Web化のための生産性向上フレームワーク※
－共通プラットフォームによる実現－

IT基盤ソリューション事業部
Web基盤ソリューションセンター
松山 圭一
Keiichi Matsuyama

当社は、Web/／分散オブジェクトによる新しいシステム構築技術のキャッチアップを行うために、WebAF（Web Application Family）と称する「共通ＥＣプラットフォーム」の開発を進めている。

主たる狙いは、受注したWebアプリケーションの早期供給による機会損失の削減と、顧客のTCO削減である。

採用している技術的手段の特徴は、アプリケーション開発を行うに際して、クライアントアプリケーションの実装形式やAPサーバのベンダによる違いを統一的なAPIによって隠蔽するようにしたことが挙げられる。

生産性／メンタナンス性／再利用性を向上させる上で、最も効果的な手段であると評価している。

本論にて、WebAFのコンセプト、および技術的手段の詳細を述べる。

※本論分は、2001年IBMソフトウェア協力企業SE論文で発表したものである。

1. はじめに

昨今のＥＣ分野におけるアプリケーション構築においては、Webブラウザをクライアントの動作環境とし、Webサーバをシステムへのポータルとするアプリケーションが常識となっている。こうした“Web化”の要求は、B2B、B2C、イントラネット、インターネットを問わず、そのスコープを急速に拡大している。さらに、Web化の拡大を加速するアプリケーションサーバやオブジェクト指向技術の登場によってWebアプリケーション自体の構築方法も変化してきている。SI'erは、このような状況に対処すべく競争力を強化し、維持していくことが不可欠である。

本論文では、SI'erが取り得る一つの選択肢として、Webアプリケーション開発のための共通基盤（共通プラットフォーム）という解決策とともに、WebAF（Web Application Family）という実現方法を示す。

2. 問題点

SI'erがWebアプリケーション開発において競争力を強化し、維持していくためには、次の２つの問題を解決することが必要と考える。

2.1. 開発リソースの不足

オンライン商品の多様化、企業情報システムのポータル化による営業活動支援、社内システムの軽量化、ブロードバンド化などが急速に進むなかで、Webアプリケーション構築プロジェクトの納期は非常に短期化している。Time-To-Marketを最重要課題とする.comサイト構築においては、200人月／期間３ヶ月程という案件も珍しくない。一方で、新規に参入するSI'erにとってWebアプリケーション開発のための技術者を案件の規模に応じてダイナミックに調達することは、新しい分野であるということもあり、難しい問題となる。

2.2. 基盤ソフトウェアの氾濫

Webアプリケーションの構築手法やスケーラビリティ、パフォーマンス、安定性を改善するための様々な先端技術や標準が次々と登場している。こうした背景においてアプリケーション開発部署は、各々のアプリケーションを実現するために様々なIT技術要素を組み合わせて案件固有のアーキテクチャ（設計思想）と実現方法で構築された基盤ソフトウェアの上にWebアプリケーションを構築している。

先端技術を導入していることが、一方では、案件獲得のための重要なセールスポイントともなりうるが、先端技術の多様化が急速なために単一のSI'er内部においてもさまざまな基盤ソフトウェアが存在し、これらの上に直接構築されたWebアプリケーションはプロジェクト固有（あるいは、顧客固有）、開発部署固有の開発物となり、SI'er社内またはSI'erとその開発パートナ全体としての開発物の再利用性を阻害することになり、開発生産性を低下させることになる。

3. 課題

Webアプリケーション分野のSI'erという視点に立つとき、上記の問題を解決するためには、次の４つの課題をクリアすることが必要と考える。課題表題に続く括弧内に主に対応する問題点を示す。

3.1. 開発リソースの確保（問題点:開発リソースの不足）

すなわち、必要なスキルを持つ技術者をプロジェクトの規模に応じてダイナミックに調達できること。

Webアプリケーションを開発するにあたって必要なスキルは、Java実装技術から始まり、Webサーバ、アプリケーションサーバ、データベースサーバの利用技術、認証、アクセス制御、トランザクション制御等多義に渡る。それぞれについて高いスキルを持つ技術者を必要なときに必要なだけ集めることは非常に難しい。

3.2. アプリケーションの中立化 （問題点：基盤ソフトウェアの氾濫）

Webクライアントの形式（HTML/JSP、JavaApplet、Javaアプリケーションなど）や、Web/アプリケーションサーバの種類によってアプリケーションの中立化が阻害されないこと。

業務要件を分析し、基本設計、詳細設計、実装へと移行する過程で、アプリケーションは本来関心を払うべきでないプラットフォーム側の制約を受ける。ここで言うプラットフォームとは、実装形式（JavaApplet、HTML/JSP、

Servlet、EJBなどのフレームワーク）やミドルウェアが提供するサービス（認証、アクセス制御、セッション管理、トランザクション管理など）などの基盤ソフトウェアである。

こうしたフレームワークやサービスが規定するAPIが、アプリケーションロジック内に入り込むことは、これらを使うアプリケーション開発者に基盤側のスキルを要求すると共に、作成されたロジック部品の再利用性、メンテナンス性を阻害することになる。

3.3. 工期の短縮（問題点：開発リソースの不足）

開発するモジュール間の関連を疎にすることで、並行開発性を上げ、工期の短縮が行えること。

画面、ロジック、データが、互いに密な関連を持つ場合、たとえばDBテーブルスキーマの変更が画面やロジックの開発に与える手戻りは、非常に大きい。また、画面だけを考えても業務の流れの変更は、画面遷移の変更となり、ソースコードレベルの画面プログラム変更を必要とする。このような並行開発性を阻害する要因を可能な限り取り除くことで、開発期間の短縮が可能になり、プロジェクト計画時の開発工数見積もりもより確実なものとなる。

3.4. パフォーマンスの確保（問題点：基盤ソフトウェアの氾濫）

従来のクライアント/サーバシステムの“Web化”は、Webブラウザ とWebサーバというプラットフォームを活用することで、BtoB、BtoCとビジネスのスコープを飛躍的に拡大する。しかしながら、エンドユーザにとって最大関心事のひとつであるレスポンスや操作性は、“Web化”することによって必ずしも改善しない。BtoBの場合によく見られるように業務が複雑な場合や扱うデータ量が多い場合には、レスポンスや操作性が低下する。

“Web化”に際しては、要求される機能を満たした上で、こうしたデメリットを最小限に抑えることが“ユーザを満足させるシステム”として必要となる。

3. 解決策

Webアプリケーションの開発物は、図１に示すようにクライアントアプリケーション、ポータルアプリケーション、

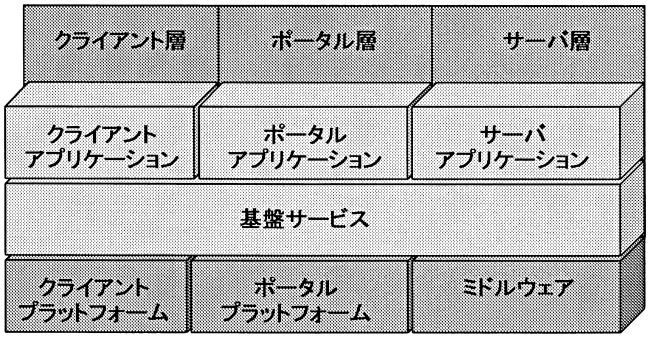

図１　Wabアプリケーションの開発物

サーバアプリケーション、基盤サービスの４つの開発物に分割できる。

本論文においては、クライアント、ポータル、基盤サービスの３つの開発物について上記の課題をクリアするための解決策を示す。

4.1. クライアントアプリケーション

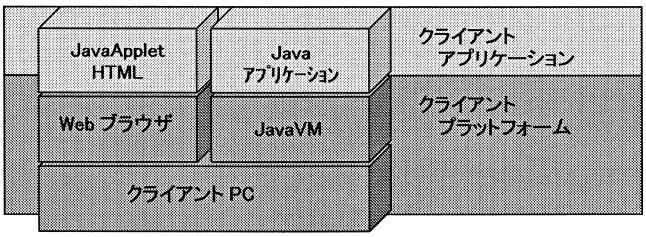

図２　Wabアプリケーションのクライアントアプリケーション

以降、Webアプリケーションを構成する開発物のうち、次のものをクライアントアプリケーションと呼び、問題解決の対象とする。

（１）JavaAppletクライアント

JDK（Java Development Kit）のAWT、Swingライブラリの画面構成部品によって構築されたGUIプログラムであり、Webアプリケーションにおいては、Webブラウザ上でAWTのフレームワークにしたがって動作する。JavaAppletで作成されたWebコンテンツは、Webサーバ上の公開された領域に配置され、クライアントの要求にしたがって、Webブラウザ上に配布（ダウンロード）される。

