PACパターンによるWebアプリケーション開発

19811166 鈴木鉄也 (諸橋ゼミ)
19811214 堀田　誠 (松谷ゼミ)

多摩大学
経営情報学部経営情報学科
2001年度卒業論文

目次

1. はじめに

現在、Webアプリケーションの開発ではアーキテクチャにMVC(Model-View-Contoller)パターンを使うことが多い。例えばサーバサイドJavaを用いた開発では、EJB(Enterprise Java Beans)-サーブレット-JSPをそれぞれModel-View-Controllerとして扱う開発が主流になっている。

MVCパターンはSmalltalkの開発環境から生まれた。もともとは単純なアーキテクチャを想定していたため、複雑なインターフェースの対話型システムには向いていないが、最近では複雑なインターフェースに対応するためのパターンも数多く生まれている。複雑なGUIアプリケーション向けにMVCパターンを拡張したMVC++パターン、JavaAPI(Swing)のために拡張したMVCパターン、MVCパターンと異なるアプローチから対話型システムを構築するPAC(Presentation-Abstraction-Control)パターン等がある。

デスクトップ同様、Webシステムをとりまく環境も発展し、アーキテクチャの重要性が高まってきている。単純だったインターフェースは次第に複雑になり、高度な機能を要求されるようになった。また、開発方法や期間、使用するツールによって選択するアーキテクチャは異なってくる。Webアプリケーションの開発方法や各社Webアプリケーションサーバ製品のアーキテクチャを研究するうちに、MVCパターンがWebアプリケーション開発に最適な方法であるのかどうか疑問を持った。

そこで注目したのがPACパターンである。PACパターンは対話型システムを「協調するエージェントの集合」と捉えており、システムの変更や拡張に強い。複雑なインターフェース、様々なコンテンツ(コンポーネント)の集合、スパイラル型の開発体制といった特徴を持つWebシステムに向くのではないかと考えた。

このレポートでは、Webアプリケーション開発におけるPACパターンの適性を検討する。はじめに、MVCパターンとの比較を交えつつPACパターンとWebシステムのアーキテクチャを確認する。以上を踏まえてWebアプリケーションにPACパターンを適用し、最後に実装を行う。

2. Presentation-Abstraction-Control パターン

PACパターンは、システム全体をエージェントと呼ばれるコンポーネントで構成する(図2-1)。さらにエージェントを３つのコンポーネントに分割することによって、人間とコンピュータの対話機能を分離する。各エージェントはControlコンポーネントを通じて通信を行う。

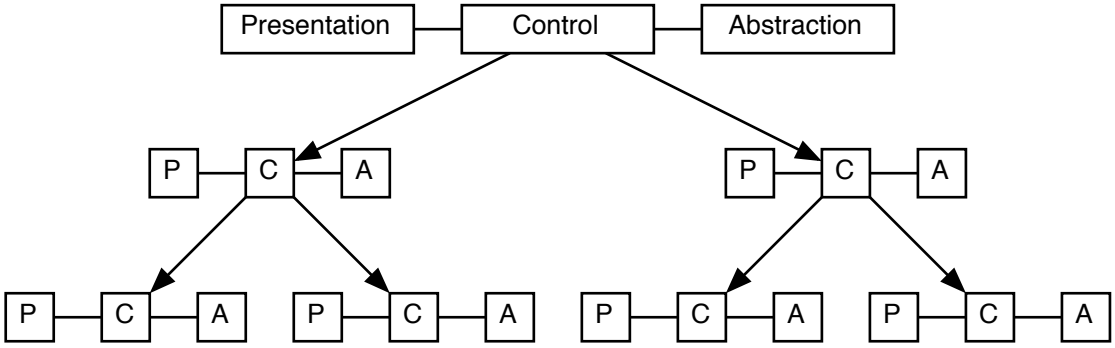

図2-1 PACパターンの構造

2.1 Presentation-Abstraction-Control

エージェントは３つのコンポーネントで構成する。３つのコンポーネントとは、Presentationコンポーネント、Abstractionコンポーネント、Controlコンポーネントである(図2-2)。各エージェントが必ずPACコンポーネントをすべて持つとは限らず、エージェントによってはPresentationコンポーネントやAbstractionコンポーネントを持たないこともある。

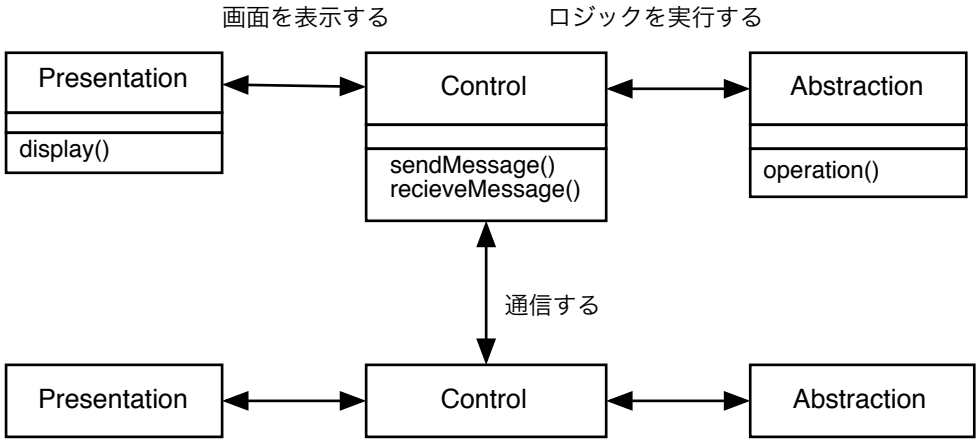

図2-2 Presentation-Abstraction-Control

2.1.1 Presentationコンポーネント

ユーザーインターフェースを提供する。エージェントはそれぞれ独自のインターフェースを提供し、複数のエージェントを組み合わせて画面を構成することもある。

2.1.2 Abstractionコンポーネント

ロジックやデータを管理する。Abstractionコンポーネント内のデータはメディアに依存しない。例えば図のデータを保持するとき、ピクセルではなくセンチメートルやインチといった単位を使う。メディアに依存しないことで、エージェントの移植性が高まる。

2.1.3 Controlコンポーネント

PresentationコンポーネントとAbstractionコンポーネントを接続する。また、他のエージェントのControlコンポーネントと通信を行う。

2.2 エージェント階層

PACパターンでは、システムをエージェントで木階層に構造化する。エージェントはその役割によって「トップレベルエージェント、中間レベルエージェント、ボトムレベルエージェント」の３つに分かれる(図2-3)。各エージェントはControlコンポーネントによって通信する。

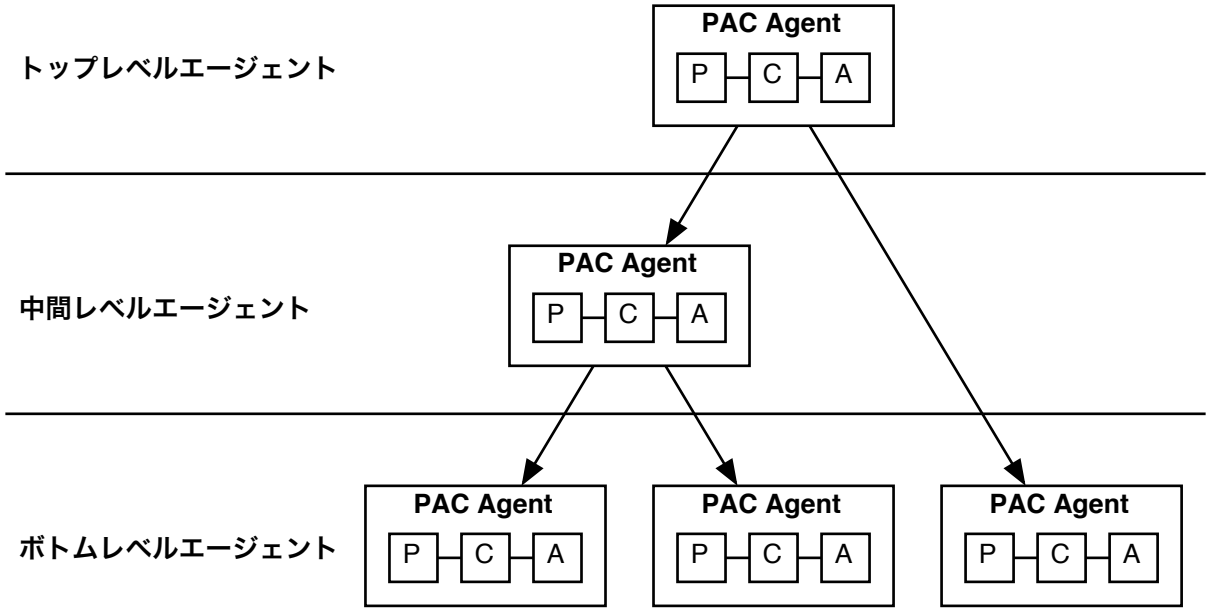

図2-3 エージェントの階層

うまく構造化されたシステムは、頂点にトップレベルエージェントを1つ持ち、下位の階

層ほど多くのエージェントを持つピラミッド型のエージェント階層になる。

2.2.1 トップレベルエージェント

重要な機能やシステム全体に共通する機能など、システムの中核となる機能を提供する。また、エージェントの階層を制御する。

Presentation
: メニューバーやアプリケーション情報を表示するダイアログボックスなど、システム全体に共通するユーザーインターフェースを提供する。

Abstraction
: システム全体に共通するデータや、中核となる機能を提供する。

Control
: 中核機能を操作するためのサービスを下位のエージェントに提供する。また、エージェントの階層を調整し、ユーザーとシステムの相互作用についての情報を管理する。

