社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－SE－159 (20)
2008／3／18

ユースケース記述に対するフレームワーク利用法の導出

善明晃由[†1] 小林隆志[†2] 佐伯元司[†1]

フレームワークを用いたソフトウェア開発では、ホットスポットの特定やカスタマイズを支援することが重要である。本稿では、分岐条件を基にフレームワークのユースケース記述に対する利用法を導出する手法を提案する。提案する手法では、フレームワークとユースケース記述の終了状態に注目し、それらの対応関係を仮定することで、フレームワークとユースケース記述の分岐条件の対応付けを充足可能性の判定問題に帰着させる。充足可能と判定された場合は、分岐条件の真偽値割当てよりフレームワークの利用法を識別できる。

Deriving Framework Usages for a Usecase Description

Teruyoshi Zenmyo ,[†1] Takashi Kobayashi [†2]
and Motoshi Saeki[†1]

In software development using application frameworks, support for identifying and customizing hot-spots is essential. This paper proposes a technique for deriving framework usages for a use case description based on branch conditions. For relating branch conditions of the use case description to one of the framework, the procedure for making the relations is regarded as a satisfiability problem by assuming relations of final conditions. The framework usages are identified by using the truth assignment of the branch conditions.

1. はじめに

高品質なソフトウェアを短期間で開発するための技術として、フレームワークが注目されている。フレームワークとは、あるドメインに共通なアプリケーションの枠組みを提供するソフトウェア部品である。

フレームワークを用いたソフトウェア開発では、ホットスポットと呼ばれるフレームワークの可変部分をカスタマイズすることで、目的のソフトウェアを実現する。このため、ソフトウェア開発者は利用するフレームワークを学習し、目的のソフトウェアを実現するために必要なホットスポットとそのカスタマイズ方法を理解する必要がある。

しかし、一般にフレームワークはホットスポットを提供するために高度な設計がなされており、経験の少ない開発者がその利用法を理解するのは困難である。そこで、計算機によってフレームワークを用いたソフトウェア開発を支援する必要がある。

我々は、フレームワークと要求仕様の振舞いに注目し、両者の対応付けを形式化することでフレームワークを用いたソフトウェア開発を支援する手法を提案している[1)]。この手法は、フレームワークとシナリオ記述をラベル付き遷移システム (LTS) でモデル化し、プロセス代数の等価性の一つである観測同値性[2)] に基づいて、シナリオ記述中の処理とその処理を実装できるフレームワークのホットスポットの対応関係を導出する。

しかし、この手法は、処理とホットスポットをそれらが実行される順序関係に基づいて対応付けるものであり、処理が条件によって分岐するような状況があつかえない。このため、複数の動作系列の分岐を含むユースケース記述とフレームワークとの対応付けが不十分であった。

本稿では、分岐条件を基に、ユースケース記述に対するフレームワークのホットスポットの利用法を導出する手法を提案する。提案する手法では、ユースケース記述とフレームワークの終了状態に注目し、それらの対応関係を仮定することで、分岐条件の対応付けを

†1 東京工業大学 大学院情報理工学研究科 計算工学専攻
Department of Computer Science, Graduate school of Information Science and Engineering, Tokyo Institute of Technology
†2 名古屋大学 大学院情報科学研究科付属 組込みシステム研究センター
Center for Embedded Computing systems (NCES), Graduate School of Information Science, Nagoya University

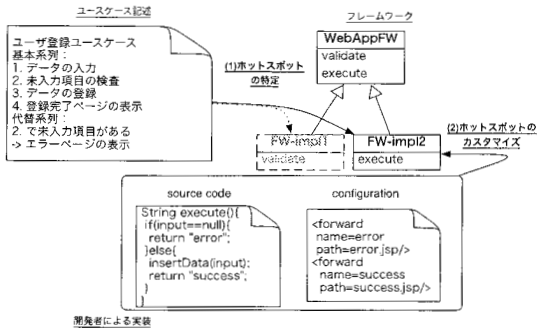

図 1 フレームワークの利用

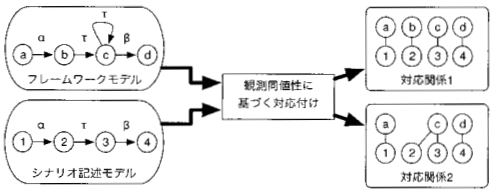

図 2 要求仕様との対応付けによるフレームワークの利用支援

充足可能性の判定問題とみなす。ユースケース記述とフレームワークの双方の分岐条件を対応付けることにより、フレームワークのホットスポットの利用法を特定することが可能となる。