JavaAppletは、現状、初期ダウンロード時間、画面遷移時間などレスポンス、操作性の面においてもHTMLで作成されたクライアントに比べ扱いにく

いものとなっているが、HTMLよりも複雑な業務を表現できるというメリットを持つ。

（2）HTML/JSPクライアント

通常のHTMLで記述されたWebコンテンツで、それ自体ブラウザ上で動作するプログラムではなく、WebブラウザがHTMLで記述されたスクリプトを解釈することで画面を描画する。したがって、クライアント機の負荷、リソース消費量は、プログラムおよびJavaVM（Vertual Machine）の実効を必要としないため非常に軽く、JavaAppletに比べてレスポンス、操作性に優れている。

さらに、JSP（Java Server Page）という技術を導入することで、Webコンテンツに記述したJavaプログラムをWebサーバ（Servlet Engine）上でコンパイル・実行し、サーバ側での処理結果に応じたダイナミックなHTMLコンテンツを生成することができる。

（3）Javaアプリケーションクライアント

JavaAppletと同じくJDKのAWT、Swingライブラリの画面構成部品によって構築されるが、あくまでもクライアント機のJavaVM上で動作するローカルアプリケーションである。

Javaアプリケーションは、Webサーバからのダウンロードではなく、事前に各クライアント機に配布、インストールされている必要があるため、Webアプリケーションとしては運用・管理面で制約を受けるが、運用時の初期ダウンロード時間を割愛できるというメリットを持つ。

以下に、これら３種類のクライアントアプリケーションに関して、３章の課題をクリアするための解決策を述べる。解決策表題に続く括弧内に主に対応する課題を示す。

4.1.2. Javaプログラミングのスキルを要求しない（課題：開発リソースの確保）

Webアプリケーション開発プロジェクトは、開発パートナを含めた開発リソースプールからプロジェクトの開発期間、リソースの要求量、リソースの要求スキルに応じてダイナミックに構成されるべきである。しかしながら、現在でもJavaプログラミングに関して経験豊富な開発者は少ない。そこでJava言語をあまり知らない“Java初心者”でも、十分に信頼性の高いクライアントアプリケーションを短期間に開発することができる仕組みを構築する。

4.1.3. 仕様変更による手戻りを最小限とする（課題：工期の短縮）

要件定義、外部仕様の決定、データベース構造の決定に要する期間は全体工期のなかで大きな部分を占め見積もりも難しい。しかしながら、通常、クライアントアプリケーションは、外部仕様、データベーススキーマの変更によって大きく影響を受け、仕様変更による手戻りが多きい。そこで、画面仕様の変更に迅速に対応でき、データベーススキーマやサーバ側アプリケーションの変更にも影響を受けないでクライアントアプリケーションを開発できる仕組みを構築する。

4.1.4. JavaAppletの弱点を補強する（課題：パフォーマンスの確保）

クライアント画面をJavaAppletで作成する場合、HTMLに比較して画面の表現方法が豊富であり、入力データチェックなどのクライアントロジックの実装も容易であるというメリットを持つ。この反面、初期ダウンロード、画面の切り替え（画面遷移）に時間がかかり操作性が悪化する場合が多い。

そこで、JavaAppletの長所を生かしながら短所を補強する仕組みを構築する。

4.1.5. 画面スタイルの統一（課題：工期の短縮）

クライアント画面の作成は複数の開発者によって分担され並行で進められていくわけだが、画面のルック＆フィールが開発者ごとにまちまちにならないように画面スタイル（フォーム）、操作性を統一する仕組みを構築する。

4.1.6. 部品蓄積と再利用の促進（課題：アプリケーションの中立化）

個々のプロジェクトという局所的なスコープにおいても画面を構成する部品（入力フィールド、ボタン、表など）、クライアントロジック、ミドルウェアとの接続部分など各画面を作る際に同じ物を繰り返し使うことになるが、個々

のアプリケーションに依存せずプロジェクト間で再利用可能な部品として蓄積し、次のWebアプリケーション開発プロジェクトの工期短縮に役立てることができる仕組みを構築する。

4.1.7. 既存開発ツールとの統合（課題：開発リソースの確保）

現状、Java言語を使ったアプリケーション開発のための統合開発環境(IDE:Integrated Development Environment)が多く市販されている。しかし、これらのIDEは業務を構成する個々の画面を作成するという範囲に留まっており、画面操作に対応するサーバへの要求発行、画面遷移などについては、Java言語でのコーディングが必要となる。またミドルウェア、サーバなどのサブシステムを常に意識したコーディングが必要となる。そこで、個々のアプリケーションやサブシステムに依存しないクライアントアプリケーション実行のための仕組みと、これに準拠する部品群を既存IDEに組み込むことで開発生産性の高い環境を提供できる仕組みを構築する。

4.2. ポータルアプリケーション

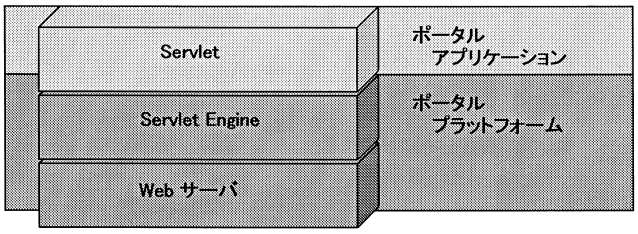

図３　Webアプリケーションのポータルアプリケーション

Webアプリケーションを構成する開発物のうちWebサーバのServlet Engine上で動作し、以下に述べるさまざまな役割を果たすものをポータルアプリケーションと呼ぶことにする。

（1）クライアントアプリケーションとの通信

ユーザの画面操作に対応して送信されてくるクライアントアプリケーションからの処理要求とデータをサーバ側のアプリケーションに伝え、処理の結果をクライアントアプリケーションに返す。

（2）認証

ポータルアプリケーションへのアクセスに対して、ユーザ情報の入力を促し、業務の実行を許されているユーザとして登録されているかどうかを認証サーバやユーザ情報のリポジトリに接続されたサービスに問い合わせて判定する。

（3）アクセス制御

認証を受けたユーザに対して、ポータルアプリケーションの実行に際して十分な権限があることをユーザ情報のリポジトリに接続されたサービスに問い合わせて判定する。

（4）セッション管理

複数ページにまたがるクライアントアプリケーションのステート情報を、ユーザ情報等に関連付けて管理する。

（5）サーバアプリケーションの実行

ユーザが画面を操作することで発行される要求に対応して、サーバ側の一連の処理を実行する。通常は、要求ごと、画面ごとに個別のServletを作成することになる。

（6）トランザクション管理

クライアントアプリケーションからの要求によって実行されるサーバ側の一連の処理を、ひとつのトランザクション境界として管理する。

以下に、ポータルアプリケーションに関して、３章の課題をクリアするための解決策を述べる。解決策表題に続く括弧内に、主として対応する課題を示す。

4.2.2. 通信プロトコルとメッセージフォーマットの統一（課題：アプリケーションの中立化）

4.1で述べただけでもクライアントアプリケーションの実装形式は、JavaApplet、Javaアプリケーション、HTML/JSPと３種類ある。ポータルアプリケーション、サーバアプリケーションとの連携には、実装形式によってさまざまなプロトコルを採用することができるが、特にWebアプリケーションにおいては、ファイアーウォールの構築ポリシーなど顧客情報部門のセキュリティ方針やインフラ構築方針によって影響を受け難いプロトコルに統一する。

プロトコル上で送受信されるメッセージフォーマットは、アプリケーションによってもまちまちである。アプリケーション固有のメッセージフォーマットを解釈する部分がアプリケーションロジックの再利用性を阻害しない様にフォー

マットを統一することになるが、ここで重要なことは、統一されたメッセージフォーマットに対する操作を抽象化することにある。すなわち、操作するデータに依存する特定のAPIではなく、データの意味（メタデータ）をキーにしてデータを操作するという汎用的なAPIに統一する。

