情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SE-167 No.17
2010/3/18

ユースケース記述からの状態遷移モデル生成

高久陽平[†1]　林　晋平[†1]　佐伯元司[†1]

ユースケース記述はソフトウェア開発における要求分析工程で用いられる．しかし，自然言語での記述は非形式的なため，ユースケース記述を網羅的に分析することは困難である．そこで，本論文ではユースケース記述からの状態遷移モデルの生成法を提案する．提案手法ではまず，ユースケース記述中の自然言語記述を解析し，格フレームを用いた形式的表現に変換する．次に，動詞の類義・対義語関係を用いて格フレームから状態変数を抽出する．ユースケース記述中の動作系列間の順序関係と，ユースケース間の関係を遷移として抽出し，状態遷移モデルを生成する．生成された状態遷移モデルをモデルチェッカに適用することで，ユースケース記述の分析を支援する．提案手法を実装した支援ツールを用いてユースケース記述を分析した結果，ユースケース記述が満たしている性質とそうでない性質を正確に判別することができた．

Generating State Transition Models from Use Case Descriptions

Yohei Takaku,[†1] Shinpei Hayashi[†1] and Motoshi Saeki[†1]

Use case descriptions are often used at the requirements analysis phase in a software development process. Since descriptions written by a natural language are informal, it is difficult to analyze them exhaustively. This paper proposes a method to generate state transition models from use case descriptions. First, we transform the descriptions to case frames. Second, we generate state variables from the case frames by the synonym and antonym relationships of the verbs. Finally, we generate state transition models, extracting inner use case transitions by the execution order of each use case and inter use case transitions by the pre/post-conditions of them. We have implemented a supporting tool for automating the method. A case study for applying the tool to use case descriptions shows the feasibility of the method.

1. はじめに

ソフトウェア開発では，高品質なソフトウェアを効率良く開発することが重要である．このためにソフトウェア開発における要求分析工程では顧客の要求を正確に分析することが必要とされ，そのためにしばしばユースケース記述が用いられる．ユースケース記述は自然言語で記述されるために，要求分析者，開発者のみならず顧客等の多くのステークホルダにとっても理解し易いという利点がある．また，要求分析という観点では，システムに求められる振る舞いを明確にするという利点がある．しかし，ユースケース記述は自然言語で記述されるために，網羅的に分析することは困難である．特に大規模開発のように大量のユースケースが記述される場合，ツールによる分析が必要になる．

本論文ではユースケース記述の分析を行う手法として，いったんユースケース記述から形式的な状態遷移モデルを生成し，その後生成した状態遷移モデルにモデルチェッカを適用する手法を提案する．状態遷移モデルの分野に関しては近年，モデル検査[1)] という分析法が盛んに研究されており，SPIN[2)] や NuSMV[3)] に代表される既存のツールを用いることで高度な分析を行うことができる．故にユースケース記述を直接分析するよりも，既存のモデルチェッカを用いて間接的に分析した方がより高度な分析を低コストで実現できる．また，ユースケース記述はシステムの振る舞いをシナリオ形式で記述したものであるため，そこには時間の流れが存在する．モデル検査の多くは仕様を時相論理式で記述するために，ユースケース記述のような時間の流れと共に状態が変化するような内容を分析するのに適している．

この手法を用いる際に特に必要となる工程は，ユースケース記述から状態遷移モデルを生成することである．提案手法ではユースケース記述の事前・事後条件，各動作記述に格フレームと動詞の類義・対義語の関係に基づき状態変数を割当て，ユースケース記述の記述順序から遷移を抽出することで状態遷移モデルを生成する．提案手法を用いることにより，ユースケース記述から自動的に状態遷移モデルを生成することができる．

本論の以降の構成は以下の通りである．まず，2 節では提案手法に関連する主要な手法について説明する．3 節では，提案する状態遷移モデル生成法を説明する．4 節では，本論文で実装した状態遷移モデル生成の支援ツールについて説明する．5 節では，支援ツールの適用事例を紹介する．6 節では，関連研究について述べる．最後に 7 節で本論文をまとめ，今後の

†1 東京工業大学
Tokyo Institute of Technology

© 2010 Information Processing Society of Japan

Vol.2010-SE-167 No.17
2010/3/18

ユースケース名　ログインする **アクタ**　利用者 **事前条件**　サイトが開いている **事後条件**　ログインしている **主系列：** **1**　利用者はログインボタンを押す **2**　システムはログインしたことを伝える **代替系列：** **2.a**　既にログインしている場合 **2.a.1**　システムはログイン済みと伝える	**ユースケース名**　ログアウトする **アクタ**　利用者 **事前条件**　ログインしている **事後条件**　ログアウトしている **主系列：** **1**　システムはログアウトしたことを伝える	**ユースケース名**　注文する **アクタ**　利用者 **事前条件**　サイトが開いている **事後条件**　商品を注文している **主系列：** **1**　利用者は注文ボタンを押す **2**　システムは注文処理を行う

図 1　ユースケース記述の例

課題について述べる．

2. ユースケース記述とその検査

2.1 ユースケース記述

ユースケース記述はソフトウェア開発における要求段階に用いられ, システムとアクタとの間で行われるやり取りを実行順に自然言語で記述したものである[4]．ここで, アクタとはユーザのようなシステム外からシステムに対し何らかの事象を引き起こすもののことである．ユースケース記述の記述項目には様々な定義があるが，本論文では Larman の定義[5]に従い，以下の記述項目を扱う：

ユースケース名　何のシナリオを記述しているのかを表す，ユースケースの名称.