2. ラベル付き遷移システムに基づくフレームワーク利用支援

2.1 観測同値性に基づくシナリオ記述とフレームワークの対応付け

フレームワークを利用したソフトウェア開発における作業を図 1 に示す。まず、ソフトウェア開発者は、目的の要求仕様と利用するフレームワークを比較し、開発者が新たに実装する必要がある処理とそれを実現できるフレームワークのホットスポットを特定する。次に、フックメソッドのオーバライドや設定ファイルを記述することによってホットスポットをカスタマイズする。

我々は、シナリオで記述された要求仕様とフレームワークの対応付けを形式化することにより、ホットスポットの特定やカスタマイズを計算機によって支援する手法を提案している (図 2)[1)]。対応付けを計算機によって行うことにより、目的の要求仕様に対するフレームワークの適合性の判定や利用可能なホットスポットの特定を自動化する。さらに、得られた対応関係を基にフックメソッドのスケルトンコードの生成すること

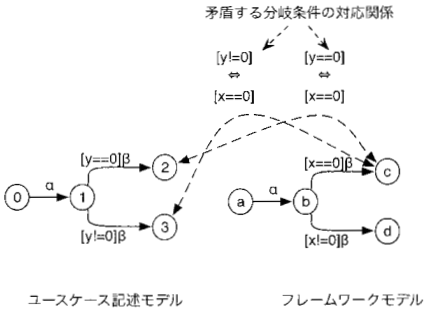

図 3 分岐条件の矛盾

で、ホットスポットのカスタマイズも支援する。

[1)] の手法では、要求仕様とフレームワークの振舞いをラベル付き遷移システム (LTS) でモデル化し、それらをプロセス代数の等価性の一つである観測同値性[2)]に基づいて対応付ける。ここで、ホットスポットは内部動作の繰り返しを持つ状態としてモデル化する。観測同値性は 2 つのプロセスの外部から観測可能な動作の順序が等しいことであり、内部動作は考慮されない。このため、内部動作の繰り返しとしてモデル化されたホットスポットは、要求仕様に記述される任意の処理と対応付けることができる。

2.2 問 題 点

分岐を含む振舞いでは、実行される動作系列が分岐条件の真偽で決まる。そこで、複数の動作系列の分岐を含むユースケース記述に対してフレームワークの利用を支援するためには、ユースケース記述の分岐条件とフレームワークの分岐条件をその真偽値割当てまで含めて対応付ける必要がある。

2.1 節で述べた手法は、一つの動作系列を対象としたものであり、複数の動作系列が存在する場合、それらを分岐条件やその真偽値割当てを無視して独立に対応付ける。このため、例えば、図 3 のように矛盾した対応付けを導出することがある。図 3 では、ユースケース記述とフレームワークが分岐条件付き LTS でモデル化されており、ユースケース記述モデルの状態 2 と状態 3 が、共にフレームワークモデルの状態 c に対応付けられている。ここで、それぞれ、状態 2 と状態 3 に到達するために成立する必要があるユースケース記述モデルの分岐条件 $[y == 0]$ と $[y! = 0]$ も、共にフレームワークモデルの分岐条件 $[x == 0]$ に対応付く必要がある。しかし、$[y == 0]$ と $[y! = 0]$ は排他的であり、一つの分岐条件で同時に充足することができないため、図 3 のような対応関係は矛盾している。

