Swingによるリッチクライアントフレームワークの構築
ー プレゼンテーション層の自動生成に向けて ー

テクニカルコンピテンシー部
標準化チーム
ITアーキテクト

金丸　正憲

Masatoshi　Kanamaru
masatoshi-kanamaru@exa-corp.co.jp

Webシステムが企業に普及するにつれ、業務システムに要求される複雑な画面を容易に表現できるリッチクライアントが関心を集めている。また、一般にコード量が多く、仕様変更が頻繁に発生するクライアント画面の開発には、生産性や品質の向上に加え保守性の確保が必要である。これらの要求に応えるため、Webシステム向けリッチクライアント開発に適用できるアーキテクチャの設計とフレームワークの構築を行った。さらに、実システムをモデルケースとして、その適用検証を行い、有効性を確認した。

1. はじめに

企業システムの主流は、ビジネス拠点の展開容易性やマーケットニーズへの即応性を向上するため、クライアント/サーバ（２階層）システムからアプリケーションサーバを使用するWebシステムに移り変わってきた。これに伴い、Visual Basic（以下VB）等をフロントエンドに使った複雑なGUI（Graphical User Interface）を持つシステムをWebシステムに置き換える案件が増加した。このようなシステムでは、既存システムで使い慣れたエクセルライクな表の操作、ツリー表示、図やグラフの表示などのリッチなGUIの実現が求められているが、Webシステムは複雑なGUI表現が苦手であり、また作成効率の面にも問題がある。

一方で、当社はMDA（Model Driven Architecture）手法を用いたWebシステム構築の自動化に取り組んできた。今までに、MVC（Model-View-Controller）フレームワークのうちModel層に着目し、DAO（Data Access Object）自動生成の技術開発を進めてきた[1) 2)]。その検証作業の中で、プレゼンテーション層のコードがシステム全体の６割近くを占めることが明確となり、その自動生成の必要性が改めて認識されていた。

本論文では、このような背景・ニーズから行ったWebシステム向けリッチクライアント用フレームワークの設計と試作結果を報告する。設計に際し、画面作成の容易性はもちろんのこと、コード量が多い画面の生産性と品質確保が重要である。ここでは、生産性の面では最近成熟してきたGUIビルダーを活用し、品質面ではMVCアーキテクチャに基づくプレゼンテーション層のフレームワークを開発するという方針に基づき設計と開発を進めた。MVCアーキテクチャは、MDAの実現という視点からだけでなく、仕様変更が極端に多い画面の保守性を高めるためプレゼンテーション層の独立を保つという視点からも重要である。

まず２章で、現行のWebシステムのクライアント開発における課題を述べ、これを解決するための要求事項を整理する。次に３章で、要求事項を実現するアーキテクチャの設計方針について示し、続く４章で、方針に基づき実施した設計内容を紹介する。さらに５章で、本フレームワークを実システムに適用した検証結果について述べ、その有効性について考察する。

2. 企業システムのプレゼンテーション層に要求される事項

2.1. Webシステムの課題

(1) HTMLの表現力不足

Webシステムで表現できるGUIは基本的にHTMLで表現できるものに限られる。これは簡単なフォーム、プルダウンメニュー、ボタン等であり、スプレッドシートのようなGUIを実現するのは難しい。JavaScriptなどのスクリプト言語をHTML中に埋め込み、リッチなGUIを提供する方法もあるが、生産性・保守性に問題があるだけでなく、ブラウザの違いで動作が変わることもある。

(2) 画面作成効率の悪さ

従来型の２階層システムでは、画面のレイアウトを確認しながら作業できるVBライクな画面作成が当たり前であった。これと比べ、Webシステムでは画面作成の効率が悪い。

当初利用されていたJSPは、HTMLに容易に動的な内容を付加できる機能を提供した。簡単なタグを付加することによりHTMLを一部変更するだけで、動的データを表示できるのが本来のデザインコンセプトである。HTMLについては、優秀なオーサリングツールが豊富にあるため、それらを用いて画面作成を行うことを想定していた。

しかし、Strutsをはじめとするフレームワークが普及した結果、フレームワークの機能を利用するためにカスタムタグをHTML画面に記述するようになった。基本的なHTMLタグの機能を使う場合でもカスタムタグを通して使用するため、一般のHTML用のオーサリングツールやブラウザでは表示が行えなくなり、逆に画面作成の生産性を悪化させるという問題が生じた。

(3) リッチクライアント用フレームワークの標準化未整備

前述した諸課題を解決するため、リッチクライアント用フレームワークが登場してきているが、それぞれに問題を抱えている（表１）。例えば、AjaxはJavaScriptを使い非同期にサーバと通信するフレームワークの総称であるが、実装はまちまちで標準化されているとはいい難い。また、B2Cの分野では、Flashが市場を席巻しているが、もともとの用途がデザインよりであり、SIerが開発者を集めにくいという問題がある。FlexやAIRのようなサーバサイドとの接続を意識した構成もあるが、一般的とはいえない。

Java系においても、Java Application Framework

る。最近発表されたJavaFX ScriptはGUIビルダーがまだ提供されていない。

これらフレームワークは、要件によっては適用することは可能であるが、企業システムで使える標準といえるものにはなっていない。

表１　代表的リッチクライアント製品

名称	ベンダー	形態
Ajax	種々	HTML + javascript
Flash	Adobe System	Plug-in
Curl	Curl (住商情報システム)	Plug-in
BizBrowser	アクシスソフト	Plug-in
Applet	Sun Microsystems	Plug-in
Smart Client	Microsoft	Application
Java Web Start + Javaアプリ	Sun Microsystems	Application
Workplace Client	IBM	Application
AIR	Adobe Systems	Application
JavaFXscript	Sun Microsystems	Application
……	……	……

