2B-01

モデル駆動開発とテスト駆動開発を統合する仕様記述方法の提案

濱野 義満† 銀林 純‡
富士通株式会社† 富士通株式会社‡

１．はじめに

これまで，ソフトウェアの品質を向上させる設計手法として，モデル駆動アーキテクチャー（Model Driven Architecture 以下 MDA と略す）[1]が取組まれてきた．しかし，組込系など限られたドメインを除き，MDA による設計手法はまだ完成しておらず，広範な適用については懐疑的な意見も多い．

また最近，別の開発手法として，より実践的なテスト駆動開発（Test Driven Development 以下 TDD と略す）[2]も注目されており，アジャイル開発等への活用が進んでいる．TDD は，ソフトウェアに期待されるテスト仕様を先に明確にし，それを満足するコードを開発するという，MDA とは全く対極にある開発手法といえる．

今回，MDA を基本として，ソフトウェアの「機能仕様」を仕様記述言語で記述した結果，ユーザの「操作シナリオ（テスト仕様）」も同じ仕様記述言語で記述できることを見出した．同じ言語体系で統一的に記述することにより，MDA と TDD の双方のメリットを活かし，ソフトウェアの品質を向上させることが期待できる．

２．課題

MDA と TDD はアプローチが異なるため，これまで別々の開発手法として扱われることが多かった．MDA と TDD の課題について述べる

2.1. MDA の課題

MDA は，機能仕様（ソフトウェアの要求仕様）と制御（ソフトウェア・アーキテクチャ）を分離することで，機能仕様の独立性を高める開発手法である．さらに，機能仕様からソースコードを全て自動生成することを最終ゴールにしているといえる．

一方で MDA は，設計した機能仕様に対するテスト手法についてあまり言及していない．そのため，テストの手法は，経験的・実践的な取組みが行われているにとどまる．例えば，「機能仕様」の構造に着目したホワイトボックステストなど，基本的に「機能仕様」通りに，システムが動くかどうかというテストを行うのが一般的である．この方法では，テストで機能仕様自身のバグを検出することは，原理的に難しい．

2.2. TDD の課題

TDD は，ソフトウェアの機能として求められる「テスト仕様」，言い換えると「ユーザの操作によって，システムから得られる結果」を，先に定義することで，それを満たすソフトウェアを開発するというアプローチである．

TDD では，テスト仕様（テストコード）とそれを満足するプログラムコードのみが存在し，機能仕様書は存在しない．TDD は，テスト仕様がソフトウェアの仕様であるという立場をとる．

しかし機能仕様書は，エンタプライズ向けのソフトウェア開発において，ユーザのレビュー等に必須であり，結局作成が必要になるので，TDD を実際に採用する場面は，限られてしまう．

３．機能と操作の仕様記述方法

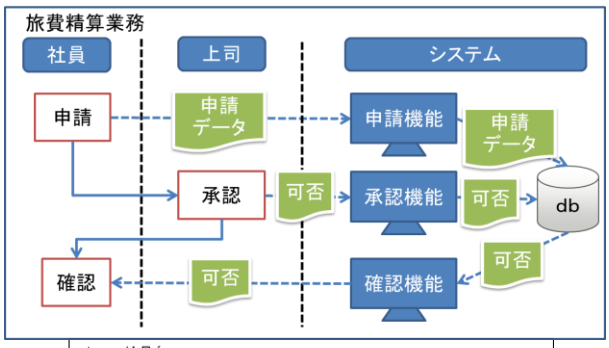
我々は前論文[3]で，業務フローに含まれる本質的な仕様の情報を,仕様記述言語で記述できるか試みた．その結果，ソフトウェアの機能仕様と利用者の操作シナリオを，同じ言語系で記述することができた．

具体的には，オブジェクト指向言語である Java を形式的に仕様記述言語（この表記を以降，**Java 形式表現**と呼ぶ）として用いた．図 1 は，Java 形式表現により，システムの「機能仕様」とユーザの「操作シナリオ（テスト仕様）」を，統一的に記述した例である．さらに図 2 は，機能仕様をシステムモデルとデータモデルの 2 層に，操作仕様をユーザモデルとメンタルモデルの 2 層に，全体として 4 層に分けて記述した例である．

