B-014

フレームワークを用いた Web アプリケーションにおける変更波及解析
Change Impact Analysis in Web Application Using Framework

高橋 明日香† 小林 洋†
Asuka Takahashi Hiromi Kobayashi

1.はじめに

近年，Web アプリケーションを開発する際，フレームワークを使用した開発が増えている．フレームワークを用いるメリットは開発効率を高めるためと言われている．しかしながら，その評価について示された文献はほとんど見当たらない．ところで，アプリケーションにおいては，一旦開発が行われた後，修正・追加が入ることが多いことから，最近では，変更時の修正の容易性が重視な指標となっている．そこで，本研究では，従来手法である JSP とフレームワーク Ruby on Rails での同一のアプリケーションの修正・追加の際の変更波及解析(Change Impact Analysis)を行った．対象とするアプリケーションとしては，図書管理システムのモデルを用いた．

2.開発における修正・追加

業務系の Web アプリケーション開発では，フレームワークが盛んに用いられるようになって来ている．その目的は，開発効率を高めるためと言われている[1]．しかしながら，その評価について示された文献はほとんど見当たらない．そこで本研究ではこの問題を扱うことにした．ところで，アプリケーションというものは，一旦開発した後，良く使われるものは通常，修正・追加の開発が入る．そこで，我々はこの点に着目して，修正・追加の際の変更箇所についての評価を行うことにした．変更箇所の評価，つまり差分の評価であれば，アプリケーションの開発効率の評価で問題になるスケーラビリティの問題は緩和されるのではないかと考えられる．本研究では，従来手法とフレームワークを用いた手法で，各々，図書管理システムモデル[2]の開発を行い，その後，修正・追加を行い，変更波及解析[3-4]によりプログラムの修正箇所の比較を行い評価することにした．

3.Ruby on Rails について

本研究では，フレームワークとして Ruby on Rails（以下，Rails）[5-6]を用いた．Rails はスクリプト言語 Ruby を用いるフレームワークで，Web システムのアジャイルな開発に用いられている．Rails の構成を図 1 に示す．Rails

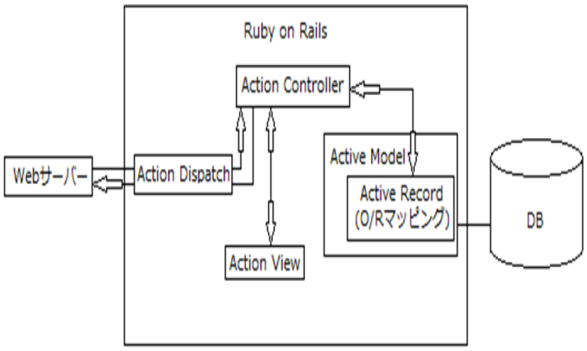


図 1 Ruby on Rails の構成

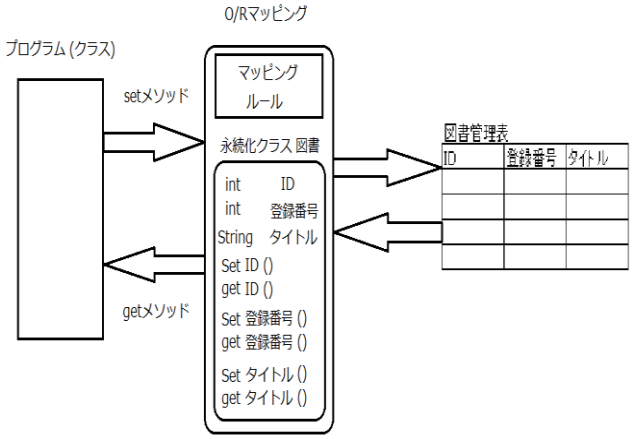


図 2 O/R マッピング概念図

では ActiveRecord という O/R マッピングのライブラリにより，データベースを使用する Web アプリケーション開発において，記述するソースコード量を削減することができる．また，MVC モデルアーキテクチャをサポートしており，データベース処理を含む業務処理部分を担当する Model，データ表示を行う View，及びこれらを制御する Controller の自動生成機能により容易にプロトタイプの構築を行うことが可能である．この際，図 2 に示すような O/R マッピングにより，オブジェクト指向プログラミング言語におけるオブジェクトの各属性値とリレーショナル・データベースのテーブルの各属性値との対応付けが行われ，プログラムとテーブルの間のデータ表現の違いが吸収される．