2.2. プレゼンテーション層への要求事項

以上より従来の２階層システムにおけるGUIのサービスレベルを落とさずにシステムを構築する際のプレゼンテーション層への要求事項をまとめると以下のようになる。

(1) リッチなインターフェースであること

リッチなインターフェースとして求められるのは以下のような機能である。

- 表の表示(ソート、列の指定、行の絞り込み機能が付加できる)
- ツリー構造の表示
- グラフや図の表示
- Drag & Drop機能

このような機能は、通常のHTMLでは実現するのが難しい。こういったリッチなインターフェースを作成できること、もしくは部品等の追加・拡張により簡単に実現できることが必要である。

(2) 画面作成が直感的で生産性が高いこと

複雑な画面作成時には、部品をDrag & Dropで任意に配置でき、対応するソースコードも自動生成できる機能を持つGUIビルダーが使えることが望まれる。これにより、GUIの配置に関する操作性が向上するのはもちろんであるが、GUIを構築するためのソースコードの記述に関しても効率化が期待できる。

(3) 高品質で変更に強いこと

GUIを持つシステムを構築する場合、GUIとしての対話処理を行うロジックと、業務を行うビジネスロジックの分離が重要である。適切なアーキテクチャや実装方針が定められていないと、両者が混同したコードを記述したり、分離の仕方が開発者によって異なったりと混乱を招く要因になる。この結果、プログラム自体がいわゆるスパゲティ構造となり、品質が低下する。このため、適切なアーキテクチャに基づき、分離の方針をはっきり示すことが重要である。

また、GUIは頻繁に変更要求が起こる部分である。画面レイアウトの変更、画面操作に伴うアクションの変更、もしくは画面部品の変更など様々な変更要求にも柔軟に対応できることが望まれる。

(4) 陳腐化しない技術であること

企業システムは数年から十数年の寿命がある。したがって言語の寿命もこれより長いものでなければならない。そのためには、言語やライブラリが安定していること、継続してメンテナンスがされていることが必要である。また、言語やライブラリの提供元が安定していることも重要である。

(5) Webシステムのメリットを享受できること

Webシステムは、クライアント側プラットフォームの変化に対して影響を受けにくい。また、アプリケーションの配信およびアップデートを行うのが非常に容易であり、システムの導入、変更が容易である。これらのメリットはそのまま享受したい。

3. アーキテクチャ設計の方針

本章では前章で述べた要求事項を満たすアーキテクチャを設計するために設定した方針について、その検討内容も合わせて述べる。

3.1. 言語

言語は、技術の陳腐化という面で不安が少なく、当社で利用実績も多いJavaを選定した。ここでは、選定時に考慮した事項について述べる。

(1) 言語は少ないほうが良い

現状のWebシステム開発では、複数の言語知識が要求される。ベースがJavaのシステムであっても、Java以外に

HTML, JavaScript, XML, その他フレームワークに特有のタグライブラリの知識等が要求され、コーディングはもとよりデバッグについても各々方法が異なる。
この結果生じる開発者の負担を軽減するため、使用する言語は少ないほうが良い。

(2) 開発者を集めやすく、保守がしやすい

開発者を容易に調達するためには、学習がしやすく普及した言語であることが必要である。保守要員の調達に関しても同様である。最近のオープン系システムの大半がJavaで開発されていることもJavaを選定した理由の一つである。

(3) リッチなインターフェースを作成するためのGUIライブラリがある

Javaには通称Swingと呼ばれているJava Foundation Classes（以下Swing）があり、表やツリー構造の表示、Drag & Dropなど高機能なGUIを作成することが可能である[7]。これ以外に同じJava上で使用でき高速な動作が特徴のSWTがあるが、標準部品を継承して自前の部品を作ることができない問題があり除外する。

また、Swingは７年以上の実績があり、陳腐化しない技術といってよい。

3.2. クライアントの形態

クライアントには次の３つの形態がある。

- ブラウザ（HTML + JavaScript）
- ブラウザ + Plug-in（Java Applet）
- アプリケーション + 配信機能（Javaアプリケーション + Java Web Start）

ブラウザは、リッチな要求に応えられないか、JavaScriptの濫用となり開発生産性・保守性に問題がある。Plug-inタイプについては、Appletを実行するJava Runtime Environment（JRE）に関して、ブラウザbuilt-inではSwingが実行できない他、歴史的にも問題が多く敬遠されている。Applet使用を禁止しているユーザ企業もある程で除外する。これに対し、アプリケーションは画面構成に制限がなく、リッチなインターフェースを作成する自由度が大きい。

ただし、アプリケーションの場合、Webシステムのメリットを損なわないために、以下の機能が必要になる。

- ブラウザからアプリケーションを起動できること
- ブラウザで実行している業務からセッションの引継ぎが可能であること
- アプリケーションが配信可能であること

これらについてはJava Web Start（JWS）を用いて実現できる。

3.3. GUIビルダーの使用

直感的な画面作成を可能にするためにGUIビルダーを採用する。また、GUIビルダーは画面構成に関するコードを自動生成するので、生産性も向上できる。

3.4. アーキテクチャとフレームワーク

基本的なアーキテクチャとして、MVCを採用した[3)4)5)]。M（Model）はビジネスロジックを司り、V（View）は画面を司り、C（Controller）は入力の制御を司る。GUIとしてのLook&FeelはVとCで提供される。変更が頻繁に行われる画面構成（View）および入力操作に伴う処理（Controller）と、ビジネスロジック（Model）をそれぞれ分離することにより、見通しが良く変更に強いアーキテクチャにすることができる。

