A-010

テストコード半自動生成を用いたプロトタイプ検査と VDM++ 妥当性確認のコスト削減

Prototype Testing using Semi-Automated Test-Code Generation for Cost Reduction of VDM++ Validation Process

森島 耕 †　　和崎 克己 ††
Koh Morishima　Katsumi Wasaki

1　はじめに

システム設計において，仕様は自然言語や表・図などを用いて記述されることが主流である．しかし，仕様自体の曖昧さや人的エラーの混入，それらに起因する欠陥発見の遅れによるコストの増大など多数の問題点がある．このような問題を取り除く有用な方法として形式手法がある．形式的な言語を用いてシステムモデルを構築し，仕様や設計の明確化・厳密化を行い，モデルを分析・検証することでシステムの高品質化が期待できる．仕様記述言語の一種である VDM++ で記述した仕様の妥当性は，援用ツールのモデル実行機能を用いて，プロトタイプのテストを行うことで確認できる．しかし，援用ツールを直接操作する際には専門の知識が必要であり，実際のシステム利用者とって相応しいものではない．

著者らは VDM++ 向けの分散モデル実行環境を開発している [1]．Java 向けに提供している API を用いることで，実行環境上にある VDM++ モデルをインタープリター実行できる．この API を組み込んだクライアントアプリケーションを GUI で構築することで，モデル実行が直感的に実施可能となる．また，このクライアントアプリケーションの実装で使われる Java Servlet のスケルトンコードを，UML アクティビティ図を用いた上位設計から自動生成するツールを開発している [2]．しかし，VDM++ コードや自動生成できない部分の Java Servlet の記述コストがかかる点や，手動での妥当性確認では十分なテストを行うことが難しく，コストがかかる点などの問題点がある．

上記の問題点に対し，本研究では VDM++ を用いた単純なイベント予約システムの仕様記述を出発点とし，飛行機席予約システムと列車予約席システムの仕様記述を行い, 予約に関するプロトタイピングのパターンを考察することで, 他の拡張予約システムのプロトタイピングにも本研究の VDM++ や Java Servlet のコード,UML 図を再利用できるようにした．また，予約システムに関する VDM++ や Java Servlet コードから必要情報を抜き出してテストコードを半自動生成し，十分な妥当性確認を半自動で行うことでコストの削減を目指している．

2　形式仕様記述言語 VDM++

2.1　形式手法と仕様記述言語 VDM++

形式手法とは，信頼性の高いシステムを作るために用いる仕様記述・設計開発・検証の技術である．形式手法の一種として形式仕様記述が存在する．形式仕様記述は理論に基づいた形で仕様を厳密，正確に書く．基本設計の段階でモデル実行を行い誤りを排除することで，手戻りによるコスト増大を防ぐことが出来る（図 1）．

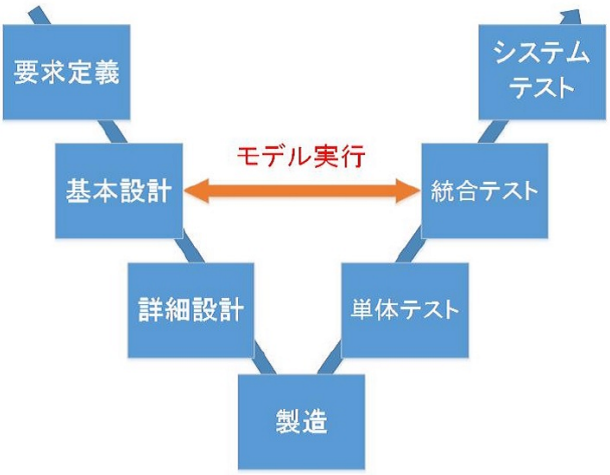

図 1　上流工程におけるモデル実行と統合テストの位置

形式仕様記述言語 VDM++ は，Vienna Development Method（ウィーン開発手法）の基盤言語 VDM-SL を，オブジェクト指向に基づいたモデル化を扱えるように文法を拡張したものである [3]．VDM++ には，不変条件，事前条件，事後条件を制約条件として記述可能な支援ツールが用意されており，これによって仕様を検査，テストすることが出来る．VDM++ には開発援用ツールが存在する．厳密さでは数学的証明を用いた定理証明系に劣るものの，専門的知識がなくとも仕様実行を用いて妥当性を確認出来る．

2.2　VDMJ

VDMJ は Java で記述された仕様記述言語 VDMSL, VDM++，VDM-RT（VDM++ の非同期リアルタイム向けシミュレーションモデル用拡張）のための VDM 開発援用ツールである [4]．構文解析器，型チェッカー，インタプリタ，デバッガ，および証明課題生成機能を持つ．Overture という VDM 向け Eclipse プラグインに用いられている．オープンソースであり，自由な改変が可能である．

3　関連研究

3.1　VDMJWeb サービス

VDMJWeb サービス [1] は，実際のインタープリタ機能を有する VDMJ モデル実行器，VDMJ のクライア

† 信州大学大学院理工学系研究科，Graduate School of Science and Technology，Shinshu University.
†† 信州大学工学部，Faculty of Engineering，Shinshu University.

