Vol.2015-SE-187 No.35
2015/3/13

VDM++におけるテストデータに基づいたリファインメント検証手法の提案

水谷祐馬[†1]　　石川冬樹[†2]

コンポーネントベース開発などのオブジェクト指向設計における段階的詳細化手法に対して，形式仕様記述を導入することは有効である． それは，段階的詳細化過程における各ステップにおけるモデル内，およびモデル間の論理的検証が可能となるためである． そのため形式仕様記述言語として，責務の切り分けすなわち機能分割を許容するリファインメント手法の実現が必要となる．

また, B メソッドや Event-B により実現されているリファインメントでは内部状態に対するリンク(接着)不変条件をベースとした検証方法をとるため，モデル化対象の分割に対応することは難しい．

これら課題に対して， VDM++でのテストデータによるリファインメントの導入による解決，を提案する．

本手法はテストデータによる外部からの仕様同一性，論理的整合性を検証する手法のため，詳細化による内部機能の分割およびインタフェース詳細化方法に対して非依存な検証が可能となり，直接的なオブジェクト指向による仕様の段階的詳細化が形式仕様記述言語上で可能となる．

また，本手法は論理的整合性の必要条件を重ねていき検証精度を高める方法のため，利用者により検証のウェイトを調整することができるなどの柔軟性を備える特徴がある．

1. はじめに

近年，日本国内でのソフトウェア産業において形式仕様記述への取り組みが盛んである．さらに，形式仕様記述の活用を促進するには形式仕様記述がウォーターフォール開発モデルに代表される段階的詳細化プロセス対して，柔軟な適用，品質条件の選択など，幅広い対応が求められる．

段階的詳細化の例としてオブジェクト指向分析/設計におけるコンポーネントベース開発を考える．コンポーネントベース開発においては, UML による概念モデルまず作成し，システムの制約を考慮したうえで，詳細化されたモデルを段階的に作成していく．この詳細化に対して，形式仕様記述言語を適用するには，オブジェクト指向で記述された詳細化前後のモデル間について整合性を検証するためのリファインメントが可能であることが望ましい．

本文においては，コンポーネントベース開発などのオブジェクト指向分析設計における段階的詳細化を直接的に形式仕様記述により実現するため，VDM++[2][6]におけるテストデータによるリファインメント手法を提案する．

本手法は設計者の意図する構造のクラス分割が実現可能という意味で柔軟であり，クラス内設計に依存しない特徴がある．また，本手法でのリファインメントはテストデータにより実現する特徴がある．既存のリファインメントとは異なり，検証精度を利用者が調整することができる．

1.1 オブジェクト指向形式仕様記述言語でのリファインメントの動機

オブジェクト指向分析設計に導入する形式仕様記述言語を選ぶ場合は，VDM++が候補になる．

実際，コンポーネントベース開発方法論におけるモデル記述は UML のクラス図，状態遷移図，シーケンス図に代表される，構造，状態，操作の表現が求められ，VDM++では論理式で表現される．また，VDM++言語仕様では事前条件, 事後条件の記述が可能であり, これは UML モデリングにおける OCL 式に相当する．

本論文では，段階的詳細化の論理的検証実現を目標とし，実現のためのリファインメント検証方法を提案する．

1.2 既存手法によるオブジェクト指向モデル詳細化実現の難しさ

VDM++以外の言語では既にリファインメント手法は確立され，ツールも充実している．ここでは Event-B[4][7]を例にとる．

オブジェクト指向分析設計では，クラス分割/責務の分割について設計者の意図を反映できる自由度が必要である．

- 状態の分割
- 構造の分割
- 責務の分割

上記に対して，Event-B の対応する概念は以下である．

- 状態は instance_variable が対応
- 構造は machine が対応
- 責務は event が対応

これらを設計者の意図に従い分割記述は工夫をすれば可能かもしれないが，一般的なオブジェクト指向という意味での直接的な表現は難しい. これは構造/状態/責務を分割するオブジェクト指向設計の詳細化について，従来のリファインメントを利用することは難しいことを意味する．

したがって，本論文では VDM++におけるリファインメント検証方法をあらたに検討する方針を選択した．

1.3 リファインメント検証方法の概要

既存の形式仕様記述における詳細化手法，すなわち Event-B に代表されるリファインメントでは内部状態に対

†1NTT ソフトウェア株式会社
NTT Software Corporation
†2 国立情報学研究所
National Institute of Informatics

