基幹業務向け WEB 基盤の構築

～ 「オープン FEP」による AP サーバ統合 ～

パナソニック電工インフォメーションシステムズ（株）

■ 執筆者Ｐｒｏｆｉｌｅ ■

甲谷　龍二

1989 年　松下電工株式会社
（現パナソニック電工株式会社）入社
基幹系業務インフラ担当
2000 年　松下電工インフォメーションシステムズ（株）
（現パナソニック電工インフォメーションシステムズ（株））転籍
2010 年　現在 技術開発部門 R&D センター所属
IS 技術担当

黒田　尚志

1982 年　松下電工株式会社
（現パナソニック電工株式会社）入社
受発注システム担当
2000 年　松下電工インフォメーションシステムズ（株）
（現パナソニック電工インフォメーションシステムズ（株））転籍
2010 年　現在 技術開発部門 R&D センター所属
IS 技術担当

■ 論文要旨 ■

2006 年7月「全てのビジネスシーンを WEB＆DIGITAL にフォーカスさせ戦力アップ」という方針が、パナソニック電工株式会社より示された。これを具現化するために、既存および新規の WEB システムに加え、1971 年からメインフレームで稼働している大規模な基幹システムまでを統合できる「高性能な WEB 基盤」が必要となった。

試行錯誤の末、自社開発し「オープン FEP」と名付けたサーバ配下に全ての WEB、AP サーバを配置するシステム構成に至った。この「オープン FEP」はリバースプロキシ機能に加え、きめ細かく高性能なロードバランス、SSO、セション管理、トランザクション処理、稼働ログ収集という機能をもつ。結果、100 万アクセス／時に耐える性能を実現し、メインフレーム並の性能と可用性をもった WEB 基盤を確立できた。本論では「オープン FEP」に至った経緯、機能、効果、そして今後の展開を論じる。

■ 論文目次

■ 図表一覧

１．情報システムの変遷

１．１　メインフレームからオープンへ

当社はパナソニック電工株式会社（以下、Ｐ電工）の情報システム部門から独立した会社であり、親会社Ｐ電工のネットワーク構築、システム開発、その後の運用・メンテナンスなど IT 全般を担ってきた。その歴史は古く 1961 年「PCS 導入」（パンチカードシステム）から始まっている（図１参）。特に現在も稼働している販売流通管理システム「アロー」は 1971 年に第１次、1982 年に第２次、1992 年に第３次と、約 10 年ごとに大規模な改変を繰り返すことにより、常に時流に即した機能・性能を維持し続けてきた「メインフレームで稼働する大規模な基幹システム」である。

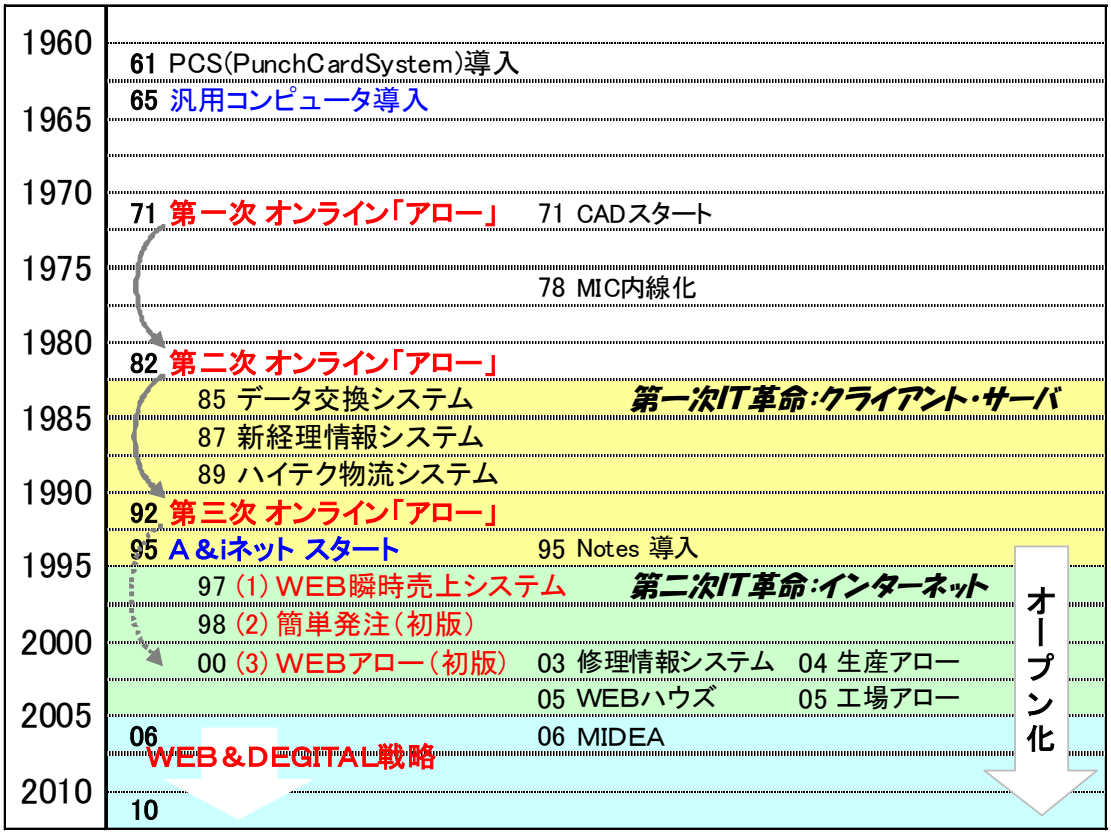

図 １　情報システムの変遷

しかしながら、1995 年にネットワークをレガシーからオープンネットワーク「通称：Ａ＆Ｉネット」へ一気に切り替えたことを機に、システム開発もオープンシステム一色となっていった。そこでメインフレームで稼働する大規模な基幹システム「アロー」はどうしていくべきなのか、2002 年（1992 年第３次から 10 年後）を目安に方針を決定するためにオープン系の技術動向を調査・試行していくことが必要となった。

１．２　緩やかなるオープン化

基幹システム「アロー」は約 3,000 画面、端末台数２万台、月間の入出庫データ 400 万件、秒あたり 200 トランザクションを１秒未満のレスポンスで処理するオンラインシ

ステムである。ユーザは事業部、営業所など社内だけでなく、協力工場、代理店まで広がり、EDI はもちろん他社とのシステム連携も数多く行っている。システムの規模、および求められるメインフレーム並の高い性能から「一挙に即マイグレーションは非現実的」と考え、当時業務への適用はなかった WEB 技術の現状・動向を調査・試行を通じてスキル蓄積をしながら「あるべき姿」を模索していくことにした。

■ WEB 瞬時売上システム

その第１弾が 1997 年に公開した「WEB 瞬時売上システム」である。これはメインフレームでリアル集計している売上情報をサーバにリアル連携させ、JAVA アプレットでブラウザ上にグラフ表現したものである。短期実現のためにメインフレーム・サーバ間のリアル連携に富士通製ミドルウェア「NetStage」を適用したものの、UI のみ WEB 化できる処理パターンに手応えを感じ、JAVA アプレットの未完成度を体感できた。

