オープンソースフレームワークを利用した WEB システム開発について

株式会社 エーアイ

■ 執筆者Ｐｒｏｆｉｌｅ ■

2001 年　株式会社エーアイ入社
株式会社ネットスクエア出向
現在　国際部所属

三　戸　鉄　也

2002 年　株式会社エーアイ入社
株式会社ネットスクエア出向
現在　国際部所属

内　海　暁　絵

■ 論文要旨 ■

現在のソフトウェア開発，とくに WEB システムの開発においては低コスト，低価格短納期への圧力が非常に大きくなっている．

利幅を減らすことなくこの要求に対応していくために生産効率の更なる向上が必要となることは言うまでも無いが，開発者間のスキル格差の激しい現状では組織全体としての開発効率を上げることが難しい．

そこでオープンソースのフレームワークを利用し，設計を標準化し再利用することによって生産効率を向上させるとともに，オープンソースコミュニティの文化と技術を技術者のスキルアップの材料とし，組織としての生産性の向上を計った．

■ 論文目次 ■

■ 図表一覧 ■

１．背景

１．１　当社概要

株式会社エーアイは昭和 63 年に広島県福山市に設立された.

設立当時はオフコンを中心とした OA 機器販売を主な事業の柱としていたが，平成 3 年度からは富士通とディーラー契約を結び，パソコン販売とパソコン向け受託開発へと事業をシフトした．現在の従業員数は約３０名である.

ただし，本論文の執筆にあたった三戸と内海は株式会社ネットスクエアへ出向して開発業務にあたっており，今回の開発案件は株式会社ネットスクエアが㈲Databank 様より受注し，三戸と内海が担当した案件である.

２．目的と要求

２．１　目的

㈲Databank は社長である曽根知英氏が過去に個人で取得したドメイン e-saketonya.net を利用して，酒卸問屋とその同一商圏の小売店間における WEB-B2B システムの提供を行う.

現在は電話や FAX などで行われている部分に対しての効率化と，小売店に対するマーケティング媒体として酒卸問屋に対してこのサービスを販売していく計画である.

今回の開発は営業展開へ向けて最小限の機能のみを実装し，今後の展開に応じて拡張を施していくものとした.

開発期間は２ヶ月であり，要求定義から据付までを含み，予算は 300 万とされた.

２．２　システム要求

(1) 受発注

本システムにて小売，卸間の受発注をサポートできること.

(2) 小売店と卸問屋の管理

システムを利用する小売と問屋を必要に応じて追加，削除できること.

(3) 商品の管理

商品の情報については Databank が一括管理する.

(4) 卸問屋別の商品登録

Databank によって登録された商品のうち，各卸問屋が取り扱う商品を登録する.

商品コードや卸価格なども卸問屋ごとに設定できるものとする．ただし，卸問屋への商品登録は Databank が行う.

(5) 小売店別の商品検索

商品の検索項目は小売店ごとにある程度カスタマイズできる.

３．開発方針

３．１　開発計画

パッケージや ASP の適用も検討したがコストと要求を満たすものは無く，社内にも再利用可能なシステムが無かったため，新規の開発となった．

また，ネットスクエアではクライアントから直に開発案件を受けることはまれであり，要件定義や設計段階に対する見積もり標準は存在しなかったため，ウォーターフォール的に開発スケジュールを見積もることはせず，納期と基本要求から大まかなスケジュール（**表１**）のみを決め，詳細については動的に計画していくことにした．

技術的にも機能的にも特筆すべき点はないのだが，ボリュームに対してコストと期間の要求も厳しく，プログラミングに割ける期間は 1.5 ヶ月程度が限度と計算されたため困難な開発が予想された．

納期遵守が危ぶまれる場合は，人員の投入などの的確な対処が取れるように達成可能性については以下の項目を用いて常にチェックした．

(1) 反復回数
(2) 作業項目における達成度
(3) 技術的な問題の有無
(4) 進歩速度

表１　開発スケジュール

作業項目	メンバー	8 月		9 月				10 月			
		3W	4W	1W	2W	3W	4W	1W	2W	3W	4W
要件定義	三戸	○	○								
	内海										
分析と調査	三戸		○	○							
	内海		○	○							
学習	三戸				○	○	○				
	内海			○	○	○	○				
開発	三戸				○	○	○	○	○		
	内海				○	○	○	○	○		
テスト	三戸								○	○	予
	内海								○	○	予

