B-012

ビジネスロジックモデリングの アプリケーション開発フレームワークへの依存性分析
Dependency Analysis for Application Development Framework of Business Logic Modeling

奥田 博隆† 小形 真平‡ 松浦佐江子†
Hirotaka Okuda Shinpei Ogata Saeko Matsuura

1. はじめに

近年，Android に代表される複数の携帯端末プラットフォームの普及により，アプリケーション開発フレームワーク（以下，単にフレームワーク）の幅が拡大した．その結果，ユーザが同一サービスを複数種類の端末から実行する形態が増えてきた．そのような同一サービスでは，ビジネスロジックを共通な要求とし，端末の種類に応じて異なるフレームワークに対応させたアプリケーションを開発できる事がビジネスロジックの再利用性の観点から望ましい．

我々は業務系 Web システム開発を対象に要求定義の理解性向上や顧客側からの妥当性確認支援を目的として Web UI(User Interface)プロトタイプ生成可能なモデル駆動要求分析手法を提案してきた[1][2]．本手法では，特定のフレームワークに特化した UI で解釈されるインタラクション部分と特定のフレームワーク非依存に再利用可能な部分を分離する為に，入出力データとシステムのエンティティを分離したモデル(要求分析モデルと呼ぶ)を構築する．しかし，顧客が，この要求分析モデルから自動生成される最終プロダクトを模したプロトタイプにより，Web UI を通じて要求を満たしているかを確認するため，開発者が定義する要求分析モデルのインタラクション部分は Web UI に特化したモデルになっている可能性がある．

Web UI に限らず，同一サービスを容易に異なるフレームワークに適応する為には，要求分析モデルに定義されるビジネスロジックがフレームワーク非依存に定義できているかを分析する必要がある．

本稿では，「長方形エディタ」を題材として Web UI を対象に分析した要求分析モデルを基に Java 言語で実装される Android，Java Applet 対象としたアプリケーション開発を実験し，提案手法の要求分析モデルにおけるビジネスロジックのフレームワークに対する依存性を分析し，端末の種類に応じて異なるフレームワークに対応させたアプリケーションを開発時の問題点を議論する．

2. 実験題材「長方形エディタ」の仕様

長方形エディタの要件は以下の通りである．幅及び高さを持つボードを定義し，ボード上に長方形を配置できるものとする．

- 長方形の作成ができる．但し，長方形は 10 個まで．
- 作成した長方形の移動ができる
- 作成した長方形の削除ができる

上記のサービス時に満たすべき条件は以下の通りである．
(1)ボードの幅や高さの上限を超えて長方形は存在できないものとする．
(2)同一の長方形は存在できないものとする．
(3)線分や点は長方形として存在できないものとする
長方形エディタのビジネスロジックとは，作成，移動の際に(1)から(3)の条件をみたすようにさせることである．

3. 要求分析モデル

ユースケースをユーザの利便性や柔軟性の観点から分割・統合した単位をサービスとして考え，アクタとシステムとのインタラクションの間に要求が顕在すると考え，振舞いとデータを数種の UML(Unified Modeling Language)を用いる事で分析・定義した手法を提案してきた．

同一サービスを容易に異なるフレームワークで適応する為には，要求分析モデルがシステムとの入出力のデータ項目を Web UI を基準として分析できる事が有用であると考えられる．

振舞いは，UMLアクティビティ図によって定義される．データの構造は，UML クラス図を用いて定義され，それらの具体値を表すデータとして UML オブジェクト図を定義する．今回の題材である「長方形エディタ」のサービスの一つである「長方形の作成」を例としたアクティビティ図を示す．(図 1)

アクティビティ図の記述方針は，特定のフレームワークで解釈する部分とそうでない部分を分離することを目的として 3 つのパーティションに分離する．アクタの行為であるデータ入力や動作の選択や入力データの構造をユーザパーティションで記述し，入力データの検査やユーザに対する通知やシステムに対するデータ構造の変換をインタラクションパーティションで記述し，ビジネスロジックとそれに関わるデータをシステムパーティションに記述する．