4.2.3. 認証・アクセス制御サービスのラッピング（課題：アプリケーションの中立化）

アプリケーションサーバなどのようにWebアプリケーション構築のために市販されているミドルウェアは、認証、アクセス制御に関するサービスを提供している。しかしながら、認証、アクセス制御といったサービスの情報ソースとなるユーザ情報は、既存のシステムによって構築されている場合がほとんどである。

こうした状況の中では、市販ミドルウェアに組み込まれている既存のサービスをそのまま使うことができず、なんらかの独自機構を顧客のシステムに合わせてポータルアプリケーション内に作り込むことになる。

これらの既存サービスやプロジェクト固有の機構に対しては、アプリケーションとの間に共通のモジュール（ラッパー）を設けて、アプリケーションの中立化を図る。

4.2.4. サーバのステートレス化（課題：パフォーマンスの確保）

クライアントアプリケーションのステート情報は、各クライアントアプリケーションに保存する。これにより、ポータルサーバは、ステートレスなサーバとなり、スケーラビリティを向上させることができる。ステート情報の保存形式は、JavaApplet、Javaアプリケーション、HTML/JSPの３種類の実装形式に関わらず共通の方式を適用し、たとえばクッキーといったようなHTMLコンテンツ固有の方式は使わない。

4.2.5. ロジックとのミドルウェアの分離（課題：開発リソースの確保）

認証、アクセス制御、クライアント情報（セッション情報）の伝達、トランザクション境界の生成といったWebサーバやアプリケーションサーバなどのミドルウェアが提供するサービスの操作をアプリケーションロジックから分離する。

ミドルウェアへのアクセス部分は、アプリケーションロジックを呼び出す共通部分として、部品化する。こうすることによって、ロジック開発者はミドルウェアの操作に関心を払う必要がなくなるとともに、アプリケーションロジックの再利用性を向上させることが出来る。

4.3. 基盤サービス

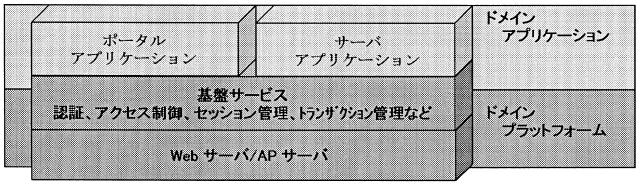

図４　Wabアプリケーションの基盤サービス

4.2で述べたポータルアプリケーションとサーバアプリケーションは、Webサーバのサーブレットエンジンやアプリケーションサーバの EJBコンテナの上に構築され、各基盤が提供する認証、アクセス制御、セッション管理、トランザクション管理などといったサービスを利用する。

ここで言うサーバアプリケーションとは、ビジネスオブジェクトとこれらを操作するビジネスプロセスからなるドメインそのものであり、これに対してポータルアプリケーションは、ドメインへの入り口を管理するアプリケーションと言える。ここでは、これら２種類のアプリケーションをまとめてドメインアプリケーションと呼ぶ。

ドメインアプリケーションを構築するプラットフォームは、納入先のインフラストラクチャ、プロジェクトの予算規模、システムの応答性確保などさまざまな要因によってサーブレットエンジン上にすべてを構築する場合や、EJBコンテナにサーバアプリケーションを分割する場合、EJBコンテナを介してさらに他のサブシステムと連携する場合などがある。

基盤サービスにとってもっとも重要な機能は、クライアントアプリケーションからポータルアプリケーション、サーバアプリケーションへとユーザ情報を確実に伝達するという点にあるが、ミドルウェアの構成によって基盤側が用意するユーザ情報の伝達経路が断たれてしまう場合もある。そこで、それぞれのミドルウェアの認証、アクセス制御、セッション管理、トランザクション管理というサービスを利用しながらも、ミドルウェアの構成にかかわらずユーザ情報を伝達できる仕組みを構築する。

5. 実現方法

上記の解決策を実現する共通プラットフォームを構築し、このプラットフォーム上でアプリケーション構築をおこなう。共通プラットフォームの実現方法の一例をWebAF（Web Application Family）と名付け、以下に説明する。

5.1. WebAFアーキテクチャ

WebAFは､共通プラットフォームであるWebAF/Client、WebAF/Portal、WebAF/FS（Foundation Service：基盤サービス）によってアプリケーションとミドルウェア以降のサブシステムを分離する。WebAFのアーキテクチャを図５に示す。

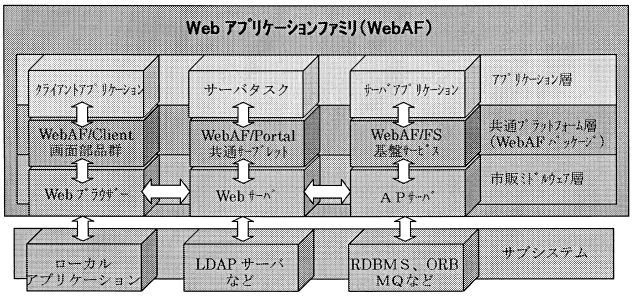

図５　WabAFアーキテクチャ

WebAFは、クライアントアプリケーション、サーバタスク、サーバアプリケーションからなるアプリケーション層、WebAF/Client、WebAF/Portal、WebAF/FSからなる共通プラットフォーム層、Webブラウザ、Webサーバ、ＡＰサーバ（アプリケーションサーバ）からなる市販ミドルウェア層３層によって構成される。また、WebAFは、市販ミドルウェア層を通して４層目のサブシステムとの通信を行う。

5.2. アプリケーション層

WebAFにおいては、アプリケーション層を以下の３種類に分類する。

（1）クライアントアプリケーション

GUI画面と画面遷移ロジックからなり、WebアプリケーションをPAC（Presentation、Abstraction、Control）に分離した場合のPresentationに相当する。クライアントアプリケーションは、WebAF/Clientが提供する共通部品群によって作成され、同じくWebAF/ClientとWebAF/Portalが提供するフレームワークによってWebブラウザを介してWebサーバと通信する。

（2）サーバアプリケーション

ビジネスオブジェクトとこれらを操作するビジネスプロセスからなるドメインを指し､WebアプリケーションをPACに分離した場合のAbstractionに相当する。サーバアプリケーションは、EJBコンテナを使用する場合は、EJB準拠のAPIを介して（各プロバイダ固有の拡張は使用しない）、RDBなどのサブシステムと通信する。

（3）サーバタスク

従来のクライアントアプリケーションでは、クライアントコード内にサーバアプリケーションを操作するプロセスが含まれていた。たとえば、GUI画面コード内にその時の業務ステップが関心のあるサーバアプリケーションを操作し、結果として表示するためのデータを取得するという手順が記述されていた。WebAFでは、このサーバアプリケーション操作部分をタスクとしてクライアントアプリケーションから分離し、Webサーバ上で動作させる。タスクは、Webサーバ上の共通サーブレット（WebAF/Portalによって提供される業務間で共通に使われるServlet）の配下で動作する通常のJavaオブジェクトとして作成される｡WebアプリケーションをPACに分離した場合のControlに相当する。

5.3. WebAF-クライアント

WebAF/Clientは、GUI画面を作成するための共通画面部品群と画面遷移を制御するコントローラによってクライアントアプリケーションを動作させるフレームワークを構成する。

5.3.1. 画面遷移制御コントローラ

WebAF/Clientでは、従来のクライアントアプリケーションをGUI画面、画面遷移ロジック、業務ロジックに分離し、相互の連携を画面遷移制御コントローラ部品によって行う。

