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

2009－SE－163 (5)
2009／3／18

要求仕様と再利用可能な実現構造の振る舞いの差分検出に基づく要求分析

朱峰 錦司[†1] 善明 晃由[†1]
林 晋平[†1] 佐伯 元司[†1]

ソフトウェア開発の要求分析工程において，要求の欠落を補填する支援法が望まれている．本稿では，パッケージソフトウェアをはじめとする，再利用可能な実現構造を知識資源として用いて，ユースケース記述における事前条件や動作系列の欠落を補填する手法を提案する．提案手法では，ユースケース記述と実現構造をラベル付き状態遷移システムに基づいてモデル化する．両モデルを合成し，実現構造の機能が実行される条件を満たしていないユースケース記述の箇所を特定することにより，条件を満たすようにユースケース記述の補填を行う．新規に SNS を開発する事例に対し，SNS パッケージ OpenPNE から構築した知識資源を用いて本手法を適用した結果，1 つのユースケース記述から，欠落が妥当に補填された 8 通りのユースケース記述を得ることができた．

Requirements Analysis Based on Differences between Behavioural Specifications and Reusable Implementation Structures

Kinji Akemine,[†1] Teruyoshi Zenmyo,[†1] Shinpei Hayashi[†1]
and Motoshi Saeki[†1]

In a requirements analysis process, supporting requirements elicitation is important. In this paper, we propose a technique to derive preconditions and events to be added to use case descriptions by using reusable implementation structures as knowledge resources. The descriptions and the implementation structures are modeled by labeled transition systems (LTS). The two models are composed, and then, the composed model is examined whether preconditions of the functions in the implementation structure do not hold. If such situation exists, the ways for completing the descriptions are identified based on the differences between two models. As a case study, we have applied the proposed technique to a use case of an SNS site with OpenPNE as knowledge resources. As a result, we have obtained eight appropriate use case descriptions from one use case description including missing requirements.

1. はじめに

ソフトウェア開発の要求分析工程において，要求の欠落を発見する支援法が望まれている．品質の高いソフトウェアを開発するためには，顧客の要求を漏れなく把握する必要がある．しかし，欠落のない要求仕様を記述することは困難であり，特にシステムの詳細な振る舞いを記述する場合，機能の欠落や例外処理などの条件分岐の欠落が生じてしまう．よって，こういった要求の欠落を発見し，補填する手法が必要となる．ここで，一般に欠落要求の補填の仕方は複数考えられるため，それらの可能性を網羅的に発見することが可能な手法が望まれる．

要求仕様の欠落を発見するには，システムが対象としている領域に関する知識資源を利用することが有用であり，知識資源を用いた様々な要求分析支援法が提案されている[1),2)]．しかし，これらの手法は発見された欠落に対して，どのような補填を行うかについての支援が不十分である．

そこで本研究は，要求仕様としてユースケース記述に注目し，ユースケース記述中の欠落の補填を支援する手法を提案する．ユースケース記述には，処理の順序関係や分岐条件が正確に記述されている必要がある．しかし，記述の際に欠落している概念があると，事前条件や基本・代替系列の記述漏れにつながってしまう．そこで提案手法では，ユースケース記述に欠落している概念を検出し，その概念に関する事前条件や動作系

†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

ユースケース名
ユーザが送信した情報を保存する．
事前条件
- ユーザがフォームに情報を入力し，送信ボタンを押した．

基本系列
(1) システムは送信された情報を受け取る．
(2) システムは情報をデータベースに保存する．
(3) システムは保存完了画面を表示する．
事後条件
- 登録完了画面が表示されている．

図 1 例題のユースケース記述

・・・
基本系列
(1) システムは送信された情報を受け取る．
(2) システムは必須項目が全て入力されているかを検査する．
(3) 必須項目が全て入力されていれば，システムは情報をデータベースに保存する．
(4) システムは保存完了画面を表示する．
代替系列
(1) 基本系列 (2) の結果，入力されていない必須項目があれば，システムは再度入力フォームを表示する．
・・・