分岐条件を対応付ける際の問題として、分岐条件の

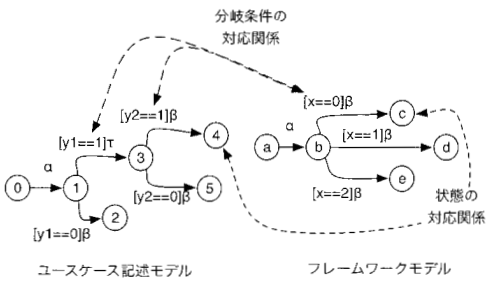

図 4 分岐条件の統合

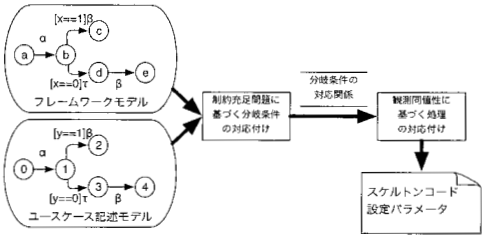

図 5 提案手法の概要

統合や複製による順序関係の変化がある。例えば、図4に示すようにユースケース記述の複数の分岐条件がフレームワークの一つの分岐条件に統合される場合がある。また、ユースケース記述の一つの分岐条件が、フレームワークの複数の分岐条件に複製される場合もある。分岐条件の対応関係における統合や複製の有無によって、フレームワークの分岐条件の順序が異なる。このため、振舞いの順序関係に基づく等価性である観測同値性では、適切な対応付けが行えない。

また、フレームワークのモデル化に関しても問題がある。フレームワークの分岐には、任意の数の枝を持つものがある。分岐条件を形式的に対応付けるには、このような分岐のモデル化が必要となる。

3. ユースケース記述に対するフレームワーク利用法の導出

2.1節で述べた観測同値性に基づく手法では、フレームワークと要求仕様の分岐条件を対応付けることができない。そこで、本節では、充足可能性の判定問題に帰着させることで、フレームワークとユースケース記述の分岐条件を対応付ける手法を提案する。

3.1 概　　要

図5に、提案手法の概要を示す。提案手法では、まず、フレームワークとユースケース記述の分岐条件を対応付け、フレームワークの分岐条件の利用法を導出する。次に、観測同値性に基づく対応付けを行いユースケース記述の実装方法を導出する。

先に分岐条件の対応付けを行うことでフレームワークモデルにおける分岐条件の順序関係が固定できる。これにより、観測同値性に基づく対応付けを行うことを可能にする。

以降、本稿では、分岐条件の対応付けについて説明する。

3.2 分岐条件の対応付けのアプローチ

2.2節で述べたように、フレームワークとユースケース記述の分岐条件を対応付ける際には、以下の問題がある。

- 分岐条件の統合や複製により順序関係が変化する
- 任意の数の枝を持つ分岐に対応するホットスポットのモデル化

提案手法では、フレームワークとユースケース記述の終了状態の対応関係を仮定し、分岐条件の対応付けを分岐条件の充足可能性を判定する問題と見なすことで、前述の問題を解決する。

ユースケース記述の終了状態とは、ユースケースが終了したときにとりうる状態である。ユースケースに代替系列が含まれる場合は複数の終了状態が存在する。例えば、図5中のユースケース記述モデルでは、状態2と状態4が終了状態である。一方、フレームワークの終了状態は、フレームワークの処理が終了したときにとりうる状態である。ユースケース記述と同様に制御構造に分岐を含むフレームワークは、複数の終了状態を持つ。例えば、図5中のフレームワークモデルでは、状態cと状態eが終了状態である。

終了状態は、ユースケース記述とフレームワークの両方において、その終了状態に到達するために成立する必要がある分岐条件の組み合わせとして表現される。例えば、図5のフレームワークモデルの状態eは、分岐条件 $[x == 0]$ で表現できる。そこで、ユースケース記述とフレームワークの終了状態が対応付くためには、各終了状態に関係する分岐条件を同時に真にする真偽値割当ての存在が必要となる。すなわち、終了状態の対応関係の妥当性の判定は、終了状態に関係する分岐条件の充足可能性を判定する問題と見なせる。ここで、分岐条件の充足が可能と判定された場合、分岐条件を充足可能な真偽値割当てよりフレームワークとユースケース記述の分岐条件の対応関係を特定できる。また、充足が可能でない場合、その終了状態の対応関係が妥当でないことがわかる。

