バリアフリーＪａｖａフレームワーク
―　「難しい」を「簡単」に　―

ささき　ひろゆき
佐々木　宏之
〔略歴〕
１９９０年　八十二システム開発株式会社入社
現在　　開発三部
第２グループ 主任システムエンジニア
八十二銀行システム部に派遣
システム経験年数　１２年

たけい　ちとせ
武井　千登世
〔略歴〕
１９９３年　八十二システム開発株式会社入社
現在　　開発三部
第２グループ 主任システムエンジニア
八十二銀行システム部に派遣
システム経験年数　　９年

原稿量
本文　　１４，５９３字
要約　　１，１９５字
図表　　１５枚

＜要　　約＞

株式会社八十二銀行は、ＣＲＭシステム開発プロジェクトを契機に「Ｊａｖａ言語を使った大規模システム開発の中で発生するさまざまな障壁を取り除き、Ｊａｖａ開発の初級者であっても高生産性・高品質のシステム開発が行える環境を提供する」ことを目指して「バリアフリーＪａｖａフレームワーク」を構築した。

Ｊａｖａを使った大規模システム開発の難しさを解決するには、初級者を含むすべての開発者が共通に効率よく作業できるソフトウェア環境であるフレームワークが必要となる。そこで、生産性・品質向上、利便性向上、システム資源削減及び保守性向上をねらいとして、「難しい」を「簡単」にできるフレームワークを自ら構築した。構築する上での工夫及び考慮点を以下に示す。

（1）Ｍｏｄｅｌ（ビジネスロジック）層のバリアフリー

Ｍｏｄｅｌ層では、開発者がビジネスロジックのみに専念して開発できるようにすることが重要である。そのために、オブジェクト指向言語であるＪａｖａとデータ格納場所として利用されるリレーショナルデータベースの間に存在する構造のバリアとデータ表現のバリアを解消したオブジェクト－リレーショナルマッピングを実現した。また、アクセス並行性を考慮して専有ロック時間を最小化した「２度読み方式によるトランザクション処理」を構築した。

（2）Ｖｉｅｗ（ユーザーインターフェース）層のバリアフリー

Ｖｉｅｗ層においては、真に使いやすいユーザーインターフェースを追求し操作性向上を図るため、HTMLタイプに加えアプレットタイプにも対応した。また、アクセス制御及び入力データ一括チェックは、XMLに関連情報を記述するだけで可能とした。

（3）Ｃｏｎｔｒｏｌｌｅｒ（制御）層のバリアフリー

Ｃｏｎｔｒｏｌｌｅｒ層においては、独立性を保ちつつ、スムーズにＭｏｄｅｌ とＶｉｅｗの制御を行えることを目指した。そのために、クライアントからの要求による制御を集中的に処理・管理できる仕組みを作成した。また、各層を双方向につなぐ「掛け橋」であるデータコンテナによるシームレスなデータ交換を実現した。更にエージェントを利用した非同期処理等により、回線接続時間及び応答時間を短縮させた。

バリアフリーＪａｖａフレームワーク利用により開発した当行のＣＲＭシステムは、２００２年１１月に本番稼動した。本システム開発では、Ｊａｖａ開発の初級者が７０％を占めたにも関わらず、従来の約３倍の生産性を実現できた。品質についても高レベル且つ均質で、高い利便性を実現できたことにより、当初設定したねらいはほぼ達成できたと総括している。

本フレームワークの構築により、Ｊａｖａの初級者であっても多様なユーザーニーズに対して高生産性、高品質で対応できる基盤が完成した。この資産を拡大発展させ、今後ますます多様化・複雑化していくであろうシステム開発ニーズにスピーディに応えていきたい。

目　次

１．はじめに

Ｊａｖａ言語を使ったＷｅｂアプリケーションは、大量クライアントへの展開容易性、保守性等の運用上のメリットから急速に広がっている。しかし、実際の開発においては、オブジェクト指向言語の特性である再利用性、拡張性、及び柔軟性が発揮できていなかったり、開発経験者の不足や必要とされる技術レベルの高さから開発効率が上がらない等の問題点を抱えているケースが少なくない。

折しも、当行はＪａｖａを使用したＷｅｂアプリケーション形態により「ＣＲＭシステム」を構築するプロジェクトを立ち上げた。２年前にＪａｖａを使用したＷｅｂアプリケーションを構築した経験はあったが、当行においても上記の問題点に直面している状態であった。これらの問題点を解決する方策として、キーポイントとなる「フレームワーク（共通の枠組み、共通部品群）」をＣＲＭシステム開発プロジェクトの中で構築した。

本論文のテーマである「バリアフリーＪａｖａフレームワーク」のねらいは、バリアフリーの一般概念である「日常生活に存在しているさまざまな障壁を取り除き、障害を持つ方や高齢者にも暮らしやすい社会を作っていく」ことをＪａｖａ開発へ適用し、

「Ｊａｖａ言語を使った大規模システム開発の中で発生するさまざまな障壁を取り除き、Ｊａｖａ言語の初級者であっても高生産性・高品質のシステム開発が行える環境を提供する」ことである。

本論文では、Ｊａｖａを使った大規模開発において直面するさまざまなバリアを打破し、アプリケーション開発について ―***「難しい」を「簡単」に***― することを目指して取り組んだフレームワーク構築についての工夫及び考慮点を中心に論述する。

２．Ｊａｖａ開発におけるバリア

２．１ Ｊａｖａ開発の難しさ

Ｊａｖａ言語による開発が初級者にとって難しいとされる原因として以下の点が考えられる。

（１）初級者にも広範囲の知識が必要

- 大規模システム構築には、文法等の基本知識の他に広範囲にわたる専門的な技術知識が必要となる。
- 再利用性、拡張性、及び柔軟性を発揮させるためにはオブジェクト指向を理解する必要がある（Ｊａｖａで「コードを書ける開発者」は大勢いるが、「オブジェクト指向で開発できる技術者」は非常に少ない）。

（２）開発支援ツールや部品の不整備

- Ｗｅｂアプリケーションサーバー製品は、ベース部分の機能提供に留まっており、大規模開発を行う上では不足している機能が多い。
- 比較的開発が容易と言われるＶｉｓｕａｌＢａｓｉｃ（以降ＶＢと略）と比較して、開発用部品が不足している。また、ユーザーインターフェースを開発する上で有効であるビジュア

ル開発ツールの使い勝手が劣る。

（３）技術者不足

・Ｊａｖａ言語の技術や開発経験を持つ技術者が不足している。

・スキルやノウハウの蓄積が少なく、継承されない。

２．２ MVC３階層モデルにおけるバリア

大規模システムは一般的に「ＭＶＣ３階層モデル」に従って構築することが推奨されている。ＭＶＣモデルは、業務処理を行う「Ｍｏｄｅｌ」、画面表示処理を行う「Ｖｉｅｗ」、業務処理と画面表示処理の間の制御処理を行う「Ｃｏｎｔｒｏｌｌｅｒ」の３つの層に分割してシステムを構築する。このモデルは各層の機能が明確に分離されているため、独立性が高く開発者間で作業分担して開発し易い。また、拡張性にも優れている。

しかし、ＭＶＣモデルによりＪａｖａ大規模システム開発をするには、図１のようなバリアが立ち塞がっており、生産性・品質向上を実現するためには、これらのバリアを打破する必要がある。

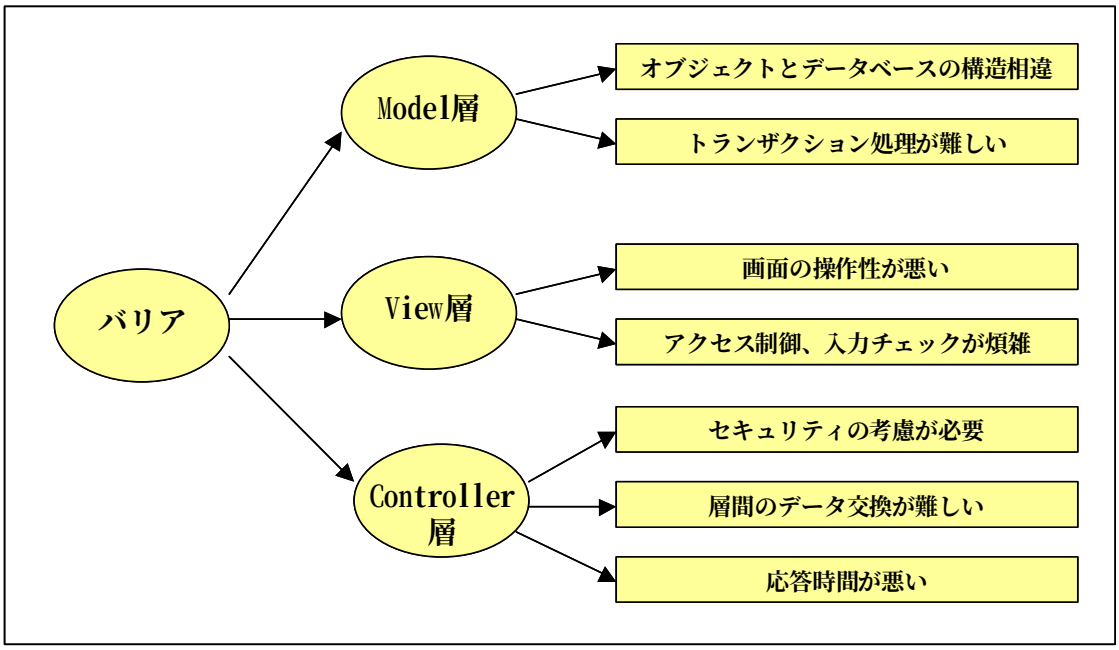

図１　ＭＶＣモデルにおけるバリアの樹形図

３．バリアフリーＪａｖａフレームワークのねらいと概要

３．１ 構築のねらい

Ｊａｖａ開発の難しさとＭＶＣモデルにおけるバリアを解決するには、すべての開発者が共通に効率よく作業できるソフトウェア環境である「フレームワーク」が必要となる。フレームワークは、モジュール群の組合せ、再利用、あるいは差分の追加プログラミングによってアプリケーションを効率的に開発できることを目指すものである。

