデルとUIを分離し、UIコンポーネントの適用範囲を広げている。これにより、L&F（ルックアンドフィール、見かけと感触）を統一し、アプリケーションのユーザの誤判断や誤操作を防ぐことを目標としている。また、UIとモデルを分離することは、信号の集合をデバイスとして抽象化することや、アクセス制御を行なうロジックの追加など、モデルに対し複雑な加工を行なうことを可能にする。

一方、ユーザの誤判断や誤動作を防ぐには、L&Fだけでなく、内部で使われるモデルの統一も重要である。例えば、単にJCA[1]をラッピングしたモデルを作成する場合でも、エラーハンドリングからスレッドセーフ性まで様々な憂慮すべき問題があり、このため実装者によってモデルの挙動が異なる可能性がある。あるいは間違った実装を行なう可能性もある。本ライブラリは、このような実装に注意が必要な箇所をカプセル化し、アプリケーション間で統一して使用することを想定したモデルを提供する。

なお、JCAはEPICS[3]によるチャネルの信号の送受信に用いられるJavaのライブラリであり、本ライブラリでは、JCAの他、より使いやすくするためにJCAに多くの機能を追加するXAL[2]も利用する。

本ライブラリは、単なるユーティリティではなくアプリケーションの構築に対し設計や方針を提示するフレームワークに近い。アプリケーションの開発は、本ライブラリを使ったプロトタイプアプリケーションを基に行なわれることが多いだろう。図 1に本ライブラリを用いたアプリケーションとして想定される全体構成の概略を示す。

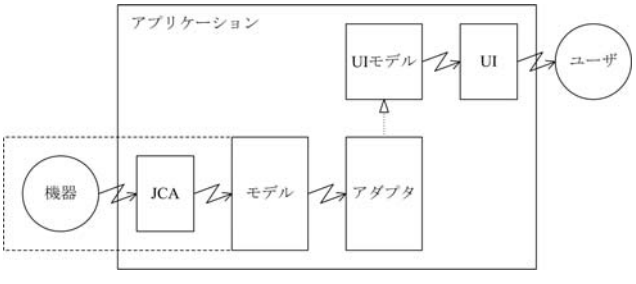

図 1 アプリケーション全体構成の概略

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

２．ライブラリ構成

本ライブラリは4つのモジュールで構成されている。これらは、変更・修正による後戻りを最小化し、また、ライブラリの適用範囲を広げるため、抽象的な表現の定義から具体的な実装を行なう方向で作成されている。以下に、各モジュールの概要を記述する。

2.1 stdlib

本ライブラリにおける最も基本となるモジュールである。表現対象のモデルとしてインタフェースを定義し、それに対するUIコンポーネントを実装している。

UIの表示対象とするモデルは、ここで定義されるインタフェースへのアダプタが作成可能であるならば、任意のものでよい。また、スレッドセーフにアクセスしたりイベントを発効したり出来るように、別途インタフェースやクラスを用意している。本モジュールを含め、４つのモジュール定義するモデルは、UIコンポーネント用のアダプタも用意している。

用意されているUIコンポーネントはおおよそ以下の種類に分類される。

(1) LED系
単色により状態を表示する。またツールチップにより必要に応じて情報を表示する機能を有する（図 2）。

(2) テキスト系
実数の表示または編集を行なう。必要に応じてツールチップを表示する機能を有する。編集用コンポーネントは、マウスやカーソルキーを用いて値の編集可、バックグラウンドの色を指定可である。ただし、編集中はそれと知らせる色が使用される（固定、プロパティファイルで変更可能）。編集をキャンセルして元の値に戻す機能がある。

(3) ストリップパネル系
上記の複数種類のコンポーネントを組み合わせて、デバイスの状態を表すためのやや複雑な情報を表示する。同じモデルにてテーブル表示用コンポーネントが使用可能である（図 3）。