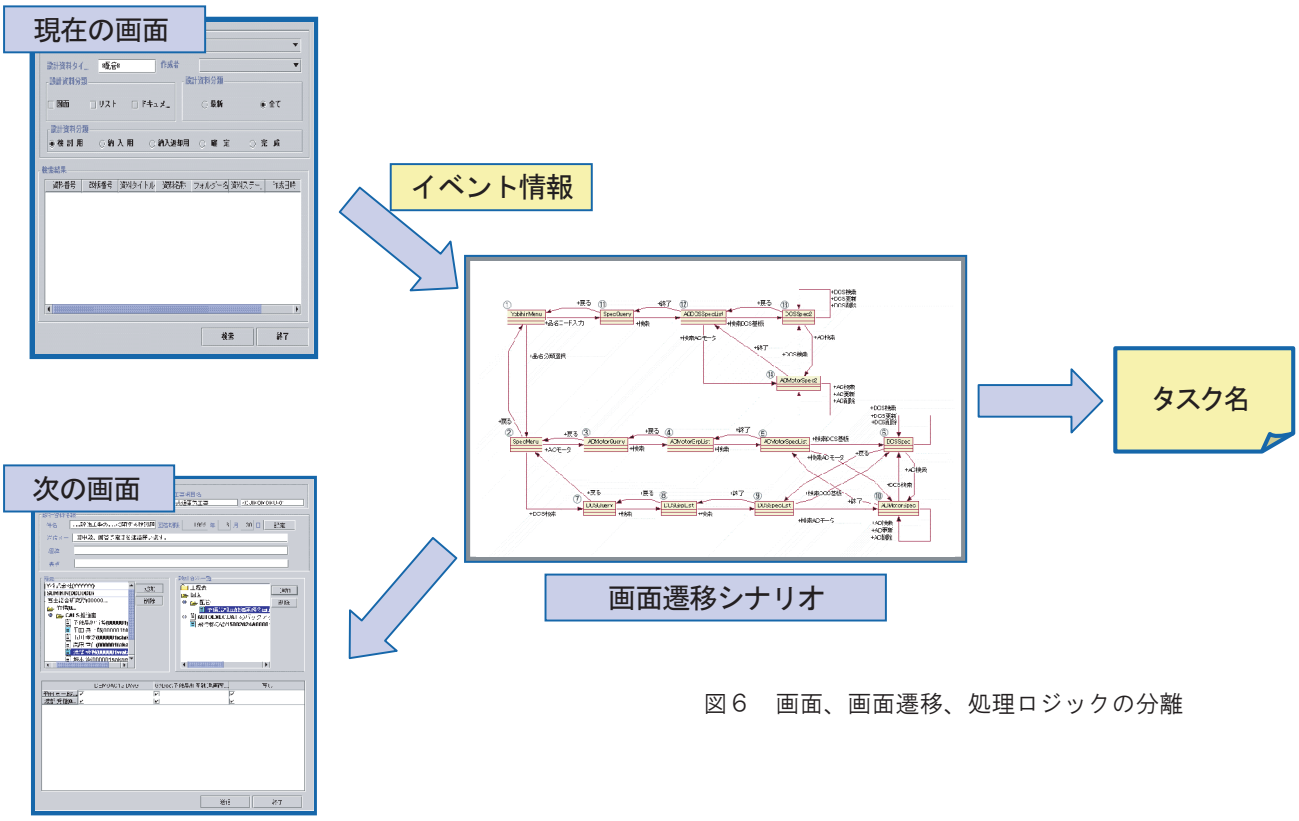

図６　画面、画面遷移、処理ロジックの分離

画面遷移コントローラは、ユーザが現在操作している画面の名前とユーザが画面を操作した結果発生したイベント情報という２つの情報を元にして、実行すべきタスクの名前とタスクが実行された結果取得したデータを表示する次の画面の名前を解決する。この名前の解決は、画面遷移シナリオオブジェクトを参照することによって行われる。画面遷移シナリオオブジェクトは、後述する専用エディタによってノンプログラミングで作成され、Javaのシリアライズドオブジェクトとして、Webサーバ上に配布される。

すなわち、従来のクライアントアプリケーションは、画面、タスク、画面遷移シナリオという３つの独立した開発物となり、画面遷移コントローラによって実行時に関連付けが行われる。また、それぞれの開発物への変更は、他の開発物に影響を与えないため、画面、タスク、画面遷移シナリオの並行開発が可能になる。

画面遷移コントローラは、JavaAppletやJavaアプリケーションでクライアントを作成する場合、WebAFの共通画面部品（画面遷移コントローラパネル）に搭載されて供給される。HTMLでクライアントを作成する場合は、Webサーバ上の共通サーブレットがこの機能を代行するが、両者ともに同一のフレームワークにしたがって動作する。

5.3.2.　共通画面部品

共通画面部品は、大きく分けてJavaApplet、Javaアプリケーションでクライアント画面を作成する場合のJavaPanel部品とHTMLでクライアントを作成する場合のJSP部品に分けられる。

（1）JavaApplet､Javaアプリケーション用共通画面部品

各業務ステップのGUI画面を作るための業務に依存しない部品群を供給する。これらの部品群はWebAFのフレームワークが規定するインターフェイスとJavaBeansインターフェイスに準拠する。統合開発環境（IDE）にライブラリとして登録されて画面作成作業者に配布されるWebAFの画面構成部品は、大別するとスタイル部品/コンテナ部品/画面制御部品がある。

（A）スタイル部品

WebAFでは、ボタン/ラベル/入力フィールド/テーブル/ツリーなどGUI画面のスタイルを決める部品群を用意している。AWTパッケージやSwingパッケージの場合、ボタン単体、フィールド単体、ラベル単体といったように画面構成部

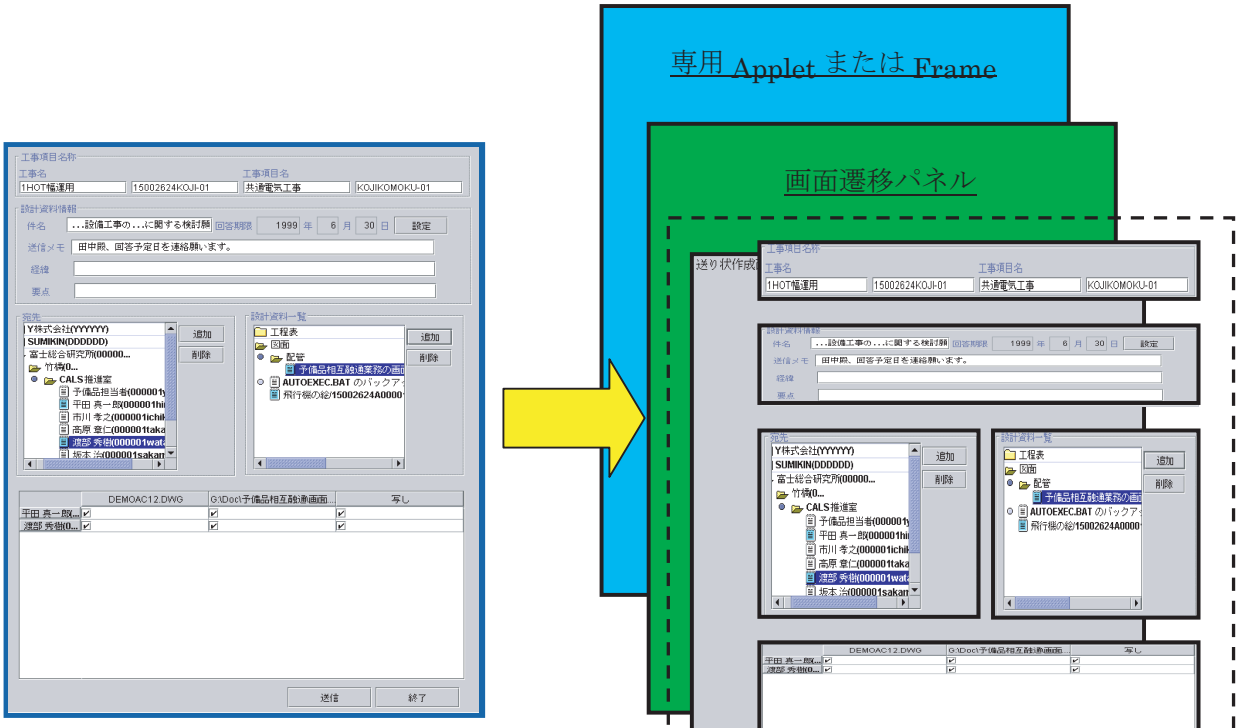

図７　JavaApplet、Javaアプリケーション用共通画面部品

品の粒度が細かくなっており、画面作成作業者が自由にフォームを作れるようになっている。これに対しWebAFのスタイル部品の粒度はAWTでいうところのPanel部品であり、カスタムエディタを装備することで一定のスタイル規則を守りながら、このPanel上に必要な部品をレイアウトすることができる。すなわち、一つのWebAFスタイル部品から、画面スタイルを守りながらもさまざまなフォームを作ることができる。たとえば、スタイル部品ごとにカスタムエディタを装備することで①のような、ごく単純なラベルと入力フィールドのペアから③のように複雑なフォームまでソースコードを編集することなしに、カスタムエディタによる属性設定だけで作成することができる。

カスタムエディタによるフォーム作成支援は、実行時においてもメリットを発揮する。上述したようにカスタムエディタを多機能なものとすることで、一つのスタイル部品からさまざまな画面が作成できるため、ダウンロードの対象となるスタイル部品クラスの種類を極端に少なくすることができる。結果として、画面遷移していくうちに新

