フレームワークを利用した開発における形式仕様記述の適用に関する検討

Examination on Application of Formal specification to Framework-based Development

演習コースⅡ　グループ2
有賀 一輝　株式会社イクズアネックス
水野 徹平　株式会社富士ゼロックスアドバンストテクノロジー
東海 政治　株式会社 NTT データ CCS

概要

あいまいな仕様による不具合を解決する方法の 1 つとして形式手法がある．これを，Web アプリケーションのフレームワークを用いた開発に適用する際の課題や問題点を，ケーススタディを通して明らかにする．また，形式記述言語 VDM++[1]をいかに活用していくべきかを議論した結果を報告する．

Abstract

Formal methods are one of the solutions for problems caused by ambiguous specifications. This work clarifies challenges and problems in applying formal methods to a development of a Web application framework through case studies. This work also reports a result of discussion about how to leverage formal specification language VDM++.

1. はじめに

近年，Web アプリケーションの開発において，フレームワークドリブンによる開発が増加している．これには手早く実装が行われるという利点がある．そういった開発においてもあいまいな仕様による要求とのズレや不具合は頻発しているのが現状である．実際に，著者らが所属している開発現場でも，仕様のあいまいさに起因する不具合が多い．

あいまいな仕様記述を改善していく方法の一つとして形式手法がある．しかし，フレームワークを用いた開発のように設計実装方法に関する制約が強い際に，それらの制約を仕様においてどれだけ記述方法に反映する必要があるのかは定かではない．

そこで，実際のフレームワークによる Web アプリケーションの開発を模した作業や，制約を考慮した仕様記述を行うことにより，VDM++言語とフレームワークを用いた開発における，形式手法の有効性と，適切な適用方法の検証と提案を行う．

2 章では目的を達成するための具体的な方法を述べ，3 章では実施結果，4 章では結果の考察，5 章では Web アプリケーションの仕様記述を行う際の提案、最後に 6 章で今後の展望について述べる．

2. 方法

はじめに，Web アプリケーションで馴染みの深いログイン機能と，そのための会員管理を行うアプリケーションに関するあいまいな仕様(図 1)を作成し，その仕様をもとに 3 つのアプローチで検証を行った．

(1) あいまいな仕様をもとに，VDM++言語で仕様記述を実施
VDM++言語を学習しながら，仕様のあいまいさの解消と，VDM++言語で記述した仕様記述をより見やすくするための検討を行う(アプローチ 1)．

(2) 実装を先行して行い，実装したソースコードから VDM++言語で仕様記述を実施
仕様記述者がフレームワークの制約を踏まえて，どのような点に留意して VDM++言語による仕様記述を行うべきかの確認・検討を行うため，仕様記述の後に実装，という通常の開発手順は踏まず，実装の後に仕様記述を行った．
実装に用いる言語およびフレームワークとして，著者らの使用経験が豊富な，Java/Struts2[2](アプローチ 2-1)と Ruby on Rails[3](アプローチ 2-2)を用いた．

会員管理として，以下の機能を提供する．

1. 会員登録機能
   (1) 一般ユーザーが会員として登録する機能．
   (2) 会員登録に必要な情報は，会員 ID，パスワード，性別，生年月日，メールアドレス．
   (3) 会員 ID はユニークとし，一度設定した会員 ID は変更不可．
   (4) 1 メールアドレスには，1 会員 ID のみ登録可能．一度登録したメールアドレスも変更可能．
2. ログイン機能
   (1) 会員は登録した会員 ID とパスワードを入力してログインが可能．
   (2) パスワードの有効期限が切れた場合はパスワードの変更を促し，パスワードを変更した場合のみ，ログインが成功する．
3. 会員情報変更機能
   (1) 会員が登録した情報の更新を行えることとする．ただし，会員 ID とパスワードは変更不可．
4. パスワード変更機能
   (1) 会員のパスワードを変更する．
   (2) パスワードを変更すると有効期限がリセットされる．
5. ログアウト機能
   (1) ログインした会員が任意にログアウトできることとする．
   (2) ログアウト後はログインに戻る
6. エラー発生時
   (1) ログイン画面に戻る．

