2009－SE－163 (3)
2009／3／18

サービス非依存インタラクションモデルを用いたシステム分割手法

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‡

サービスの成立条件が満たされない場合の手戻りや途中終了の発生は，システムを複雑化させる一因となっている．本稿ではサービスが成立する条件を MDA（モデル駆動アーキテクチャ）の１つのプラットフォームと考え，複雑化の要因となるユーザとシステムのインタラクションをサービスに依存しないインタラクションとしてモデル化する．本稿では，このようなインタラクションモデルに基づき，システムを開発する手法を提案し，ユースケース分析段階で切り分けた例外に関する要件をコードにおいて分離し，システム開発の明確化およびコードの複雑度の緩和を図る．本稿ではサービス非依存なインタラクションモデルを PIM として定義し，これを用いたシステム開発手法を提案する．図書管理システムを例として，その開発方法を示すとともに，本手法を用いた開発実験を行いその有効性について議論する．

A System Development Method Based on a Service Independent Interaction Model

Mina Amakawa† Saeko Matsuura‡
Graduate School of Engineering, Shibaura institute of technology Department of electronic engineering and computer science†
Shibaura institute of technology Department of electronic information system‡

To reduce the source code complexity, we define a set of conditions for executing the services on a platform of an MDA (model-driven architecture) system. These conditions, which help in correctly executing each service of a system, bridge the gap between the service logic and the interaction part of the system. This paper proposes a system development method based on a service independent interaction model, which is a PIM (platform independent model). A book management system exemplifies the development process based on a service independent interaction model. Moreover, we discuss effectiveness of our method based on an experimental development.

1. はじめに

オブジェクト指向の分析過程である要求分析の手法の一つにユースケース分析がある．ユースケース分析ではシステムの振る舞いを基本・代替・例外の三種のフローで定義する．オブジェクト指向ではオブジェクト間のメッセージのやり取りでシステムを実現するため，一連のフローを実現する操作は各オブジェクトに付随する操作としてまとめられる．このためシステムの静的構造を表すクラス図中にはユースケースを実現するメソッドの呼出順序や例外が発生した場合のアクターとのやり取りのフローを明示的に記述することができない．基本・代替はシステムがサービスを達成するフローだが，例外はなんらかの原因でサービスを提供できない場合のフローであり，フローの手戻りや途中終了が発生する．このため，例外フローのメソッド呼出順序は煩雑になりやすく，ユースケース分析段階で明示的に分析したにも関わらず，コードの段階においてはシステムを複雑化させる一因となっている．しかし，例外発生の場合には，サービスが成立する条件に関わるデータと振舞いの仕様（入出力およびデータの制約）にのみ依存してフローを定義することができることに着目し，これをアクターとシステムのインタラクションとしてモデル化する．インタラクションはアクターとの境界を実現する実装技術や言語に依存しないモデル，すなわち MDA (Model Driven Architecture) における PIM(Platform Independent Model) としてモデル化する．このインタラクション PIM から実装技術に依存した具体的なインタラクション PSM (Platform Specific Model) を基本および代替フローを実現したサービスロジックとの接合点を埋め込むフレームワークとして生成する．このフレームワークに対して規定された要件を定義することによってサービスロジックとは独立したインタラクションの実装を行う．

サービスが成立する条件を切り口とし，成立時の条件下でのロジックの定義と不成立の場合のアクターとシステムのインタラクションの定義を分離して開発することにより，ユースケース分析段階で切り分けた例外に関する要件をコードにおいて分離し，システム開発の明確化およびコードの複雑度の緩和を図る．

本稿ではこのようなサービス非依存なインタラクションモデルを用いたシステム開発手法を提案する．更に，本手法を用いた開発実験を行いその有効性について議論する．

2. 開発概要

2.1. システム開発手順

システム開発はつぎの手順で行う．

1)ユースケースを抽出する．本稿ではユースケースをサービスと同義とする．

2)各サービスに必要な入力データとサービスが成立する条件を分析し，基本的なサービスのフローをアクティビティ図で定義する．ここで，インタラクションとサービスロジックの役割を明確にする(3.1節)．

3)アクターがシステムとどのようにやり取りを行いたいかを分析し，必要な複数の入力を精査するパターン・精査における例外およびサービスが成立しない場合の手戻りのフローをアクティビティ図で定義する．ただし，インタラクションで定義するアクションは3.2節で述べる種類に限定される．

4)サービスロジックの基本および代替フローを従来とおりオブジェクト指向分析・設計で定義する(4.1節)．

5)4節で述べる手順で，2) で定めたインタラクションとサービスの接合点をラッパークラスとして定義する．さらに，3) で特定したアクションをインタラクションPIMを具体化したPSMフレームワークに定義し，システムを完成する．

2.2. インタラクションPIM

インタラクションとは，アクターがシステムに対して働きかけ，システムが何らかの返答を行うという一連の手続きを指す．そこで，サービスに必要な入力データとサービスが成立する条件により，インタラクションとサービスロジックを分離する．