アクタ名　アクタとして定義される者の名称.

事前条件　ユースケースが実行される前に満たされなければならない条件.

事後条件　主系列が最後まで実行されたとき，満たされなければならない条件.

主系列　ユースケースにおいてシステムとアクタの間で正常にやり取りが行われたときのシナリオ.

代替系列　主系列がアクタの入力やシステム内部の状態により最後まで正常に実行されない場合の分岐でのやりとりを表すシナリオ.

なお, 本論文では主系列及び代替系列中のシナリオにおける動作記述単位をステップと呼ぶ.

ユースケース記述の例を**図 1** に示す．例えば，ユースケース「ログインする」はサイトが開いているときにのみ実行可能で，2 ステップの主系列を持つ．ステップ 1 の実行終了後すでにログインしている場合，ステップ 2 は実行されずにステップ 2.a.1 が実行される．そうでない場合，ステップ 2 が実行され，このときにのみ事後条件で示された条件であるログインしていることが満たされる．このように, 2.a は代替系列の実行の事前条件となる．

ステップの記述には図 1 に示した他にも，“〜へ進む” や “ユースケース「〜」を起動する” のようなものがある[6]．前者は同ユースケース記述内の他のステップへの遷移（Goto）, 後者は他のユースケース記述のステップへの遷移（Include）を表す.

2.2 モデル検査

モデル検査とはある状態遷移系が与えられた論理式を満たすかどうかを, 状態遷移系の動作を計算機上で模倣し, 網羅的に調べることによって検証する手法である[1]．モデル検査を行うことで状態遷移系がいずれ必ず目標とする状態へ遷移できるか（活性）や望まぬ状態へ遷移することはないか（安全性）等を調べることができる．モデル検査の代表的なツールとして SPIN や NuSMV 等がある. 本論文では, 特にソフトウェア開発のために開発された SPIN を用いる.

SPIN は 1980 年代にベル研究所で開発されたモデルチェッカであり, 1991 年に一般公開された後は現在に至るまで多くの研究者達によって活発に研究がなされてきた．SPIN では状態遷移系を記述言語 Promela, 仕様となる論理式を線形時相論理（LTL）で記述する. SPIN は検証の結果, 与えられた論理式の真偽を出力し, もし偽であった場合は反例となる実行系列を一つ出力する.

2.3 ユースケース記述と状態遷移モデルとの関係

ユースケース記述をモデル検査を用いて分析するためには, 状態遷移モデルを生成する必要がある．状態遷移モデルを生成するには状態と状態遷移の導出法が必要である．提案手法では, ユースケース記述におけるシステムとアクタとのやり取りの中で行われる動作に注目して状態を抽出する．例えば, 記述中のステップ「利用者はパスワードを入力する」を考える．提案手法では, この記述をパスワードを入力する動作が行われたことによる, “パスワードが入力されている状態” への変化と捉える．このように, ユースケース記述中である動作が行われたか否かを状態変数とみなすことでシステムの状態を抽出する．さらに, ユースケース記述内には記述が動作順序に並んでいるため, 動作の実行順序から状態遷移を抽出できる．また, 異なるユースケース記述間の動作順序は, 各ユースケース記述の事前・事後条件により抽出する.

3. 状態遷移モデル生成

3.1 概　　要

提案手法のプロセスの概要を**図 2** に示す．手法は大きく，出力の状態遷移モデルにおける状態の抽出と状態遷移の導出の 2 プロセスに分かれる．まず，ユースケース記述に書かれた