４．効果

従来，設計書に記載される内容は，表形式の情報以外に，日本語のなどの自然言語での記述

Methodology of the Formal Description Unifying MDA and TDD
† YOSHIMITSU Hamano, FUJITSU LIMITED
‡ JUN Ginbayashi, FUJITSU LIMITED




```
class 社員{
 void 申請(){申請データ s; システム.申請画面(s);}
 void 確認(){boolean[] b = システム.確認画面();}
}
class 上司{
 void 承認(){boolean b; システム.承認画面(b);}
}
class 業務シナリオ{
 void 旅費精算業務(){社員.申請(); 上司.承認(); 社員.確認();}
}
class システム{
 申請データ[] db;
 boolean[] rst;
 void 申請機能(申請データ s){db.add(s);}
 void 承認機能(boolean s){rst.add(s);}
 boolean[] 確認機能(){return rst;}
}
class 申請データ{}
```

図1. 業務フローとJava形式表現の例

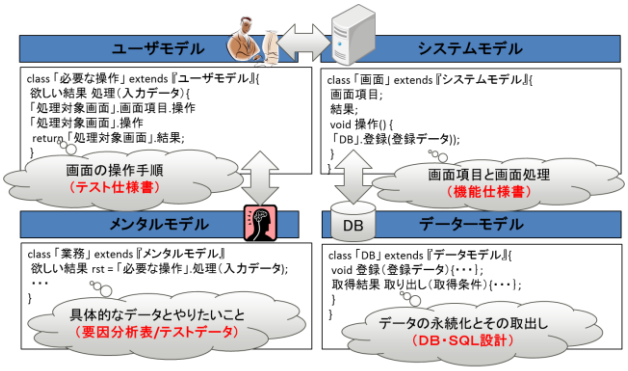


図2. Java形式表現による4層モデルの例

が少なくない．そのため，設計内容の整合性を自動でチェックできたとしても表形式等の定式化された部分に限られ，自然言語による記述部分は，人手によるレビューに頼らざるを得ない．従って，設計の品質を確保するために，多くの時間とコストが必要である．

仕様を Java 形式表現で記述することで，マシンリーダブルな形式となり，整合性のチェックや仕様のテストが可能となる．詳細について以下で述べる．

4.1. 整合性のチェック

Java 形式表現で記述された仕様の情報は，形式的にコンパイルが可能である．この際，文法的なチェックを自動で行うことができる．さらに命名規約のような業務的なルールについても，容易に追加可能である．自然言語による記述を許す限り，小さなミスや曖昧な記述を排除することは難しく，それを発見するには人によるレビューが必須である．この自動チェックにより，レビュー工数を削減するとともに，早期に仕様のバグを発見し，手戻りによるコストの増大を防ぐことができる．

4.2. 仕様アニメーション

コンパイルにより，Java 形式表現で記述された仕様から，実行可能な Java バイトコード（クラスファイル）を生成することができる．これを実行することは，仕様アニメーション（仕様のテスト）を行っていることに他ならない．

さらに，図 2 で示した 4 層のモデルを採用すれば，操作シナリオ側は，操作手順（ユーザーモデル）とテストデータ（メンタルモデル）が分離された形となり，従来のテスト設計の手法が活用できると考えられる．

この仕様アニメーションにより，大部分の仕様バグを，設計段階で検出することができる．

５．おわりに

ソフトウェアの「機能仕様」と利用者の「操作シナリオ」を，Java 形式表現で統一的に表記できることを示した．それにより，文法チェック，仕様アニメーションが可能となる．

機能と操作を同時に記載することは，これまで経験的に，機能設計とテスト設計を表裏一体で同時に進め，設計品質を高める取組み（W モデル等）と一致する．

今後，機能仕様のモデル変換を実現することで，プログラムの自動生成が期待される．また，このモデル変換は，操作シナリオからテストコードへの変換と，論理的に同じであり，テストコードも自動生成できる可能性が高い．

但し課題として，仕様が Java 形式表現で記述されることで，可読性が悪くなるという点が挙げられる．これについても論文[3]で示したように，利用者が見慣れた設計書のビューに投射することで，解決できると考えられる．現在筆者らは，実現に向けた検討を継続中である．

参考文献

[1] Object Management Group , MDA（Model Driven Architecture）, http://www.omg.org/mda/

[2] Kent Beck・長瀬嘉秀，「テスト駆動開発入門」，ピアソンエデュケーション, 2003

[3] 濱野義満, 銀林純：業務フロー図に含まれる仕様を形式表現する手法の提案について, 情報処理学会関西支部大会研究報告, B-04, 2014