ＣＲＭシステム開発にあたり、以下に示す４項目をねらいとするフレームワークの導入を決定した。

（１）生産性、品質向上

・ビジネスロジックに専念できる開発を実現（記述コード量を８０%削減）。

・開発負荷の削減と開発期間の短縮化（生産性を従来の３倍に向上）。

（２）利便性向上

・ＶＢで作成した画面と遜色ない高操作性を提供。

・大量ユーザー（クライアントパソコン２８００台）、WAN環境（１２８ｋｂｐｓ）であっても高パフォーマンスを実現（原則８秒以内のレスポンス）。

・アクセス並行性を兼ね備えた信頼性が高いトランザクション処理を実現。

（３）システム投資削減

・高スペックのサーバーシステム資源がなくても稼働可能にする。

・低スペックパソコンでも稼働可能にする。

・Ｗｅｂアプリケーションサーバー、フレームワーク等のソフトウェア投資を最小化する。

（４）スキル軽減、保守性向上

・複雑な処理の隠蔽により、必要となるスキルレベルを軽減し教育コストを削減する。

・可読性の向上、開発者による品質差異を縮減することにより変更容易性を向上させる。

（注）括弧内は当行のＣＲＭシステムでの要件・目標値

しかし市販フレームワーク製品を調査したところ、すべての要件を満たせるものは存在せず、必ずしも根本的な問題解決にはならない場合が多いことがわかった。主な問題点を以下に示す。

生産性・品質	カバーされている範囲が小さい（ＭＶＣの全層をカバーできていない）。 データのやり取りがシームレスにできない。 ソースコードが未公開であるため、デバッグしにくい。
利便性	画面作成用部品が貧弱（ＨＴＭＬのみが対象で、アプレットには未対応）。 ＤＢアクセス処理において、短時間での排他制御が考慮できていない。 パフォーマンスが良くない（パフォーマンスチューニングが難しい）。
システム投資	稼働条件が高く、高価なシステム資源を必要とする。
スキル・保守性	熟練スキルが必要な部分がある。

そこで、上記の４項目のねらいを満たす「バリアフリーＪａｖａフレームワーク」を自ら開発することにした。

３．２ フレームワークの概要

当行が開発した「バリアフリーＪａｖａフレームワーク」の特徴を以下に記す。

機能	（１）Ｍｏｄｅｌ層 ・容易なオブジェクト－リレーショナルマッピングを実現。 ・ＤＢ接続時間、ＤＢロック時間の短縮化。 （２）Ｖｉｅｗ層 ・ＨＴＭＬに加えアプレットタイプもサポート。 ・アクセス制御、入力データ一括チェックの実現。 （３）Ｃｏｎｔｒｏｌｌｅｒ層 ・シームレスなデータ交換の実現。 ・エージェントを利用した非同期処理の実現（時間がかかる処理を考慮）。 ・通信量削減策の実施（ＷＡＮ環境での利用を考慮）。 （４）共通 ・フレームワークの対象としてＭＶＣの全層をカバー。 ・エンタープライズアプリケーションに必要なユーティリティの提供。 ・今後のバージョンアップを考慮した実装クラスの切り替え機能。
開発スタイル	・ビジネスロジックに注力した開発の実現。 ・継承を使った差分プログラミング。 ・ＸＭＬによる情報定義。 ・効率良いデバッグの実現。

図２にバリアフリーＪａｖａフレームワークの概要構成を示す。

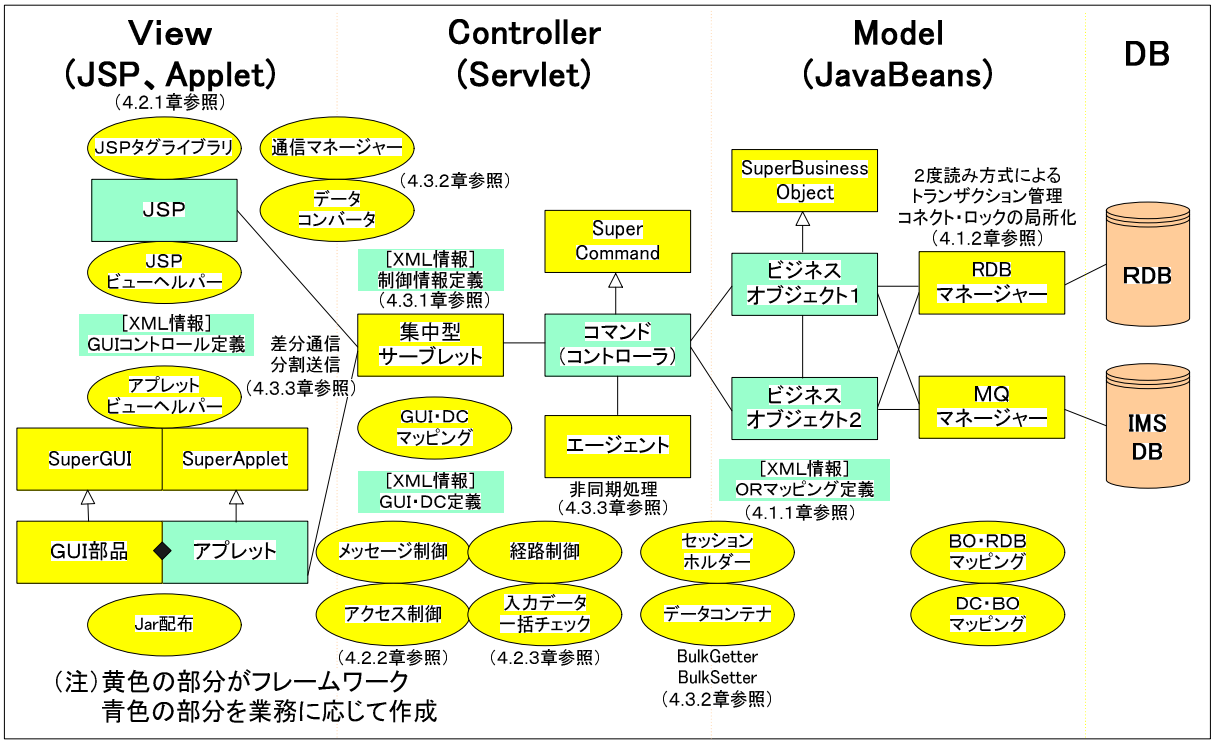

図２　フレームワーク概要構成

４.「難しい」を「簡単」に　－バリアフリー構築の工夫と考慮点－

Ｊａｖａアプリケーション開発について、「難しい」を「簡単」にできる「バリアフリーＪａｖａフレームワーク」を構築する上での工夫及び考慮点を、ＭＶＣ各層に分けて説明する。

４．１ Ｍｏｄｅｌ（ビジネスロジック）層のバリアフリー

Ｍｏｄｅｌ層は、開発者にビジネスロジックの開発に専念可能な環境を提供することがテーマである。その観点から現状の問題点の整理と対策について以下に記述する。

４．１．１ オブジェクト－リレーショナルマッピングの開発

（1）オブジェクトとデータベースのミスマッチの問題

問題を説明するために、当行ＣＲＭシステムにおけるダイレクトマーケティングのキャンペーン企画変更画面を例にする。本画面とＳＱＬの関連を図３に、対応するクラス図とＥＲ図の関連を図４に示す。