ントとして外部と通信を行うための機能が備わっている VDMJC から構成されている（図 2）．VDMJ のクライアントである VDMJC は Apache Axis2 を用いて Web サービスとして公開される．Web コンテナとして Apache Tomcat を用いている．Web サービスの機能を利用するため提供されている Web API（VDMJCC）を用いて，クライアントアプリケーションは VDMJWeb サービスと SOAP 通信を行うことが出来る．

Web ブラウザから送られてくるリクエストは Tomcat 上の Java Servlet で受付け，Web API を介して VDMJ サービスへ転送後，モデルがステップ実行処理される．VDMJ のインタープリタによる処理結果は，Web API(VDMJCC) を用いた Java Servlet により Web ブラウザに表示される．その結果を見てユーザは妥当性確認を行う．ユーザが行った妥当性確認の結果を踏まえ，VDM++ を記述する開発者は再度 VDM++ を記述する．

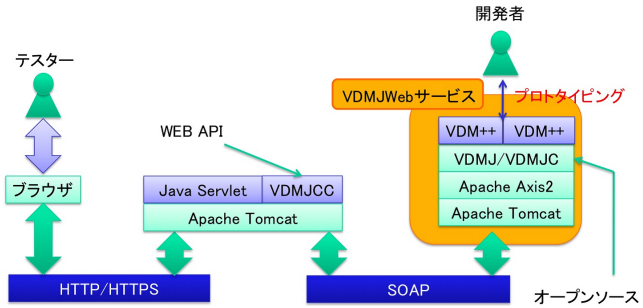

図 2　VDMJWeb サービスの構成

3.2　Java Servlet スケルトンコード生成器

Java Servlet スケルトンコード生成器とは著者らが開発している，UML アクティビティ図を用いて対象システムの画面遷移条件およびロジックフローを記述し，その成果物を用いて VDM++ で記述された仕様と一対一に対応した Java Servlet スケルトンコードを自動生成するツールである [2]．VDMJWeb サービスと通信を行う Java Servlet スケルトンコードを自動生成することで，プロトタイピングにおける記述コストを削減し，上位設計モデルの再利用性向上に寄与する．

3.3　イベント管理システムプロトタイピング

イベント管理システムは，イベントに対してユーザが予約をするという単純な予約システムとなっており，VDM++ を用いて仕様記述がされている．このシステムで扱う情報は，ユーザと部屋，イベントの三つである．機能としては，イベント登録やイベント予約，部屋追加，ユーザ追加などがある．

4　高速プロトタイピング手法

4.1　高速プロトタイピング手法の流れ

高速プロトタイピング手法では以下の流れでプロトタイピングを行う [5]．

1. パターンの決定
2. UML 図の記述
3. VDM++ コード記述と単体テスト
4. Java Servlet 自動生成と結合テスト

パターンの決定では，新しく作成するシステムにおいてどのような仕様が要求されていて，どのような機能が必要で，従来あるプロトタイプのどの部分が再利用可能なのかを決定する．UML 図の記述では，従来のクラス図において再利用可能なクラス・メソッドや不必要なクラス・メソッドを選択し，クラス名やメソッド名の変更を行う．従来のものだけでは不十分な場合は，クラス・メソッドを追加して新たな仕様に対応する．また，Java Servlet スケルトンコードを生成するために必要な場合はアクティビティ図を記述する．VDM++ コード記述と単体テストでは，クラス図に則って VDM++ コードの記述を行い，援用ツールで実行し，単体テストを行う．Java Servlet 自動生成と結合テストでは，ツールを使用して Java Servlet スケルトンコードを生成し，VDMJWeb サービスを用いて結合テストを行い妥当性を確認する．

4.2　高速プロトタイピング手法の適用例

高速プロトタイピング手法のケーススタディとして，次の 3 ステップによる仕様追加に伴う上位設計を試みた: (1) 既存の単純なイベント管理システムを出発点とする (2) イベント管理システムをもとに，飛行機席予約システムを考える．仕様の差異は，席のグレードが存在する図 1 上流工程における妥当性確認の全体フロー点と予約対象に期間がある点である．(3) 飛行機席予約システムをもとに，列車席予約システムを考える．仕様の差異は，予約情報に区間が存在する点である．

