Linux と各種 OSS プロダクトを用いたシステム構築事例

System construction case using Linux and various OSS products

鮫 島 荘 介，中 村 彰 彦

要　約　本稿では，実際のシステム開発プロジェクトへの Linux®/OSS（オープンソースソフトウェア）ミドルウェアと OSS フレームワークの適用事例を紹介する．

まず，料金システムの概要とシステム構成を説明する．そのうえで，本システムに用いられている Linux/OSS ミドルウェアの採用理由を述べる．また，Java フレームワークとして，Struts/SpringFramework を採用することになった選定過程と利用にあたっての工夫を記述する．

最後に，1 年半の稼働実績を証跡とし，OSS がミッションクリティカルなシステムへの使用に充分耐えうることを提示する．

Abstract　This paper introduces the case examples of application of Linux/OSS middleware and the OSS framework to the actual system development project.

First of all, it explains the outline and the system configuration of the charge calculation system, and then discusses the reason for adoption of the Linux/OSS middleware used for this system, as well as the selection process in adopting Struts/SpringFramework as the Java framework and devices in using one.

Finally, it presents that OSS is good enough for mission-critical system use by evidence of one and a-half years of operation results.

1. は じ め に

矢野経済研究所による調査[1]によれば，2004 年から 2005 年にかけて，Linux を含めた OSS 導入率が 32％から 16.8 ポイント増加し，48.8％となった．Linux/OSS の導入が急速に進んでおり，市場規模が大きく伸びていることが伺える．このような状況を受け，ベンダ各社は Linux/OSS の適用範囲を，Web サーバや部門サーバを中心とした限られた領域から，エンタープライズ分野，さらにミッションクリティカル分野と，高い信頼性が求められる領域へと拡大すべくサービスを拡充している．

また Java 言語の開発においては，OSS のフレームワークを始めとするライブラリ製品が積極的に利用されている．この領域では，早くからエンタープライズ分野で適用され，ミッションクリティカル分野での利用も珍しいことではない．

このように拡大傾向にある Linux/OSS だが，Linux/OSS ミドルウェアについては採用を不安視する声もまだまだ多く，また OSS ライブラリ製品については，似て非なる製品が多数存在するため，開発するシステムに適した製品の選定に，知識やアプローチが必要な状況である．

本稿では，公共サービス企業 A 社への料金計算システム開発事例を通じ，適用した Linux/OSS ミドルウェアと OSS フレームワークについて，その選定過程と利用にあたっての工夫を紹介する．また実際の稼働実績の評価から，Linux/OSS の信頼性についても言及する．

2. 料金計算システムについて

本章では，料金計算システムの概要を記述する．

2.1 システムの要件

料金計算業務は，その名の通り金銭を扱う業務であること，多数の顧客分の情報を高速に処理する必要があることから，システムの完全性確保および，高速な業務処理が求められる．

これらに加え，経営戦略によって予想されうる計算方法の追加や，他社との差別化を図るために，システムがサービスの多様化に柔軟に対応できることも求められた．

また，料金計算業務は月初に集中する特性があるが，顧客への請求書発行業務は遅延の許されないミッションクリティカルな業務であり，システムには高い信頼性，可用性が必要となる．以上の要件を下記にまとめる．

- 料金計算結果，請求書の完全性
- 顧客数に対応できる業務処理性能
- 計算方法やサービスの多様化に伴うシステム改修の柔軟性
- ミッションクリティカル業務に伴う高信頼性・可用性

2.2 システム概要

システムの概要を図１に示す．

料金計算システムを大別すると，下記機能から構成される．

- 顧客・契約マスタデータ管理機能
- 使用量データ登録機能
- 料金計算機能
- 請求情報出力機能

A 社各担当の業務内容と上記機能を結び付けて料金計算システムの概要を記述する．営業担当者は日々の営業活動の一環として，システム上の顧客情報，契約情報を管理する（事前準備）．料金計算時は，まず，使用量データ担当者が使用量データをシステムに登録する（①）．

