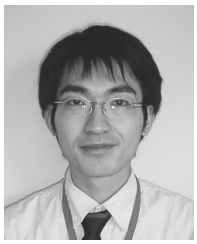
J2EE による Web n 階層アプリケーションの実装

SI 推進本部
IT ソリューショングループ
今井雅之

ネットシステム事業部
清水弘典

ネットシステム事業部
田中伸叔



ネットシステム事業部
田中宗弘

1．はじめに

当社 EJB（Enterprise JavaBeans）推進プロジェクトは、1999年10月に発足した当社内の研究プロジェクトである。同プロジェクトでは、Web アプリケーション構築技術、特に EJB/アプリケーションサーバー製品を中心とする、J2EE（The Java 2 Platform, Enterprise Edition）関連技術の調査、およびノウハウの蓄積を目的として活動を行い、その実装例としてプロトタイプを作成した。本稿ではプロトタイプをアーキテクチャの観点から解説し、Web n 階層アプリケーションにおける J2EE 技術の適用の一例を示したい。

2．J2EE とは

2.1 Java 適用分野の変化－Applet からサーバーサイドへ

登場した当初には Applet をはじめとして、クライアントサイドでの利用がメインだった Java であるが、最近はアプリケーションサーバー製品を用いた、サーバーサイドでの利用が増え注目されている。そこには次のような要因が考えられる。

(1) Web 技術との親和性

HTML に組み込んで利用する Applet にみられるように、Java は初期の頃から Web/ネットワーク系のサービスを実装しており、高い親和性をもっている。

(2) 移植性の高さ

“Write Once, Run Anywhere”という言葉に表されるように、サポートするプラットフォームであれば同一のコードが実行可能であり、さまざまな稼働環境に対応できることが Java の特長である。したがって、実装されたコンポーネントを再利用できる可能性も高く、Java ベースのビジネス・コンポーネント製品も登場している。

(3) サーバーサイド技術の充実

CGI（Common Gateway Interface）を代替する Servlet や分散オブジェクト、トランザクション管理などのサーバーサイド技術が登場し、エンタープライズ・アプリケーション構築の広範な領域を Java 技術でカバーすることが可能となった。JavaScript などの HTML スクリプト言語や CGI で一般的に用いられる Perl などに比べて、本格的なオブジェクト指向言語としての Java のメリットを享受できるため、Web アプリケーション構築における生産性、再利用性の向上が期待できる。

⑷ クライアントサイド Java における問題の回避

Applet や Swing[*1]などの GUI（Graphical User Interface）系 Java 技術では、稼働環境による互換性やパフォーマンスの問題があったが、非 GUI のサーバーサイド Java ではその制約を受けにくい。したがって、GUI に HTML を採用し、サーバーサイドに Java を利用するアーキテクチャをもつ Web アプリケーションでは、Java のメリットを活用できることになる。

2.2 J2EE＝エンタープライズ Java 技術の総称

J2EE は、エンタープライズ Java 技術の総称であり、n 階層アプリケーション構築をサポートするものと位置付けられている（表１参照）。

表１　エンタープライズ Java 技術（J2EE）の一覧

JDBC	データベース接続
JTA(Java Transaction API) JTS(Java Transaction Service)	トランザクション管理
JNDI(Java Naming and Directory Interface)	ネーミング/ディレクトリサービス
RMI(Remote Method Invocation) RMI over IIOP	分散オブジェクト呼び出し
Java IDL	CORBA マッピング
JMS(Java Message Service) Java Mail JAF(JavaBeans Activation Framework)	メッセージングサービス
Servlet	Web サーバー拡張
JSP(Java Server Pages)	動的 HTML
EJB(Enterprise JavaBeans)	サーバーサイドコンポーネント

2.3 J2EE 対応のアプリケーションサーバー

⑴ J2EE の実行環境

アプリケーションサーバーはサーバーサイド・コンポーネントの実行環境を提供するものであり、J2EE 対応のものでは一般的に Servlet、JSP(Java Server Pages)、EJB を中心とした J2EE 仕様、および Web サーバー機能などが実装されている。