サービスの成立条件はそのサービスへのアクターからの入力がサービスを実行するために適正であること，およびシステム要件によって決定されるシステムの内部データに依存する条件の検査によって特定することができると考えられる．すなわち，前者の場合にはユーザから入力を受け付け，それが適正な値であるかを判定し，適正でなければその旨をユーザに通知して入力要求を繰り返すか，サービスの提供を停止する．また，後者の場合には，検査結果に応じては，その要因となるユーザの入力を再度要求するか，サービスを提供できない旨をユーザに通知する必要がある．このようなやり取りがシステムの要件ごとに決定される．

このようにインタラクションはサービスの実装には非依存なアクションによって定義することができる．

2.3. MDA

MDAはシステムの実働する環境に非依存なモデルと環境独自の情報を組み合わせ，環境に依存したモデルを生成することでシステムを具体化する手法である．モデルを下位モデル，ひいては実働ソースコードに自動変換することでモデル間の整合性を保証する．これによってモデルを資産化することがMDAの目的である．

システムの実働環境をMDAではプラットフォームと呼ぶ．プラットフォームに依存しないモデルをPIM，プラットフォームに合わせて具体化したモデルをPSMと呼ぶ．MDAではPIMをPSMに変換することを繰り返し，モデルを段階的に詳細化する．

既存のMDAではプラットフォームにOSや言語等の実装レベルの環境を想定するものが通常だが，本研究ではユーザ種別やアクセス時刻等に類する要求レベルの環境であるサービスの仕様をプラットフォームと捉え，サービス非依存のインタラクションPIMを用いて開発を行う．インタラクションPIMについては3.2項で詳述する．

2.4. 図書管理システム

提案する手法のテストケースとして図書管理システムを構築した．図書管理システムは研究室の所蔵する図書の所在を管理するものであり，図書の貸出・返却・検索・追加・削除の機能を持つ．システムを利用するユーザは研究室所属の教員および学生（研究室内利用者）とそれ以外の利用者（研究室外利用者）である．研究室外利用者が利用できる機能は研究室の蔵書の検索のみである．

3. サービスとインタラクションの分割

3.1. サービスの分析と例外の抽出

例として「図書の貸出」サービスの実現を考える．まず「図書」というオブジェクトの規定を行う．図書はISBN・タイトル・著者・出版社という要素を持ち，帯出者としてユーザと関連する．また，研究室に図書が複数あるため，図書の集合体である蔵書というオブジェクトを考える．なお，ISBNは図書を特定するための国際標準であり，10桁または13桁の数字で表現される．当システムでは10桁の数字とする．

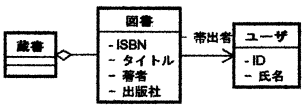

図1 図書管理システムのオブジェクト

このオブジェクト群が貸出というサービスを行う場合，まず蔵書の中から貸出を行う図書を特定する必要がある．ここでは図書の特定のためのキーをISBNとした．アクターはシステムにISBNを入力する．システムが図書を特定し，当該図書が貸出条件を満たしていることを確認して，貸出を行う．図2にこれを図示する．

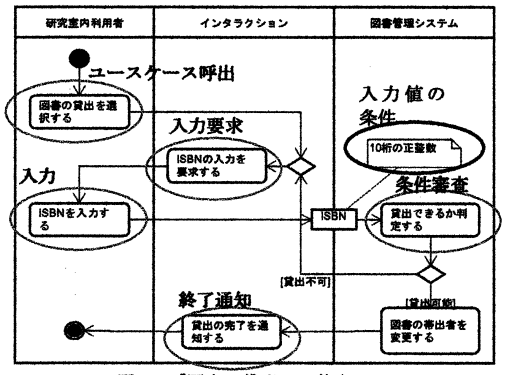

図2 「図書の貸出」の基本フロー

サービスは図2のような五種のアクションを経て達成される．インタラクションとサービスを分割して開発するため，システムをインタラクションとサービスロジック（ここでは図書管理システム）という二種のパーティションで区切って記述する．インタラクションパーティション内にはシステムが外部に対してアプローチを行うアクションを記述する．ここでは入力要求と結果出力のアクションが該当する．

システムが図書を特定するにはまず，アクターの入力した値が ISBN として正しい必要がある．ISBN の正しさとは，上述の規定より 10 桁の正整数であることである．この「入力値の正しさ」がサービスを達成するために満たすべき条件の一つであり，違反した場合は例外を発生する．このため，入力値を表すオブジェクトノードに入力値の満たすべき条件を付記しておく．ISBN の正しさはシステムの提供するサービスを問わず一定である．このため，ISBN の正しさを保証するという責務はインタラクションに含まれる．

一方で，図書の貸出条件も満たすべき条件である．ここではシステムの要件として「蔵書の中に ISBN に該当する図書がある」ことと「図書の中に貸出中でないものがある」ことを図書の貸出条件とした．

