B-007

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開発の提案

A System Development Method based on a Service Independent Interaction Model

天川　美那†　　小形　真平†　　松浦　佐江子‡
Mina Amakawa　Shinpei Ogata　Saeko Matsuura

1.　はじめに

現在のソフトウェアは日増しに複雑化してきている。システムが提供するサービスの増加とともに新たな技術の開発が進み、後付的に付与されるケースも多い。こうした部分的な仕様変更に対応するために、システムをある一定の基準で切り分けるという、いわゆる分割統治の方法論が有効である。しかし、対象アプリケーションへの依存性が強いか、使用時における切り分けの判断が難しいといった理由から方法が有効活用できない場合が多い。近年注目を集めている MDA(Model Driven Architecture)[1]は OMG(Object Management Group)が提唱するモデルを主体とするシステム開発技法であり、プラットフォーム分割が分割統治の鍵となる。

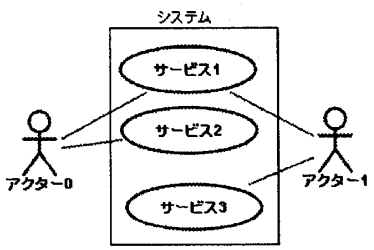

図１　システムのメタモデル

システムはユースケース分析により図１のようなメタモデルで表現することができる。本稿ではここでのサービスはユースケースと同義である。システムの要素にはサービス、アクターの他に、サービスとアクターを繋ぐ線の部分（関連）がある。関連はアクターのサービスに対する権限とともにアクターがサービスを実行する際のさまざまなインタラクションを意味している。このため、サービスを分析する過程でインタラクションの分析とサービスロジックの分析が同じモデルで行われることが、開発およびモデルにおける複雑さの要因となっていると考える。
我々はサービスの成立・不成立の境界を一つのプラットフォームと見て、システムをサービスロジックとインタラクションに分割したシステム開発方法を MDA に基づき提案する。

2.　インタラクションのパターン化

システム構築の典型的な手順では、要求分析を経たのちにシステム分析のフェーズへ移行する。ここではサービスロジックの構造と機能を分析する。サービスロジックを定義したら、次にバウンダリという形でユーザや外部システムとのインタラクションを定義していく。
システムの複雑さの要因のひとつは例外や代替による基本フローからの処理の遷移を基本フローのロジックを分析・設計するのと同時に行うことである。基本および代替ではロジック内のルールとしてサービスが正当な値を提供する手順を定義するが、例外はサービスロジックが不成立となるデータを排除し、適切な処置を施すことを目的とする。
サービスロジックが不成立となるケースには二通りある

1. サービスへの外部からの入力が型および値の制約を満たさない場合
2. サービスを実現するアクションの一部が制約を満たさない場合

これらの場合も含めてサービスのシーケンスを分析すると、バウンダリ部を含むシーケンスは非常に煩雑なものになる。そこで、基本・代替フローが不成立になるケースに着目し、アクターとシステムの間のインタラクション部として分析・定義することを試みた。

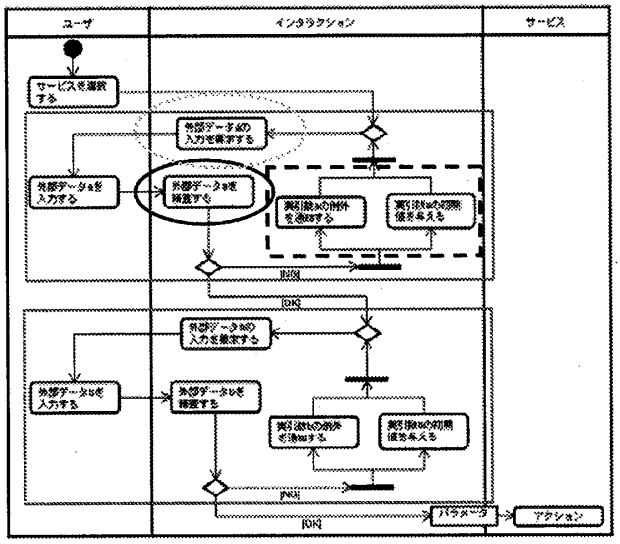

図２　インタラクションパターン
「外部データを取得する_複数実引数１」

```
private void getInput_multil(int num){
  String in = null;
  this.input = new Input(this.action[num].action);
  for(int i=0;
      i<this.action[num].action.param.getArgNum();
      i++){
    while(in == null &&
          this.action[num].action.param.args[i].chkArg(in)){
      in = this.input.inputSingle();
      this.getException(in);
    }
  this.action[num].action.param.makeParam();
}
```