⑵ コンテナベース

J2EE の特徴の一つとして、コンテナベースの実行環境をあげることができる。コンテナとはコンポーネント特有のサービスを提供する実行環境であり、Web コンポーネントや EJB コンポーネントはアプリケーションサーバーのコンテナ上で動作する。例えば EJB コンテナはネーミング、トランザクション、永続化といったサービスを提供しており、それらを利用するためのコード記述が基本的に不要となる。また、コンテナの動作は実行時パラメータで指定できるため、コード記述なしでのカスタマイズが可能である。

⑶ 主な製品

J2EE 技術を利用した主な製品を次にあげる。

- BEA WebLogic　・IBM Websphere
- Oracle Application Server
- SilverStream Application Server
- Sun-Netscape iPlanet Web Server
- Inprise Application Server etc.

３．プロトタイプについて

3.1 システム概要

今回のプロトタイプでは、社内機器(ソフトウェア、ハードウェア）の管理システムを想定し、下記のシステム要求を設定した（図１参照：画面イメージ)。

図１　システム画面のイメージ

- 機器管理情報（機種、リース期間、管理者など）の登録、照会、修正。
- 発注状況の管理ソフトウェアの使用者またはインストール先機器へのライセンス割り当て状況の管理。
- 全てのオペレーションはWebブラウザを利用して行う。

3.2 開発手順

開発手順の流れを以下に示す。

⑴ 要求定義

成果物：ユースケース図、シナリオ

⑵ ユースケース分析

ユースケース記述からの分析クラスの抽出、およびシナリオごとのシーケンス図作成を行った。

成果物：分析クラス、シーケンス図

＊１）Swing : Java の GUI 部品パッケージで、ウィンドウズ・システムに依存せずに描画を行う。そのため環境による動作の違いを意識する必要がなく複雑な部品を提供できるが、実行するクライアントにある程度のマシンパワーが要求される。

⑶ **アーキテクチャ分析／設計**

ユースケースの内、代表的なものについてパッケージ分割、ハードウェア配置の検討を行った。

成果物：クラス図(主にパッケージ間の依存関係を記述)、シーケンス図

⑷ **ユースケース／クラス設計**

アーキテクチャ設計の結果を踏まえ、分析 Model を詳細化した。

成果物：設計クラス、シーケンス図

⑸ **実装**

アーキテクチャ階層ごとに担当を分担し、プログラムを作成した。

⑹ **テスト**

3.3 開発・実行環境、および使用したツール

開発・実行環境、および使用したツールを以下に示す。

・Windows NT Server4.0（OS）
・JDK1.2.2-001（Java 開発キット）
・Visual Cafe standard edition（Java 統合開発環境）
・WebLogic4.5.1（アプリケーションサーバー）
・SQL Server7.0（DBMS）
・Rational Rose98（オブジェクト指向モデリングツール）

4．ソフトウェア・アーキテクチャ

4.1 ソフトウェア・アーキテクチャとは

ソフトウェア・アーキテクチャとは、ある観点からみたシステムの構造である。適切な構造に基づいて作られたシステムは、構成する各部分の役割が明確であるため理解が容易であり、また独立性が高いことから変更や再利用への柔軟な対応が可能となる。特に最近では２階層クライアント/サーバー（C/S）から３階層 C/S、Web n 階層への変化、インターネット、分散コンポーネント、レガシーシステムとの統合といった多様な要素が存在する中で、適切なアーキテクチャを決定し、システム構築の指針とすることの重要性が高まっている。

4.2 プロトタイプのアーキテクチャ

このプロトタイプでは、画面によるデータ入力、検索、表示という典型的な対話型システムであるため、MVC（Model-View-Controller）アーキテクチャ・パターンをベースとして採用した。パターンとは、これまでの経験をもとに再利用可能な形で蓄積された問題解決策であり、新たに解決策を生み出すよりも実績のあるものを適用することで手間を減らし、信頼性をあげることが目的である。アーキテクチャ・パターン以外にも最近注目を集めているデザイン・パターンやアナリシス・パターンなどがあり、パターンはソフトウェア開発のあらゆる局面で使用することができる。