以上，二種の条件を審査するアクションをアクティビティ図の中に追加する．同時に，審査を行って例外が検出された場合に例外を通知するアクションを追加し，手戻り先のアクションを指定する．図 3 にこれを図示する．

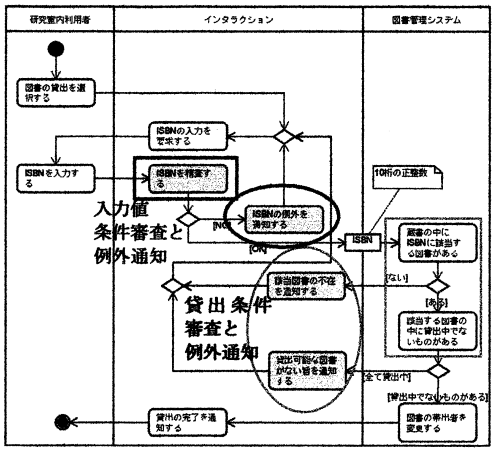

図 3 「図書の貸出」の基本・例外フロー

入力値の条件審査はインタラクションパーティション，サービス（貸出）の条件審査はサービスパーティションに記述する．二種の条件審査に伴う例外通知はアクターへのアプローチであるため，全てインタラクションパーティションに記述する．

なお，上の貸出条件を抽出する際，蔵書中に同じ図書が二冊以上存在する場合を想定している．これを実現するため，オブジェクト群に変化が生じる．ここでは図書のプロパティである本というオブジェクトを加えた．本オブジェクトには冊数属性がある．同一の ISBN を持つ複数の図書は同一の本を持ち，本は冊数として何冊の図書が自分と関連しているかを記録する．図 4 は本オブジェクトを加えた図書管理システムのクラス図である．サービスの条件審査アクションは，サービスロジック内部のデータがサービスを提供できる状態にあることを調べるものである．システムの全てのユースケースの条件審査アクションに対応するオブジェクト群を考えることで，サービスロジックを実現するためのオブジェクトを過不足なく備えた概念モデルを作成することができる．

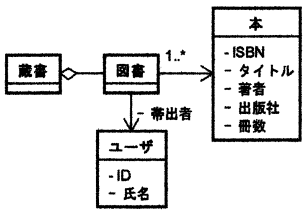

図 4 図書管理システムの概念モデル

3.2. インタラクション PIM

貸出ユースケースでは入力値が ISBN 一つだったが，サービスによっては一般に 0 個以上の複数の値を同時に入力する．対して，複数の値を何度かに分けて入力する場合もある．更に，例外発生時に手戻りを起こすのではなく，そのままユースケースを終了する場合がある．

このようにインタラクションの実現方法はサービスによって異なる．しかし，インタラクションシステムの行う処理は入力要求・入力精査・例外通知・終了通知・サービスロジックのアクションの呼出の五種である．これらの処理はサービスの内容に依存せずに表現することができる（図 5）．

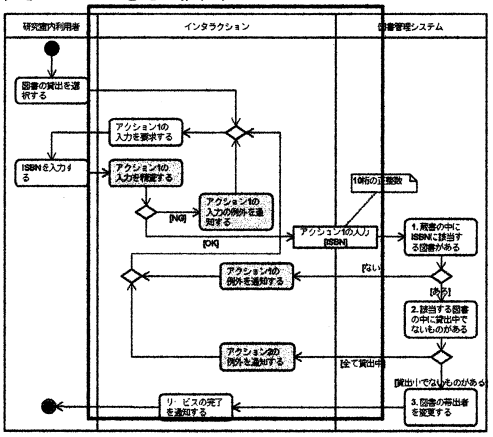

図 5 サービス非依存なインタラクションの記述

上述したサービスによるインタラクションの違いはインタラクションシステム内にあらかじめ用意した入力要求や入力精査等の機能の呼出順序を変えることで実現することができると考え，これらの機能を持つインタラクションシステムのサービス非依存な PIM を図 6 のように定義した．

アクションクラスはサービスパーティション内のアクションにあたる．アクションは引数であるパラメータと返戻値であるリザルトを持つ．パラメータは 0 個以上の実引数から構成される．実引数はアクションに与えられる実際のデータである．実引数には選択取得のものと通常取得のものがあり，選択取得の実引数を活性化する際はインタラクションシステム内部からデータを取得し，選択肢を生成する．リザルトには条件審査アクションのリザルトとサービスロジック内部の状態変更を行うアクションのリザルトの二種がある．条件審査アクションのリザルトの返戻値は必ず論理型である．

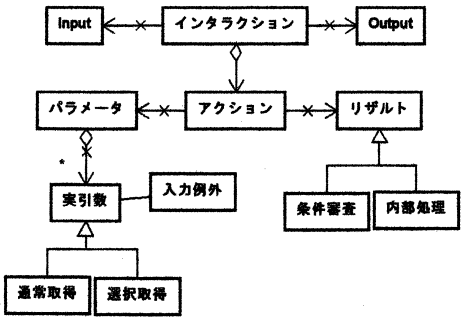

図6　インタラクションPIM（概略）

インタラクションクラスはユースケースを達成するための全てのアクションを把握している．Input とOutput はシステムが外部アクターに対して実際にアプローチする，つまり入出力を行うクラスである．