する不変条件をベースとした検証が行われる．

詳細化過程では上位モデルでの内部状態が仮で設定される可能性がある．その状態から不変条件により性質を引き継ぐといった場合，詳細時の設計要件と仮の内部状態が競合する可能性があり望ましくない．したがって，本論文においてはモデルの内部状態ではなく，操作に対する入出力列の集合の比較によるリファインメント検証を提案する．一般に VDM++モデルは状態を持つことから，単独での操作ではなく，操作を繰り返しすべての処理結果を確認することで本来外部からは確認されない内部状態の変化を間接的に確認する．この繰り返し操作の確認により，モデル全体の比較を実現する．

ここで，モデルの同一性という用語を定義する．二つのモデルが与えられ，それらモデルが同一モデルであることをモデル外部の入出力インタフェースが同一視できること，任意の繰り返しの操作について処理結果が同一であること，と定義する．

本論文においては，入出力データに関する意味的な同一視を設計者の意図で与えることにより，モデル間の比較検証を可能とする．

この定義は，要求定義から基本設計に至る過程における初期モデルから責務/構造/状態を分割したのちの詳細モデルまで一貫して，外部操作に対する処理結果について一貫しているかを確認することを意味する．

これにより，設計者が詳細化したモデルが同一モデルであるかの検証が実施可能となる．この同一性検証を詳細化前後のモデル比較に適用し，同一性を確認することで，リファインメントされたモデルであることを検証する．

一方，アニメーション可能な形式仕様記述言語において，このような操作による処理結果を評価することは，テストケースによる検証ともとらえることができる．すなわち，二つのモデルでのテストケースを揃え，比較することでリファインメント検証が実現することが可能となる．

本リファインメント検証で用いる入出力列の集合もテストケースと呼ぶこととし，本リファインメント検証をテストケース・リファインメント検証と名付ける．

テストケース・リファインメント検証では，すべての組み合わせの操作に対する処理結果パターンが同一であることが，モデルそのものが同一であることが十分条件である．しかし，この検証は現実的ではないため，検証者が設定した粒度でのテストケースにより同一性検証を行う．これは同一性モデルであることの必要条件を得ることなる．この必要条件は設計者含むモデルのステークホルダが機能実現のために定義した条件であればよい．これにより，本来必要とされる，すなわちステークホルダが関心とする機能が適切に詳細化される．

また，検証精度を高めたい場合には必要条件を積み重ねることで対応できるのが本手法の特徴であり，どこまで精度を高めるかについては検証者により選択されることとなる．

その他特徴として，テストケース・リファインメント検証は仕様記述における内部状態の記述方法に依存しない．そのため，アーキテクチャ設計過程での状態の分割に自然と対応することができることから，システム設計における詳細化に対応しやすいことが特徴である．

また，テストケース・リファインメント検証においてはリファインメント対象のシステム境界のみを関心事とするため，内部の状態および操作ロジックの表現に依存しないことから，モデル詳細化時の方針にリファインメント検証が影響を及ぼすことはない．

テストケース・リファインメント検証は状態変化および操作ロジックを均一なデータ集合として表現し，検証することからツールによる検証の自動化が期待される．

2. 概念整理

2.1 リファインメントの分類

リファインメントの利用は 2 種に分類される．一つ目は，抽象的な仕様から具体的なプログラムへ具体化する利用であり，B メソッド[3]などで実現されている．二つ目は段階的に機能や仕様の構成を追加する利用で，Event-B などにより実現されている．前者は垂直リファインメントと呼ばれ，後者は水平リファインメントと呼ばれている．本論文においてもこの呼び方にならう．

オブジェクト指向分析/設計への形式仕様記述を適用し，リファインメントによる詳細化を行う場合は以下の適用が自然である．

- オブジェクト指向分析 : 水平リファインメントによる形式仕様記述の適用
- オブジェクト指向設計 : 垂直リファインメントによる形式仕様記述の適用

このうち，水平リファインメントは Event-B が対応する．

一方，オブジェクト指向設計へ自然に対応する垂直リファインメントは存在しない．なぜなら，オブジェクト指向設計で発生する構造，状態，操作の分割と対応する既存リファインメントで許容される操作が対応しないためである．

例えば複数個のサーバによる構成が見込まれるシステムを開発する際に全体のモデルから基本設計を経て，サーバごとのモデルへ詳細化する際に，どのサーバへどの状態，どの責務を分担させるかをその時点で決定し，クラス分割することが必要となる．この時に，例えば Event-B を利用しようとした場合は，安直に一つの全体の machine を複数の machine に分割することが候補になるが，そのような使い方は誤りであり，システムを識別する記述は別に整理する必要がある．