また、任意の枝を持つ分岐のモデル化は、終了状態の対応関係の仮定に基づくフレームワークモデルの変

新規注文ユースケース
基本系列：
(1)　利用者は注文データを入力する。
(2)　システムは未入力の項目がないか確認する。
(3)　システムは発送先の変更がないか確認する。
(4)　システムは注文確認画面を表示する。
代替系列：
3a　2 で未入力の項目がある場合は、注文入力画面を表示する。
4a　3 で発送先の変更がある場合は、発送先入力画面を表示する。

図 6　ユースケース記述の例

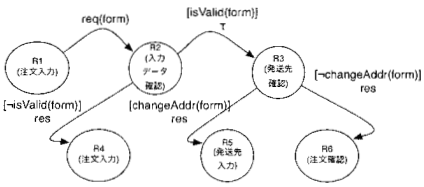

図 7　ユースケース記述のモデル

換により実現する。フレームワークモデルでは、任意の枝を持つ可変な分岐と固定数の枝を持つ不変な分岐を区別してモデル化する。可変な分岐は、終了状態の対応関係の仮定に基づいて、その仮定の基で存在しうる枝のみを持つ不変な分岐に変換する。

4. 適 用 例

本節では、ネット通販システム JPetStore[3] における新規注文ユースケースを Struts フレームワーク[4] を用いて実装する場合を例に、提案手法の詳細を述べる。

4.1 ユースケース記述モデル

本節で用いる新規注文ユースケースのユースケース記述を図 6 に示す。図 6 のユースケース記述は、分岐条件として未入力項目の有無と発送先変更の有無を含み、それぞれ一つの代替系列を持っている。

図 6 のユースケース記述を LTS でモデル化したものを図 7 に示す。また、図 6 と図 7 の対応関係を表 1 に示す。表 1(a) はユースケース記述における動作、表 1(b) は分岐条件のモデル上での表現を説明したものである。

ユースケース記述の分岐条件は述語を用いてモデル化する。図 7 のモデルでは、分岐条件が入力データ $(form)$ を引数とする述語として表現されている。これは、分岐条件の真偽は入力データによって決まるためである。

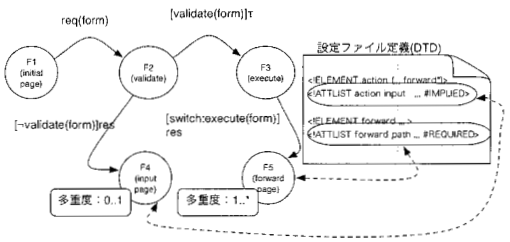

図 8　Struts のモデル

新規注文ユースケースは 3 つの終了状態を持つ。それぞれの終了状態と分岐条件の関係を以下に示す。

- 未入力項目があり $(\neg isValid)$、注文入力画面が表示される。
- 未入力項目がなく発送先変更があり $(isValid \wedge changeAddr)$、発送先入力画面が表示される。
- 未入力項目、及び発送先変更がなく $(isValid \wedge \neg changeAddr)$、注文確認画面が表示される。

4.2 フレームワークのモデル

Struts のモデルを図 8 に示す。また、図 8 の各状態を以下に説明する。

F1　利用者からの入力を待っている状態
F2　`ActionFrom` クラスの `validate` メソッドを実行している状態
F3　`Action` クラスの `execute` メソッドを実行している状態
F4　設定ファイル (`struts-config.xml`) の `input` 属性に指定されるページが表示されている状態
F5　設定ファイルの `path` 属性に指定されるページが表示されている状態

ユーザの入力に対する Struts の処理では他のメソッドも呼ばれるが、本節では簡素化のため、`validate` メソッドと `execute` メソッドのみに注目する。

