Notes と Java Servlet による Web 購買システム

関西営業所　柴田正昭

１．はじめに

Web システムの開発事例が、SOFTECHS 誌上を賑わせている昨今であるが、関西営業所にも遅ればせながら昨年（1999年）末に、ビジネス・チャンスが巡ってきた。しかも、最新の Java Servlet による、Web システムの構築である。本稿では、このプロジェクト事例を紹介する。今後の Web システム開発に、わずかながらでも役に立てば幸いである。

２．システムの概要

本システムは、関西電力株式会社殿（以降、関西電力殿と称す）の資材部門における購買システムの一部で、部門外契約を対象としたシステムである。部門外契約は事務用品や消耗品などを対象品目とし、契約額は少額であるが、契約件数が多い上に承認行為が多く、手続きに時間が掛かってしまう。そこで、本システムでは承認行為を見直して簡略化した上でワークフローを導入し、契約手続に費やす工数の大幅な削減を図った。

請求から発注、検収までの処理を Web で行い、検収完了データをホストで稼動している資材基幹システムに接続して、ホストで計上・支払い処理を行う。他システムとのインタフェースは、資材基幹システムの他に２つある。１つは取引先への注文書メールを送信するための EDI（Electronic Data Interchange）サーバーへのデータ送信である。部門外契約取引では、当面はメールと FAX によって発注をサポートするが、将来的には EDI 取引への発展も構想に組み込まれている。もう１つは、経理部門とのリアルタイムなデータ授受である。これは、経理部門に請求データを渡して、経理項目の妥当性チェックを行う仕組みになっている。

図１に購買システムのワークフローと他システムとのインタフェース・ポイントを示す。

ワークフローの通常の流れはシンプルであるが、承認却下、照査却下に加えて契約内容の変更があり、全てを組み合わせると何十通りものフローになり、契約手続の進捗状況の管理を複雑にしてしまっている。

３．システムの構成

3.1　機能構成

本システムでは、Notes（FormWave）、Web（Java Servlet）、さらに DB サーバー上で C、および PL/SQL にて開発したバッチを使い分けた。それぞれの役割分担は、図１のワークフローに示すとおりである。

Notes に FormWave[*1]をアドインしたワークフローを、ユーザー・インタフェースの中心に置き、請求データの登録、検収データの入力、照査や承認のためのデータ照会を Web アプリケーションによって補った。取引先への注文書メールの発信やホストへのデータ接続は、バッチによって実現した。

開発は、関電情報システム株式会社殿（以降、KIS 殿と称す）との共同開発で行った。Notes とバッチのホストに関連する機能については、関西電力殿の保守を長年にわ

＊１）FormWave：申請承認回覧業務で発生する各種伝票・帳票などを、現行のフローに即して簡単に電子化でき、ワークフローシステムが簡単に構築できる「電子フォーム支援システム」。Notes 既導入が前提となる。（IBM 製）

たって担当し、資材システムに長じているKIS殿が開発を担当した。Webとバッチの注文書作成やメール送信などの機能はCACが開発を担当し、それぞれの特性を活かせるように分担して作業にあたった。