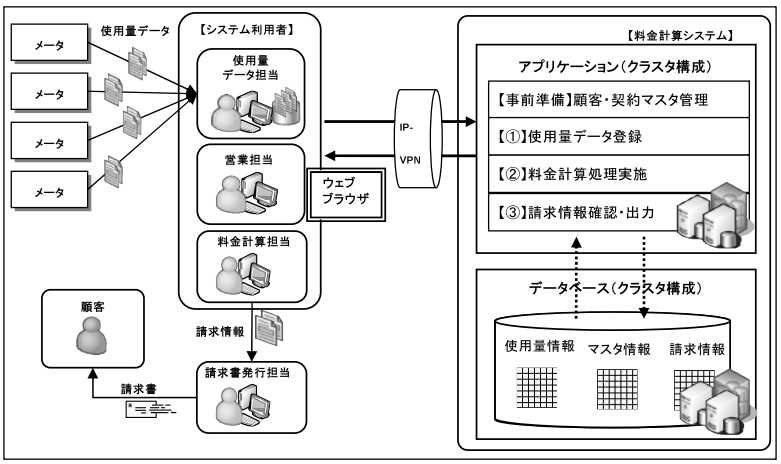


図１　料金計算システム概要

この後，料金計算担当者が料金計算機能を用いて請求情報を作成する（②）．この料金計算時に，営業担当者によって管理されている各マスタ情報が用いられることになる．請求情報作成後は，営業担当者がウェブブラウザを通して請求情報を確認後，料金計算担当者がシステムから請求情報を出力する（③）．この請求情報は，請求書発行担当者により別のシステムに投入され，作成された請求書が顧客に送付される．月次料金計算業務で作成した請求情報はシステムに蓄積され，営業担当者が履歴情報として確認可能である．

サーバ構成としては，DB（データベース）サーバ，AP（アプリケーション）サーバの両方に高信頼・可用性が求められるため，共にクラスタ構成としている．

3. Linux・OSS ミドルウェアの適用

本章では，料金計算システムのシステム構成と，このシステム構成を採用するに至った背景，理由を記述する．

3.1 システム構成

本システムの構成を図 2 に示す．本システムは，ウェブブラウザをユーザインタフェースとして利用するウェブアプリケーションである．サーバ構成としては，DB サーバと AP サーバから構成される．AP サーバの基本構成としては，OS（オペレーティングシステム）にレッドハット®社Enterprise Linux（Red Hat Enterprise Linux），アプリケーションサーバに Tomcat® を用いている．Tomcat は，サン社 Java™VM（Java バーチャルマシン）上で動作しており，これらの上で料金計算システムアプリケーションが動作する．DB サーバの基本構成としては，OS は同じく Red Hat Enterprise Linux，DBMS（データベース管理システム）はオラクル社 Oracle Database 10g™ を採用している．また，2 章の要件で述べたように，料金計算業務は月初に集中するミッションクリティカルな業務であるため，AP サーバ，DB サーバ共にハードウェアの冗長化を行い，アクティブ/スタンバイのクラスタリング化を実現している．

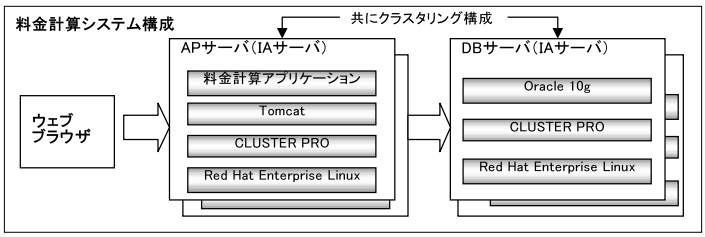


図 2　システム構成

3.2 選定理由

前節の通り，本システムでは，OS に Red Hat Enterprise Linux，アプリケーションサーバに OSS である Tomcat を採用している．Linux/OSS 構成を採用した理由は，下記を総合的に判断した結果である．

1）機能比較と導入コスト

一般的にオープンシステムを構築する際に検討対象となる OS は，Windows®，UNIX®，Linux である．まず，これらについて，アプリケーションを動作させるという基盤として十分かを確認した．選択する OS は，システム要件であるウェブアプリケーションの基盤となりうる事が機能面の条件となる．上記 OS は全て JavaVM が動作し，JavaEE（Java Platform,Enterprise Edition）の基盤として問題ない．加えて Windows については，IIS®（インターネットインフォメーションサービス）＋ASP.NET も選択肢として加わる．

次に，導入コストの視点で比較を行った．本システム構築の初期検討段階において，初期導入コストを抑えることも成功要因の一つとして挙げられていた．OS の選定は，ハードウェアの選定でもある．OS メーカによって独自開発した CPU アーキテクチャ上で動作するものは価格が高い傾向にあるため，UNIX＋UNIX メーカハードウェアの組み合わせは選択肢から外すこととし，Intel Architecture（IA）サーバ上で動作する OS から選定することとした．

ミドルウェアについても同様に価格面を考え，OSS 製品から選定することとした．

2）サポート体制

システム導入に際して，同時に検討しなければならないことに運用・保守フェーズがある．この運用・保守フェーズの重要な位置を占めるものがソフトウェアサポートである．

日本ユニシスグループにて，ワンストップサポートサービスを実現するために，各種 OS およびミドルウェアについて専属のサポート部門がある．

現在対応可能な OS については，Windows，UNIX，Linux があり，OSS アプリケーションミドルウェアについては Tomcat，JBoss®（JBoss Application Server）がある．