© 2010 Information Processing Society of Japan

Vol.2010-SE-167 No.17
2010/3/18

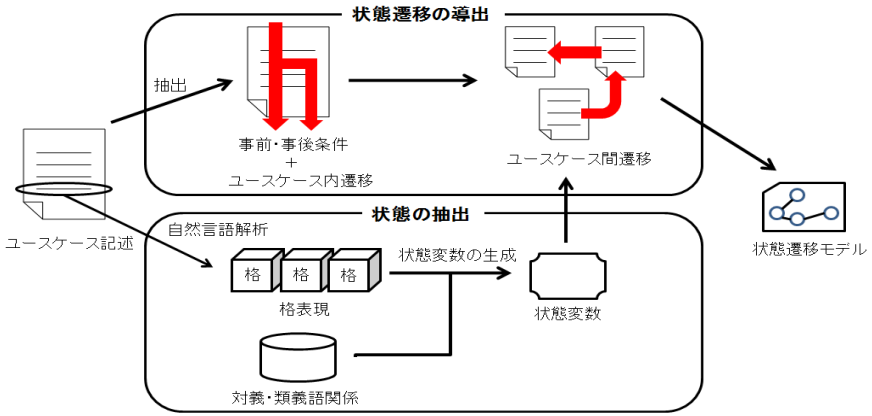

図 2 状態遷移モデル生成プロセス

各記述を，格フレームを用いた形式表現に変換した後，動詞の類義・対義語関係を用いて状態変数を抽出し，システムの状態を特定する．次に，ユースケース記述内の動作系列によりユースケース内の遷移を，状態変数と事前・事後条件によりユースケース間の遷移を導出する．以降では，図 1 に示したユースケース記述を例に，状態遷移モデルの生成法を説明する．

記述のステップには同ユースケース記述内の他のステップへの遷移（以下 Goto ステップと呼ぶ），他のユースケース記述のステップへの遷移（以下 Include ステップと呼ぶ）という特殊なステップが用いられる場合がある．

3.2 状態変数の抽出

ユースケース記述中の各記述は，まず形態素解析によって形態素に分けられた後，格フレームの格スロットに格納される．格フレームとは Fillmore により提案された文章の型[7]で，文章を動詞とその動詞に対して名詞句を意味役割別に区別したもの（深層格）に形式化する．各フレームにおける各深層格を格スロットと呼ぶ．本論文では格助詞を基に格フレームへの格納を行う．

本来, 格助詞に基づく格は表層格であり, 深層格とは必ずしも一致しない．しかし, 自然言語解析の分野では, 名詞や動詞及び文末表現等を用いて深層格を推論する高度な解析手法[11]等が研究されている．本論文ではこの問題は今後の課題とし, ここではひとまず格助詞に基づいて格フレームへの格納を行うことにする．

例えば，ユースケース「ログインする」のステップ 1「利用者はログインボタンを押す」を格

表 1 格スロットの割当て表

	動作主格	対象格	道具格	源泉格	目的格
助詞	は, が	を	で	から	に, へ, まで

スロットに格納することを考える．この文章は，“利用者（名詞）は（格助詞）ログインボタン（名詞）を（格助詞）押す（動詞）” と形態素解析される．格助詞「は」は動作主格,「を」は対象格を表しており，これより前述の文は (動作主格: 利用者, 対象格: ログインボタン, 述語: 押す) という格フレームに変換される．本論文では，**表 1** に示す 5 種類の深層格を用いる．表 1 は各深層格と助詞との関係も含んでおり，これに基づいてユースケース記述中の各文章を格フレームへと変換する．

次に，含まれる動詞の関係に注目し，格フレーム表現から状態変数を抽出する．

まず，ユースケース記述中に含まれる動詞 V の揺れの吸収，対義語の関係の考慮のため，あらかじめ割当て関数 $f : V \rightarrow V^+ \cup V^-$ による動詞の名寄せを行う．ここで，$V^+ = \{v_1, \ldots, v_m\} \subset V$, $V^- = \{\bar{v}_1, \ldots, \bar{v}_m\} \subset V$ はそれぞれ肯定，否定動詞の代表の集合であり，v_i と $\bar{v}_i$ は対義語の関係にある．f は V 全体を V^+ または V^- へ写像し，これによりすべての動詞を，あらかじめ定めておく V^+, V^- へと分類する．例えば，v_i = ログインする，$\bar{v}_i$ = ログアウトする，等の動詞が V^+, V^- に含まれ，f(ログオフする) = ログアウトする，である．

本論文では，対象格と肯定動詞の対を状態変数とみなす．例えば，ユースケース「ログアウトする」の事前条件「ログインしている」は，ログインの状態が真になった場合とみなせ，これは空を対象格，ログインするを述語とする状態変数 $\langle null, \text{ログインする} \rangle$ への $true$ の割当てとする．一方，事後条件「ログアウトしている」はログインの状態が偽になることとみなせ，これは $\langle null, \text{ログインする} \rangle$ への $false$ の割当てとする．状態変数は対象格 o と肯定動詞 v の対 $(o, v) \in Var$ で表し，ユースケース記述から状態変数への割当て $A = Var \times \{true, false\}$ を抽出することにより必要な状態変数を決定する．一般に，ユースケース記述中のあるステップの記述から対象格 o，動詞 v' を発見したとき，$f(v') = v \in V^+$ ならば $\langle \langle o, v \rangle, true \rangle$ を，$f(v') = \bar{v} \in V^-$ ならば $\langle \langle o, v \rangle, false \rangle$ を抽出する．

これらの抽出により，各ユースケース記述における事前・事後条件，各ステップは状態割当てとして表現できる．ユースケース記述 u における事前条件は $Pre(u) \subseteq A$，事後条件は $Pos(u) \subseteq A$，各ステップは $f \in A$ と変換される．