図 2 詳細化されたユースケース記述（検査処理を行う場合）

・・・
事前条件
- ユーザがフォームに情報を入力し，送信ボタンを押した．
- 必須情報は全て入力されている．(システムは空データも登録する．)

・・・

図 3 詳細化されたユースケース記述（検査処理を行わない場合）

列の補填を支援する．補填のために用いる知識資源としては，パッケージソフトウェアをはじめとする，再利用可能な実現構造を利用する．

提案手法では，まずユースケース記述中のシステムの処理と実現構造の機能とを対応付ける．次に，実現構造でのみ存在する動作系列から，ユースケース記述において振る舞いが定義されていない条件を特定する．そして，特定された条件に基づき，ユースケース記述の補填案を分析者に提示する．

本稿の以降の構成を示す．2 節では例題を通して，本研究の目的を述べる．3 節では本研究のアプローチと提案手法の概要を述べ，4 節で例題を通して手法の詳細について述べる．5 節では，事例適用による提案手法の評価について述べる．6 節ではいくつかの関連研究を紹介し，本研究との差異を示す．7 節で本稿をまとめ，今後の課題を述べる．

2. ユースケース記述の欠落

本節では，例題を通して本研究の目的を明確にする．本研究では要求仕様として，UML のユースケース記述に注目する．

例題として，個人情報を登録する Web アプリケーションを構築する際の要求仕様について，ユーザが個人情報を登録する際のシステムの振る舞いに関する**図 1** のユースケース記述を考える．

このユースケース記述には情報の検査処理に関する概念が欠落しており，この欠落に対して 2 通りの補填の仕方が存在する．一般に，このようなユーザが入力した情報をデータベースに保存するアプリケーションを構築する際，入力された情報を保存する前に，例えば必須項目が全て入力されているかどうかなどの，情報の妥当性を検査する処理を行うことが多い．そして検査処理の結果，入力されていない必須項目があった場合は，保存処理は行わずに例外処理へ進む（**図 2**[*1]）．一方，入力される情報として空欄も許可する場合は，それをユースケース記述の事前条件に記述するべきである（**図 3**）．

このような欠落が補填されないまま開発が実装工程へ進んでしまうと，本当は必要である検査処理が実現されないことが考えられる．モジュール結合の際にこの未実装が発覚すると，大きな開発手戻りが発生してしまう．

よって本研究は，図 1 のような，事前条件や処理，および処理に関連する条件分岐が欠落してしまっているユースケース記述に対して，欠落している概念を特定し，それらを的確に補填する支援を行うことを目的とする．

3. 再利用可能な実現構造を用いたユースケース記述の欠落の補填

3.1 アプローチ

2 節で述べたような欠落がある場合，ユースケース

*1 図中の下線は図 1 との差分を示している．図 3 に関しても同様である．

記述において振る舞いが定義されていない条件が存在する．例えば，図1のユースケース記述では，入力データが妥当でない場合の振る舞いが定義されていない．そこで，このようなユースケース記述において振る舞いが定義されていない条件を特定することで，欠落している概念を検出できる．

本研究ではこれを実現するために，知識資源として，アプリケーションフレームワークやパッケージソフトウェアなどをはじめとする，再利用可能な実現構造を利用する．再利用可能な実現構造は，対象領域に関する様々な機能を提供している．また一般に，対象領域の概念に関する様々な条件（例えば，妥当性検査の有無）に対応する動作系列が網羅的に実現されている．

振る舞いが定義されていない条件を特定するために，以下の3点を考える．

- 対象としているシステムの状態
- システムの処理が実行されるための事前条件
- システムの処理が実行された後の事後条件

システムの初期状態は，ユースケース記述の事前条件である．また，処理の事前・事後条件は実現構造から得る．ある処理に関して，システムの状態がその処理の事前条件を満たしていれば，その処理が実行される．そして，処理が実行されると，その処理の事後条件によりシステムの状態が更新される．