■ 簡単発注画面

次に実施したのが「WEB で誰もが直感的に使える簡単発注画面ができないか」という試みである。メインフレーム・サーバ間の連携処理を今回は市販のミドルウェアでなく自社開発し、JAVA サーブレットを使い実現させた。連携技術を内作し「ビジネスロジックはメインフレームのまま、UI のみ WEB 化」という処理パターンを修得できたこと、JAVA サーブレットに将来性を感じるとともに、フレームワークの必要性に気付いたことが収穫であった。

■ WEB アロー

これら２つの試行により WEB の性能と技術が見え、いよいよ WEB 基盤として「ブラウザだけで今まで通りメインフレームの画面が使え、かつ新たに開発する WEB 画面もシームレスに使える稼働環境」を構築することにした。この環境が実現できれば、メインフレームを使い続けながら順次 WEB 化を進めるという「緩やかなるオープン化」への道が見えるのである。この取り組みは 1998 年 10 月 JAVA サーブレットの再学習から始め、フレームワークの自社開発に着手した。1999 年３月からは「WEB アロー構築」と称したプロジェクト体制をとり、スクラップ＆ビルドを繰り返し 2000 年５月にようやく初版の「WEB アロー」を完成させることができた（図２）。

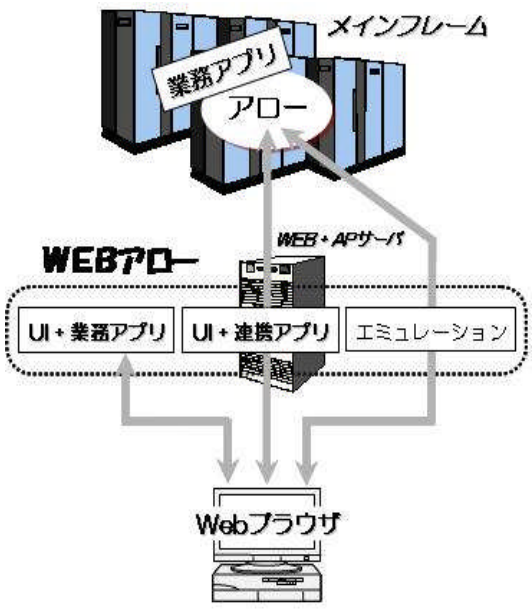

図 2　WEB アロー

その後 2002 年 10 月にテスト本番開始にまでたどり着けたが、実際の本番利用に耐え得るレベルにまで磨き上げるにはさらに約２年の時間を要した。2005 年７月には登録端末台数が 15,000 台にまで拡大したものの、そのユーザの大半は従来端末利用とは無縁であった営業セールスによる照会業務利用であり、フル活用されている端末は少なかった。

ヘビーユーザである業務担当者は市販エミュレータを使い続けており、「緩やかなるオープン化」のための WEB 基盤確立にはさらに時間を要した。

１．３　「WEB アロー構築」でわかったこと

「WEB アロー」は富士通メインフレームで稼働する AIM オンラインを対象とした WEB エミュレータであると同時に、JAVA サーブレット開発環境としてフレームワークを提供し、メインフレームとサーバのそれぞれのアプリを WEB ブラウザだけでシームレスに使える WEB 基盤である。この初期構築段階で次の３点が明確になった。

（１）　端末にインストールが必須であったエミュレータ・ソフトが不要となることで「コスト削減」はもちろん、ユーザの裾野を代理店から工務店・工事店・販売店へ拡大できる。

（２）　市販のエミュレータ・ソフトに比べ、自作したばかりの WEB エミュレータは機能面・性能面でまだかなりの改善が必要であった。特にベテランユーザの見栄え、操作性、レスポンスに対するこだわりは半端でなかった。

（３）　自作した WEB エミュレータは１トランザクションの処理は軽いが、処理集中による負荷が非常に高いことがわかった。この対策としては高性能サーバ１台よりも、標準的なスペックのサーバを複数台並べロードバランスさせるというシステム構成の方が有効であることがわかった。

２．「全ては WEB ポータルから」を実現するために

２．１　WEB&DIGITAL 戦略

「WEB アロー」が安定した頃、大きな節目が訪れた。2006 年７月に「全てのビジネスシーンを WEB&DIGITAL にフォーカスさせ戦闘力アップを図る」という方針がＰ電工より打ち出され、現行 SCM の対象範囲の拡大と連携の強化実現に向けた企画がスタートした。これにより取り組むべき課題が変わった。「メインフレームの基幹システム「アロー」をマイグレーションでき得る基盤構築」という従来からの課題に加え、「既に乱立しだしていた WEB システムと、これから構築していく WEB システムまでを統合できる基盤」という要件が加わり、より範疇が広く深い課題となった。

つまり、ユーザがシステム構成を意識することなく活用できる「メインフレームと WEB の両方を使ったハイブリット型システム」を、メインフレーム並の性能・安定性・運用性をもって実現できる WEB 基盤・システム構成を構築することが新たな目標となった。

２．２　WEB 基盤の要件

具体的にどんな WEB 基盤を目指さなければならないのか。システム要件として特に重視した５つのポイントを列挙する。

（１）「メインフレーム並」をオープンで実現

〇高性能・高負荷耐久性

・平均秒 200 トラン、ピーク時はその２倍の負荷に耐える

・１トランのサーバ内レスポンスは１秒以内

〇永続性のあるセション管理

・オンライン当日中のセション維持

〇安定性：ハード・ネットワークの冗長化

・多重化により部分的なシステム障害に耐え得る構成

〇運用性：システム構成の可用性・可変性・拡張性

・システム構成の変更・拡張が容易

・内部統制、システム監視に使えるシステム稼働状況の把握

・安全なアプリケーションの本番移行

・アプリケーション障害発生時に本番モジュールをリアルタイム入替え

（２）サーバ統合できる構成

〇既存はもちろん、新規開発の WEB システムまでを対象

〇イントラネットからだけでなくインターネットからの利用も想定したシステム構成

（３）全てのシステムがシングルサインオンで利用できる

〇ユーザ ID による認証処理への負荷集中対策

〇セション管理のために端末特定情報とユーザ ID を紐付け管理

〇パスワードの定期的変更を強要、パスワードの連続入力ミスによる使用制限

（４）メインフレームの基幹システムを「緩やかにオープン化」できる

〇WEB エミュレーション機能実装と、WEB システムとのシームレスな共存（ハイブリット型）

〇メインフレームのトランザクション情報をほぼリアルタイムにサーバへ連携

（５）システム開発の品質・性能の維持と生産性向上ができる開発実行環境の提供

〇フレームワークによりアプリケーション開発手法と構造の標準化

２．３　あるべきシステム構成

この５つのシステム要件と「WEB アロー構築」の経験から、あるべきシステム構成をより具体的に構想した。

（１） 複数台稼働させた AP サーバに処理を分散させて、信頼と処理能力の向上を図る。

（２） 複数台の AP サーバに可能な限り均等にリクエストを割り振る機構が必須である。

（３） 複数台の AP サーバ、そのサーバ内のプロセスが常に稼働している保証はない。稼働している AP サーバ、プロセスへのみ適切にリクエストを再送できる機能により、高い負荷分散性能が実現できる。