また、フレームワークなどに比べて局所的、抽象的であるため適用範囲が広く、新たな蓄積も比較的手軽なことからソフトウェア再利用の手段の一つとして積極的に活用すべきであろう。

次に、今回採用した MVC パターンの概要と当プロトタイプへの具体的な適用方法について説明する。

4.3 Model-View-Controller アーキテクチャ・パターン

MVC アーキテクチャ・パターンは、対話型システムにおいて、エンティティ・オブジェクト（Model）、ユーザーインタフェースの表現（View）、ユーザー入力に対する制御（Controller）を分離して設計し、再利用性と柔軟性を高めることを目的とする（図２参照）。

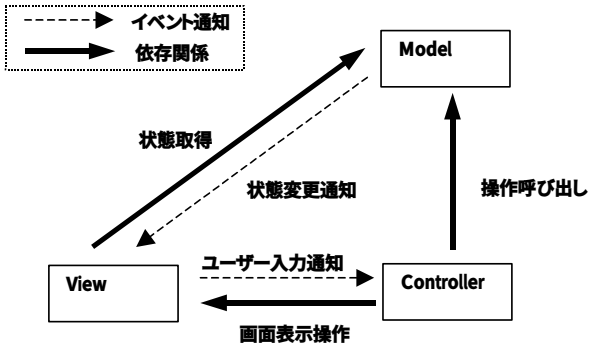


図２　MVC アーキテクチャ・パターン

これにより、例えば他の部分に影響を与えずにユーザーインタフェースを変更することが可能となる。各階層はそれぞれ以下のような役割をもっている。

⑴ **Model**

エンティティ・オブジェクトであり、アプリケーションデータとビジネスロジックを表現する。状態に変更があった場合は関連する View にそのことを通知する。

⑵ **View**

Model の状態を画面などに表示する。Model から状態が変更したという通知を受けると表示を更新し、常に Model の最新状態を表示する。ユーザーの操作があった場合は Controller にイベントを伝達する。

⑶ **Controller**

View から伝達されたユーザー操作イベントに従って Model や View を操作し、所定の作業を行わせる。つまり、Controller がそのオペレーションの振る舞いを制御する中心的な役割を果たすことになる。

4.4 プロトタイプへの MVC パターンの適用

このプロトタイプでは、J2EE を利用するという前提

と実装の単純化のためにMVCパターンを一部変更して適用した（図３参照）。

なお、基本アーキテクチャの動きについては新規データ登録オペレーションを例にとったシーケンス図を示す（図４参照）。

第一の変更点として、Controllerを、ユーザー入力イベントの受付と画面の動きを制御するView Controller（Servlet）、およびオペレーションの動きを制御するOperation Controller（Session Bean）に分割した。これはView階層とController階層の独立性を高めることが目的で、例えば今回JSP＋Servletで構築したユーザーインタフェース部分を、高機能なGUIをもつJavaアプリケーションに置き換えることも可能となる。

また、MVCパターンではViewがModelの状態を自ら取得して表示することになっているが、Controller経由でModelのスナップショットを受け取って表示するように変更した。これはViewをModelに依存させずModelの実装方式を変更可能にしておくこと、Entity Beanで実装されるModelをViewがアクセスすることでトランザクションの範囲がViewまで広がるのを避けることが目的である。

以上の変更の結果、本来のMVCパターンとはやや離れてしまったが、（Operation）Controllerが主導し、ViewとModelを分離して仲介を行うという点では、結果的にPresenter-Abstract-Controlパターン（本稿末の参考文献１参照）に近づいたといえる。

