ユースケースの例外抽出手法におけるシステム品質改善の検証実験

天川 美那† 松浦 佐江子‡
芝浦工業大学 大学院 工学研究科 電気電子情報工学専攻†
芝浦工業大学 システム工学部 電子情報システム学科‡

1. はじめに

規模が大きく複雑なシステムを開発する上で，構造化プログラミングやオブジェクト指向等に代表される分割統治的な手法が有効である事は広く知られている．しかし，システムを分割する指標がアプリケーションドメインに依存しているため，分割されたシステム部品を異なるシステムに再利用する事は難しかった．これを解決するべく提唱されたのがMDA(Model Driven Architecture)[1]である．ソースコードはシステムのアーキテクチャや実装に関わる様々な要件に依存して定義されるため，その複雑度が増加する．MDAはこれら要件の仕様をプラットフォームとして依存性のないモデルと依存したモデルを分離する事で，システムの複雑さを緩和させる開発手法である．

複雑さの切り口は他にもある．システムはサービスが成立する場合には所定の手続きに従って，基本または代替のフローを実行する．これらの処理はサービスの仕様に従って定義される．一方サービスが成立しないケースは多様であるが，こちらは前述の二つのフローと違い，サービスの仕様に依存する例外とそうでない例外の両方を想定して定義する必要がある．

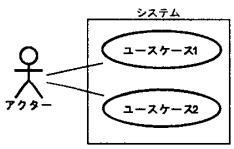

図1 システムのメタモデル

システムはユースケースの集合と外部のアクター，およびアクターとユースケースを繋ぐ関連からなる（図1）．関連はアクターとシステムのインタラクション（相互対話）機能を包含している．しかし，システムの最終的な設計を定義するクラス図にはフローの違いやそれに伴うインタラクションの変化は明確に記述されない．これによって三種のフローがクラス図の中で混ざり合い，ソースコードの複雑化や再利用性の低下の原因となる．

本研究ではシステムの提供するサービス（ここではユースケースと同義）の成立条件をプラットフォームの一種と捉え，この二種の例外を区別する事で，サービス非依存のインタラクションPIM(Platform Independent Model)を定義する．システムをインタラクションとサービスに分割して二つのシステム部品を独立に分析・設計し，システムの複雑さの緩和を図る．[2]

2. 開発手順

まずサービスを定義し，そのサービスの仕様をインタラクションPIMに反映させる．

1. ユースケースの抽出
   従来手法を用いてユースケースを抽出する
2. アクティビティ図の作成
   各ユースケースをアクティビティ図で記述する
3. インタラクションパーティションの挿入
   例外の種別を元にインタラクションをサービスと分離し，2で作成したアクティビティ図にインタラクションパーティションを設ける
4. サービスロジックの作成
   従来手法を用いて，サービスパーティション内のアクションを実現するモデルとソースコードを記述する
5. インタラクションの具体化
   前述のインタラクションPIMをJavaに特化させたPSM(Platform Specific Model)を作成し，作成したサービスロジックの情報を埋め込んでソースコードを作成する

3. 評価実験

以上の開発手法がシステムの品質向上と効率化に及ぼす効果を評価するため，図4のインタラクションPIMと付随するソースコードを用いて実際にシステムを開発する実験を行った．

3.1. 実験概要

被験者は本学の学部3年生である．3年前期の実習「情報処理I」で開発した長方形描画エディタを本手法を用いて再び開発する．長方形エディタは一定サイズのボード上で長方形を作成・拡大・削除する．また，現在ボード上にある全ての長方形を表示する．UIはCUIとし，言語にJavaを用いた．

被験者5名に対し，図書管理システムの開発事例を挙げて開発手順の説明を行った．学生はまず2章の手順3の段階で作成したアクティビティ図と概念モデル図を提出する．モデルがシステム要件を満たしていない場合は不足がある旨を指摘する．また，この時点でユーザにとって使いやすいインタラクションの再考を促す．上を踏まえて再度分析してから次ステップに移行する．

最終的に提出される成果物はアクティビティ図，サービス部のクラス図，ソースコードの3種である．

3.2. 評価

提出されたソースコードと前回の実習時のソースコードに同じテストケース（表1）を適用し，ソースコードとモデルの整合性を踏まえた上で，システムの改善点及び問題点について評価する．また，ソースコードの行数やメソッドの詳細レベルを比較し，品質の変化を見る．

表1 テストケースの項目

機能性	基本フローが正しく実行できる
	入力例外に正しく対処できる
	内部例外に正しく対処できる
使用性	入力のタイミングが適切である
	(作成時)長方形の上限数を超えていれば処理を終了
	(拡大・削除時)長方形がボード上になければ処理を終了
	例外メッセージが適切である(例外発生箇所の明確さ)

- **ソースコードの品質向上**

実習時のソースコードと本手法を用いた場合のソースコードを比較した．表2はサンプルとして二人の学生の

An Experiment for System Quality Improvement Ability of Exception Extraction Method for Use Case
† Mina Amakawa, Graduate School of Engineering, Shibaura institute of technology Department of electronic engineering and computer science
‡ Saeko Matsuura, Shibaura institute of technology Department of electronic information system

結果を提示している．なお，今回分の分析結果についてはサービス部のみの値を表している．

表2　ソースコードの分析結果

	学生A		学生B	
	前回	今回	前回	今回
クラス数	4	5	4	5
メソッド数	29	45	22	51
入力を行うクラス数／メソッド数	1/3	0	3/4	0
出力を行うクラス数／メソッド数	2/11	0	4/11	0
最長メソッド行数	146	6	44	10
最長メソッド名(前回)	cmd		main	
最長メソッド名(今回)	isExpand Rectangle		checkOverBoard Expand	

