情報処理学会論文誌　Vol. 50　No. 12　3052–3073 (Dec. 2009)

ユースケース間の関係を考慮した網羅的な受け入れテストの支援

雁行進夢[†1]　久保淳人[†2]　鈴木三紀夫[†3]
鷲崎弘宜[†1,†4]　深澤良彰[†1]

ユースケース（Use cases）に基づいて進められるソフトウェアの開発（たとえばユースケース駆動のオブジェクト指向開発［Rosenberg（著），三河ら（訳）］）において，ユースケースから受け入れテストのテストシナリオを作成することができる．しかし，ユースケースの実行フローの識別を人手で行う手法では，特にユースケース間の関係が複雑な場合に，実行フローの識別に漏れが出る可能性がある．また，ユースケースを用いた受け入れテストに関して，広く受け入れられた網羅性の定義がなかったため，テスト終了の判定に対して属人性が高くなる．本稿では，まずユースケースを用いた受け入れテストの網羅性を定義し，次に，ユースケースの実行フローを自動で識別したうえで，指定の網羅性を満たすテストシナリオ群とテストプログラムの雛形を自動生成する手法を提案する．提案手法により，ユースケースの実行フローの識別の漏れがなくなり，受け入れテストの終了判定について属人性を排除できる．また，テスト環境の雛形の自動生成により受け入れテストの信頼性を損なう可能性を防止できる．

A Technique for Supporting Comprehensive Acceptance-test Based on the Inter-use Case Reletions

Susumu Kariyuki,[†1] Atsuto Kubo,[†2] Mikio Suzuki,[†3]
Hironori Washizaki[†1,†4] and Yoshiaki Fukazawa[†1]

In software development process using use cases, such as use case driven object-oriented development, test scenarios of the acceptance test can be built from the use case. But manual listing of execution flows of complex use cases sometimes induces incomplete coverage of possible exection flows, especially if the relations between use cases are complex. Moreover, lack of widely-accepted coverage metric of acceptance test results ambiguous judgement of acceptance test's completion. We propose metric of acceptance test's coverage using use cases and automated generations precedure of test scenarios and skeleton code in specified condition of coverage. Our metric can reduce the leakage of detection of execution flows of use cases and unit standard of judgement of acceptance test's completion. And we expect improvement of the efficiency of the acceptance test because of automated generation procedure test scenarios and skeleton code.

1. はじめに

受け入れテストとは，開発されたシステムが顧客の要求を正しく実現しているかを，顧客が主体となって確認する作業である[1)]．機能要求を表す手法の 1 つとして，ユースケース[2)]がある．ユースケースとは，利用者や外部システム（これらはアクタと呼ばれる）から見たシステムの外部機能を表現したものである．ユースケースによって機能要求が識別されている場合は，ユースケースの実行フローを人手で識別し，識別された実行フローをもとにテストシナリオを作成する．ユースケースの数が数十，数百など，規模が大きい場合などには，人手での識別が困難になる可能性があり，実行フローの識別に漏れがでるおそれがある．特にユースケース間の関係が複雑な場合はその可能性は高まり，受け入れテストの信頼性を損なうおそれがある．

一方，テスト終了判定の指針としてテスト網羅性が用いられる．テスト網羅性とは，テストすべき項目数に対する実際にテストされた項目数の割合である[3)]．テスト網羅性の例として，ホワイトボックス・テストにおける命令網羅や分岐網羅などがある．しかし，ユースケースから識別されたテストシナリオを対象とする受け入れテストに関して，広く受け入れられた網羅性の定義がなく，受け入れテストの終了判定の属人性が高い．結果として，受け入れテストの網羅的な実施についてブレが生じ信頼性を損なうおそれがある．

本稿では，ユースケース記述をもとに識別したテストシナリオを対象として，受け入れテストの網羅性を定義する．この定義のもとで，ユースケース記述とドメインモデルおよびテスト網羅性を入力して，機能テストを対象範囲とする，特定のテスト・フレームワーク形式の受け入れテスト環境を自動生成する手法を提案する．提案手法により，ユースケースから

†1 早稲田大学
Waseda University
†2 国立情報学研究所
National Institute of Informatics
†3 TIS 株式会社
TIS, Inc.
†4 国立情報学研究所 GRACE センター
National Institute of Informatics GRACE Center