次に，「長方形の作成」で使用するデータ定義として，システムパーティション内に登場するオブジェクトノードやアクションの目的語を entity としクラス図に定義する（図 3 メソッドが定義されない状態のクラス）．システムパーティションに対する入出力構造は，ユーザパーティションの入出力構造と対応付く様にクラスに定義する(図 4 の左のクラス)．特に，システム内部のデータ構造である entity は要求分析段階で明らかになり，entity となる指標はシステム内部にアクション中に目的語として登場するオブジェクトノードの型やクラス名が該当する．

4. 要求分析モデルから設計モデル

4.1. 要求分析モデルが実装技術に依存している可能性

要求分析モデルが実装技術に依存する可能性があり得る，その可能性が見られた箇所を示す．

† 芝浦工業大学大学院理工学研究科電気電子情報工学専攻，Shibaura Institute of Technology, Graduate School of Engineering, Electrical Engineering and Computer Science
‡ 芝浦工業大学大学院工学研究科機能制御システム専攻，Shibaura Institute of Technology, Graduate School of Engineering, Division of Functional Control Systems

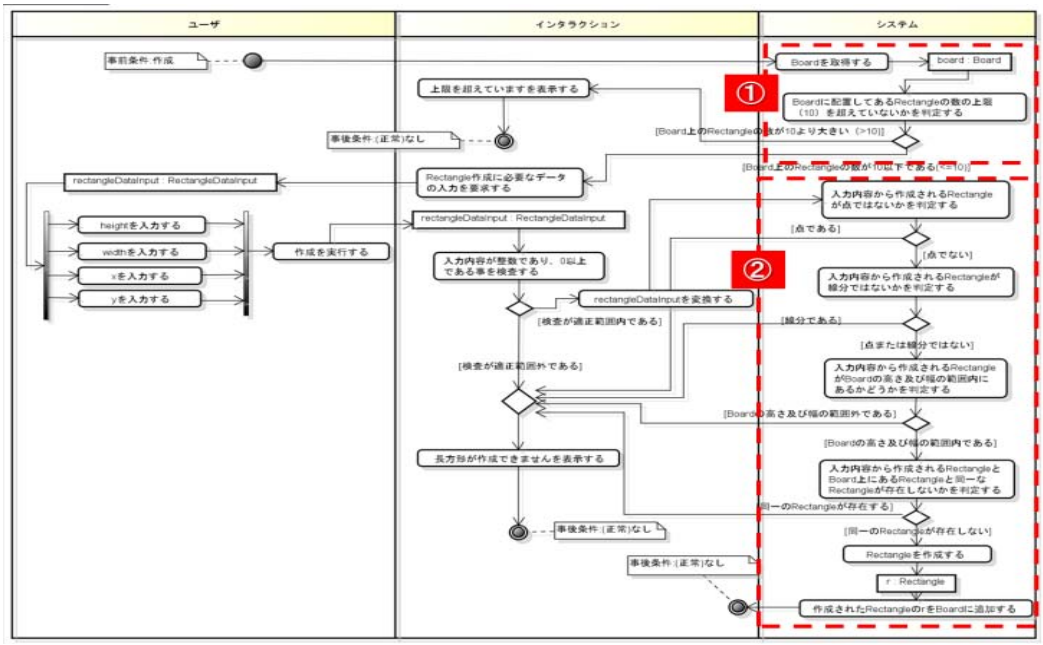

図 1：「長方形を作成する」サービスのアクティビティ図

長方形エディタにおいて長方形の閲覧を行い，次のサービスを決定するサービスがある．そこで，Web UI プロトタイプを意識した出力構造を要求分析モデルに定義すると，図 2 のような出力を得る．