† 東海大学大学院工学研究科 Tokai University

4.図書管理システムの開発と修正・追加

図書管理システムモデルを基に，従来の Java の Web アプリケーション開発では標準的な JSP（Java Server Pages）と Rails を用いて各々 Web アプリケーションの開発を行った．JSP での開発では，Rails との比較が可能なように，MVC モデルを採用した．なお，この最初の開発により Rails についても習熟することが出来た．最初に作成したアプリケーションでは，図書の一覧表示，図書の登録，図書の削除の機能があり，図書の属性は，登録番号，分類番号，ISBN，タイトル，著者，出版社，出版日，分類の 8 つからなる．

このアプリケーションに属性の追加と更新機能の追加を以下のように行い，JSP と Rails での各々の変更箇所の比較を行った．

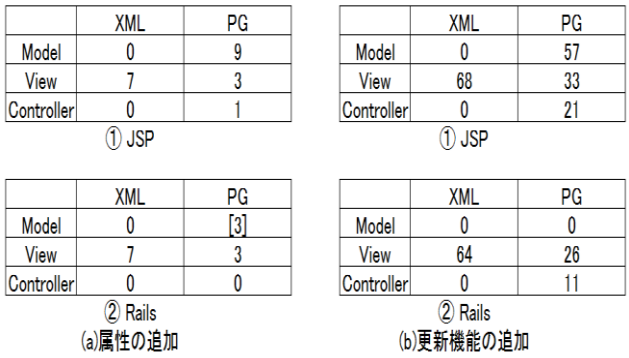
（a）属性追加：図書の属性として*価格*の追加を行った．図書一覧，図書登録，及び図書削除の画面に価格の属性を追加した．なお，図書削除の画面では,一旦，全属性を表示しデータの確認後削除を行うようにしたため価格の表示が必要になっている．

（b）更新機能追加：最初に作成したアプリケーションでは，あえて作成していなかった更新機能を追加した．更新機能は，メニュー画面から図書更新を選択すると，一旦，全ての図書データを表示し，その後更新図書の選択を行えるようにした．以上のことを実装するために View ではメニュー画面に図書更新のハイパーリンクの追加，更新図書選択画面の作成，更新画面を作成した．Model では，図書選択画面で選択した図書の全ての属性を表示後，該当箇所の属性値を入力し更新ボタンを押すと更新が行えるようにした．Controller では，View と Model の登録番号の受け渡しと取得データの受け渡しを行うようにした．開発したアプリケーションの画面を付録に示す．これは，JSP と Rails どちらの開発でも共通なものである．

5.評価

データベースのテーブルへの属性追加に関しては，Rails ではコマンドを入力するだけでテーブルの更新が自動的に行われるため，ソースコードの追加は不要である．一方，JSP では，SQL 文と Java 文を追加する必要がある．しかしながら，一般的にはテーブルの変更がある場合には，アプリケーションの Model，View，及び Controller に関わる処理へ変更が波及するため，これらの部分についてソースコードの追加・変更が必要となった．

ハイパーリンクを用いたページの遷移に関しては，JSP では URL の記述によって指定するのみである．一方，Rails では URL に応じて Controller 内の表示・登録・削除・更新に関するメソッドを指定するための Routing の記述が