2.2.2 中間レベルエージェント

下位階層のエージェントの組み立てと調整を行う。

Presentation
: エージェント固有のデータへのインターフェースを提供する。

Abstraction
: 下位階層のエージェントの組み立てと調整のためのロジックを持つ。

Control
: トップレベルエージェントやボトムレベルエージェントのControlコンポーネントと同じく、PresentationコンポーネントとAbstractionコンポーネントの仲介や上下階層エージェントとの通信を行う。

2.2.3 ボトムレベルエージェント

システムの機能の一部を提供する。ボトムレベルエージェントの機能は様々であり、入力したデータからグラフを作るといった単純なものから、システムの全てのデータを管理するような複雑な機能になることもある。

Presentation

エージェント固有のデータへのインターフェースを提供する。

Abstraction

エージェント固有のデータを管理する。トップレベルエージェントのAbstractionコンポーネントとは違い、ボトムレベルエージェント固有のデータに依存するエージェントはない。

Control

PresentationコンポーネントとAbstractionコンポーネントを仲介し、これらのコンポーネントが互いに依存することを防ぐ。また、上位の階層のエージェントと通信する。

2.3 処理の流れ

各エージェントがすべてのコンポーネントを持つとは限らないため、エージェントによって処理が異なる。ここでは、リクエストを受け取り、データを変更してから画面表示を行う処理を例にとる(図2-4)。

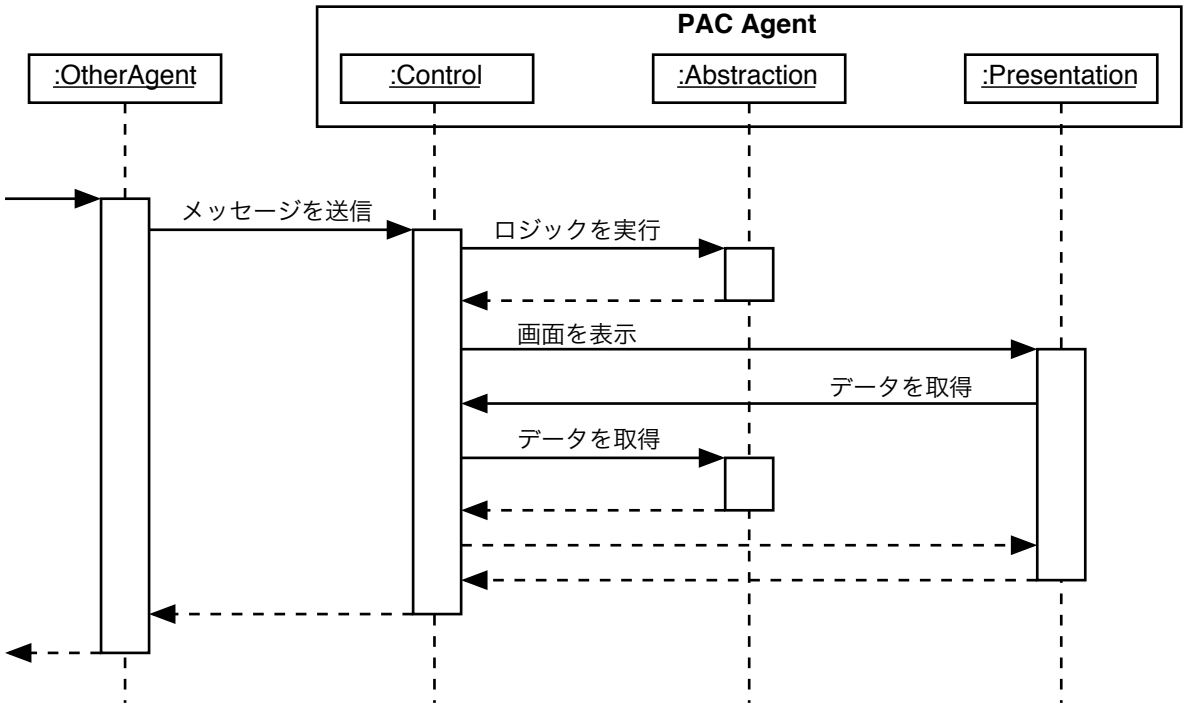

図2-4 PACエージェントの処理の流れ

ユーザーはPresentationコンポーネントを通してデータの入力を行う。Presentationコンポーネントは入力されたデータを基に、Controlコンポーネントにリクエストを転送す

る。Controlコンポーネントは Abstractionコンポーネントにリクエストを送ってロジックを実行し、データを変更する。データの更新が終わったら、画面を表示するためPresentationコンポーネントにリクエストを送る。Presentationコンポーネントは、画面表示に必要なデータを取得するためにControlコンポーネントにリクエストを送り、Abstractionコンポーネントからデータを取得する。Presentationコンポーネントは取得したデータを基に画面を表示し、エラー等の結果をControlコンポーネントに返す。最後にControlコンポーネントが呼び出し元のエージェントにエラー等の結果を返して終了する。

2.4 特徴

2.4.1 メリット

データの独立

ControlコンポーネントがPresentationコンポーネントとAbstractionコンポーネントを仲介し、他エージェントとの通信を管理することで、各エージェントとコンポーネントのデータモデルを独立して開発することができる。

変更と拡張の支援

一部のエージェントのPresentationコンポーネントやAbstractionコンポーネントを変更しても、他のエージェントに影響を及ぼさない。このため、ユーザーインターフェースの変更やチューニングを独立して行うことができ、他エージェントをシステムに追加することも容易になる。

マルチタスクの支援

エージェントを異なるスレッド、プロセス、マシンに分散させることが容易になる。その場合は、Controlコンポーネントに通信機能を持たせるだけで、他のコンポーネントに影響はない。

2.4.2 デメリット

複雑なシステム

あまり細かく機能を定義すると、エージェントが増えてシステムが複雑になる。ボトムレベルエージェントをまとめるには中間エージェントが必要になるので、エージェントの設計は環境を十分に考慮する必要がある。

複雑なControlコンポーネント

Controlコンポーネントは、他のコンポーネントや他のエージェントと通信を行う。そのため、Controlコンポーネントのパフォーマンスがシステム全体のパフォーマンスに大きく影響する。Controlコンポーネントの役割を明確に決めて実装するべきである。

効率性

エージェント間の通信のオーバーヘッドがシステムのパフォーマンスを下げる可能性がある。これは深刻な欠点になる可能性が高く、PACパターンを採用するときには十分な判断を行わなければならない。

適用可能性

アプリケーションのドメインが小さいほど、ユーザーインターフェースの類似性が大きいほど、PACパターンの適用性は小さくなる。逆に、広範囲のドメインや独自のインターフェースを提供するアプリケーションはPACパターンに向く。

2.4.3 Model-View-Controller パターンとの比較

MVC(Model-View-Controller)パターンは、対話型システムのアーキテクチャを定義するとPACパターンと比較されることが多い。

MVCパターンは３つのコンポーネントで構成する。ロジックとデータを持つModelコンポーネント、画面表示を行うViewコンポーネント、入力されたデータをModelコンポーネントに渡すControllerコンポーネントである(図2-5)。

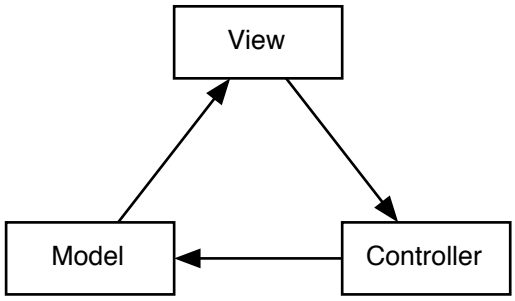

図2-5 MVCパターンの構造

MVCパターンの特徴は更新伝播メカニズムである(図2-6)。クライアント(Observer)がControllerにデータ更新を要求すると、Controllerは受信したデータを必要に応じて変換し、Modelに渡す。Modelは自身のロジックを実行してデータを変更し、Viewにデータ更新を報告する。最後にVIewがインターフェースを表示して終了する。