（４） 従来のようにシステム単位や業務毎に専用サーバを立ち上げることは禁止する。複数のシステムや業務を１台に収容する「汎用的な AP サーバ」を数多く並列稼働させる構成により、AP サーバの統合を維持・推進する。

（５） 「AP サーバの汎用化」によりサーバ性能を 100％引き出し、可用性・拡張性を高めるだけでなく、アプリケーションの安全な保守・柔軟な運用まで実現する。

（６） 「複数台の汎用 AP サーバ」を前提にすると、使用するミドルウェアのライセンスおよび保守費用はサーバ台数に比例し増大してしまう。従ってミドルウェアを選択する時には、求める性能・安全性・費用や障害対策等を考慮・吟味した上で、積極的にオープンソースソフトウェア（OSS）を適用する。

以上の６つのポイントより、汎用化した複数台の AP サーバ群を高性能かつ柔軟に制御できるフロントエンドシステム「オープン FEP」の必要性が見えてきた。

2．4　「オープン FEP」自社開発の決断

「オープン FEP」が担うべき役割・機能をどう考えたのか。最も重要な機能は「高負荷への耐久性能を上げるためにトランザクション制御を非同期化」し実装することである。

（1） クライアントからのリクエストを集中的に受け取りキューへ書込む
（2） 各 AP サーバのプロセス稼働状況まで理解し処理を振り分ける
（3） AP サーバからの回答をキュー制御

この機能は既製品のロードバランサ導入では解決できない。通常のロードバランサはネットワーク機器の一部として設置する単なる負荷分散装置である。しかし我々が求めたのは、AP サーバのプロセスまで理解し振り分けることで、最適なアプリケーションのサーバ配置を実現できるフロントエンドシステムである（図３）。

また「オープン FEP」が担うべき主な機能として、その位置付けが「リクエストを集中的に受ける」立場であることから、シングルサインオン（以下、SSO）を実現する「ユーザ認証機能」、およびセキュリティと実績分析のための「稼働ログ取得機能」がある。いずれも複数の AP サーバ群分を一括して引き受けるため、負荷集中への耐久性能は最高レベルが要求される。

こうして求める機能・性能と必要性を踏まえ自社製リバースプロキシ「オープン FEP」開発に踏み切った。この決断を後押しする背景としては、メインフレームでの「フロントエンドプロセッサ（FEP）」構築・運用の実績と、1995 年以降に繰り返し実施してきたオープン化への試行錯誤の中で得た知識・スキル・経験があった。

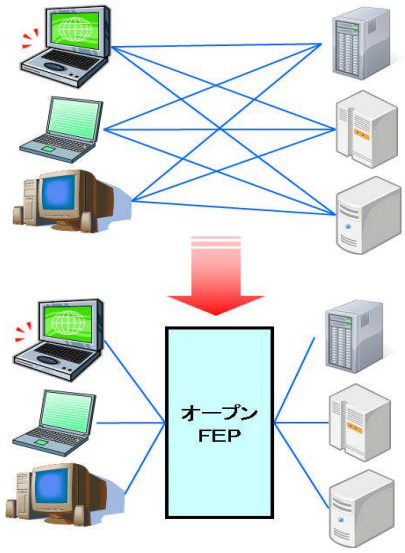

図 3　概念図

3．「オープン FEP」への期待

これまで「オープン FEP」構想に至る経緯とシステム要件から描いたシステム構成を、開発サイドの視点で述べてきた。この章では視点を変え、システムのオーナでありユーザである P 電工の IT 統制部門、この WEB 基盤を使いアプリケーションを開発する部門、そして出来上がったシステムを運用する部門の各視点で WEB 基盤に期待していた要望を列挙する。この要望は構築時にシステム要件となっただけでなく、「オープン FEP」の自社開発を決断するための有力な材料であった。

3．1　IT 統制部門（ユーザ）からの要求（図４）

■ 統制されたユーザ ID、パスワードで認証の一元化を実現したい

個別に WEB システムが開発された結果「独自のユーザ ID 管理によるユーザ認証機構」となり、各サイトへのリンクを集約してもそれぞれにログインを要求される状況を解消したい。今まで認証をかけていなかった WEB サイトも SSO に対応できること。画像等のコンテンツも設置場所次第で認証の対象にする。そのために必要となる全社

規模のユーザ ID 統制は別プロジェクトで推進する。

■ 全社のアクセス統計を取得できる

ユーザがどのシステムをどの程度利用しているか、より詳細な情報を取得したい。ユーザ ID と共にアクセス統計が取得できれば、部署、部門情報を付与して全体的なアクセス統計を分析することが可能になり、今後の方針決定の指標として活用できる。

従来、個別に構築された WEB システムからアクセス統計を収集するには非常に手間がかかる上に、WEB サーバのアクセスログだけではユーザ ID まで特定できない場合が多く、指標として活用しづらい。

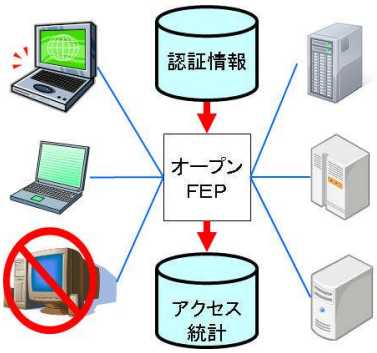

図 4　ユーザ要件

■ 規定回数以上パスワードを間違えたユーザをブロックする

セキュリティの統制意識を高める必要がある。単にユーザ ID とパスワードが一致しているかをチェックするだけでなく、何度もパスワードを間違えるユーザはログインできないようにブロックし、セキュリティ強化を図ることが必要である。

WEB システム、WEB サイト別にセキュリティ統制を実施するのは困難であり対応工数も増大する。一方、老朽化し改造不可能なシステムも存在する。これらのシステムも含め「オープン FEP」経由の接続に変更することで、ブロック機能の利用を可能にしたい。

3．2　アプリケーション開発部門からの要求（図５）

■ システム別にログインページをデザインしたい

ログインページは業務サービスの顔であり、その印象は非常に大切である。よってそれぞれのデザインにはこだわりたい。認証を一元化するからといって「オープン FEP」が表示する固定のログイン画面では納得できない。