	XML	PG
Model	0	9
View	7	3
Controller	0	1

① JSP

	XML	PG
Model	0	57
View	68	33
Controller	0	21

① JSP

	XML	PG
Model	0	[3]
View	7	3
Controller	0	0

② Rails

(a)属性の追加

	XML	PG
Model	0	0
View	64	26
Controller	0	11

② Rails

(b)更新機能の追加

図 3 修正・追加における変更波及解析（LOC）

必要となるため，JSP より Rails の方が変更箇所が多くなった．

フォルダ数について見ると，フォルダ数はアプリケーションの規模や作成者の意図によって変化するものである．eclipse で開発する場合には，JSP の場合には，デフォルトで出来る 7 個に加え，アプリケーションを Model・View・Controller に分割して各々フォルダを作成するならば，更に 3 個のフォルダが必要となり，これが小規模開発で最低限必要なフォルダ数ということになるだろう．一方，Rails では，デフォルトで 36 個のフォルダが自動的に作成される．一般的には，フォルダ数が多いと修正・追加箇所の特定に時間がかかると言えるが，Rails でのこれらのフォルダは，役割別に分かれているので，Rails に習熟すれば修正・追加箇所の特定はそれほど困難ではないと考えられる．

4．での(a)属性追加と(b)更新機能追加における，JSP と Rails での追加・変更について定量的にコード行数 LOC（Lines of Code)で比較すると図 3 のようになった．図 3 では，Model，View，Controller に区分し，各々 XML と言語（Java または Ruby）の追加・修正の行数を示している．

属性追加の場合には，JSP では Model にコードの追加が必要だった．一方，Rails ではコマンドによる操作が必要であった．コマンドとプログラムのコードを同等に扱って良いものかという問題はあるが，一応ここにコマンド 3 行分を[]付で記しておくことにした．View については，どちらもコードの他 XML の記述の変更が必要だった．Controller については，JSP では価格の値を View へ受け渡すコードを 1 行記述する必要があったが，Rails では必要なかった．

更新機能追加の場合には，Model については，JSP では登録番号等の 9 つの属性を表示するための値の受け渡しと更新を行うために 57 行のコードの追加が必要であった．一方，Rails ではコードの追加は必要なかった．View については，JSP ではリストボックスを使用する際，繰り返し文を使用したのに対して，Rails では

繰り返し文が不要であったため，Rails の方がコードが 7 行少なくて済んだ．なお，View については，この他どちらも XML での記述の変更が必要だった．Controller については，JSP では更新を行う 9 つの属性を Model へ受け渡すためのコードの追加が必要であった．一方，Rails では 9 つの属性を受け渡す Ruby の文が一行で済んでしまうことから，Rails の方が 10 行少なくて済んだ．

6.おわりに

フレームワークを用いた場合のアプリケーションの修正・追加の容易性を評価する一つの方法として，本研究では Rails を取り上げ JSP と比較した変更波及解析を行った．Rails はアジャイルな Web アプリケーション開発では良く用いられているが，JSP との比較では，言語の違いがあるために必ずしも妥当でないと思われる．

そこで，引き続き Java ベースのフレームワーク Hibernate により同様の研究を進めて行く予定である．また，今回の比較では，データベースのテーブルに関わる機能追加を行ったが，アプリケーションではテーブルの CRUD に関わらない機能も考えられるので，更なる検討を続けて行きたい．

参考文献

[1] 満田成紀，福安直樹: ウェブアプリケーションフレームワーク利用に潜む課題，コンピュータソフトウェア，Vol.27,No.3，pp.2-12 (2010).

[2] 中所武司，藤原克哉: Java による Web アプリケーション入門，サイエンス社 (2005).

[3] B. G. Ryder, F. Tip: Change Impact Analysis for Object-Oriented Programs, PASTE'01, pp.46-53 (2001).

[4] 早瀬康裕，松下誠，楠本真二，井上克郎，尾林健一，吉野利明: 影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案，電子情報学会論文誌，Vol. J90-D, No.10, pp.2736-2745 (2007).