また、フレームワークの設計に際しては、品質向上のため共通的な処理を最大化すること、維持性向上のため部品の組立て・再構成が容易な仕組みを提供すること、の２点に留意する。

3.5. DIコンテナ

部品の組立てを実現する機構として、DI（Dependency Injection）コンテナを使用し、プログラムの処理をハードコーディングするのでなく、できるだけ宣言的に記述された設定ファイルで行う方針とした。依存性を設定ファイルで記述することで、部品化した相互に非依存なクラスの組立て方法が明確にできる。

さらに、AOP（Aspect Oriented Programming）機能と組み合わせることで、ソースコードを変更することなく機能を追加できる。DIとAOPの組み合わせは、最近のDIコンテナでは一般的になってきた[12) 13)]。

AOPは、プログラムコード中に散在して現れるログやトランザクションの管理など、本流のロジックとは無関係な横断的関心事（cross cutting concern）をコードから切り離し、本流のロジックに集中してもらうための仕組みであ

る[11]。横断的関心事の処理を機能として織り込む場所については、変更要求が発生した時にプログラムコードに影響を与えないよう、自動生成されるコード内とする。

以上述べたDIとAOPにより、各部品に手を加えずに、部品の組立てを自由に、理解しやすく記述できる。これにより、変更に強いシステムにすることができる。

4. アーキテクチャの設計

この章では、３章の方針に基づいて設計したアーキテクチャについて述べる。アーキテクチャの説明に先立ってまずGUIビルダーの選択について説明し、その後MVCアーキテクチャの全体像とフレームワーク部分の役割について記述し、最後に、配信機能にまつわる解決策について述べる。

4.1. NetBeans IDEのMatisse GUIビルダー

NetBeans IDEはMatisse GUIビルダーになってVBに匹敵する操作性・簡易性を実現できた[6]。画面作成は、Matisse GUIビルダーで行い、生成された画面構築用のコードをそのまま使用することにした。これにより、画面構築の容易性を確保し生産性の向上を図る。以下、操作性と簡易性を実現するために導入された技術について簡単に紹介する。

4.1.1. Group Layoutと直感的操作

Swingでは複数のウィンドウシステムに対応するために、LayoutManagerを使用して部品配置の差異を吸収する設計になっている。また、LayoutManagerは複数利用することができ、配置のポリシーによって使い分けるようになっている。ところが従来のものでは、部品配置前にLayoutManagerが計算する画面レイアウトを予想し難く、グラフィカルな開発環境の下でも、直感的に部品を配置することが難しかった。

Matisseでは、以下の機能を追加することにより上記の問題を解決した。

(a) Group Layout（図１）
- 縦または横配置のグループ化
- グループの階層化(入れ子)

(b) マージン(Gap)の設定（図２）
- 部品間のマージン
- フレームの端からのマージン

この他、フォントのベースラインに合わせて水平方向の配置を決めることもでき、視覚的に自然な配置が行える。

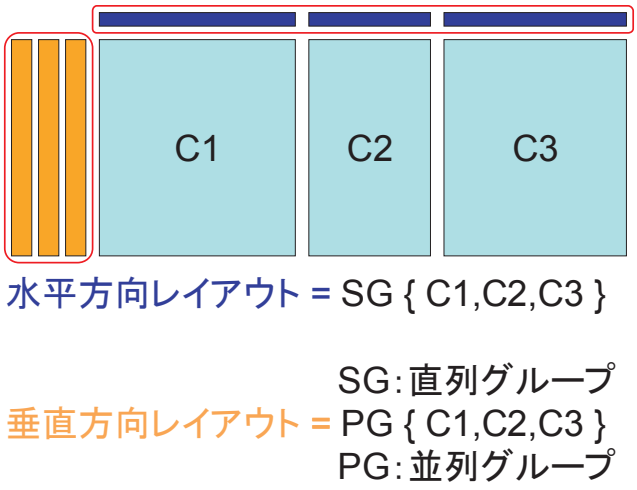


図１　Group Layout

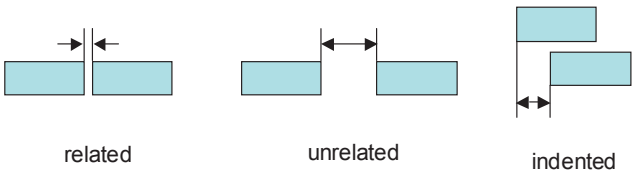


図２　Gapの種類

4.1.2. NetBeans IDE

NetBeans IDEのMatisse GUIビルダーは、上述のGroup Layout、Gapの機能を最大限に引き出すように設計されており、配置画面上で補助線を表示することによりLayout、Gapおよびベースラインの設定を直感的に行えるようになっている。（図１，図２）

このため、従来のJava系IDEにはなかったVBライクな直感的な操作が可能になった。将来的には他のJava系のIDEもこの流れに沿っていくものと思われる。

4.2. アーキテクチャの設計

設計したMVCアーキテクチャを図３に示す。ここで、アプリケーション領域とはユーザが作成するかGUIビルダーによって自動生成される部分、フレームワーク領域とは本システムで提供するフレームワーク部分である。MVCの