図 1. 会員管理のあいまいな仕様

この作業を通して，VDM++言語の学習と，フレームワークによる開発で VDM++言語を利用することの利点や困難さ，有効性や運用方法の知見を得ることを行う．

3. 結果

今回試行したアプローチ 1，アプローチ 2-1，アプローチ 2-2 のうち、ここでは，Java/Struts2 によるアプローチ 2-1 の結果を詳細に述べる．

3.1. 実装環境

Java/Struts2 による実装を行うために準備した環境は表 1 のとおりである．

表 1. 実装環境で使用した製品

製品	説明
JDK 1.6	Java 開発キット
Tomcat6.0	アプリケーションサーバー
MySQL5	データベース管理システム(RDBMS)
Struts2	Java Web アプリケーションフレームワーク

3.2. Java/Struts による実装

表 2 に示す種類の成果物を作成した．

表 2. 成果物一覧

成果物	言語	説明
JSP	JSP	HTML を動的に生成するテンプレートファイル
フレームワーク設定ファイル	XML	画面遷移や画面項目の入力チェックを定義するファイル (struts.xml, validation.xml 等)
アクションクラス	Java	画面の入力フォームに対応するクラス 画面からの入力値を受け取り，ビジネスロジックを呼び出したのち，その結果を画面に出力するという処理を行う
ビジネスロジッククラス	Java	主にデータベース操作を行うクラス
データベース	SQL	-

なお，実装時に仕様があいまいで不明確な部分に関しては，これまでの経験や仕様の追加検討によって補った仕様が存在する．実装時に補った仕様の例は以下のとおり．

- 複数の会員が同時に操作した際に会員 ID やメールアドレスが重複するなどのデータの不整合が発生しないよう，同時実行性を考慮．
- メールアドレスの正確な仕様を反映．（大文字と小文字が違うだけのアドレスは同じとみなすなど）

3.3. VDM++による記述

表 2. 成果物一覧のうち，VDM++言語による仕様記述の対象としたのは，Java 言語で作成したアクションクラスとビジネスロジッククラスである．その際，Java 言語による記述をそのまま VDM++言語による表現に変換するように留意した．

各クラスにおける記述方法は以下のとおりである.

・アクションクラス(図 2)

アクションクラスは, ほぼ Java 言語の記述を一対一で VDM++の仕様記述に変換することができた(図 3). そのかわり, 画面入力値をアクションクラスのメンバ変数に設定したり, 画面遷移先を操作の戻り値で表現したり, Struts2 のフレームワークとしての設計方針に依存した仕様記述となった.

また, 事前条件・事後条件・不変条件の記述がなく, すべての操作が陽定義(操作の本体を手続きとして定義したもの)となった.

```
/**
 * 会員登録画面に入力された内容をセッションに保存します。
 *
 * @return
 */
public String entry() {
        final Member input = getInput();
        final MemberManager.Result result = getMemberManager().checkRegister(input);
        switch (result) {
        case DUPLICATE_MEMBER_ID:
                addFieldError("memberId", getText("member_exists"));
                return INPUT;
        case DUPLICATE_EMAIL:
                addFieldError("email", getText("duplicate_email"));
                return INPUT;
        }
        setTransactionData(input);
        return SUCCESS;
}
```

図 2. 会員登録画面の Java ソースコード

```
--
-- 会員情報入力
--
public entry : () ==> seq of char
entry () == (
        dcl input : Member := getInput();
        let result = getMemberManager().checkRegister(input)
        in (
                if result = <duplicate_member_id>
                then return INPUT
                elseif result = <duplicate_email>
                then return INPUT;
        );
        setTransactionData(input);
        return SUCCESS;
);
```