これらの構成でシステム構成を決定できれば，顧客のシステムに何か問題が発生した場合でも，日本ユニシスグループ内で商用ソフトウェアと同様に素早く責任を持って対応することが可能である．

3）技術者のスキル

開発メンバのスキル領域も検討した．顧客からの要件の範囲内で，開発ベンダ側体制が得意とするシステム構成にすることは，技術リスク軽減に伴う品質向上と開発期間・コストの圧縮に繋がり，双方にメリットがある．

OS については，開発メンバは UNIX，Linux を用いたシステム開発経験が多く，初期教育コストの視点から Linux を選択することが望まれた．

また，システム要件であるウェブアプリケーションを開発するにあたり，技術者のスキル領域を考え実装言語を Java とした．さらにここでミドルウェアに焦点を当てると，OSS のアプリケーションサーバは，サポート可能な Tomcat，JBOSS のどちらからかということになるが，Tomcat はウェブアプリケーション構築の初期研修や市販書籍において，採用されているケースが圧倒的に多く，技術者が慣れている Tomcat を採用する方が有利と考えた．

昨今の短期開発をにらむと，開発者の得意とする技術領域を選択することが重要である．技術者の教育コストの削減は，Linux/OSS ミドルウェアを選定する上での大きな理由の一つである．

4） 顧客内のシステム状況の分析

システム構成の設計段階で顧客が導入済みの他システム状況を分析する機会があった．

OSについては，メールシステムで一部Windowsを利用しているものの，業務に関するシステムではLinuxを採用しており，体制および運用面を考え，OSを統一させることがメリットであると判断した．

またアプリケーションサーバも，顧客内システムでTomcatを利用し，信頼性をはじめ悪い評価が無く，Tomcatを採用することが自然であると判断した．

5） Red Hat Enterprise Linuxの採用実績

上記を総合的に判断しLinuxを採用する判断に至ったが，数あるLinuxディストリビューションの中からRed Hat Enterprise Linuxを採用することになった最大の理由は実績である．世界No.1のLinuxディストリビューションであるRed Hat Enterprise Linuxは，日本国内においてもここ数年間のベンダ別出荷シェアが50%超となっており，採用実績の面から信頼性を高く評価した．なお，Linuxディストリビューションの市場動向は，インターネット等で調べることで容易に確認可能である．

3.3 クラスタリング構成について

料金計算システムは，請求書情報を作成するシステムであり金銭を扱うシステムである．業務的な完全性はもちろん，契約通りの日程で顧客に請求書を送付する必要があり，システムへの信頼性の要求は高く，高可用性が求められる．

この要求を満たすために，APサーバ，DBサーバ共にクラスタ構成を実現している．クラスタリングソフトには商用ソフトであるNEC社製Cluster Pro®を採用した．システム構成における他の部分がOSSを採用している中で，クラスタリングソフトのみが商用であるのは，実績を含めた信頼性を優先したからである．クラスタリングソフトは，障害発生時の回復のためのソフトであるので，他のソフトに比べ特に信頼性が求められる．実績のあるOSSのクラスタリングソフトが無いこともあり，LinuxやTomcatとの稼働実績が豊富であるCluster Proの採用に至った．初期コストを抑えるという命題だけを見て，闇雲にOSSを採用することが良いわけではなく，適材適用の考えを持つことが非常に重要である．なお，データベースソフトとして，Oracle Database 10gを採用している理由も同様に信頼性を考慮してのことである．

本システムにおいて，クラスタリングソフトが監視している対象を記述する．APサーバにおいては，料金計算アプリケーション，Tomcat，パブリックLANを監視している．一方，DBサーバにおいては，Oracle，パブリックLANを監視している．これらに異常が発生すると，システムとして機能しなくなり，業務に著しい影響が発生する．クラスタリングソフトは，これらの監視対象に異常を検知すると，待機系に系切替（フェイルオーバ）を行う．

4. OSSフレームワーク，ライブラリの適用

Java言語，特にJavaEEを利用した企業向けシステム開発において，開発生産性および品質向上を目的としてフレームワーク製品を利用する事が一般的である．フレームワーク製品には，デファクトスタンダードと言えるOSS製品もあり，多くのミッションクリティカルシス