未定義属性

抽象メソッド

図３　図２のソースコード

†芝浦工業大学大学院工学研究科電気電子情報工学専攻,
Graduate School of Engineering, Shibaura institute of technology
Department of electronic engineering and computer science
‡芝浦工業大学システム工学部電子情報システム学科,
Shibaura institute of technology Department of electronic information system

1 の場合はサービスが必要とするデータの制約およびその入力に対する精査と失敗の場合の処理を分析する。2 の場合はアクションの制約が不成立だった場合の処理とインタラクション内での分岐先を分析する。
このようなサービスの成立・不成立の切り分けにより 1 および 2 の振舞いはサービスに依存しない振舞いとして定義することができる。ここでの「サービスに依存しない」の意味は「サービスはアクション列で構成され、外部データ a、b を得て起動するあるアクションに対して、外部データ a の入力を求め、精査し、適切であれば外部データ b の入力・精査を行う。適切でなければ、メッセージを付加して再度入力を求める」といったように、具体的なサービス構成要素名に依存しないことである。このような振舞いをインタラクションパターンとして定義した。図 2 はこのインタラクションパターンをアクティビティ図で定義したものであり、サービス成立・不成立の切り分けによるインタラクションのモデルが図 4 のインタラクション PIM である。また、図 3 はインタラクション PIM に基づくパターンの Java 言語による定義である。

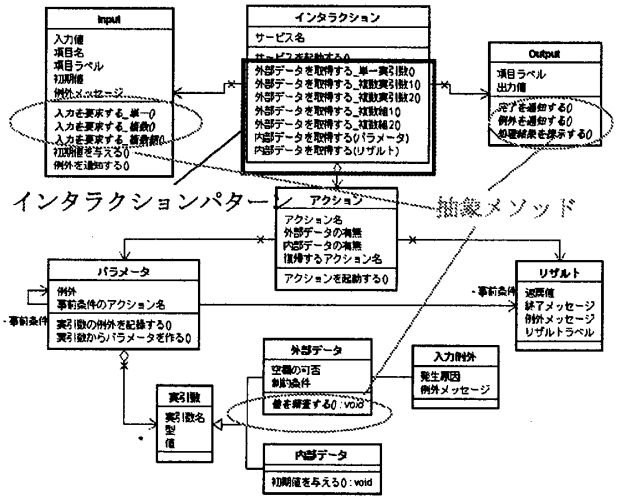

図 4　インタラクション PIM

図 2 のようなパターンの記述要素は図 4 のインタラクション PIM の要素によって定義されている。インタラクションクラスのメソッドは一つずつのパターンを示しており、PIM 内のメソッドの呼び出し順序を記述している。パターンは不成立のケース 1、2 の場合についてそれぞれ複数用意されている。開発者はサービスロジックが不成立になるケースを分析し、デバイスの制限やユーザの使用性などを踏まえて、インタラクションパターンの中から相応しいものを選択する。例えば図 2 のインタラクションパターンはインターフェースに CUI を用いる場合によく行われるものだが、Web ベース GUI を用いる場合は「外部データ a、b を一度に取得し、一つずつ精査を行ってどちらも適切であることを確認する」というパターンが妥当である。
PIM には未定義の属性と抽象メソッドが含まれている。後述の手順で属性を定義することで PIM をサービス依存の PSM に変換し、抽象メソッドを手動で実装することでソースコードを完成させる。
通常はサービスの中に組み込まれているインタラクションの処理をサービス非依存のパターンという形であらかじめ定義することで、サービスとインタラクションを分離できる。これによってシステムの処理系列の複雑さを緩和する。また、使用性の高いインタラクションを分析しやすくなる。

3.　ケーススタディ

3.1. 図書管理システム

今回作成したシステムは研究室所蔵の書籍の貸し借りを管理するものである．システムの機能として貸出・返却・図書の追加・図書の削除・検索の五種を考える．
ここでは図書を「ISBN・タイトル・著者・出版社・貸出状態・帯出者」という六つの属性からなるオブジェクトと定義した．なお、ISBN（国際標準図書番号）[2]とは世界共通で図書を識別するための 13 桁の番号からなるコードである．当該システムでは管理の都合上、ISBN を 9 桁の正整数として扱う。
本研究では同一の機能を持つ図書管理システムを CUI と Web ベース GUI の二種のインターフェースで実装する。これはインターフェースの仕様変更がサービスロジックに影響を及ぼさないことを確認するためである。要求抽出については今回の実験の範疇には含めず、すでにユースケース図とシナリオは与えられているものとして実験を行う。