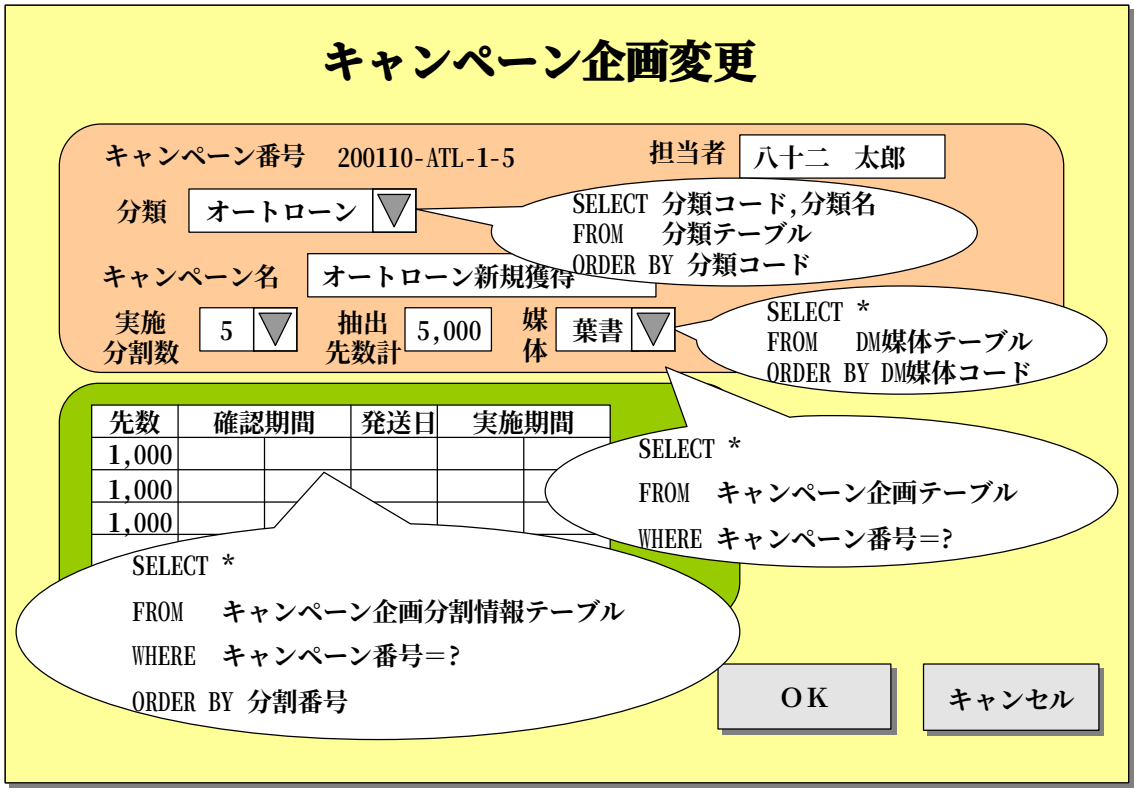

図３　キャンペーン企画変更画面とＳＱＬの関連

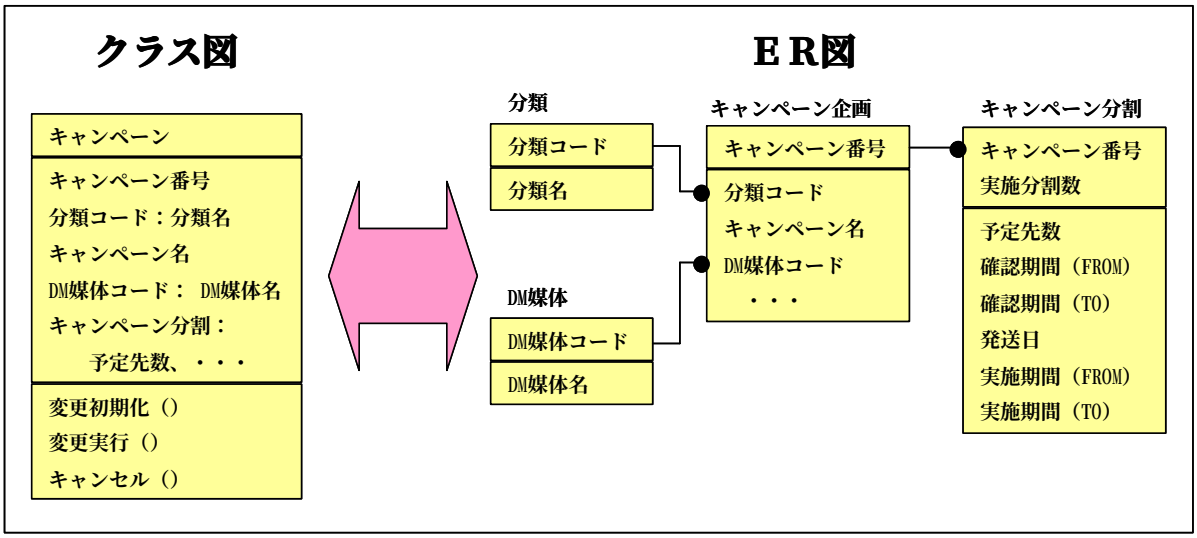

図４　キャンペーン企画のクラス図とＥＲ図の関連

ビジネスクラスはデータモデルのエンティティにメソッドが付随したものと考えられるが、正規化されたＥＲ単位のビジネスクラスでは粒度が細かすぎて、処理が煩雑になる。そこで、図４のクラス図のように業務処理の効率を考慮し、複数のエンティティをまとめて粒度が大きいビジネスクラスを作成することが多い。本例でもわかるように、１つのビジネスクラス（キャンペーン）が、マスターテーブル（キャンペーン企画）、明細テーブル（キャンペーン分割）、及びコードテーブル（分類、ＤＭ媒体）と関連を持つ。

この場合、クラス設計者には物理的なＥＲの構造に関する情報や知識を要求されることになる。この問題から設計者を開放し、ビジネスロジックに専念できる環境を提供することをねらいとして開発したのが、オブジェクト－リレーショナルマッピング（以下、ＯＲマッピング）である。

Ｊ２ＥＥでは、一般的にＯＲマッピングをＥＪＢにより実装することが推奨されている。１オブジェクトがＤＢテーブル１レコードに対応するような単純な関係のものであれば、プログラミングせずに、配備記述ファイルに対応を記述するだけでＤＢ処理をＥＪＢ実行環境に任せることができる。しかし、複雑なＯＲマッピングの不適合や技術スキルの高さ及びパフォーマンスの問題により、あまり利用が進んでいないのが実状であるため、今回はＪＤＢＣを使ってＪａｖａＢｅａｎｓとして開発した。

（２）ＯＲマッピングの機能

オブジェクト指向言語であるＪａｖａとデータ格納場所として利用されるＤＢの間には以下のバリアが存在する（バリアの概要については図５に示す）。

・構造のバリア

クラス、継承、委譲といったオブジェクト指向特有の複雑な構造を表現できるＪａｖａに対し、ＤＢは２次元の表と表間の関連とでデータを表現する。

・データ表現のバリア

多彩なデータ型を持つＪａｖａに対し、ＤＢのデータ型は原則、数値型・文字型・日付型といった基本データ型に限られる。反面、Ｊａｖａはデータ型の桁数を厳密に定義できないが、ＤＢは厳密に定義できる。

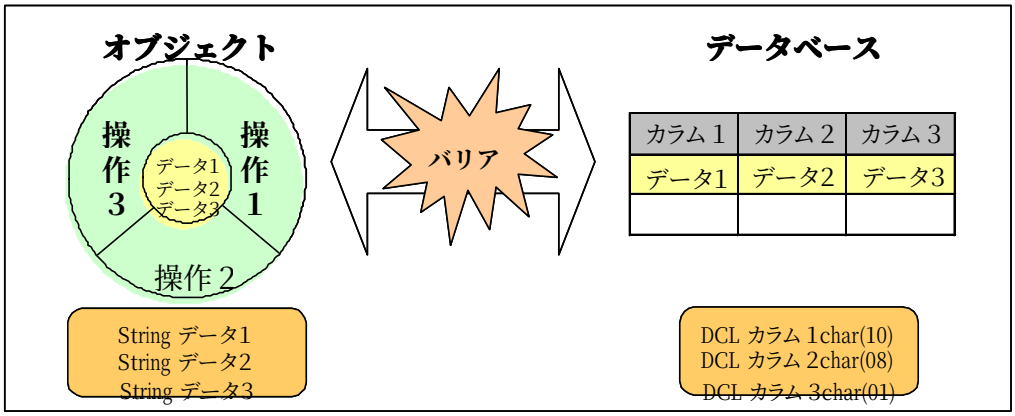

図５　オブジェクトとデータベース間のバリア

これらのバリアを解消しＯＲマッピングを実現するためには、オブジェクト側から見る属性に対応するＤＢのテーブル・列名、更新の有無等の情報が必須となる。このＯＲマッピング情報をＸＭＬで記述することで、ソースコードを変更することなくマッピングを追加・変更可能にし保守性を向上させた。加えてＳＱＬのパフォーマンスチューニングも行い易くした。図４で取り上げたキャンペーン企画のＯＲマッピングＸＭＬ定義を以下に示す。

```
<Item IName=“キャンペーン企画" mode="RW">
  <Sql s=" SELECT  * FROM キャンペーンテーブル WHERE キャンペーン番号= '%1'" />
  <Map DCName="キャンペーン番号" DCType=“テキストボックス" Target="キャンペーン番号" />
  <Map DCName=“選択分類コード” DCType=“コンボボックス” Target=“分類コード” />
  <Map DCName=“選択DM媒体コード" DCType="コンボボックス" Target="媒体コード" />
     ・・・
</Item>
<Item IName=“分類" mode="RO">
  <Sql s=" SELECT 分類コード,分類名 FROM 分類テーブル ORDER BY 分類コード" />
  <Map DCName=“分類” DCType=“コンボボックス” Target=“分類コード,分類名" />
</Item>
<Item IName=“DM媒体" mode="RO">
  <Sql s=" SELECT DM媒体コード, DM媒体名 FROM DM媒体テーブル ORDER BY DM媒体コード" />
  <Map DCName=“DM媒体” DCType=“コンボボックス” Target=“DM媒体コード, DM媒体名" />
</Item>
<Item IName=“キャンペーン分割" mode="RW">
  <Sql s=“ SELECT  * FROM キャンペーン分割テーブル WHERE キャンペーン番号= ‘%1’ ORDER BY 分類番号" />
  <Map DCName=“キャンペーン分割” DCType=“テーブル” Target=“先数,確認期間,発送日,実施期間" />
</Item>
```

ＯＲマッピング機能はビジネスクラスのスーパークラスに実装した。主なメソッドの機能概要は以下の通りである。

生成メソッド	ＯＲマッピングＸＭＬ情報を読み込む。 対象の全ＳＱＬを実行する。 ＳＱＬの結果セットをデータコンテナ（データの入れ物、詳細は４．３．２参照）に装填する。
データ取得メソッド	データコンテナ内の対象データを取得する。
データ設定メソッド	データコンテナ内の対象データに値を設定する。
更新メソッド	設定されたデータコンテナ内容に基づいて、更新対象の全テーブルを更新する（ＸＭＬ定義のｍｏｄｅがＲＷのＳＱＬが対象）。