テムに採用され，また商用フレームワーク製品の内部に取り込まれている．

このような状況の中で，Java 開発を行うアーキテクトの関心は，OSS を利用するかどうかではなく，プロジェクトの特性を理解した上で，OSS を含めた多くの製品の中からどの製品を選択し，いかに工夫を凝らした適用をするかに力点が置かれていると考える．

本章では，料金計算システムでのケースを紹介する．

4.1　システム構成

本システムで利用した，フレームワーク及びライブラリの構成を図 3 と表 1 に示す．

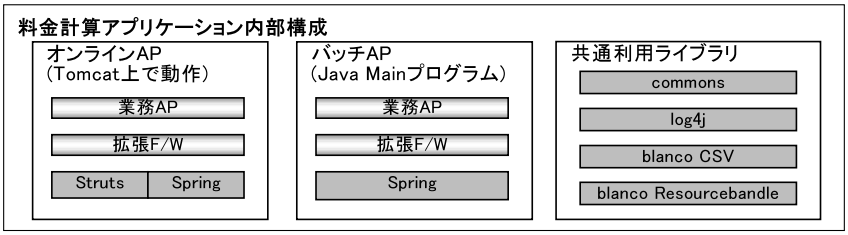


図 3　ソフトウェア構成

表 1　システムを構成するソフトウェア

ソフトウェア名	ソフトウェア詳細
Struts	OSS のフレームワーク．プレゼンテーションの実装をサポートする．
Spring	OSS のフレームワーク．DI（Dependency Injection），AOP（Aspect Oriented Program）を実現する．またプレゼンテーションやデータベースアクセスをサポートする．
Commons	OSS の便利なライブラリ群．複数のコンポーネントから構成されている．本システムでは，logging，BeanUtils コンポーネントを明示的に利用している．
log4j	OSS のログ出力ライブラリ．
blanco CSV	OSS．Excel 成果物からソースコードを自動生成する blanco Framework のサブパッケージ．Excel に CSV ファイルの入出力情報を記述すると，CSV ファイルの読み込み/書き込み処理を行うソースを自動生成する．
blancoResourceBundle	OSS．blanco Framework のサブパッケージで，外部定義ファイルから値を読み取り，プログラム中で扱うためのソースを自動生成する．

4.2　フレームワーク製品の選定について

本システムではフレーワーク以外にも多くの OSS プロダクトを選定している．本節では，システムのアーキテクチャに大きな影響を与えるフレームワーク製品の選定について，本プロジェクトで適用したアプローチを述べる．

4.2.1　フレームワーク製品選択の考え方

フレームワーク製品の選択にあたって，図 4 に示すように，各種フレームワークが守備範囲とするレイヤを定義し，それぞれについてプロジェクトとして求める要件を表 2 に整理した．レイヤは，プレゼンテーション，サービス，データアクセスの三つに分類した．

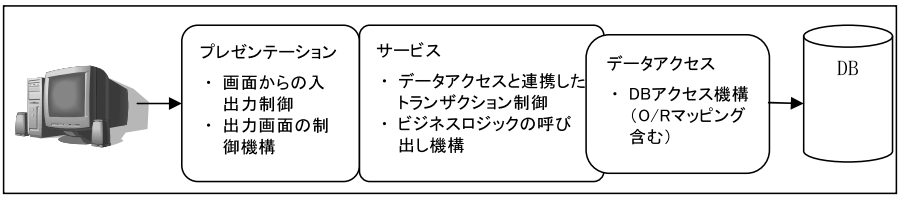


図4　レイヤ定義

表2　レイヤ毎に定義した要件

レイヤ	要件
プレゼンテーション	・プレゼンテーションの実装は非常に手間がかかるため，技術習得コストが特に小さいもの(重要)
サービス	・技術習得コストが小さいもの ・暗黙的トランザクションの制御ができること(重要) ・テスト効率やバッチでの利用を考え，コンテナの動作に JavaEE 環境が不要なもの
データアクセス	・技術習得コストが小さいもの ・サービスコンテナと連携したトランザクション制御が行えること ・スキーマからアクセス処理の自動生成ができること

4.2.2　製品の選択肢と選択理由

表3に示すように，レイヤ毎に具体的な選択肢を挙げ，要件を判断基準として製品の選択を行った．選択にあたっては，他のレイヤへの影響が大きいサービスレイヤから検討を行っている．