3.2. システム構築のフロー

- サービスごとにアクティビティ図を作成する

定義した各サービスを実現するためのフローとシステムおよびユーザの振る舞い（アクション）を定義する．

- 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

ユーザとシステムの間のやりとりに注目し，システム側のパーティションのアクションをインタラクションパーティションとサービスパーティションに振り分ける。

- サービスロジック部をサービスメタモデルで定義する

サービスパーティションのアクションおよびアクションの入出力を後述のサービスメタモデルを使って詳細定義する。

- サービスロジックを実装する

上の定義に基づいて、サービスロジック部を実装する。

- インタラクションパターンを決定する

2.3 項で定義したインタラクションパターンの中から、システムの仕様や実装デバイスに相応しいものを選択する。

- インタラクション PIM を具体化する

決定されたインタラクションパターンとサービス定義に基づいて PIM を具体化し、PSM を作成する。更に PSM の中の抽象メソッドで定義された箇所を詳細化し、ソースコードを作成する。

4.　システムの構築

4.1. アクティビティ図の作成

我々は要求の妥当性に焦点を当て，要求分析プロセスで形式的なユーザインターフェースプロトタイプ生成を行う研究を進めている[3]．このとき，フローの主体や振舞いの対象を明確にするために，アクティビティ図の記述に際してのルールを設けている．本稿でもこのルールに則り，ユースケース記述からアクティビティ図を作成した．
更に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。システム側のパーティションのアクションから、ユーザからの入力受付・ユーザへの情報提示・入力精査を行うアクションをインタラクションパーティションに移項する。

手順に基づいて詳細化した「貸出」のアクティビティが図 5 である．上記のアクティビティ図作成作業を全てのサービスについて行う．

続いて、作成したアクティビティ図のインタラクションパーティションの内容を参照しながら、インタラクションパターンを選択する。2 章で述べた複数のインタラクションパターンのうち、1 の入力例外時および 2 のアクション例外時のパターンをそれぞれ選択する。「貸出」の場合は一度の入力で ISBN という一つの値を取得するため、インタラクションクラスの中のパターンメソッド「外部データを取得する_単一実引数()」を採用した。また、二つのアクション例外パターンについては「例外発生時に特定アクションに戻る()」を採用し、戻り先のアクションに「図書を検索する」を指定した。

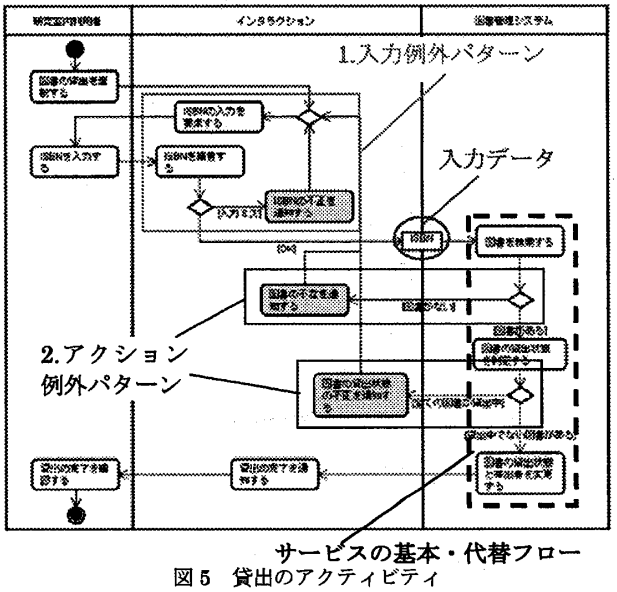

図 5　貸出のアクティビティ

インタラクション部はフローの成立条件を判定するアクションとその入出力を把握し、条件不成立の場合の処理を記述している。図 5 中の図書管理システムパーティションに記述されているのは「貸出」を達成するアクションの一部であるが、仮にこれらのアクションがサービスパーティションで一つのアクションとして記述されていると、インタラクション側で例外ケースの発生を感知できず、処理を定義できないことになる。