３．２　システム要件とオープンソース

コストを下げるために，OS には Linux を，データベースには PostgreSQL を採用し開発言語には二人のスキルと将来のパフォーマンス要求を考慮して Java を採用した．

人員も期間もタイトなため，何らかのミドルウエアが必要であることは間違いないが，学習や購入にかけられるコストは最低限に抑えなければならない．

制約条件を満たせるものは J2EE 準拠のフレームワークと Jakarta プロジェクトによるオープンフレームワークがあったが，学習コストが低いことから Jakarta プロジェクトで開発が進められている struts を利用することにした．

それをうけて，WEB サーバには apache1.3.27 を Servlet コンテナには tomcat4.0.4 を採用することにした．

Struts は MVC アーキテクチャに基づいて作られており，Servlet をロジック実装部，JSP をインターフェース部として明確に分割する．

そのほかに提供する主な機能は以下である．

(1) リクエストと Servlet の関連付け
(2) Servlet の実行結果と JSP の関連付け
(3) リクエスト時のパラメータを JavaBean として抽象化
(4) Taglib による JSP の拡張

これらを XML 形式の設定ファイルによって制御するというのが struts の骨子である（**図１**）．

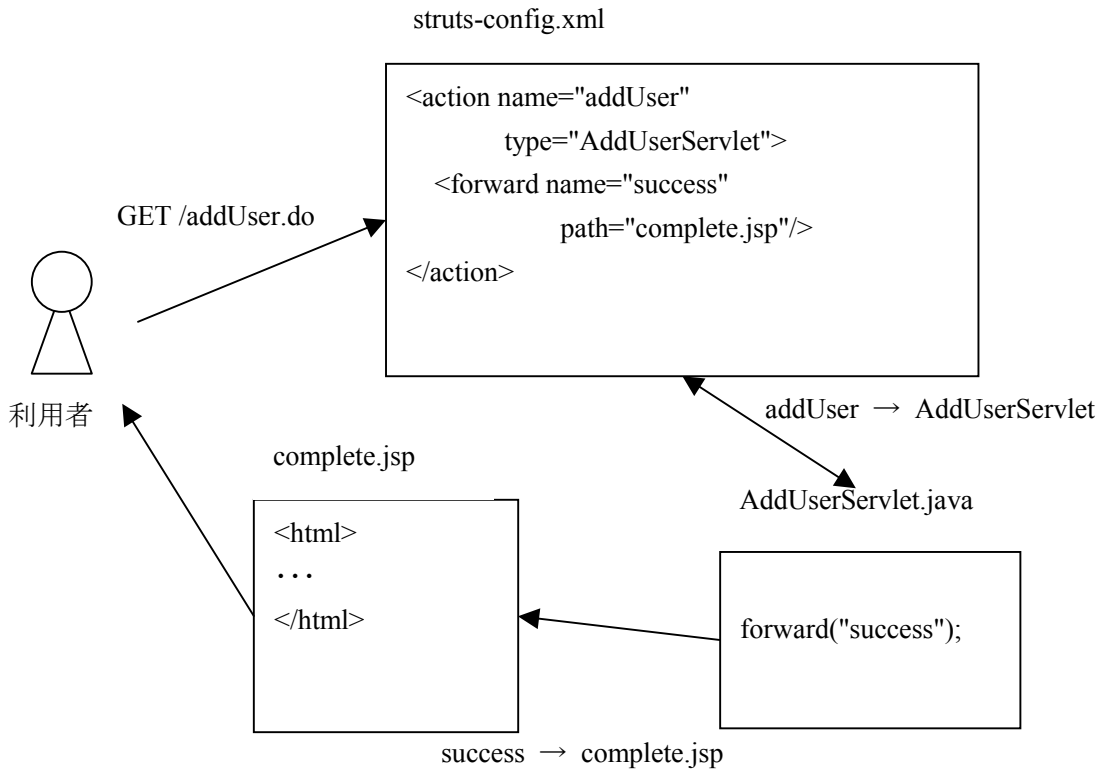

図１　Struts の動作概念図

３．３　開発方針

反復的な開発の長所を最大限に生かすため，XP(eXtreme Programing)からいくつかのプラクティスを導入した．ほとんどが既存の開発でも行われていたことであるが，「プラクティス」であると定義することで，その作業を積極的に意識しようと意図した．