手法の手順を以下に示す．

(1) ユースケース記述におけるシステムの処理と実現構造が提供する機能とを対応づける．
(2) システムの各状態で取りうる条件と，その状態で実行される処理の事前条件とを比較することで，ユースケース記述において振る舞いが定義されていない条件を特定する．
(3) ユースケース記述の補填案として，特定された条件を除外するような事前条件を追加するか，条件に対して振る舞いを定義するかを分析者に提示する．

3.2 提案手法概要

提案手法の概要を**図4**に示す．

提案手法では，まず実現構造とユースケース記述を，ラベル付き状態遷移システム（LTS）に基づいてモデル化する（①）．LTSは，3.1節で述べたシステムの状態と，それに基づく条件を扱えるように拡張する．

次に，ユースケース記述の処理と実現構造の機能とを対応づける．対応付けは，LTSの合成手法を適用し，両モデルを合成することで行う[3]（②）．こうすることにより，形式的に両者の対応をとることができる．

そして，得られた合成モデル上で，ユースケース記述において振る舞いが定義されていない条件を特定する．特定された箇所に対して，遷移の事前条件を満たすように，以下の3通りのユースケース記述の補填案を分析者に提示する（③）．

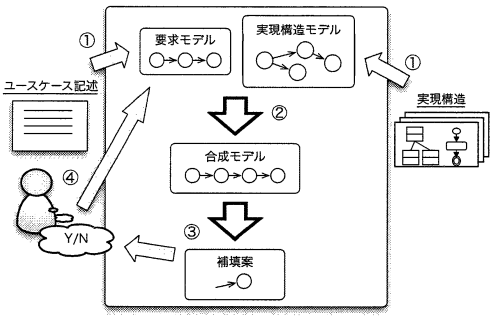

図4 提案手法概要

(1) 事前条件の追加
(2) 処理の追加
(3) 条件分岐の追加

分析者は，提示された補填案から適切なものを選択し，ユースケース記述に反映させる（④）．これにより，ユースケース記述で考慮されておらず，実現構造にのみ含まれる概念を補填することができる．提案手法では，この一連の動作を，実現構造でのみ考慮されている概念がなくなるまで繰り返すことにより，欠落の少ないユースケース記述を得る．

4. 適 用 例

本節では，2節に示した例題に対して，Webアプリケーションフレームワークである Ruby on Rails[4] を知識資源として用いて提案手法を適用する．適用例を通して，本手法の詳細について述べる．

4.1 LTSに基づくモデル化

4.1.1 実現構造のモデル化

提案手法では，実現構造をLTSに基づいてモデル化する．実現構造モデルには，遷移に事前・事後条件を付与する．実現構造が提供する機能を遷移とし，その機能の実行されるための条件を遷移の事前条件，その機能による作用を事後条件としてモデル化する．事前・事後条件は，システムの状態を表す状態変数に基づいて表現する．

実現構造の例として，Ruby on Rails のソースコードの一部を**図5**に示す．これは，2節の例題で取り上げた，保存処理や検査処理にあたるメソッド `save_with_validation` である．このメソッドでは，保存処理の前に自動で検査処理を行うかどうかを指定

```
1  def save_with_validation(
2              perform_validation = true)
3    if perform_validation && valid? ||
4                      !perform_validation
5      save_without_validation
6    else
7      false
8    end
9  end
```

図 5 Ruby on Rails のソースコードの一部

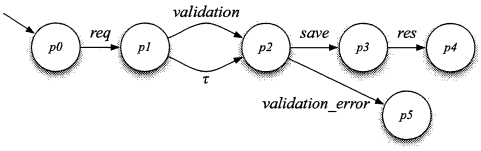

図 6 実現構造モデルの例（LTS のみ）

できるようになっている．検査処理を行った結果が不正だった場合には保存処理が失敗する．

まず，5 行目の `save_without_validation` というメソッド呼び出しが保存処理にあたる．保存処理が実行されるためには 3–4 行目に記述されている条件を満たす必要がある．この条件は，2 行目で与えられる，検査処理を行うかどうかを制御する変数 `perform_validatoin` と，検査処理の結果を真偽で返すメソッド呼び出し `valid?`によって構成されている．検査を行うように指定されたとき（`perform_validatoin==true`）は検査処理が実行され，さらにその結果が妥当だった場合（`valid?==true`）は保存処理が行われる．また，検査を行わないように指定されたとき（`perform_validatoin==false`）も，保存処理が行われる．一方，それ以外のとき，すなわち検査処理を行うように指定され（`perform_validatoin==true`），かつ，その結果が不正だった場合（`valid?==false`）は，保存処理は行われずに 7 行目へ移り失敗する．