各部はアプリケーション領域とフレームワーク領域にそれぞれまたがった形になる。

例えば、View部では、クライアントの画面に対応する個々の画面部品のインスタンスはアプリケーション領域に配置される。これらのインスタンスを統一的に管理するための機構となる「静的構造」をフレームワーク領域で提供し、全体でViewの役割を担っている。後述するが、ControllerとModelについても同様で、本システムの特徴的な構造になっている。

これにより、アプリケーション毎に異なるインスタンスをフレームワーク領域のAPIを通じて統一的に扱うことを可能にした。また、後述するDIコンテナや状態遷移機構を利用して、宣言的に依存関係を記述できるようにもなる。

4.2.1. View部と静的構造

View部の画面部品は、画面の土台とするフレームの上に、部品を意味的にまとめて構成したパネルを置き、そこに個々の画面部品を配置する、という階層的なツリー構造になる。これはMatisseによって自動生成され、アプリケーション領域に属する。

フレームワーク領域に属する静的構造は、この階層構造を保持し、動的構造やアプリケーション領域からの要求を受け、指定された部品のインスタンスを返す、という機能を提供する（図４）。

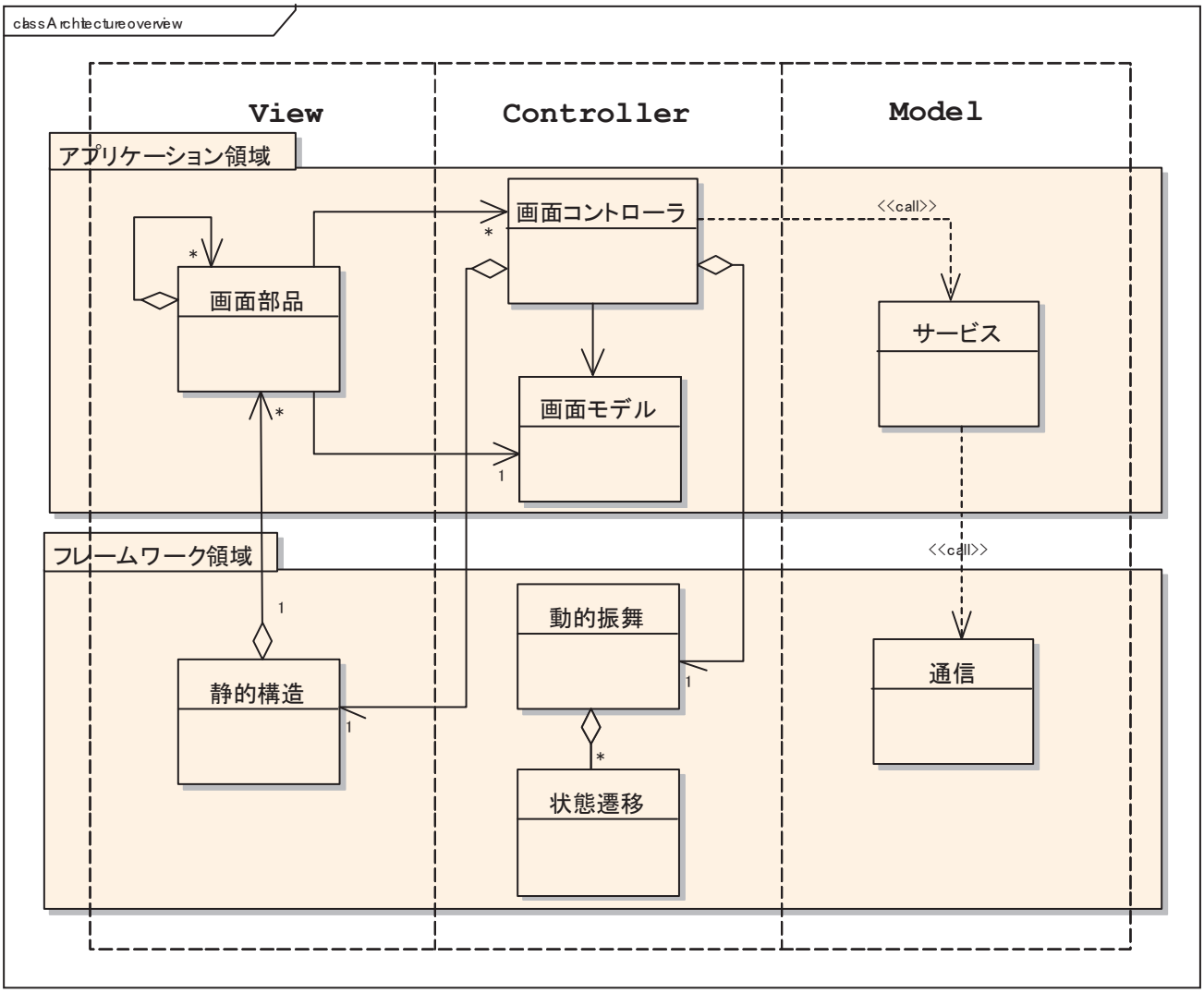


図３　アーキテクチャ全体像

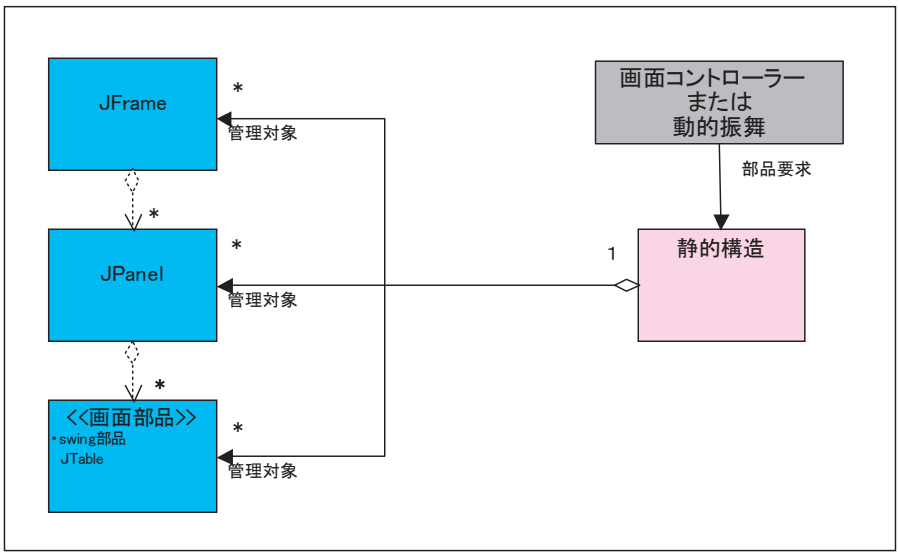


図４　静的構造

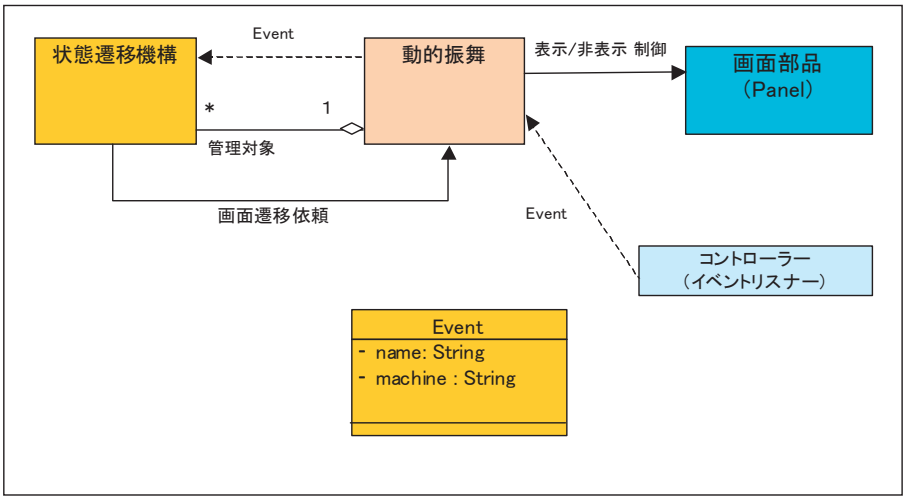


図５　動的振舞い

4. 2. 2.　Controller部と動的振る舞い

Controller部のアプリケーション領域は､画面コントローラと画面モデルに分かれる。これはどちらもユーザが記述する。

画面コントローラは、動的振る舞いを保持し、実際の動作を行うロジックを記述する。ロジックには、Model部の呼出しや、画面部品へのイベントの通知、ポップアップ画面の管理などがある。画面コントローラは､個々のGUI部品に対応するものではなく、複数のGUI部品を一つのグループとして保持・管理する画面部品であるパネルに対応して作成する。