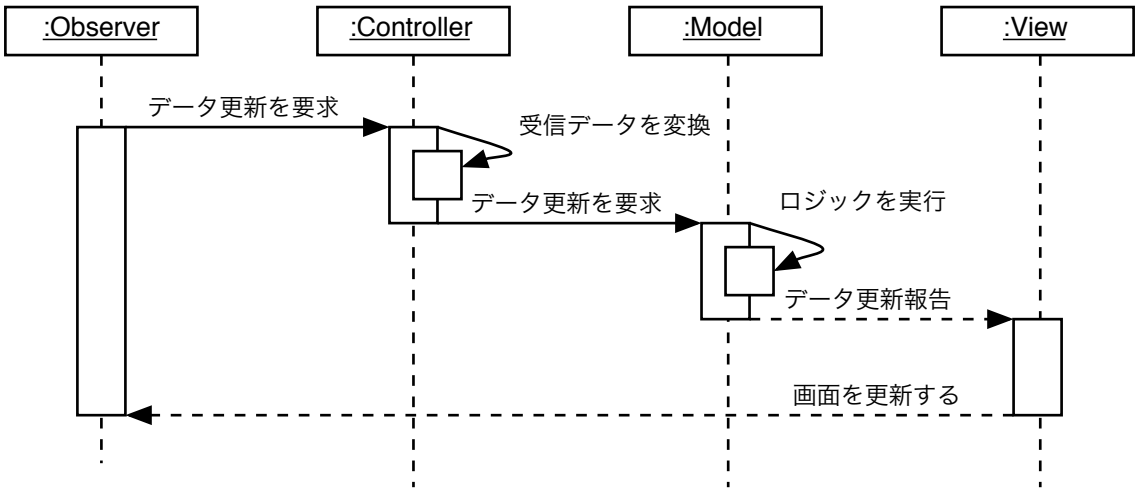

図2-6 MVCパターンの処理の流れ

MVCパターンとPACパターンの違いを挙げる。

インターフェースとデータモデルの依存関係

PACパターンではPresentationコンポーネントとAbstractionコンポーネントが独立しているが、MVCパターンではViewコンポーネントがModelコンポーネントに密接な関連を持つ。

仲介コンポーネントの役割

MVCパターンではControllerコンポーネントが、PACパターンではControlコンポーネントがそれぞれ他のコンポーネントを仲介する。しかし、MVCパターンでは他コンポーネントを直接操作することはない。各コンポーネントはデータ更新のメッセージを送るだけである。

アーキテクチャ

PACパターンではシステムをエージェントの階層に構造化するが、MVCパターンでは、Model-View-Controllerの3つのコンポーネントに構造化する。システムを拡張する場合は3つのうちどれかのコンポーネントに追加する。

大きく拡張する場合や大規模なアプリケーションの場合、システムをMVCコンポーネントで分割し、PACパターンのようにコンポーネント間の通信をContorllerコンポーネントで行う方法もある(図2-7)。

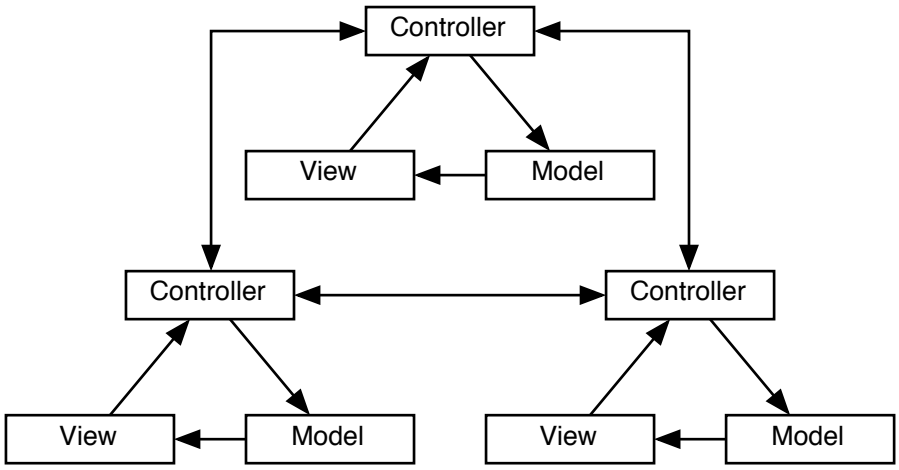

図2-7 Controllerによるコンポーネント間の通信

3. Webシステムのアーキテクチャ

3.1 クライアント-サーバ型ネットワーク

Webシステムは、クライアント-サーバ型のネットワークで構成する(図3-1)。クライアント-サーバ型ネットワークとは、クライアントがサーバにリクエストを送り、サーバがレスポンスを返す仕組みである。Webシステムでは、ブラウザとWebサーバがクライアント-サーバに相当する。

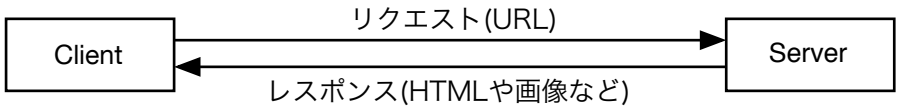

図3-1 クライアント-サーバの構造

例えば、クライアントがサーバにデータ(URL)を要求すると、サーバがクライアントにデータ(HTMLや画像など)を送信して画面を構成する(図3-2)。

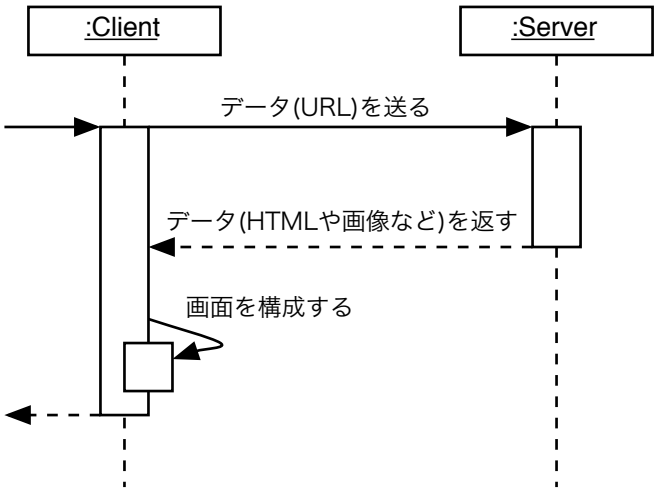

図3-2 クライアント-サーバの処理の流れ

3.2 特徴

3.2.1 HTTP(Hyper Text Trasfer Protocol)

Webシステムでは、クライアント-サーバ間の通信にHTTP(Hyper Text Transfet Protocol)というプロトコルを使用する。HTTPは主にHTMLを送信するためのプロトコルである。このプロトコルは、一回の通信ごとに途切れる。そのため部分的に画面を更新することはできず、通信のたびにすべての画面を構成しなければならない。

3.2.2 Webページ

Webサイトは統一感を出すための基本レイアウトを持つ。たいていのWebサイトでは、Webページにおいてナビゲーションや著作権表記等を表示する箇所が決まっている(図3-3)。

図3-3 Webページの構造

従って、Webサイトは基本レイアウトに沿ったWebページの集合体ということになる(図3-4)。各ページで共通する部分が多く、1ページごとにすべてのHTMLを組み立てるとなると、当然余分に生成する部分が発生する。

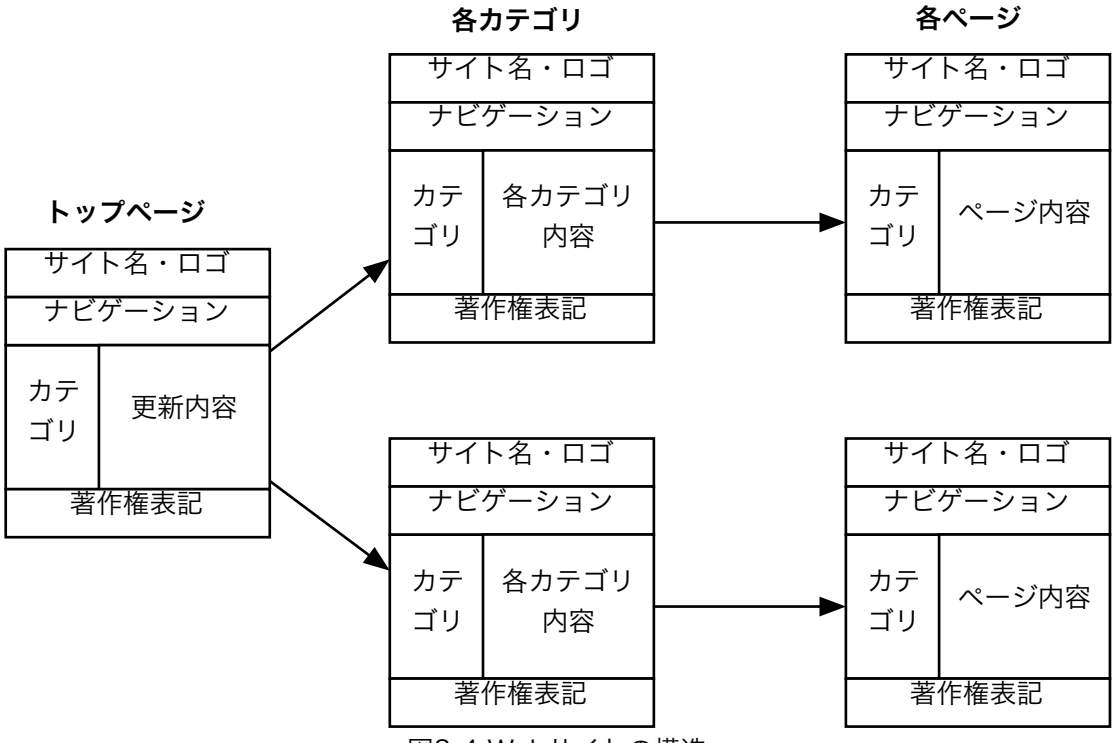

図3-4 Webサイトの構造

3.2.3 インターフェースとロジック

Webシステムでは、インターフェースとロジックを完全に分離することはできない。クライアント-サーバ型ネットワークの性質上、インターフェースにデータ送受信を行うためのロジックが入るからである。