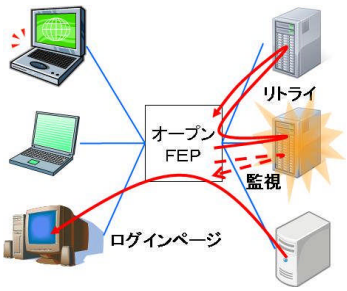

図 5　アプリ要件

■ ユーザへのエラー応答を軽減する

実際に稼働するアプリケーションは静的なコンテンツではないため、タイミングによっては回復可能なエラーを発生させる場合がある。「オープン FEP」のロードバランス機能はサーバまたはプロセスの死活監視をするので、同一プロセスが稼働中の他サーバでそのリクエストを再処理できれば、ユーザへのエラー応答を軽減できる。

■ アプリサーバの性能を 100%活用したい

アプリケーションサーバの処理能力を、ユーザからのリクエストを処理するためだけにフル活用したい。一般的なロードバランサは特定の URL へ一定間隔で監視用の接続を実行するためアプリケーションサーバにとって負荷増となる。しかしアプリケーションサーバは障害等で停止している割合のほうが圧倒的に少なく、一定間隔での常時監視は無駄な負荷を生む。リクエストに対する異常の検知から復旧までの時間を死活監視すれば、アプリケーションサーバの負荷を削減できる。

３．３　システム運用部門からの要求（図６）

■ アプリケーション部門で容易に利用可能であること

従来から使用しているロードバランサはネットワーク部門で管理されている。設定変更は構成変更作業として位置づけられ、アプリケーション部門の望む対応速度での作業実施が困難である。稼働状態の確認だけでもネットワーク部門へ都度問い合わせる必要がある。つまり各部門で余計な手間とタイムラグが発生している。

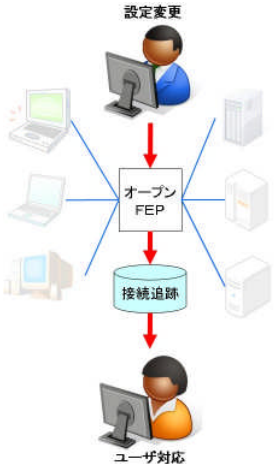

図 6　運用要件

■ ユーザからのアクセス状況を追跡できること

過去数世代の接続履歴を保持しておき、ユーザからの問い合わせに対応できるようにする。「いつまで、どのシステムにアクセスしていたのか」という事実関係を過去に逆昇り確認した上での電話対応を可能にするには、サポート担当者に専用の接続履歴照会画面が必要である。ユーザから調査用情報を聞き取ることにも限界があり、特に「使えない」「画面がでない」等の初歩的な問い合わせについては、システム側で事実確認ができるようにしてほしい。

■ 少ない台数でリクエストを中継できること

「オープン FEP」のサーバ台数を抑えることで SSL 証明書の必要数を削減でき、ランニングコスト抑制だけでなく、運用の省力化、グリーン化にもつながる。

また、アプリケーションサーバは負荷とダウンへの対策として規模の拡大にあわせた台数追加が必要であるが、「オープン FEP」は最小限の構成にとどめ稼働情報の分散化を防ぐべきである。故に「オープン FEP」の軽量化と使用リソースの軽減は徹底してほしい。

４．「オープン FEP」を核としたシステム構成

様々な要件とメインフレーム並の性能・安定性・運用性を求め設計されたシステム構成であるが、具体的には「オープン FEP」の処理構造と、それを核に組み込まれたシステム構成について述べる。最大のポイントはリクエストを一手に受ける「オープン FEP」への負荷集中対策である。

４．１　「オープン FEP」の処理構造

「オープン FEP」は Apache＋Tomcat 上で稼働する JAVA プログラムである。保守担当者が変わっても継続的にメンテナンス可能とするために JAVA 言語で実装することにした。初期モデルではサーブレットとして開発したが、高負荷なトラフィックに耐え切れないため、マルチスレッドで動作するサーバプログラムに構造を変更せざるを得なかった。開発言語は JAVA のままであり、処理性能を必要としない管理プログラムはサーブレットである。内部構造は処理特性によってブロック化し、それぞれに特化した処理のみを実行する複数のスレッドから成り立っている（図７）。

(1) リスナー

クライアントの接続要求を受信する。中継処理は一切行わない。

(2) メイン

リクエストの状態を判断し、適切なスレッドにリクエストを振り分ける。

(3) ログイン

ユーザログイン用のページを表示する。ログインページの表示はページデザインをアプリケーションサーバから取得する場合がある。アプリケーションサーバのレスポンスが全体に影響しないように分離した。

(4) 認証

ユーザ ID、パスワードのチェックには LDAP にアクセスする必要がある。高負荷時に LDAP のアクセス待ちで全体がスローダウンしないように分離した。

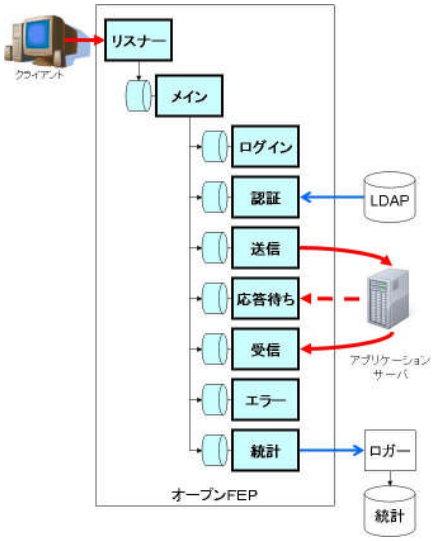

図 7　処理構造

(5) 送信

リクエストをアプリケーションサーバへ送信する場合、コネクションタイムアウトが発生する可能性がある。接続待ちが全体に影響しないように分離した。

(6) 応答待ち

リバースプロキシで一番影響を受ける箇所がアプリケーションサーバからのレスポンス待ちである。レスポンスをブロックせずに受信し、独自にタイムアウト判定を行うことで、アプリケーションサーバの所要時間に影響を受けない構造にした。応答速度はサービスごとに特性があるため、応答の速いものはできるだけ速く、遅いものは遅くてもいいように、段階的に応答待ちスレッドを準備した。

(7) 受信

レスポンスを受信しクライアントへ送信するスレッドである。一般的にリクエストよりもレスポンスの方がデータサイズが大きいため、データのコピーにも若干の時間を要する。

(8) エラー

エラー発生時にはエラーログの取得、ページの組み立てが必要になる。サービスとしては成立しないため処理優先度を低く設定した。

(9) 統計

アクセス統計はユーザに何もサービスを提供する必要がない。よって収集項目が揃った段階で非同期に取得する。システムが過負荷に陥った場合は統計データを破棄する回避ロジックを実装した。

４．２　システム構成

次に「オープン FEP」を核としたシステム構成を紹介する（図８）。まずリスクと負荷の分散、セキュリティに配慮し、「オープン FEP」はイントラネット用とインターネット用とを別に設置した。インターネット用の「オープン FEP」のみを DMZ（非武装地帯）に配置した。これによりアプリケーションサーバ、DB サーバは全て内部ネットワークに収容し共有することができた。