この機能を使用すれば、ＳＱＬの照会結果を取得して画面に表示させる、あるいは画面の入力値をＤＢへ反映させるといった処理を簡単に実現できる。各ビジネスクラスはこのスーパークラスを継承し独自ロジック部分のみを差分プログラミングすることにより、ビジネスロジックに注力した開発が可能である。ビジネスクラスの実現イメージを図６に示す。

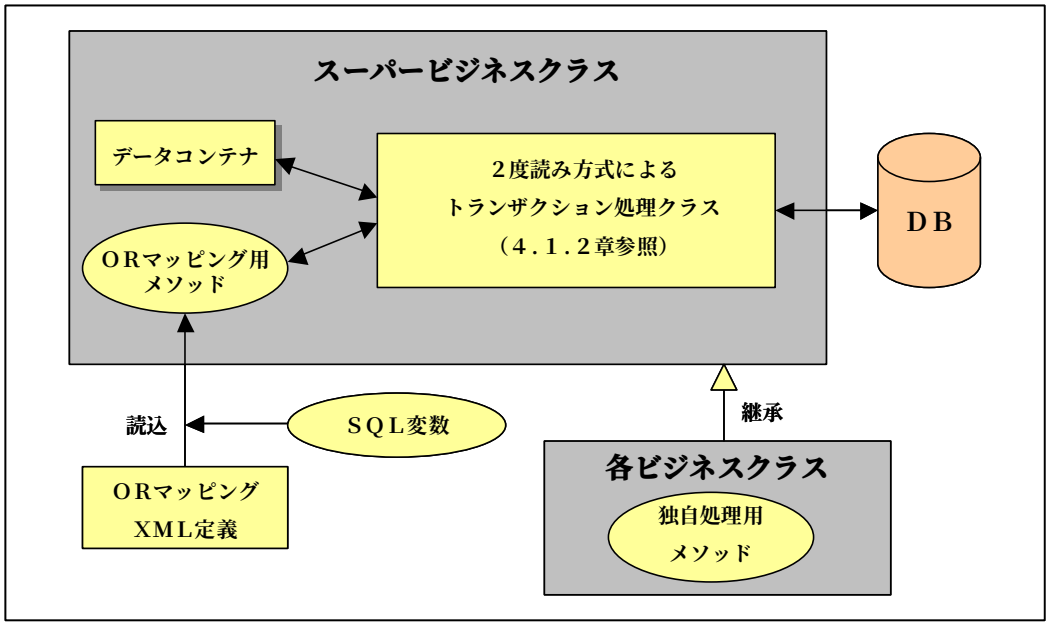

図６　ビジネスクラスの実現イメージ

４．１．２ データベースのロック制御

ＤＢの更新を伴うトランザクション処理において、ＤＢの整合性を保つためにはロックを掛ける必要がある。この場合、ユーザーのアクセス並行性を考慮し処理効率を上げるためには、データの最初の取得からＤＢ更新までのロックでは冗長過ぎ、ＤＢ更新時だけでは他の処理要求と競合した場合に整合性が破壊される可能性がある。

この課題を解決するために「２度読み方式によるトランザクション処理」を採用した。本処理では、画面表示用の最初のデータ取得からユーザーが画面入力を行う間はロックせず、更新ボタン押下で更新要求が出たタイミングでロックし読み込んだデータと最初の取得データを比較する。その結果、内容に変更がなければ更新し、変更があれば他ユーザーの更新要求が割り込んだと見なして処理をエラーとする方式である。

図３で取り上げたキャンペーン企画処理を例とした処理フローを図７に示す。

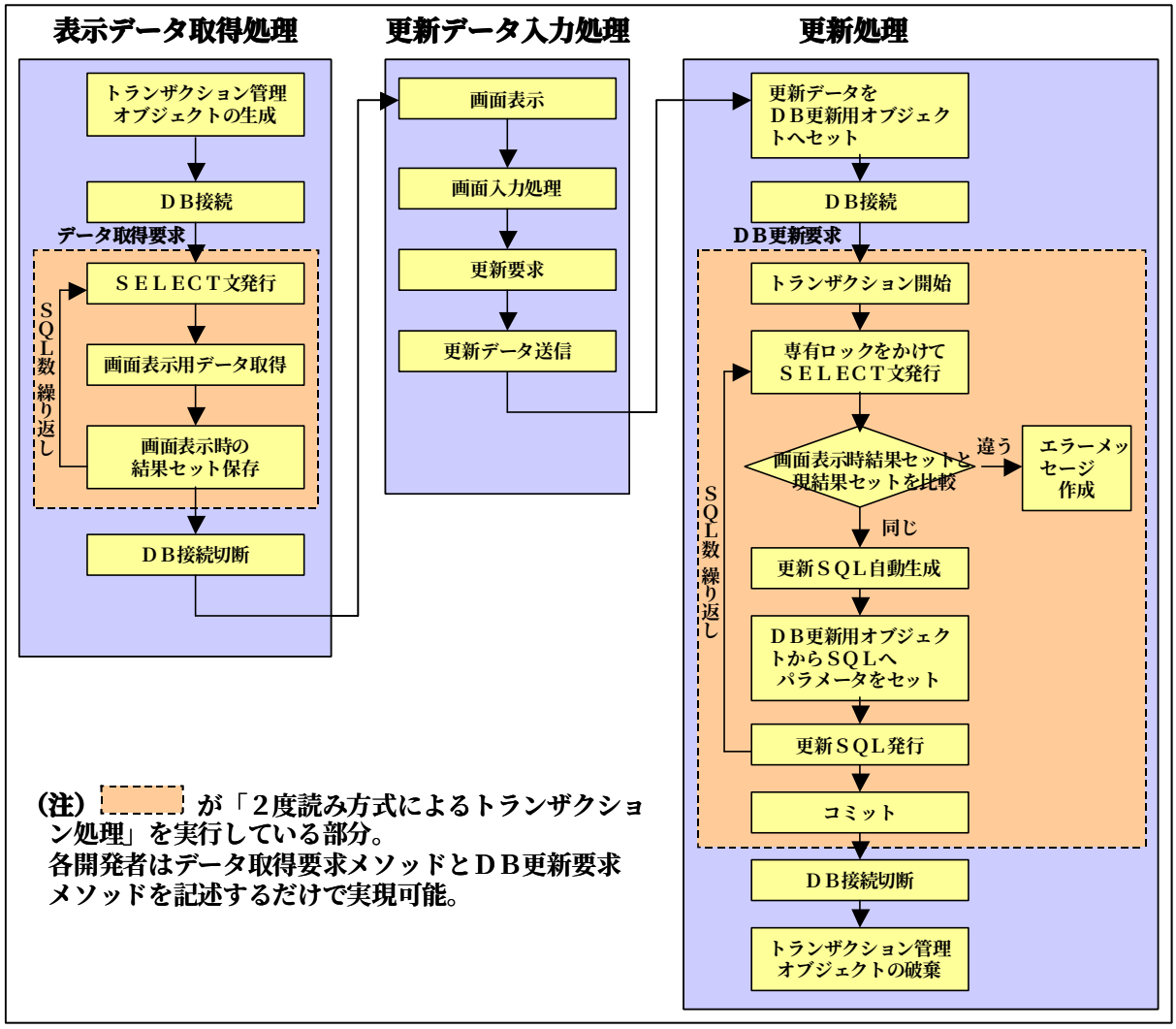

図７　２度読み方式によるトランザクション処理フロー

２度読み方式によるトランザクション処理を、Ｊａｖａで実装する上でのポイントと考慮点を以下に記す。

・ＤＢアクセスクラス（ＪＤＢＣのＡＰＩを利用)、リザルトセット管理クラス（１つのＤＢアクセスクラスに対応したＳＱＬ検索結果をハンドリング）及びトランザクション管理クラス（複数のＤＢアクセスクラスを束ねてトランザクションを実行）により構成。

・今後のＪＤＢＣのバージョンアップによる変更を考慮し、インターフェースと実装クラスを分離（実装クラスの切り替え可能）。
・２度読み時の整合性判定はレコード数、カラム数、及び全レコードの内容比較による。
・更新用ＳＱＬはカラム名、カラムの型、レコード特定キー及び更新データ値を元に自動作成（リザルトセットのメタデータを利用、Ｊａｖａのデータ型をＤＢのデータ型に変換）。
・複数テーブルにまたがる更新やＳＱＬバッチ処理のような複雑な処理にも対応。

４．２ Ｖｉｅｗ（ユーザーインターフェース）層のバリアフリー

Ｖｉｅｗ層は、他の層に依存せずユーザーインターフェースに徹した機能を提供することがテーマである。その観点から現状の問題点の整理と対策について以下に記述する。

４．２．１ HTML・アプレット両タイプへの対応

ユーザーへの高操作性の提供と開発者への作成容易性の提供を目的として、Ｖｉｅｗ開発支援機能を作成した。

（1）高操作性の提供

高操作性を提供するためにはユーザーのキー操作及びマウス操作をトリガーにしたイベントドリブン処理が必要である。

しかしＨＴＭＬベースの画面はＧＵＩ部品と機能がともに貧弱なものに限定されてしまうため、じゅうぶんなイベントドリブン処理ができない。そこで、ＨＴＭＬのみでは実現できない、真に使いやすいユーザーインターフェースを追求してアプレットにも対応した。