この一連の処理は，**図 6** に示す実現構造モデルとしてモデル化できる．このとき，遷移に付与される条件は**表 1** のようになる．このモデルの構築手順を以下に示す．　まず，遷移に付与する条件に先立って，LTS のみを構築する．初期状態を用意し，外部から入力を受け取る機能として req というラベルを持った遷移を作る．次に，検査処理である `valid?`は，変数 `perform_validatoin` の値により実行される場合とされない場合があるので，それぞれの場合に対応する $validation$ と τ というラベルを持った遷移を作る．τ 遷移とは，LTS において外部から観測されない，すなわち外部から見たら何も起きていないことと等しい遷移である[5]．そして，保存処理である `save_without_validation` も実行される場合と，実行されずにエラー画面を表示する場合があるので，それぞれに対応する $save$ と $validation\_error$ というラベルを持った遷移を作る．さらに，結果を出力する処理として res というラベルを持った遷移を作る．

表 1　遷移の事前・事後条件

遷移	事前条件
$validation$	$var \in \{\text{TRUE}, \text{FALSE}\}$
τ	$var \in \{\text{NULL}\}$
$save$	$var \in \{\text{TRUE}, \text{NULL}\}$
$validation\_error$	$var \in \{\text{FALSE}\}$

表 2　状態変数 var の導入

perform_validation \ valid?	true	false
true	TRUE	FALSE
false	NULL	

次に，遷移に条件を付与する．本稿では簡単のため，前述したシステムの状態を表す 2 つの要素を**表 2** に示すようにまとめることで，3 値をとりうる 1 変数 var として扱う．　まず，検査処理 `valid?`の事前条件は，図 5 の 3 行目より `perform_validatoin==true` のときなので，$validation$ 遷移の事前条件は表 2 より $var \in \{\text{TRUE}, \text{FALSE}\}$ となる．同様に，τ 遷移の事前条件として，$var \in \{\text{NULL}\}$ を与える．以下同様に事前条件を与えることで，表 1 のように構成する．

また，この例では，処理の実行が状態変数 var に作用することはないので，事後条件を持つ遷移はない．

以上より，図 6 と表 1 で表現される実現構造モデルを構築することができた．

4.1.2　ユースケース記述のモデル化

ユースケース記述も実現構造と同様に，LTS に基づいてモデル化する．ユースケース記述におけるシステムの処理を遷移とし，ユースケース記述の事前条件は，LTS の初期状態に条件を付与することでモデル化する．初期状態の条件は，実現構造の遷移の事前・事後条件と同様に，状態変数に基づいて表現する．

例として，図 1 のユースケース記述は，**図 7** に示す要求モデルとしてモデル化できる．　まず，ユースケース記述の基本系列の 3 つの各処理に対して，それぞれに対応付く $req, save, res$ というラベルを持つ遷移を作ることで LTS を得る．次にユースケース記述の事前条件に注目する．図 1 にはユーザの行動に関する事

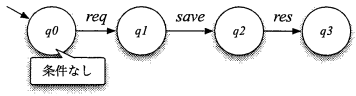

図 7 要求モデルの例

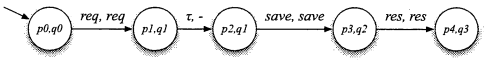

図 8 合成モデルの例

前条件が記述されているが，これは情報の保存処理や検査処理には全く関連しないものである．よって，この場合は初期条件に何も指定されていないものとする．

4.2 ユースケース記述と実現構造との対応関係

提案手法では，ユースケース記述と実現構造との機能の対応関係を得るために，共有アクションに基づくLTS の合成手法[5] を適用することで，要求モデルと実現構造モデルとを合成する．合成モデルの状態，遷移はそれぞれ要求モデルと実現構造モデルの状態，遷移を要素として持つ．

