Rails アプリケーションのアーキテクチャに関する研究

2003MT009 伴野 哲也　2003MT011 深谷 洋　2003MT075 直良 誠

指導教員 青山 幹雄

1. はじめに

現在 Rails フレームワークが注目されている. その理由は開発期間を著しく短縮することができるためである. しかし, 開発が大規模になるにつれ, アプリケーションの全体像が把握できなくなり, リファクタリングがしにくくなる問題点が挙げられる. 私たちはその解決策として, リファクタリングや Rails アプリケーションの MVC アーキテクチャおよび規約を視覚化するツールを提案し, 例題を用いて検証した.

2. Rails

2.1. Rails とは

Rails フレームワークとは, 設定より規約や, コードを自動生成などによって, 開発期間を著しく短縮することができるフレームワークである.

2.2. Rails の特徴

2.2.1. 規約に基づく

Rails には, 設定よりも規約という特徴がある.

例えば, Rails では「モデルクラスにはマッピングしたいテーブルの単数形を命名する」という規約がある. これに則って「Members」テーブルとマッピングしたいクラスに「Member」という名称をつけるだけで自動的に O/R マッピングされて簡単にデータベースアクセスできるようになる.

2.2.2. 同じ作業を繰り返さない

Rails には, 同じ作業を繰り返さない(Don't Repeat Yourself)という規約がある. 従来の Web アプリケーションフレームワークでは, 毎回同じコードを書かなければならない場合が数多くある. Rails は, このような重複を回避することによって, 開発スピードが上がり, 変更にも強くなる.

2.2.3. コード自動生成

Rails には, Scaffold というコード自動生成機能がある. Web アプリケーションからデータベースを操作する新規作成(Create), 抽出(Read), 更新(Update), 削除(Delete)の機能を提供する. 以下に, コードを自動生成する手順を示す.

(1) DB の作成を記述
(2) Rails アプリケーションを生成
(3) DB のプロパティを記述
(4) モデル, コントローラのテンプレートを生成
(5) Scaffold によるコード自動生成

表 1 に, 自動生成されたファイルを示す.

表 1 自動生成される MVC ファイル群

機能	ファイル
モデル	item.rb
ビュー	edit.rhtml, list.rhtml, new.rhtml, show.rhtml items.rhtml, _form.rhtml
コントローラ	items_controller.rb

2.3. Rails のアーキテクチャ

Rails は MVC アーキテクチャを採用した Web アプリケーションを開発するためのフレームワークである.

2.4. Rails の開発方法

Rails では, アジャイル(反復型)開発プロセスでの利用を想定しており, 各イテレーションでの作業量を従来のフレームワークにくらべ大幅に削減している.

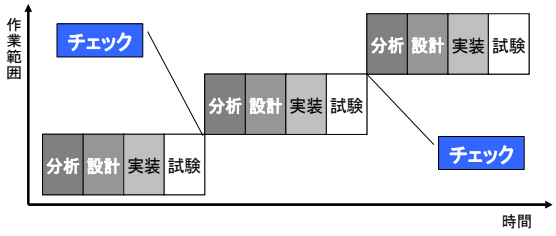


図 1 アジャイル開発

3. リファクタリング

3.1. リファクタリングとは

リファクタリングとは, ソフトウェアの振舞いを保ったままで, 内部構造を改善するようにコードを変更する技術である. リファクタリングの目的として, ソフトウェア設計の向上, ソフトウェアの理解の易化, 開発効率の向上などがある.

3.2. リファクタリングカタログ

リファクタリングカタログは, リファクタリングの技術をまとめたもので, 現在 72 種類のリファクタリングがある[2].

4. 問題点と解決策の提案

4.1. Rails におけるリファクタリングの問題点

アジャイル開発プロセスは機能追加によるコード変更が反復ごとに生じることを前提に考えられているため, リファクタリングの機会が多く, その重要度が高いといえる. さらに, Rails の MVC アーキテクチャや記述量削減方法に従うこと

が，リファクタリング適用時に与える制約についても考える．

本研究では，Railsを利用した開発におけるリファクタリングの問題点として，次の 3 つに着目した．

(1) アジャイル開発プロセスとリファクタリング

(2) 規約とリファクタリング

(3) コードの自動生成とリファクタリング

(1)は，開発が大規模になるにつれ，プログラムの全体像が把握できなくなり，リファクタリングが困難になる．アジャイル開発プロセスでは，設計に時間を割かず，動くプログラムを重視し機能を追加していく．アプリケーションはイテレーションごとに変化していくため，ドキュメントの作成に時間をかけることができない．そのためプログラムの全体像が把握できなくなる．