（2）作成容易性の提供

Ｊａｖａ言語の知識を持たないデザイナーでも容易に画面を作成できるように、ビジュアルツールにより開発可能にした。

ＨＴＭＬタイプの開発は、標準ＧＵＩ部品でビジュアルに開発した後、ＪＳＰタグライブラリを補足追加するスタイルにした。また、アプレットタイプ開発は、事前登録した多機能ＧＵＩ部品を使用しビジュアル開発可能とした。

（3）開発支援機能

各タイプの主な開発支援機能を以下に示す。

（a）HTMLタイプ

ＪＳＰタグライブラリの例	
・Ｈｅａｄｅｒタグ	文字コード設定や不正アクセス防止等、表示開始時に必要な処理を実行。
・ＣＳＳタグ	画面解像度を判定し該当スタイルシートを適用。
・Ｓｃｒｉｐｔタグ	ボタンの２度押しによる２重送信防止等、共通的に必要な機能を実行（ＪａｖａＳｃｒｉｐｔとして出力）。
・Ｌｏｏｐタグ	繰り返し処理をサポート。

ＪＳＰビューヘルパー
・画面表示用データを一元的に取得可能。 ・表示用に文字型へ型変換。

（b）アプレットタイプ

スーパークラス	
・アプレットに共通する機能を実装。 ・各アプレットはこのスーパークラスを継承して作成。	
ＧＵＩ部品の例	
・数値型テキストボックス	数値入力制限、カンマ表示、最小最大値指定機能。
・リストボックス	複数項目選択、項目位置揃え、罫線表示機能。
・グリッド	入力制限、表計算関数指定、レンダラ設定、セルのマージ、スクロール固定、複数選択機能。
・データ連結	ＧＵＩ部品とデータ保有オブジェクトを動的に連結。
・レイアウト管理	画面解像度を判定し表示サイズを調整。
Ｅｘｃｅｌ連携	
・Ｅｘｃｅｌと連携してシミュレーション、グラフといった高度なＯＡ機能を提供。	

４．２．２ アクセス制御

ログインしたユーザーの利用権限に応じて、画面内容や実行可能な機能を変更する等のアクセス制御が必要になるケースは多い。しかし、アクセス制御を行うにあたって、ユーザーが所属する部店番コード等の特定の属性情報を判断ロジックに組み込んでしまうと汎用性が落ちてしまう。そこで、ユーザーの所属グループが保有しているロールを元に、ボタンの表示／非表示、入力（選択）の可否等のアクセス制御を行う仕組みを作成した。

この仕組みにより、プログラミングせずに、ＸＭＬに関連情報を記述するのみで詳細なアクセス制御が可能になった。図８にロールによるアクセス制御の例を示す。

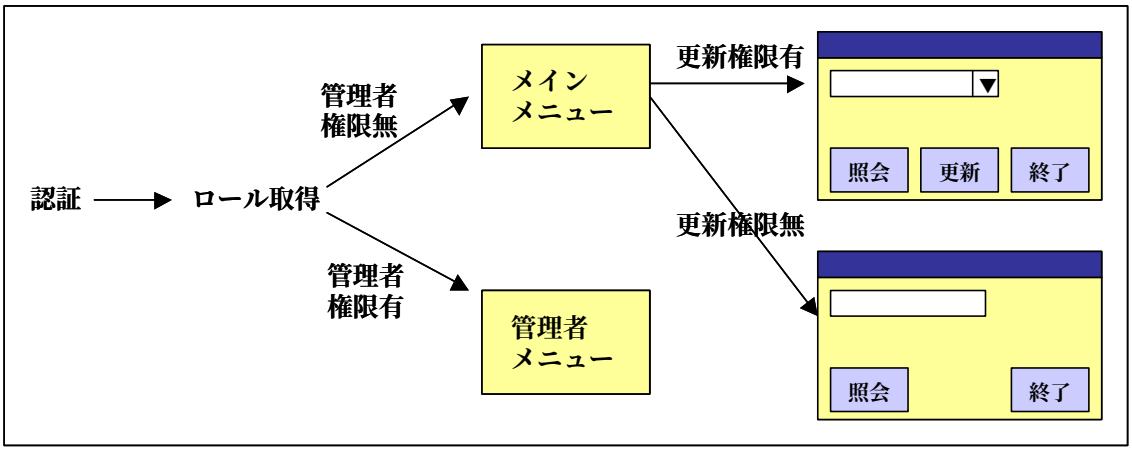

図８　ロールによるアクセス制御の例

４．２．３ 入力データ一括チェック

入力データチェックは更新処理が必要なシステムにおいて不可欠な機能である。入力チェックの実行はクライアント側とサーバー側のいずれか、あるいは両方で行う方法が考えられるが、サーバー側の入力チェックは必須であり、クライアント側は補足的に使用すべきである。その理由は、ＨＴＭＬで使用するＪａｖａＳｃｒｉｐｔ等のスクリプト言語はＷｅｂブラウザの設定変更で無効にすることが可能なため、クライアント側のみの入力チェックでは好ましくないからである。

入力チェックを個別に記述してしまうと、同種条件文のロジックが各プログラムに重複してしまい、再利用性が失われてしまう。また、入力項目が多い場合、プログラミング及びテストの負荷が大きくなる。

そこで入力チェックで多数を占める基本的チェックをサーバー側で一括実行する仕組みを作成した。これは各画面に対応した入力チェック情報をＸＭＬに記述することで、入力項目が必須項目か、データ型は何か、桁数はいくつか等のチェックを実行するものである。この仕組みにより、個別プログラムでは必要に応じて多項目間の関連チェックを記述するのみに留められる。図９に入力データの一括チェックのイメージを示す。

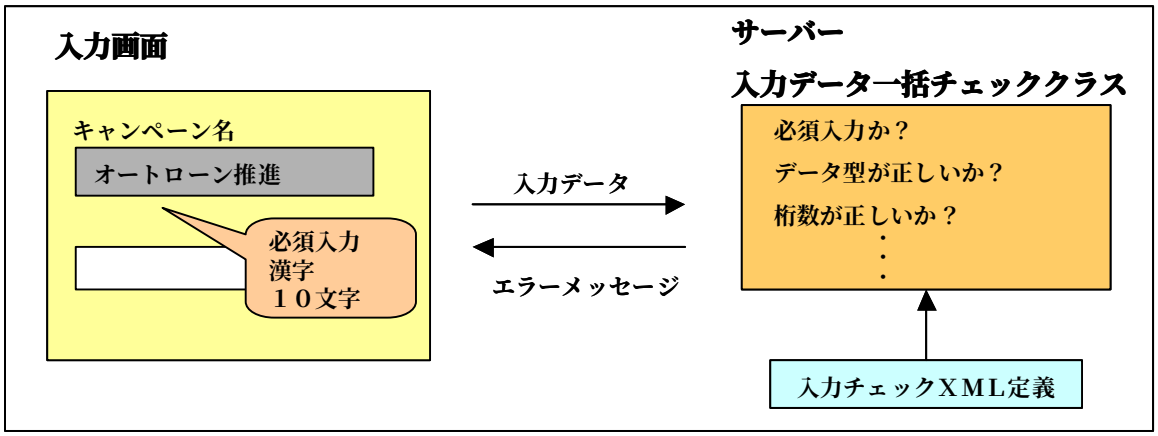

図９　入力データ一括チェックのイメージ

４．３ Ｃｏｎｔｒｏｌｌｅｒ（制御）層のバリアフリー

Ｃｏｎｔｒｏｌｌｅｒ層は、開発者にＭｏｄｅｌ とＶｉｅｗのスムーズな制御機能を提供することがテーマである。その観点から現状の問題点の整理と対策について以下に記述する。

４．３．１ 制御の一元化

クライアントからの要求に応じて、セキュリティサービス（ユーザー認証、アクセス権限付与、不正アクセス防止等）、ビジネスロジックの呼び出し、画面の送信、接続制限、エラーハンドリング等の制御機能をサーバー側で実現する必要がある。これらの機能の重複を排除し、一元的に処理・管理できるサーブレットを作成した。

Ｖｉｅｗからのコマンド（処理命令）に応じて、ＸＭＬに定義した制御情報を元に処理を選択・

実行できる仕組みを作成することで、クライアントからの個別の要求に汎用的に応えられるようにした。制御情報の定義例を以下に示す。

```
<CommandDef>
 <!-- 前処理（本処理に入る前に必要な処理） -->
 <!--例：認証有無・セッション情報保持に係るチェック、アクセス量把握のためのページカウント -->
  <Command id="PreCommand"   class="CmdPre"     jsp="errorPage.jsp" />
 <!-- 後処理（本処理に後に必要な処理） -->
  <Command id="AfterCommand" class="CmdAfter" />
 <!-- 本処理 -->
  <Command id="Login"        class="CmdLogin"   jsp="Login.jsp" />
  <Command id="CpTitle"      class="CmdCpTitle" jsp="CpTitle.jsp" framename="FrmCpTitle" />
    ・・・
</CommandDef>
```

また、システムの構成情報（ＵＲＬ、ＨＴＴＰとＨＴＴＰＳの切り替え、ＤＢ設定情報等）はプロパティファイルに記述することで設定・変更作業を容易にした。

４．３．２ シームレスなデータ交換の実現

（1）Ｖｉｅｗとの通信

アプレットに対応するには、ＨＴＭＬとのアクセスの違いをサーバーサイドで吸収することが必要となる。そのために、「データ保有オブジェクト」と「通信データ形式」をアプレットに合わせて開発した。データ格納オブジェクトは再帰的なデータ構造を持つデータコンテナクラスとして提供した。また、データコンテナをＸＭＬデータに変換して通信可能にする通信マネージャークラスを作成した。加えてＨＴＭＬからの送信データをアプレットと同じ形（データコンテナ）に変換するデータコンバータクラスも作成した。これらの仕組みにより、画面タイプに依存しないＭｏｄｅｌ層の開発を実現できた。図１０にクライアントとの通信イメージを示す。