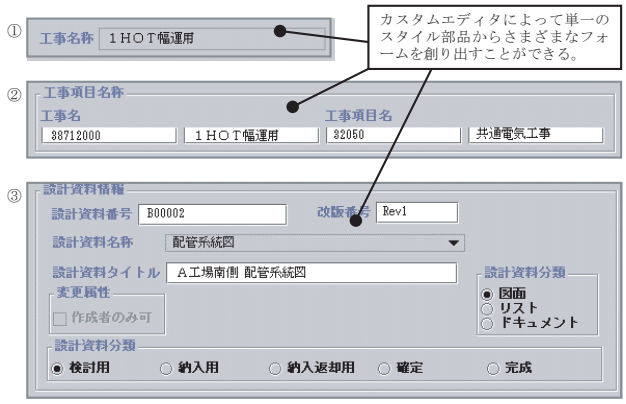

図８　一つのスタイル部品から創り出されるさまざまなフォーム

規にダウンロードするべきスタイル部品クラスがなくなり、画面切り換えの時間を短縮することができる。また、カスタムエディタは、ダウンロードの対象とならないため、多機能化によって大型化しても実行時のレスポンスに影響を与えない。

(B) コンテナ部品

コンテナ部品はさまざまなスタイル部品を載せるためのベース（台紙）となる部品であるが、同時

に画面遷移コントローラがタスク名や次画面を画面遷移シナリオオブジェクトから検索する際のキーとなるイベント情報を生成する。

画面を組み立てるという作業はコンテナ部品の上にコンテナ部品を重ねていき、トップのコンテナ部品上にスタイル部品をレイアウトするという作業であり、画面作業者はIDE上でドラッグ＆ドロップの操作をすることで、この階層構造を作っている。

コンテナ部品は、この階層構造を画面作成時に生成、保持しボタンなどが押された際に、これらのイベントソースからルートのコンテナ部品へと逆にたどるパス（以下、イベントパスと呼ぶ）を画面遷移コントローラにイベント情報として渡す。スタイル部品もコンテナ部品の一種であるためスタイル部品内のイベントソースの名前が重複しない限りイベントパスは業務内でユニークとなり、タスクや次画面を検索するためのキー情報となる。

ルートのコンテナ部品は、他のコンテナ部品と異なり、その画面上に配置されたすべてのスタイル部品とコンテナ部品が共通に参照できるデータ格納部品を持っている。各スタイル部品とデータ格納部品との関連付けも、各スタイル部品のカスタムエディタによって設定できる。

(C) 画面制御部品

画面制御部品は、JavaAppletまたはJavaFrameとして供給される。これらの部品には画面遷移コントローラが搭載されていて、IDE上で作成した一つの業務を構成する複数の画面を画面遷移シナリオオブジェクトにしたがって遷移させるとともにWebAF/Portalとの通信を行い、ユーザの画面操作に対応した要求の発行とサーバ側の処理結果の取得を行う。JavaApplet用の画面制御部品の場合は、さらに次画面クラスのダウンロードをWebブラウザに依頼する。

（2）HTML用共通画面部品

HTMLという実装形式でクライアント画面を作成する場合に付いては、JSP（Java Server Page）によって造られた基本的な部品群を用意する。図９にフレーム分割を行った場合の画面構成例を示す。

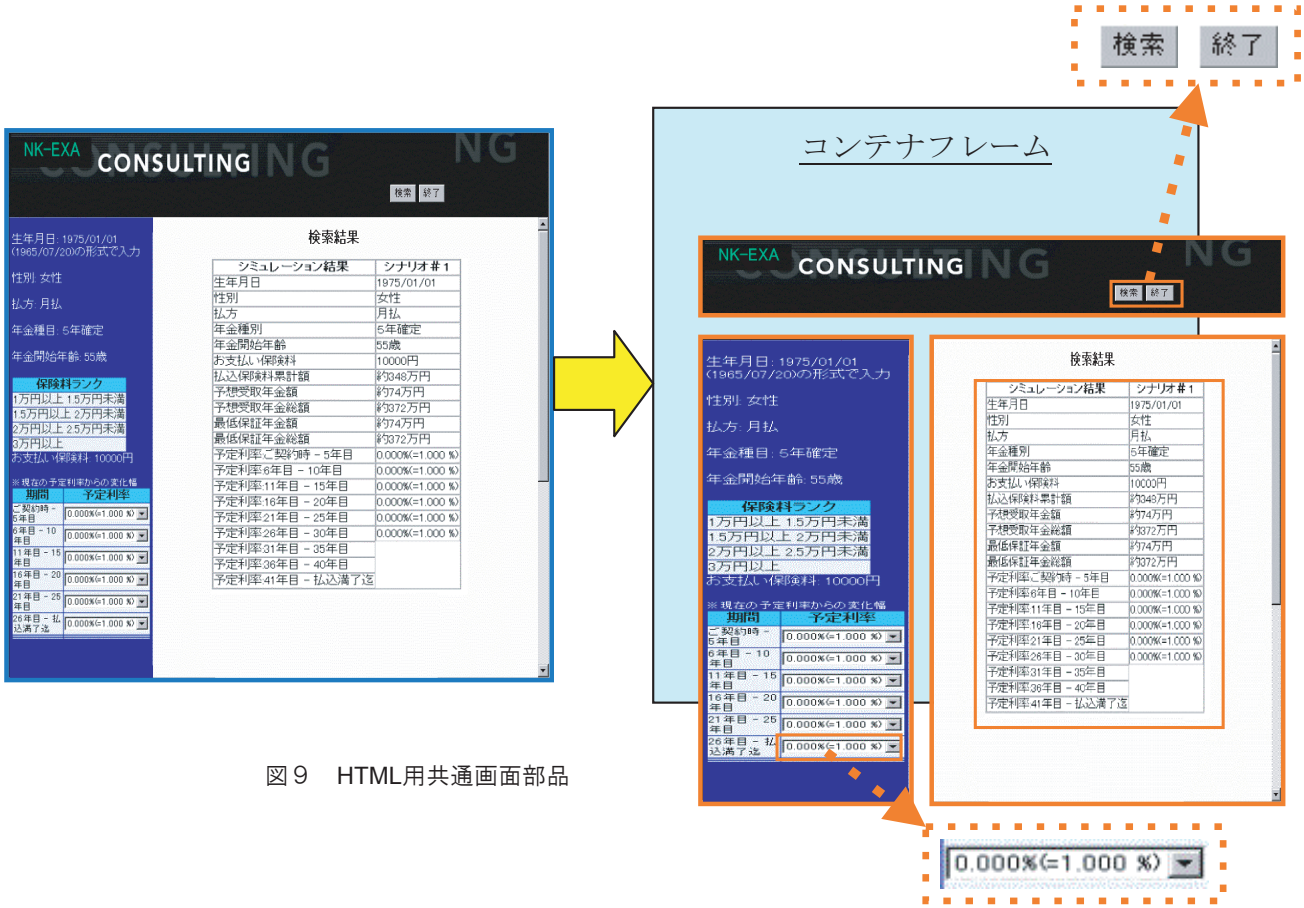

図９　HTML用共通画面部品

HTMLでWebAFのクライアントを作成する際の画面構成部品は、JavaAppletやJavaアプリケーションでクライアントアプリケーションを作成する場合の画面構成部品に比較するとテキストフィールド、各種ボタン、コンボボックス、テーブルなどHTMLで表現可能な単体部品となる。部品のレイアウトや画面の色などは、HTMLのスクリプティングによって行うため、開発効率は落ちるが、HTMLコンテンツ内にJava言語の記述をする必要がないので、JSPを使いながらもavaプログラミングのスキルを必要としない。

コンテナフレームは、フレーム分割を行う際に各フレームが共通に参照するデータをhiddenパラメータとして保持する。

JavaAppletやJavaアプリケーション用に提供されている画面制御部品（画面遷移コントローラを搭載した部品）が果たす役割は、共通サーブレットに移されるが、まったく同一のフレームワークにしたがって動作する。

5.3.3. クライアントアプリケーション開発支援環境

市販されているJava統合開発環境（IDE：Integrated Development Environment）と組み合わせることでノンプログラミング環境を実現することができる。