図 3. 会員登録画面の VDM++言語記述

・ビジネスロジッククラス

ビジネスロジッククラスは主に SQL を実行するコードで構成されている．SQL 文やデータベース制約をそのまま VDM++言語に変換することはできなかった．そのため，事前条件・事後条件・不変条件として仕様記述を行い，すべての操作は陰定義(操作の本体は定義せずに，実行前と実行後に成立している条件のみ定義したもの)となった(図 4).

```
-- 型定義
types

-- 会員情報登録結果
public RegisterResult = <success> | <duplicate_member_id> | <duplicate_email>;

-- 変数定義
instance variables

-- 会員テーブル
private members : inmap seq of char to Member := {|->};

-- メールアドレス一意制約
private emailToMemberId : inmap seq of char to seq of char := {|->};

-- 操作定義
operations

-- 会員情報登録
public register : Member ==> RegisterResult
register(member) == is not yet specified
post (
        (
          member.getMemberId() in set dom members~ and
          RESULT = <duplicate_member_id>
        ) or (
          member.getEmail() in set dom emailToMemberId~ and
          RESULT = <duplicate_email>
        ) or (
          members~ munion {member.getMemberId() |-> member} = members and
          emailToMemberId~ munion {member.getEmail() |->
          member.getMemberId()} = emailToMemberId and
          RESULT = <success>      )
);
```

図 4. ビジネスロジックの VDM++言語記述

4. 考察

今回，それぞれのアプローチで VDM++言語による仕様記述を作成した経験から，実際の開発における仕様書作成時に考慮すべき点や，そのときの仕様記述はどうあるべきかについて考察する．

4.1. VDM++言語を導入するメリット

- 仕様記述を VDM++言語で記述することにより，日本語で記述されているために混入するあいまいさ，暗黙の知識による属人性の多くを排除でき，開発者による仕様の誤解を低減できる．
- ビジネスロジッククラスから作成した VDM++言語による仕様記述は，要求仕様から作成する仕様記述に近い形となる．そのため，ビジネスロジッククラスの設計に先立って要求仕様が VDM++言語で記述してあれば，ビジネスロジッククラスの設計にかかる工数の削減を見込むことができる．

4.2. VDM++言語を導入するデメリット

- VDM++言語を習得するためには，少なからずコストが発生する．
- VDM++言語による仕様記述を作成するには，プログラミングに近いスキルが要求される．そのため，スキルの低い担当者が仕様記述を行った場合，フィージビリティが低い，あるいは実装が非効率になってしまう仕様記述を作成してしまう可能性がある(もちろんこの問題は VDM++言語に限った話ではない)．

4.3. VDM++を導入すべきか

昨今，Struts などのフレームワークを用いた Web アプリケーション開発は定型化が進んでいる．フレームワーク自体の標準化もさることながら，仕様書のフォーマットや書き方なども標準的なものが存在する[4]．今からこれらの標準を破棄し，VDM++言語のような汎用的な仕様記述言語を全面的に導入する必要性は低いと考える．しかし，標準的な仕様書の中には自然言語や疑似コードのような文法で記述している箇所がある[5]．それらの部分を VDM++言語に置き換えて記述することが妥当であると考えられる．

5. 提案

5.1. VDM++言語の導入範囲

実際に適用した経験によれば，画面の定義やデータモデルの定義は，画面入出力項目明細[5]やエンティティ定義書[6]等，これまでに使用してきたフォーマットをそのまま使用したほうがよい．ただ、画面入出項目明細の一部や、画面で発生したイベントに対する動作を定義する画面アクション定義[5]の記述は VDM++言語で代替可能であり，かつ，アクションクラスの実装に流用可能なため、VDM++言語による仕様記述を行うことの意義は大きい．その際，画面アクション定義にすべてを記述せず，要求仕様をもとに VDM++言語で作成したビジネスロジッククラスの操作を呼び出すようにすると効果的である．

5.2. VDM++言語の記述方法

VDM++言語での仕様記述は，言語を習得していない人から読むことへの抵抗感を持たれてしま

うことがある．実際，第 3 者に VDM++言語による仕様記述(図 5)を見せた際に「これではプログラムと変わらない」との感想を受けた．しかしながら，VDM++言語での仕様記述に日本語による記述を加え，再度，第 3 者に VDM++言語による仕様記述(図 6)を見せたところ，「仕様書として違和感が少なくなった」という評価を得た．