例に挙げた基本設計工程ではシステム全体の境界と責務の概要からサーバ個別の責務の分割および境界の詳細化を行うことであるとする場合，システム境界の詳細化と責務の分割を同時かつ自然に行われることが望まれる．この点において，Event-B ではシステム境界を主眼としたモデル記述およびリファインメントではないこと，B メソッドでは境界の詳細化(入出力パラメータの詳細化)に対応しないことなどに起因し，オブジェクト指向設計の詳細化へは自然に対応しない．

本論文ではオブジェクト指向設計に対応する垂直リファインメントの実現を具体的に論ずるため，便宜上仕様記述言語を VDM++に限定する．オブジェクト指向表現が可能，逐次処理による表現が可能かつアニメーション可能であれば他言語でもよい．

2.2 用語定義

本論文において，異なるモデル間での比較時に同一という用語を用いる．本論文独自の使い方のため定義する．

(1) モデル間でデータが同一

本論文においては，異なる二つのモデルを扱い，モデル間でデータを比較する．モデル間でデータが同一であるとは，モデルを記述した人(設計者と呼ぶ)のみ識別可能である．つまり，型・名称が異なったパラメータでも設計者が同じ概念としてそれぞれのモデルでデータが定義されていればそれは同じデータであると取り扱える．それを論理的に表現し，その論理に基づいてデータの関係性が定義され，その定義に基づいて二つのデータの対応が確認できた場合，それらデータは同一であるという．

このとき定義される設計者が同一であることを意味する写像を同一視写像と呼ぶ．

例えば，顧客データを取り扱うシステムを表現したモデル A とモデル B でユーザデータという概念をモデル A では token 型[a]，モデル B では名前(Seq of char 型[b])，年齢(int 型)で構成される Record 型で定義されているとする．このとき，設計者が同一視写像を与えるとは，モデル A の token a をモデル B の名前 b を用いて，mk_token(b)=a [c]が成り立つと同一であると定義することをいう．

(2) モデルが同一

異なる二つのモデルについて，同一であることを以下の条件を満たす場合と定義する．

A) モデル外部から操作可能な操作，関数が 1 対 1 に対応すること
B) 対応する操作，関数の入出力データ型に対して，モデル間の同一視写像が与えられていること
C) モデル初期状態からそれぞれのモデルに同一入力データによる同一操作を任意に実行した場合において，結果が同一であること．特にデータが出力される場合はその出力が同一であること

これは，操作や入出力のデータ型が異なっていたとしても，概念的に同一であることを論理的に定義することで異なるモデルの比較を可能とする．

例えば，入力した整数を記憶し，今まで入力した整数の合計を返却するモデルを二つ用意したとき，以下の基準で同一有無を判断する．これは，入力型や内部状態などが一致しない場合においても，設計者が意図した操作の対応関係が論理的示されることで同一とみなすことの例である．

- 片方のモデルのみ，本来隠蔽化されるべき今まで保持した整数を確認する操作がある(操作が二個に増えている)場合，同一のモデルではない
- 片方のモデルは nat での入力だがもう片方のモデルでは real の入力かつ事前条件に 0 以上の整数であること以外差異がない場合，入力値を同一視写像が定義されることで同一のモデルとなる(ただし，引数の型と事前条件が関連する例となるため，検証が複雑となる)
- 片方のモデルは内部変数が配列で今まで入力した nat をすべて保持し，もう片方のモデルが今まで入力した nat の合計値のみを保持する以外差異がない場合，入力値を nat から nat への恒等写像による同一視写像を与えることにより，同一のモデルとなる

2.3 リファインメント検証要件

オブジェクト指向設計におけるリファインメント実現の要件は以下の通りであり，以降個別に詳細化する．

1. 下位モデルが上位モデルの詳細化であることが検証されること
2. オブジェクト指向設計手法に対応すること

2.3.1 詳細化検証要件

下位モデルが上位モデルの詳細化であるとは，モデルとして同一であり，かつ入出力データ，クラス構成，内部関数構成が詳細になっていることが要求される．ただし，モデルが同一であること以外は設計者により詳細化方針が決定されることを前提としており，細かい要求は与えない．