インタラクション PIM 内には図 5 のインタラクションパーティション内のアクションで記述された内容「アクション x の入力を要求する」「アクション x の例外を通知する」等がメソッドとして定義されている．これらのメソッドはインタラクションクラスの持つユースケース達成のためのアクション系列に応じて適宜呼び出され，アクションを起動するために必要な正しい入力値の取得と，アクションによって行われた処理の結果の通知，発生した例外の通知を行う．

4.2 項でこの PIM をサービス依存の PSM に変換する手順を説明する．

4.　システムの接合と実装

ここでは分割したサービスとインタラクションを接合し，実働システムを完成させる手順を示す．

4.1.　サービスの実装とラッパークラスの定義

まずシステムを実装する言語を用いてサービスロジックを実装する．サービスロジックは入力されるデータが全て正しいという前提の下で構築される．本手法ではサービスとインタラクションを異なるサブシステムとして作成するため，異なるシステム間でデータをやり取りする際にいくつかの制約が生じる．基本的にサービスロジックは開発者が自由に設計・構築することができるが，インタラクションシステムとの接合のために一部を規定に沿って設計する必要がある．

4.1.1.　アクション種別の制約

サービスパーティションのアクションには条件審査アクションとデータの参照および変更のアクションの二種がある．

条件審査アクションを実現するメソッドは全て論理型の返戻値を持つ．システムが条件を満たしていれば返戻値は真，満たさなければ偽となる．

データの参照・変更アクションについては特に制限はない．ただし，インタラクション部で処理の結果を表示するために処理後の値を引き渡す場合は，サービスロジック部での結果表示用のストリームを作成し，インタラクション部に返戻するものとする．

4.1.2.　ラッパークラスの定義

ラッパークラスではサービスパーティション内のアクションを図書管理システムの実装言語である Java に依存したメソッドとして定義し，シグネチャを定める．その際，上述の制約に従うものとする．

- アクション：蔵書中に ISBN に該当する図書があるメソッドのシグネチャ：
  boolean searchBook(double ISBN)
  これは条件審査のアクションなので，返り値は boolean 型になる．入力のオブジェクトノードがある場合は引数として与える．なお，ISBN は整数だが，10 桁の整数は整数型の表現できる範囲を超える場合があるため，浮動小数点型の値とした．
- アクション：該当する図書の中に貸出中でないものがある
  メソッドのシグネチャ：boolean chkRentalState()
  これも条件審査のアクションなので，返り値は boolean 型になる．引数に相当するオブジェクトノードとリンクしていないため，引数はなし．
- アクション：図書の帯出者を変更する
  メソッドのシグネチャ：void cngTaker()
  これはデータ参照・変更のアクションである．インタラクション部に渡す値はないので，返り値は void 型になる．なお，インタラクション部に渡す値がある場合は Java の文字列型配列である String[]型を返り値の型に設定する．

これらのメソッドをラッパークラス内に定義する．

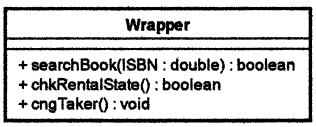

図7　ラッパークラスの定義

ラッパークラスは名前を Wrapper とする．また，ラッパークラスのアクションを実現するメソッドのアクセス修飾子は全て public である．

ラッパークラスはサービスロジックを実現する全てのクラスと関連を持つ．実際の条件審査処理やデータの参照・変更処理はサービスロジックのラッパー以外のクラスで行われる．ラッパーの役割はこれらのメソッドをパッケージ外部から呼び出せるようにすることである．ラッパークラスを通じてサービスロジックのメソッドを外部から呼び出すことで，インタラクションがサービスを実現する．

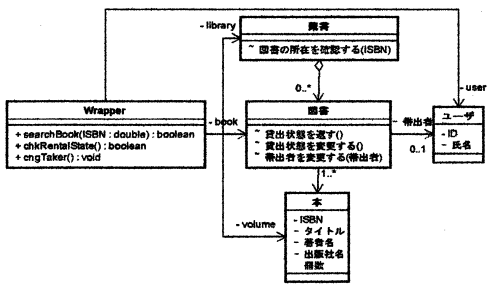

図8　サービスロジックの分析モデル（一部）

蔵書・図書・本・ユーザの 4 クラスの定義は図 4 の通りである．Wrapper に定義された三種のメソッドを実現する具体的な処理として各クラスがメソッドを持つ．また，ラッパークラスはアプリケーションに属す