画面モデルは、画面部品が表示するデータを保持するクラスで、Swingの画面部品APIで規定されているものである。

フレームワーク領域の動的振る舞いは、後述する状態遷移機構のインスタンスを保持しており、画面コントローラはこの情報から実際の画面遷移の実行も行う(図５)。また、静的構造と画面モデルを利用して、実際に画面に表示されるデータを設定する等の動作も行う。

通常Swingのプログラミングは、画面部品と画面コントローラ、画面モデルの３者を使って行うが、GUIビルダーを利用すると画面部品に画面コントローラが組み込まれてしまう。本システムではビルダーの機能を使わずに画面コントローラを強制的に独立したクラスにすることでMVCの分離を実現した。

4.2.3. Model部

Model部はビジネスロジックを実行する。通常はサーバに実行を任せるが、クライアントで完結することもできる。サーバで実行する場合には通信が必要であり、最も汎用性の高いHTTP通信用のライブラリを作成した。

4.2.4. 状態遷移機構

画面遷移やワークフローを柔軟に実行できるように、状態遷移機構を開発した。

状態遷移機構は、XMLファイルに状態遷移の情報を記述し、それを読み込んで、イベントの動作に合わせて実行するフレームワークである(図６)。XMLファイルの記述方法は､W3Cが制定している汎用的な状態遷移記述形式であるState Chart XML(SCXML)のサブセットとした[14]。

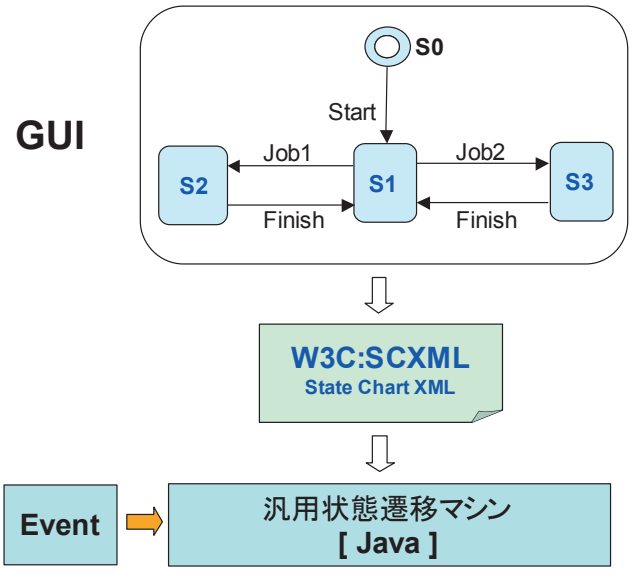


図６　状態遷移機構

4.3. 組立てツールとしてのDIコンテナ

4.2節で説明したアーキテクチャを組立てるツールとしてはAOP機能を持つDIコンテナが適当であるが、後述のScripting機能を満足するDIコンテナがないため、新規に開発した。