認証用データはオープン LDAP に格納し、マスター・スレーブ構成をとり負荷対策とした。パスワード変更対応においてはマスター・スレーブ間の同期処理は必要である。

アクセス統計やログメッセージの蓄積は、負荷集中に耐え得るよう外部に専用のサーバを準備した。統計情報の編集は基本的にバッチ処理で行っている。

システム全体の負荷状況を監視できる監視用プロセスは、アプリケーション状態の監視も兼ね専用のサーバとし負荷集中への対策とした。

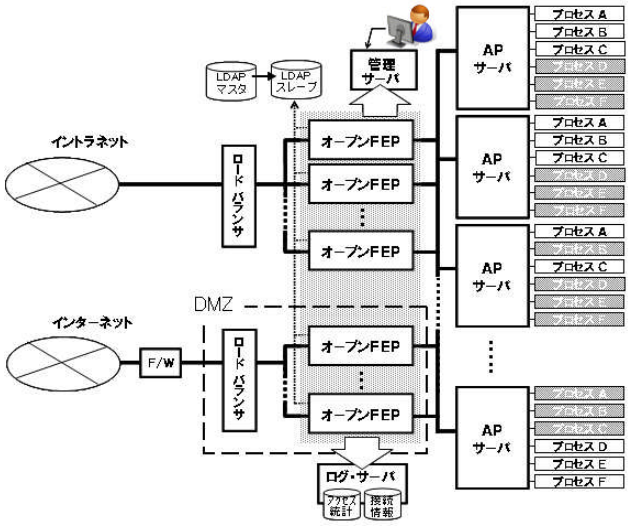

図 8　システム構成

このシステム構成を構想・決断するに際しては、ちょうど試行し始め手応えを感じていた「ブレードサーバ」と「仮想化」という２つの技術の存在が大きかった。ハード障害はブレードサーバのもつフェイルオーバー機能と「オープン FEP」のもつ死活監視機能で対応した。このことにより投資と運用工数がかかるクラスタ化やスタンバイシステムの準備は不要となった。数多く導入するサーバには、求める性能により実機でなく仮想化を採用した。AP サーバにセットアップするミドルウェアを含むモジュール群は全て同じ構成として、どのサーバでどの業務を稼働させるのかは「オープン FEP」のきめ細かいバランサ機能で制御する方式とした。これによりシンプルかつ柔軟な運用性と拡張性を得て可用性を向上させることができた。

こうしてメインフレーム並の性能・安定性・運用性をもつシステム構成・WEB 基盤が構築できたが、安定稼働までには処理方式の見直し、様々なチューニングを要した。

５．安定稼働までの道程

「オープン FEP」を核としたシステム構成は、約１年という短期で初版を構築し稼働させることができた。そしてこの WEB 基盤を使った新しい業務システムが開発され本番稼働し始めると利用するユーザも徐々に増えていった。構築着手から約２年後の 2008 年９月からはユーザ拡大と利用推進の施策が打たれ、一挙に「オープン FEP」への負荷集中度合いが高まりだした。構築時より過負荷対策を取っていたが、安定稼働までには予想外の障害が発生し、様々なロジック見直し、チューニングの必要に迫られた。

５．１　アクセス数増加でオープン FEP 増大の危機！

(1) ハングアップ

プロジェクト後半の 2008 年９月末よりユーザテストを開始。60～70 万件/日だったアクセス数が 150～160 万件/日に急増。「オープン FEP」は稼働３台、予備２台の計５台を準備。午前中に過去最高の９万件/時アクセスを突破したところで「オープン FEP」がハングアップした。予備の２台を投入、計５台稼働でなんとかその日を乗り切ることができた。

(2) 逃げ切れ

利用ユーザが想定数まで達していないにも関わらず、５台の「オープン FEP」ではこれ以上の負荷に耐えられないと判断し、急遽５台を増強し計 10 台構成でなんとかユーザテストの負荷に耐えた。

しかし翌 10 月からはトライアルユーザへの展開を予定していた。想定ユーザ数はテスト時の３倍のため、やむを得ずさらに 20 台の「オープン FEP」を増強し計 30 台構成とした。それでも稼働統計を取得する余裕はなく、ログ関連処理を全て停止させた基本機能のみに制限した稼働とした。

(3) どうする？

ユーザ数に応じてサーバ数を追加すればなんとか負荷には耐えたが、このままでは運用工数・費用がかかり過ぎる。これはあくまで緊急処置であり早急に収束させねばならない。どうする？

「オープン FEP」の処理性能を引き上げるための内部構造見直しが必須と考え、高負荷テストを繰り返し構造上の問題抽出に専念した。その結果、アプリケーションサーバからの応答が遅延した場合に「オープン FEP」自身のスレッド数が増殖しリソースを圧迫、処理能力が著しく低下することが判明した。

そこでスレッドの処理効率をあげるために基本構造から再設計するという修正案が浮かんだ。サーブレットの制御を離れ独自にマルチスレッドを管理するロジックを組み込むのである。しかし今までかけてきた工数と根本的なロジック修正であることにプロジェクト内で賛否が分かれた。徐々に増える負荷と迫るユーザ展開の期日に幾度の協議の結果、プロジェクト内で修正実施を決断した。修正版「オープン FEP」としてリニューアルできたのはユーザテストから３ヶ月、2008 年も暮れようとしていた。

(4) ひと安心

修正版「オープン FEP」の適用により、稼働３台という当初の台数に戻しても安定稼働できるようになった。ピーク時のアクセス数も 2008 年 12 月の 280 万/日から順調に伸長し、2009 年 10 月には 780 万/日まで増加した。

５．２　アップロード、ダウンロードのファイルサイズが・・・

(1) 何が滞留しているのか？

2009 年３月、個別に稼働していた見積もりシステムを「オープン FEP」経由のサービスに切り替えた。過負荷対策で内部構造を見直したにも関わらず処理の滞留が発生した。一旦予備サーバを見積もり専用サーバとして立て回避した。

原因究明のため滞留データを解析すると、アップロード、ダウンロードのデータ転送に時間がかかっていることが判明。1 メガバイト程度で数秒滞留していた。数キロバイトから数 10 キロバイトを想定して設計した送受信スレッドでは、設計どおりの滞留状況であったが、この滞留は他業務のレスポンスまで劣化させるため緊急対策が必要であった。

(2) どうする？

解決策としては、アップロード、ダウンロードを専用スレッドとして独立させるという設計方針をすぐに決めた。しかし実装時の判断基準がなかなか決まらなかった。アップロードリクエストとダウンロードレスポンスを確実に見分ける方法に戸惑った。今思えば確実さを追求するあまり、堂々巡りに陥っていたのかもしれない。

結果的にアップロードはデータの長さ、ダウンロードはデータの種類を基準に判断することになった。アップロード、ダウンロードは利用頻度と優先度の違いから、主処理の送受信スレッドよりも少ない数で稼働させた。一度にたくさんの転送が発生しても負荷がかかりすぎないよう運用面を優先した判断である。