る全てのクラスのインスタンスを持つ．

ラッパークラスはあくまで他のクラスに定義されたメソッドを呼び出すためのものであり，

- 他クラスのメソッドを呼び出す
- 自分の属性に値を格納する

以外の処理を行ってはならない．

4.2. PIM の具体化

インタラクション PIM を 4.1 項で作成したサービスロジックの情報を元に詳細化し，実装する．

サービスはサービスパーティションのアクションを順次実行して行くことで達成される．インタラクションは各アクションに必要な入力を与え，例外・完了の出力を行い，例外時に前の手順に戻って再び入力からやり直す，といったフローの組み立てを行う．まずはインタラクション PIM を図書管理システムのインタラクションのフローを定め，UI (User Interface)に特化した PSM を作成する．

4.2.1. インターフェースデザインとインタラクションパターン

インタラクション PIM では，入出力を行うメソッドを全て Input クラスと Output クラスに集約させている．インタラクション PIM はインタラクションの実装から独立している．システムの外部に対してどの情報を受け取り，出力するかをアクティビティ図中に定義したが，入出力の方法については規定がない．ここでは上の PIM にインタラクションの実装の情報を付加し，PSM に変換する．

インタラクションの実装を規定する要素はデバイス・入出力機器・UI 等がある．2.2 項で述べたように，MDA ではプラットフォーム境界としてこうした実装機器を動作させる箇所とそうでない箇所を分けることが一般的であった．しかし，特定のデバイスで最適なアクターとシステムのデータの交換方法が他のデバイスでは冗長になり得る．

例としてユースケース「図書の追加」のインタラクションを CUI で実装したものと web GUI で実装したものを図 9・図 10 に示す．

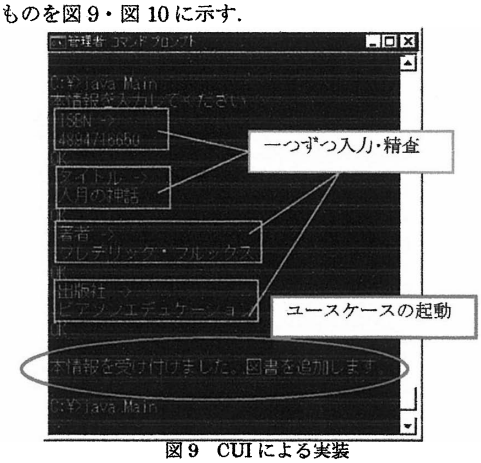

図 9 CUI による実装

「図書の追加」ユースケースを実現するには ISBN・タイトル・著者・出版社の 4 つのデータが必要である．CUI でこのインタラクションを実装する場合，図 9 のように 4 つのデータを一つずつ入力させ，入力された値ごとに逐次精査を行い，入力値が正しければ次の入力を要求する．4 つのデータ全ての入力と精査が完了した時点でユースケースを起動する．

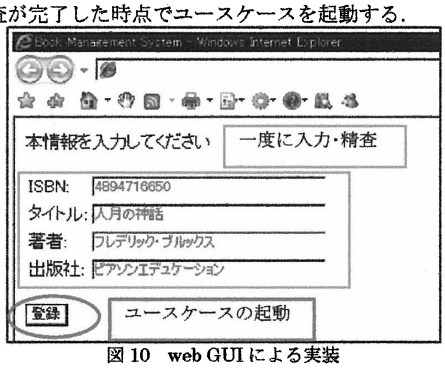

図 10 web GUI による実装

一方 web GUI でこのインタラクションを実装する場合，図 10 のようにまず 4 つのデータを全て入力させる．データの精査も 4 つを並列して行い，1 つでも正しくなければ例外を通知して再び 4 つのデータの入力要求を行う．全てのデータが正しければ自動的にユースケースの起動に移る．

こうした入力・精査の逐次および並列的な実行順序の組み合わせをインタラクションパターンと呼ぶ．CUI で web GUI のインタラクションパターンを実現する場合はありうるが，web GUI で CUI のインタラクションパターンを実現する場合，ISBN の入力欄を提示し，入力を送信して審査を行い，例外が発生しなければタイトルの入力欄を提示... というインタラクションになる．この場合は頻繁に画面の遷移が起こり，手順が冗長になる．

こうした側面を踏まえ，インタラクション PIM では実装機器からモデルを独立させると同時に，入力パターンと精査パターンという二種のインタラクションパターンを設けた．開発者は実装機器に応じてインタラクションパターンを選択する．また，ユーザにとって使いやすいインタラクションについてパターンを通じて議論することができる．

インタラクションパターンはアクションクラスの中で設定される．パターンそのものは列挙型で規定した．入力パターン・精査パターンともに逐次・並列のうちのいずれかを選択する．これらを組み合わせた 4 パターンのいずれかでインタラクションを表現する．

最後に，Input クラスおよび Output クラスの具体的なインターフェースの設計を行う．両クラスには抽象メソッドが設けられており，その中で開発者の記述するメッセージの提示と値の取得，およびサービスの結果の出力を行う．