(2)は，Rails の規約により縛られる要素間の結合度が高くなり，リファクタリング時のコード変更に対して連鎖的な変更が生じる可能性がある．

(3)は，Rails では自動生成によりディレクトリ構造，ファイルの配置，クラスの継承関係，メソッドの配置などがある程度固定されるため，リファクタリングにおいて制約を受ける．

4.2. 視覚化ツールの開発

Rails における開発では，開発が大規模になるにつれ，アプリケーションの全体像が把握できなくなり，リファクタリングが困難になる問題点があった．さらに，Rails の想定する開発手法を利用することによって生じる様々な規約についても，リファクタリングをする上で問題となる．そこで，本研究では，リファクタリングを支援するために，Rails アプリケーションの全体像すなわち MVC アーキテクチャと，それらが持つ制約を視覚化するツールを提案する．このツールを用いることにより，設計に時間をかけることなく，プログラムの全体像を把握し，コードを整形しながら開発を進める環境を目指す．また，これにより，自動生成されるファイル群についてもその構造を示すことができる．

4.3. 視覚化ツールの機能

4.3.1. ディレクトリとファイルの表示

MVC を構成するファイルとそのヘルパークラスのファイルを表示する．表示するのはファイル名，ファイル内で使用するインスタンス変数，メソッド(ファイルがクラスファイルの場合)の 3 つである．これは UML のクラス表記法を利用する．ディレクトリ構造の視覚化は，そのサブディレクトリやファイルを四角形の枠で囲うことで表現する．

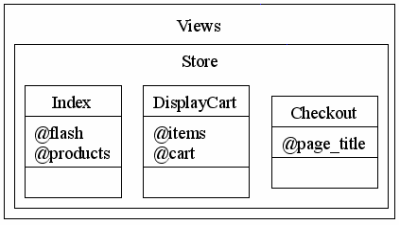


図 2 ディレクトリとファイルの表示形式

4.3.2. 継承と関連の表示

クラスの継承および関連を UML の表記法で表示する．親クラスが Rails のプロジェクトディレクトリにない場合，Rails が標準で提供するコンポーネントを利用することが考えられる．この場合は利用するコンポーネントと親クラス名を円で囲うことで表現する．

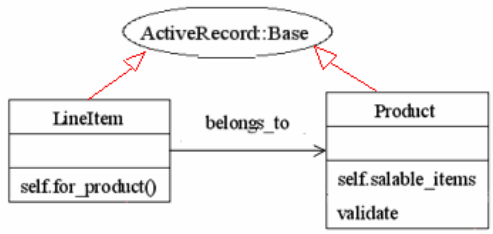


図 3 継承と関連の表示形式

4.3.3. ファイルの命名規約の表示

Rails では，ビューファイルを格納するディレクトリ名とヘルパーファイルのファイル名は，コントローラ名により制約を受ける．この制約を色の異なる矢印で結び表現する．

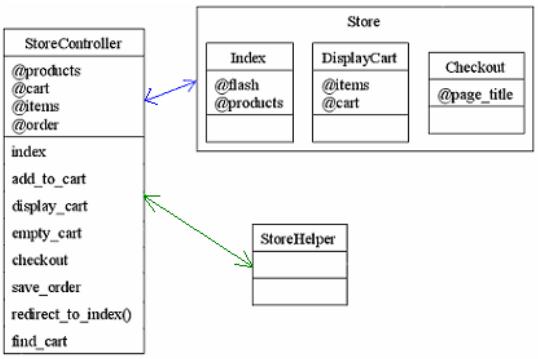


図 4 ファイル命名規約の表示形式

4.4. ツールの実装

ツールは Ruby で記述した，rake 形式のファイルである．コードは 213 行となった．rake とは，Ruby 版の make であり，Ruby プログラムをビルドする Ruby プログラムである．

今回のようにツールとして独自の rake タスクを追加する場合は，lib/tasks ディレクトリ以下に，”.rake”という拡張子を持つファイルを置くと，Rakefile から自動的にインクルードされる [3]．このツールは，ファイル名を show_rails_tasks.rake として，lib/tasks ディレクトリに配置した．そして，”rake show_rails”という rake コマンドを用いてこのタスクを呼び出すことで処理を実行できるように，ファイル内に記述した．

このツールの入力は，Rails プロジェクトの app ディレクトリであり，出力は dot 形式のベクタ画像である．このベクタ画像は，AT&T の Graphviz というオープンソースのグラフ描画アプリケーションで展開が可能であり，グラフ描画に特化している [1].

5. 例題によるリファクタリングの分析

5.1. 例題の概要

ツールの機能を Web ショッピングアプリケーションの作成により確認した. 扱うデータとその間の関係を図 5 に示す.