（1）統合開発環境

現在、さまざまなIDEが市販されている。Java言語でGUIを作成する際のさまざまな画面構成部品（JavaBeans）がライブラリとしてこうした開発環境に供給されている。

しかし、これらのIDEは業務の個々の画面を作成するという範囲に留まっており、画面遷移ロジックやミドルウェア、サーバなどのサブシステムを常に意識したコーディングが必要となる。

WebAFは、画面遷移コントローラの部品供給と各画面構成部品へのカスタムエディタによる入力支援によって、画面作成者にドラグ＆ドロップ操作と属性設定だけのノンプログラミング環境を提供する。

（2）シナリオエディタ

先に述べた画面遷移シナリオオブジェクトの作成においては、シナリオエディタという簡易ツールを供給する。画面作成者が画面仕様にしたがって作成した画面クラス群を読み込み、コンテナ部品に記録された画面構成情報をもとに各画面操作に対応して実行されるタスク名、画面遷移の仕方（次にどの画面に遷移するかとか次の画面にどのようにデータを渡すかなど）を設定する。

従来、プログラミングによってハードコーディングされていたこれらの処理を簡易エディタによる属性設定に置き換えることで、画面、画面遷移、タスクの三者がソースコードレベルで相互に依存しない開発環境を実現することができる。

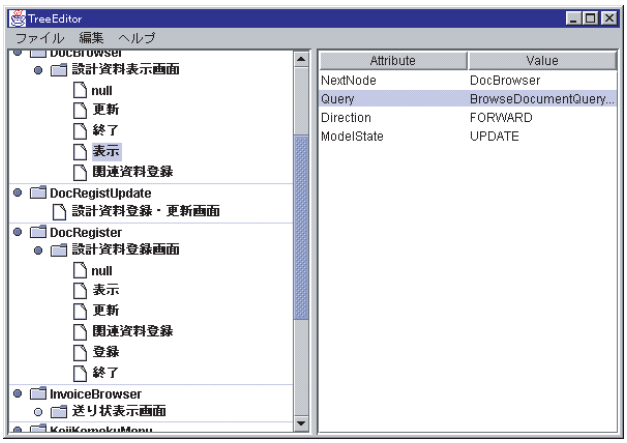

図10　シナリオエディタ

5.4. WebAF-ポータル

WebAF/Portalは、図11に示すように共通サーブレット、タスク、トークンによって構成する。

5.4.1. 共通サーブレット

共通サーブレットは、3種類（JavaApplet、HTML/JSP、Javaアプリケーション）の実装形式で作成されたクライアントアプリケーションとの通信を行う。クライアントアプリケーションと共通サーブレット間の通信は、トークンと呼ぶ共通のフォーマットで記述されたメッセージをHTTPプロトコル上で送受信することによって行われる。トークンは、Javaのオブジェクトとして生成され、シリアライズ（直列化）されてHTTPプロトコル上に乗せ運ばれる。

共通サーブレットは、渡されたトークンから要求名、データを読み出し、該当するタスクを選択、実行する。通信プロトコルとメッセージフォーマットを統一することで、ポータルアプリケーションを通信部分と業務ロジック部分

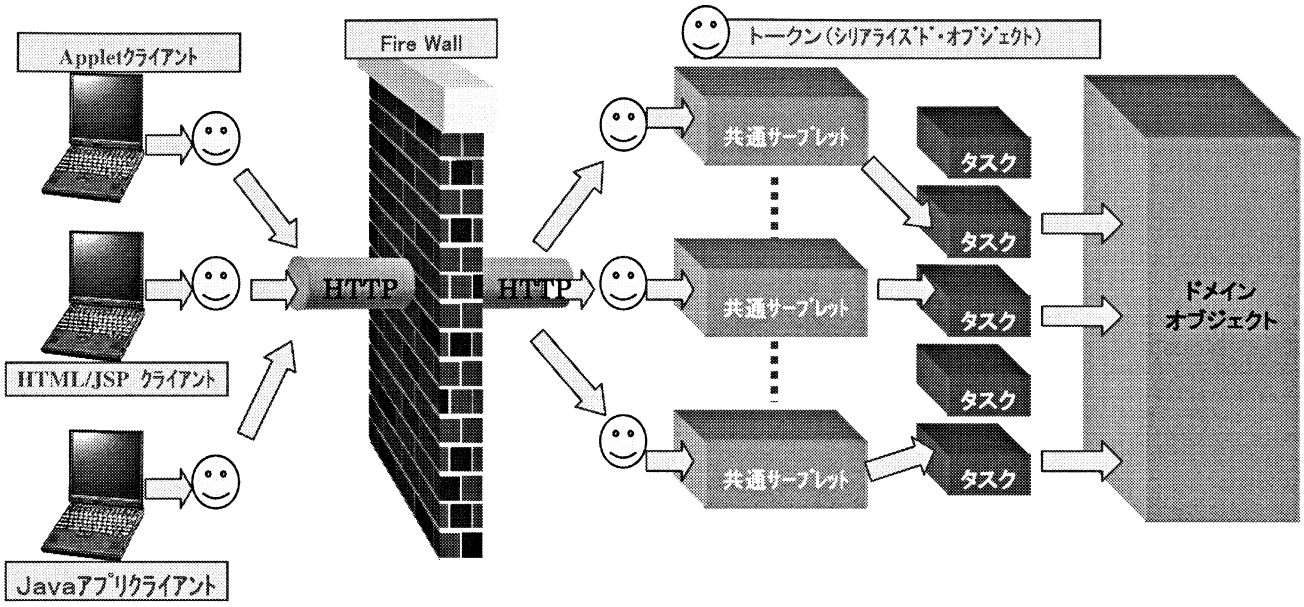

図11　WebAF/Portal

に分離し、前者を共通サーブレットとして単一化することができる。すなわち、共通サーブレットは、業務に対して１種類のみ用意され、業務を構成する複数の業務ロジックを呼び出すディスパッチャとして動作する。

5.4.2. タスク

タスクは、クライアントアプリケーションからの要求に対応して、共通サーブレットによって生成され実行される業務ロジックであるが、タスクは、図12に示すように２つの部分に分かれる。

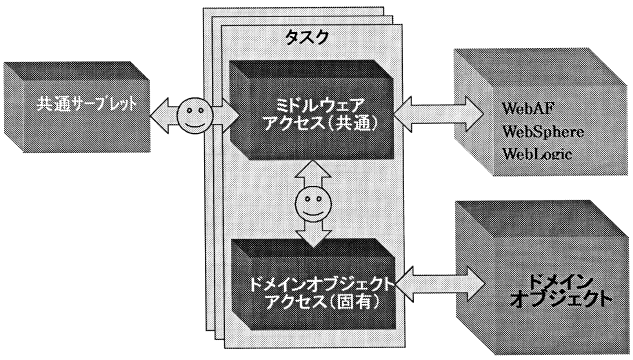

図12　タスク

認証、アクセス制御、ユーザ情報の伝達、セッション管理、トランザクション管理といったミドルウェアやサブシステムにアクセスしてサービスを受ける部分（共通：ミドルウェアアクセス）を、業務ロジック部分（固有：ドメインオブジェクトアクセス）と分離する。

共通サーブレットは、タスクの共通部分を呼び出して実行する。タスクの共通部分は、所定の処理を行った後にタスクの固有部分を呼び出して実行する。

タスクの共通部分は、WebAFによってスーパークラスとして供給されるため、ロジック開発者が関心を持つ必要の無いミドルウェア操作をタスクの固有部分に記述する業務ロジックから取り除くことができ、ロジック開発に注力することができる。

また、プロバイダの違いやバージョンの違いによるミドルウェアの呼び出しシーケンスの違いもタスクの共通部分で吸収することによって、異なるミドルウェア間でのタスクの再利用が可能になる。

5.4.3. トークン

クライアントアプリケーションとポータルアプリケーションとの間のメッセージフォーマットは、クライアントアプリケーションの実装形式に関わらず、トークンという共通のフォーマットとする。

トークンには、クライアントアプリケーションからの要求名やデータ、ユーザの情報などがメタデータ（データの意味）とデータというセットで格納されている。

たとえば、共通サーブレットは、トークンから“要求名（タスク名）”を取り出して、タスクを生成、実行し、タスクの共通部分は、“ユーザID”を取り出してアクセス権限の判定を行い、タスクの固有部分が、“入力データ”を取り出して業務ロジックを実行する。取得された処理結果は、“出力データ”としてトークンに格納され、往路と同じ経路をたどってクライアントに返される。