例として，4.1 節で構築した実現構造モデルと要求モデルは**図 8** に示すように合成できる．合成モデルの構築の手順を以下に示す．まず，実現構造モデルと要求モデルの初期状態を対応づけ，初期状態 $(p0, q0)$ を作る．$p0$ と $q0$ から共に req 遷移が発生するので，それらを対応付けた (req, req) というラベルを持つ遷移り，それぞれの遷移先の状態を対応づけた状態 $(p1, q1)$ を作る．次に，$p1$ と $q1$ から同じ遷移は発生しないが，$p1$ からは外部観測不可能な τ 遷移が発生するので，これに対応づく遷移を合成モデルに追加する．このとき，要求モデルでは遷移は発生しないので $(\tau, -)$ というラベルを持つ遷移とする．以下，同様に構築していくことで，図 8 の合成モデルを得ることができる．

4.3 ユースケース記述の欠落個所の特定

提案手法では，3.1 節で述べたユースケース記述において振る舞いが定義されていない条件を，4.2 節で得られた合成モデル上で特定する．まず，合成モデルの各状態に，その状態で取りうる条件（状態変数の値）の集合を与える．そして，各状態で取りうる条件のうち，遷移が不可能なものが存在するかどうかを検査する．遷移が不可能な条件があった場合は，その状態変数に関する概念がユースケース記述に欠落しているとみなす．

例えば，**図 9** は，欠落がない場合の例である．この例では，合成モデルのある状態が，v という状態変数について，$v \in \{\mathrm{x}, \mathrm{y}, \mathrm{z}\}$ という条件を持っている．そして，$v \in \{\mathrm{x}, \mathrm{y}\}$ の場合に発生可能な実現構造の遷移

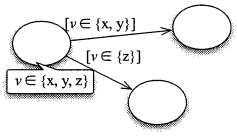

図 9 欠落がないモデル

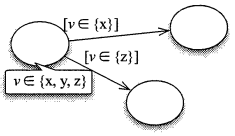

図 10 欠落があるモデル

と，$v \in \{\mathrm{z}\}$ の場合に発生可能な実現構造の遷移がある．この 2 つの遷移によって，遷移元の状態において v が取りうる 3 値とも許容されるので，この状態において欠落はないとみなす．一方，**図 10** は，欠落がある場合の例である．この例では，$v \in \{\mathrm{x}\}$ の場合に発生可能な遷移と，$v \in \{\mathrm{z}\}$ の場合に発生可能な遷移しかない．このとき，遷移元の状態において，$v \in \{\mathrm{y}\}$ のときは遷移が行えないため，欠落があるとみなす．

この検査を合成モデルの初期状態から行う．まず，初期状態の条件は，実現構造モデルで考慮されている各状態変数に対して以下のように決定する．

- 要求モデルに対象の状態変数に関する初期条件がなければ，合成モデルの初期状態ではその状態変数は定義域の全ての値を取りうることとする．
- 要求モデルに対象の状態変数に関する初期条件があれば，それを用いる．

合成モデル中のある状態において欠落がないと判断できた場合は，条件を次の状態へと伝播させ，再度この検査を行う．合成モデルに含まれる全ての最終状態まで検査が完了すれば，対応づけている実現構造に含まれる機能に関して，ユースケース記述には欠落がないとみなす．

条件の伝播方法は以下の 3 通りのいずれかに従う．

- 遷移に事前・事後条件がない場合は，遷移元の状態の条件をそのまま遷移先の状態の条件とする．
- 遷移に事後条件がある場合は，その事後条件を遷移先の状態の条件とする．
- 遷移に事後条件がなく事前条件のみがある場合は，遷移元の状態の条件と遷移の事前条件とで共通している条件を，遷移先の状態の条件とする．