まず、DIコンテナにより依存性を持つことなくMVCの各部品を接続することができる。依存性は、外部ファイル(XML)に定義して実行時に注入(inject)される。(図７，図８)

結果としてXML形式でコンポーネントの依存性が宣言的に記述される。

また、AOPについては、本システムでは、Matisseの自動生成したコードを書き換えることなく、Controller呼出し等のコードを織り込むために使用した。これにより、Matisseが自動生成したコードに手を入れずに利用することができ、自動生成したコードと手書きのコードを分離することができる。

さらに、Script機能をサポートした。Java SE6から、JSR-223 (Scripting for Java Platform)が盛り込まれ、スクリプト言語が使用できるようになった[10]。Javaとスクリプト言語は相互に呼出すことができる。DIコンテナを糊(glue)としてモジュールを接着し、その間のメッセージのやり取りを行うことができる(図９)。

このScript機能はDIコンテナと状態遷移機構の設定ファイルに適用し、以下を実現した。

- DIコンテナによるインスタンス生成順序の制御(図10)
- DIコンテナによるコンテナ外部のインスタンス取得(図10)
- 状態遷移機構による定義ファイルのアクション実行(図11)

なお、実装に当たっては、Java SE5との互換性を考慮し、今回はJSR-223の機能は使わずにBeanShellを用いて行った[8]。

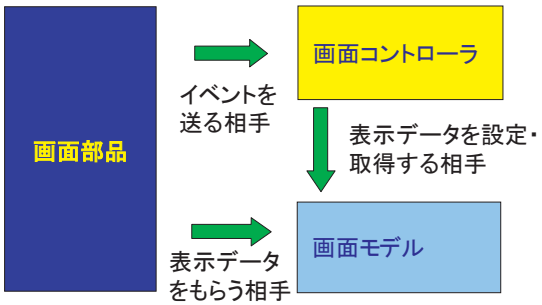


図７　部品間の依存性

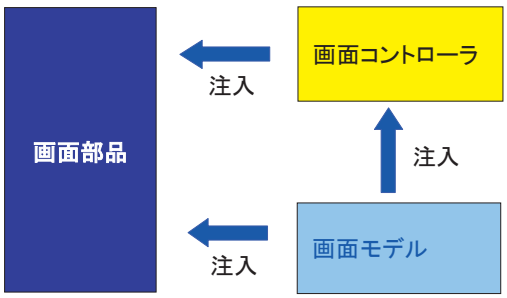
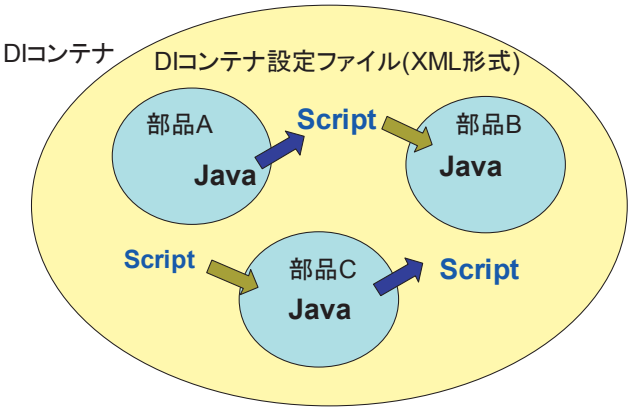


図８　DIコンテナによる依存性の注入



DIコンテナを介してJavaとスクリプトが連携

（スクリプトはXML と容易に組み合わせることができる）

図９　Java とスクリプトの連携

オブジェクト生成の順番を管理

```
<component name="jFrame" script="new JFrame()"/>
<component name="jPanel" script="jFrame.getJPanel">
<binding>jFrame</binding>
</component>
<component name="jTree" script="jPanel.getJTree">
    <binding>jPanel</binding>
</component>
```

スクリプトで外部のインスタンス(ここではモデル)を設定

```
<component name="jTree" script="jPanel.getJTree">
    <binding>jPanel</binding>
    <operation script="setTreeModel( treeModel )"/>
</component>
<compoent name="treeModel" script="new DefaultTreeModel()"/>
```

図10　Script機能を利用したDIコンテナ設定ファイル例

スクリプト

```
<onexit>
    <action script='x.setInvisible( y )'>
        <binding name="behavior" arg="x"/>
        <binding namespace="job1" name="job11" arg="y"/>
    </action>
    <action script='x.setForeground( Color.BLUE )'>
        <binding namespace="menu" name="jLabel1" arg="x"/>
    </action>
</onexit>
```