選択肢に挙げたプロダクトは，期待する機能を持つことを大前提とし，かつ，一定レベルの信頼性があると判断したものである．

採用にあたっては，利用経験も含めた実績と，ツールのサポートなども含めた情報量の多さを判断基準としている．

表3　レイヤ毎の候補製品と判断結果

レイヤ	選択肢	採用，不採用理由
サービス	・Spring(採用)	・JUnit やバッチプログラムからも利用できる(○) ・日本語書籍が2冊存在した(◎)
	・Seaser2	・JUnit やバッチプログラムからも利用できる(○) ・検討当時(2005/9)，書籍が発売されていなかった(×)
プレゼンテーション	・Struts(採用)	・開発メンバ(パートナー企業含む)に Struts の利用経験者が多数存在した(◎) ・Struts は書籍が豊富(◎) ・“SSH”(Struts/Spring/Hibernate) という組み合わせは世界的に実績が多い(○)
	・Spring WebMVC	・Spring の本にも，WebMVC の記述は少なかった(×)
データアクセス	・Hibernate	・Hibernate はデータアクセス部品の自動生成可能であった(○) ・評価の高い日本語書籍が無かった(×)
	・iBATIS	・書籍が無い．また全体的に情報が少ない(×) ・検討当時(2005/9)自動生成ツールが存在しなかった(×)
	・Spring JDBC Template(採用)	・データアクセス部品の自動生成の仕組みはない(×) ・JDBC の基本的な API を利用するため，メンバが利用方法にとまどうことがない (◎) ・Spring の書籍に詳細な記述があった(◎)

4.3　フレームワーク製品を前提としたアーキテクチャと工夫

フレームワーク選定後，選定したフレームワークの機能や特徴を考慮して，アーキテクチャの検討を行った．

4.3.1　クラス構造の定義

アーキテクチャのベースとなるフレームワークの選定に続き，その上でどの様な役割を持つクラスを作成するのかを決める必要がある．

本システムでは，業務処理の部品化が求められており，以下の点に注意して，クラス構造定義を行った．

- 画面とコントローラ，ビジネスロジックの分離
  ビジネスロジックが画面表示情報やトランザクションの単位，バッチの処理内容に依存しないように，その役割とビジネスロジックの呼び出しを担うコントローラを定義する．
- ビジネスロジックをオンラインとバッチで共用
  オンラインとバッチで同一のビジネスロジックを共用することで，同一の処理が点在することを防ぐ．

プレゼンテーションを担う Struts は，自身が管理するコンポーネントが決まっていることから，定義の無い Spring 部分にどの様な役割のコンポーネントを定義するかが検討ポイントとなる．

1）　オンラインクラス構造の定義

本システムでは，図 5 と表 4 に示すようにコンポーネントを定義した．

オンライン処理は，通常 1 インタラクションを 1 トランザクションで処理できれば良く，Spring を利用した暗黙的トランザクションを利用して，トランザクションの開始/終了の境界を考慮して検討している．

ビジネスロジックは，トランザクションスクリプト[*1] またはシンドメインモデル[*2] という概念をベースに，振る舞いを Service クラスに記述し，扱う属性を Entity に記述する構成とした．再利用を想定するクラスは，Service，Entity，DAO を加えた 3 クラスである．

その 3 クラスを画面等の外部要因から守るために，トランザクション内のコントローラとして Control，その結果を受けて画面をコントロールする Struts 上のコンポーネント群として Action，ActionForm，JSP という構成になっている．

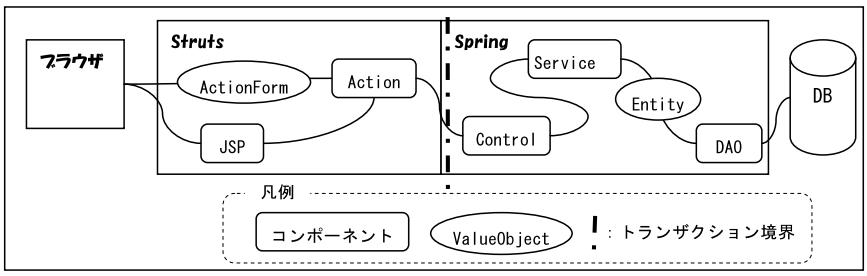


図 5　オンラインクラスのコンポーネント構成

表4　オンラインクラスのコンポーネント一覧

コンポーネント名	作成単位	説明
Action	インタラクション毎	ブラウザからのリクエストを処理し，バックエンドの呼び出しとその結果から JSP の呼び出しを行う．
ActionForm	画面毎	画面からのリクエストを格納する型クラス．
JSP	画面毎	Action から set された値を受け，HTML を生成する．Struts タグ library を利用して作成する．
Control	インタラクション毎	Action から呼び出され，複数の Service の呼び出しを制御し，バックエンドの処理結果を Action に返す．Control は制御のみを行い業務処理や DB へのアクセスは行わない． Control の業務処理メソッドがトランザクションの単位となる．
Service	オブジェクト毎	業務オブジェクト．Control から呼び出され，DAO を呼び出しつつ業務を処理し，結果を Control に返す．
DAO	テーブル毎＋必要都度	DB へのアクセスを行う SQL を記述するクラス．O/R マッピングを役目とし，戻り値は SQL の結果を set できるオブジェクトとなる．
Entity	テーブル毎＋必要都度	レコードやSELECTの取得結果を格納する型クラス．