次章以降では、アーキテクチャの階層ごとに関連技術とその適用方法を説明する。

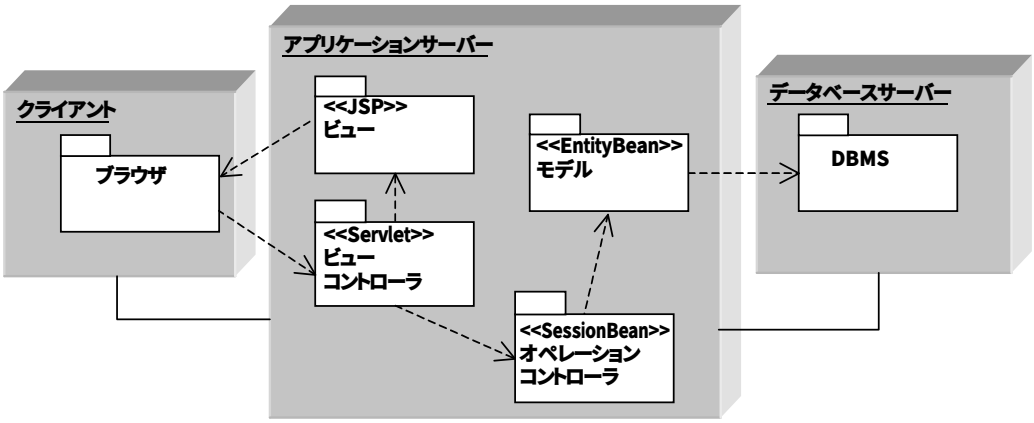


図３　プロトタイプへのMVCパターンの適用

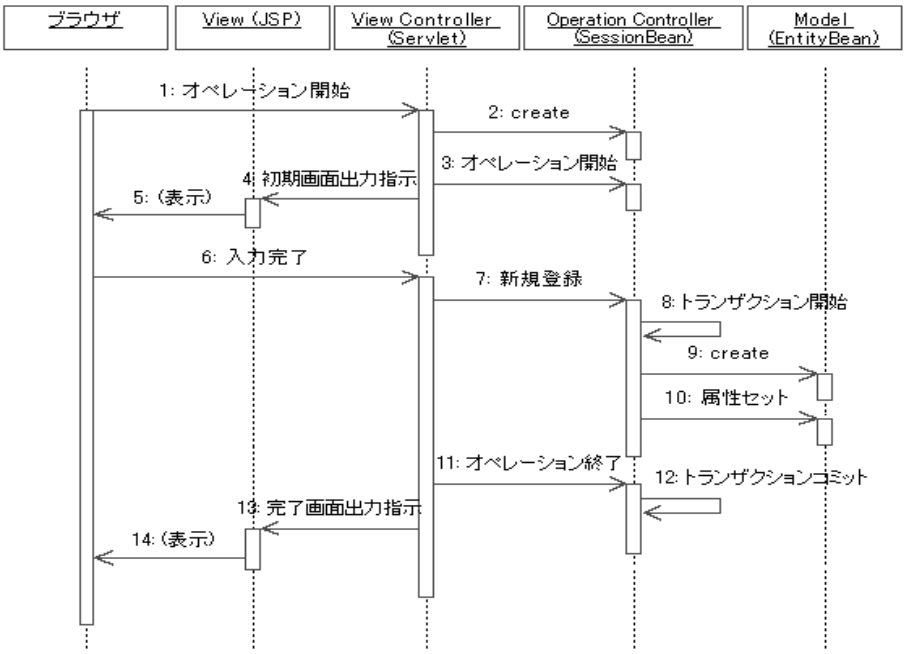


図４　新規データ登録オペレーションのシーケンス

５．Model 階層

5.1 関連技術

⑴ JDBC

JDBC は Java アプリケーションから JDBC API を利用してデータベース・アクセスを行う仕組みである。使用する DBMS に応じた JDBC ドライバが必要となるが、サードパーティー製を含め多くの DBMS 用のドライバがリリースされている。

⑵ RMI（Remote Method Invocation）

分散オブジェクト呼び出しの仕組みである。当初は、Java オブジェクト相互間においてのみ使用可能であったが、下位プロトコルに IIOP を使う RMI over IIOP の登場により、CORBA オブジェクトとの通信も可能となった。