Struts の処理は、`validate` メソッドの返り値と `execute` メソッドの返り値によって分岐する。`boolean` 型の返り値型を持つ `validate` メソッドによる分岐は、有限 (2 つ) の枝を持つ。図 8 のモデルでは、$[validate(form)]$ と $[\neg validate(form)]$ という排他的な分岐条件によって、`validate` メソッドによる分岐が表現されている。

これに対して、`execute` メソッドは `ActionForward` クラスのインスタンスを返し、その後の処理はそのインスタンスの内容によって分岐する。このため、`execute` メソッドによる分岐の枝の数は任意である。提案手法において、このような任意の数の枝を持つ分岐は、switch 型の分岐条件を用いて表す。状態 F3 の後の分

ユースケース記述	モデル（図 7）
1	R1 $\xrightarrow{req}$ R2
2	R2
3	R3
4	R3 $\xrightarrow{res}$ R6
3a	R2 $\xrightarrow{res}$ R4
4a	R3 $\xrightarrow{res}$ R5

(a) 動作の対応

ユースケース記述	モデル（図 7）
未入力項目あり	$\neg isValid(form)$
未入力項目なし	$isValid(form)$
発送先変更あり	$changeAddr(form)$
発送先変更なし	$\neg changeAddr(form)$

(b) 分岐条件の対応

表 1 ユースケース記述とモデルの関係

	R4	R5	R6
パターン 1	F4	F5	F5
パターン 2	F5	F4	F5
パターン 3	F5	F5	F4
パターン 4	F5	F5	F5

表 2 仮定した終了状態の対応関係

岐は execute メソッドの結果によって任意の数の枝を持つことを、図 8 の $[switch : execute(form)]$ は示している。

図 8 のモデルにおいて終了状態は、状態 F4 と状態 F5 である。後述する対応付けを効率化するため、フレームワークモデルの終了状態には多重度を付加する。ここで、多重度とは、各終了状態と対応付くことが可能なユースケース記述モデルの終了状態の数である。多重度を導入することによって、仮定する終了状態の対応関係を制限でき、分岐条件の対応付けを効率化できる。図 8 において、初期状態 (F1) から F4 へ到達するパスが一通りであり、設定ファイルの input 属性が省略可能であるため、終了状態 F4 の多重度は 0..1 とする。また、終了状態 F5 の多重度は、execute メソッドの返り値に応じて任意の数のパスが存在し、設定ファイルの path 属性が省略不可能であるため 1..* とする。

4.3 フレームワーク利用法の導出

4.3.1 終了状態の対応関係の仮定

提案手法では、まず、フレームワークのモデルとユースケース記述のモデルにおける終了状態の対応関係を仮定する。本節の例では、フレームワークモデルの終了状態の多重度に基づいて、終了状態の対応関係は表 2 のように仮定される。多重度を用いることで、仮定するパターンの数を限定し、後の対応付け手続きを効率化できる。

表 2 は、終了状態の仮定として 4 つのパターンがあることを示している。例えば、パターン 1 では、図 7 のユースケース記述の終了状態 R4、R5、R6 が、それぞれ図 8 フレームワークのモデルの F4、F5、F5 に対応すると仮定している。

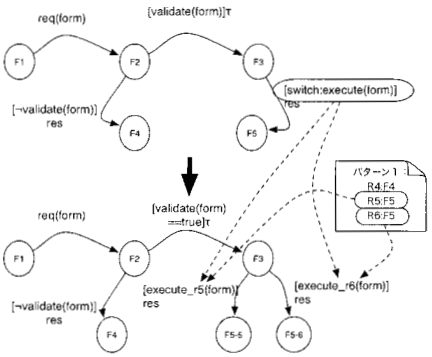

図 9 switch 型分岐条件の有限化

4.3.2 任意の数の枝を持つ分岐の変換

終了状態の対応関係を仮定することで、任意の数の枝を持つ分岐を有限な枝を持つ分岐に変換できる。有限な枝を持つ分岐への変換の例を図 9 に示す。図 9 は、終了状態の対応関係として表 2 のパターン 1 を仮定した場合の例である。パターン 1 を仮定した場合、execute メソッドの返り値による分岐の枝は 2 つである。そこで、図 8 のモデルにおける分岐条件 $[switch : execute(form)]$ は、二つの分岐条件 $[exectute\_r5(form)]$ と $[execute\_r6(form)]$ に変換できる。