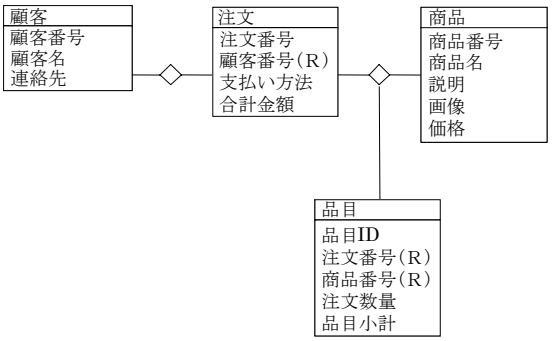


図 5 Web ショッピングアプリケーションのデータの関係

アジャイル開発プロセスは動くプログラムを中心に開発を行うため, 階層の浅い部分から実装していく. よって次の順序で開発を行った.

1) 商品管理機能の作成
2) 商品カタログの表示機能の作成
3) ユーザ登録機能の作成
4) カート機能の作成
5) 注文機能と発送機能の作成

5.2. リファクタリング

(1) コードの重複によるメソッドの引き上げ

商品管理機能で作成した Admin コントローラと商品カタログ表示機能で作成した Store コントローラの間にコードの重複が発生した. 重複しているのは, Rails を日本語表記に対応させるための set_charset メソッドである. ここでは, Admin コントローラと Store コントローラの重複部を Application コントローラに引き上げることでリファクタリングを実現した. その実現方法を図 6 に示す.

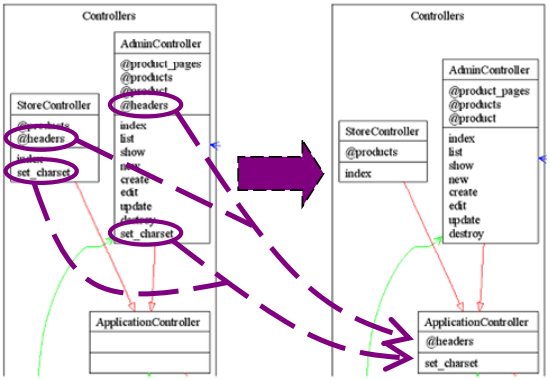


図 6 メソッドの引き上げ

(2) メソッド呼び出しによる置き換え

カート機能の作成で作成した Cart モデル内にコードの重複が発生した. 重複内容は, Cart モデル内で使用するインスタンス変数の初期化処理と, カートの中身を空にする empty!メソッドの処理がまったく同じであることである. この問題に対するリファクタリングは, 初期化処理内の重複部を empty!メソッドの呼び出しに置き換えることで実現した. これを図 7 に示す.

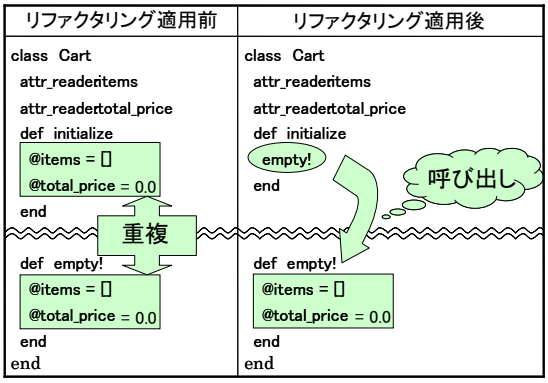
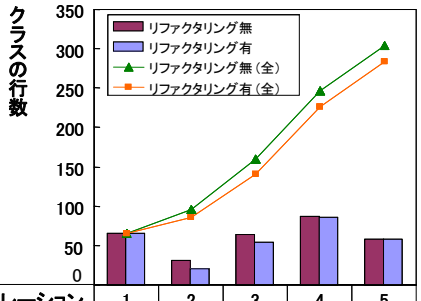


図 7 メソッド呼び出しによる置き換え

5.3. 各イテレーションにおけるリファクタリングの分析

例題においてリファクタリングを行った場合と, 行わなかったと想定した場合について, 全クラス行数とその増加量のイテレーションごとの推移をグラフにした. これを図 8 に示す.