3.3 状態遷移の導出

3.2 節より得られる状態割当て A より，状態遷移モデルにおける各状態 $s \in S$ と状態遷移

© 2010 Information Processing Society of Japan

```
1  stack = ∅, cache = ∅, T = ∅ ; 初期化
2  procedure ExtractTransition(U, s_0)
3    for each u ∈ U s.t. Pre(u) ⊆ assign(s_0) do stack.push(⟨s_0, u⟩)
4    while stack ≠ ∅
5      ⟨s, u⟩ = stack.pop
6      if ⟨s, u⟩ ∉ cache then ; まだ遷移先が生成されていない場合
7        cache ← cache ∪ ⟨s, u⟩
8        TraverseStep(s, uの最初のステップ)
9      end if
10   end while
11 end procedure
12 procedure TraverseStep(s, f)
13   if f が Goto ステップ then
14     TraverseStep(s, f の Goto 先のステップ)
15   else if f が include ステップ then
16     TraverseStep(s, f の include 先のユースケース記述の最初のステップ)
17   else
18     if a ∈ s s.t. var(a) = var(f) then s' = s[a/f_i] else s' = s
19     if f が代替系列の最初のステップ then g = 代替系列への条件 else g = null
20     T ← T ∪ ⟨s, s', g⟩
21     if f が最終ステップ then
22       if f が主系列の最終ステップ then s' ← [s'/Pos(u)]
23       if f が include されたユースケース中のステップでない then
24         for each u' ∈ U s.t. Pre(u') ⊆ assign(s') do stack.push(⟨s', u'⟩)
25       end if
26     else
27       for each f' ∈ succ(f) do TraverseStep(s', f')
28     end if
29   end if
30 end procedure
```

図 3 アルゴリズム

$t \in T \subseteq S \times S \times G$ を決定する．ここで，各状態は状態変数の割当ての集合 $\mathrm{assign}(s) \subseteq A$ と対応付いている[⋆1]．また，$G \subseteq A$ はガード条件の集合であり，遷移が成り立つ状態変数の割当てを示す．

以上のもとで，状態遷移モデルを生成するアルゴリズム ExtractTransition を**図 3** に示す．このアルゴリズムの入力はユースケース記述の集合 U とシステムの初期状態 s_0, 出力は状態遷移の集合 T である．システムの状態は各ユースケース記述の事前・事後条件に用いられている状態変数の割当てを含んでいる．図 1 では，(サイト, 開く), (*null*, ログイン), (商品, 注文) の 3 変数がこれに該当する．ユーザがこれに初期値をそれぞれ *true*, *false*, *false*（以下，(t, f, f) と表記）と与えた初期状態 s_0 から遷移の導出を行った様子が**図 4** に示されている．提案手法では、図 4 のような木構造を構成しながら状態遷移を導出していく．まず，s_0 が事前条件を満たしている各ユースケース記述 u_0「ログインする」と u_2「注文する」を発見する．次に，これらに対して s_0 から各ステップの状態変数割当てを順に解釈し，状態遷移を抽出していく．主系列と代替系列に含まれるステップは，木構造の動作順序に従っており，代替系列の発生点が木の分岐点となる．あるステップ f の次ステップの集合は $\mathrm{succ}(f)$ で得られ，これを順にたどりながら，各ステップから抽出した状態変数割当て f をシステムの状態 s に適用してシステムの新状態 s' を得，s から s' への遷移を導出する．ここで，f の変数がシステムの状態 s に含まれない場合は，システムの状態は変化しない（$s' = s$）．また，f が Goto ステップや Include ステップであった場合，対応した遷移先へのジャンプや他ユースケースへの遷移を追加する．さらに, f が代替系列の最初のステップとなるときは, そのための条件をガード g として与える. 以上を繰り返し，f が主系列の最終ステップに到達した場合，状態に事後条件 $Pos(u)$ をさらに適用し，ユースケースの終了状態を得る．以降，終了状態が事前条件を満たしているユースケース記述が繰り返し検索され，新しい状態が見つからなくなるまで同様の導出を繰り返す．変数 *cache* はある状態であるユースケース記述に到達したことを記憶するキャッシュであり，図 4 の例ではユースケース記述「ログインする」の代替系列の最終ステップ 2.a.1 の終了状態が s_0 と等しくなっているため，これ以降の探索を打ち切っている．**図 5** に提案手法によって生成された、状態遷移モデルを示す. ここで図 5 の各ノードは各ユースケース記述を表すサブマシン状態であり, サブマシン状態内の様子は省略する. 各サブマシン状態に含まれるスタブ状態は, そのユースケース記述の代替系列の最終状態を表している.

4. 支援ツールの実装

本論文では, 提案手法を用いた状態遷移モデル生成支援ツールのプロトタイプを実装した. 本章では, 実装した支援ツールの実装方法と動作について説明する.