モデルが同一であることの定義は 2.2(2)のとおりである．上位モデルと下位モデルが同一であることは，詳細化の動機すなわち，ある目的のプログラムを得るための基本的な要求を変わらずに引き継ぐことを担保するために必須な検証観点である．本手法では，要求の引き継ぎに関する検証を直接的に行っている．この点は従来のリファインメント手法より要求からそれないという意味で望ましい．

a VDM においては，演算やメモリ上への配置などを定めない識別子（トークン）を抽象的に表す型を用いることができる．
b VDM における文字列型．
c この等式は，識別子の作り方と，詳細化されたモデルで与えた識別子の型表現とに対応関係を定義している．

2.3.2 オブジェクト指向設計対応要件

オブジェクト指向設計あるいはそれを発展させたコンポーネントベース開発方法論においては，段階的詳細化過程において以下が重視される．

1. 早期のインタフェースによる分割統治
2. 分割による責務の移譲
3. 機能分割およびデータ分割の明確化

これらの重視させる点に対して，個別のモデルを記述する点において VDM++は十分な表現を持つ．

さらにリファインメントするためにはインタフェースが達成すべき責務に対し分割統治され，内部的な機能およびデータの分割については自由とする必要がある．これが，オブジェクト指向設計に対応するための要件となる．

3. テストケース・リファインメント検証

3.1 リファインメント要件からの整理

段階的詳細化過程において，仕様が詳細化される際の特徴を以下に設定する．

A) 機能分割
B) ロジック詳細化
C) 入出力詳細化

1.2 既存手法によるオブジェクト指向モデル詳細化実現の難しさにて論じたとおり，外部観点のみの条件で検証可能なリファインメント手法の定義が本質的な課題である．

3.2 手法への制約

以上からリファインメント検証手法に対する制約あらためて以下に設定する．

- 設計者が与えた同一視写像基づいた検証を行うこと
- 外部操作によるモデルの挙動によりモデルが同一であることが検証されること

3.3 手法定義

3.3.1 手法概要

VDM++を利用し，以下の方法でリファインメント検証を実現する．

A) リファインメント前後の意味として同一な入出力に関してマッピングを定義する(マッピングは例えば，token 型と Record 型としてもよい)．すなわち，同一視写像を定義する
B) VDM++仕様モデルを入出力データの集合，すなわちテストケースによる表現へ射影する(モデルが同一であることの比較用のデータを整理する)
C) テストケースに対して，A で定義した同一視写像を用いて，データが同一であることを検証する．
D) すべてのデータが同一であることが確認できた場合，そのモデルが同一であることを検証したという

上記手法のポイントはロジック表現による仕様記述モデルを入出力の列の集合によるデータ表現に射影し，射影されたデータ列について同一性を検証する点である．

このようなロジックによる仕様表現とデータ集合による仕様表現を取り扱う点については，Specification by Example[1] が参考になる．(Specification by Example では例示により仕様を与えているが，本手法は仕様から例示(テストデータ)への表現の変換を行っているため，考え方として全く同一というわけではないが，ステークホルダが関心事とする条件を利用するならば観点として同一となる．)

仕様モデル変換イメージを以下に示す．

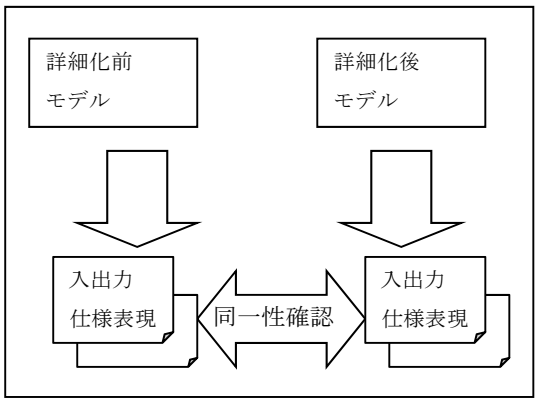

図 1　モデル変換イメージ

なお，リファインメントを検証するためのモデルとして以下の検証モデルを VDM++により記述する．

- 検証対象モデル(詳細化前，詳細化後モデル)
- リファインメント検証モデル

リファインメント検証モデルは，後述される検証対象モデルの分類により定義されたデータ群を入力として詳細化を検証するモデルとなる．

3.3.2 例とするモデル

手法を説明する過程で例もあわせて記述する．例のために以下のモデルを準備する．

・ ログイン機能を有する Web システムの設計を行う場合，ログイン ID とパスワードをシステム内部に保持し，ログイン状態を管理するシステム

このシステムのログイン管理のみを取り扱うモデルを詳細化する．

このモデルはログイン，操作という二つの操作と，ユーザという集合とユーザごとのログイン状態というモデル内変数で構成された 1 クラスで表現する．

このモデルを E と名付ける．

モデル E の詳細化において，以下の機能・非機能要件を付加する．

A) Web システムへはユーザ端末のブラウザを用いて NW 経由で接続すること
B) 同一ブラウザ接続内はログイン状態を維持すること
C) 接続元ブラウザの変更を検出した場合は再ログインを要求すること