５．３　収容システムの増加で統計出力が間に合わず！

(1) バッファ排他がシステム全体をロック

2009 年９月「オープン FEP」がダウンした。アクセス統計を書き出すバッファにロックがかかり、要求量にスループットが追いつけない状況となった。

処理速度の向上を狙い、一旦統計用バッファに書き出した後に非同期でアクセスログを外部に転送する構造だった。にも関わらず「オープン FEP」配下に収容するシステム数の増加によりバッファへの書き込み頻度があがり、ログ転送処理が追いつかなくなったのである。万一にでもログ転送が停滞しメモリ圧迫することを防ぐために「バッファ内のログを破棄する」という回避ロジックが動作したのだが、一定量のバッファを一括破棄するのに数秒間バッファをロックしてしまうため、主処理に数秒間の排他が発生。その結果スループット低下を招いた。

(2) どうする？

バッファの排他範囲とオーバーフロー時の破棄の処理に問題があることはわかった。バッファをスレッド単位に分割し他スレッドと競合しない構造に変更し、排他範囲の局所化を狙った。開発用サーバがハングアップする負荷をかけ再現確認を実施した。排他の衝突は殆ど見られず、ロック回避に成功したように見えた。

しかし連続稼働テストを実施していくと徐々にメモリを圧迫していくことがわかった。スレッド単位に分割したバッファがそれぞれ領域を獲得し始めると、総量として従来よりも多くの領域を使用してしまうのである。スレッドの稼働率は一定ではないため、余分となったバッファ領域を一定量まで縮小させる対策をとり、ようやくメモリ圧迫を回避できた。

６．導入効果、そして二次効果

「オープン FEP」を自社開発し、それを核としたシステム構成、WEB 基盤を構築できたことにより、「全ては WEB ポータルから」の実現、約４，０００台の市販エミュレー

タの廃止、ユーザ裾野の拡大など当初より想定していた成果をだすことができた。その他に想定していなかった効果、および二次効果を得ることができたことは、まさに「オープン FEP」効果である。

６．１　アクセス統計を分析、今後の方針を立案

「オープン FEP」が取得する全システムのアクセス統計は、毎月 IT 統括部門に送られる。IT 統括部門ではユーザ別、システム別等アクセス状況を多角的に分析し、利用推進の啓蒙、不要システム見直し等の方針立案に役立てられている。想定していた以上に詳細な情報が「オープン FEP」配下の全システムを対象に収集できる。科学的に適確な対策が打てるようになったと好評である。1 日分の全アクセスを業務別、利用者グループ別、サーバ別に処理量をサマリーした分析例を図９に示す。

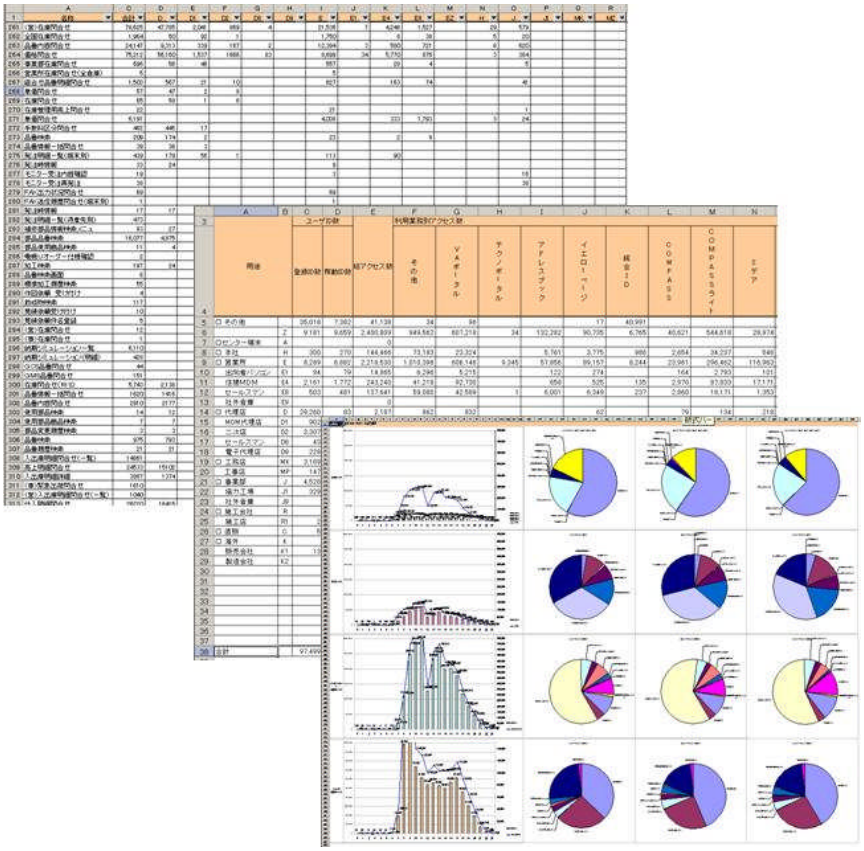

図 9　アクセス統計分析例

６．２　利用者別に接続元クライアント PC を限定

特定のユーザ ID には利用できるクライアントを制限することにした。これは基幹系業務を WEB 化するにあたり必要なセキュリティ要望で、関連会社のユーザ ID はいろいろな場所からログインできないように制限を設けるというものだった。ユーザ ID の取得に関してはルールを設けることができるが、その後の運用に関しては徹底し追跡することが困難である。よって「社外用ユーザ ID は特定のクライアント PC でしか利用できない」という制限が必須となった。

対処としては「オープン FEP」での認証時にクライアント PC を特定し、同じ PC からの接続しか受け付けない機構を実装するだけで実現できた。PC のクラッシュや交換時にはそのままでは接続できなくなり、ユーザ、ユーザサポート部門共に手間がかかってし

まうが、セキュリティ面を優先するとの判断からロック機構を搭載した。

６．３　パスワードに有効期限を持たせる

利用可能なユーザ ID であっても一定期間(30 日間程度)を経過した場合は強制的にパスワードを変更させる（図 10）というセキュリティ強化を行った。

「オープン FEP」は関連する全ての WEB システムを中継している。そのため全社的なセキュリティ統制を実施する場合でも個別にシステムを修正する必要は無い。新しいセキュリティルールを「オープン FEP」に搭載するだけで全システムを同じ制限にあわせることができる。ホームページ等の静的なコンテンツを持っているだけのサーバも「オープン FEP」経由にするだけで対応することができた。