⑶ JTA（Java Transaction API）

トランザクション管理のための API であり、アプリケーションやアプリケーションサーバーが利用する API や XA インタフェースの Java マッピングを提供する。

⑷ EJB（Enterprise JavaBeans）

主にクライアントサイドの GUI コンポーネントのための規格である JavaBeans に対し、サーバーサイドのビジネス・コンポーネントのための規格が EJB（Enterprise JavaBeans）である。EJB 仕様では用途に応じたいくつかの種類のコンポーネントが定義されている（表２参照）。

表２　EJB コンポーネント

Entity Bean		永続的オブジェクトをあらわすもので、プライマリーキーによって識別され、全クライアントからインスタンスが共有される。永続化の方式により CMP と BMP に分類される。
	Container Managed Persistance（CMP）	永続化の管理を EJB コンテナが行なうもので、EJB コンポーネントにはデータベースアクセスロジックの記述が不要となる。
	Bean Managed Persistance（BMP）	EJB コンポーネント自身で永続化の管理を行なうもので、データベースアクセスをカスタマイズしたい場合に用いる。
Session Bean		クライアントごとにインスタンスが作成され、通常クライアントと同じライフサイクルで利用される。状態保持方法により、Stateful と Stateless に分類される。
	Stateful	クライアントからの複数回の呼び出しの間でオブジェクトの状態を保持する。
	Stateless	クライアントからの呼び出しの間でオブジェクトの状態を保持しない。

EJB コンポーネントは、アプリケーションサーバー製品によって提供される EJB コンテナと呼ばれる実行環境で動作する（図５参照）。

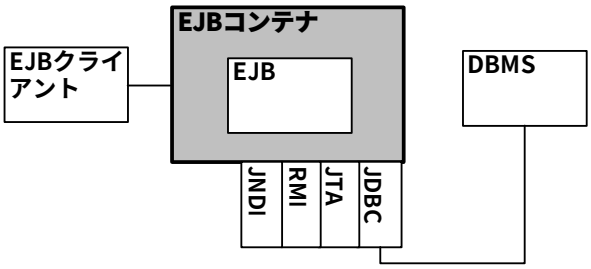


図５　EJB コンテナ

EJB コンテナが JDBC、JTA、RMI などのシステムレベルの処理を行ない、EJB コンポーネントはコンテナが提供する高レベルなサービスを利用することでビジネスロジックに専念することができる。

また、コンテナの実行時パラメータの設定により、EJB コンポーネントを作成し直すことなくデータベースやトランザクション設定の変更が可能である。これにより、異なる環境で再利用する場合にもコード変更が不要となるため、汎用的なコンポーネントを作成するための大きなメリットとなる。

5.2 プロトタイプへの適用

⑴ 概要

エンティティ・オブジェクトである Model 階層はデータベースへの永続化、トランザクション管理が必要であるため Entity Bean を採用した。永続化方式については CMP を採用し、データベース・アクセス・ロジックの記述を不要とした。

⑵ 関連の永続化

Java で参照の保有として実装されたオブジェクトの関連は、そのままリレーショナル DB に格納することはできないため、何らかの変換を行う必要がある。ここでは参照先オブジェクトのプライマリーキー項目を外部キーとして保存する方法をとった（図６参照）。

DB から関連を復元する場合は外部キーを用いて参照先の EJB を検索する。

このほかに EJB オブジェクトを一意に識別する EJB ハンドル[*2]を利用する方法も考えられる。参照先オブジェクトのハンドルをシリアライズ[*3]して DB に格納しておき、復元する場合はハンドルから EJB オブジェクトを取得することで関連の永続化を実現できる。しかし、EJB ハンドルは別のサーバーへの移行など実行環境が異なると無効になる可能性がある点、外部キーを格納することでテーブ

＊２）EJB ハンドル：EJB オブジェクトを一意に識別するもので、保存しておいた EJB ハンドルを用いて必要なときに元の EJB オブジェクトを再取得することができる。