イテレーション	1	2	3	4	5
リファクタリング無	65	31	64	87	57
リファクタリング有	65	21	54	86	57
リファクタリング無(全)	65	96	160	247	304
リファクタリング有(全)	65	86	140	226	283

メソッドの引き上げ　メソッド呼び出しによる置き換え

図 8 クラスの行数とリファクタリング

図 8 に示すように, メソッドの引き上げによってコードを 10 行削減できた. さらに, 引き上げたメソッドの機能を再利用するクラスを新規作成する場合, 引き上げ先のクラスを継承することによってメソッドを再利用できる構造となった. こ

れは，メソッドの引き上げを行わなかった場合，イテレーション 3 においても 10 行のコードの重複が起こっていることからその効果が確認できる．また，メソッドのコードを変更する場合も，引き上げたメソッドを変更するだけで済む．

一方，メソッド呼び出しへの置き換えは 1 行と削減量の面では効果がなかったが，構造の面ではメソッドの引き上げと同様に，変更箇所を 1 つにまとめることができた．

これらのことから，メソッドの引き上げは 2 つ以上のクラスにまたがる大域的なリファクタリング，メソッドの抽出は単一クラス内のコードを修正する局地的なリファクタリングと考えられる．

6. 評価と考察

ツールが対応する MVC アーキテクチャに対する視覚化は次の 5 つである．

(1) ディレクトリの表示
(2) ファイルの表示
(3) クラスの継承関係の表示
(4) モデルクラスの関連の表示
(5) ファイルの命名規約の表示

MVC の 3 要素を最上層，コードを最下層と考えると，視覚化のレベルは表 2 に示すように分類できる．

表 2 視覚化のレベル

層	抽象度	ツールでの対応		例題のリファクタリング
MVCの3要素	最高	全部	対応機能: (1) (2)	
ディレクトリとファイル配置	高	全部	対応機能: (1) (2)	
ファイル内の詳細情報とそれらの関係	中	一部	対応機能: (1) (2) (3) (4) (5)	・メソッドの引き上げ ⇒(1)(2)(3)が対応
コード	低	なし	未対応	・メソッド呼び出しへ置き換え ⇒ツールの効果なし

表 2 からわかるように，このツールにおいては，ディレクトリとファイルの配置の層までは(1)と(2)によりすべて判別可能である．ファイル内の詳細情報とそれらの関係の層は(1)と(2)と(3)と(4)の対応する範囲までとなっており，すべてではない．そのため，例えばメソッドの親クラスへの引き上げといった大域的なリファクタリングを行う場合，効果があるといえる．しかし，メソッド呼び出しへの置き換えといった局地的なリファクタリングに対しては効果がないと考えられる．

また，(4)および(5)の機能は例題における 2 つのリファクタリングのどちらにも効果が期待できない．

このように，今回のツールでは，すべての規模のリファクタリングに対し効果があるわけではない．また，(4)や(5)の機能のように，効果が期待できない，もしくは，リファクタリングと関係の低い機能も実装している．

7. 今後の課題

今後，今回のツールに対し，視覚化する情報の追加や削除を行う必要がある．その際，クラス図の表記法の中でもどの表記を採用するか，他の構造図や，UML 以外の表現方法は採用できないか，などを検討する必要がある．そのとき，Railsを利用した開発においてどのようなリファクタリングが多く行われるかも考慮すべきである．

最終的にはリファクタリングに対する効果が高い表記法で表記することになるが，それにはリファクタリングに対する効果を定義する尺度が必要である．例えば，例題ではメソッドの引き上げに対して 3 つの機能が効果を発揮していたが，他の 2 つの機能は例題中でのリファクタリングには効果をなさなかった．これより，この 2 つの機能は将来削除する可能性が高くなるといったことが考えられる．

また，開発の規模によっては，図の複雑化や巨大化の問題が考えられる．これは，このツールがプロジェクトディレクトリ内のディレクトリ構造を出力するためである．今後，視覚情報を階層化やモジュール化して出力する必要がある．

階層化とは，目的(ユーザインタフェースやビジネスロジックなど)に応じて層を定義し，上位層が下位層の内部の仕組みを理解することなくそれらを利用するように分割したものである．Rails ではビューを最上層のユーザインタフェース層とし，次にアプリケーション層としてコントローラ，次にビジネスロジック層としてモデルという分割が考えられる．また同じ層においてもモジュール化により複数の機能を 1 つの機能にまとめることが可能であり，これらの概念を用いることで図の複雑化や巨大化を回避することが可能となる．

8. まとめ

本研究では，Rails を利用したアジャイルな Web アプリケーション開発の支援として，Rails を使用したアプリケーションの MVC アーキテクチャおよび規約を視覚化するツールを開発した．ツールはディレクトリとファイル構造の視覚化を主体とし，それらの関係を UML の継承と関連の表記法を用いて表した．また，命名規約の表記も取り入れた．

このツールにより，全体像を参照しながら，リファクタリングを行うことが可能となり，それによって，アジャイルな開発で多発する機能の追加，変更への適応性が向上した．

参考文献

[1] AT&T, Graphviz, http://www.graphviz.org/.
[2] M. Fowler, et al., リファクタリング，ピアソン・エデュケーション, 2000.
[3] D. Heinemeier Hansson et al., 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.