このことから，VDM++言語での仕様記述に日本語による記述を加えることは，形式手法で記述した仕様記述をより見やすくし，日本語による仕様記述に見慣れた人達にとっても効果的であることが判明した．

```
-- 会員情報登録
public register：Member ==> RegisterResult
register(member) == is not yet specified
post (
        (
          member.getMemberId() in set dom members~ and
          RESULT = <duplicate_member_id>
        ) or (
          member.getEmail() in set dom emailToMemberId~ and
          RESULT = <duplicate_email>
        ) or (
          members~ munion {member.getMemberId() |-> member} = members and
          emailToMemberId~ munion {member.getEmail() |->
          member.getMemberId()} = emailToMemberId and
          RESULT = <success>
        )
);
```

図 5. VDM++言語での仕様記述

```
-- 会員情報登録
public 登録する：『会員』 ==> 「会員登録結果」
登録する(会員) == is not yet specified
post (
        (
          会員.会員 ID 取得() in set dom 会員テーブル~ and
          RESULT = <会員 ID 重複>
        ) or (
          会員.メールアドレス取得() in set dom メールアドレス一意制約~ and
          RESULT = <メールアドレス重複>
        ) or (
          会員テーブル~ munion {会員.会員 ID 取得() |-> 会員} = 会員テーブル and
          メールアドレス一意制約~ munion {会員.メールアドレス取得() |->
          会員.会員 ID 取得()} = メールアドレス一意制約 and
          RESULT = <成功>
        )
);
```

図 6. 日本語の記述を加えた VDM++言語での仕様記述

6. おわりに

今回は 3 つのアプローチで，共通の仕様をもとに Web アプリケーションの開発を行い，それぞれで形式手法についての知見を得ることができた．今後はこれをさらに進め，共通の仕様であることを利用し，それぞれのアプローチの比較を実施したい．また，実際のプロジェクトに VDM++言語を導入し仕様記述のあいまいさを除去することで，後続フェーズにおける手戻りを減少させる，という VDM++言語導入の目的がどの程度実現できたか検証したい．

謝辞

本研究を進めるにあたり，ご指導を頂いた荒木教授，栗田主査，石川副主査に深謝する．また，研究会を運営していただいた日科技連の皆様，例会やメーリングリストでの議論を通じて多くの示唆を頂いた演習コースⅡの方々にもお礼を申し上げる．

参考文献

[1]石川冬樹(2011)『VDM++による形式仕様記述』(トップエスイーシリーズ 実践講座)
荒木啓二郎監修 近代科学社.

[2]The Apache Software Foundation『Apache Struts2』(2012/01/22 現在)
http://struts.apache.org/2.x/

[3]山田祥寛(2011)『Ruby on Rails 3 アプリケーションプログラミング』技術評論社.

[4] IPA/SEC 要求・アーキテクチャ領域 機能要件の合意形成技法WG(2010)
『機能要件の合意形成ガイド(ver.1.0)』(2012/01/22 現在)
http://sec.ipa.go.jp/reports/20100331/cbgfr_20100331v1.00_gaiyo_01.pdf

[5] IPA/SEC 要求・アーキテクチャ領域 機能要件の合意形成技法WG(2010)
『機能要件の合意形成ガイド(ver.1.0) 分冊３ 画面編』(2012/01/22 現在)
http://sec.ipa.go.jp/reports/20100331/cbgfr_20100331v1.00_gamen_01.pdf

[6] IPA/SEC 要求・アーキテクチャ領域 機能要件の合意形成技法WG(2010)
『機能要件の合意形成ガイド(ver.1.0) 分冊４ データモデル編』(2012/01/22 現在)
http://sec.ipa.go.jp/reports/20100331/cbgfr_20100331v1.00_datamodel_all.pdf