テストシナリオを機械的に識別できるため，受け入れテストの対象となるテストシナリオの漏れがなくなり，受け入れテストの終了判定に対して属人性を排除することが可能となる．また，受け入れテスト環境の雛形の自動生成により受け入れテストの効率の向上が見込める．

2. ユースケースを用いた受け入れテストとその問題点

2.1 ユースケースとユースケース間の関係とドメインモデル

ユースケースとはアクタから見たシステムの外部機能を表現したものである．ユースケースはシステムの内部構成に触れず，アクタの視点に基づくシステムへの要求を整理することに用いられる．ユースケース記述とは，ユースケースの詳細をテキストで記述したものである[2)]．ユースケース記述のフローに記載されている 1 アクションをユースケース・ステップと呼ぶ．以下では，ユースケース・ステップを単にステップと呼ぶ．また，本稿ではフローの実行されうる組合せによって導かれるステップのシーケンスを，実行フローと定義する．Unified Modeling Language（UML）2.0[4)] によるホテル予約システムのユースケース図の例を**図 1** に，「施設を予約する」ユースケース記述を**図 2** に，「部屋を予約する」ユースケース記述を**図 3** に，「ログインする」ユースケース記述を**図 4** に，「盗聴する」ユース

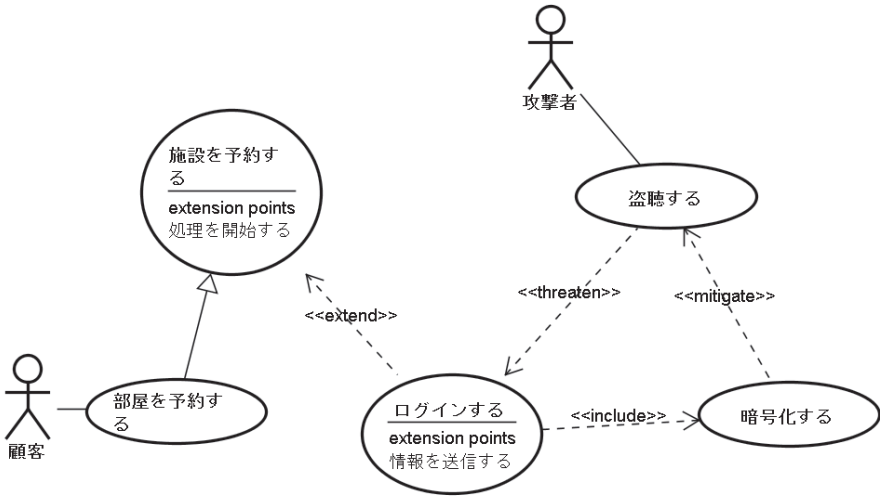


図 1　ユースケース図の例
Fig. 1　Example of use case diagram.

ユースケース：施設を予約する
基本フロー名：施設を予約する
1. システムは利用可能な設備を表示する
2. 顧客は設備を選択する
3. システムは選択された設備の利用にかかる料金の合計を表示する
4. システムは利用可能な設備の数をデータベース上で減らす
5. システムは選択された設備の新たな予約を作成する
6. システムは予約確認番号を表示する
7. システムはユースケースを終了する
代替フロー：
代替フロー名：重複する申請
「重複する申請」フローは「利用可能な設備の数をデータベース上で減らす」ステップの後に分岐する
A1. 1. システムは顧客がこの新たな予約は重複であると指示した場合ユースケースが終了する
移行条件：
同じ予約 (同じ顧客，メールアドレス，開始・終了日) が存在する場合
拡張点：「処理を開始する」
「施設を予約する」フローにおける「利用可能な設備を表示する」ステップ

図 2　「施設を予約する」ユースケース記述
Fig. 2　Use case description of reserving facility.

ユースケース：部屋を予約する
ユースケース：「施設を予約する」を特化する
基本フロー名：施設を予約する
1. システムは利用可能な設備を表示する
2. 顧客は部屋を予約することを選択する
3. 顧客は部屋のタイプを選択する
4. 顧客は部屋の料金を確認する
5. システムは選択された設備の利用にかかる料金の合計を表示する
6. システムは利用可能な設備の数をデータベース上で減らす
7. システムは選択された設備の新たな予約を作成する
8. システムは予約確認番号を表示する
9. システムはユースケースを終了する
代替フロー：
代替フロー名：重複する施設
「重複する施設」フローは「利用可能な設備の数をデータベース上で減らす」ステップの後に分岐する
A1. 1 システムは顧客がこの新たな予約は重複であると指示した場合ユースケースが終了する
移行条件：
同じ予約 (同じ顧客，メールアドレス，開始・終了日) が存在する場合

図 3　「部屋を予約する」ユースケース記述
Fig. 3　Use case description of reserving room.