このほか、コマンドの実行を通知するボタンや実行状態を表すテキスト表示など、基本的なUIコンポーネントが用意され、そのモデルの内容を反映する構造になっている。

2.2 stdlib.ca

チャネルのモデル化と、そのモデルに対するUIへのアダプタの生成に有用なクラスを定義するモジュールである。また、デバイス等の「状態」とそれを提供するモデルの枠組み及びUIへのアダプタもここで定義する。後者の具体的な実装は、次のモジュールで行なわれる。

チャネルのモデル化に関しては、JCAから得られるデータやエラー情報を保持し、状態の変更をイベントとして外部に通知する（通知は、別スレッドで起こるため、スレッドセーフ性に注意する必要は依然として残る）。JCAから得られたデータには概念的に複数の情報が含まれる可能性があり、ここから情報を取り出すためにビット演算等を施すことができる。これに伴い、不必要な通信が発生しない様にJCAに対するモニタリングはプーリングされる。また、チャネルへのアクセス制限を外部から与える機能があり、これはインタフェースとして表現され、次のモジュール stdlib.ca.devices にてデバイスの状態によるアクセス制限として用いられることを想定している。

2.3 stdlib.ca.devices

本モジュールでは、デバイスという抽象概念を実装する。デバイスは、単にチャネルの集合であるだけでなく機器状態の提供も行なう。それによりチャネルのアクセス権の制御にフィードバックし影響を与える可能性がある。さらに、デバイスからUIコンポーネントへのアダプタだけでなく、デバイスに対する汎用的なUIコンポーネントが実装されている（図 4）。

デバイスの最も基本的な構成要素はチャネルであるが、前節までに定義したチャネルの概念では情報が足りないため、さらに幾つかの情報を追加したクラスを定義している。例えば、チャネルはその用途に応じ、使用すべきデータ型やその情報の扱い方・対応するUIコンポーネントが変わってくるが、これを幾つかの「タイプ」にグループ化している。この他にも、ストーブやリードバックといったチャネル間の関係と扱い方についての情報や、属するデバイスの状態に対するアクセス権の情報を持たせている。

このモジュールでは、ユーザが必要とする付随情報の全てを含ませることを想定しているため、今後の要求の変化に応じ、プログラム内部を柔軟に変化させる、または、用途に応じて別途新規にモデルを作成するなどの対応が必要であろう。

2.4 stdlib.ca.devices.io.relaxer

前節までに定義したモデルに対し実際にインスタンス化してデータを設定する方法として、プログラム中にハードコーディングするよりは、外部リソースに分離する方が、データの変更に強く、またコンポーネントやアプリケーションの適用範囲を広げることに繋がる。また、必要なデータを他のデータから自動生成を行なうことや、逆に他のアプリケーション用のデータへ変換する可能性も持ち、また一般にデータはアプリケーションよりも寿命が長く、それだけで価値があるものと捉えることができる。

コードの観点からは、ロジックの記述を一箇所に固めて抽象化することにより、本質的なロジックが浮き彫りになり、ロジックの変更や不具合に強くなる。

本モジュールでは、前節までに定義したモデル等に対し、その構築情報やUIコンポーネントへマッピングの情報を外部ファイルから読み込む機能を持つ。

外部ファイルの書式は、RelaxNG[5] と呼ばれるXMLのスキーマで定義している。この利点は、

Relaxer[6] というコード自動生成ツールが整備され、開発や保守に優位であることが挙げられる。

本ライブラリを用いて作成したアプリケーションが使う外部ファイルとして、以下の３種類を想定している。

(1) モデル自身
UIに依存せずデバイスやチャネルに対して抽象的な名前が割り当てられる。

(2) UIコンポーネントへのマッピング
(1)で定義したモデルをどのようにコンポーネント用のデータに振り分けるかを記述するUI依存のデータであり、そのマッピング自身に名前を付ける。

(3) GUIアプリケーション上へのマッピング
アプリケーションに配置されたUIコンポーネントに対し、(2)で定義したどのマッピングを使用するかを記述する。

ここで、(3)はむしろ本ライブラリを用いたアプリケーションの作成者がその必要性を判断して作成する類のものであるが、(2)のマッピング名をプログラム中にハードコーディングするより外部ファイルに記述することで、同じアプリケーションを異なるデータセットにて流用できるように作成するのが望ましいだろう。