4.3.3 充足可能性の判定問題への帰着

終了状態の対応関係は、フレームワークモデルの終了状態とユースケース記述モデルの終了状態の組み合わせを一つ以上含む。仮定した終了状態の対応関係が妥当であるためには、全ての終了状態の組み合わせについて、関連する分岐条件を同時に真にする真偽値割当ての存在が必要である。

例えば、表 2 のパターン 1 が妥当であるためには、以下の 3 つの条件が必要である。

条件 1 $\forall form.$
$(\neg isValid(form) \leftrightarrow \neg validate(form))$

条件 2 $\forall form.$

$form$	$isValid$	$changeAddr$
$f1$	$True$	$True$
$f2$	$True$	$False$
$f3$	$False$	$True$
$f4$	$False$	$False$

表 3 $form$ のドメイン

$form$	$validate$	$execute\_r5$	$execute\_r6$
$f1$	$True$	$True$	$False$
$f2$	$True$	$False$	$True$
$f3$	$False$	$False$	$False$
$f4$	$False$	$False$	$False$

表 4 フレームワークモデル中の真偽値割当て (充足可能性問題の解)

$form$	$validate$	$execute\_r5$	$execute\_r6$
$f1$	$True$	$True$	$False$
$f2$	$True$	$False$	$True$
$f3$	$False$	$\star$	$\star$
$f4$	$False$	$\star$	$\star$

表 5 フレームワークモデル中の真偽値割当て (モデルの構造を用いた簡素化)

$$(isValid(form) \wedge changeAddr(form)) \leftrightarrow (validate(form) \wedge execute\_r5(form)))$$

条件 3 $\forall form.$
$$(isValid(form) \wedge \neg changeAddr(form)) \leftrightarrow (validate(form) \wedge execute\_r6(form))$$

図 7 のユースケース記述モデルの分岐条件を構成する述語は、$isValid$ と $changeAddr$ 2つである。そこで、本節の例における form のドメインを表 3 に定義する。表 3 のドメインを用いることで、例えば、前述の条件 1 は以下のように変換できる。

$validate(f1) \wedge validate(f2)$
$\wedge \neg validate(f3) \wedge \neg validate(f4)$

条件 2、3 についても同様に変換することで、パターン 1 の妥当性は以下の式に対する充足可能性問題を解くことで確認できる。

$validate(f1) \wedge validate(f2) \wedge \neg validate(f3) \wedge \neg validate(f4)$
$\wedge(validate(f1) \wedge execute\_r5(f1)) \wedge \neg(validate(f2) \wedge execute\_r5(f2)) \wedge \neg(validate(f3) \wedge execute\_r5(f3)) \wedge \neg(validate(f4) \wedge execute\_r5(f4))$
$\wedge \neg(validate(f1) \wedge execute\_r6(f1)) \wedge (validate(f2) \wedge execute\_r6(f2)) \wedge \neg(validate(f3) \wedge execute\_r6(f3)) \wedge \neg(validate(f4) \wedge execute\_r6(f4))$

この充足可能性問題を解くと、フレームワークモデルに含まれる述語の真偽値割当てが、表 4 のように求まる。また、図 9 のモデルにおいて、 述語 $execute\_rs5$ と述語 $execute\_rs6$ が評価されるのは述語 $validate$ が $True$ と評価される場合のみである。そこで、真偽値割当てを表 5 のように簡素化できる。表 5 において $\star$ はドントケア ($True$ と $False$ のどちらでもよい) を表す。

isValid \ changeAddr	$True$	$False$
$True$	$True$ (f1)	$False$ (f2)
$False$	$\star$ (f3)	$\star$ (f4)

表 6 execute_r5 のカルノー図

4.3.4 分岐条件の対応付け

提案手法において分岐条件は述語で表現される。そこで、フレームワークモデル中の述語をユースケース記述モデル中の述語を用いて表現することで、分岐条件を対応付けることができる。