ユースケース：ログインする
基本フロー名：ログインする
1.　システムは顧客に ID とパスワードの入力を促す
2.　顧客は ID とパスワード入力する
3.　システムは入力された情報を暗号化して送信する
4.　システムは認証する
5.　システムは基底ユースケースを実行する
代替フロー：代替フロー名：認証に失敗する
「認証に失敗する」は「認証する」の後に分岐する
A1. 1 システムは基底ユースケースを終了する
移行条件：
登録されていない ID かパスワードと ID が対応できない時
包含：
「ログインする」フローの「入力された情報を暗号化して送信する」ステップは，
「暗号化する」ユースケースを実行する
拡張：「処理を開始する」の前
拡張点：「情報を送信する」
「ログインする」フローにおいて「入力された情報を暗号化して送信する」ステップ

図 4　「ログインする」ユースケース記述
Fig. 4　Use case description of logging in.

ユースケース：盗聴する
基本フロー：盗聴する
1. 攻撃者は送信された情報を盗聴しようとする
拡張：「情報を送信する」の後

図 5　「盗聴する」ユースケース記述
Fig. 5　Use case description of tapping.

ユースケース：暗号化する
基本フロー名：暗号化する
1. システムは送信される情報について暗号化する

図 6　「暗号化する」ユースケース記述
Fig. 6　Use case description of coding.

ケース記述を**図 5** に，「暗号化する」ユースケース記述を**図 6** に示す．

UML2.0 で定義されているユースケース間の関係（包含，汎化，拡張）を用いて既存のユースケースを変更せずに動作を追加することで，ユースケース記述を再利用することが可能となる．

包含（include）　包含とは，あるユースケースの中に別のユースケースの処理が含まれることを表す関係である．図 1 では，「ログインする」ユースケースは「暗号化する」ユースケースを含む．図 4 内において，「包含：「ログインする」フローの「入力された情報を暗号化して送信する」ステップは，「暗号化する」ユースケースを実行する」の部分が，「ログインする」ユースケースは「暗号化する」ユースケースを含むことを表している．

汎化（generalization）　汎化とは，ユースケース間の is-a 関係である．図 1 では，「部屋を予約する」ユースケースは「施設を予約する」ユースケースの一種であることを意味する．図 3 では，「ユースケース：「施設を予約する」を特化する」の部分が，汎化関係があることを表している．

拡張（extend）　拡張とは，既存のユースケースに別のユースケースの内容を追加する関係である．拡張されるユースケースは，拡張点（extention points）と呼ばれる追加的な振舞いを挿入可能な特定の実行ポイントを持つ．拡張するユースケースは，拡張点を指定することで追加する場所を指定する．図 1 では，「ログインする」ユースケースは「施設を予約する」ユースケースを拡張する．拡張点として「処理を開始する」を指定する．

また，拡張関係の応用例として，ミスユースケースやセキュリティユースケースがある[5]．ミスユースケースとは，設計中のシステムに対する敵対的なアクタの視点から記述するユースケースである．セキュリティユースケースとは，ミスユースケースの影響を緩和するユースケースである．文献 5) では，あるミスユースケースがユースケースに「脅威」を与えることを，threaten という関連として定義し，あるユースケースがミスユースケースの脅威を「緩和」していることを，mitigate という関連として定義する．また，脅威は拡張を特化した関連と定義できると述べている．図 1 では，「盗聴する」ユースケースが「ログインする」ユースケースに脅威を与えることを，`<<threaten>>` のステレオタイプ付きの矢印で表している．同様に図 1 において，「暗号化する」ユースケースが「盗聴する」ユースケースの脅威を緩和していることを `<<mitigate>>` のステレオタイプ付きの矢印で表している．

ドメインモデルとは，開発者が対象とする問題領域の正確な理解を目的として，対象業務の世界を構成する本質的，かつ，注目する概念と概念間の関係を表すモデルである[6]．図 1 に表されているホテル予約システムのドメインモデルの一例を**図 7** に示す．図 7 において「部屋」や「予約」など，ホテル予約システムに関して主要な概念が表現されている．