例として，図 8 上で図 1 に欠落があるかどうかを検査する．実現構造モデルで考慮されている状態変数は var のみである．図 7 より，初期状態における要求モデルの条件はないので，合成モデルの初期状態 $(p0, q0)$ の条件は $var \in \{\mathrm{TRUE}, \mathrm{FALSE}, \mathrm{NULL}\}$ となる．実現構造モデルの req 遷移は事前条件も事後条件もないので，初期状態の条件はそのまま次の状態 $(p1, q1)$ へ伝播する．しかし，この $(p1, q1)$ の条件は，次の $(\tau, -)$ 遷移の事前条件を満たすことができない．実現構造モデルの τ 遷移は $var \in \{\mathrm{NULL}\}$ の場

合にして発生しないという事前条件があるため，状態 $(p1, q1)$ において $var \in \{\mathrm{TRUE}, \mathrm{FALSE}\}$ の場合は，このτ遷移が発生できない．よって図1のユースケース記述には，状態変数 var に関連する欠落があると考えられる．

4.4 欠落の補填案の提示

提案手法では，4.3節で述べた検査の結果，ユースケース記述において振る舞いが定義されていない条件が特定された場合，この条件を除外する，もしくは条件に対する振る舞いを定義するために，3.2節で述べた3通りの補填案を分析者に提示する．

まず，事前条件を追加することで，システムの取りうる状態変数を値を制限することができる．これにより，遷移が不可能な状況を回避できる．同様に，状態変数を制限する処理を追加することでも，遷移が不可能な状況を回避できる．また，代替系列を追加することで，初期のユースケース記述では考慮されていなかった状態を扱うことができる．

提案手法では，この3つの補填方法を以下に示す通りに実現する．

(1) 対象の状態から可能な遷移の事前条件を，要求モデルの初期状態の条件として設定する．

(2) 実現構造モデルから，対象の状態において許容可能な条件に状態変数を更新するような事後条件を持つ遷移を，欠落が検出された状態以前に発見し，それを要求モデルに追加する．

(3) 実現構造モデルから，対象の状態において許容不可能な条件を事前条件として持つ遷移を，欠落が検出された状態以前に発見し，それを要求モデルに追加する．

例として，4.3節で特定された振る舞いが定義されていない条件に対しては，以下のようにユースケース記述を補填できる．

- τ遷移の事前条件 $var \in \{\mathrm{NULL}\}$ を要求モデルの初期状態の条件として設定する．(補填方法(1)より)
- $var \in \{\mathrm{TRUE}, \mathrm{FALSE}\}$ を事前条件として持つ *validation* 遷移を要求モデルに追加する．(補填方法(3)より)

前者の補填を行うと，要求モデルは**図11**のようになる． これは，図1のユースケース記述に“検査処理をしない”という事前条件を追加することで，図3を得ることと同等である．この場合，本手法を再度適用するとこれ以上は欠落は発見されない．

一方，後者の補填を行った後に再度手法を適用すると，初期条件 $var \in \{\mathrm{TRUE}, \mathrm{FALSE}\}$ の追加，さらに *validation_error* 遷移の追加，が順に欠落要求の補填案として提示される．どちらも採択すると，最終的に要求モデルは**図12**のようになる． これは，図1のユースケース記述に対して，1) 基本系列への検査処理の追加，2) 事前条件への“検査処理をする”の追加，3) 検査処理の結果が不正だった場合の代替系列の追加，の3つの補填が行われたことを意味する．2の“検査処理をする”という事前条件は自明なため，これを省略すれば図12の要求モデルは図2と同等である．

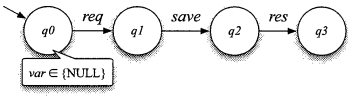

図11 補填された要求モデル（1）

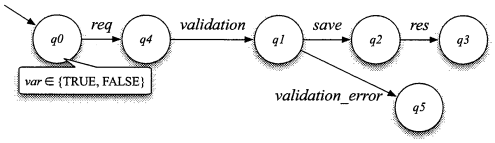

図12 補填された要求モデル（2）

以上より，例題に対して本手法を適用することで，2通りの補填されたユースケース記述を得ることができた．

5. 事例による評価

5.1 目的と事例設定

提案手法の有用性を評価するため，事例に本手法を適用する．本手法を用いることで，ユースケース記述の欠落に対して様々な補填が可能かを確認することを目的とする．

事例として，新規にソーシャル・ネットワーキング・サービス（SNS）サイトを開発する場合を想定し，既存のSNSサイトを参考に初期要求として“他人の日記を読む”処理に関する**図13**のユースケース記述を用意した．