4.1 実 装 方 法

本論文では以下のようなもと, 支援ツールを実装した.

プログラミング言語 本支援ツールはオブジェクト指向プログラミング言語 Java により実装を行った.

入力と出力 本支援ツールの入力となるユースケース記述はテキストファイル, 出力はモ

⋆1 状態変数の割当てが一致しても，ステップの実行文脈が異なれば異なる状態となり得ることに注意されたい.

© 2010 Information Processing Society of Japan

Vol.2010-SE-167 No.17
2010/3/18

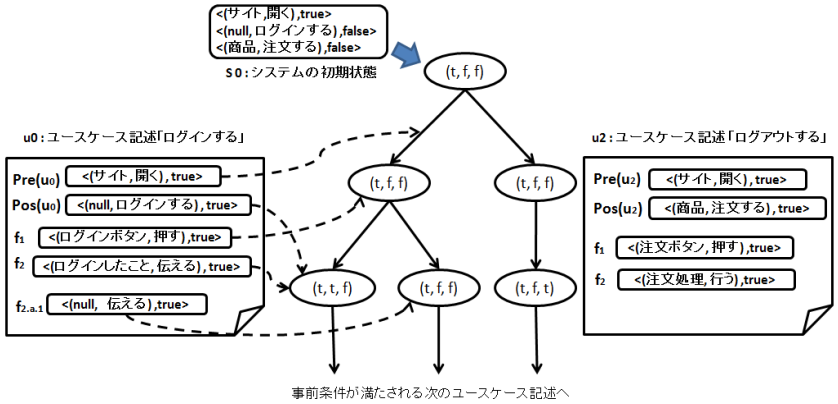

図 4 状態遷移モデルの生成

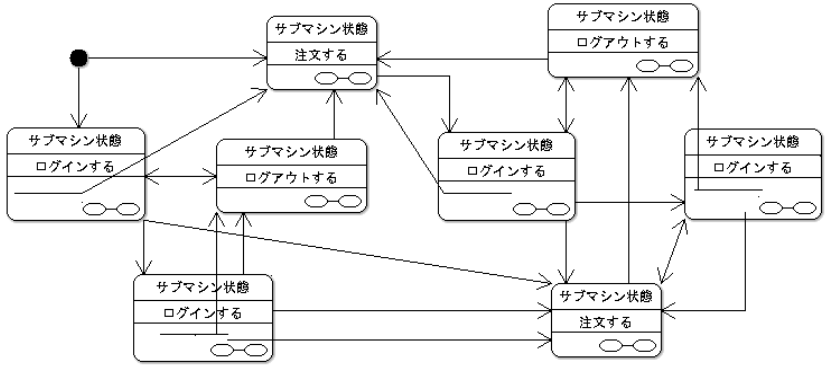

図 5 状態遷移モデルの例

デルチェッカ SPIN の仕様記述言語 Promela で記述された pml ファイルとした. ユーザはユースケース記述を支援ツール上で直接記述するのではなく, テキストエディタ等で別途に記述しテキストファイルを作成する. 1 つのユースケース記述に対し 1 つのテキストファイルを作成し, 作成したテキストファイルは全て支援ツールに入力する. 出力となる pml ファイルは SPIN の入力となる.

形態素解析 本論文ではユースケース記述の構文解析に形態素解析を用いている. 本支援ツールではこれを形態素解析器 Sen[8] を用いて実装した.

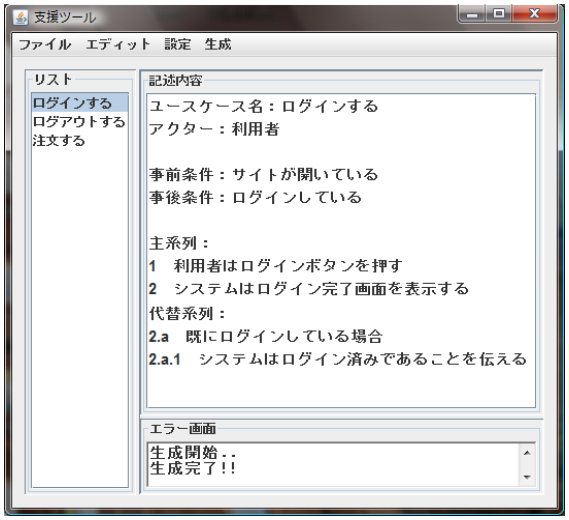

図 6 支援ツールの画面例

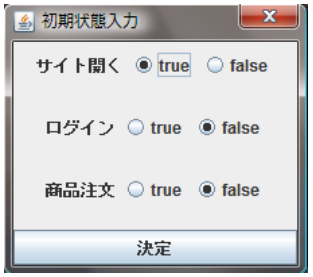

図 7 初期状態の入力画面例

動詞関係の定義 3.2 節で述べた様に状態変数を抽出するにあたり, 動詞関係が定義されたデータが必要になる. 本支援ツールではあらかじめこの動詞関係をデータベースとして蓄えておき, 状態変数を抽出する際に利用した.