上記要件により，ユーザ端末側のブラウザ上で動作するプログラムのサブモデル 1 と Web システムそのもので動作するプログラムのサブモデル 2 で構成されるモデルに詳細化される．

サブモデル 1 はユーザ操作を受け付け，cookie に相当する状態を持ち，サブモデル 2 に対して制御を要求する．サブモデル 2 はログイン状態を確認し，要求制御を処理する．

このモデルを F と呼び，そのサブモデル 1 を F_1，サブモデル 2 を F_2 と呼ぶ．

3.3.3 検証手法詳細

本リファインメント検証手法の詳細を説明する．

説明にあたり，仕様記述内でのモデルの範囲と分類を定義する．構造のイメージを以下の図に示す．

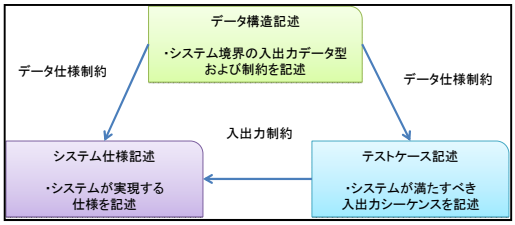

図 2　モデル構造イメージ

分類の説明を以下に示す．

・データ構造記述

モデルのデータ型の定義のみを行う．同一視写像はデータ構造記述間で定義される．

・システム仕様記述

詳細化対象のモデルを記述する．外部操作における入出力パラメータはデータ構造記述に従う．

・テストケース記述

システム仕様を検証するテストデータを記述する．このテストデータはモデル検証そのものにも利用し，入力値に対して期待する出力を行うかの確認に利用される．検証後はそのテストデータはモデルの入出力結果を表現するデータ集合となる．検証時にはモデル間での比較を行うため，テストデータは初期状態からの連続した操作を行う表現とする．

本リファインメントでは上記により定義したモデル構造に対して，詳細化されたことを検証する．検証イメージを以下の図に示す．

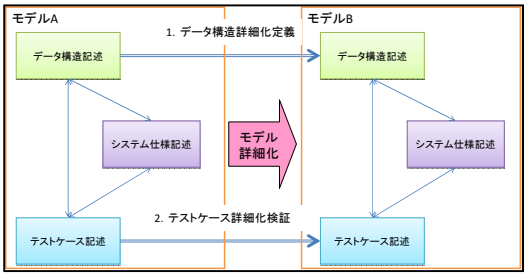

図 3　リファインメント検証イメージ

以下，各種記述方法について説明する．

3.3.3.1.　モデル記述定義

分類に基づき，モデルの記述方法を定義する．制約を与えることで記述への定義とする．

3.3.3.1.1.　データ構造記述

入出力に関するデータ構造を以下の制約で記述する．

(制約)

- データ構造記述に閉じた VDM++ Class を定義する
- モデル化方針に応じたシステムにおける入出力データ型を type 節で規定する
- 各データ型に対して functions 節でデータの許容される値域を規定する
- システムに依存しない普遍的な定数制約は value 節で記述する
- operations 節，instance variables 節を記述しないこと
- 出力に関する例外については，クラス定義をおこなうこと

上記制約を設けた根拠は，以降のモデル記述において，システム記述への継承，テストケース記述への継承が必要となるため，入出力データ定義のみ分離して記述する制約を与えた．

仮にシステム仕様記述とデータ構造記述を同一 Class 内で定義した場合，テストケース記述を行う際に，不要な operation 節や instance variables 節の継承が伴うため，概念的および作業実行上望ましくない．

例とするモデルでは，モデル E はログイン情報という token が type 定義され，モデル F ではログイン ID，パスワード，ブラウザ識別子という seq of char が type 定義される．また，処理結果として，モデル E は，ENUM の集合{E_ログイン成功，E_ログイン失敗}，{E_利用成功，E_利用失敗}が定義され，モデル F は ENUM の集合{F_ログイン成功，F_ログイン失敗}，{F_利用成功，F_利用失敗，F_再ログイ