本節では、フレームワークモデルの述語とユースケース記述モデルの述語をカルノー図を用いて対応付ける方法を説明する。例として、終了状態の対応関係として表 2 のパターン 1 を仮定した場合の述語 $execute\_rs5$ とユースケース記述モデルの述語を対応付けることを考える。述語 $execute\_rs5$ を図 7 中の述語で構成される論理式とすると、フレームワークモデルとユースケース記述モデルで form の定義は共通であることから、$execute\_rs5$ のカルノー図として表 6 を得ることができる。

表 6 から、$execute\_rs5 := changeAddr$ と定義できる。同様にして、$validate := isValid$、$execute\_rs6 := \neg changeAddr$ と定義できる。これより、終了状態の対応関係として表 2 のパターン 1 を仮定した場合に、以下のフレームワーク利用法があることがわかる。

- `validate` メソッドに未入力項目の検査処理を実装し、未入力項目があった場合の表示ページを設定ファイルの `input` 属性に設定する。
- `execute` メソッドに発送先変更の確認処理を実装し、その結果と関連付けて、設定ファイルの `path` 属性を設定する。

4.3.5 利用法の評価

フレームワークモデルの述語は、分岐点で呼ばれるフックメソッドと関連している。例えば、述語 $validate$ は、`validate` メソッド (図 8 の状態 F2 で呼ばれる) と関連している。また、ユースケース記述モデルの述

	validate	*execute*
パターン 1	$isValid$	$\bullet changeAddr$ $\bullet \neg changeAddr$
パターン 2	$\neg isValid \vee$ $changeAddr$	$\bullet isValid$ $\bullet \neg isValid$
パターン 3	$\neg isValid \vee$ $\neg changeAddr$	$\bullet isValid$ $\bullet \neg isValid$
パターン 4	$True$	$\bullet \neg isValid$ $\bullet isValid \wedge changeAddr$ $\bullet isValid \wedge \neg changeAddr$

表 7 例題における述語の対応関係

語は、分岐条件の判定に必要な処理を示している。例えば、述語 $isValid$ は、未入力項目の検査処理 (図 7 の状態 R2 で実行される) と関連している。そこで、述語の対応関係からフックメソッドに実装する必要がある処理を識別できる。さらに、フックメソッドに実装する処理を識別することで、フレームワークの利用法を評価できる。

表 2 の全てのパターンについて、提案手法によって導出されるフレームワークモデルの述語とユースケース記述モデルの述語の対応関係を表 7 に示す。表 7 は、各終了状態の対応関係のパターン (表 2) におけるフレームワークモデルの述語の定義を示す。なお、述語 *execute* については、複数の述語に変換 (例えば、パターン 1 では $execute\_r5$ と $execute\_r6$ に変換) されるため、変換後の述語の定義を箇条書きで示す。

表 7 をみると、パターン 4 を仮定した場合、述語 *validate* は常に真で、述語 *execute* にユースケース記述モデルの 2 つ述語 ($isValid$ と $changeAddr$) が対応している。これは、パターン 4 の仮定に基づいてユースケース記述を実現する場合、カスタマイズするホットスポットは一つでよいが、一つのホットスポットが複雑になることを示唆している。

また、パターン 2 を仮定した場合、述語 *validate* と述語 *execute* メソッドの両方で未入力項目の検査に関する述語 $isValid$ が含まれている。これは、パターン 2 の仮定に基づいてユースケース記述を実現する場合、未入力項目の検査を `validate` メソッドと `execute` メソッドの両方、すなわち、二重に実装する必要があることを示唆している。パターン 3 についても同様に二重実装が必要となる。そこで、本節の例では、パターン 1 またはパターン 4 を仮定した際に導出される利用法が適切であると判断できる。