＊３）シリアライズ：オブジェクトをファイルなどに格納できる形式に変換すること。

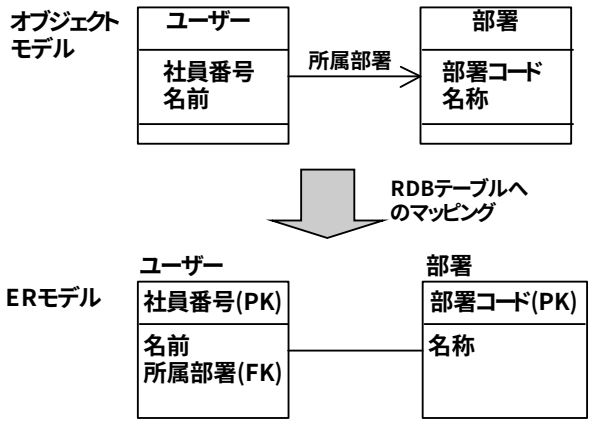


図6　実装オブジェクトの変換

ルの構造としても関連を表現でき、SQL でのアクセスも可能である点から前者の方法をとった。

(3) EJB 利用における留意点

Entity Bean で永続化の処理をコンテナが行うことは大きなメリットであるが、データベースと直結していることによる副作用も考慮する必要がある。例えば、EJB オブジェクトを生成した時点でデータベース・レコードが挿入されるため、そのオブジェクトを利用し終わるまでの長期間のトランザクションとなる可能性がある。また、作業用の一時オブジェクトとしての利用が難しいことや、EJB の現在の仕様では継承について規定されていないため場合によって実装上の工夫が必要となり、分析モデルを素直に表現できない可能性がある、といった問題もある。

以上の点に対する解決策として、Entity Bean をデータベース・アクセス用途に限定された DBMS ゲートウェイとして利用する方法が考えられる（図7参照）。

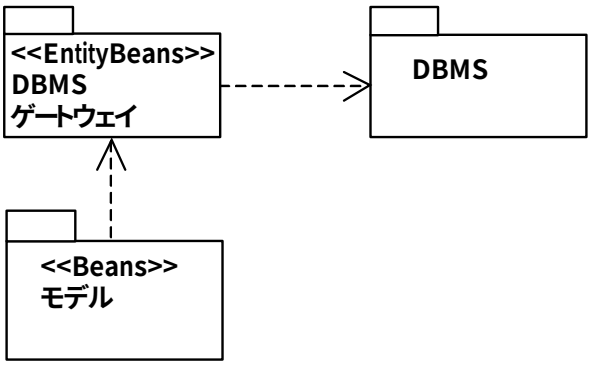


図7　Entity Beans を利用したゲートウェイ

この場合 Model を通常のクラスとして実装し、データベース・アクセス処理のみを DBMS ゲートウェイである Entity Bean に依頼する形になる。なお、この場合は JapanPLoP[*4]にて公開されている Persister デザイン・パターンを応用し、Entity Bean を Persister[*5]として利用する方法も考えられる。

このように EJB は手軽に各種サービスを利用することができるが、コンテナ処理や分散オブジェクト呼び出しのオーバーヘッドといった点も含め、メリット／デメリットと効果的な利用法を十分に検討する必要がある。

6．View 階層

6.1 関連技術

(1) Servlet

Web サーバーの拡張として、ブラウザなどからの HTTP リクエストを処理しレスポンスを返す仕組みであり、現在広く利用されている CGI に相当する役割をもつ。CGI と比較して、マルチスレッドによるプロセス管理オーバーヘッドの減少、他の Java 技術との統合による各種サービスの利用、異種プラットフォームへの高い移植性といった利点がある。

(2) JSP

サーバーサイドで動的に HTML などの Web コンテンツを生成するための仕組みで、JSP 拡張タグを用いた Java コードの記述や JavaBeans コンポーネントの利用が可能である。JSP ページはテキストベースの HTML ファイルとして作成するが、そこから JSP コンテナによってサーブレットが生成され、実行される。