変更行数について評価を行った結果，イベント管理システムの VDM++ の全体行数 1,375 行に対して，それぞれ新しく追加した VDM++ の行数は 80 行と 71 行であった．飛行機席予約システムの主な変更点は，予約対象に期間という概念が追加されたことによる属性の追加と，予約情報を持つ属性に新たに便，日付，ランクの情報を関連付けた点であった．列車席予約システムの主な変更点は，停車駅のリストを持つ属性を追加した点と，予約情報を持つ属性に新たに区間情報を関連付けた点と，予約可能な席を取り出す処理部分の変更であった．

4.3　結果

単純なイベント管理システムの仕様記述から，拡張機能を持つ予約システムの仕様記述を行った．これにより，乗り物全般の予約システムについては本研究のプロトタイプを再利用することで高速，低コストでプロトタイピングを行うことが可能となった．

5　テストコード半自動生成

5.1　テストコード半自動生成を用いたプロトタイプ検査

テストコード半自動生成による妥当性確認の全体像を図 3 に示す．本研究では，VDM++ コードや UML 図から必要な情報を抜き出してテストコードを半自動生成し，十分な妥当性確認を行うことを目指している．テストコード半自動生成を用いたプロトタイプ検査の流れとしては，まず記述した UML 図からテストの流れを，それを元に VDM++ コードから自動生成に必要な情報を抜き出す．次にそれらの情報を元にテスト回数や，テスト方法などの書かれた独自のスクリプトコードを半自動

生成し，必要部分は手動で追記する．最後に，形式化したテストシナリオから，Mock 機構を有する Java 実装コードを変換器によって得る．現状としては，半自動生成を行うテストコードの枠組みの考察を行った．また，独自のスクリプトコードを考案し，そのスクリプトコードから Java テストコードへの変換器を作成している．

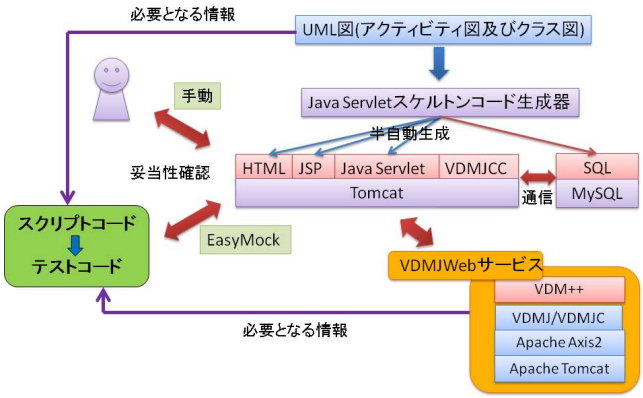

図 3　テストコード半自動生成による妥当性確認フローの概要

5.2　EasyMock

EasyMock は Java 用のモックオブジェクトライブラリの一つである [6]．モックオブジェクトとは，擬似的なオブジェクトを作り、本物のオブジェクトに成り代わりテストを行うものである．モックオブジェクトにテスト中の振る舞いを設定することで，テスト対象と依存関係にある相手が未実装であったり，テスト環境から接続不可であってもテストを行うことが可能となる．

5.3　Java テストコード

今回考察した最終的に半自動生成される，予約システムプロトタイプ向けの Java テストコードは，大きく四つの部分に分けられる．まず一つ目は，package 文や import 文などのプロローグ部分である．二つ目は，変数宣言部分である．ここでは，VDM++ コードから抜き出した情報を元にしたユーザリストやカレンダーオブジェクトなどを用意する．三つ目は，テストのシナリオを書く部分である．ここでは，予約を繰り返し行うようなテストを行うため繰り返し分の入れ子構造となる．四つ目は，モックに関する部分である．半自動生成される Java テストコードによるプロトタイプ検査は，EasyMock を利用して行うため，そのために必要となる記述を行う．

5.4　スクリプトコード

UML 図や VDM++ コードから必要情報を抜き出し，それらの情報を元に半自動生成するテスト回数や，テスト方法などの書かれた独自のスクリプトコードを考案した．予約システムの検査項目を表 1 に示す．提案手法であるスクリプトコードの文法と，検査項目に基づいたテストスクリプト例を，それぞれ List1，2 に示す．