Webシステムにおいてインターフェースとロジックを実装する方法は様々であるが、HTMLとCGIによるデータ送受信の例を示す。HTMLでは、formタグを使ってデータ送信のための設定を行う。設定する要素は「送信先のプログラム、送信方法、データ取得のキー」である。

HTML

```
<form action="/cgi-bin/getenv.cgi" method="post">
<input type="text" name="myname">
</form>
```

データはHTTPリクエストとして送信される。CGIでデータを取得するには、環境変数やアクセスメソッドを用いる。Perlの場合、環境変数にキーを指定することで送信されたデータを取得する。

CGI(Perl)

```
$myname = $ENV{'myname'}
```

このように、なんらかの形でロジックを意識することになり、インターフェースとロジックは切り離せないものになっている。

4. PACパターンによるWebアプリケーション開発

4.1 Presentation-Abstraction-Control

4.1.1 Presentation

Webページ全体を管理する。3.2.2で述べた通り、WebページはWebサイト固有の基本レイアウトを持つ。そこで、WebページをWebPageオブジェクトとWebComponentオブジェクトに分けて管理する(図4-1)。具体的なインターフェースと附随するロジックはすべてWebComponentオブジェクトが持ち、WebPageオブジェクトはWebComponentオブジェクトを組み合わせて画面を構成する。

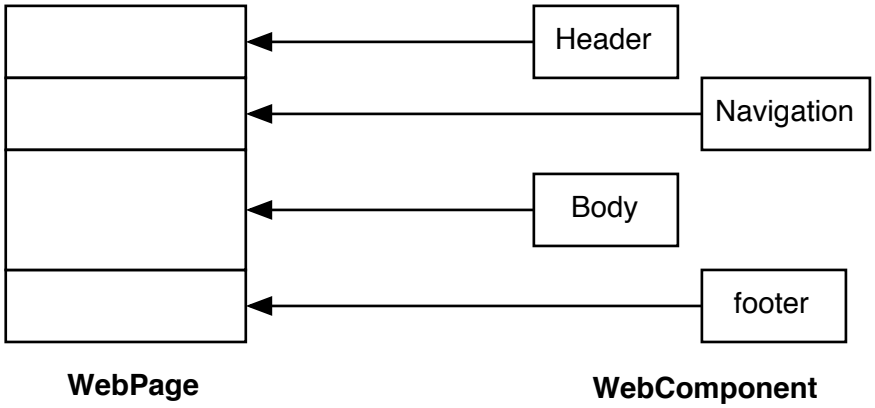

図4-1 WebPageオブジェクトとWebComponentオブジェクト

4.1.2 Abstraction

通常のPACパターンと同様、ロジックとデータを管理する。

4.1.3 Control

クライアントからのリクエストを受け付ける。その他は通常のPACパターンと同様、PresentationコンポーネントとAbstractionコンポーネントの接続と、他エージェントとの通信を行う。

4.2 エージェント階層

4.2.1 トップレベルエージェント

Presentation

トップページと、すべてのエージェントに共通するWebComponentオブジェクトを

提供する。

Abstraction
: ライブラリやフレームワーク等、エージェント全体で共有するロジックを提供する。

Control
: 下層エージェントを管理する。

4.2.2 中間レベルエージェント

Presentation
: 主にWebPageオブジェクトを持つ。下層エージェントの持つWebComponentオブジェクトを組み合わせてWebPageオブジェクトを構成する。

Abstraction
: 下層エージェントを管理するためのロジックを持つ。

Control
: 上下階層のエージェントとの通信を行う。

4.2.3 ボトムレベルエージェント

Presentation
: WebPageとWebComponentを持つ。中間レベルエージェントと組み合わせる場合、WebComponentオブジェクトを多く持つ。

Abstraction
: エージェント固有のロジックとデータを持つ。

Control
: 上層エージェントや他のエージェントと通信を行う。

4.3 処理の流れ

図4-2は、Webアプリケーションの基本的な処理の流れである。

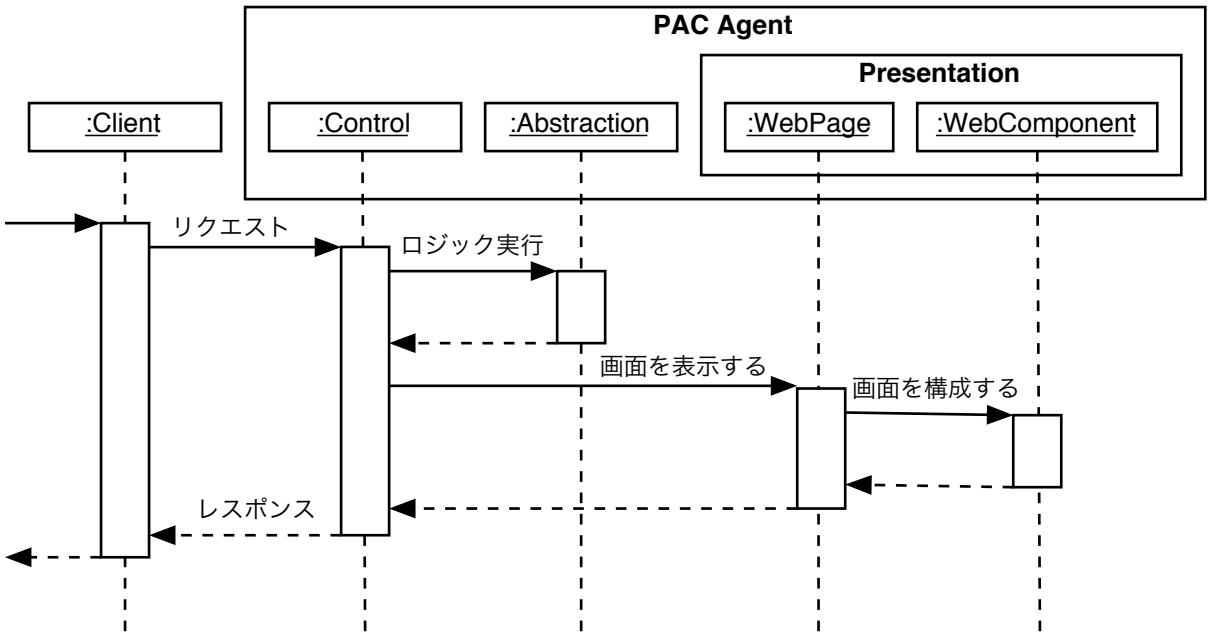

図4-2 Webアプリケーションの処理の流れ

4.4 特徴

4.4.1 メリット

効率的なインターフェースの開発

WebページをWebComponentオブジェクトを組み合わせて生成することで、静的HTMLのみの開発よりも効率的にインターフェースを開発することができる。

インターフェースとロジックの分離

PresentationコンポーネントとAbstractionコンポーネントが独立しているので、インターフェースとロジックをある程度独立して開発できる。

4.4.2 デメリット

主なデメリットは通常のPACパターンと同じである。

5. 実装

サンプルとして、JavaのCollectionフレームワークを紹介するWebアプリケーションを実装する。言語にはJavaを使い、サーブレットとJSPで実装する。

5.1 設計

5.1.1 仕様

JavaのCollectionフレームワークのうち、Set、List、Mapを紹介するページと、それを応用したサンプルページを用意する(図5-1)。

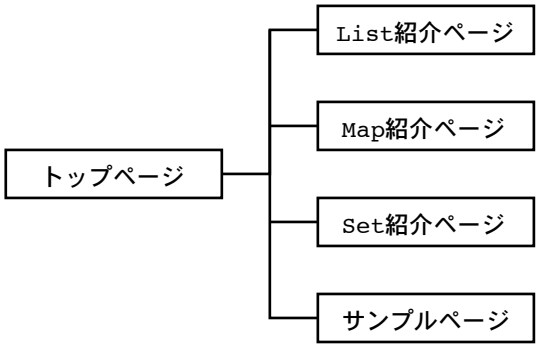

図5-1 Webアプリケーションの構造

トップページ(図5-2)に各Collection紹介ページとサンプルページへのリンクを用意する。Collectionのどれかをクリックすると、実装クラスの例を紹介するページへリンクする。