3.2 システム構成

システム構成の検討は、システムのプラットフォームの選定と検証から始めた。Notesは既に導入されており、グループウェアとして本番運用されているので、当初から採

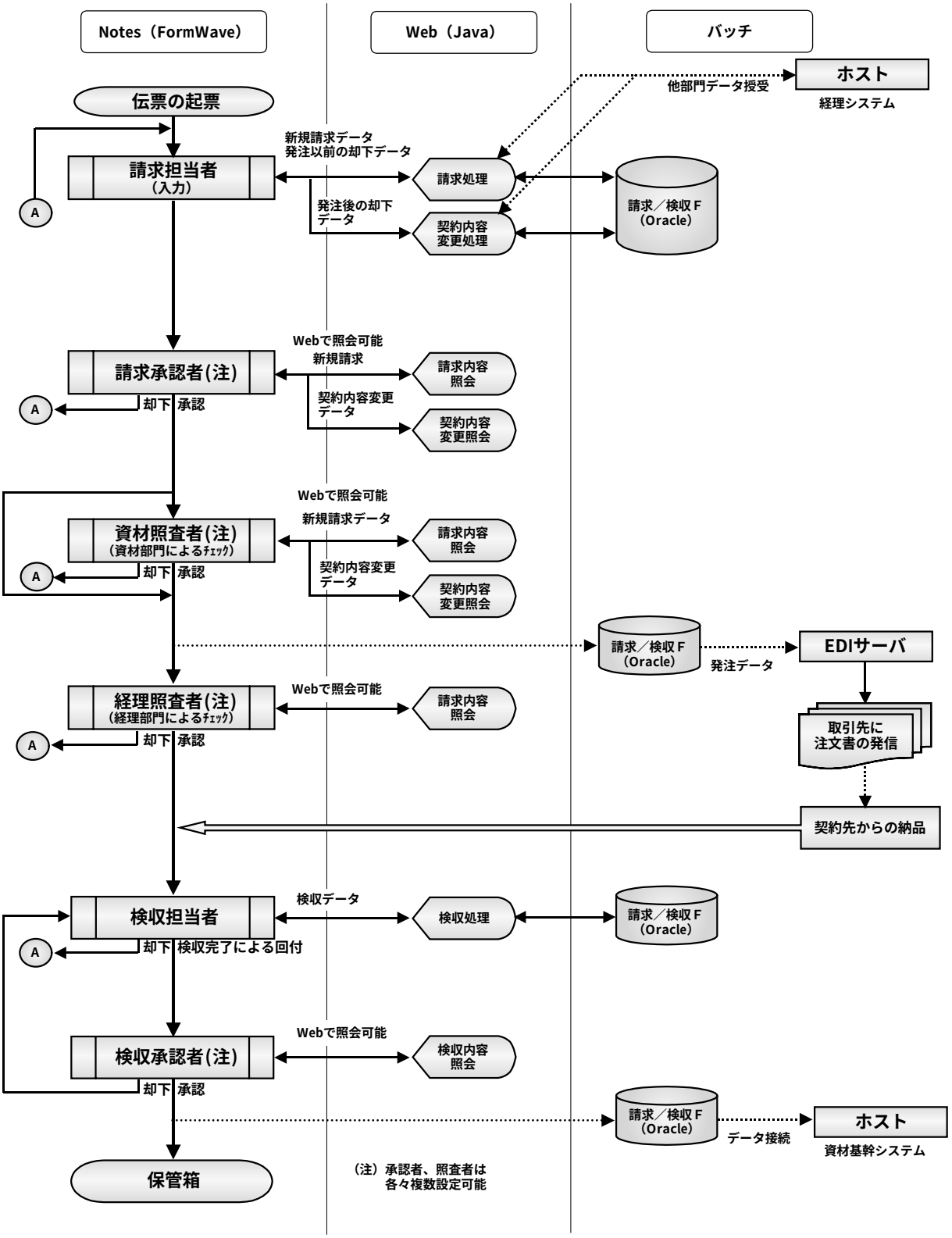

図1　購買システムのワークフローとインタフェース・ポイント

用が確定していた。しかし、Webアプリケーションについては、OAS（Oracle Application Server）とPL／SQLの組み合わせやNotes Dominoなども候補に挙がった。最終的には、関西電力殿のWebアプリケーション構築標準である、という理由でJavaに決着した。

Javaのプラットホームも検討段階でNES(Netscape Enterprise Server）からOASへ、さらにWeb Sphereへと変遷していった。NES（バージョン3.51）ではセッション管理クラスがサポートされていないため、複数画面にまたがる処理が実現できない。業務要件を単一の画面で実現することは不可能なので、セッション管理クラスをサポートするOASに切り替えた。OASからWeb Sphereへの切り替えの経緯については後述する。

FormWaveからWebへは、FormWaveの伝票にURLを貼り付けてIE（Internet Explorer）を起動する仕組みになっている。Servletは機能ごとに作成しているが、伝票には固定のURLしか指定できないので、振り分け処理用Servletを間にはさんで、データの状況に応じて目的のServletに振り分けている。

FormWaveでWebから入力したデータを参照したり、承認結果をDBに反映する必要がある。そこで、ダミー未決箱[*2]を設けて、DBに対して伝票表示に必要な項目の取得や、手続進捗状況の更新を行っている。

システム構成を図２に示す。

４．プロジェクトの始動

プロジェクトの開始にあたって、最初の課題はJava技術の習得であった。関西営業所が単独で準備していたのでは時間がかかりすぎるので、産業事業部に技術支援を要請した。折りよく当社では、Java技術展開の推進を本格的に開始した時期であり、Javaエンジニア育成プログラムにプロジェクト参画予定メンバーをタイミングよく投入でき、さらに技術研究室をはじめ、社内各部署の積極的な協力も得られた。Java開発の中心となるメンバーには、Java Servlet、Java Beansなどの研修を受講させることができ、Javaの全社的な技術展開という追い風を受けて順調な滑り出しであった。

ターゲットとなるシステムの開発環境が整うまでの間、営業所内に暫定的な開発環境を構築してHTML自動生成、DBアクセス、セッション管理などJavaの動作検証に努めた。暫定版の開発環境はLinuxを搭載したPCサーバー