(1) メタファ（比喩）

プラクティスの中で最も抽象的なものであるが，開発者間にスキル格差が大きい場合は必須である．局所的な場面においての「望ましい組み方は？」「どの定石をつかうのか？」などについて開発者間で同意を得ておかなければならないからである．

システムの設計思想や構造を開発者間で共有するために，積極的にシステムのモデル図を作成した．これは作成が定められたフォーマルな文書ではなく，あくまで意識の統一のために作成したものである．

(2) シンプル設計，シンプル実装

その時点における要求を満たすため，必要最小限で実装を行うこととした．

共通機能の再利用やオブジェクト指向なども一切考慮しないものとする．

設計，実装は読んで分かりやすいことと，後で修正しやすいことを第一として進めていく．

(3) テスト駆動開発

実装を始める前にテストから作成することとした．

(4) リファクタリング

実装，設計は絶対なものとせず，より良い実装や設計が明らかになれば既存の部分であっても積極的に修正した．

(5) ソースコードの共有

開発者はソースコードのすべてにおいて責任をもつこととし，開発者であればどの部分の変更も許可されることとした．

これらXPのプラクティスとオープンソースフレームワークとの関わりについては次章で述べていく．

４．実際の開発

４．１　ユースケースの定義とデータ設計

「ユースケース」は「機能」がシステムの立場から見たものであるのに対し，ややクライアントよりの概念である．

機能がどの要求に関連しているかを開発者間で共有するためと，機能テストの単位として用いるためにユースケースの概念を用いて要件トレース表（**表２**）を作成した．

この作業は原則として顧客が在席する場で行い，遷移を含む画面設計もこのフェーズで行った．

データ構造に対しては従来どおりの手法を適用した．

表示に必要とされるデータを帳票サンプルなどの形でクライアントから受け取り，これを正規化することで論理的なデータモデルを作成した．また，シンプル設計に基づいて，この時点でのオブジェクト的なデータモデルとの整合性は一切考慮しなかった．

その結果，総画面数は 96，総機能数は 112 となった．これはネットスクエアにおいて３人月と見積もられるボリュームの約 1.3 倍であった．

データ構造は正規化の結果，総テーブル数は 23，データ項目数は 148 となり，こちらは３人月の見積もりに対してやや多いが，平均的なものであった．

（設計を除いた開発期間 1.5ヶ月に開発人数の２人を掛けて３人月と計算した）

表２　卸問屋管理に関する要件トレース表

要求	ユースケース	機能
卸問屋の管理	卸問屋登録	登録画面表示機能
		登録機能
	卸問屋削除	削除確認画面表示機能
		削除機能
	卸問屋検索	検索画面表示機能
		検索機能
	卸問屋一覧表示	一覧画面表示機能
		一覧取得機能
	卸問屋編集	編集画面表示機能
		情報更新機能

４．２　アーキテクチャ設計

一般における解釈では，struts の各要素を以下のように MVC の各層と関連付けることが多い．

Model ： Servlet

View ： JSP

Controller ： struts-config.xml

しかし，ごく小規模なアプリケーションを除いて，この分類は十分ではない．

Servlet が行うべき処理の中で，ユーザのセッション管理，業務ロジックなどとともにデータベースへのアクセスを実装させようとすれば非常に煩雑な作業になり，開発効率は低いままになってしまう．

この問題を解決するために J2EE デザインパターンにおいて SessionFacade と呼ばれるパターンを応用し，データベースと Servlet 間に中間層を設けた．

この SessionFacade を抽象アプリケーションと捉え，単独で完結したアプリケーションとなるように設計することによって，ユーザごとのセッション処理は SessionFacade 単位で行えばよいことになるし，データベースへのアクセスなどに関する複雑なロジックは Servlet から一切隠蔽されることになる（**図２**）．