4.2.2. ラッパークラスのシグネチャの解釈

図 11-1 は図書の貸出ユースケースを実現するアクション群のラッパークラスへのマッピングである．このマッピングは 4.1 項の手順で実現される．アクティビティ図を反映したラッパークラスの情報をインタラクション PIM にマッピングすることで，分析したフローをクラス図上に再現する．

図 11-2 のインタラクション PIM は図 6 のインタラクションおよびアクションクラスに相当する箇所であ

る．ユースケース名として「図書の貸出（lendBook）」を格納し，図書の貸出ユースケースを実現する3つのメソッド名をアクション名として格納する．これを全てのユースケースについて行うことで，インタラクションクラスがユースケースを実現するためのアクション系列を把握する．

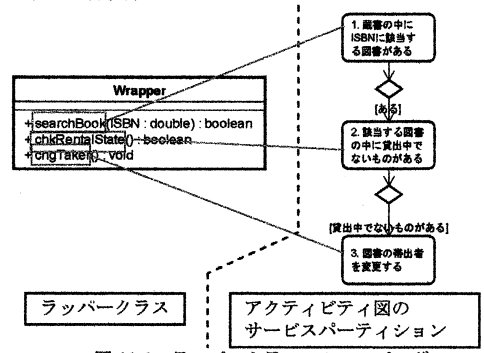

図 11-1　ラッパークラスへのマッピング

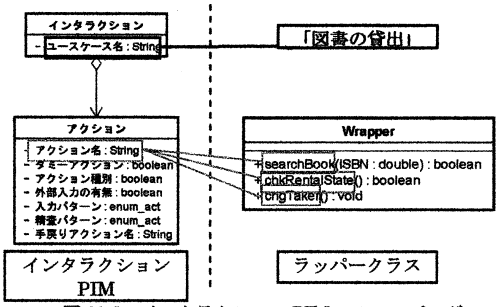

図 11-2　インタラクション PIM へのマッピング

図 11 の例のように，インタラクション PIM とラッパークラス，ラッパークラスとアクティビティ図間でそれぞれ対応付けがなされている．開発者は以下のラッパークラスの定義をインタラクション PIM に記述することで PIM を詳細化する．

・　外部入力の有無

アクションクラスの外部入力の有無属性を決める．この属性は論理値であり，外部入力があれば真，なければ偽となる．外部入力の有無はアクティビティ図のアクションにオブジェクトノードからのリンクがあるかないかで判断することができる．ラッパークラスではアクションクラスのアクション名に相当するメソッドが引数を持つか否かで外部入力の有無を判断することができる．引数を持っていれば，外部入力の有無属性の値は真である．

・　引数の名前と型

ラッパークラスのメソッドの引数の名前と型を，実引数クラスの実引数名属性とその子クラスである通常取得クラスの型属性に格納する．また，実引数クラスのラベルについてもここで設定しておく．実引数名はソースコードが引数名として解釈できる文字列である．ラベルは入出力の際にアクターに提示し，アクターが理解できる文字列である．たとえば本のタイトルであれば実引数名を「title」，ラベルを「書名」のように設定する．

・　アクション種別

サービスパーティションのアクションには条件審査と内部の値の変更・参照の二種がある．アクションの種別ごとにフローの遷移が変わるため，種別を定めておく．アクションクラスのアクション種別属性は論理値であり，条件審査アクションであれば真，変更・参照のアクションであれば偽とする．アクティビティ図中ではアクションの直後にデシジョンノードに遷移していれば条件審査アクション，アクションノードまたはオブジェクトノードに遷移していれば変更・参照のアクションである．ラッパークラス中ではメソッドの返戻値が論理型であれば条件審査アクション，void 型もしくは文字列型の配列であれば変更・参照のアクションである．

・　リザルト種別

システムはアクションの種別に基づいて，アクションの持つリザルトクラスの実体となる子クラスを選択する．条件審査アクションであれば条件審査クラスを，変更・参照のアクションであれば内部処理クラスをインスタンス化してリザルトの実体として持たせる．

4.2.3.　条件審査および例外発生時の処理

二種の例外発生時のフローの遷移と入力精査の内容，および通知する例外メッセージの設定を行う．

・　入力値の例外が発生した場合

入力値が規定の条件を満たさない場合，システムは例外を通知して同じ値の再入力を求める．これはインタラクションの仕様として全ての入力について一律に定められているため，例外後の遷移については開発者が何らかの設定を行う必要はない．

実引数クラスの「個別の制約条件で精査する」メソッドが，入力値の審査を行う具体的な処理を記述するメソッドである．具体的な内容は PIM の段階では記述されていないため，モデル上では抽象メソッドになっている．メソッドの処理はメソッドが精査する入力値の制約の内容を実現するものである．