5. 関連研究

コンポーネントなどの再利用部品の検証については、様々な手法が提案されている。[5] では、コンポーネントの合成により構築されたシステムにおいて、複数のコンポーネントに関連する振舞い特性の検査手法を提案している。[6] では、並列性を管理するコントローラのパターンと、各スレッドがコントローラを正しく利用するかを検証する手法を提案している。しかし、いずれも、コンポーネントやパターンの利用法が決定した後に、時相論理などで指定された要求を検査するものであり、再利用部品の利用法を導出するものではない。

[7] や [8] などコンポーネントやアーキテクチャの選択や評価については様々な取り組みがある。[7] では、品質特性間のトレードオフを考慮したアーキテクチャ選択法が、[8] では、非技術的な要求に基づいてコンポーネントの選択を支援する手法が提案されている。しかし、いずれも、例えば、利用可能なホットスポットの識別などフレームワーク特有の課題は対象としていない。

フレームワークのような再利用部品を用いたソフトウェア開発では、再利用するソフトウェア部品に合わせて要求分析を行うことがある。また、オープンソースソフトウェアの評価と要求分析を関連付けて行うことの有効性が報告されている[9]。提案手法で得られる要求仕様とフレームワークの対応関係の[9] のような要求分析手法への応用は今後の課題の一つである。

6. まとめ

本稿では、分岐条件に基づきユースケース記述に対するフレームワークの利用法を導出する手法を提案した。提案した手法では、ユースケース記述とフレームワークの終了状態に注目し、それらの対応関係を仮定することで、分岐条件の対応付けを充足可能性の判定問題に帰着させる。

フレームワークとユースケース記述の分岐条件を対応付ける際には、分岐条件の統合や複製によって多様な対応付け方がある。提案手法では、分岐条件の充足可能性の判定問題に帰着させることで統一的な対応付けを可能にする。

また、フレームワークの分岐には枝の数が可変なものがある。このような可変な分岐は、ユースケース記述中の任意の数の枝を持つ分岐と対応付く必要がある。提案手法では、このような可変な分岐を、終了状態の対応関係の仮定に基づき固定の枝を持つ分岐に変換する。これにより、任意の数の枝を持つ分岐と対応付くフレームワークの分岐のモデル化を実現する。

これまでに、我々は、プロセス代数の観測同値性に基づいて、要求仕様に記述される処理とそれを実装できるホットスポットを対応付ける手法[1] を提案してい

る。今後は、観測同値性に基づく手法と本稿で提案した手法の統合や、フレームワークのモデルを洗練することで、フレームワークを用いたソフトウェア開発のさらなる自動化を目指す。また、ラベル付き遷移システムに基づくモデル検査[10]との連携によるフレームワークの検証手法についても検討する予定である。

参考文献

1) Zenmyo, T., Kobayashi, T. and Saeki, M.: Supporting Application Framework Selection Based on Labeled Transition Systems, *IEICE Transactions on Information and Systems*, Vol.E89–D, No.4, pp.1378–1389 (2006).
2) Milner, R.: *Communication and Concurrency*, Prentice Hall (1989).
3) JPetStore, Available at `http://ibatis.apache.org/`.
4) Apache Struts, Available at `http://struts.apache.org/`.
5) Blundell, C., Fisler, K., Krishnamurthi, S. and Hentenryck, P. V.: Parameterized Interfaces for Open System Verification of Product Lines, *Proceedings of the 19th IEEE international conference on Automated software engineering*, pp.258–267 (2004).
6) Betin-Can, A. and Bultan, T.: Verifiable Concurrent Programming Using Concurrency Controllers, *Proceedings of the 19th IEEE international conference on Automated software engineering*, pp.248–257 (2004).
7) Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. and Carriere, J.: The Architecture Tradeoff Analysis Method, Technical Report CMU/SEI-98-TR-008, Software Engineering Institute, Carnegie Mellon University (1998).
8) Carvallo, J.P., Franch, X. and Quer, C.: Managing Non-Technical Requirements in COTS Components Selection, *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pp.323–326 (2006).
9) Paech, B. and Reuschenbach, B.: Open Source Requirements Engineering, *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pp.257–262 (2006).
10) Magee, J. and Kramer, J.: *Concurrency, State Models & Java Programs*, Wiley (1999).