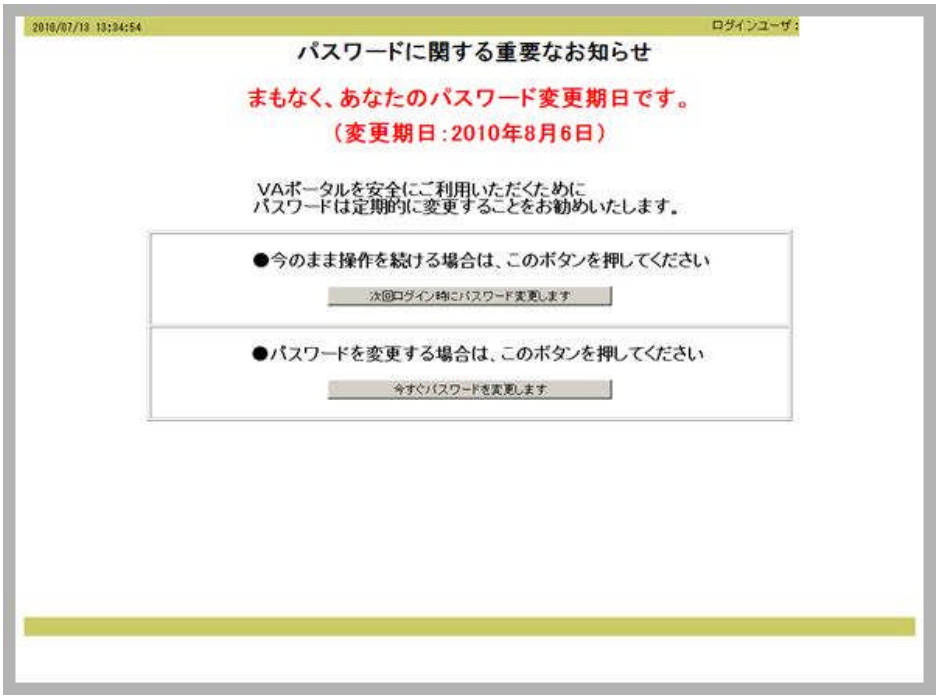

図 10　パスワード変更画面

６．４　サービス時間を制御できる

サービス提供時間も「オープン FEP」で制御でき、ユーザへのサービス提供時間外はシステム部門のメンテナンス時間とした。システムメンテナンス時には動作確認のためにオープンテストが必要である。「オープン FEP」がリクエストを中継しなければ、システムは稼働状態でもユーザからはサービスを利用できないようにできる。

また日々のバックアップや週末の定期処理に対しても、スケジュールを「オープン FEP」に設定しておけばサービス利用不可のページが表示される。これでメンテナンス中ページへの差し替えやロードバランサの設定変更等の余計な作業が不要となった。

６．５　DMZ には「オープン FEP」だけを設置しておけばよい

インターネット・ユーザからのリクエストを受け付ける「オープン FEP」だけを DMZ に設置すれば、他のサーバはイントラネット・ユーザと共用できる。アプリケーションサーバは「オープン FEP」からしか接続されず、もちろん DB サーバも DMZ からアクセスされる必要はない。F/W を限定的に設定できセキュリティを向上させ、かつシンプルな

システム構成となる。

ネットワーク部門にもアプリケーションごとに DMZ 内の構成を悩む必要がなくなり、構成管理が簡素化できたと好評である。

６．６　ポータルシステムダウン時に代替ページを表示

企業内ポータルが全ユーザのトップページとして設定されており、様々な業務へのリンクも設定されている。反面、ポータルがシステム障害でダウンすると必要な業務へのリンクがたどれずユーザが混乱することになる。

ポータルを担う AP サーバも「オープン FEP」配下であるため、「オープン FEP」はポータルの全障害を検出することができ、障害検出時には主要な業務システムへのリンクを持った代替ページを表示させることができる（図 11）。ポータル全障害は滅多に発生するものではないが、発生時にユーザの混乱を最小限に抑えることができる。

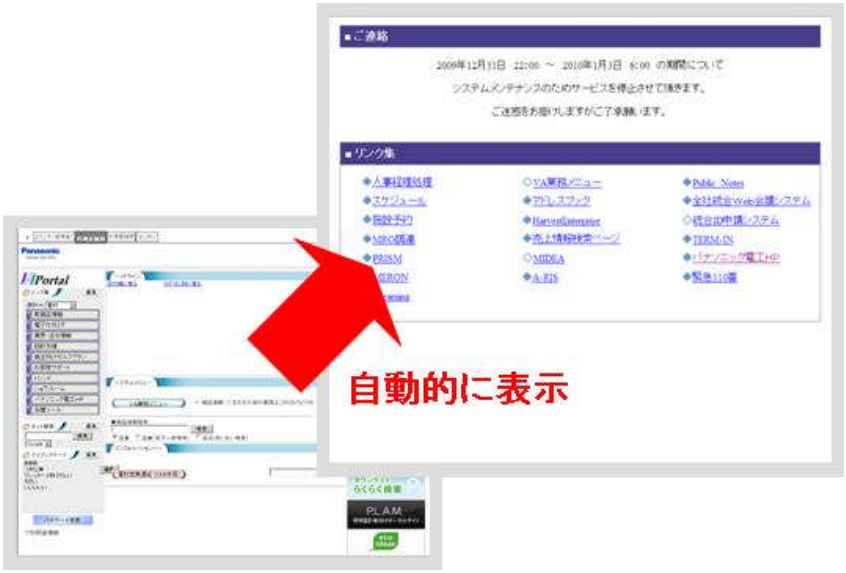

図 11　代替ページの表示例

６．７　サーバメンテナンス時に簡単構成切り替え

夏季休暇や年末年始などの長期連休中にサーバメンテナンスを行う時には、「オープン FEP」によりアプリケーションサーバの接続先を変更して対処することができる。

ブレードサーバの導入が進む中、長期連休はブレード筐体のメンテナンスが実施できる少ないチャンスである。1 筐体には数十のサーバが搭載されているので基本的には全システム停止が望ましいが、最低限の問い合わせ業務だけでもサービスを続けなければならないのが実情である。

事前にサーバ管理部門、アプリケーション部門と調整し縮小構成を準備する。作業前に「オープン FEP」の接続先設定を変更するだけで、メンテナンス期間中の稼働サーバを絞り込むことができる。またサービスから切り離されているため、本来のサーバ群はメンテナンス完了後そのままの環境でシステム稼働確認テストを実施できるというメリットも享受できる。

６．８　負荷状態を一括で監視

WEB システムは自由にリンクできるため、レスポンスの悪化を招いている箇所を特定

しにくいという特性がある。そこで「オープン FEP」のリクエスト処理状況を、リアルタイムに全プロセスを一元監視できるように専用の可視化画面（図 12）を準備した。

過負荷になりかかっている、或いはリクエストが滞留し始めているという予兆をいち早く捕まえることができる。システム全体が完全に停止する、もしくはユーザからの問い合わせが来る前に対応を開始することができるのである。障害発生時の影響時間を短縮することができるようになった。