また，Servlet と SessionFacade 間のデータの受け渡しについては JavaBean を利用する．

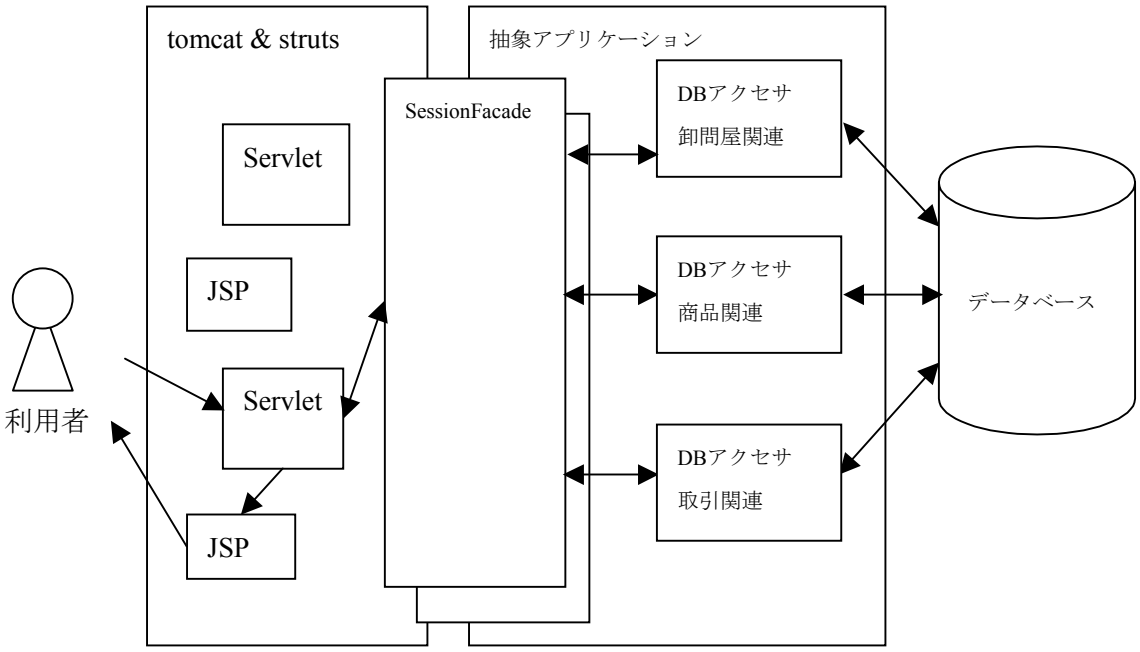

図２　アーキテクチャ図

４．３　実装

リクエスト処理とユーザインターフェース表示部はStrutsによってサポートされているため，プログラミング作業の大部分を抽象アプリケーションの作成に費やすことができた．

まずは SessionFacade において擬似データを返すメソッドを準備し，内海がそれを利用するようにServletを組み，三戸がDBアクセサの完成に合わせてメソッドが本来の機能をするようにに置き換えていった．

つまりSessionFacadeはServletから見ればスタブであり，DBアクセサから見ればドライバとなる．このようにしてインターフェース側とデータベース側から挟み込むように実装を行った．

４．４　リファクタリング

「シンプル設計」に基づいてアプリケーション作成を行えば，ユースケースが実装されていくに伴ってあちこちに共通化できるロジックや最適化できる設計が多く見つかる．

今までの開発ではよほど影響が大きいか，要する工数がよほど少なくない限り修正作業を行うことは無かったが，今回は積極的に修正をした．

これを工数が発散することなく繰り返すには，開発者間で実装方針に対して細かい点まで同意がなされていることが必要条件である．そのために学習段階からメタファを利用して，方針に対しての同意を形成した．

実際にリファクタリングの対象となった事柄のうち，比較的大きなものを挙げる．

(1) JavaBeanとデータベーススキームとの関連

データベースのスキームをJavaBeanに１対１で対応させていたが，表示部においてもデータ取得時においてもパフォーマンスの面で問題が出てきたため，表示に必要な項目に合わせて複数の JavaBean を複数内部に含む表示用 JavaBean を定義し，データベースからは一回のトランザクションで取得できるようにした．