DI コンテナーから名前を指定して Java オブジェクトを引き当てる

図11　Action記述にScript機能を利用した状態遷移設定例

4.4.　Java Web Start (JWS)とセッションの引継ぎ

配信に使用するJWSの機能と、フレームワークの一部として開発したブラウザからのセッションの引継ぎ機能について記述する。

4.4.1.　Java Web Start (JWS)

JWSは、Java プログラムを Web ブラウザから起動し、Javaアプリケーションの形態で実行する技術である。Java SE 5以降では標準で利用可能である。（図12）

これによりWebシステムと同じようにアプリケーションの配信が行え、リッチなGUI の恩恵を享受できる。

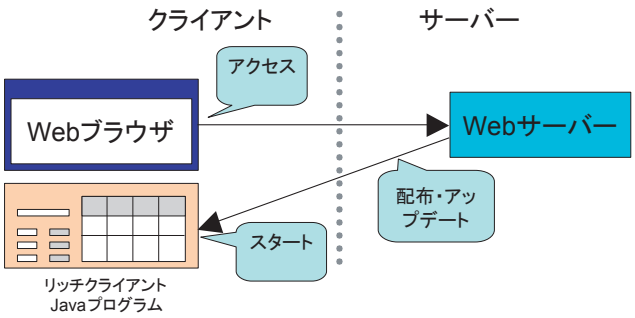


図12　Java Web StartによるJavaプログラムの配布と起動

4.4.2.　セッションの引継ぎ

JWSには、Webシステムで実行されていたセッションを引継いでJavaアプリケーションに渡す機能は提供されていない。

JWS機能の仕様はJava Networking Launching Protocol & API(JSR-56)において規定されている[9]。アプリケーションの起動にはWebページからXMLで記述されたJNLPファイルへリンクを張る。このJNLPファイルでアプリケーションの起動条件を指定できる。ここにセッション情報を記述してアプリケーションに渡せばよい。具体的には、セッション毎に変わるセッション情報に対応して、JNLPファイルを動的に生成することで、セッションの引継ぎを実現した。

5.　検証と考察

本章では、検証に用いたアプリケーションを簡単に説明した後、検証項目を確認し、検証結果と考察および課題について述べる。

5.1.　検証に使用したアプリケーション

社内向けWebシステムとして実際に稼働している、社員

の技術スキルを管理する「技術マップ」と呼ばれるアプリケーションで検証を行った。

検証においては、サーバ側のビジネスロジックはそのまま使用した。Webのフロントエンド部は変更し、本システムで作成したリッチなJavaアプリケーションからHTTP経由でサーバ側のサービス層に接続し、ビジネスロジックを実行できるようにした。

検証では全22ユースケースのうち、画面が複雑な更新と参照に関する２ユースケースをクライアント側に実装した。

アプリケーションの設計、製作の過程および動作確認により、後述する検証項目をクリアしていることを確認した。

5.2. 検証項目

今までの議論から、設計や技術の選択において解決されているものを除き、検証項目としては、以下のものがある。

- 生産性に関わるもの
  - (1) 画面作成の容易性
  - (2) 画面コードの自動生成
  - (3) その他の自動生成
- 品質と保守性に関わるもの
  - (4) MVCの分離
  - (5) 変更容易性
- 機能に関わるもの
  - (6) 配信機能とセッションの引継ぎ

各々について次節で結果と考察を述べる。

5.3. 検証結果と考察

(1) 画面作成の容易性

これについては検証アプリケーションの画面を、Matisseを用いて作成した時の使用感を持って検証とするが、操作感については満足のいくもので、画面作成が非常に容易に行えた。前述したように、これはGapとGroupLayoutを表す補助線を表示する機能によるものである。

また、自作した部品やオープンソースの部品を自由にMatisseに登録できるので、標準でない画面部品を使用しても、標準部品と同じ操作感で容易に画面作成が行える。

(2) 画面コードの自動生成

Matisseが自動生成した画面部品の生成コードの割合を測定することで検証した。

サーバとの通信部分を除いたクライアント側コードにおいて、コメントを除くソース有効行は1705行であったが、このうちMatisseにより生成された画面生成のコードは約24%(416行)であった。

通信を除くクライアント部分はシステム全体のプレゼンテーション層といえるが、そのうち約1/4を自動生成できたことになる。このことから画面生成をGUIビルダーに任せる方針の正しさと、生産性への寄与が検証できた。

(3) その他の自動生成

その他の自動化に向けた考察として、自動生成可能なものを以下に示す。

(a) DIコンテナ設定ファイル

(b) 状態遷移機構設定ファイル

これら設定ファイルの記述内容を自動的に導出または補完するツールを作成したが、(a)については、コンポーネントの依存性の記述を導出や補完により支援するだけでなく、スクリプト機能で記述するコンポーネント生成を対話的に支援する機能が有効であった。(b)については、状態遷移表から設定ファイルを自動生成するバッチ的なツールが有効であった。

(4) MVCの分離

設計におけるMVCの分離を徹底するために、プログラミングガイドとして以下の規約を定めた。

- コントローラの実装に当たっては、GUIビルダーの機能は使わず、画面部品とは独立したコントローラのクラスを作ること。
- 画面モデルと画面部品に業務的なメソッドを含めたい場合には、そのインターフェースを定義し、画面モデルまたは画面部品を継承して業務的なロジックを記述すること。
- 画面モデルを作成する場合は、画面部品および画面コントローラに依存しないこと。

最初の項目は、強制的にDIを使用させるために定めている。DIコンテナを使うことにより、基本的には、依存関係を持たないものを作成し、設定ファイルに宣言的に依存関係を記述するので問題は生じにくく、確認しやすい。２番目の項目は業務ロジックの分離を図り、３番目の項目は画面モデルの独立性を高めることで、いわゆるスパゲティ構造を避けるための規約である。