表 1　予約システムのテスト項目（例）

便（Flight）	JL002（便名・固定）
クラス（Rank）	economy（エコノミークラス・固定）
搭乗日（Date）	2016 年 6 月 1 日〜6 月 3 日 （+2 日分オーバヘッド含む・列挙）
予約者名（userID）	⟨Asan, Bsan⟩（ユーザ名・列挙）
予約席数（number）	1 席分〜9 席分（+3 席オーバヘッド含む・乱数）
テスト用イテレータ：	Flight(FIX) 〉 Rank(FIX) 〉 Date 〉 userID 〉 number 〉 reservationCheck(args)

List 1　スクリプト言語の文法

```
Script →  Pre (Variable)+ Test Mock

Pre → (Pack)? Imp Class
Pack → "package" Identifier ";"
Imp → "import" "normalpackage" ";"
Class → "class" Identifier ";"

Variable → "begin" "variable" "{" ( Vari | Vdmjc
     ";" | VdmjcStub ";" | Testp ";")+ "end" "
     variable" "}"
Vari → (VarDecls | ListDecls | Identifier | "Date"
     Identifier) (Modif)? (Initializer)? ";"
Vdmjc → "Vdmjcc" Identifier "="  "(" Identifier ","
      "\"" Identifier "\"" "," "\"" Identifier "\""
     ")"
VdmjcStub → "Vdmjcstub" Identifier "=" Identifiersp
Testp → "TestProto"
VarDecls → TypeID Identifier
ListDecls →
      "List" "<" ("String" | "int") ">" Identifier
TypeID → "String" | "Int" | "Test"
Modif → "with_fix" | "with_overhead"
Initializer → "=" (Identifier | Identifiersp |
     VarExp | Functions | List)
Functions →
      Function "(" Identifier "," Identifier ")"
Function → "Stringplus" | "Dateplus" | "Numplus"
List → "[" (Identifiersp ("," Identifiersp)* |
     VarExp ("," VarExp)*) "]"

Test → "begin" "test" "{" (Stmt)+ "end" "test" "}"
Stmt → Identifier  "=" "numrandom" "(" Identifier
     "," Identifier ")" ";"
        | "for" "(" Identifier ";" Identifier ";"
            Identifier ";" Exp ")" Stmt
        | "foreach" "(" Identifier ":" Identifier
            ";" Option ")" Stmt
        | "if" "(" Cond ")" Stmt ("else" Stmt | ε)
        | "print" "(" "\"" Identifier "\"" ")" ";"
        | Identifier "=" Exp ";"
        | Identifier "=" Identifier "(" Arg ")" ";"
            //rc, rcc
        | "{" (Stmt)* "}"
Option → "asc" | "desc" | "random"
Arg → ("String")? Identifier ("," ("String")?
     Identifier)*
Cond → Exp CompOp Exp
CompOp → "=" "=" | "!" "=" | ">" | "<" | ">" "=" |
      "<" "="
VarExp → VarTerm ("+" VarTerm | "-" VarTerm)*
VarTerm →
      VarFactor ("*" VarFactor | "/" VarFactor)*
VarFactor → "(" VarExp ")"
        | "+" VarFactor
        | "-" VarFactor
        | Number
Exp → Term ("+" Term | "-" Term)*
Term → Factor ("*" Factor | "/" Factor)*
Factor → "(" Exp ")"
        | "+" Factor
        | "-" Factor
        | Number
        | Identifier

Mock → "begin" "mock" "{" (Mockfuns)+ "end" "mock"
     "}"
Mockfuns → Identifier "(" Arg ")" "{" Mockfun "}"
Mockfun → ("(" Identifier ")")? Identifier "("
     Identifiersp ("," Identifiersp)* ")" ";"

Identifier → Letter (Letter | Digit)* | "_" (Letter
      | Digit)*
Identifiersp → "\"" (Letter | Digit)+ "\""
Number → (Digit)+
```

List 2　スクリプトコード例

```
package Flight;
import normalpackage;
class FlightCode;

begin variable{
 ...
 String flight with_fix = "JL002";
 String rank with_fix = "economy";
 String userID;
 String date;
 String number;
 List<String> userID2List with_overhead;
 Date date3 with_overhead;
 Int number2 with_overhead;
 List<String> userIDList = ["Asan", "Bsan"];
 Date date1 = [2016,6,1];
 Date date2 = [2016,6,3];
 Int dplus = 2;
 date3 = datePlus(date2, dplus);
 Int count = 50;
 Int max = 9;
 Int min = 1;
 Int cplus = 3;
 number2 = numPlus(max, min, cplus);
 Int rc;
 Int rcc;
 ...
end variable
}

begin test{
 for(date; date1; date3; 1){
  foreach(userID : userIDList2 ; asc){
   number = numRandom(number2, min);
   rc = reservationCheck(stub, obj, userID, flight,
        rank, (String)date, (String)number);
   if(rc < 0){
    ...
   }
   if(rc == 1){
    rcc = reservationConfirmationCheck(stub, obj,
         userID, flight, rank, (String)date, (String
         )number);
    if(rcc >= 0){
     print成功("");
    }
    else{
     print失敗("");
    }
   }
  }
 }
end test
}

begin mock{
 init(stub, obj){
  ...
 }
 ...
end mock
}
```