ン要求}が定義される．特に再ログイン要求は，詳細化でブラウザ識別子に伴う処理結果の詳細化となる．

3.3.3.1.2. システム仕様記述

システムの仕様(ロジック)に関するシステム仕様記述を以下の制約で記述する．

(制約)

- operation 節あるいは function 節に対して，システム分析により抽出されたシステム境界インタフェースを具備した VDM クラスを定めること．ただし，複数クラスを定めてもよい
- システムインタフェースに対応する operation 節あるいは function 節の入出力型はデータ構造記述の定義のみを利用すること．ただし，システム内での分割およびその入出力について制限は発生しない
- システム上発生する条件に対して仕様として異常とする場合は，異常制御を事前条件ではなく，制御内に記述すること

これら制約は詳細化過程において，システム分析により抽出された外部インタフェースの数とそれぞれの意味は変わらないことを意図する．加えて，それぞれのインタフェースに対して，データ構造で定義したデータ型のみを入出力に利用することで，後のリファインメント検証をわかりやすいものとする．

例とするモデルでは，モデル E はログイン(引数 : ログイン情報)という操作と利用(引数：ログイン情報)という操作が定義し，モデル F ではサブモデル F_1 にログインという操作(引数：ログイン ID，パスワード，ブラウザ識別子)と利用(引数：ログイン ID，パスワード，ブラウザ識別子)という操作が定義し，サブモデル F_2 はサブモデル間のみの制御を受け付けるとする．

3.3.3.1.3. テストケース記述

システム仕様を検証し，入出力データ集合による仕様表現としてのテストケース記述を以下の制約で記述する．

(制約)

- テストケースはデータ構造記述により定義された入出力型により定義されること
- テストケース記述は，リファインメント検証方針に応じて，適切な網羅性が設定されていること．
- テストケースはシステムインタフェースに対応する operation あるいは function に対しての実行に限定する
- operation に対するテストはモデル初期化からのシーケンスを実行するテストケースとすること
- テストケース記述はテスト実行とデータでクラスを分離すること

上記制約を設けた根拠は，統一された入出力データ列による仕様のデータ表現に射影するためである．つまり，初期化直後からの振る舞いをデータ列として取り扱うことで，ロジックを表現する．このような射影により得られたデータ列を後述されるデータ構造詳細化定義からモデルの同一性を確認する．射影されたデータ間で同一性が確認できなければ検証対象のモデルは同一ではないことになる．

例とするモデルでは，モデル E はログイン時に設定した token と同じ token による操作しか受け付けないことを検証するテストデータを設定し，モデル F では，ログイン ID とパスワードが正しい場合，かつログイン時に設定したブラウザ識別子を伴う操作しか受け付けないことを検証するテストデータを設定する．

3.3.3.2. リファインメント検証モデル定義

前節において，個別モデル記述の定義を与えた．本節では，2 種のモデルに対して，リファインメント検証，すなわちモデルが詳細化されていること検証するためのモデル定義を与える．なお，リファインメント検証モデルも VDM++により記述する．

以下，詳細化前モデルをモデル A と呼び，詳細化後モデルをモデル B と呼ぶ．

3.3.3.2.1. データ構造詳細化定義

データ構造詳細化定義とは，詳細化前後のモデルに対して同一視写像を与えることを意味する．

(定義)

1-1.F_d は，各データ項目に対して，概念的に意味が変わらないこと．すなわち，設計者の意図する詳細化となっていること

1-2.F_d は，D_a の 1 要素と D_b の 1 以上の複数要素について対応を与えること．また，D_b の 1 要素に対して，複数の D_a は対応しない．すなわち，複数の概念が混在し，再構成されていることがないこと．

1-3.F_d の定義域は D_b 全体であること

空間 D_x とは，データ構造記述により定義されたデータ型と function で定義された値範囲により定められたデータ全体を意味する．

写像定義で与えた制限設定の根拠を以下に示す．

1-1 : 本詳細化で必要な異なるデータ型の意味的紐付けを定める．本内容は仕様記述を行うものが概念データモデル等との関連性から正しく定める必要がある

1-2 : 詳細化前後でデータの対応に対して意味の混在を抑止するための制約にあたる．本制約が無い場合は外部入出力

インタフェースの意味が変わる可能性を許すことになる
1-3 : 写像の定義域が詳細化後に許容されるデータパターンが増加することを抑止する．詳細化過程において，詳細化後に取扱うデータは必ず，詳細化前データに対応することで検証時の矛盾を除去する