・　内部データについての例外が発生した場合

この例外については例外通知後に遷移するアクションがサービスの仕様によって異なるため，開発者による戻り先の指定が必要である．「図書の貸出」ユースケースには二つの内部状態条件審査アクションがある．サービスパーティション内のこれらの条件審査で例外が発生すると，フローはインタラクションパーティションの例外通知アクションに遷移する．例外通知アクションののち，フローは「アクション 1 の入力を要求する」アクションに遷移する．アクション 1 とはサービスパーティション内の一つ目のアクション「蔵書の中に ISBN に該当する図書がある」を指す．よって，二つの条件審査アクションの手戻りアクション名属性には「scarchBook」が格納される．

また，例外を発生した時点で通知してユースケースを終了するケースがある．この場合は手戻りアクション名属性に空値 null を格納する．

・　例外メッセージの設定

全ての例外の通知メッセージを設定する．例外通知アクションを実行するのはインタラクションクラスを通じた Input・Output クラスのいずれかであるが，通知する内容は条件審査クラスの中で規定する．また，

処理が正常に終了した場合の通知メッセージを内部処理クラスの中に規定する．

5. 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，従来手法で8ヶ月前に開発した同じ課題をインタラクション PIM と付随するソースコードを用いて同一人が開発する実験を行った．

5.1. 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盤上に長方形を作成・拡大・削除する．また，現在盤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．学生はまずアクティビティ図を作成し，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．

5.2. 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)長方形の上限数超過を初めに審査する
	(拡大・削除時)盤上の長方形の有無を初めに審査する
	例外メッセージが適切である(例外発生箇所の明確さ)

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．

5.2.1. メトリクスについての評価

・ 学生Aについて

クラス数については要求仕様の規定による．

学生Aではサービスロジックから入出力を行うメソッドが完全に排除された．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と言える．一方，メソッド数は約2倍になっているが，メソッドのステートメント数は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ことで，各審査の処理が一つずつのメソッドとして分断されたためである．これにより，前回実習時では一つの大きなメソッドの中に包含されていた異なる条件審査の処理が独立したことが分かる．

また，表2中の最長ステートメント数のメソッド名から，前回はサービスを決定するコマンド入力のメソッドである cmd に包含されていた処理が適宜分離されていることが分かる．今回の最長メソッドは条件審査のメソッドであることがメソッド名から明白である．

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	8	3
メソッド数	29	45	80	39
入力を行うクラス数／メソッド数	1/3	0	1/1	0/0
出力を行うクラス数／メソッド数	2/11	0	2/2	1/1
最長メソッド行数	146	6	51	6
メソッドあたりの平均行数	12.7	2.3	5.6	2.5
最長メソッド名(前回)	cmd		cmdIntersect	
最長メソッド名(今回)	isExpand Rectangle		getIndex	

・ 学生Bについて

学生 B は前回の実習時にシステムの持つ機能単位を重視した分析を行っている．その際，入出力・コマンド選別・条件審査・システム内部の値の変更をシステムの持つ機能と考え，メソッドの責務をある程度分割している．更に独自クラスである IOManager クラスを定義し，入出力に関する処理をこのクラスのインスタンスで行っている．以上の理由から，学生Bの前回のクラス数およびメソッド数は学生Aよりも多くなっている．今回の開発では入出力の責務がインタラクション部に移行したことでサービスロジックの責務が減少し，クラス数・メソッド数が減少した．反面，メソッドのステートメント数は学生A同様に減少している．学生Bは前回の実習時にもメソッドの責務の分割を行っていたが，分割の指標が曖昧であった．また，学生Bが前回の実習時に行った IOManager クラスの作成はインターフェースの分離である．これにより，入出力の処理を書き換えることが容易になる．しかし，インターフェースのみの分離では入出力の方法を変えることができても，入出力の有無自体やタイミングを変えることはできない．入出力の仕様変更に耐えうるシステムを作成するためには，インターフェースではなくインタラクションを分割することが重要であると言える．

本手法によりメソッドの責務が明確に分割され，メソッドの粒度が統一された．また，要求分析の結果を反映するアクティビティ図が成果物として作成された．これにより，要求段階の仕様変更を実装レベルで反映することが容易になる．更にインタラクションをサービスロジックと分離したことで，サービスの実装とインタラクションの実装を切り分けた．これにより，入出力インターフェースを含むインタラクションをサービスと独立した状態でデザインできるようになった．

また，二人の学生の作成した成果物の比較により，異なる開発者の間でメソッドの詳細レベルが統一されたことが言える．これにより，メソッドが構造化され，ソースコードの可読性が向上したと言える．

5.2.2. テスト結果についての評価

表3は表1のテスト項目についてテストを行った結果である．表の数字は「要件を満たさなかった項目数／全項目数」となっている．

表3　表1のテスト結果

	学生A		学生B	
	前回	今回	前回	今回
基本フロー	1/4	1/4	0/4	2/4
入力例外	2/9	1/9	0/9	1/9
内部例外	1/9	5/9	2/9	0/9
入力タイミング	x	x	o	o
上限数超過	x	x	o	x
長方形の有無	o	o	o	x
例外メッセージ	前回より詳細化		前回より粗い	