各Collection紹介ページでは、2つずつ実装クラスの例を紹介している。今回のサンプルで紹介するCollectionクラスは、すべて仕様のみを記述したインターフェースであり、実際に使用するのはCollectionインターフェースを実装したクラスになる。例えばSet紹介ページでは、Setインターフェースを実装したクラスHashSetとTreeSetを紹介している。

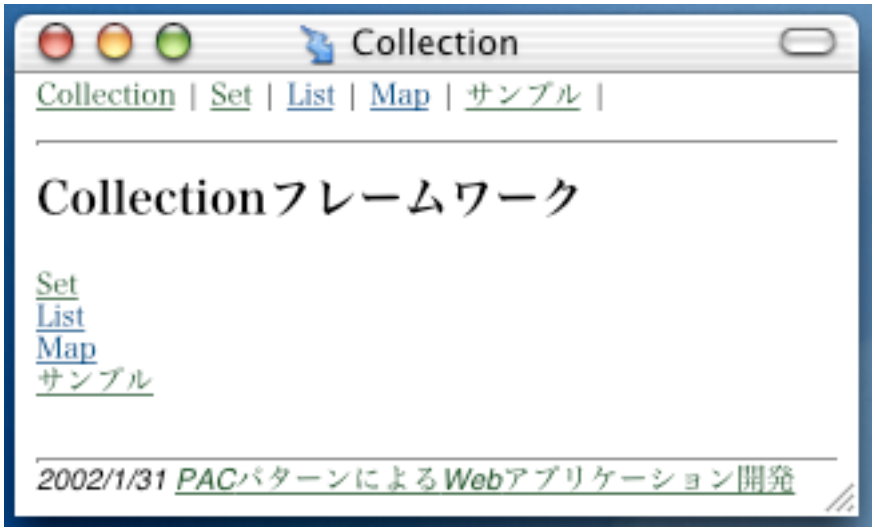

図5-2 トップページ

サンプルページ(図5-2)では、各Collectionクラスの実験を行うことができる。Collectionを選択し、フォームに適当なデータを入力する。送信ボタンを押すと、入力したデータを選択したCollectionに格納した結果を表示する。かっこ内はそれぞれのフォームに入力したデータである。

図5-2では、"Strategy, Composite, Observer"の順にデータを入力した結果である。入力したデータは、HashSetに以下のような組み合わせで「入力した順に」格納される。

キー	値
element1	Strategy
element2	Composite
element3	Observer

Iteratorを使ってHashSetの中身を「格納されている順に」取り出したのが、画面下に表示される結果である。HashSetはキーと値を結び付けたデータを管理するクラスであり、データを入力した順番通りに格納するとは限らない。従って、結果はデータがランダムな順で表示される。他のCollectionクラスのサンプルも、データを「入力した順に」格納し、「格納されている順に」取り出している。

図5-3 サンプルページ

5.1.2 運用環境

サーブレットとJSPで実装したので、サーバサイドJavaの環境であればOSを問わず動作する。テストにはMac OS X 10.1上でResin 2.0.4を利用した。

5.1.3 エージェント

エージェントの階層を3段階に分ける(図5-4)。すべてのエージェントを管理するcollectionAgentエージェントをトップレベルエージェント、listAgentエージェントとsetAgentエージェントを中間レベルエージェント、mapAgentエージェントとsampleAgentエージェントをボトムレベルエージェントとする。

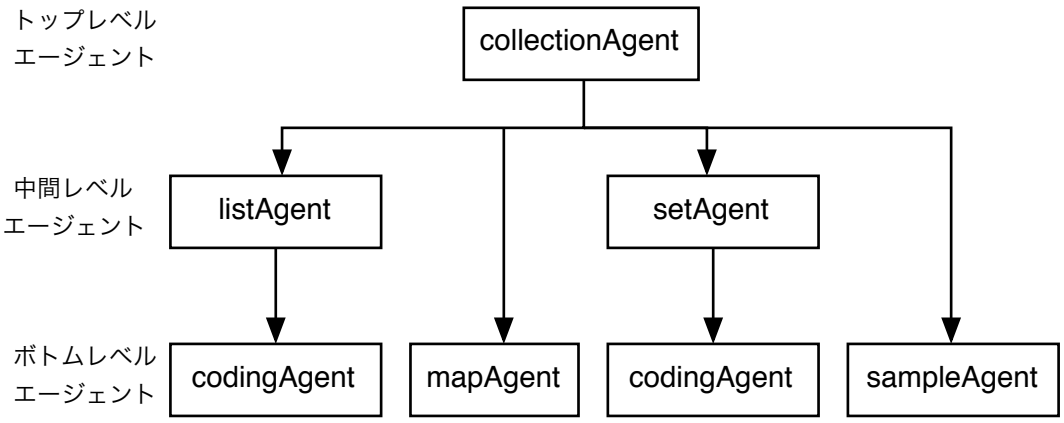

図5-4 エージェントの階層

5.1.4 インターフェース

Webページの構成を3〜4つに分け、WebComponentコンポーネントとして各エージェントのPresentationコンポーネントに配置する。

トップページ

上からヘッダー(headerComponent)、ナビゲーション(navigationComponent)、ページ本文(contentsComponent)、フッター(footerComponent)に分ける。

紹介ページ

ヘッダー、ナビゲーション、各Collecion紹介文(arrayListComponent等)、フッターに分ける。

サンプルページ

ヘッダー、ナビゲーション、データ入力フォーム(formComponent)、結果表示(resultComponent)、フッターに分ける。

5.2 実装

5.2.1 ディレクトリ構造

ディレクトリ構造は、エージェントの階層とほとんど同じになる(図5-5)。ただし、JSPと

サーブレットではディレクトリが異なる。ResinサーバではJSPファイルを/docディレクトリに、サーブレットを/WEB-INF/classディレクトリに置く。

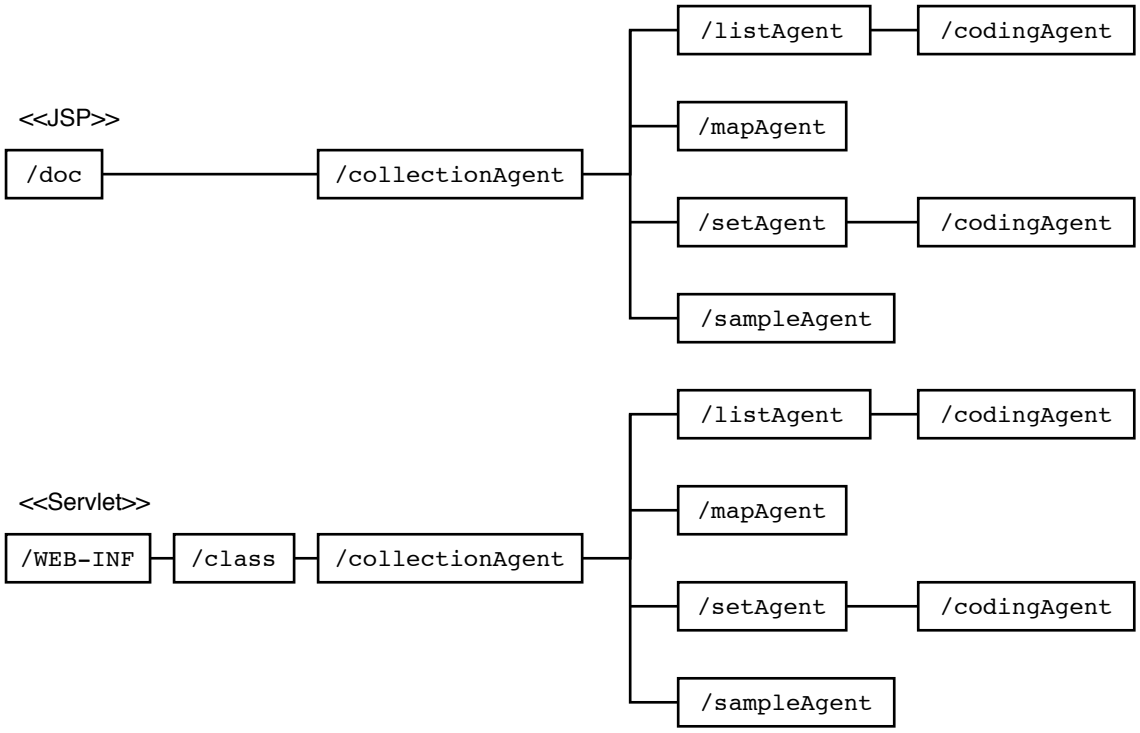

図5-5 ディレクトリ構造

5.2.2 Presentation-Abstraction-Control

Presentation

JSPで実装する。WebPageコンポーネントのファイル名は"Page"を、WebComponentコンポーネントのファイル名は"Component"を含む。

WebPageコンポーネントからWebComponentコンポーネントを呼び出すには、includeディレクティブを使って他エージェントのControlコンポーネントを呼び出し、WebComponentコンポーネント名をクエリに記述する(例1)。