[5] D. Thomas, D.H. Hansson，前田修吾監訳: Rails によるアジャイル Web アプリケーション開発，オーム社 (2006).

[6] まつもとひろゆき: プログラミング言語 Ruby の世界普及戦略，情報処理学会デジタルプラクティス，Vol.2，No.2，pp.74-79 (2011).

付録：開発アプリケーションでの画面

図 A1　メニュー画面

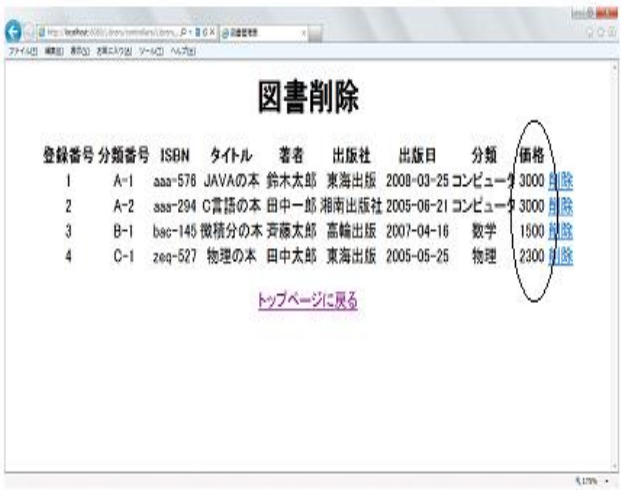


図 A4　図書削除画面

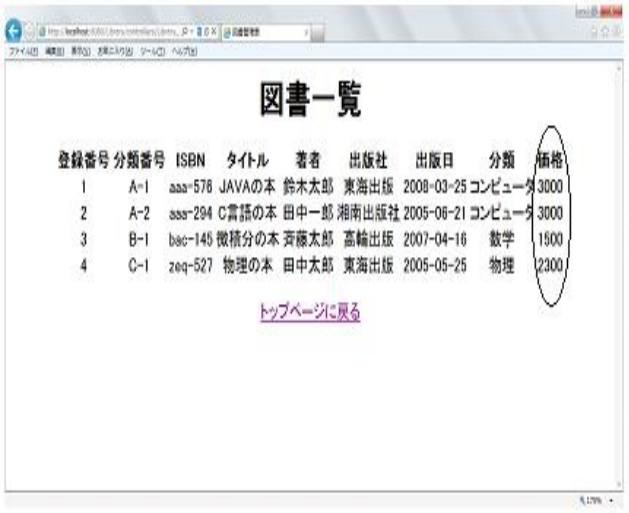


図 A2　図書一覧表示画面

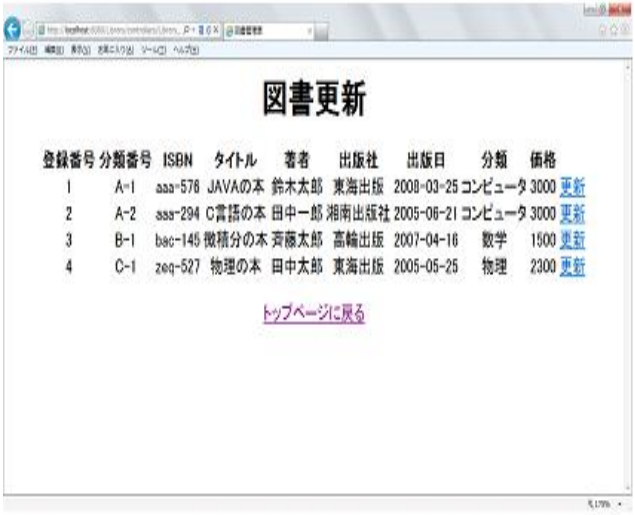


図 A5　図書更新選択画面

図 A3　図書登録画面

図 A6　図書更新画面