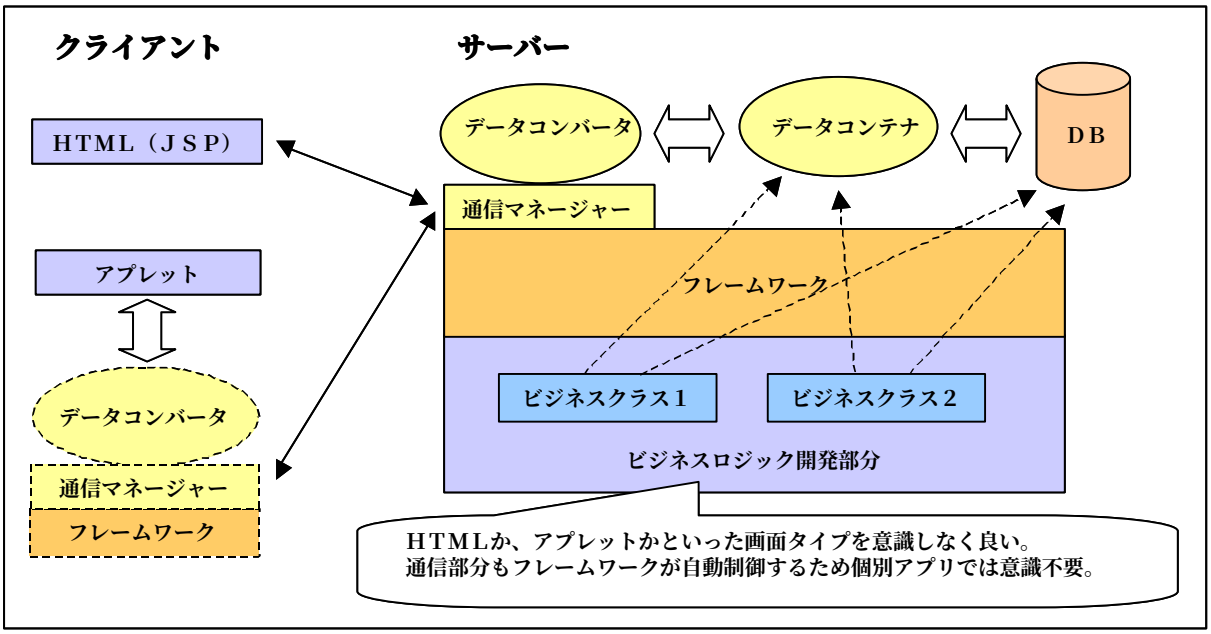

図１０　クライアントとの通信イメージ

（2）データ保有オブジェクトへの共通アクセス

オブジェクト指向ではカプセル化の観点からクラスが保有する各データ項目属性に対してｇｅｔｔｅｒとｓｅｔｔｅｒを１つずつ作成し、このメソッドを介してアクセスさせるようにするのが通例である。 しかし、データ項目数が多い場合には、属性定義と単純なｇｅｔｔｅｒ、ｓｅｔｔｅｒだけで相当数のプログラミングが必要になる。また、関連する複数のビジネスクラス名と各属性の取得メソッド名をＶｉｅｗ層に記述する必要があるため、データ項目の追加・変更・削除に大きく依存してしまい独立性が損なわれてしまう。

これらの問題を解決し、データ交換を容易にするためにデータコンテナを作成した。データコンテナとは、「あらゆるデータの入れ物」であり、グルーピングを行えるようにフォルダとファイルの関係のような再帰的な構造とした。また、一元的な取り扱いを可能するために各データは項目名と値との組合せで構成した集合オブジェクトに格納した。これにより個々の属性、ｇｅｔｔｅｒ及びｓｅｔｔｅｒ定義は不要になり、データ項目がいくつあっても２つの汎用的なｇｅｔｔｅｒ、ｓｅｔｔｅｒメソッドでアクセス可能となった。

更に、データへの個別アクセスに加え、バルクゲット（一括取得）、バルクセット（一括設定）によるアクセスを作成した。ＳＱＬ照会結果を一括取得して画面に表示させる処理はバルクゲットを使用し、画面に入力された値を一括でＤＢへ反映させる処理はバルクセットを使用することにより簡単に実現できる。

現在は画面定義用、ＧＵＩ部品用、ＤＢ用、ＡＣＬ用、及びメッセージ用として５種類のデータコンテナを定義している。図１１にデータコンテナのクラス図を示す。

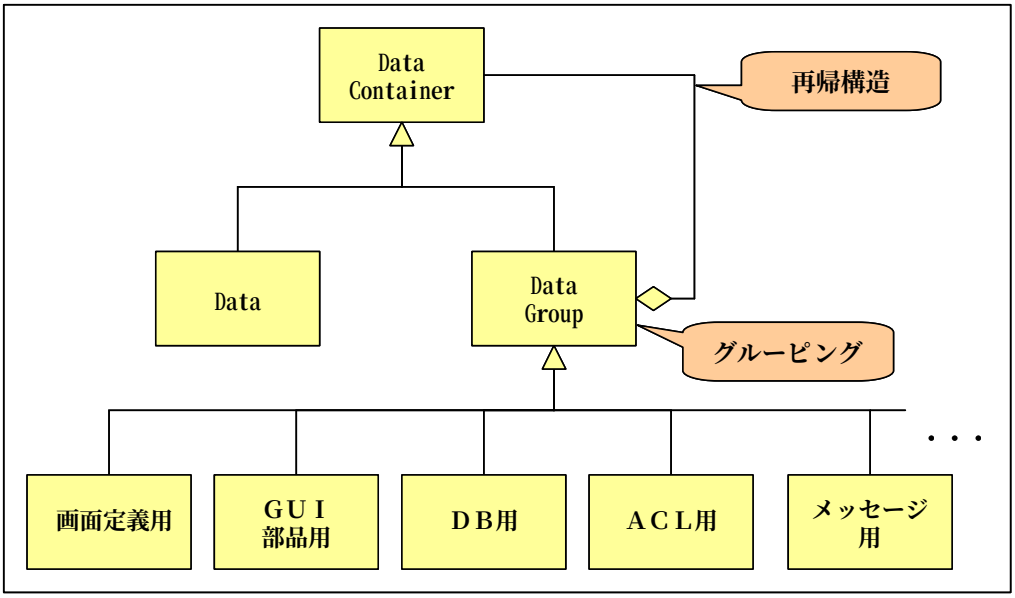

図１１　データコンテナのクラス図

４．３．３ 応答時間の短縮

市販フレームワークの中には、多機能であるがパフォーマンスが良くないものもある。しかし、機能とともにパフォーマンスは非常に重要な要件である。応答時間を短縮させるために実施した対策を以下に示す。

（1）ベンチマークテストの実施

ベンチマークテスト用にアクセスパターン別のプロトタイプを作成し、テストツールを用いて負荷テストを実施した。負荷テストでは、サーバー同時アクセス数の設定を１、２０、５０、１００と変更して各機能別レスポンスタイムを測定することでボトルネックの箇所を特定し、チューニングを行った。パフォーマンスチューニングに係るポイントを以下に示す。

- ・XML情報読込やＪＤＢＣデータソース取得等、１回のみ実行すればよい処理はシングルトンにして起動時に実行する。
- ・ＨＴＴＰセッションに保存するオブジェクトサイズを小さくする。また、消し忘れ防止策として、３画面以上経過したオブジェクトを自動削除する機能を作成する。
- ・ＪＤＢＣ接続の獲得やクローズ処理のオーバーヘッドを避けるためにコネクションプーリング機能を使用する。
- ・ＤＢロック時間を最小化するために、適切なＩｓｏｌａｔｉｏｎレベルを設定する。
- ・外部設定した間隔でガーベッジコレクションを実行させる仕組みを作成する。
- ・ログ出力処理のディスクアクセス逐次化によるレスポンス悪化防止策として、デバッグ時及び本番運行時にログ出力レベルと出力クラスを切り替える。
- ・文字の連結処理にＳｔｒｉｎｇクラスを使うと大量の一時クラスを消費するためＳｔｒｉｎｇＢｕｆｆｅｒクラスを使用する。

（2）エージェントを利用した非同期処理

当行のＣＲＭシステムの応答速度には、インターネットの世界で言われている「８秒ルール」を適用した。しかし、全件の顧客情報のような大量データが対象となる処理や複雑な計算が繰り返し必要となる処理においては８秒ルールを守れない。また、接続状態のまま長時間ユーザーを待たせるのは不都合である。場合によっては、応答時間を短縮させるために高スペックのサーバー導入が必要になる可能性もある。

この問題の解決策として「エージェントを利用した非同期処理」の機能を取り入れた。本機能は、株式会社インテックの製品である「エージェントシステム」をベースに開発した。これは時間がかかる処理をエージェント（人が行う情報処理を支援または肩代わりしてくれるソフトウェアプロセス）に依頼し、後に処理結果を取得できる機能である。実際の処理時間に関わらず短時間で応答を返すことで、接続状態でのクライアントの処理待ちを解消することができる。図１２にエージェントを利用した非同期処理の概要を示す。