4.2 動　　作

図 6 に本支援ツールの起動中の画面を示す. 本支援ツールは 3 つの領域からなる. 左側は入力されたユースケース記述のユースケース名の一覧, 右上側は具体的なユースケース記述の記述内容, 右下側は状態遷移モデルを生成する際に生じたエラー内容を表示する領域である. ユーザは入力したいユースケース記述（テキストファイル）をツールに入力する. ユースケース記述はそのユースケース名がリストに追加され, ユーザがリスト上のユースケース名をクリックすると, クリックされたユースケース記述の記述内容が中央部に表示される. ツールはユースケース記述から状態変数を抽出し、システムの初期状態をユーザに問い合わせる (**図 7**). 以上により, 支援ツールは状態遷移モデルを SPIN の Promela 記述として出力する. ユーザは出力された pml ファイルを SPIN を用いて分析する. 本論文では SPIN の GUI フロントエンドである xspin を用いる. 図 1 のユースケース記述から生成された pml

© 2010 Information Processing Society of Japan

Vol.2010-SE-167 No.17
2010/3/18

ファイルを xspin に読み込んだときの画面を**図 8** に示す．ここで, 例として「ログインした利用者だけが注文をすることができる」を分析する．この性質は, xspin では

```
[]((p && q) -> r)                    (1)
#define   p   usecase_number==2      (2)
#define   q   mainflow==2            (3)
#define   r   login==1               (4)
```

と入力する．式 1 において, “p && q” は式 2, 3 より注文処理がされるステップ (2 番目のユースケース記述「注文する」のステップ 2) を表し, “r” は式 4 よりログインしている状態を表している．また, “[]L” は「L が常に成り立つ」, “L -> R” は「L が成り立つならば, R も成り立つ」ということを表している．その後, **図 9** の画面を開き xspin にユースケース記述の満たすべき性質となる式 1〜4 を入力し、検証を行う．xspin は SPIN を用いて真偽を検証し, 図 9 の下側に結果を出力する．この例では “not valid” と出力されているため, 論理式は偽と分かる．このようにして, 本支援ツールを用いることにより, ユースケース記述を網羅的に検証することができる．

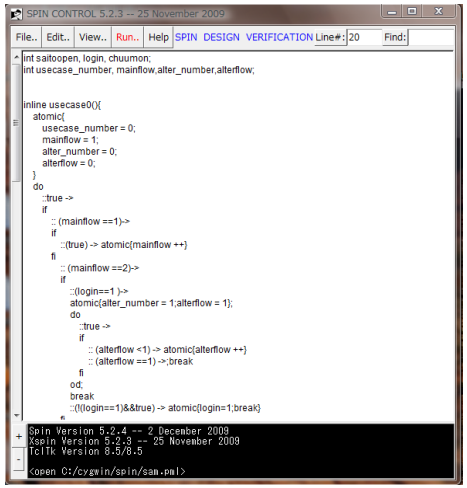

図 8 xspin の読み込み画面

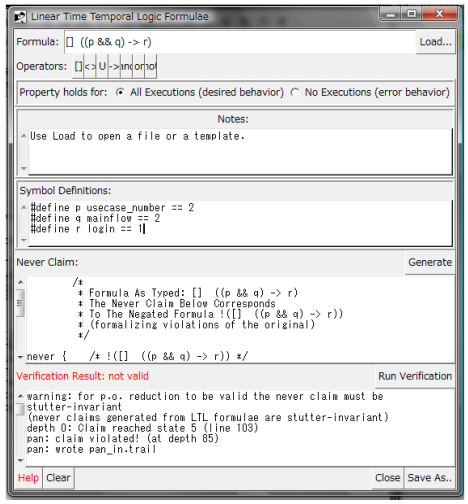

図 9 xspin の LTL 入力画面

表 2 実験結果

被験者	手段	性質 1	性質 2	性質 3	性質 4	性質 5	性質 6	正解数	所要時間
A	手作業	○	×	○	×	○	○	4	17 分
B	支援ツール	○	○	○	○	○	○	6	26 分

5. 適 用 例

5.1 設 定

本支援ツールを用いた適用例を示す．本論文では設定として**図 10** に示されるショッピングサイトシステムに関する 15 のユースケース記述 (合計 143 ステップ) とそれらのユースケース記述が満たすべき以下の性質 6 つを用意した．

(1) ログイン中の利用者は常に入会している状態である．
(2) アカウント設定は入会している利用者だけが行う．
(3) 入会していない利用者は退会処理されない．
(4) 利用者がログインしたとき, カートの中は空の状態である．
(5) 商品の注文はログイン中の利用者だけができる．
(6) 注文のキャンセルはログイン中の利用者だけができる．