なお，データ構造詳細化定義では，出力についての対応関係が仕様記述詳細化の自由度へ影響する．
例えば，詳細化前では入力パラメータのバリデーションチェックが行われないモデルで詳細化後モデルでは入力のバリデーションチェックを行うモデルであると仮定する．その場合は，出力の同一な対応関係として，処理成功と処理成功が対応し，加えて，処理成功と入力データ違反が対応する．
これは，詳細化前モデルでは，入力データ違反を考慮せず，処理成功として取扱う仕様に対して，詳細化後仕様では入力データ違反と処理成功を区別して取扱う詳細化が行われていることに対応する．
例とするモデルでは，モデル E とモデル F の同一視写像は，mk_token(ログイン ID ^ パスワード)=ログイン情報と写像{F_利用成功 |-> E_利用成功, F_再ログイン要求 |-> E_利用成功, F_利用失敗 -> E_利用失敗}, {F_ログイン成功 -> E_ログイン成功, F_ログイン失敗 E_ログイン失敗}により与える．

3.3.3.2.2. テストケース詳細化検証

定義した同一視写像によりテストケースを検証する．
テストデータは，入出力の列の集合であると定義した．この入出力の列について，同一視写像により詳細化前後のデータが対応しているか検証が行われる．検証すべき内容を以下に記述する．

(定義)
モデル A のテストケース集合 T_a とモデル B のテストケース集合 T_b に対して，T_b が T_a の詳細化であるとは，F_t : T_b -> T_a が存在し，以下の条件を満たすことを意味する．
(テストケース集合定義)
T_x = set of Seq of test_unit;　(x = a, b)
Test_unit ::
　operation_name;
　input;
　output;
(写像定義)
2-1.∀t_a∈T_a　∃t_b∈T_b　st F_t(t_b)=t_a
2-2.∀t_b∈T_b　∃1 t_a∈T_a　st F_t(t_b)=t_a
2-3.t_a := [t_a_1, t_a_2, ･･･, t_a_n]　　(n ∈ nat1)
　　⇒ F_t(t_a) := [f_t(t_a_1), f_t(t_a_2), ･･･, f_t(t_a_n)]
2-4.t_a_n := (operation_name, input, output)
　　⇒ f_t(t_a_n) := (f_t_op(operation_name), f_t_in(input), f_t_out(output))
2-5.f_t_op := 実行 operation に関するモデル B からモデル A への一対一対応
2-6.f_t_in = f_d_in　(f_d_in は，F_d の入力データによる制限)
2-7.f_t_out = f_d_out　(f_d_out は，F_d の出力データによる制限)

本定義は，詳細化後の入出力データの列に対して，同一な詳細化前の入出力データの列が唯一存在し，また詳細化前の入出力データの列に対して漏れなく詳細化後の入出力データの列が存在することを以って，詳細化が行われていることを意味する．
写像定義で与えた内容に関して以下に説明する．

2-1 : すべての詳細化前入出力データの列に対して，対応する詳細化後入出力データの列が一つ以上存在することを確認する．これにより，仕様反映の漏れを検出する．
2-2 : すべての詳細化後入出力データの列に対して，対応する詳細化前入出力データの列が唯一存在することを確認する．これにより，詳細化後の設計者の意図しない仕様追加を検出する．
2-3 : テストケースの対応は，テストケース内の各実行内容の対応に分解されることを定義している．これにより，操作個別の対応を確認することを示している．
2-4 : 各実行内容の対応は，実行オペレーションの対応，入力の対応，出力の対応に分解されることを示している．
2-5 : 実行対象(外部インタフェース)が同一のものである対応を示している．
2-6 : 入力が同一視写像により同一であることを確認する．
2-7 : 出力が同一視写像により同一であることを確認する．

以上から，同一な操作を行った場合，同一な結果が得られることを確認することから，同一モデルであることの必要性が検証されたこととなる．(網羅性の粒度が任意となるテストを前提とするため，同一であることの十分性は検証できない．)
例のモデルでは，モデル F でログイン-成功，利用-成功，利用-再ログイン要求という制御に対して，同一視写像により得られたデータによりモデル E を動作させた場合，ログイン-成功，利用-成功，利用-成功となる．これは，モデル E で再ログインという概念がないため，設計者の意図で成功と同義と同一視写像を定義したことにより，同じ振る舞いとして扱う．このように抽出したテストデータについてすべてのマッピングをとり，同一データによる操作を行った結果が等しいとき，テストデータに定める範囲において