このためには抽象アプリケーション自体の仕様を修正する必要があったが，既存の構造を残しつつ緩やかに新しい構造に移行していった．

古いインターフェースについてはコンパイル時に警告を出すようにして新しいインターフェースへの変更を促した．

修正作業については，三戸が抽象アプリケーションの変更を Servlet に反映させることもあれば，内海が変更を調べてServletに反映させることもあった．

(2) データの妥当性チェック

データの妥当性チェックは JavaBean の各受け取り部において行っていたが，Strutsのソースを参考に，JavaBean内部で行うように修正した．

５．評価と今後の課題

５．１　顧客側からの評価

(1) 十分な品質

検査によって３つのバグが発見されたが，どれも簡単な修正作業で対処できた．

これはネットスクエアにおける平均品質とほぼ同等である．

(2) 低価格，短期間

ほかのプロジェクトと比較して，77%ほどの工数で完成できたことを考えると価格面，開発期間の両面から見ても効果があったといえる．

５．２　開発側からの評価

(1) アプリケーションの規模が見積もりやすい

アプリケーション開発に必要な工数は，要求，開発言語，プラットフォームなどによって大きく増減し，決定事項が多いほど見積もりは正確になる．

Struts を継続して利用することで精度の高い見積もりが可能になるだろう．

(2) 適切にシステムを分割できる

以下の値をもって同工数のプロジェクト平均と比較した（**表３・表４**）．

クラス数と平均行数はシステムの規模を表し，標準偏差をもってクラスの規模のばらつきを表すとした．

さらに，クラス設計の良否を計るために「依存／関連」を用いた．

クラス A がクラス B のメソッドを利用する場合を「A と B は関連する」と定義し，更にそのメソッドの戻り値の型が不定の場合や，引数として渡したオブジェクトを内部で操作する場合を「関連クラスの内部実装に依存する」と定義し，依存クラス数を関連クラス数で割ることでクラス同士の依存度合いを表すとした．

この値が小さいということは，互いに変更の影響を受けにくい部品群でシステムが構成されていると言うことである．

５%有意水準でカイ２乗検定を行った結果，行数と標準偏差には有意な差は見られないが，依存／関連には有意な差が見られた．

表３　評価項目

クラス数	：クラス数合計	：小さいほど良い
平均行数	：１クラスの平均行数（コメント含まず）	：小さいほど良い
標準偏差	：行数の標準偏差	：小さいほど良い
依存／関連	：関連クラスの内部実装に対する依存	：小さいほど良い

表４　比較表

	クラス数	平均行数	標準偏差	依存／関連
Struts 利用開発	248	84.5	86.6	0.30
フレームワーク非利用開発	273	78.1	86.9	0.49

５．３　課題

(1) 定量的な評価の難しさ

品質を下げることなく開発期間の短縮とコストの低下を実現できることは定量的に測定できたが，Struts が直接もたらす効果である設計の最適さやコーディング方針の統一度などは測定が困難であった．

依存／関連をもってクラス間の依存度を表し，これによって最適なクラス構造であるかどうかを測定したが，もっとよい評価基準があるかもしれない．

また，反復的開発における最終設計への収束速度も速まると思われるが，今回以外に反復的開発を行っていないために比較できなかった．

加えて標準偏差の値は平均行数に対して大きすぎるものであった．

これはクラス自体が大小のいくつかの種類に分類可能であるにも関わらず，総合して計算したためとおもわれる．これは設計思想の違うシステムにおいて両者に対して公平な分類方法が見つからなかったためである．

(2) 実際の運用はまだ開始されていない

実際のサービスが開始されていないため，本当に将来の変更に柔軟に対応できるのかは測定できていない．これは今後にわたって追跡調査する．

６．むすび

要求の厳しい案件に対して新技術を用いることには非常に不安があったが，オープンソースのものは広く提唱されている方法論や開発手法を踏まえたものが多く，一般書籍において解説されているような設計，プログラミング，テストにおける手法や方法論がそのまま適応可能であった．そういう意味では，ドキュメントの量は豊富であると言えよう．

また，デザインパターンなどとの親和性の高さは，ケーススタディ的なオブジェクト指向開発を可能にし，古典的な開発手法からオブジェクト指向開発への移行を容易なものにできるのではないかと思う．