一方，集合からある要素を一つ特定する方法として，Android 等ではグラフィカルに描画を行い，その長方形を選択する（タッチする，クリックする）という方式があり得る．ある要素を特定する方法として，画面上の座標として 1 点がシステムに対する入力となる．

図 2 に示す様に長方形を一覧の中からラジオボタンで選択するという選択方式は，入力構造のクラス定義にはシステムへの入力となるインデックス情報が明示されないが，開発者は，アクティビティ図のユーザパーティションに定義した「～を単数選択する」アクションが Web UI のラジオボタンに変換される事を知っている為，Web UI に特化している事が言える．

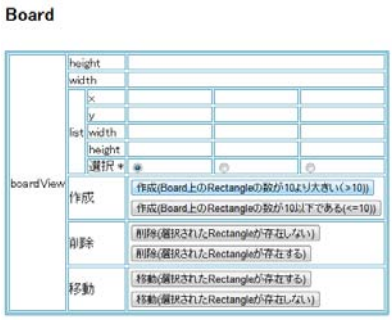

図 2：長方形の閲覧サービス

4.2. フレームワークに共通したビジネスモデルの設計

本稿では，Java 言語フレームワークを選択したので，ソースコードを再利用する事が可能である．そこで，再利用を行う為に，両方のフレームワーク特有の構造に非依存な様に要求分析モデルからビジネスロジックを定義した．まず，サービス毎にシステムパーティションに登場するアクションに注目して，entity にあるクラスにロジックを割り振った．割り振った根拠はアクションとして目的語をクラス名（オブジェクトノード名の型）が入る場合や対象となるクラス（○○に～）名である．図 3 の様になった．

次に，サービス全体を成立させるために，サービスを提供するクラスとしてそれぞれのロジックを統括するクラスが必要である．具体例として，長方形エディタの要求分析モデルで，「長方形を作成する」サービスを元に説明する．アクティビティ図のシステムパーティションへの入力フローと出力の正常フローのアクションとオブジェクトノード群を 1 つの単位として，それをプロセスという．図 1 では点線部の①と②に相当する．そのプロセスのアクティビティ図の出現順をプロセス間の実行順序と捉え，その単位をひとつのサービスとして捉える．図 1 では，①を行ってから②が行われるという意味で，①と②をあわせて 1 つのサービスとして捉える．

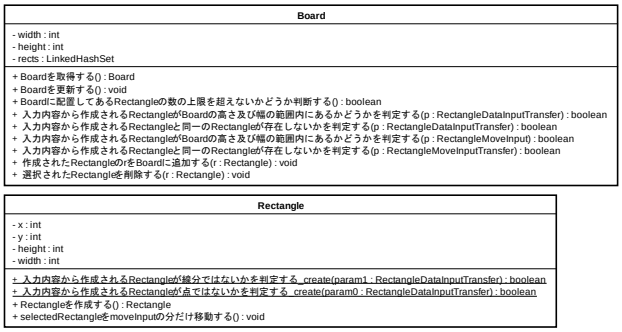

図 3:entity にビジネスロジックを割り振ったクラス

また，要求分析モデルの前提として，サービ中に登場したオブジェクトノードは一度出現したものは後でも使えるという前提をおいている．その為それに対応した設計が必要である．

今回は，プロセスやサービスの統括を表すクラスとして「Rectangle Editor」というクラスを作成した．プロセス毎メソッドとして定義し，その中で Entity に割り振られたメソッドを呼ぶという設計にした．このプロセスは正常

フローを元に定義されるので，例外フローを例外フローの数分だけ例外クラスを定義し，プロセス毎に呼ばれるメソッド内で例外処理を行う設計とした．

4.3. Android に依存した UI の設計と実装

Android では，画面と状態を持つクラスとして Activity というクラスが用意されている．そこで，要求分析モデルで定義される入出力構造である RectangleDataInput（図 4 左のクラス）クラスや BoardView（図 5 左のクラス）クラスを Activity として用い，属性をその上の UI 要素として設計し，実装する方針とする．今回は Web UI プロトタイプの画面遷移と同様の画面遷移を実装する．

まず，RectangleDataInput という入力構造の属性は Android フレームワークの UI 要素である EditText や Button として対応付くので以下のとおりに実装した(図 4).