6 つの性質には 15 のユースケース記述によって満たされているものと満たされていないものが混在している．これらを 2 名の被験者 A, B に渡し, 与えられた各性質を 15 のユースケース記述が満たしているか否かを分析させ, もし性質が満たされていないと判断したときは, 反例となる実行系列を一つ示すことを求めた．被験者 A には手作業の分析を行わせた．また被験者 B には本支援ツールと xspin を用いて同様の内容を分析させた．以上のもとで被験者 A, B の分析に費やした時間 (所要時間) と正しく分析できた性質数 (正解数) を比較した．さらに, 分析後各被験者に分析の難しかった点をアンケートとしてたずねた．なお被験者は著者の所属する研究室のソフトウェア開発に精通した学生である．

事例結果は**表 2** のようになった．表 2 の “○” はユースケース記述がその性質を満たしているか否かを被験者が正確に判別できた（正解）ことを示しており, 被験者 A は 4 つの性質, B は 6 つの性質全てを判別できた．また “所要時間” は全ての分析にかかった時間を表している．

表 2 から本支援ツールを用いることによって手作業では発見できなかったユースケース記述中の欠陥の発見が可能になることが確認できた．被験者 A が不正解した性質 2, 4 を分析した結果, 大量のユーケース記述を手作業で分析を行った場合, ユースケース記述中の各

© 2010 Information Processing Society of Japan

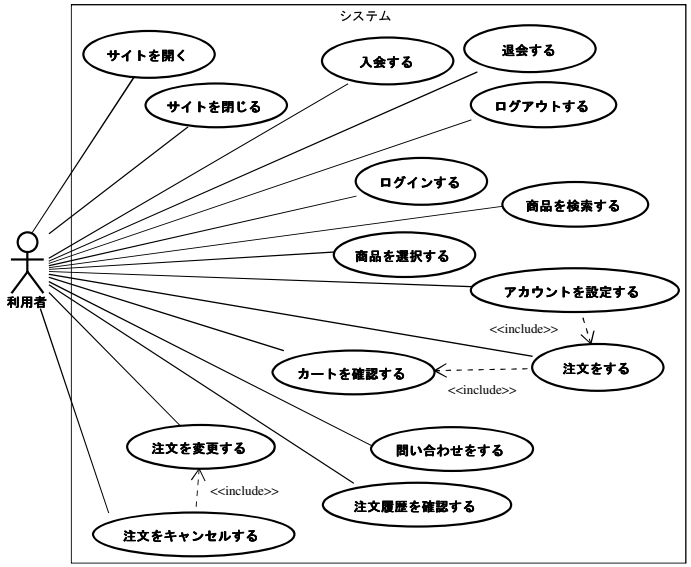

図 10 ユースケース図

ステップにおけるシステムの状態を正確にとらえられなかったことが分かった.

例えば, 性質 4“利用者がログインしたとき, ショッピングカートの中は空である” は, ユースケース記述 “退会する” において利用者がショッピングサイトの会員をやめる際に “ショッピングカート内の商品を空にする” という内容が記述されていないために満たされていなかった. アンケートの結果からは, 被験者 A はショッピングカートに商品が入っているか否かのシステムの状態は “カートを確認する” のようなユースケース名に「カート」を含むユースケース記述を中心にしか調べていなかったことが分かった.

表 2 よりユースケース記述の分析で本支援ツールを用いたときの所要時間は, 手作業で分析を行ったときの所要時間よりも多くなっている. 自然言語の記述を手作業で分析した場合, “多分この性質は満たされているだろう” のように十分な確証を得ずに分析を打ち切ってしまうことが要因の一つとして挙げられる. また, 本支援ツールを用いる際, 自然言語で書かれた性質から LTL へ変換する作業を手作業で行う必要がある. この作業に比較的多くの時間を費やしてしまうことが本支援ツールを用いたときの所要時間が多くなってしまった原因として考えられる. 今後の課題として, この作業にかかる時間を少なくすることが重要である.

6. 関 連 研 究

Sinnig らはユースケース記述のステップを 6 種類に分類し, 各種類ごとに LTS への変換方法を定義した[6]. しかし, この分類では各ステップの記述の意味まで解析することができず, 変換した LTS の各ノードが表すシステムの状態を知ることはできない.

Whittle らはシナリオ内の主要なステップに対し, 事前・事後条件を状態変数を用いて記述することで状態遷移モデルにシステムの状態を付加している[9]. しかしこの手法は設計段階での使用を想定しているため UML のシーケンス図を入力とし, 事前・事後条件は OCL で記述することになっている. 故にこの手法をユースケース記述に適応すると多くのステークホルダはユースケース記述に加え, ユースケース記述内の主要な各ステップに対して事前・事後条件を定義した記述を別途に読まなくてはならなくなる. これは専門外である多くのステークホルダにとってさらなる負担になり, 要求段階での手法として十分でない. 本論文のように格フレームを用いると, ユースケース記述内の各ステップの記述内容を分析することができシステムの状態を追うことができるため, 生成する状態遷移モデルの各ノードにもシステムの状態を付加することができる.