インタラクションが把握するサービスのアクションは

- ユーザから入力を受けるアクション
- フローの分岐の直前にあるアクション
- システムの内部状態を通知するアクション

の三種である。サービスのフローが不成立になる条件をあらかじめ抽出しておき、それに見合うアクションを選出する。アクションの選択が不適切な場合、インタラクションを正しく定義できずにサービスとインタラクションの相互独立を維持できない可能性があるため、サービスを構成するアクションの抽出には注意を払う必要がある。

4.2. サービスロジックの作成

図 5 の図書管理システムパーティション内のアクションを実装する。インタラクション部とサービスロジックは相互独立であるため、サービスロジックの分析や定義に基本的に制約はない。ただし、インタラクション部の未定義属性を定義するため、サービスメタモデルに沿ったサービスアクション群の分析を行う必要がある。

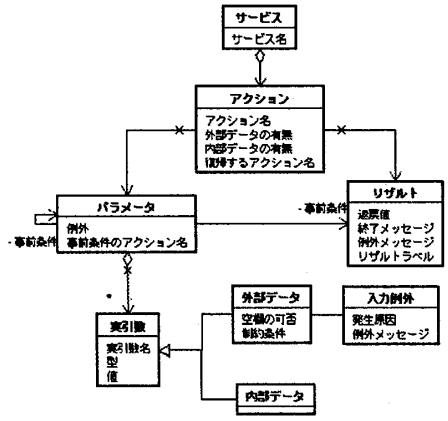

図 6　サービスのメタモデル

図 6 が用意したサービスメタモデルである。サービスは一つ以上のアクションの系列から構成され，アクションはパラメータとリザルトを一つずつ持つ．パラメータはシステム外から取得する外部データとシステム内部の値である内部データの集合である．内部データをパラメータとして用いる場合は，前回以前のアクションでシステム内部の値を取得し，それをパラメータとして与える．なお、このメタモデルは副作用を持たない関数型言語的にアクションを表現しており、オブジェクト指向の観点からはやや冗長なモデルである。本研究の開発手法ではインタラクションとサービスを分割しており、インタラクションを成立させるためにデータの入力と出力を明確に定義する必要があるので、このモデルによる分析過程を必須のものとした。

ここで定義した属性の値を図 4 のインタラクション PIM の中に格納する。サービスのメタモデルと Input、Output 以外のインタラクション PIM のクラス名は一対一対応している（ただし「サービス」クラスは「インタラクション」クラスに対応）。サービスメタモデルの各クラスが持つ属性名と PIM の各クラスが持つ属性名も同様に一対一で対応している。この対応付けに沿って、インタラクション PIM の属性にサービス依存の属性の値を格納していく。

4.3. インタラクション PIM の具体化

4.1、4.2 項の作業を全てのサービスとアクションについて行い、各アクションに適用する二つの例外パターンとサービス依存の属性を定義した。これにより、図 4 で示したインタラクション PIM は図書管理システムに特化したインタラクション PSM に変換されている。最後に抽象メソッドを手動で実装し、実働するインタラクションシステムを完成させる。

抽象メソッドで表現されているのは Input、Output クラスの画面作成用のメソッドと外部データクラスの「値を精査する」メソッドである。外部データの値の制約条件は 4.2 項のサービスメタモデル内で定義されているが、制約条件に基づく実際の判定処理をここで記述する。貸出のアクション「図書を検索する」の外部データ ISBN の制約条件は「428 から始まる 9 桁の正整数」である。これを記述したものが図 7 のソースコードである。なお、このメソッドは図 3 の中で呼び出されている「chkArg(String)」と同じものである。

以上の手順でインタラクションとサービスロジックを構築する。

```
protected boolean chkArg(String arg){
  if(!this.chkType(arg)) return false;//型精査（共通）
  if(this.value < 4230000000) return false;
}

private boolean chkType(String arg){
  this.value = null;
  if(this.nullOK && arg == null);//null可かつ値がnullなら終了
  else{
    if(this.type.equals("null")){
      if(arg != null) return false;
    }
    try{
      if(this.type.equals("double")){
        this.value = Double.valueOf((String)arg);
      }
      else if(this.type.equals("String")){
        this.value = (String)arg;
      }
    }catch(NumberFormatException e){
      return false;
    }
  }
}
```

図 7　ISBN 精査のソースコード

5.　適用実験

作成したインタラクション PIM のサービスに対する独立性を検証する．サービスに依存しないインタラクション PIM の条件として

1. インタラクション仕様の変更に対して，サービスロジックに変更が生じない
2. サービスロジックの機能拡張に対し，インタラクション PIM および具体化手法に変更がない