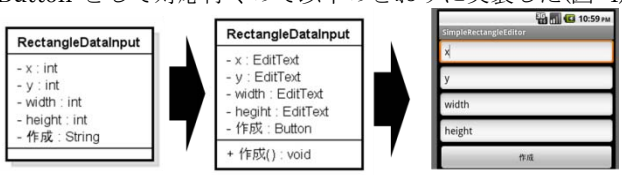

図 4：入力構造を Android の UI と対応

次に出力構造の属性である「作成」，「移動」，「削除」は Android フレームワークの UI 要素の Button として対応付く．しかし，クリックした座標 X と Y は，画面をタッチ（クリック）した結果得られる座標であり，EditText に対応させる事はできない．また，長方形の描画領域を表す為には，Android では，View クラスを用いる必要がある．そこで，View クラスを継承した RectangleBoardView クラスを定義し，BoardView クラスの属性とした．それを以下のように実装した（図 5）．

図 5：出力構造を Android の UI と対応

ビジネスロジックと UI をどのように繋げるかが課題となる．特にビジネスロジックへのデータ受け渡しという点については，詳しく 5.2 節で解説する．ボタン等のトリガが実行されると，UI に入力された要素をビジネスロジックの入力として変換をかけてビジネスロジックに渡す．

4.4. Applet に依存した UI の設計と実装

4.3 節と同様に，入出力データ構造を Applet フレームワーク固有の UI に設計する．今回，Android では Web プロトタイプの画面遷移通りに実装を行ったが，Applet では，使用可能な画面サイズやフレームワークの画面遷移の実現容易性の観点から，1 つで長方形エディタのサービスを実現しようと考えた．

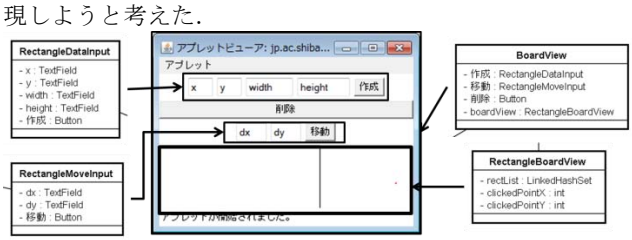

図 6：Applet と入出力構造の対応関係

そこで，Applet を継承した BoardView をクラスとして定義し，Panel を継承した RectangleDataInput をクラスとして定義し，Applet の UI 要素である AWT の要素と対応付くものを夫々 Button や TextField に変換した．その一方で，長方形の描画領域については Canvas を継承したクラスである RectangleBoardView を定義し，実装した．実装と設計モデルとの対応関係を図 6 に示す．

5. 考察

5.1. ビジネスロジックと入力データとの依存性分析

4.1 で述べた様に，要求分析モデルでは顧客との要求を確認する為に Web UI を想定する為に，システム内部で想定されるデータ構造に対する概念とシステムへの入力として与えられるデータ構造に対する概念がフレームワーク間で異なる可能性が考えられる．例えば，あるリストから単数選択し，それをトリガとしてページ遷移する場合，ページ間で情報を受け渡す必要がある．そこで，受け渡される情報は Web では一般に選択された要素をインデックスとして渡す．一方で，描画領域で表現する事が望ましい場合は，Android や Applet では，選択された要素を座標として渡す．ビジネスロジックにとっては，インデックス情報と座標情報のどちらともシステムにとって意味のある情報に変換してロジックを実行する必要がある．