両者とも入力例外時の処理はほぼ正しく定義されていた．これは，提供したインタラクション部の仕様で「入力精査を通過するまで再入力を繰り返す」というロジックがあらかじめ実現されているためである．ここ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，基本フローおよび内部例外審査の例外発生時に問題が発生した．開発者はアクティビティ図の定義をラッパークラスを通じて言語依存のインタラクション PSM に手動で記述する必要がある．しかし，インタラクション PIM とアクティビティ図の各要素の対応が煩雑であり，開発者にとっては理解しにくいため，作業中にモデルの定義とソースコードの記述に齟齬が生じた．このため，サービスロジックに条件審査アクションを用意しているにもかかわらず，フローの中でそのアクションを呼び出していないという現象が起こった．また，アクションを呼び出した場合も，例外フローに遷移せずに基本フローに遷移してしまうという現象が起こった．

学生Bのサービスロジックに出力用のメソッドが残留した原因もここにあると考えられる．サービスからインタラクションにデータを返す方法が分からなかったため，サービス部で出力を行ったものと考えられる．

また，学生Aの場合は前回の例外メッセージの内容が粗かったため，審査条件ごとにメッセージを規定させることでメッセージの内容が洗練された．一方学生Bは前回の実験で詳細にメッセージを設定していたが，例外の発生箇所と例外メッセージを定義する箇所が分離したことで，例外メッセージの定義方法を十全に理解できなかったことがメッセージの粗さに繋がった．

これらの問題に対応するために，アクティビティ図の要素名を抽出し，自動でインタラクション PIM 内にマッピングする変換機構を作成することを考えている．

6.　おわりに

本稿ではシステムの例外フローに着目し，要求レベルのシステム実働環境であるサービスをプラットフォームと捉えて，システムをサービスとインタラクションに分割して開発する手法を提案した．

従来の MDA ではプラットフォームとして実装レベルの環境（OS・言語・デバイス等）を考えるものが多数を占めている．MDA ではプラットフォームとして捉えるべき対象の規定がなく，開発者たちはプラットフォームとなりうる環境を模索してきた[1]．同一のサービスを提供するシステムを異なる環境で動作させることを考えた場合，表層的に変化するのは実装であるため，まずはこれを切り離すことが考えられた．こうした実装レベルでのプラットフォーム分割は，ソフトウェアシステムの開発に適用されると同時に，多岐に渡る実装デバイスを持つ組込みシステムの分野で特に注目され，研究が進められている[2]．提唱の初期段階ではモデルをソースコードのレベルまでブレイクダウンする技術に関心が高まっており、設計から実装のフェーズを自動変換することが強調されていた。しかし、実装可能なモデルを記述することは難しく、また開発に有効であるとも限らない[3]。モデルの相互変換機構の研究も進められている[4]が、変換元のモデル自体の品質について言及するものは少ない。

本手法ではプラットフォームとしてサービスを考え，要求分析段階で記述されたモデルの内容をシステム分析・設計段階のモデルにマッピングした．より抽象度の高いレベルでプラットフォーム分割を行うことで，複雑なシステム要求を整理し，下流の工程に要求を正確に伝えることを可能にした．

また，インタラクション部を設けることで，サービスの実装に影響を与えずにシステムの入出力を作成することができる．システム外部への値の入出力に注目することで，システムの使用性についての議論が容易になった．インタラクションシステムは UI の実装を含む．実装の違いによって生じる入出力の仕様変更をインタラクション部で吸収できるため，サービスと UI を平行して開発することができる．これが組込み開発における協調設計の助けになると考えている．

本手法はシステム完成後の管理・保守に対する効率化を目的としているため，システムを初めに構築する開発者にとってはコストの大きな開発手法になる．開発者がインタラクションとサービスを分離することによるメリットを認識する必要がある．また，適用実験の結果から，手法を複雑にしている一因として，モデル間のマッピングを手動で行っていることが挙げられる．変換を自動化するべく，現在改良中である．

インタラクション PIM 自体も改良する必要がある．インタラクションが備える機能として，値が初期値を持つ・値の空欄を許可する・再入力時に前回の入力内容を保持する・先に入力した値に応じて次以降の値の制約条件が変化する，等がある．

これらのシステムの改良とともに，システムの保守性・使用性の検証のため，本研究の提案手法にのっとって開発したシステムの 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]安東孝信: “MDA に基づくソフトウェア開発と従来手法の比較，及び実適用へ向けての考察”，情報処理学会研究報告, 2003-SE-140, pp109-116, 2003

[2]細合晋太郎, 岸知二: “ハードウェア情報を含めた MDA の提案と実装”，情報処理学会研究報告, 2007-EMB-005, pp33-40, 2007

[3]D. Thomas, "MDA: Revenge of the Modelers or UML Utopia?", IEEE Software, vol. 21, no. 3, pp. 15-17, 2004

[4]Q. Lan: “Research on Variability Metamodeling Method”, Pervasive Computing and Applications, 2006 1st International Symposium on Volume, Issue, pp 861 - 865, 2006