知識資源として，オープンソースのSNSパッケージソフトウェアであるOpenPNE[6]のソースコードをもとに，以下の4つの概念を考慮した実現構造モデルを構築した．

- PCからのアクセスと携帯電話からのアクセスを分けるかどうか
- ユーザがログインしているかどうか
- PCからの閲覧時，ユーザが閲覧しようとしている日記へのアクセス権を持っているかどうか

ユースケース名
他人の日記を読む.
事前条件
- ユーザは対象の日記へのリンクをクリックした.

基本系列
(1) システムは日記の情報を受け取る.
(2) システムは情報を元にデータベースから日記の情報を所得する.
(3) システムは日記を表示する.

事後条件
- 対象の日記が表示されている.

図 13 初期要求のユースケース記述

- 携帯電話からの閲覧時，ユーザが閲覧しようとしている日記へのアクセス権を持っているかどうか

5.2 評価方法

以下に評価の流れを説明する．まず，用意したユースケース記述をもとに要求モデルを構築する．次に，要求モデルと用意した実現構造モデルから，要求モデルの欠落の補填案を得る．得られた補填案から，現実的であると考えられるものを選択し，その補填案を要求モデルに反映し再度手法を適用する．採用する補填案が複数ある場合は，全ての場合についてこれを行う．

提案手法を適用するために，本手法の支援ツールのプロトタイプを開発した．ツールは要求モデルと実現構造モデルを入力とし，モデルの合成工程と差分検出による欠落要求の発見行程を自動で行い，欠落要求の補填案を出力として提示する．今回の評価にはこの支援ツールを利用した．

5.3 結果と評価

手法適用の結果，最終的に計 12 通りのユースケース記述を得ることできた．これらのうち 8 通りは，機能の組み合わせに関して妥当な補填が行われていることを確認した．この 8 通りは，知識資源として用いた実現構造を考慮した場合に考えうる全てのユースケース記述のパターンを網羅していた．よって本手法は有用であるといえる．

一方，残りの 4 通りに関しては，例えば “PC からの閲覧時は日記の閲覧権限チェックを行うが，携帯電話からの閲覧時は権限チェックを行わない” のように，権限チェックの機能の有無がアクセス形態によって異なるものであった．これは，実現構造内で権限チェックを行うかどうかに関する状態変数が，アクセス形態ごとに個別に用意されていたからである．しかし，このような提示は現状のユースケース記述と矛盾していることが自明であり，分析者がそのような補填案の選択を避けることは容易であると考える．

6. 関連研究

要求仕様の欠落の発見については，様々な手法が提案されている．

Kaiya らは，ドメインオントロジを，知識資源として利用した要求分析支援法を提案している[1]．ドメインオントロジとは，特定の領域に関する概念，および概念間の関係を定義したものである．要求文とドメインオントロジとを対応付け，推論に基づき 1) 追加するべき概念，2) 削除するべき概念，3) 再考するべき要求文，を分析者に提示する．この研究は，要求文の矛盾の発見も可能である一方，追加するべき概念を発見することを目的としており，その補填の仕方に関しては言及されていない．

Watahiki らは，自然言語で記述されるシナリオを格文法に基づいてモデル化することで，既存のシナリオを知識資源として再利用する手法を提案している[2]．既存のシナリオはシナリオセットとしてモデル化され，シナリオの一文を格文法に基づいて表現し，また文間の順序関係や依存関係が定義されている．記述中のシナリオとシナリオセットとを対応付け，推論に基づき必要なシナリオ文を分析者に提示する．この研究はシナリオ文の順序関係を考慮した補填を行うことができるが，我々の研究とは違い，分岐条件を扱うことはできない．しかし一方で，我々はユーケース記述の動作系列名と実現構造の機能名とが完全一致していることを前提としているが，Watahiki らの手法は自然言語に基づく柔軟な対応付けが可能である．