5.5. WebAF-基盤サービス

Webアプリケーション構築の際に必要となるいくつかの機能を基盤サービスとして提供する。これらの基盤サービスに対する操作は、共通サーブレットやタスクの共通部分で行われ、判断に必要なパラメータは、プロパティエディタによって設定されるため、アプリケーションロジック内には、基盤サービスに関する操作は一切記述されることがなく、WebAFやミドルウェアに対して中立なロジック部品として作成可能となる。

5.5.1. プロパティエディタ

アプリケーション層、共通プラットフォーム層、市販ミドルウェア層の３層に渡る属性設定（プロパティ設定）を統合的に編集するためのツールを提供する。

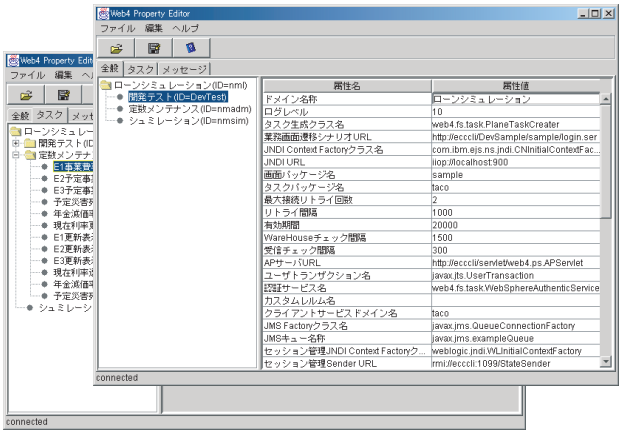

図13　プロパティエディタ

従来アプリケーション中にハードコーディングされていたミドルウェア操作時のパラメータ、アプリケーションサーバなどのミドルウェアごとに、それぞれの設定ツールによって行われていた運用時の設定、WebAFが提供する基盤サービスの設定などをプロパティファイルという定義体に記述することで、アプリケーションロジックから基盤操作の処理を分離することができる。

プロパティエディタは、こうしたさまざまな属性設定作業を単一のエディタ上で統合的に行い、XMLで記述された各プロパティファイルに反映する。

5.5.2. 認証、アクセス制御サービス

クライアントアプリケーションから送られてきたリクエスト、すなわちトークンに書き込まれている情報に対して、共通サーブレットが認証を行い、認定されたユーザに対してだけタスクの生成を行う。タスクの実行に際しては、タスクの共通部分が同じくトークンからユーザ情報を取り出して、アクセス権限の判定を行う。

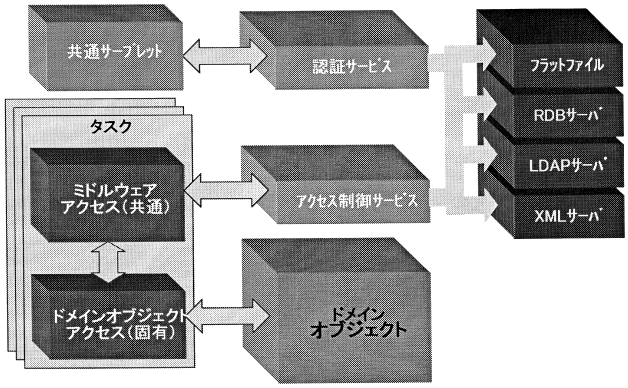

図14　認証・アクセス制御

タスクの共通部分によってクライアントが十分なアクセス権限を持っていると判定された場合のみ、タスクの固有部分、すなわちアプリケーションロジックが実行される。

これらの、認証サービス、アクセス制御サービスについては、APIを規定するのみで、さまざまなユーザ情報のリポジトリや認証サーバに対して実装を変えることになる。

5.5.3. 業務セッション管理サービス

ひとつの業務は、複数の業務ステップによって構成される。この一連の業務ステップ（画面）をひとつのセッションと見なし、“業務セッション”と呼ぶ。

ユーザが業務を遂行中にクライアントPCやブラウザ、ネットワークに障害が起こった場合､ユーザはそれまで行った作業を復旧後に再度やり直すことになる。

業務セッション管理サービスは、障害が発生する直前の

クライアントの状態を“仕掛かり状態”としてユーザ情報と関連付けてサーバ側に保存する。これによって、ユーザが復旧後に業務に再入してきた際には、再度認証した後に仕掛かり状態の画面を再生することが可能となる。したがって、ユーザは障害発生直前の画面から業務を再開することが出来る。

また、次に述べるトランザクション管理サービスと連携させることによって、クライアント側に障害が起きた場合でもトランザクション境界の維持が可能となる。

(1)仕掛かり状態の保存

クライアントから要求があると、共通サーブレットはクライアントからトークンをHTTP経由で取得する。取得したトークンにはユーザ情報、リクエスト情報、画面情報が格納されている。共通サーブレットはそのトークンをタスク共通部分に渡し、タスク実行要求を出す。要求を受けたタスク共通部分は、まずタスク固有部分に対してタスクの実行要求を出し、その結果として先ほどのトークンにタスクの実行結果が格納されたものを取得する。次に、タスク共通部分は業務セッション管理サービスにトークンを渡し、仕掛かり状態の更新を要請する。要求を受けた業務セッション管理サービスは、ユーザ情報をキーにして、受け取ったトークンを前回のものと入れ替えて、仕掛かり状態の永続化を行う。タスク共通部分は業務セッション管理サービスが永続化を行っていると同時に、共通サーブレットへ結果が格納されたトークンから画面情報を切り離したものを送出する。トークンを受け取った共通サーブレットは、このトークンを基に次画面の生成を画面コントローラに要求し画面遷移を行う。

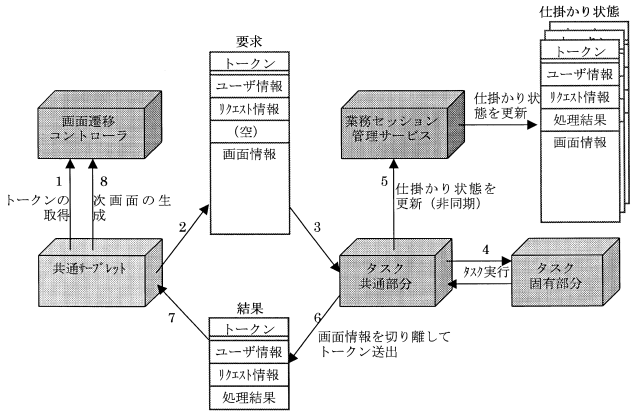

図15　仕掛かり状態の保存

(2)仕掛かり状態の再生

仕掛かり状態の再生要求を受けた共通サーブレットは、業務セッション管理サービスへ要求を受けたクライアントの仕掛かり画面への復帰要求を出す。要求を受けた業務セッションサービスは永続化先からこのクライアントの仕掛かり状態をユーザ情報のキーで検索する。仕掛かり状態が見つかったらそのトークンを復元し、共通サーブレットに送出する。トークンを受け取った共通サーブレットはそれをもに画面コントローラへ仕掛かり画面の再生要求を出し、次画面を表示する。

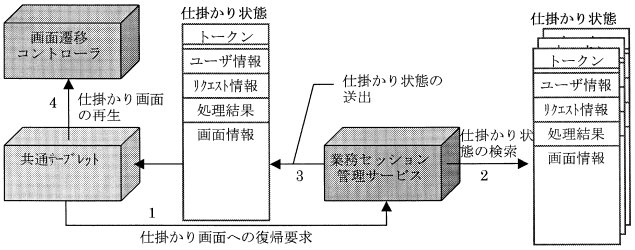

図16　仕掛かり状態の再生

5.5.4. トランザクション管理サービス

トランザクション境界の範囲として、以下の３種類を用意する。どの範囲を境界とするかは、プロパティーエディターによってタスクの属性として設定する。設定された情報はタスクの共通部分によって実行時に判断され、トークンから取り出したユーザ情報に対して、ひとつのトランザクション境界を割り当てる。したがって、タスクの固有部分、すなわちアプリケーションロジック内にトランザクションに関する記述が入ることがなく、ソースコードを変更することなしに、トランザクション境界の範囲を変更することが出来る。WebAFのトランザクション管理サービスは、クライアントアプリケーションに対して以下の３種類のトランザクション境界を維持することを保証する。