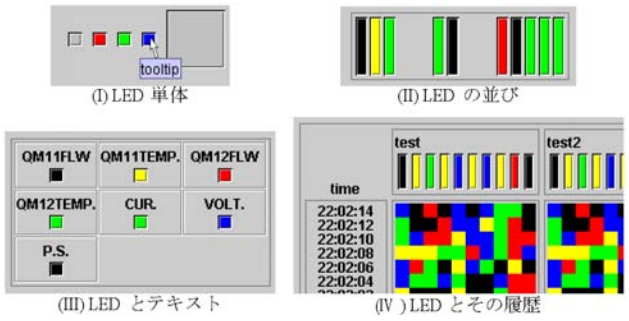

図 2 LED系UIコンポーネント

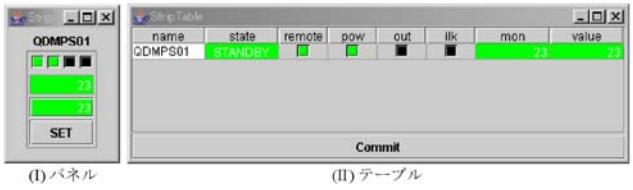

図 3 ストリップ系UIコンポーネント

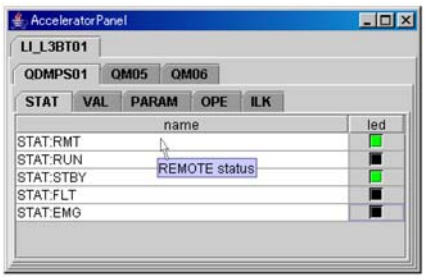

図 4 デバイス用汎用UIコンポーネント

３．問題点と今後の予定

そもそも、現状に適合させたより適切なライブラリへ洗練するには、実際に本ライブラリを用いてアプリケーションを作成し、そのフィードバックを取り込むことを繰り返すことが不可欠である。

以下に、現状の問題点とその解決案について記述する。

3.1 データ

現状では、XML文書から設定を読み込む方法を用意しているが、CSVやデータベースなどその他の方法で格納されているデータを読み込む手段や、これらのデータを設定ファイルに変換する方法を用意し、適用範囲を広げる。

3.2 モデル

様々なデバイスに対応するため、また、不必要な柔軟性を削り構造を簡素化するため、必要に応じて内部モデルの修正や概念の拡大を行なう。

デバイスというのは複数のチャネルを保持するという以外は様々な形態がある漠然としたものであり、具体的な型構造を決めるには十分な考察が必要である。デバイスの概念を修正・決定しライブラリに反映させていくのは、これからの課題である。

3.3 構造

ライブラリという中間層を入れることで、プログラミングが返って難しくなる可能性がある。ライブラリ自体の分かりやすさ、使い勝手はアプリケーション開発の効率性に影響するだけでなく不具合の防止・発見に繋がる。

現在のところ、手戻りや修正による影響を最小限に抑えるため、抽象化から具体化への段階を４つのモジュールで表している。必要に応じ、任意の抽象化の段階が使用可能という適用性・汎用性を優先したが、このため末端のコンポーネントのAPIだけでなく、内部のさまざまなAPIが公開される。エンドユーザ向けにAPIを限定し内部構造を固定するには未解決な要素が多く、これを決めていく必要がある。

3.4 ユーザインタフェース

アプリケーションユーザの負担を軽減するため、UIのルールを分かりやすく単純化することが必要である。実状に合わせて修正・変更する。

参考文献

[1] http://jca.cosylab.com/
[2] http://www.sns.gov/APGroup/appProg/xal/xal.htm
[3] http://www.aps.anl.gov/epics/index.php
[4] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994) Design Patterns: Elements of Reusable Object Oriented, Software. Addison-Wesley
[5] http://www.relaxng.org/
[6] http://www.relaxer.org/