例1) collectionAgentエージェントのheaderComponentコンポーネントを呼び出す

```
<jsp:include
   page="/servlet/collectionAgent.Control?action=headerComponent"
   flush="true"
/>
```

例2は、WebComponentコンポーネントであるsampleAgentエージェントのformComponent.jspファイルである。フォームデータの送信先はControlコンポーネントであり、アクションとしてAbstractionコンポーネントを呼び出すよう隠蔽フォームで指定している。

例2) sampleAgentエージェントのformComponent.jspファイル

```
<h2>サンプル</h2>

<form action="/servlet/collectionAgent.sampleAgent.Control"
      method="post">
<input type="hidden" name="action" value="abstraction">

<p>
Collection:<br>
<select name="collection">
<option value="HashSet">Set:HashSet</option>
<option value="TreeSet">Set:TreeSet</option>
<option value="ArrayList">List:ArrayList</option>
<option value="LinkedList">List:LinkedList</option>
<option value="HashMap">Map:HashMap</option>
<option value="TreeMap">Map:TreeMap</option>
</select>
</p>

<p>
Element1:<br><input type="text" name="element1"><br><br>
Element2:<br><input type="text" name="element2"><br><br>
Element3:<br><input type="text" name="element3"><br>
</p>

<input type="submit" value="Submit">
</form>
```

Abstraction

通常のJavaまたはサーブレットで実装する。Abstractionコンポーネントはロジックのみを持ち、フロー制御を行うことはない。

例3はsampleAgentエージェントのAbstractionクラス中のコンストラクタメソッドである。Collection名と要素を引数として与えると、指定したCollectionクラスのインスタンスを生成する。

例3) Abstractionクラスのコンストラクタメソッド

```
Abstraction( String collection, String element1,
             String element2,   String element3 )
{
    this.collection = collection;
    this.element1   = element1;
    this.element2   = element2;
    this.element3   = element3;

    if ( collection.equals( "HashSet" ) )
         doHashSet();
```

```
        if ( collection.equals( "TreeSet" ) )
             doTreeSet();
        if ( collection.equals( "ArrayList" ) )
             doArrayList();
        if ( collection.equals( "LinkedList" ) )
             doLinkedList();
        if ( collection.equals( "HashMap" ) )
             doHashMap();
        if ( collection.equals( "TreeMap" ) )
             doTreeMap();
    }
```

Control

サーブレットで実装する。PresentationコンポーネントとAbstractionコンポーネントを使用してフロー制御を行うほか、他エージェントと通信を行う。

例4はsampleAgentエージェントのControlクラスである。HTTPリクエストからクエリを取得し、PresentationコンポーネントやAbstractionコンポーネントを呼び出している。action属性に"abstraction"を指定した例2のフォームからデータが送られると、例3のAbstractionクラスを呼び出してロジックを実行し、Presentationコンポーネントのindex Page.jspファイルを呼び出して画面を生成する。

例4) sampleAgentエージェントのControlクラス

```
package collectionAgent.sampleAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
        "/collectionAgent/sampleAgent/indexPage.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );
        String collection = request.getParameter( "collection" );
        String element1   = request.getParameter( "element1" );
        String element2   = request.getParameter( "element2" );
```

```
        String element3   = request.getParameter( "element3" );

        if ( action == null )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "indexPage" ) )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "abstraction" ) ) {

            // ここからロジックを実行する
            Abstraction ab = new Abstraction( collection,
                                  element1, element2, element3 );
            request.setAttribute( "collection",
                                  ab.getCollection() );
            request.setAttribute( "element1",
                                  ab.getElement1() );
            request.setAttribute( "element2",
                                  ab.getElement2() );
            request.setAttribute( "element3",
                                  ab.getElement3() );
            request.setAttribute( "resultElement1",
                                  ab.getResultElement1() );
            request.setAttribute( "resultElement2",
                                  ab.getResultElement2() );
            request.setAttribute( "resultElement3",
                                  ab.getResultElement3() );

            // ロジック実行後、画面を生成する
            gotoPage( indexPage, request, response );
        } else
            gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

5.2.3 トップレベルエージェント

トップレベルエージェントはcollectionAgentエージェントのみである。collectionAgentエージェントはトップページであるindexPage.jspファイルと、4つのWebComponentコンポーネントを持つ。

WebComponentコンポーネントのうちheaderComponent.jsp、footerComponent.jsp、navigationComponent.jspの3つのファイルはすべてのエージェントで共有する。

collectionAgentエージェントのファイルは以下の通りである。WebPageコンポーネントのかっこ内は使用するWebComponentコンポーネントである。

collectionAgent

```
indexPage.jsp(headerComponent.jsp, footerComponent.jsp,
              navigationComponent.jsp, contentsComponent.jsp)
headerComponent.jsp
footerComponent.jsp
navigationComponent.jsp
contentsComponent.jsp
Control.java
```

5.2.4 中間レベルエージェント

中間レベルエージェントは、listAgentエージェントとsetAgentエージェントである。両方ともボトムレベルエージェントにそれぞれcodingAgentエージェントを持つ。listAgent、setAgentの両エージェントはWebPageコンポーネントのみを持ち、ボトムレベルエージェントの持つWebComponentコンポーネントを使用して画面を生成する。

listAgent

```
indexPage.jsp(headerComponent.jsp, footerComponent.jsp,
              navigationComponent.jsp, arrayListComponent.jsp,
              linkedListComponent.jsp)
Control.java
```

codingAgent

```
arrayListComponent.jsp
linkedListComponent.jsp
Control.java
```

setAgent

```
indexPage.jsp(headerComponent.jsp, footerComponent.jsp,
              navigationComponent.jsp, hashSetComponent.jsp,
              treeSetComponent.jsp)
Control.java
```

codingAgent

```
indexPage.jsp
hashSetComponent.jsp
treeSetComponent.jsp
Control.java
```

5.2.5 ボトムレベルエージェント

ボトムレベルエージェントは、mapAgentエージェント、sampleAgentエージェント、codingAgentエージェントである。ボトムレベルエージェントはWebPageコンポーネントとWebComponentコンポーネントを持ち、エージェント内で1つのデータモデルが完結する。

mapAgent

```
indexPage.jsp(headerComponent.jsp, footerComponent.jsp,
              navigationComponent.jsp, hashMapComponent.jsp,
              treeMapComponent.jsp)
hashMapComponent.jsp
treeMapComponent.jsp
Control.java
```

sampleAgent