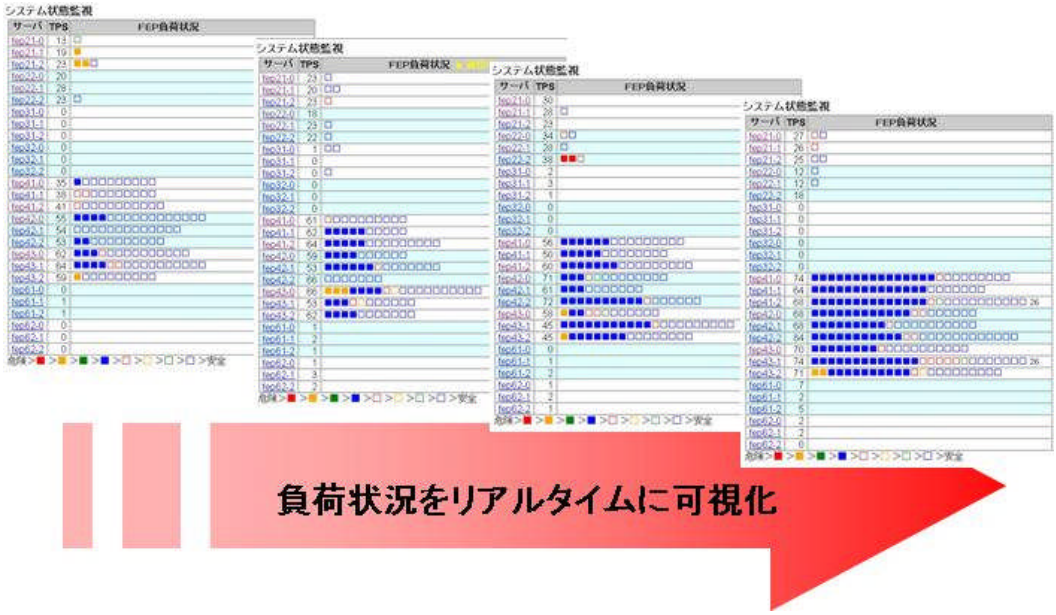

図 12　負荷状況の表示例

６．９　他認証システムとのシングルサインオン連携

個別にポータルサイトを導入しているユーザもシングルサインオンが可能になった。他ポータルサイトにサービスだけをリンクさせる場合、再ログインを要求されないようにする必要がある。「オープン FEP」が他サイトでの認証結果を連携できるようにし、既存の WEB システムも「オープン FEP」配下への統合が可能となった。

６．１０　OSS 適用

「オープン FEP」配下に複数台の汎用化させた AP サーバという構成のため、特に数を要する WEB サーバ、AP サーバ、DB、LDAP、「オープン FEP」に適用するミドルウェアには OSS を適用している。効果を製品価格に換算し概算すると、一時投資を約２億円、年間保守費を約４千万円削減できたことになる。この OSS 適用は構築時だけでなく、今後必要となるシステム構成拡張を考えても必要な決断であった。またこれ以外にも統合開発ツール、リソース管理、ログ監視ツールにも OSS を適用している。

７．今後の展開

７．１　システム間通信への適用

ブラウザからのリクエストだけでなく、システム間通信も「オープン FEP」経由にすることで以下のメリットを狙う。

○ 回復可能エラー時にリトライできる

○ 接続先変更が柔軟に行える
○ 負荷状況が可視化できる

SOA で構成された大規模システムはサービス間の連携が重要になる。レスポンス悪化や障害の切り分けにもサービス間の通信状況が可視化できる方が、障害調査だけでなく運用も容易になる。しかしながら、サービス連携はブラウザからのアクセスとは負荷や頻度等の特性が違うため「オープン FEP」に機能拡張が必要である。

段階的にサービスの接続経路を「オープン FEP」経由に変更していくことを検討中である。負荷の集中と障害発生時の影響局所化を考慮し、専用の「オープン FEP」を稼働させる必要がある。

７．２　信頼性の向上

リバースプロキシであるため全リクエストは必ず「オープン FEP」を経由してサービスされている。システム構成上シングルポイントとなり障害時に影響を与える範囲が大きい。そのためハード障害に備え最低でも２台構成に、ネットワーク機器は二重化し、負荷状況は常時監視を実施している。

スレーブ用 LDAP サーバは複数台稼働させているが、１台の「オープン FEP」で使用できる LDAP サーバは１台しか設定できない。そこで LDAP サーバ障害時に動的に接続先を変更できる機能を搭載する予定である。複数台の「オープン FEP」が稼働中の LDAP サーバに集中するとシステムダウンを引き起こす可能性がある。LDAP の接続先変更に際しては、複数台から処理負荷集中しないように制御する方策を検討する必要がある。

一般的な対策を講じているが危険性は変わらない。システム全体から見てより高い安定性を確保できる方策を見つけ出さなければならないのが今後の課題である。

７．３　管理機能の拡充

「オープン FEP」の設定には XML 形式ファイルを使用している。拡張性を見込んで採用した XML 形式だが、機能拡張と共に文法が複雑化してしまっている。

にも関わらず現時点で設定画面は存在していない。XML ファイルを直接編集しており運用上の危険がある。設定変更は慣れた担当者のみが実施することで運用回避しているが、あくまで回避策である。

従って設定変更画面を準備するのが急務である。バックアップと変更履歴を自動取得できる機能を実装し、かつ設定変更に起因する障害にすぐに対応できる運用環境を確立することが肝要である。

７．４　最後に

前述したとおり、2006 年 7 月の「WEB&DIGITAL にフォーカス」宣言から約１年後には「オープン FEP」を核とした WEB 基盤およびシステム構成が立ち上がり始めた。その後の約２年間は安定稼働を最優先にしながら性能の引き上げに注力し続けた。まさに走りながらの基盤構築・チューニングではあったが、それを支えたのは長年培ってきたメインフレームを使いこなす実績と技術力を裏付けに、「近い将来に来る」と判断し 1997 年から取り組み続けてきたオープン化への試行錯誤である。その成果を「オープン FEP」構築に集約することができた。

また、「オープン FEP」構想を考えるタイミングが「ブレードサーバ」と「仮想化」

という技術の出現とマッチしたことは、不思議なめぐり合わせである。できあがったWEB 基盤、システム構成を眺め直してみると、奇しくも最近話題であるクラウド構築の基礎となる「標準化」と「仮想化」を取り込んだシステム基盤である。「オープンFEP」の性能が出なかった時に、日々その台数を増強し 30 台にまでなってしまったことは５章の１節でご紹介した。増強せざるを得なかったことは不本意であったが、１日で10 台、20 台のサーバを増強できたことこそ、システムの柔軟性と拡張性を証明するものである。

現在は「メインフレームとサーバのハイブリット型」の名のもと、特に重要かつ重い更新業務をメインフレームに残している。今後はそのマイグレーション手法について、試行錯誤しながら方向性を探っていかねばならない。

新たに出現する IT 技術をベースに、集約と分散を周期的にくりかえし進化していく情報システムの潮流の中、日々目まぐるしく変わり現れてくる IT 技術を見極め、当社企業スローガンである「コンピュータを意識させない情報システムの創造をめざして」を実践するために、「オープン FEP」をさらに磨き上げながら次のステージを目指したい。

最後に JAVA 修得から始まり「オープン FEP」構築に到るまで、その主旨を理解頂きご協力いただいた関係者の皆様に感謝いたします。

以　上