Uchitel らは，LTS に基づいてユースケース記述をモデル化し，考慮されていない状態を発見する手法を提案している[7]．LTS に状態変数変数を導入し，ある状態における条件と，その状態から発生している遷移の事前条件とを比較することで，考慮されていない条件がないかどうか検査する．この研究では，知識資源を用いていないため，分析者は対象のシステムに関する状態変数を全て把握している必要がある．

7. おわりに

本研究では，再利用可能な実現構造を知識資源として用いて，ユースケース記述の欠落の補填を支援する手法を提案した．提案手法では，まずユースケース記述と実現構造を，システムの状態を表す状態変数と，それに基づく遷移の事前・事後条件を導入したラベル付き状態遷移システムを用いてモデル化した．また，両モデルを合成することで，ユースケース記述の処理

と実現構造の機能との対応関係を表現した．そして，対応付いた機能の事前条件を満たしていないユースケース記述の箇所を特定し，条件を満たすようにユースケース記述の補填を行った．また本研究では，提案手法の有用性を確認するため，新規にSNSを開発する事例に対し，SNSパッケージソフトウェアOpenPNEから構築した知識資源を用いて本手法を適用した．その結果，1つのユースケース記述から，欠落が妥当に補填された8通りのユースケース記述を得ることができ，本手法の有用性を確認できた．

今後の課題を以下に示す．

大規模な適用実験

本研究は扱った事例は規模の小さいものであった．そのため，より大規模な適用実験を行い，効率よくユースケース記述の欠落の補填を支援できているかどうかや，得られたユースケース記述が妥当なものであるかどうかを確認する必要がある．

対応付け手法の改善

提案手法では，要求モデルと実現構造モデルとを対応づける際，同じ処理は機能名（LTSの遷移のラベル）も同じであるという前提をおいている．しかし，この前提は現実的ではないため，辞書などを用いて機能名の差異を吸収する手法が必要がある．また，提案手法では要求モデルと実現構造モデルとが初期状態から終了状態まで対応付かなければならない．より柔軟な対応付けを行うためには，モデルの部分一致に基づく対応付け手法が必要である．

矛盾した補填方法の提示への対処

提案手法では，欠落の補填案として，考えられる可能性を全て提示する．しかし，それらの中には明らかに現状のユースケース記述に矛盾したものも含まれる．また，本手法はひとつのユースケース記述について閉じており，複数のユースケース記述間で矛盾した提示をしてしまう可能性もある．こういった矛盾した補填案の提示を防ぐため，提示の際に現状のユースケース記述の状態や，過去の補填案の選択を反映する手法が必要である．

参 考 文 献

1) Kaiya, H. and Saeki, M.: Using Domain Ontology as Domain Knowledge for Requirements Elicitation, In *RE 2006: Proceedings of the 14th IEEE International Requirements Engineering Conference*, pp.189–198 (2006).
2) Watahiki, K. and Saeki, M.: Scenario Patterns Based on Case Grammar Approach, In *RE 2001: Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, pp.300–301 (2001).
3) Zenmyo, T., Kobayashi, T. and Saeki, M.: A Technique to Check the Implementability of Behavioral Specifications with Frameworks, In *APSEC 2008: Proceedings of the 15th IEEE International Conference on Asia-Pacific Software Engineering Conference*, pp. 111–118 (2008).
4) Ruby on Rails. http://rubyonrails.org/.
5) Magee, J. and Kramer, J.: *Concurrency : State Models & Java Programs*, John Wiley & Sons, Inc., (1985).
6) OpenPNE. http://www.openpne.jp/.
7) Uchitel, S., Kramer, J. and Magee, J.: Behaviour Model Elaboration Using Partial Labelled Transition Systems, In *ESEC/FSE-11: Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp.19–27 (2003).