全ての学生について，サービス部から入出力を行うメソッドが排除された．これにより，前回はソースコード中に散在していた入出力用のメソッドをインタラクション部に集中させ，サービスの実装からインタラクションを独立させた．一方メソッド数は学生 A・B とも約 2 倍になっている．しかしメソッドの行数自体は最大で 4 分の 1 から 20 分の 1 程度に減少している．これはアクティビティ図内で条件審査のアクションを明示する事で，審査の処理がメソッドとして分離されたためである．更にメソッド名から，前回はコマンド入力メソッドや main メソッド内に包含されていた処理が分離された事が分かる．以上より，メソッドの詳細レベルが適切になる事でソースコードが構造化され，可読性が向上したと言える．

- **モデルとソースコードの相違**

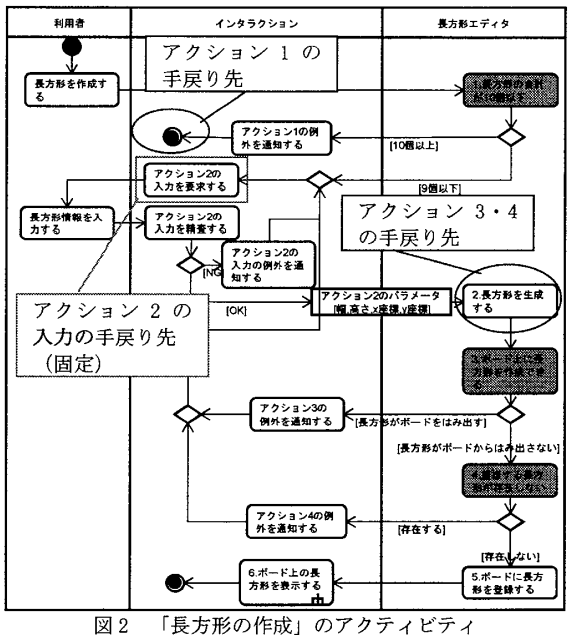

図2　「長方形の作成」のアクティビティ

入力例外時の処理はほぼ正しく定義されている．提供したインタラクション部で，開発者の記述した入力値ごとの制約に対する精査ロジックを元に，例外メッセージの提示と再入力の要求を正しく行う事ができた．基本フローのテストも全員が通過しており，この 2 項目ではインタラクション PIM を用いた開発手順を再現できている．

しかし，内部例外時の処理中に問題が発生した．例として長方形の作成サービスを挙げる（図 2）．作成の達成条件は「ボード上の長方形が上限数未満」「長方形がボードの大きさを超えない」「既存の長方形と重複しない」の 3 つである．また，入力値として幅・高さ・x 座標・y 座標の 4 種を必要とする．各入力値に制約が設けられている．例外が発生した場合，システムは例外メッセージを提示してサービスを終了，もしくは前のアクションに戻って処理をやり直す．入力例外の場合は同じ値について再入力を求めるため，手戻り先のアクションが既に決定している．しかし，内部例外が発生した場合は手戻り先のアクションが異なるため，どのアクションまで手戻りを起こすのかを開発者が定義する必要がある．

提出されたアクティビティ図をチェックした段階ではほぼ全てのサービスにおいて手戻り先のアクションが適切に指定されていた．アクティビティ図で定義した情報をインタラクション PIM に反映する際に問題が生じた．

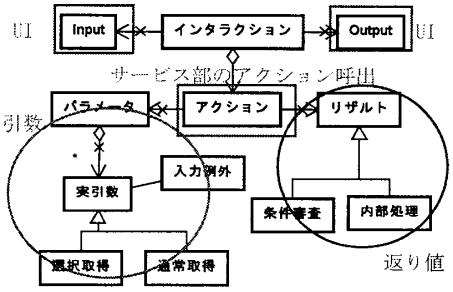

図3　インタラクション PIM

図 3 はインタラクション PIM の概略図である．開発者は図 2 で定義したサービス部のアクションのシグネチャ・入力値の制約・例外メッセージ・手戻りアクションを，図 3 の PIM を Java で具体化したソースコードの中に手動で記述する必要がある．図 2 のこれらの要素を図 3 中のどこに記述すればいいのかを開発者が理解しにくいため，モデルの定義とソースコードが食い違うという問題が発生したと考えられる．

このため，内部での例外を正しく検出しているにもかかわらず，例外メッセージを提示して基本フローのアクションに進んでしまうという現象が起きた．同様に，使用性を高めるために「長方形の上限数を超えていれば処理を終了する」という定義がモデル中では行われているにも関わらず，ソースコードに反映されていないという現象が起きた．

4.　まとめ

本手法ではサービス成立に必要な情報と不成立時の処理をフレームワークとして提示している．モデルの記述とソースコードの記述が一致しない現象は，アクティビティ図からインタラクション PIM へのマッピングを定義し，アクティビティ図の情報を自動的に PIM へ反映させる事で解決できる．これを作成する事を当面の目標としている．また，システムの保守性・使用性の検証のため，UI を CUI から GUI, web GUI に変更した場合にシステムに生じる変化を調査したいと考えている．

参考文献

[1]　S.J.メラー：”MDA のエッセンス”, 翔泳社 (2004)

[2]　M. Amakawa, S. Ogata, S. Matsuura, “A System Development Method Based on A Service Independent Interaction Model”, proceeding of the 9th IASTED International Conference, pp154-159, 2008