（1）表 1 は，テストコード半自動生成の対象とする飛行機席予約システムで想定されるテスト項目の例である．飛行機席予約システムのテストでは，便名（Flight）とクラス名（Rank）を固定して，指定する日付とユーザ名のすべてのパターンの予約を繰り返す．今回は 6/1〜6/3 の日付と，Asan，Bsan というユーザを用意してそのすべての組み合わせのパターンについてテストを行うことを表している．

（2）List1 は，テストケースの記述に使用するスクリプト言語の文法である．このスクリプト言語は大きく，import 文などのプロローグ部分，変数宣言部分，テストを行う部分，Mock 部分の 4 つのパートに分かれている。文法ではそれぞれ，Pre，Variable，Test，Mock の順に並び Script を構成している．また，“/*” から “*” までの間と，“//” から行末まではコメントとして無視する．

（3）List2 は，List1 の文法をもとに記述した，半自動生成を目指すスクリプトコード例である．スクリプトコードでは，まずパッケージ名，クラス名と import 文の記述を行う．次の “begin variable” から始まり “end variable” で終わる部分では，変数宣言を行う．使用できる型は，String 型，Int 型，Date 型，List 型である．スクリプトコード内で使用する変数はすべてこの部分で宣言しなければならない．また，“with_fix” と “with_overhead” は変数を修飾し，その変数が固定値か可変値なのかを表している．次の “begin test” から始まり “end test” で終わる部分では，実際に行うテストのシナリオを手動で記述する．今回は対象として予約関係システムのプロトタイプを想定しているため，予約が段々と埋まっていくようなテストを行う必要がある．そのため，このスクリプトコードでは予約を行うユーザと予約する日付を変化させ，繰り返し予約を行えるよう for 文と foreach 文に対応した．この例では，6/1〜6/3 の日付と，Asan，Bsan というユーザを用意してそのすべての組み合わせのパターンについてテストを行っている．また，if 文での分岐と print 文にも対応した．次の “begin mock” から始まり “end mock” で終わる部分では，EasyMock を使用する上で必要な情報を記述する．

6　まとめと今後の課題

高速プロトタイピング手法の適用例として，単純なイベント予約システムをもとに，拡張機能を持つ予約システムのプロトタイピングを行った．これにより，本研究のプロトタイプを再利用し，乗り物全般の予約システムの高速，低コストでのプロトタイピングが可能となった．今後，拡張機能のパターンを追加していくことで，様々なシステムのプロトタイピングを高速，低コストで行えると考える．また，この予約関係のプロトタイプに対して，繰り返し行うような妥当性確認にかかるコスト削減とより正確な妥当性確認を目指し，テストコード半自動生成を用いたプロトタイプ検査のシステムを考案した．このシステムで使用するスクリプト言語を考え，テストコードへの変換器を作成している．

今後の課題としては，予約以外のプロトタイピングのパターンの追加と，テストコード半自動生成を用いたプロトタイプ検査において，VDM++ コードや UML 図から必要情報を抜き出してスクリプトコードの雛形を自動生成する部分のシステムの作成を行う．

参考文献

[1] 村林 慧，多田 圭佑，和崎 克己：VDMJ と Apache Axis2 を用いた上流工程におけるモデル実行環境の構築，第 13 回情報科学技術フォーラム，(FIT2014)， 2014.

[2] 多田 圭佑，和崎 克己：VDM++ 分散モデル実行環境を用いたプロトタイピングと非同期 UI への適用，第 14 回情報科学技術フォーラム，(FIT2015)， 2015.

[3] 石川冬樹：VDM++ による形式仕様記述，近代科学社，2011.

[4] VDMJ–Free Development software downloads at SourceForge.net，http://sourceforge.net/projects/vdmj/

[5] 森島 耕， 和崎 克己：UML 図と VDM++ コードの再利用による高速プロトタイピング手法，平成 27 年度電気関係学会東海支部連合大会講演論文集， 2015.

[6] EasyMock：http://easymock.org/