2)　バッチクラス構造の定義

バッチについても，同様に図6と表5に示すアーキテクチャを決定した．オンラインクラス構成で定義したビジネスロジックをそのまま利用すること，トランザクション制御を自動的に行わずコントローラに任せること，および多重処理の考慮を行っている．

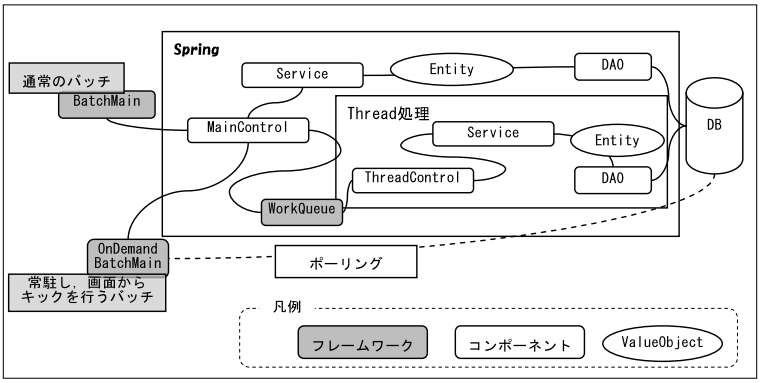


図6　バッチクラスのコンポーネント構成

表5　バッチクラスのコンポーネント一覧

コンポーネント名	作成単位	説明
OnDemand BatchMain	フレームワーク	常駐するバッチ．テーブルに未処理のレコードがあるかをポーリングし，あればレコード上の情報を基に業務処理である MainControl を呼び出す．
BatchMain	フレームワーク	コンソールから起動されるバッチ．コンソールから渡される引数を基に，MainControl を呼び出す．MainControl の処理が終了後，Java のプロセスは終了する．
MainControl	バッチ毎	バッチ処理のメイン処理を記述する．このコンポーネント内で Service を呼び出し，バッチ処理を実施しても良いが，複数スレッドでバッチ処理を平行して実行する場合は，ThreadControl オブジェクトインスタンスを WorkQueue に登録することで実現する．
WorkQueue	フレームワーク	MainControl によりセットされたオブジェクトを，登録された順に自身がプールしているスレッドに割り振り，ThreadControl の特定のメソッドを呼び出し，業務処理を実行させる． プールしているスレッド数は 10 個である．
ThreadControl	任意	WorkQueue により割り振られたスレッド上で動作する Control．Service の呼び出しおよびトランザクションの制御を記述する(オンラインと異なり，トランザクションの管理はバッチの処理として実行する)．
Service		オンラインと同じため省略
DAO		オンラインと同じため省略
Entity		オンラインと同じため省略

バッチのクラス構成では，スレッド処理の考慮によってクラスが分かれたが，コントローラである MainControl，ThreadControl にはビジネスロジックの呼び出しに加え，トランザクション制御も行わせている．これは，オンラインと違い，バッチには複雑なトランザクション制御が求められるケースがあるためである．

4. 3. 2　ApplicationContext の分割について

SpringFramework では，管理クラスの情報を記述する ApplicationContext.xml も様々な処理が記述される重要なソースファイルであると考えられる．このソースファイルを修正せず，オンラインとバッチでクラスの共用を可能とするために，ApplicationContext についても図 7 に示すような役割で分割を実施した．

ApplicationContext の再利用について，考慮すべき点と実施した内容は以下である．

- 共用しないコンポーネントは，別の ApplicationContext として作成する．
- オンラインとバッチでは，データソースおよびトランザクションの制御クラスが異なるため，再利用するためにはデータソース，トランザクション制御に依存しない ApplicationContext にする必要がある．

上記に対応するため，本システムでは，コンポーネント毎に全ての ApplicationContext を分割し，かつデータソース，トランザクションの関連 bean の定義も別の ApplicationContext に分割して外部参照するようにしている．