```
indexPage.jsp(headerComponent.jsp, footerComponent.jsp,
              navigationComponent.jsp, formComponent.jsp,
              resultComponent.jsp)
formComponent.jsp
resultComponent.jsp
Abstraction.java
Control.java
```

6. まとめ

PACパターンは大規模アプリケーションに適用したときに最も効果を発揮する。しかしWebアプリケーションにおいては、小中規模であっても十分効率のいい開発ができるだろう。コンポーネントベースの開発であるために、スパイラル型の開発が多いWebアプリケーション開発において有利になるからである。

ただし、実際に応用するのが難しい点もある。分業とコストの問題である。インターフェースとロジックの開発を分業で行う場合、デザイナーの負担が大きくなる可能性がある。デザイナーとプログラマーは、コンポーネントやインターフェースに関するロジックについて綿密な打ち合わせを行わなくてはならない。デザイナーはコンポーネントを考慮してインターフェースデザインを行う必要があり、さらにHTMLでコンポーネントを実現しなくてはならないのである。また、通信にかかるコストやControlコンポーネントの実装が複雑になるので、厳しいパフォーマンスを要求するシステムに適用するのは難しい。

Webシステムのアーキテクチャを調べると様々な場面でMVCパターンを目にするが、大規模なWebシステムを考慮したアーキテクチャでは当然必ずしもMVCパターンとは限らない。J2EE(Java2 Enterprise Edition)を利用したWebアプリケーション開発では、アーキテクチャを多層構造で設計することが多い。J2EE開発の経験から作られたJ2EEパターンカタログでは、Webアプリケーションを5つの層(クライアント、プレゼンテーション、ビジネス、インテグレーション、リソース)に分け、それぞれの層を対象にパターンが位置付けられている。このように、WebアプリケーションのアーキテクチャはMVCからレイヤーの多重階層構造に移りつつある。PACパターンも例外ではなく、いずれエージェントの階層構造を採用するWebアプリケーションも登場するだろう。

今回はPACパターンにもJavaにもあまり習熟していなかったこともあり、あまりうまく設計・実装ができなかった。今後も引き続き研究を続け、Webアプリケーションのためのフレームワークを開発する予定である。

参考文献

[1] Erich,Gamma.,他:オブジェクト指向による再利用のためのデザインパターン改訂版, ソフトバンクパブリッシング,(2001)

[2] PLoPD Editors:プログラムデザインのためのパターン言語,ソフトバンクパブリッシング,(2001)

[3] FrankBuschmann.,他:ソフトウェアアーキテクチャ,近代科学社(2000)

[4] Martin,Fowloer.:リファクタリング,ピアソンエデュケーション,(2000)

[5] 株式会社永和システムマネジメント　オブジェクト倶楽部:オブジェクトハンドブック, ピアソンエデュケーション,(2001)

[6] Tomas,J.Mowbray.,William,A.Ruh.:Inside CORBA,アジソンウェスレイパブリッシャーズジャパン,(1998)

[7] 青木淳:SmallTalk Software Development, http://www.sra.co.jp/people/aoki/SuperAsciiJ/index.html

[8] 蓮見,樽沢:J2EEパターン;JavaWorld12月号,IDGジャパン,p74-104(2001)

[9] 上条晃宏:MVCモデル2とフレームワーク;JavaWorld7月号,IDGジャパン,p58-64(2001)

[10] Amy,Fowler.:A Swing Architecture Overview; http://java.sun.com/products/jfc/tsc/articles/architecture/index.html, Sun Microsystems,Inc.(1995)

[11] Markku,Vourenmaa.:Automatic Presentation of Model Data in MVC++ Applications,University of Tampere Department of Computer and Information Sciences,(2000)

[12] Jim,Conallen.:UMLによるWebアプリケーション開発,ピアソンエデュケーション,(2000)

[13] Paul,Harmon.,他:Webアプリケーションのためのシステム開発とアーキテクチャ,日経BP社,(2001)

[14] バンザイヒロアキ:ロジックとデザインについて考える;技術評論社,ソフトウェアデザイン 2001年10月号(2001)

[15] 須加力:Webアプリケーションサーバ完全構築ガイド,日経BP,(2000)

[16] 吉原,他:開発生産性で選ぶWebアプリケーションサーバ:JAVA PRESS Vol.14,(2001)

[17] 渡邊,他:アプリケーションサーバによるWebシステム構築技法;WEB+DB PRESS Vol.1,(2001)

[18] 磯蘭水:Webアプリケーションサーバを知る;技術評論社,ソフトウェアデザイン 2002年2月号(2002)

[19] Sun Microsystems,Inc.:Java Servlet API, http://java.sun.com/products/servlet/2.3/javadoc/index.html, Sun Microsystems,Inc.

[20] Ken,Arnold.,他:プログラミング言語Java第3版,ピアソンエデュケーション,(2001)

[21] Marty,Hall.:コア・サーブレット&JSP,ソフトバンクパブリッシング,(2001)

[22] Caucho Technology,Inc.:Resin,http://www.caucho.com/,Caucho Technology,Inc.

サンプルコード

1. collectionAgent

indexPage.jsp

```
<jsp:include
    page="/servlet/collectionAgent.Control?action=headerComponent"
    flush="true" />
<jsp:include
  page="/servlet/collectionAgent.Control?action=navigationComponent"
  flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=contentsComponent"
    flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=footerComponent"
    flush="true" />
```

headerComponent.jsp

```
<html>
<head>
<title>Collection</title>
<body>
```

footerComponent.jsp

```
<hr>
<address>
    2002/1/31 PACパターンによるWebアプリケーション開発<br>
</address>
</body>
</html>
```

navigationComponent.jsp

```
<p>
  <a href="/servlet/collectionAgent.Control">Collection</a> |
  <a href="/servlet/collectionAgent.setAgent.Control">Set</a> |
  <a href="/servlet/collectionAgent.listAgent.Control">List</a> |
  <a href="/servlet/collectionAgent.mapAgent.Control">Map</a> |
  <a href="/servlet/collectionAgent.sampleAgent.Control">サンプル</a>
  <hr>
</p>
```

contentsComponent.jsp

```
<h2>Collectionフレームワーク</h2>
```

```
<p>
    <a href="/servlet/collectionAgent.setAgent.Control">
        Set</a><br>
    <a href="/servlet/collectionAgent.listAgent.Control">
        List</a><br>
    <a href="/servlet/collectionAgent.mapAgent.Control">
        Map</a><br>
    <a href="/servlet/collectionAgent.sampleAgent.Control">
        サンプル</a><br>
</p>
```

Control.java

```
package collectionAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
           "/collectionAgent/indexPage.jsp";
    private final String headerComponent =
           "/collectionAgent/headerComponent.jsp";
    private final String footerComponent =
           "/collectionAgent/footerComponent.jsp";
    private final String navigationComponent =
           "/collectionAgent/navigationComponent.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        if ( action == null )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "headerComponent" ) )
            gotoPage( headerComponent, request, response );
        else if ( action.equals( "footerComponent" ) )
            gotoPage( footerComponent, request, response );
        else if ( action.equals( "navigationComponent" ) )
```

```
            gotoPage( navigationComponent, request, response );
        else
            gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

2. listAgent

2.1 listAgent

indexPage.jsp

```
<jsp:include
    page="/servlet/collectionAgent.Control?action=headerComponent"
    flush="true" />
<jsp:include
  page="/servlet/collectionAgent.Control?action=navigationComponent"
  flush="true" />
<jsp:include
    page="/servlet/collectionAgent.listAgent.codingAgent.Control?
          action=arrayListComponent"
    flush="true" /><br>
<jsp:include
    page="/servlet/collectionAgent.listAgent.codingAgent.Control?
          action=linkedListComponent"
    flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=footerComponent"
    flush="true" />
```

Control.java

```
package collectionAgent.listAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
        "/collectionAgent/listAgent/indexPage.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

2.2 codingAgent

arrayListComponent.jsp

```
<h2>ArrayList</h2>
```

```
<p>
  Listは、配列に要素を保持する基本的なリストである。
ArrayListは、配列に要素を保持する基本的なリストの実装である。
</p>
```

2.2.2 linkedListComponent.jsp

```
<h2>LinkedList</h2>
<p>
  Listは、配列に要素を保持する基本的なリストである。
LinkedListは双方向リンクリストであり、パフォーマンス特性がArrayListの逆である。
</p>
```

Control.java

```
package collectionAgent.listAgent.codingAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    private final String arrayListComponent =

"/collectionAgent/listAgent/codingAgent/arrayListComponent.jsp";
    private final String linkedListComponent =

"/collectionAgent/listAgent/codingAgent/linkedListComponent.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        if ( action == null )
            gotoPage( arrayListComponent, request, response );
        else if ( action.equals( "arrayListComponent" ) )
            gotoPage( arrayListComponent, request, response );
        else if ( action.equals( "linkedListComponent" ) )
```

```
            gotoPage( linkedListComponent, request, response );
        else
            gotoPage( arrayListComponent, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

3. setAgent

3.1 setAgent

indexPage.jsp

```
<jsp:include
    page="/servlet/collectionAgent.Control?action=headerComponent"
    flush="true" />
<jsp:include
  page="/servlet/collectionAgent.Control?action=navigationComponent"
  flush="true" />
<jsp:include
    page="/servlet/collectionAgent.setAgent.codingAgent.Control?
          action=hashSetComponent"
    flush="true" /><br>
<jsp:include
    page="/servlet/collectionAgent.setAgent.codingAgent.Control?
          action=treeSetComponent"
    flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=footerComponent"
    flush="true" />
```

Control.java

```
package collectionAgent.setAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
        "/collectionAgent/setAgent/indexPage.jsp";
    private final String headerComponent =
        "/collectionAgent/setAgent/headerComponent.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        if ( target == null )
            gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

3.2 codingAgent

hashSetComponent.jsp

```
<h2>HashSet</h2>
<p>
  Setは、同じ要素を複数個入れることができないデータ型である。
HashSetは、ハッシュテーブルを使用して実装されたSetである。
</p>
```

treeSetComponent.java

```
<h2>TreeSet</h2>
<p>
  Setは、同じ要素を複数個入れることができないデータ型である。
TreeSetは、木構造で要素を保持するSetである。
</p>
```

Control.java

```
package collectionAgent.setAgent.codingAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    private final String hashSetComponent =
       "/collectionAgent/setAgent/codingAgent/hashSetComponent.jsp";
    private final String treeSetComponent =
       "/collectionAgent/setAgent/codingAgent/treeSetComponent.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        if ( action == null )
            gotoPage( hashSetComponent, request, response );
        else if ( action.equals( "hashSetComponent" ) )
            gotoPage( hashSetComponent, request, response );
        else if ( action.equals( "treeSetComponent" ) )
```