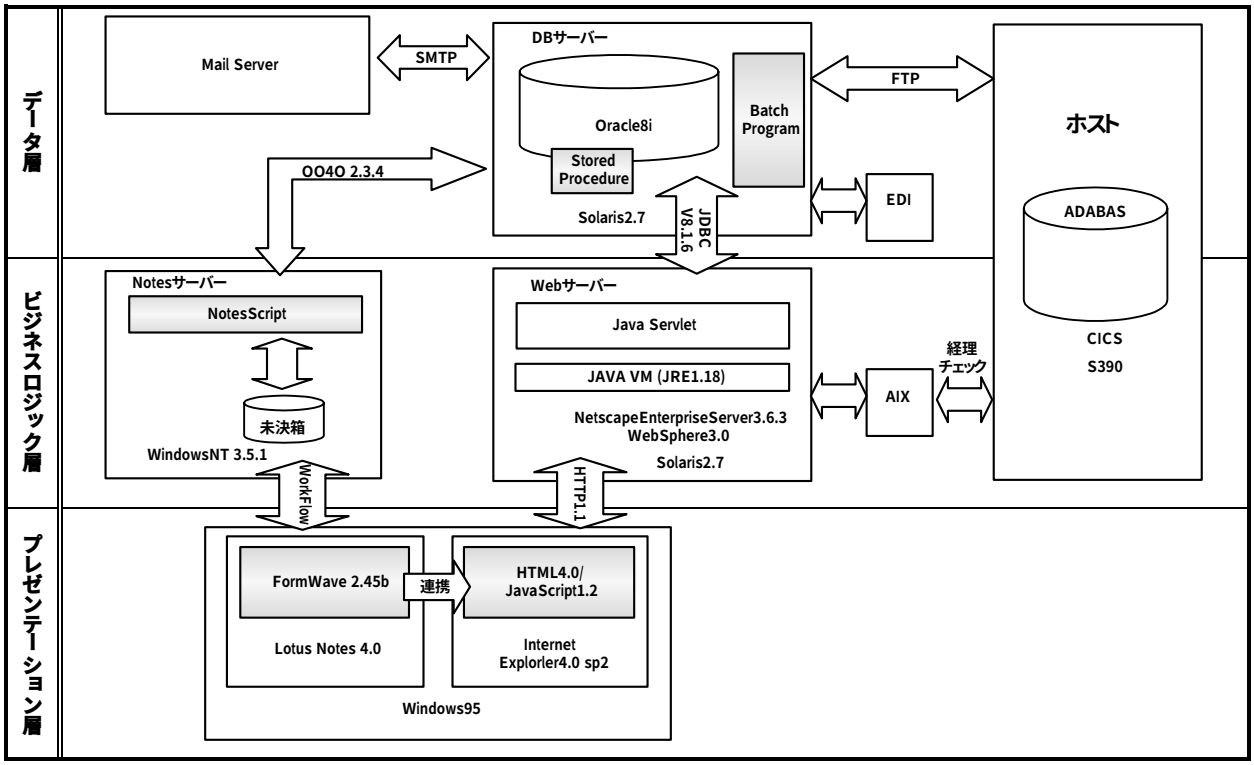

図２　システム構成

＊２）ダミー未決箱：FormWaveは、Notesサーバー上に作られた複数の未決箱の間をデータ移動してフロー制御を実現するが、途中にユーザーに見えない未決箱を設け、機械処理を行わせている。この機械処理用の未決箱をダミーの未決箱と称している。ダミーの未決箱からVB Scriptを起動して、DBサーバーへのアクセスなどを行う。

をプラットフォームに、WWW サーバーに Apache1.3.6、アプリケーション・サーバーに Apache Jserve1.0、Java 開発環境は JDK1.1.7、JSDK2.0、DB は Oracle8.0.5（JDBC8.0.4.0.5）を搭載した。

動作検証は予想以上に手間取り、定番の“Hello World”を表示するまでに、実に２週間も費やしてしまった。そのほとんどは環境構築の工数であり、本番のターゲット開発環境でも同様の問題の発生が懸念された。

５．プロトタイプの作成

Java の動作検証と並行して、プロトタイプ（フレームワーク）の作成に着手した。プロトタイプの成否がこのプロジェクトの成否に関わるので、産業事業部の技術陣とレビューを繰り返し、プロトタイプ完成までに３度版を重ねた。

第１版、第２版、第３版の各プロトタイプの構造を図３、図４、図５に、それぞれ示す。

プロトタイプ第１版は継承（Extends）を前面に押し出しているが、フローが判りにくい上に、画面数の分だけ Servlet が必要になってしまい、実装するには難があった。第２版は、第１版から作成方針を180度転換して、すべての継承をインタフェースに置き換えたが、フローが１つの Servlet に集約されてしまうことで処理が集中してしまい、レスポンスが悪くなる結果となった。そして、プロトタイプの完成版となる第３版（Xbase と命名）は、第1,2版の欠点を補い、継承とインタフェースをバランスよく配置しており、Servlet 数も機能の分で済み、管理上も申し分ない出来となった。第３版は技術研究室にてモデリングした。

Xbase には、オブジェクト指向型の開発設計に基づいたデザインパターンが採用された。Java は C＋＋などと同様にオブジェクト指向言語であり、プロトタイプを作成する上では、オブジェクト指向設計は必須であったが、我々の当初の知識では「オブジェクト指向＝継承」という程度で、デザインパターン利用技術を習得しながらの開発となった。デザインパターンを適用するには熟練が必要な上に、適用の正しさを検証する必要があり、単に知識として知っているだけでは使いこなせないことを、身をもって経験した。