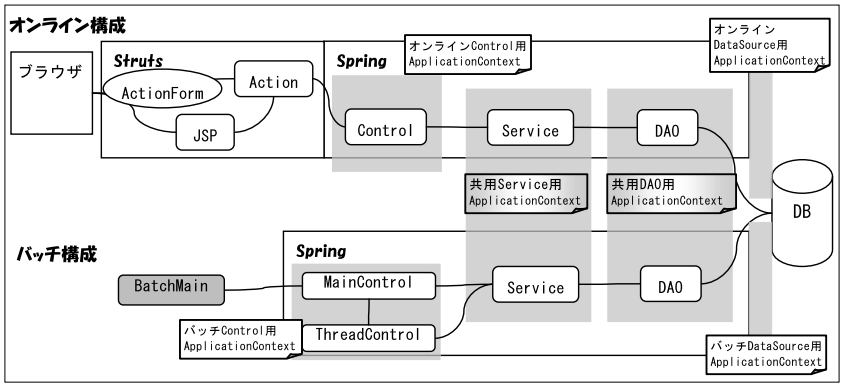


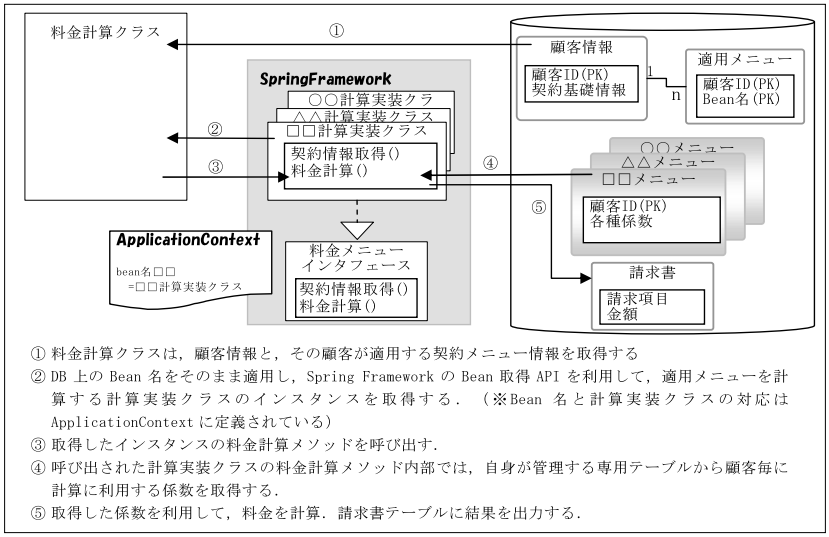
図 7　ApplicationContext の分割とコンポーネントの関係

4. 3. 3　インタフェースベース設計/実装によるさらなる部品化

SpringFramework 等の DI（Dependency Injection）コンテナを利用することで，インタフェースベースの設計/実装[*3] がなされ，実装クラスが隠蔽される事からクラス間の依存度が下がり，実装クラスを容易に入れ替えるなど，より変化に強いシステムが構築できる．本システムでは，インタフェースベース設計をさらに業務的に工夫して利用することとした．

料金計算システムは，新たなサービスにあわせ，契約メニューを柔軟に追加できる必要がある．契約メニューとは，使用量からの料金の算出の仕方のバリエーションであるが，算出に利用する情報や条件が大きく異なり，算出式はもちろん，扱うデータ項目も大きく事なる場合があり，汎用的なデータ構造を定義することが難しい．

そこで，Spring とインタフェースベース設計を利用し，料金メニューをプラグイン的に追加可能な構成を作成した．処理フロー及び構造を図 8 に示す．



① 料金計算クラスは，顧客情報と，その顧客が適用する契約メニュー情報を取得する
② DB 上の Bean 名をそのまま適用し，Spring Framework の Bean 取得 API を利用して，適用メニューを計算する計算実装クラスのインスタンスを取得する．（※Bean 名と計算実装クラスの対応は ApplicationContext に定義されている）
③ 取得したインスタンスの料金計算メソッドを呼び出す．
④ 呼び出された計算実装クラスの料金計算メソッド内部では，自身が管理する専用テーブルから顧客毎に計算に利用する係数を取得する．
⑤ 取得した係数を利用して，料金を計算．請求書テーブルに結果を出力する．

図 8　料金メニューのプラグイン構成と処理フロー

設計のポイントは，料金計算のために必要なクラスは全て同一のインタフェースを実装する点と，Bean 名を直接テーブルに格納する点にある．

上記構成により，新たな契約メニューへの対応は，計算に必要な情報を格納するテーブルの追加と計算クラスを追加するのみで可能となり，既存のテーブル，クラスには影響を与えずに行うことができる．