(3) Applet

HTML に埋め込まれてブラウザ上で実行される Java プログラムであり、HTML のみを利用した場合に比べて高度なユーザーインタフェースやクライアントサイド・ロジックを構築することが可能となる。ただしサーバーからの Applet プログラムのダウンロード時間がかかることやブラウザが対応する JDK のバージョンによる非互換性といった制約がある。

(4) JavaScript

HTML 中に記述するクライアントサイド・スクリプトである。クライアントで処理を行えるため、サーバーとの通信を減らすことができるが、HTML に直接記述することによる保守性の低下や、ブラウザの種類による互換性に

＊4）JapanPLoP：パターンの普及を促進するために、国内におけるパターン流通の場を提供することを目的として設立された団体。（http : //www.kame－net.com/jplop/）

＊5）Persister：ビジネスオブジェクト（ここでは Model オブジェクト）に対して RDB 等の記憶媒体の検索や登録、更新を行う。記憶媒体への永続化処理を Persister が行うことでビジネスロジックとの分離をはかることができる。

ついて留意する必要がある。

6.2 プロトタイプへの適用

(1) View Controller

View Controllerはユーザー入力イベントを受け付けて適切なOperation Controllerに振り分け、その結果をViewに表示させる役割をもつ。ここではWebブラウザとの通信をHTTPで行う前提としたため、View ControllerにはHTTPリクエストを容易に扱うことのできるServletを採用した。

(2) View

ViewにおいてはHTMLベース、もしくはAppletを使用するという選択肢があるが、比較的単純な入力画面であり、特に高度なクライアントサイド・ロジックは不要なことからHTMLベースとした。ただし、サーバーとの通信回数を少なくするため、単項目チェックはJavaScriptを用いてクライアントサイドで行うこととした。

また、データをブラウザに表示するための動的HTML生成にはJSPを採用した。ServletにHTML生成ロジックを記述することも可能であるが、JSPは画面表現部分のみが分離されるため保守性の向上が図れるほか、テキストベースのHTMLファイルとして作成するためHTMLエディタが利用できるといった利点がある。

7．Controller

7.1 関連技術

(1) EJB

５章参照のこと。

7.2 プロトタイプへの適用

View ControllerがオペレーションのユーザーインタフェースⅡ御を行うのに対し、Operation ControllerはビジネスロジックⅡ御の役割をもち、View Controllerによって振り分けられたユーザー入力イベントに応じてModelを操作し、その結果をView Controllerに返す。また、Modelに対するトランザクションⅡ御もここで行ない、クライアントのセッションごとにその状態を保持しておく必要がある。

以上の条件から、クライアントごとにインスタンスが作成され、複数回の呼び出し間で状態を保持するStateful Session Beanを採用した。また、EJBがサポートする分散オブジェクトサービスを利用して、View階層をJavaアプリケーションやCORBAアプリケーション（RMI over IIOPをサポートしたEJBコンテナの場合）に置き換えることも可能となる。

8．おわりに

今回作成したプロトタイプは、J2EE技術全般の検証という観点と限られた時間からごく単純なシステム要求を設定したが、その範囲ではEJBを筆頭にJ2EE技術による生産性向上や柔軟なアーキテクチャの実現といったメリットを感じることができた。今後は、より大規模なモデルへの適用や再利用性を念頭においたアーキテクチャの検証などを課題としていきたい。

〈参考文献〉

1．F．ブッシュマン，R．ムニエ，H．ローネルト，P．ゾンメルラー，M．スタル：ソフトウェア・アーキテクチャ，トッパン，1999

2．Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition（http://java.sun.com/j2ee/blueprints/index.html）

3．Richard Monson-Haeful：Enterprise JavaBeans, O'Reilly & Associates, 1999

4．I．ヤコブソン，G．ブーチ，J．ランボー：UMLによる統一ソフトウェア開発プロセス，翔泳社，1999

5．I．ヤコブソン：オブジェクト指向ソフトウェア工学OOSE，トッパン，1995

6．JapanPLoP Webサイト（http://www.kame-net.com/jplop/）