第１版から第３版までの各プロトタイプの特徴と長所・短所を表１に示す。

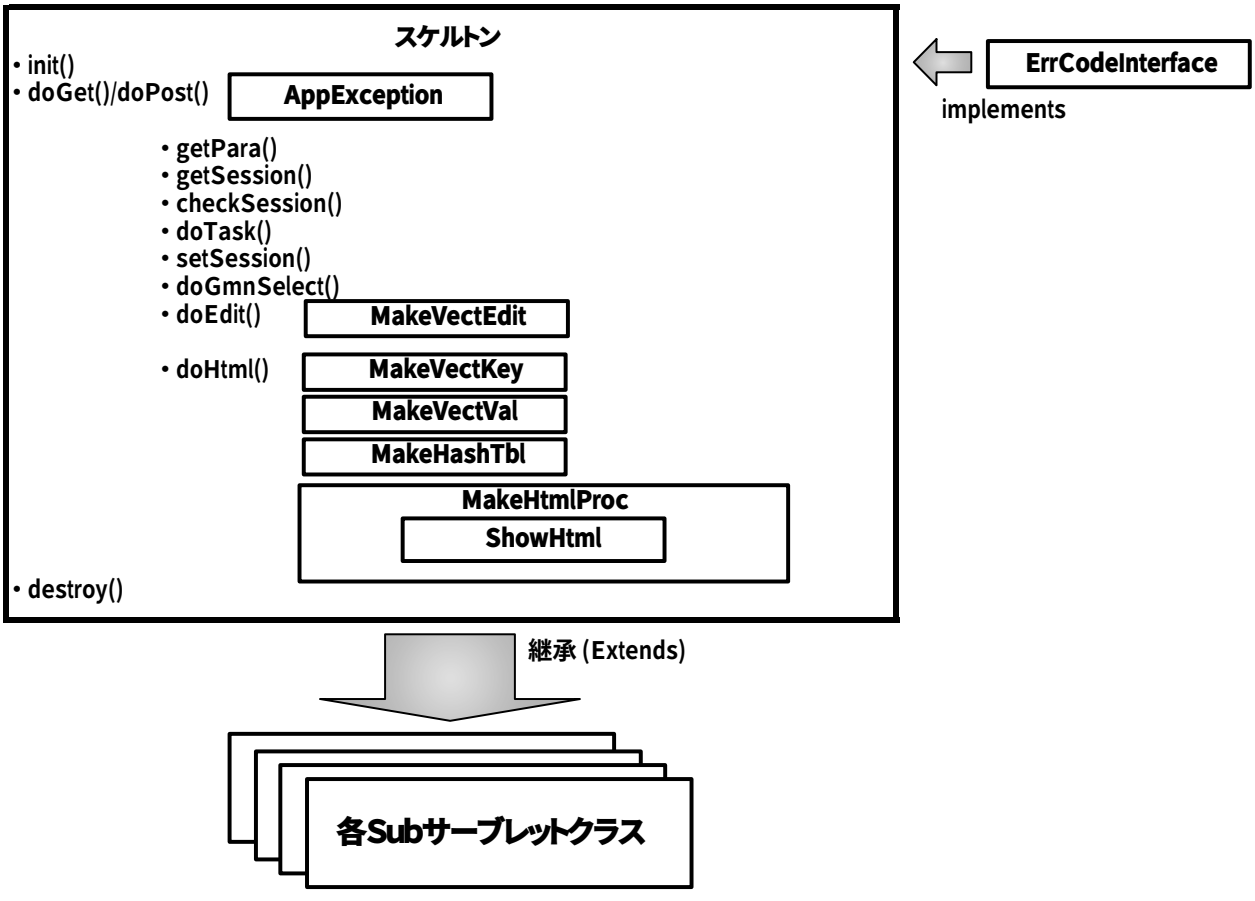

図３　第１版プロトタイプ構造

BaseServlet

```
import   kelib.xxxx.*;

public   class   Baseservlet
           extends          HttpServlet
           implements   ErrMsgInterface{

private   SkeltInterface   skeltinterface;

Init()
{}
doGet(){
      doProc();
}
doPost(){
      doProc();
}
doProc(){
      getPara();
      doGmnSelect();
      doInit();
            •
            •
            •
      doOutput();
}
destory()
{}
doErr()
{}
dohtmlErrSend()
{}
private getPara()
{}
private doGmnSelect(){
      共通処理
      個別処理
      if(インタフェース1の時){
            skeltinterface   =   new Pseikyu20IF;
      }else   if(インタフェース2の時){
            skeltinterface   =   new Pseikyu21IF;
      }else
                              •
                              •
                              •
      }
}
private doInit()
{
      共通処理
      個別処理
      skeltinterface.doinit();
}
            •
            •
            •
private doOutput()
{
      共通処理
      個別処理
      skeltinterface.doOutput();
}
```

インタフェースのクラス
（Skeltoninterface）

```
packege   kelib.xxxx;

public interface SkeltInterface {
      doInit();
      checkPara();
      getSession();
      checkSession();
      doTask();
      setSession();
      doOutput();
}
```

インタフェース1（Pseikyu20IF）
の実装クラス

```
packege   kelib.xxxx;

public   class   Pseikyu20IF
  implements   Skeltinterface,
                  ErrMsgInterface{
      doInit()
      {}
      checkPara()
      {}
      getSession()
      {}
      checkSession()
      {}
      doTask()
      {}
      setSession()
      {}
      doOutput()
      {}
}
```