リファインメントされたモデルであると言える．

3.3.3.3. 検証内容整理

詳細化定義においては，モデル間のデータ同一性に基づき，仕様を射影表現した入出力データ列の集合に対する対応関係を検証することで詳細化であるかを確認した．
これは，図 4　検証イメージで示すようなシステム仕様記述をテストデータに同一視可能な変換が得られた場合，モデルとして同一であるという十分条件になることとも関連する．実際，仕様記述モデルをテストデータに変換するには膨大なデータ記述が必要なため，現実的ではないが，効率的な変換が可能であれば必要十分条件を得ることも可能である．

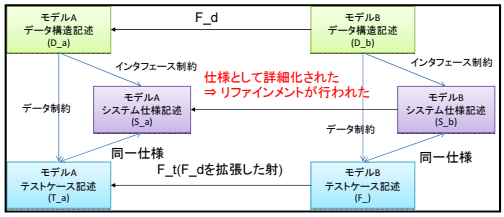

図 4　検証イメージ

実際，例においても設計者の意図する詳細化が行われているかの検証を行ったが，すべての入出力について検証を行っているわけではない．この例では，モデルを分割しても同じ動作となるか，再ログインの概念を導入してもほかの制御を損なわないかの検証を主眼とした場合のテストケース抽出とした．さらに，異常系制御についても詳細化検証を行いたい場合は，その観点でのテストデータの同一性検証を行うことで対応可能である．

3.4 モデル詳細化手順例

今までの検証方法について実施手順例を以下に示す．

1.モデル A を記述する

1-1.モデル A のデータ構造を記述する(データ形式を決定する)
1-2.モデル A のシステム仕様を記述する
1-3.モデル A のテストケースを記述する(テスト実行まで)

2.モデル B とリファインメント検証モデルを記述する

2-1.モデル B のデータ構造を記述する
2-2.モデル A とモデル B の間でのデータ構造詳細化定義を行う
2-3.モデル B のシステム仕様を記述する
2-4.モデル B のテストケースを記述する(テスト実行まで)
2-5.モデル A とモデル B の間でのテストケース詳細化定義を行う
2-6.モデル A とモデル B の入出力パラメータを調整する(リファインメント検証モデルが正しい場合でも，具体的な値の対応関係に調整が必要になる)

2-6 では，テストデータが同一であるかを検証するが，モデル A と B でそれぞれ独立してテストデータを決めた場合は，一致することはほぼない．したがって，2-4 でテストデータを定義する際には 1-3 で定義したテストデータと同一なテストデータを用意することを意識する必要がある．

4. 課題

テストケース・リファインメントを定義することで，オブジェクト指向でのモデル詳細化に関する論理的な整合性(同一性)の検証方法が得られた．前述したとおり，テストケースの条件に依存してロジックの同一性をどれほど検証できるかの粒度が異なるため，本検証手法は同一であることを証明するのではなく，同一ではないものを検出する手法といえる．これは，他の定理証明に基づいたリファインメントに比べライトウェイトと言える．また，リンク(接着)不変条件を与える代わりに本手法では同一視写像を与える点もライトウェイトといえる．

しかし，ライトウェイトな分，テストケースにより検証精度が左右される．どのようなテストケースによりどのような観点での非同一性が検出されるかを明らかにしない状態では，品質確認が行える検証として本手法を導入することは難しい．

したがって，本手法はどのようなテストデータを用いることでどのような同一性検証が行えるかの整理が今後必要である．

謝辞　本手法提案に至った機会を与えてくださった国立情報学研究所 トップエスイープロジェクトの皆様, NTT ソフトウェアの皆様に感謝いたします．

参考文献

[1]　Gojko Adzic: Specification by Example: How Successful Teams Deliver the Right Software, Manning Pubns Co , 2011.
[2]　石川 冬樹: VDM++による形式仕様記述 (トップエスイーシリーズ 実践講座), 近代科学社, 2011
[3]　来間 啓伸: B メソッドによる形式仕様記述―ソフトウェアシステムのモデル化とその検証 (トップエスイー実践講座), 近代科学社, 2011
[4]　中島 震, 来間 啓伸: Event-B: リファインメント・モデリングに基づく形式手法, 近代科学社, 2015
[5]　中島 震: 形式手法入門 ロジックによるソフトウェア設計, オーム社, 2012.
[6]　VDM information web site
http://www.vdmtools.jp/
[7]　Home of Event-B and the Rodin Platform
http://www.event-b.org/