ことが挙げられる．

1 については CUI・web ブラウザ GUI の二種のユーザインターフェースについて PIM を具体化することで検証できたものとし，2 の場合について適用実験を行う．

ここではサービス「図書を追加する」の機能を拡張する。図書を追加する場合，現在のサービスの仕様では図書が蔵書にある場合、ない場合にかかわらず「ISBN・タイトル・著者名・出版社名」を入力する。しかし、同じ図書が既に蔵書にある場合は所蔵冊数を増やすことで追加の処理を完了できる。使用性の観点から見て無駄な入力は極力省くことが好ましいので、「同じ本が既に存在する場合は入力項目を省略する」という機能を設けることを考える。このサービスの場合は図書を識別する ISBN 以外の情報はシステム内部に既に存在するものを使って処理を行うことができるので、サービスを実現するアクション系列に内部に情報が存在するか否かを判定するアクションを追加する。

新しいアクションを付け加える場合、まずそのアクションがフロー成立の条件判定を行うものか否かを判断する。条件判定のアクションである場合は 4.2 項の手順に則ってアクションを分析し、インタラクション PIM に情報を追加する。通常のシステム開発ではこうした処理フロー変更によって各部に起こる影響の範囲が定まっておらず、バグの原因になりやすい。特にフローが不成立だった場合にアクションの手戻りが起こる場合は顕著である。インタラクション PIM に沿って開発を行うことであらかじめ全体の流れを把握し、変更箇所を明確にすることができる。

図 8 はフロー変更後の図書追加のアクティビティ図である。サービス「図書を追加する」を達成するアクション群の一番初めに「図書を検索する」というアクションを追加する。このアクションはユーザからの入力を受け付けるので、入力例外のパターンを選択し、サービスメタモデルに従って分析を行う。更にアクションの結果次第で「既存の図書を追加する」というアクション，「新規の図書を追加する」というアクションに分岐する。アクション「新規の図書を追加する」はフロー変更前の「図書を追加する」アクションと同じ内容であるため、ISBN 以外の必要な外部データの入力をユーザに対して要求し、精査を行ってアクションを起動する。「既存の図書を追加する」アクションは外部からの入力を必要としないので、サービスロジック内で処理を完了する。

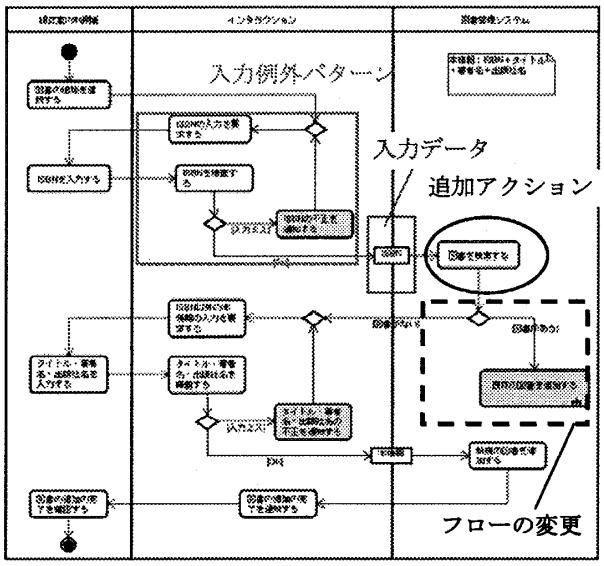

図 8　図書追加の処理フロー変更

6.　今後の課題

本稿では外部データの取得パターンに関して、ユーザによる自由入力の場合を主に考えた。しかし、典型的な入力には「データを列挙し、その中から選ぶ」という選択入力が存在する。今回提示したインタラクション PIM では選択入力のパターンを記述するには不十分である（提示する選択肢を得る手順や選択入力の場合の精査等）。これに対応する形で PIM を定義しなおす必要がある。

また、出力部についてのパターンが不十分である。出力には少なくとも「処理の成功／失敗を通知する」ものと、図書検索等の「ユーザの欲しい情報を列挙する」ものが考えられる。こうした出力のパターンを画面の要素に対応付けることで、開発者が手動で定義する作業を軽減することができると考えている。

7.　参考文献

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

[2] ISBN(International Standard Book Number), http://www.isbn-international.org/, The International ISBN Agency

[3] 小形真平，松浦佐江子：UML の要求分析モデルからの Web アプリケーションプロトタイプ自動生成，情報処理学会研究報告，Vol.2008，No.29，pp9-16 (2008)