インタフェース2（Pseikyu21IF）
以降の実装クラスも同様

図４　第２版プロトタイプ構造

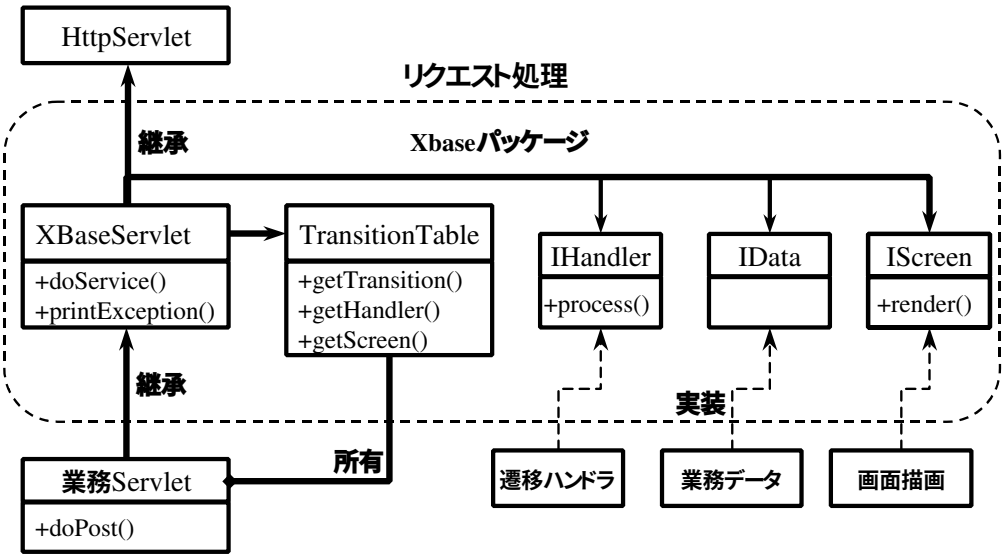

図５　第３版プロトタイプ構造

表１　各プロトタイプの特徴と長所・短所

	第１版	第２版	第３版
特徴	・継承を利用し、共通機能をスーパークラスにまとめ、各サーブレットは、スーパークラスの各メソッドをオーバーライドして、各個別機能の実現を行う。 ・サーブレットは画面数分存在する。 ・処理フローは、スーパークラスが担う。	・継承を廃止し、インタフェースを利用。 ・サーブレットは全体で一つ。画面数分、インターフェス実装クラスが存在する。 ・処理フローはインタフェースを実装した各クラスが担う。	・継承と、インタフェースの双方を利用。 ・サーブレットは業務単位に存在する。 ・処理フローはスーパークラスが担う。 ・遷移情報はファクトリを使って抽象化し、XML ファイルに外出しする。 ・遷移処理だけでなく、データクラス、画面出力クラスもインタフェースの実装を行っている。 ・Factory パターン、Strategy パターン等、オブジェクト指向に基づいたデザインパターンを採用。
長所	・共通処理のスーパークラスにまとめられ、冗長性が少ない。 ・構造が単純で分かりやすい。	・処理フローを各クラスが担うため、処理が分かりやすい。 ・共通処理がスーパークラスにまとめられ、冗長性が少ない。	・共通ロジックにはスーパークラス、個別ロジックにはインタフェースを用いて、継承を用いることによるデメリットがおさえられている。 ・画面の遷移情報を外部ファイルとして外出しして、実装依存のコーディングを防いでいる。 ・遷移処理、データ処理、画面出力処理と機能を分割したクラス設計により、ベースサーブレットの構造を知らなくても業務サーブレットの開発が容易である。 ・サーブレットが業務単位で分割していて、管理しやすい。
短所	・処理フローをスーパークラスが担い、各サーブレットは必要時のみ部分的にメソッドのオーバーライドを行うため、各サーブレットから全体のフローが読みとりにくい。 ・画面数サーブレットが存在し、そのすべてがサーバーに常駐してしまう。 ・継承構造が基本となっているため、保守性、再利用性が低い。	・サーブレットが全体で１つであるため、１サーブレットで全業務をまかなう構造となってしまう。 ・遷移情報がハードコーディングされている。	・ベースサーブレットの構造が複雑で分かりにくい。

６．量産体制の確立

１本目の Servlet は作成完了まで、約８人月かかった。Servlet の中で最も難易度が高かったということもあるが、共通クラスの見直しと作成、さらに Xbase の理解徹底などの作業に時間がかかったためである。しかし、Xbase の完成により、アプリケーションの構築は、表２に示す10種類の業務クラスから必要なクラスを選択して、実装するだけの作業にパターン化できた。この段階で OAS の環境も把握でき、開発が軌道に乗ってきた。

プログラム設計書は詳細設計と Java Doc（HTML ドキュメントを作成する JDK のツール）を使用した。本番環境は JDK1.1.7バージョンであったが、JDK1.2を利用した方がドキュメント完成時のデザインがよくなるので、Java Doc を作成する時だけ JDK1.2を使用した。

表２　業務クラス一覧

クラス名（機械名称）	クラス名（日本語名称）	クラス説明
XXXHandlerY_Z	ハンドラークラス	画面遷移により発生する処理を担うクラス。画面遷移のパス分存在する。 主に、パラメータデータの取得、DB 検索、セッションデータの編集を行い、次画面遷移用情報、および、 DB 更新用データのセットを行う。
XXXScreenY	スクリーンクラス	ハンドラークラスにより編集されたセッションデータを、HTML のデータ項目名をキーとするハッシュテーブル の値に格納する。この際、数値のカンマ編集等、表示に関する制御を行う。
XXXParamY	パラメータクラス	画面からの入力値等、パラメータの値を取得するクラス。複数のハンドラーでパラメータの取得処理が重複した場合、 あるいは、パラメータから取得した値の編集内容が多い場合に作成し、各ハンドラーからメソッドの呼び出しを行う。 入力処理がない遷移の場合は不要。
XXXCheckY	関連チェッククラス	パラメータから取得した値の関連チェックを行うクラス。この処理によりエラーとなった場合、次画面への遷移を行わず、 同画面にてエラーメッセージの出力を行う。ただしワーニングの場合は次画面にてワーニングメッセージの表示を行う。 入力処理がない遷移の場合は不要。
XXXExistCheckY	存在チェッククラス	パラメータから取得した値の存在チェックを行うクラス。DB 検索によりレコードの存在チェックを行い、不存在の場合には 同画面にてエラーメッセージの出力を行う。コード等のレコードが存在した場合には、相当するコード名称の再設定を 行う項目もある。これらの一連の DB 検索処理は DBAccessor を介さない。
XXXDBAccessor	DB アクセスクラス	DB 検索の場合には、DB 検索クラスのメソッド呼び出しを行った後、その処理で DB アクセス用データクラスに格納された 検索結果を、業務データクラスに格納する。（DB アクセス用データクラスと業務データクラスが同一の場合は不要。） DB 更新の場合には、業務データを DB アクセス用データクラスに格納した後、DB 更新クラスのメソッド呼び出しを行う。
XXXFinder	DB 検索クラス	DB アクセスクラスから呼び出される DB 検索メソッドを保持するクラス。
XXXWriter	DB 更新クラス	DB アクセスクラスから呼び出される DB 更新メソッドを保持するクラス。
XXXData	業務データクラス	セッションにより管理される業務データを保持するクラス。業務間で共通のデータ型は kelib.data 内で定義される。
XXXServlet	サーブレットクラス	業務サーブレットクラス。Init()メソッドで DB コネクションと遷移情報の解析・登録処理を行う。 doService()メソッドでは、スーパークラス XBaseServlet クラスの doService()メソッドが実行される。