Nebute らは本論文と同じようにユースケース記述の事前・事後条件からユースケース間の状態遷移を導出している[10]. しかし, この研究でもユースケース記述の事前・事後条件をあらかじめ形式化された表現で記述する必要がある. 本論文では格フレームを用いることで自然言語で書かれた記述から状態遷移を導出している. また, この研究ではユースケース記述内の各ステップの状態遷移を考慮していないため, 代替系列が存在する場合は, あらかじめ代替系列の事後条件をユースケース記述に明記しなければならない. 本論文では各ステップの記述を解析できるので, 明記する必要がない.

7. お わ り に

本論文ではユースケース記述を網羅的に分析することが困難であるという問題に対し, 格フレームを用いたユースケース記述からの状態遷移モデル生成法を提案し, 生成された状態遷移モデルにモデルチェッカを用いることでユースケース記述を分析する解決法を示した. また本論文が提案する状態遷移モデル生成法の支援ツールの実装を行った. 支援ツールを用いることで, 15 のユースケース記述が 6 つの性質を満たしているか否かを正確に分析できることを確認し, 提案手法の有用性を示した.

本論文では格フレームへの格納において, 記述中の格助詞に基づいて格フレームへの格納

© 2010 Information Processing Society of Japan

表 3 格フレーム構造の利用例

番号	動作主格	対象格	目的格	動詞
1	利用者	カード	システム	渡す
2	システム	カード	利用者	渡す

を行った．3.2 節で述べたように, 格助詞に基づいて得られた格は表層格であり深層格ではない．よって現手法では全ての記述に関して正確に深層格を判別することはできない．名詞や動詞及び文末表現等を用いて深層格を推論する高度な解析手法[11] を利用することで格フレームへの格納の精度を高めたい．

本論文では状態変数を抽出する際に動詞の類義語・対義語の関係を用いた．しかし, 現手法だけでは全ての記述に対して正確に状態変数を抽出することは不可能である．例えば,「利用者はシステムにカードを渡す」と「システムは利用者にカードを渡す」の 2 文は互いに相反する意味を表現しているが, 本論文の状態変数抽出法では 2 文ともに ⟨(カード, 渡す), true⟩ 等の同じ変数割当てを抽出してしまう．この問題は格フレームの構造を利用することで解決することができる．上記の 2 文を格フレームに格納した結果は**表 3** のようになる．表 3 では, ”渡す”のような動詞では 1 と 2 の記述間で動作主格と目的格が反転している．このような関係を検出することで相反する意味を持った記述を導出ことができると考える．本論文では今後このような導出を用いることで状態変数抽出の精度を高めたい．

本論文ではモデルチェッカ SPIN を用いてユースケース記述の分析を行った．SPIN は論理式を LTL で記述するため, 本支援ツールを用いてユースケース記述を分析するためには自然言語で記述された性質を LTL に変換する必要がある．この際に本支援ツールが作成した pml ファイル内のソースコードの内容の一部を理解する必要があり, これはユーザにとって不便である．今後, 自然言語で書かれた性質から LTL を自動生成する機能を加えることで, ユースケース記述の分析の効率の向上を図りたい．

参 考 文 献

1) Clarke, E.M., Grumberg, O. and Peled, D.A.: *Model Checking*, The MIT Press (2000).
2) Holzmann, G.J.: *The SPIN Model Checker*, Addison-Wesley Professional (2003).
3) Cimatti, A., Clarke, E., Giunchiglia, F. and Roveri, M.: *NuSMV: A New Symbolic Model Verifier*, Lecture Notes in Computer Science, Vol.1633, pp.495–499, Springer (1999).
4) Schneider, G. and Winters, J.P.: *Applying Use Cases: A Practical Guide*, Addison-Wesley Object Technology Series, Addison-Wesley Professional (2001).
5) Larman, C.: 実践 UML-オブジェクト指向分析設計と反復型開発入門，ピアソン・エデュケーション，第 3 版 edition (2007).
6) Sinning, D., Chalin, P. and Khendek, F.: LTS Semantics for Use Case Models, Proceedings of the 2009 ACM Symposium on Applied Computing, pp.365–370 (2009).
7) Fillmore, C.J.: Lexical Entries for Verbs, *Foundations of Language*, pp.373–393 (1968).
8) Sen: https://sen.dev.java.net/.
9) Whittle, J. and Schumann, J.: Generating Statechart Designs From Scenarios, Proceedings of the 22nd International Conference on Software Engineering, pp. 314–323 (2000).
10) Nebute, C., Fleurey, F. and Traon, Y.L.: Automatic Test Generation: A Use Case Driven Approach, *IEEE Transactions on Software Engineering*, Vol.32, No.3, pp. 140–155 (2006).
11) 田辺利文，吉村賢治，首藤公昭：格助詞「に」の深層格推定，福岡大学工学集報，Vol.83, pp.31–34 (2009).

© 2010 Information Processing Society of Japan