5.　稼働後の評価

料金計算システムは，本番稼働後 1 年半が経過している．顧客からは，システムの機能性，信頼性，改修性などの面を総合的に判断して高く評価されている．

信頼性については，ハードウェア/OS/ミドルウェアを含めて AP サーバ，DB サーバ共に，非常に高い．このことは，システム再起動が年 1 回のサーバルーム停電時にしか行われていないことからも明らかである．また，料金計算処理時間や画面操作時の応答速度も問題なく，パフォーマンス面でも安定して稼働している．

フェイルオーバの件数については，DB サーバは本番稼働後一度も発生していない．AP サーバについては，本番稼働後 10 ヶ月目に Tomcat が動作する JavaVM の障害があり，フェイルオーバが発生した．障害の発生はクラスタリングソフトにより自動検知され，システムはすぐに復帰したため，料金計算業務に対する影響は小さかった．本障害は，JavaVM のオプション設定に起因するもので，現在は解決済みである．

システム要件として挙がっていた改修/機能追加への柔軟性についても十分に評価されている．稼働後，今までに三度の改修を行っているが，顧客要望に対して，想定された既存部品の改修・必要部品の追加という限定された範囲で容易に対応できている．これらは，OSS を利

用した部品化の効果が十分に発揮されている結果である．

6. おわりに

5章で述べた安定稼働が示すように，Linux/OSSは決して特別なものではなく，システム構成検討の際の選択肢として商用製品と同等に検討を行うべきであることを実証している．

また同時に，バランスを保った構成が実現できたと考えている．本実績を今後のシステム開発，アーキテクチャ選定に活かしていただければ幸いである．

OSSが市民権を得たと言われているが，企業システムに対して，本格的に採用されているケースは全体の割合からするとまだまだ少ない．だが，メーカ，ベンダ各社は，Linux/OSS関連ビジネスの伸びを予想し，サービスメニューの整備やサポート体制の拡充を行っている．本事例も含め，実績が見えるようになってきたことで，Linux/OSSの採用に拍車がかかるであろう．

＊1　トランザクションスクリプト：ビジネスロジックを実装するアーキテクチャパターンの一つ．データアクセスロジックがプロシージャに含まれている手続き型の設計．

＊2　シンドメインモデル：ビジネスロジックを振る舞いと，データアクセスを持たないシン（Thin：薄い）データに分けるアーキテクチャパターン．

＊3　インタフェースベース設計/実装：クラスを設計/実装する際に，機能を提供するインタフェースと，そのインタフェースを実装した具象クラスをわけて考える設計/実装方式．

参考文献

［1］（株）矢野経済研究所，「Linux/OSSのユーザ導入実態調査2006」http://www.yano.co.jp/press/

［2］長谷川 裕一，伊藤 清人，岩永 寿来，大野，（株）豆蔵，「Spring入門」，技術評論社，2005年4月

［3］河村 嘉之，竹内 祐介，首藤 智大，吉尾 真祐，「実践Spring Framework」，日経BP社 2005年5月

［4］石井 真，阿島 哲夫，（株）カサレアル，「Jakartaプロジェクト カンタンStruts1.2」，秀和システム 2005年4月

［5］Ben Galb，Chanoch Wiggers，中川 和夫，「Jakarta Tomcatエキスパートガイド」，ソフトバンクパブリッシング 2003年9月

［6］Jason Brittain，Ian F. Darwin，村上 雅章，「Tomcatハンドブック」，オライリー・ジャパン 2003年12月

［7］須賀 幸次，木村 聡，西川 麗，高安 厚思，白井 博章，椎野 峻輔，岡薫，藤村 浩士，ひがやすを，「Seasar入門」，ソフトバンククリエイティブ 2006年3月

［8］arton，「Seasar2で学ぶDIとAOP」，技術評論社 2006年9月

［9］Christain Bauer，Gavin Ki，倉橋 央，勝嶌 和彦，「HIBERNATE イン アクション」，ソフトバンククリエイティブ 2006年1月

［10］Martin Fowler，長瀬 嘉秀，（株）テクノロジックアート，「エンタープライズ アプリケーションアーキテクチャパターン」，翔泳社 2005年4月

［11］［ThinkIT］第一回：時代は今「DIxAOPコンテナ」，http://www.thinkit.co.jp/free/compare/15/1/1.html

執筆者紹介　鮫　島　荘　介（Sousuke Samejima）
2001年日本ユニシス(株)入社．JavaEE関連開発の技術支援，開発プロセス策定支援，フレームワーク保守開発に従事．現在，エアラインシステムプロジェクト システム開発室に所属．

中　村　彰　彦（Akihiko Nakamura）
2004年日本ユニシス(株)入社．公共機関向けシステム開発にて，要件定義から実装，基盤整備と幅広く従事．現在，エネルギー第二統括プロジェクト ソリューションプロジェクトに所属．