XXX・・・・・　業務パッケージ ID
Y, Z・・・・・　画面 ID

７．最後の試練

ホストと他部門サーバー間のデータ授受は、当初は MQ Series[*3]で実現する予定であった。しかし、MQ Series では送受信の全てがタイマー監視で実現されているため、１時間以上ものタイムラグ発生の可能性が予想され、即時に発注を要する緊急処理には対応できない。そこで、リアルタイム処理を必須要件として、ホストと CICS[*4]（Customer Information Control System）連携が可能な Web Sphere の採用が決定した。Web Sphere では、CTG（CICS Transaction Gateway）により、Java から CICS へのアクセスを可能にしている。OAS から Web Sphere の切り替えに伴い、OS も Soralis2.6から2.7に、JDK も1.1.6から1.1.7に切り替えた。しかし、７月末の本番稼動まで、残すところ２カ月となった５月末である。暫定環境、開発環境と環境設定がネックとなってきた経緯を考えると、納期や品質を厳守できるか懸念もあったが、結果的には切り替え作業は特に問題なく終了できた。稼動継続するにつれ、Web Sphere でもログが累積されて起動できなくなる等、多少の問題は発生したが、広い世界シェアを誇る Apache をベースとしているだけに、動作の安定度は高いものがあった。

８．Java の評価

Java Servlet について生産性、互換性、性能の点から若干の考察を行ってみる。

8.1　生産性について

表３に Java Servlet の開発実績を示す。

表内の平均１、および合計１は全 Servlet を対象として

＊３）MQ Series：単一の API、非同期処理により、分散して構築された複数の自動化されたシステムを統合し、一貫した開発を可能にするメッセージング・ミドルウェア。（IBM 製）

＊４）CICS：ミッション・クリティカルなアプリケーションに対して、実績あるオンライン・トランザクション管理機能を提供するアプリケーション・サーバー。（IBM 製）

表3　Java Servlet の開発実績

Java Servlet	HTML									Javaクラス					Stored Procedure					XML			DB		工数合計
	本数	ステップ	JS ステップ	サイズ	入力数	出力数	釦数	工数	工数平	本数	ステップ	サイズ	工数	工数平	本数	ステップ	サイズ	工数	工数平	ステップ	サイズ	工数	入力	出力	
E44A101	10	3,280	866	102	258	1994	35	11.0	1.1	46	7,762	288	113.0	2.5	7	1,654	84	22.0	3.1	217	8	10.0	6	3	156.0
E44A102	4	2,221	712	70	77	373	14	2.0	0.5	28	6,656	248	22.5	0.8	4	1,137	58	2.5	0.6	102	4	0.5	5	3	27.5
E44A103	4	1,031	41	32	0	476	8	2.0	0.5	17	2,546	107	14.0	0.8						76	3	1.0	5	1	17.0
E44A104	3	967	83	22	2	214	9	1.5	0.5	19	2,558	104	13.5	0.7	1	67	4	0.5	0.5	72	3	0.5	7	1	16.0
E44A105	2	408	21	12	0	177	6	0.2	0.1	12	1,654	72	4.7	0.4						42	2	0.1	7	0	5.0
E44E100	2	289	101	9	5	6	3	1.0	0.5	8	597	15.5	2.8	0.4						29	1	0.5	0	0	4.3
E44E101	2	217	44	8	303	1705	6	1.0	0.5	16	1,218	41	9.5	0.6	1	40	2	0.5	0.5	49	2	0.5	1	1	11.5
E44E102	2	279	72	9	24	141	10	1.0	0.5	19	2,105	68	10.5	0.6						74	3	0.5	1	1	12.0
E44E103	7	801	200	28	32	460	21	3.5	0.5	30	2,505	85	16.5	0.6	3	119	4	1.5	0.5	143	6	0.5	1	1	22.0
E44E104	3	686	209	22	10	66	13	1.5	0.5	22	2,428	90	15.5	0.7						102	4	0.5	5	0	17.5
E44Z100										6	809	22	1.8	0.3									0	0	1.8
E44Z101	7	1,751	912	59	7	39	24	3.5	0.5	28	3,621	103	11.6	0.4						83	3	0.5	2	0	15.6
E44Z102	3	685	164	24	14	53	12	1.5	0.5	17	1,469	43.2	6.1	0.4						69	3	0.5	1	0	8.1
E44Z103	3	604	184	22	14	33	12	1.5	0.5	17	1,321	40	6.1	0.4						70	3	0.5	1	0	8.1
E44Z104	1	387	30	15	50	3	2	0.5	0.5	6	368	11	1.8	0.3						23	1	0.5	0	0	2.8
E44Z105	1	279	36	12	40	3	2	0.5	0.5	6	363	11	1.8	0.3						23	1	0.5	0	0	2.8
E44Z106	1	113	30	4	1	3	2	0.5	0.5	6	365	11	1.8	0.3						23	1	0.5	0	0	2.8
E44Z107	1	109	19	4	40	200	2	0.5	0.5	11	733	22	3.1	0.3						23	1	0.5	1	0	4.1
E44Z108	1	286	39	10	11	3	2	0.5	0.5	6	364	11	1.8	0.3						24	1	0.5	0	0	2.8
E44Z109	1	153	37	6	12	3	2	0.5	0.5	6	364	11	1.8	0.3						24	1	0.5	0	0	2.8
平均1	3.1	766		24.7	47.4	313.3	9.7	1.8	0.5	16.3	1,990	70.2	13.0	0.6	3.2	603	30.4	5.4	1.1	67	2.7	1.0	2.2	0.6	17.0
合計1	58	14,546		470	900	5952	185	34.2	0.6	326	39,806	1403.7	260.2	0.8	16	3,017	152	27.0	1.7	1,268	51	19.1	43	11	340.5
平均2	2.7	626		20.4	35.7	219.9	8.3	1.3	0.5	14.7	1,687	58.7	7.7	0.5	2.3	341	17	1.3	0.5	58	2.4	0.5	1.9	0.4	9.7
合計2	48	11,266		368	642	3958	150	23.2	0.5	280	32,044	1115.7	147.2	0.5	9	1,363	68	5	0.6	1,051	43	9.1	37	8	184.5