特に文字ベースで考えるインデックス情報とグラフィックベースで考える座標情報とは指し示す意味は同じであるが，どちらの方式が相応しいかは適応するフレームワークによる．そこで，システム内部のデータ構造に沿う様に等価な形で入力構造を変換する必要があると考える．このようにする事で，ビジネスロジックが対象とするフレームワークに応じて夫々，座標情報かインデックス情報かを判断して書き換える必要がない．

つまり，システム内部で想定されるデータ構造に対する概念と外から与えられるデータ構造に対する概念が開発フレームワーク間で異なる可能性がある事がわかる．

5.2. フレームワーク非依存にビジネスロジックを定義する方法

そこで，同一サービスを容易に異なるフレームワークに適応する為に 5.1 で述べた問題点を解消する方法として，要求分析モデルに設計情報として新たな要素を定義する．その為に，フレームワーク特有の要素（例えば、EditText 型の属性）をシステムが要求するプリミティブ型や Entity 型への変換を行う．これが有ることで，システム内部のデータ構造に沿う様に等価な形で入力構造を変換することができる．

具体的には，インタラクションパーティションに「～を変換する」（図 7 の①）というアクションを設け，システム内のビジネスロジックが入力データに依存しないようにする事が望ましい．

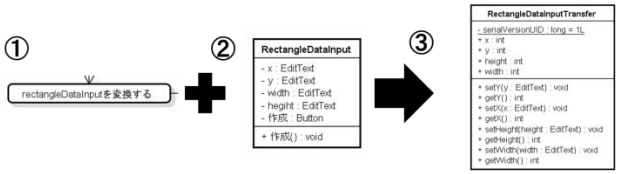

図 7：「～を変換」する設計クラスにする例

「～変換する」アクションは，設計段階で，具体的に次のようなクラス定義を用いて対応付ける．「入出力データ構造名(図 7 の②)＋Transfer」(図 7 の③)というクラスを定義し，属性として，変換したい対象のデータを持つ．そ

れらに対して，Setter 及び Getter を定義し，Setter の定義に，変換元のデータを定義する．Setter の実装で，変換元のデータから変換したい対象にふさわしいデータ構造に変換する様に定義する．ここで定義したクラスは，4.2 節で説明した，「Rectangle Editor」クラスの引数として与えられ，更に，entity のロジック（メソッド）の引数としても与えられる．

5.3. ビジネスロジックへの依存性分析

フレームワークによってビジネスロジックの依存性が，大きく分けてプロセスを単位として入れ替わる可能性とプロセス間の接合が起きる可能性がある事を Android と Applet を比較する事で得た．

その根拠となるサービスは「長方形を作成する」（図 1）である．また，Android における「長方形を作成する」場合の画面遷移を図 8 に表現した．次に Applet における「長方形を作成する」を図 9 に表現した．

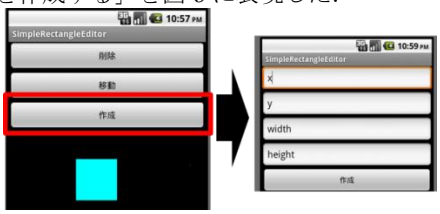

図 8：Android 版「長方形を作成する」画面遷移

図 9：Applet 版「長方形を作成する」

5.3.1. ビジネスロジックのプロセスが接合される可能性

「長方形を作成する」サービスで，フレームワークによってプロセスとプロセスがまとめられ，ひとつのプロセスになる事（接合と呼ぶ）がわかった．Android アプリケーションでは，図 8 に示す様に「作成」ボタンをタッチ後，画面遷移が起き，図 1 にある入力項目を入力する項目が現れ，「作成」ボタンをタッチする事で長方形を作成がされる流れになる．それに対して，図 9 に示す様に Applet アプリケーションでは，図 1 にある入力項目を入力し，「作成」ボタンを押す事で長方形が作成される流れになる．