(A) シングルリクエスト境界

クライアントアプリケーションからの要求に起因して実行される複数のサーバアプリケーションをトランザクション境界に入れ、境界内でエラーが起きたときに、これらのサーバアプリケーションですでに行われた処理をロールバックする。た

とえば、画面上の一つのボタン操作によって起きるサーバアプリケーションの実行をトランザクション境界とする。

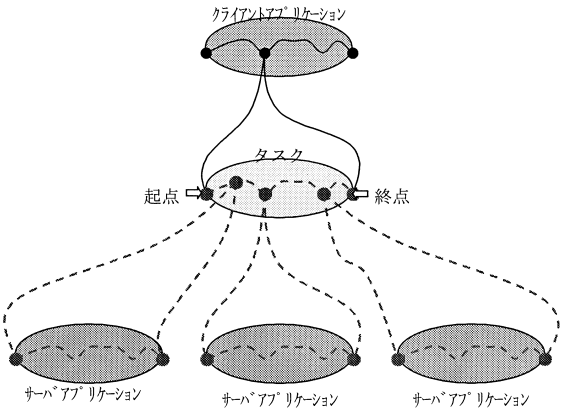

図17　シングルリクエスト境界

(B) マルチリクエスト境界

クライアントアプリケーションからの連続した複数の要求に起因して実行される複数のサーバアプリケーションをトランザクション境界に入れ、境界内でエラーが起きたときに、これらのサーバアプリケーションですでに行われた処理をロールバックする。たとえば、ひとつの画面上にある複数のボタン操作や業務を構成する複数の画面でトランザクション境界を生成する場合である。

(C) マルチクライアント境界

基本的にマルチリクエスト境界の場合と同じであるが､Webブラウザやネットワークの障害によって復旧後、業務に再入してきたクライアントに対してトランザクション境界の維持を保証する。

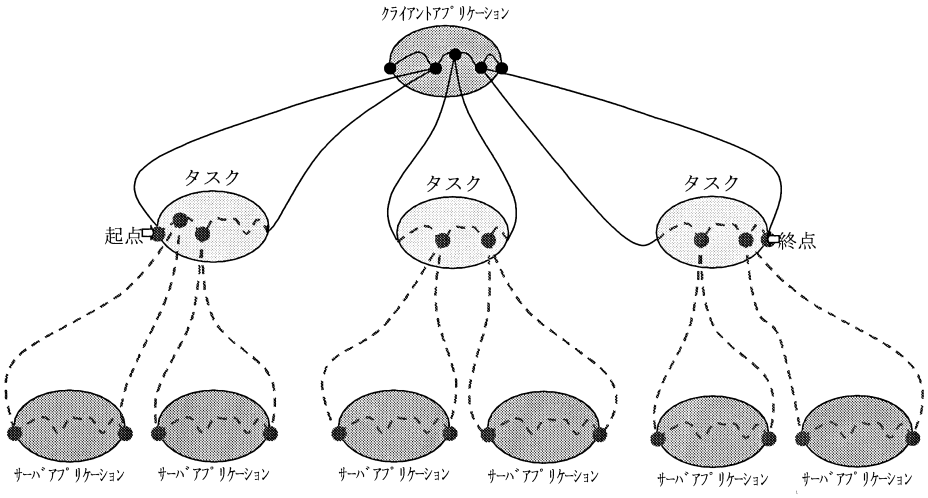

図18　マルチリクエスト境界

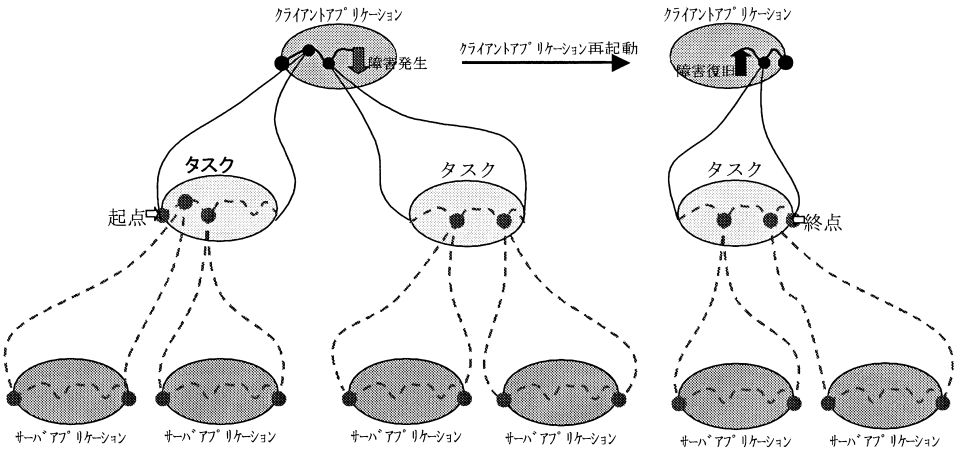

図19　マルチクライアント境界

5.6. WebAFによる開発プロセスの並列化

WebAFでのクライアント／ポータルアプリケーション実装作業は、前述したように画面作成、タスク作成、画面遷移シナリオ作成の３工程であるが、各工程に発生した仕様変更による手戻りが他の工程に波及することがなく、並列化することができる。

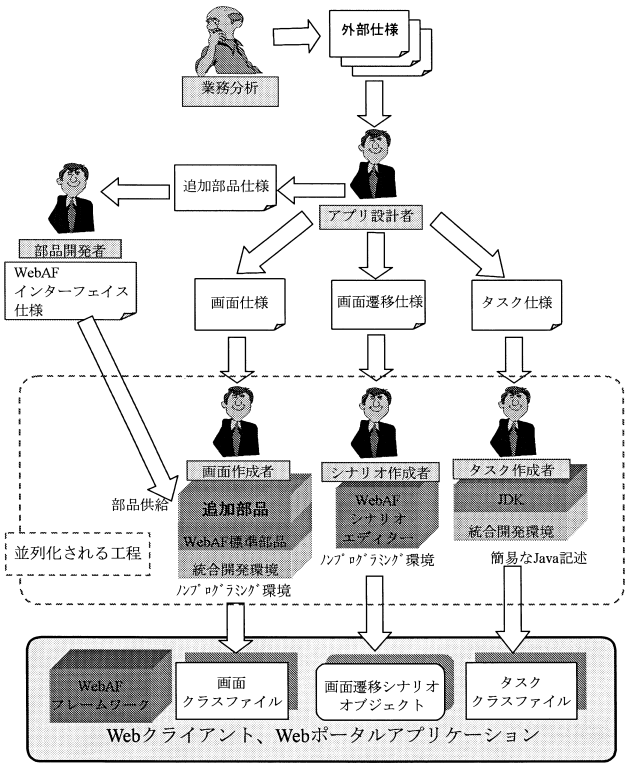

図20　WebAFによる工程の並列化

図20に示すように、業務分析、基本設計、詳細設計を経て、作成された画面仕様、タスク仕様、画面遷移仕様は、それぞれを実装するのに必要なスキルを持つ開発者に渡される。

画面作成者は自分が組み立てている画面から発行される要求や、次にどの画面に遷移するかといったことを全く意識する必要がなく、WebAFが供給する画面部品を使って画面仕様を満たす入出力フォームを組み立てる。タスク作成者も画面や画面遷移といったことに関心を払うことなく、タスク仕様を満たすタスククラスを実装する。シナリオ作成者は、WebAFが提供するシナリオエディタによって画面遷移シナリオオブジェクトを作成する。

６．終わりに

これまでにも、さまざまな方法で部品の蓄積・再利用という試みが行われてきた。これらの試みはオブジェクト指向技術の助けを借りて、コンポーネット流通という形で市場にも現われてきている。SI'erの視点に立つとき、部品の蓄積・再利用という命題は、生産性向上という命題に置き換えることが出来る。また、SI'erにとってもっとも価値ある資産は、ビジネスオブジェクトやビジネスプロセスで構成されるサーバアプリケーションであり、これらの開発に注力でき、かつ、再利用可能な資産として蓄積していくための共通プラットフォームが必要である。

本論文においては、Webアプリケーション構築にあたってもっとも大きな部分を占めるサーバアプリケーション層の開発に言及していないが、この層についても、他の層と同様の視点に立ち問題解決を行っていくことを今後の課題としたい。

＜問い合せ先＞
ＩＴ基盤ソリューション事業部
Web基盤ソリューションセンター
Tel.　044-540-2066　松山　圭一
E-mail：keiichi-matsuyama@exa-corp.co.jp