注釈1）　平均1、合計1：全 Servlet の平均、合計。平均2、合計2：E44A101を除く Servlet の平均、合計。
　　E44A101 には Xbase 作成の工数も含まれているので、アプリケーション構築工数には、平均2、合計2を適用する。
注釈2）　工数：合計工数（人日）。工数平：平均工数（人日／本）。
注釈3）　サイズの単位はKB。

おり、平均２と合計２は、１本目（E44A101）を除いた Servlet を対象としている。１本目の開発時に、クラス設計・製作を合わせて行っているので、純粋にアプリケーションの生産性を図るには、平均２と合計２の数値を採用する。Servlet 作成作業は HTML（画面）、クラス（ソース）、Stored Procedure（DB 更新時のみ使用）、XML（画面遷移情報を定義）を作成することである。プログラム設計から単体テスト完了まで１つの Servlet あたり、平均9.7日、Java クラスの平均ステップ数が1,687ステップなので、１日あたり174ステップ／日と、初めての開発担当としては、まずまずの生産性が得られた。しかし実感としては、Java を採用することにより生産性が上がる、とすることには多少の補足が伴うようだ。きちんとクラス設計を行い、徹底した部品化を行えば、生産性を上げることは可能である。だが、それはある程度のシステム規模と開発期間が確保できた場合であり、短期間の開発の中で生産性を上げることは、とても難しい。

8.2　互換性について

前章で述べたように、期せずして JDK のバージョンアップと Java プラットホームの変更を経験した。JDK1.1.6から JDK1.1.7にバージョンを上げるのは、いわゆるマイナーチェンジなので問題ないことは予想していたが、OAS から Web Sphere への切り替えがスムーズに運ぶとは、あまり思えなかった。Java は実行環境を選ばない、という喧伝ではあるが、実際には環境（プラットフォーム等）ごとに異なるモジュールを必要としたという過去の経験もあり、さらに暫定環境から開発環境への移行時に文字化けなどで苦労した経緯もあってのことである。

しかし、悪い予想に反して OAS から Web Sphere への切り替えでは、ほとんど問題が生じなかった。強いて発生したといえば、日本語のエンコーディング方法の違いと、Web Sphere では XML パーサー（XML ファイルを操作するクラス）が独自機能であったために手直しを必要とした、というくらいである。Java Servlet で作成している限りでは、アプリケーション・サーバーの違いによる互換性の問題は、ほとんどないといえる。非推奨の API を使用しないように、極力注意したことも功を奏した。

8.3　性能について

インタープリタ言語ということで、当初からレスポンスの悪さを危惧していたが、Servlet のみで作成したためか、体感的には全く問題なかった。Web サーバー[*5]立ち上げ直後に起動すると、Servlet をメモリーにロードするために数分かかるが、立ち上げ時の１回だけなので、これは実務上の問題点にはならない。Web サーバー起動時に自動的に Servlet を起動する設定にすれば回避できる。

製作の中で性能に関して考慮したのは、DB 接続と入力チェックの２点である。図６に Servlet の構造を示す。

DB の接続／解除のオーバーヘッドを避けるために、初期処理の init（ ）メソッドで DB の接続を行い、destroy（ ）メソッドで DB の接続解除を行っている。Web Sphere

---

＊５）Web サーバーには、UNIX サーバー「GP7000S モデル45」を使用。400MHz の UltraSPARC-Ⅱを２CPU、メモリを１GB 搭載。（富士通製）