要求分析モデルにおける「長方形を作成する」では，プロセス①を行い，入力項目の入力を要求し，トリガが実行されると，プロセス②を実行するのが流れである．要求分析モデル通りの順序で Android は実装されている．それに対して，Applet の場合は，「作成」ボタンが押された後にプロセス①を実行後，逐次プロセス②を実行する事になる．その結果，プロセス①とプロセス②が接合されたといえる．

フレームワークとどのように依存性があると考えられるかを述べる．要求分析モデルではユーザとのインタラクションがフローに沿って記述されているが，フレームワークによっては，そのフローを接合する事が適切な場合があり，それに影響され，システム側のプロセスが接合される事となる．

5.3.2. ビジネスロジックの実行順序が変わる可能性

フレームワークによっては，プロセスの実行順序を入れ替えて表現する事が適切な場合が考えられる．具体的には「長方形を作成する」サービスでは，要求分析モデルでは，プロセス①を経てプロセス②が実行される．今回の Applet の実装方法では，プロセス①内で行われるボードの長方形の上限判定をプロセス②の後で行う様に変更すれば，画面を更新した際にボードの状態に応じて（長方形の数が上限に達する事等）「作成」ボタンを無効化する等する事で，作成に必要な項目を入力した後，「作成」ボタンを押し，「長方形が作成できない」旨を通知される事はなくなる．

しかし，要求分析モデルでは，プロセスの実行順序が変わらない前提で，オブジェクトノードが実行手順にそってシステムに保持されており，サービス実行中は使用可能であると考えている．その為，プロセスの実行順序を変更すると，オブジェクトノードの保持の順序関係が崩れる．「長方形を作成する」で考えると，プロセス①で取得したオブジェクトノードをプロセス②でも使用している為，プロセスを入れ替えるとオブジェクトノードの順序関係が崩れる事がわかる．したがって，プロセスの実行順序を変更する為にはシステムのプロセスの中身を書き換える必要が出てくる．

つまり，プロセスの実行順序が変わり得る場合，共通にフレームワークへの適応する事は難しいと考える．

5.4. 入力方式の変更によるビジネスロジックの依存分析

Android に代表される携帯端末プラットフォームの特徴として，フレームワーク固有の多様な入力方式があり得る．例えば，画面上で長方形ドラックする事で長方形を移動させる事が可能である．そこで，ビジネスロジックがフレームワーク特有の入力方式によってプロセス実行順序の変更やプロセス間の接合することが無いかを考えた．

Android による実装に対して，長方形の移動をドラッグによる方法と傾けることによる方法の 2 方法に関して追加実装した．

その結果として，要求分析モデルでは「移動」というボタンが移動のトリガとなっていた．それに対して，タッチする事がトリガとなる．入力方法が異なっていても，本質的にシステムに対する入力は変わらないと考えられる．入力方式の変更が影響を与える箇所として，インタラクションのトリガとして働く箇所（例えば，ボタン）で，接合される可能性がある．一方，ビジネスロジックに対する接合や順序入れ替えは，起きなかった．

6. まとめ

要求分析モデルを異なるアプリケーション開発フレームワークに適応し，比較実験行った結果，ビジネスロジックに与える影響を分析することができた．

その結果，要求分析モデルで影響を与える箇所を特定することができた．更に，要求分析モデルのビジネスロジックがフレームワークの非依存になる様な方法を定義した．

今回は，Java 言語を対象としたフレームワークに対して適応し比較検討したが，異なる言語のフレームワークに対して適応を試みたい．

参考文献

[1] 小形真平,松浦佐江子: UML 要求分析モデルからの段階的な Web UI プロトタイプ自動生成手法,コンピュータソフトウェア,Vol.27, No.2,pp.14-32. (2010)

[2] 奥田博隆, 小形真平,松浦佐江子: CRUD 分析中心の業務ロジックモデリングと機能型プロトタイプ自動生成, 電子情報通信学会技術研究報告,vol.110,No.468, pp.73-78(2011)