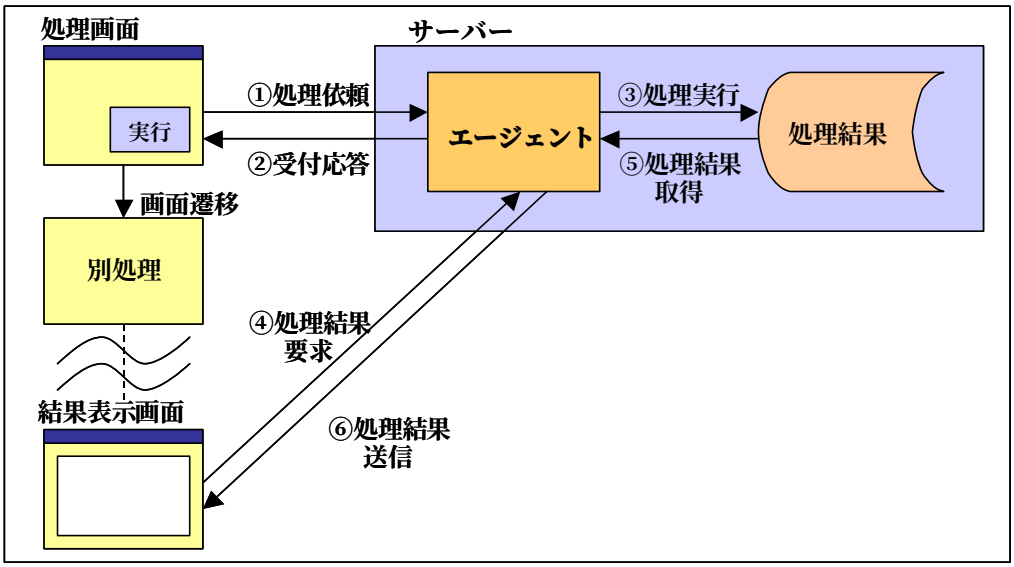

図１２　エージェントを利用した非同期処理

（3）通信量削減

昨今、ネットワークの進化により急速に通信速度が上がっているが、ＷＡＮ環境の末端回線速度は６４ｋｂｐｓあるいは１２８ｋｂｐｓの場合が多い。当行でも、ＷＡＮ環境の末端の回線速度は１２８ｋｂｐｓである。そこで、低速ネットワーク環境においても応答速度を劣化させないように、以下の通信量削減策を実施した。

（a）差分通信

クライアントとの通信において、全データを毎回送信する必要はない。例えば、画面の一部を再表示したり、変更入力したりといった場合には変更があったデータのみを送信すればよい。

そこで、必要なデータのみを差分通信する仕組みを作成した。通信データはＳＯＡＰへの今後の

拡張性を考慮し、ＸＭＬに変換する仕様にした。図１３に差分通信のイメージを示す。

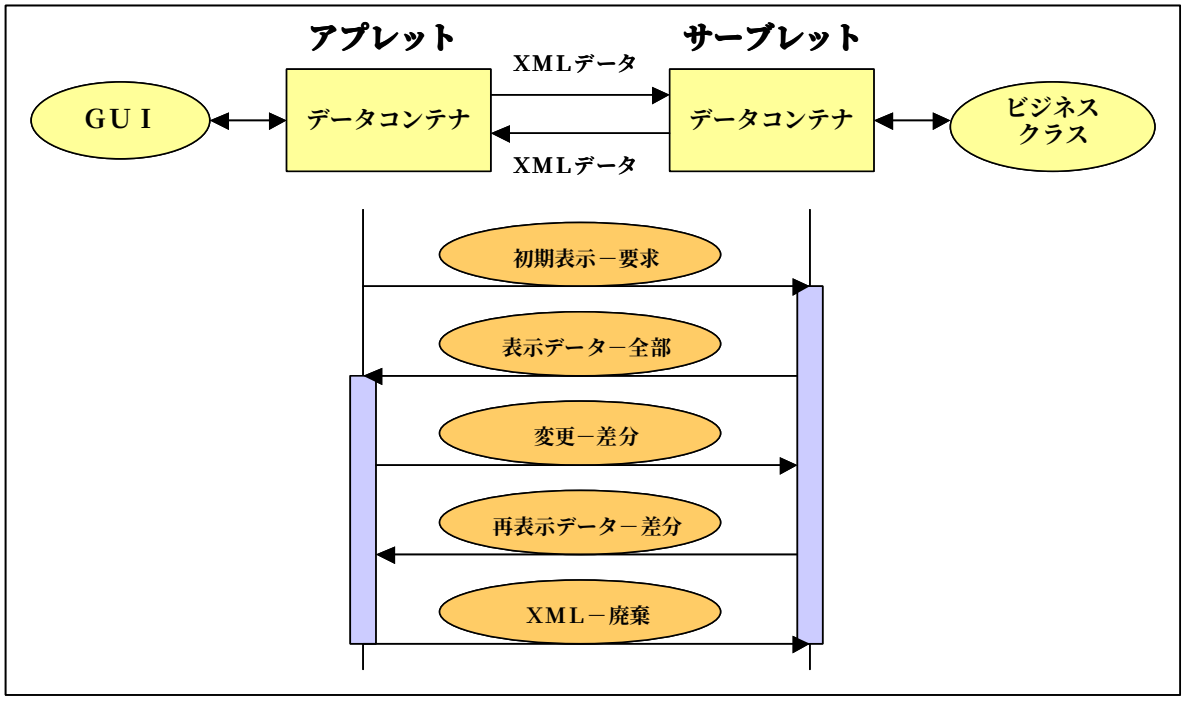

図１３　差分通信イメージ

（b）送信データ分割

明細一覧画面では、表示対象のデータ量が非常に多くなってしまう可能性がある。対象データが大量になってしまった場合、ネットワーク負荷が大きくなり応答時間が大幅に劣化してしまう。

そこで、分割送信の仕組みを作成することで本問題点を解決した。この仕組みは送信対象データを一旦サーバーにキャッシュし、ページ要求がある都度、指定数のデータをクライアントへ送信するものである。

（c）ダウンロード時間削減

画面にアプレットが含まれている場合、呼び出される都度、アプレットのダウンロードに時間がかかるという問題点がある。

そこで、自動配布可能というアプレットのメリットを生かしつつ、アプレットをクライアントパソコンのハードディスクに明示的にキャッシュする仕組みを構築することで本問題点を解決した。この仕組みは実行時にクライアントパソコンのアプレットのタイムスタンプをサーバーのものと比較してバージョンチェックを行い、バージョンが異なっている場合にのみダウンロードを促すものである。図１４に自動配布処理のイメージを示す。

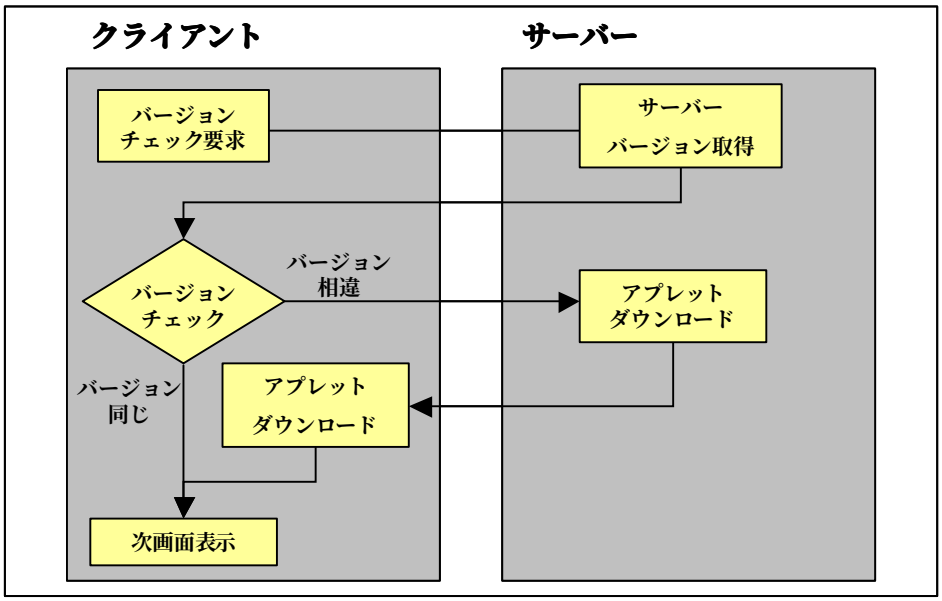

図１４　自動配布処理イメージ図

５．導入効果と今後の展望

５．１ 導入効果

これまで述べてきた「バリアフリーＪａｖａフレームワーク」利用により開発した当行のＣＲＭシステムは、２００２年１１月に本番稼動した。本システム開発について、他フレームワークと比較した場合の利点、及び当行で２年前に行ったＷｅｂシステム開発と比較した評価結果を以下に記す。

５．１．１ 他フレームワークと比較した利点

（1）Ｓｔｒｕｔｓとの比較

有名なＳｔｒｕｔｓフレームワークは、当行の「バリアフリーＪａｖａフレームワーク」と同様、ＭＶＣ３階層モデルに基づいた機能を有する。Ｓｔｒｕｓは軽量フレームワークであるため、機能がカバーされていない以下の部分については独自開発する必要がある。