＊６）コネクションプール：データベースへのセッションを事前に確保しておき、複数の Servlet で共有する機能。

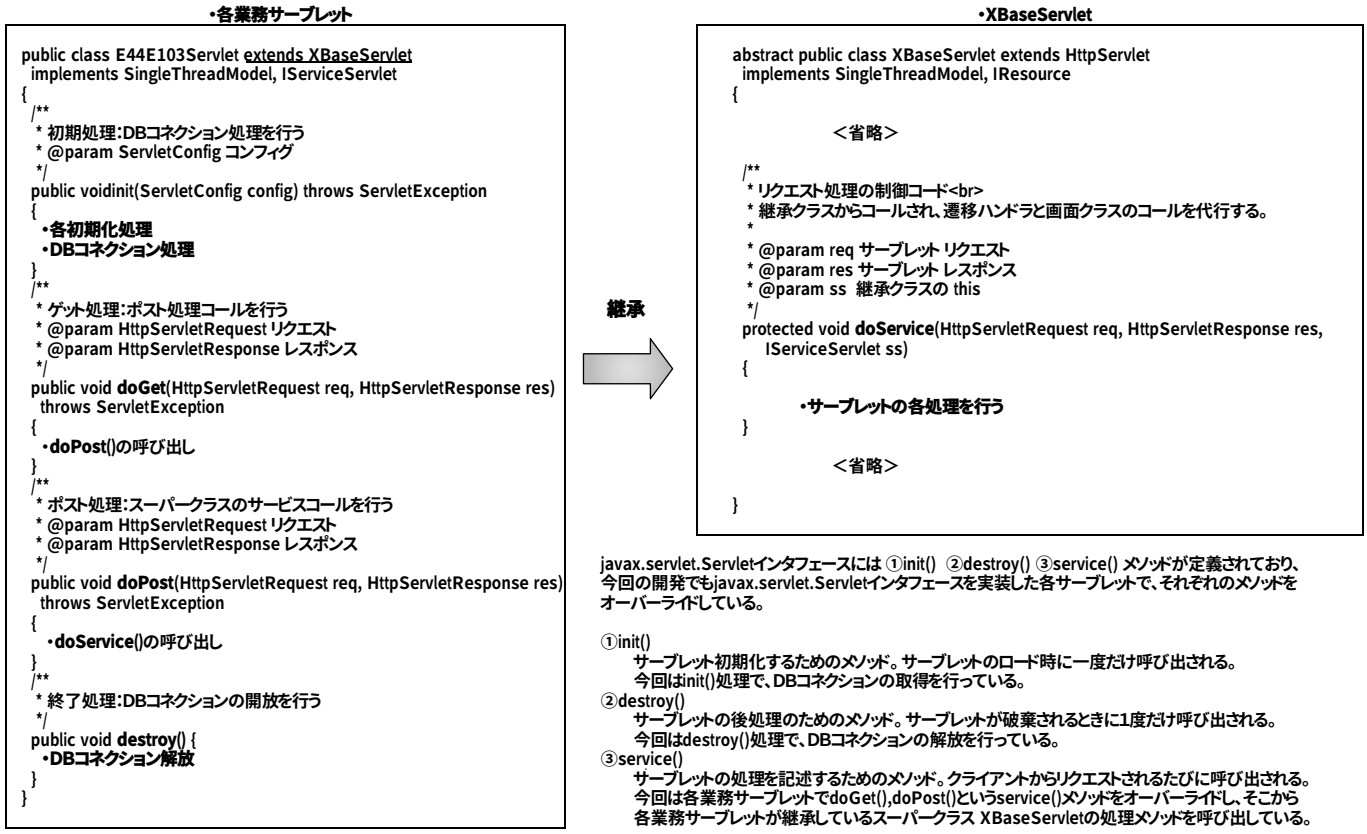

図６　Servlet の構造

にはコネクションプール[*6]の機能があるが、今回は使用していない。理由は、シングルスレッドモデルを推奨するOAS で開発したためである。自力でコネクションプールすることも考え、部品なども用意したが、DB が本システム専用に割り当てられており、接続数の制約が少なかったこともあり、あえて単純な方法を選択している。

Unicode とマッピングできないコード[*7]は入力不可としているが、入力チェックは Java Script にて行い、サーバーに無駄にアクセスしないで済むように配慮した。

９．おわりに

本プロジェクトは、成功裡に終了することができた。しかし、あえて反省の目をもって振り返ってみると、環境構築やプロトタイプ作成の過程において、今後に改善すべき点がいくつかある。特にプロトタイプ作成はモデリングの良し悪しで、その後の展開が大きく違ってくる。今回は産業事業部の技術陣の支援により乗り切ることができたが、Java で開発を行う以上、プロジェクトの立ち上げ当初にオブジェクト指向を理解し、クラス設計をできる人間を育てることが必要であった。ただし、これは即席にはいかないので、日頃の訓練が必要でもある。

本プロジェクトで、Java 開発に必要な要素は何であるか、実践を伴って理解できた。しかし、まだ Java の世界にほんの少しだけ足を踏み入れた程度である。JSP（Java Server Pages）、EJB（Enterprise Java Beans）など、未経験の Java 技術はまだまだある。昨今では、Java の将来にかげりが見えた、という記事もときおり見かける。しかし、富士通、日本 IBM 他の６社が、Java 技術を活用した業務用ソフト開発の効率化のための協議会「EJB コンポーネンツに関するコンソーシアム」(仮称）を設立する、というニュースが報じられたことでも、当分の間は Java の活躍は続きそうである。

今後、関西営業所で Java 技術のスキル拡充を図るためには、オブジェクト指向設計者の教育と、今回において基礎を築いた Xbase の評価、再利用が必須と思われる。

最後に、技術研究室、ならびに Java メーリングリストにてご指導頂いた方々に、この場を借りて感謝いたします。

＊７）Shift-JIS コード等を Java の内部コードである Unicode に変換して内部処理を行った後、元のコードに変換しても同じコードに戻らないコードがある。これをマッピングできないコードといい、ローマ数字や CJK コード（中国、日本、韓国の漢字をもれなく定義するためのコードセット）などが該当する。