```
            gotoPage( treeSetComponent, request, response );
        else
            gotoPage( hashSetComponent, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

4. mapAgent

indexPage.jsp

```
<jsp:include
    page="/servlet/collectionAgent.Control?action=headerComponent"
    flush="true" />
<jsp:include
  page="/servlet/collectionAgent.Control?action=navigationComponent"
  flush="true" />
<jsp:include
    page="/servlet/collectionAgent.mapAgent.Control?
          action=hashMapComponent"
    flush="true" /><br>
<jsp:include
    page="/servlet/collectionAgent.mapAgent.Control?
          action=treeMapComponent"
    flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=footerComponent"
    flush="true" />
```

hashMapComponent.jsp

```
<h2>HashMap</h2>
```

```
<p>
  Mapは、キーと値の組を保持するデータ型である。
HashMapは、ハッシュテーブルを使用して実装されたMapである。
</p>
```

treeMapComponent.jsp

```
<h2>TreeMap</h2>
<p>
  Mapは、キーと値の組を保持するデータ型である。
TreeMapは、木構造でキーと値の組を保持するMapである。
</p>
```

Control.java

```
package collectionAgent.mapAgent;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
"/collectionAgent/mapAgent/indexPage.jsp";
    private final String hashMapCompponent =
        "/collectionAgent/mapAgent/hashMapComponent.jsp";
    private final String treeMapCompponent =
        "/collectionAgent/mapAgent/treeMapComponent.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );

        if ( action == null )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "indexPage" ) )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "hashMapComponent" ) )
```

```
            gotoPage( hashMapCompponent, request, response );
        else if ( action.equals( "treeMapComponent" ) )
            gotoPage( treeMapCompponent, request, response );
        else
            gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
                           HttpServletRequest request,
                           HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                 throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

5. sampleAgent

indexPage.jsp

```
<jsp:include
    page="/servlet/collectionAgent.Control?action=headerComponent"
    flush="true" />
<jsp:include
  page="/servlet/collectionAgent.Control?action=navigationComponent"
  flush="true" />
<jsp:include page="formComponent.jsp" flush="true" /><br>
<jsp:include page="resultComponent.jsp" flush="true" />
<jsp:include
    page="/servlet/collectionAgent.Control?action=footerComponent"
    flush="true" />
```

formComponent.jsp

```
<h2>サンプル</h2>

<form action="/servlet/collectionAgent.sampleAgent.Control"
      method="post">
<input type="hidden" name="action" value="abstraction">
```

```
<p>
    Collection:<br>
    <select name="collection">
        <option value="HashSet">Set:HashSet</option>
        <option value="TreeSet">Set:TreeSet</option>
        <option value="ArrayList">List:ArrayList</option>
        <option value="LinkedList">List:LinkedList</option>
        <option value="HashMap">Map:HashMap</option>
        <option value="TreeMap">Map:TreeMap</option>
    </select>
</p>

<p>
    Element1:<br><input type="text" name="element1"><br><br>
    Element2:<br><input type="text" name="element2"><br><br>
    Element3:<br><input type="text" name="element3"><br>
</p>

<input type="submit" value="Submit">
</form>
```

resultComponent.jsp

```
<h2>結果</h2>
```

※かっこ内は入力したデータ

```
<p>
    <strong>Collection:</strong>
    <%= request.getAttribute("collection") %>
</p>
<p>
    <strong>Element1:</strong>
    <%= request.getAttribute("resultElement1") %>
    (<%= request.getAttribute("element1") %>)
</p>
<p>
    <strong>Element2:</strong>
    <%= request.getAttribute("resultElement2") %>
    (<%= request.getAttribute("element2") %>)
</p>
<p>
    <strong>Element3:</strong>
    <%= request.getAttribute("resultElement3") %>
    (<%= request.getAttribute("element3") %>)
</p>
```

Control.java

```
package collectionAgent.sampleAgent;
```

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Control extends HttpServlet
{
    // WebPage
    private final String indexPage =
        "/collectionAgent/sampleAgent/indexPage.jsp";

    private String target;
    private String action;

    public void doGet( HttpServletRequest  request,
                       HttpServletResponse response )
                throws ServletException, IOException
    {
        target = request.getParameter( "target" );
        action = request.getParameter( "action" );
        String collection = request.getParameter( "collection" );
        String element1 = request.getParameter( "element1" );
        String element2 = request.getParameter( "element2" );
        String element3 = request.getParameter( "element3" );

        if ( action == null )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "indexPage" ) )
            gotoPage( indexPage, request, response );
        else if ( action.equals( "abstraction" ) ) {
            Abstraction ab = new Abstraction( collection, element1,
element2, element3 );
            request.setAttribute( "collection", ab.getCollection()
);
            request.setAttribute( "element1", ab.getElement1() );
            request.setAttribute( "element2", ab.getElement2() );
            request.setAttribute( "element3", ab.getElement3() );
            request.setAttribute( "resultElement1",
ab.getResultElement1() );
            request.setAttribute( "resultElement2",
ab.getResultElement2() );
            request.setAttribute( "resultElement3",
ab.getResultElement3() );
            gotoPage( indexPage, request, response );
        } else
            gotoPage( indexPage, request, response );
    }

    private void gotoPage( String path,
```

```
                            HttpServletRequest request,
                            HttpServletResponse response )
                    throws ServletException, IOException
    {
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher( path );
        dispatcher.forward( request, response );
    }

    public void doPost( HttpServletRequest request,
                        HttpServletResponse response )
                throws ServletException, IOException
    {
        doGet( request, response );
    }
}
```

Abstraction.java

```
package collectionAgent.sampleAgent;

import java.util.*;

public class Abstraction
{
    private String collection;
    private String element1;
    private String element2;
    private String element3;
    private String resultElement1;
    private String resultElement2;
    private String resultElement3;

    Abstraction( String collection, String element1,
                 String element2,   String element3 )
    {
        this.collection = collection;
        this.element1   = element1;
        this.element2   = element2;
        this.element3   = element3;

        if ( collection.equals( "HashSet" ) )
            doHashSet();
        if ( collection.equals( "TreeSet" ) )
            doTreeSet();
        if ( collection.equals( "ArrayList" ) )
            doArrayList();
        if ( collection.equals( "LinkedList" ) )
            doLinkedList();
```

```
    if ( collection.equals( "HashMap" ) )
        doHashMap();
    if ( collection.equals( "TreeMap" ) )
        doTreeMap();
}

private void doHashSet()
{
    HashSet hs = new HashSet();

    hs.add( element1 );
    hs.add( element2 );
    hs.add( element3 );

    Iterator i = hs.iterator();
    resultElement1 = (String)i.next();
    resultElement2 = (String)i.next();
    resultElement3 = (String)i.next();
}

private void doTreeSet()
{
    TreeSet ts = new TreeSet();

    ts.add( element1 );
    ts.add( element2 );
    ts.add( element3 );

    Iterator i = ts.iterator();
    resultElement1 = (String)i.next();
    resultElement2 = (String)i.next();
    resultElement3 = (String)i.next();
}

private void doArrayList()
{
    ArrayList al = new ArrayList();
    String[] s = new String[] { element1, element2, element3 };

    List li = Arrays.asList( s );
    al.addAll( li );

    ListIterator i = al.listIterator();
    resultElement1 = (String)i.next();
    resultElement2 = (String)i.next();
    resultElement3 = (String)i.next();
}

private void doLinkedList()
```

```
{
    LinkedList ll = new LinkedList();
    String[] s = new String[] { element1, element2, element3 };

    List li = Arrays.asList( s );
    ll.addAll( li );

    ListIterator i = ll.listIterator();
    resultElement1 = (String)i.next();
    resultElement2 = (String)i.next();
    resultElement3 = (String)i.next();
}

private void doHashMap()
{
    HashMap hm = new HashMap();

    hm.put( "element1", element1 );
    hm.put( "element2", element2 );
    hm.put( "element3", element3 );
    resultElement1 = (String)hm.get("element1");
    resultElement2 = (String)hm.get("element2");
    resultElement3 = (String)hm.get("element3");
}

private void doTreeMap()
{
    TreeMap tm = new TreeMap();

    tm.put( "element1", element1 );
    tm.put( "element2", element2 );
    tm.put( "element3", element3 );
    resultElement1 = (String)tm.get("element1");
    resultElement2 = (String)tm.get("element2");
    resultElement3 = (String)tm.get("element3");
}

public String getCollection()
{
    return collection;
}

public String getElement1()
{
    return element1;
}

public String getElement2()
{
```

```
        return element2;
    }

    public String getElement3()
    {
        return element3;
    }

    public String getResultElement1()
    {
        return resultElement1;
    }

    public String getResultElement2()
    {
        return resultElement2;
    }

    public String getResultElement3()
    {
        return resultElement3;
    }
}
```