- Ｍｏｄｅｌ層は、Ｃｏｎｔｒｏｌｌｅｒからのアダプターのみの提供のため、ＯＲマッピングのバリアやＤＢのロック制御の問題は解決されない。
- Ｖｉｅｗ層は、ＪＳＰ作成を支援するカスタムタグを提供しているが、アプレットは未対応。
- Ｃｏｎｔｒｏｌｌｅｒ層は、アプリケーションの流れを制御する機能を提供しているが、画面遷移の順序は保証されず、ブックマーク等による不正なＵＲＬ直接呼び出しが可能である。
- ＭｏｄｅｌとＶｉｅｗの設計段階での密接度が高く、且つＭｏｄｅｌ設計時にセッションやスコープを意識しなければならない。

一方、当行の「バリアフリーＪａｖａフレームワーク」はＭＶＣ全層の機能をカバーしており、純粋にビジネスロジックに特化してＭｏｄｅｌ設計を行える。また、ＤＢアクセスロジックの元と

なるＳＱＬが外部ファイルで管理されているため、ビジネスロジックをトランザクション制御ロジックとＤＢアクセスロジックとに分離して開発できる。そのため、開発、不具合修正、及び故障解析について負荷を大幅に削減できる。

（2）ＥＪＢを使用したフレームワークとの比較

ＥＪＢは移植性と拡張性を考慮しており将来的には優れた基盤になると考えられるが、現状では以下の問題点を抱えている。

・パフォーマンス面の問題

分散アプリケーションを前提としているためリモートメソッドコールを多用している。また、ＤＢアクセスをラッピングしているため最適化されたコード・ＳＱＬを記述できない。

・ＯＲマッピングの問題

１オブジェクトがＤＢテーブルの１レコードに対応するような単純なマッピングではない粒度の大きいマッピングが必要になる場合にはＪＤＢＣのコードを直接記述しなければならない。

・システム資源の問題

ＥＪＢ実行環境には多くのシステム資源が必要になる。そのため、高スペックのマシンと高額なＷｅｂアプリケーションサーバー製品の導入が必要になってしまう場合が多い。

・技術レベルの高さの問題

開発する上では厳格なアーキテクチャを理解していることが前提となるため、通常の部品開発よりも高いスキルが必要になる。また、デバッグの難易度も高い。

一方、当行の「バリアフリーＪａｖａフレームワーク」はオブジェクト指向を中心に構築されているが、機能分割を開発者に分かり易く示す工夫が盛り込まれており、移植性や拡張性に加え簡易性も実現している。

５．１．２ ねらいの達成度

（1）生産性、品質

ＣＲＭシステムの開発においては、Ｊａｖａ開発の初級者が全体の７０％を占めたにもかかわらず、従来の約３倍の生産性を達成した。開発工程別の「生産性の向上率」と「生産性向上の主たる理由」を以下に示す。

開発工程	生産性向上率	主たる理由
詳細設計	２．５倍	設計書に記述すべき部分の削減 パターン別に標準化
プログラミング	３．３倍	ビジネスロジックの開発に専念 部品化による再利用の促進により従来の２０％程度のプログラミング量で開発
テスト	３．１倍	今まで重複していた機能の検証負荷の軽減 デバッグ容易性の向上

（2）利便性

当行ＣＲＭシステムは、稼働後３ヶ月を経過したところであり、現段階での評価は困難である。しかし、利用したユーザーからは、既に操作性やパフォーマンスについて高い評価をいただいている。その要因は以下の点にあると考えている。

- 画面タイプとしてＨＴＭＬに加えアプレットにも対応してＶＢで作成した画面と遜色ない高操作性を実現できたこと。
- アクセス並行性を考慮した２度読み方式によるトランザクション処理を実現できたこと。
- フレームワークのチューニングに加え、差分通信や送信データ分割等の通信量削減策により大量ユーザー及びＷＡＮ環境下であっても高いパフォーマンスを確保できたこと。

（3）必要なシステム資源

時間がかかる処理でもエージェントを利用した非同期処理により、実際の処理時間に関わらず短時間で応答を返すことを実現し､必要となるサーバースペックを抑えることで投資額を軽減できた。ＳＱＬ集計処理でのデータ件数と回線接続時間の相関グラフを図１５に示す。

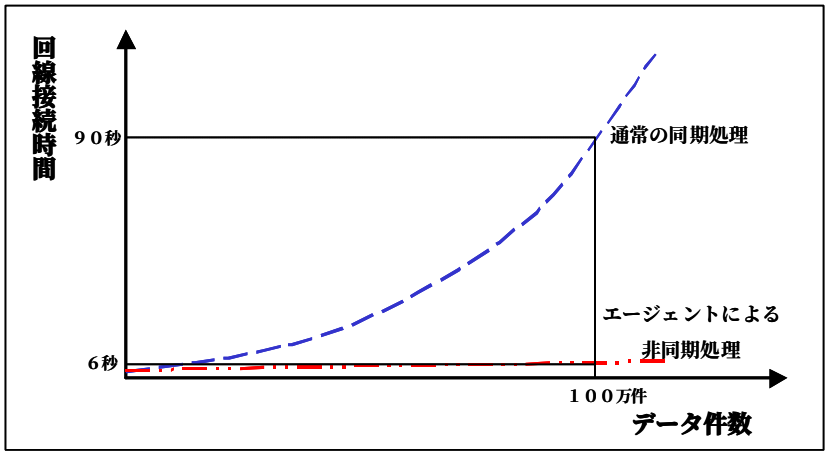

図１５　ＳＱＬ集計処理でのデータ件数と回線接続時間の相関グラフ

また、クライアント側の機能をユーザーインターフェース部分に特化させることで、低スペックパソコンでも動作可能にした。

（4）スキル軽減、保守性

複雑な処理を隠蔽した高水準レベルのインターフェース提供により、開発者に必要なスキルを軽減させた。ＭＶＣ各層の独立性を高めることで開発者のレベルに合わせた分業が可能になった。Ｊａｖａ言語の知識を持たないデザイナーでも関連情報のＸＭＬへの記述とビジュアルツールを使用した開発スタイルの実現により容易に画面を作成できる。また、開発者による品質差異が縮減し、可読性及び変更容易性を向上させることができた。

５．２ 今後の展望

（1）一層の生産性・品質向上

本フレームワークを利用したＪａｖａシステム開発はまだ始まったばかりであり、生産性・品質をより向上させるために努力が必要と認識している。特にフレームワークを実際に利用した開発者の意見を反映させ、より使い易くしていくことが必要である。

品質向上のためにはテスト工程の一層の改善が必要となる。反復型開発や関連プログラムの変更では、繰り返しテストを行う必要があるため負荷が非常に大きくなる。じゅうぶんな品質を確保しつつ生産性を向上させるには、フレームワークを考慮したテスト項目チェック表の制定といったチェックの充実とともにテストの自動化が必要と考える。今後、ツール導入によるテスト効率化を検討していきたい。

また、マニュアル等のドキュメント整備は必要であるが作成負荷は大きい。そこで、作成負荷の軽減を図るためにプログラムソース内のコメントとコードにより、詳細仕様に係るドキュメントを自動作成する仕組みを整備していきたい。

（2）新技術への対応

Ｗｅｂ及びＪａｖａ技術の進化は非常に速い。今後、多数の新技術の中からユーザーニーズへの対応や開発効率の向上に真に有用なものを見極めて、フレームワークに取り入れていきたいと考える。

例えば、Ｃｏｎｔｒｏｌｌｅｒ層を現在注目されてきているＷｅｂサービスに対応させ、Ｊ２ＥＥと対抗関係にありユーザーインターフェースに優れている「.ＮＥＴ　Ｆｒａｍｅｗｏｒｋ」（マイクロソフト株式会社が提唱）をＶｉｅｗ層に使用することで、OAツールとも密に連携可能なより高い操作性をユーザーに提供できる。本フレームワークは各層の独立性が高いため、このような拡張が行い易い。

また、フレームワークを拡張する際には、今回のＣＲＭシステム開発においても行ったように費用対効果を考慮して自ら作成すべき部分と市販の製品を取り入れた方がよい部分を見極めていくことも重要である。

６．おわりに

「バリアフリーＪａｖａフレームワーク」を利用して開発を行ったＣＲＭシステムは、開発者の７０％がＪａｖａの初級者であったにもかかわらず、当初の計画通りカットオーバーさせることができ、その後も安定稼動している。生産性、品質ともに高レベルを維持して開発を行えたこととユーザーに高い利便性を提供できたことにより、当初設定したねらいはほぼ達成できたと総括している。

とはいえ、既にＣＲＭシステムの第２次開発も始まっており、Ｊａｖａを使った大勢の開発者による大規模システム開発をより一層 ー***「難しい」を「簡単」に***ー していくためには継続的な努力が必要であると気を引き締めている。

本フレームワークの構築事例が、フレームワーク開発を検討されている方々や市販フレームワーク製品導入を検討されている方々にとって何らかの参考になれば幸いである。

本フレームワークの構築により、多様なユーザーニーズに対してＪａｖａの初級者であっても高生産性、高品質で対応できる基盤が完成した。この資産を継続的に維持し拡大発展させ、今後ますます多様化・複雑化していくであろうシステム開発ニーズにスピーディに応えていきたい。

以上

参考文献

１.「実践UML：パターンによるオブジェクト指向開発ガイド」、Craig Larman 著、ピアソン・エデュケーション、１９９８

２.「オブジェクト指向における再利用のためのデザインパターン」、Erich Gamma/Richard Helm/Ralph Johnson/John Vlissides 著、ソフトバンク、１９９９

３.「ｃｏｒｅ Ｊ２ＥＥ Ｐａｔｔｅｒｎｓ」、Deepak Alur/John Crupl/Dan Malks 著、ピアソン・エデュケーション、２００２