また、設計やコーディングのミスにより、MVCの依存関係に矛盾を生じ、相互に依存する実装になってしまった場合、DIコンテナの実行時に検出されるケースもあり、組

立てる時点で修正できる。

本システムを用い、これらの規約に従いコーディングを行った結果、スパゲティ構造に陥らず、MVCの分離を完全に行うことができた。

(5) 変更容易性

典型的な例として、画面部品を入れ換える変更要求があった場合を想定し、以下の単純な手順だけで変更ができることを検証した。

(a) 変更後に使う画面部品を用意し、Matisseにより画面構成を変更する。
(b) 対応する画面コントローラ(必要であれば加えて画面モデル)を作成または変更する。
(c) DIコンテナ(必要であれば加えて状態遷移)の設定ファイルを書き換える。

ここで画面部品は単一のGUI部品でも複数のGUI部品を集めたパネルでもよい。どちらの場合も影響範囲が特定しやすく、Modelや他の部品には影響を及ぼさず、変更が容易にできる仕組みが提供できた。

(6) 配信機能とセッションの引継ぎ

動作確認により検証を行ったが、新たな問題として、完成したアプリケーションをJWSで配信できるようにするには、電子署名が必要であることがわかった。

JWSアプリケーションは、セキュリティ上ローカル資源へのアクセスが制限され、ネットワーク接続はダウンロード元のホストに限定される。この制限を超える場合はセキュリティ設定が必要であり、セキュリティ設定を行うアプリケーションは、必ず署名されたjarファイルで配布される必要がある。

ところがアプリケーションが使用しているJava SE 6中のJAX-Bライブラリの中で、システムプロパティを変更するものがあった。これが上記の制限に違反しておりエラーとなるため、実行できない。

このため、jarファイルに署名をつけることで対処した。アプリケーションがローカル資源へのアクセスをしている訳ではないので本来署名は必要ない。JAX-Bライブラリの実装の改善が望まれる。

セッションの引継ぎについては、Web画面で認証を行った後に、JWSによりアプリケーションを起動しセッション情報を引継いで起動させることで検証し、問題なく動作した。

5.4. 課題

今後の課題としては以下の４点がある。

(1) リッチな部品の開発

リッチクライアントでは、部品の機能が、その部品を使った画面の機能を決める。したがって、Swingで提供されていない機能については部品の作成が必要である。汎用的な部品を作れば再利用により生産性もさらに向上する。このような汎用的部品をあらかじめ用意しておけば、便利である。

(2) 部品ライブラリの収集

上記のような部品は、一般にオープンソースとして公開されていることも多い。公開された部品を集めて整理し、状況に応じてすぐに使えるようにしておくことも重要である。

(3) 開発工数の検証

生産性の検証として、開発工数の比較を行っていない。通常の開発と比較し、生産性が計測できるとよい。

(4) Model部の自動生成との連携

これまでの研究成果であるMDAによるModel部の自動生成の取り組みと連携し、サーバ側も含めたシステム全体の開発生産性、品質の向上について相乗効果が得られるとよい。

6. おわりに

本論文では、リッチクライアントを実現するSwingによるフレームワークの開発について述べた。構築に当たって画面作成の容易性、生産性、品質、変更容易性を考慮し、要求事項を達成することができた。

リッチクライアント向けフレームワークは百花繚乱の如く発表されているが、決定的なものは出ておらず、提案したフレームワークによる手法が有効な解決策となることを期待している。

今後は、5.4節で述べた課題の解決を図っていくと同時に、このフレームワークで得られたノウハウを生かし、MDAによるソフトウェア開発の生産性向上に寄与していく所存である。

参考文献

1) 土屋　正登, "ビジネスアプリケーション分野におけるMDAの限界と活用", exa review vol.6,　P1-8, 2006.6
2) 小松　清希, "MDA技術の効果と課題 - オープンソースモデル・トランスフォーマ AndroMDA -", exa review vol.5, P77-84, 2006.12
3) Martin Fowler et al., "Pattenrs of Enterprise Application Architecture", Addison-Wesley, 2003
4) Frank Buschmann et al., "Pattern- Oriented Software Architecture: A System of Patterns", John Wiley & Sons, 1996
5) Eric Gamma et al., "Design Patterns - Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995
6) Sun Microsystems, Inc.& Collabnet, Inc., "NetBeans", <http://www.netbeans.org/>
7) Sun Develper Network (SDN), "Java SE Desktop Technologies", <http://java.sun.com/javase/technologies/desktop/techoverview.jsp>
8) Patrick Niemeyer, "BeanShell", <http://www.beanshell.org/>
9) Java Community Process, "JSR-000056 Java Network and Launching Protocols & API (Final Release)", <http://jcp.org/aboutJava/communityprocess/final/jsr056/index.html>
10) Java Community Process, "JSR-000223 Scripting for the Java Platform (Final Release)", <http://jcp.org/aboutJava/communityprocess/final/jsr223/index.html>
11) AOP Aliance, "AOP Aliance", <http://aopalliance.sourceforge.net/>
12) The Seasar Foundation et al., "Seasar Project", <http://www.seasar.org/>, 2007
13) Interface21, "Spring Framework", <http://www.springframework.org/>
14) W3C, "State Chart XML (SCXML): State Machine Notation for Control Abstraction", <http://www.w3.org/TR/scxml/>, 2007