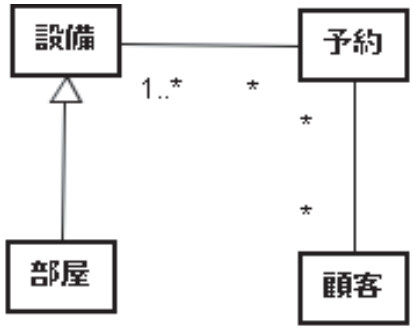


図 7　図 1 に対応するドメインモデルの一例
Fig. 7　Exapmple of domain model.

2.2　受け入れテスト

ユースケースは，顧客の要求を表す．そのため，ユースケースから作成されたテストシナリオを用いてテストすることで，顧客の要求が正しく実装されているかテストすることができる．顧客の要求に関するテストの 1 つに受け入れテストがある．受け入れテストとは，開発しているソフトウェアが，顧客の望む内容を本当に実装しているかどうか，顧客が主体となってテストすることである．本稿では，受け入れテストの 1 つである，顧客の要求する機能が実装されているかどうかについてテストする機能テストに着目する．

2.3　Framework for Integrated Test

Framework for Integrated Test（FIT）[7] は，Ward Cunningham が開発した受け入れテストに使用可能なテストフレームワークであり，以下の特徴がある．

- 自動テストが可能なこと
- HTML の表形式でテストシナリオを作成するため，プログラムの書き方が分からない顧客でもテストシナリオを作成できること
- Java や Ruby など，多言語に対応していること

FIT は以下の要素から構成される．

- ランナ．テスト実行プログラム
- フィクスチャ．テスト記述ファイルとテスト対象プログラムの対応付けを定義したもの
- テスト記述ファイル．テストシナリオを記述したファイル
- テスト結果ファイル．テスト結果

FIT によるテストの実行の流れを**図 8** に示す．

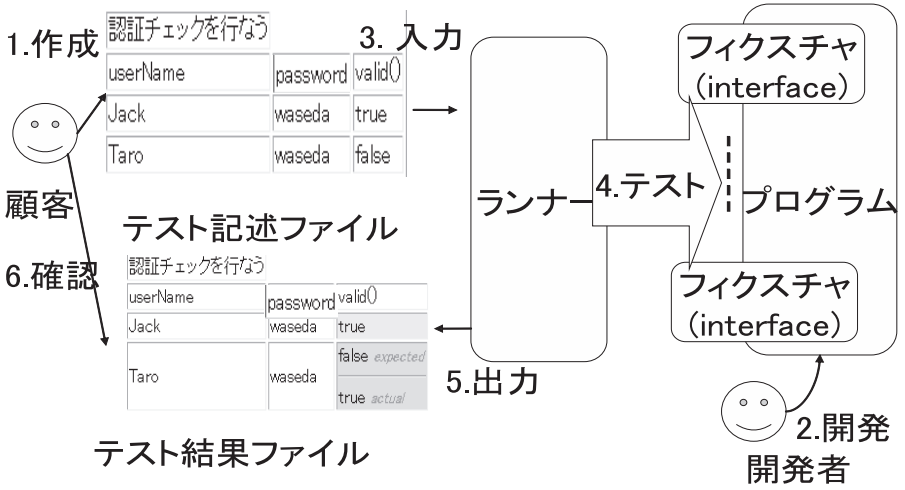


図 8　FIT の実行の流れ
Fig. 8　Outline testing using FIT.

（1）　顧客は，HTML 形式でテスト記述ファイルを作成する．具体的には，プログラムに望む機能をテストする内容を，テスト記述ファイル内に HTML の表として記述する．図 8 内の「認証チェックを行う」を例に用いて説明する．顧客は，1 行目の「認証チェックを行う」でフィクスチャを指定する．また，顧客は，2 行目にテスト項目として，userName，password，valid() の 3 つを扱うことを記述する．3 行目以降でテストシナリオを表し，3 行目では userName に Jack，password に waseda を代入し，valid() メソッドを実行した結果が true になるかどうかテストすることを表す．

（2）　開発者は，テスト記述ファイル内に記述された表が示す内容を満たすように，プロダクトプログラムを開発する．また，開発者は，テスト記述ファイル内に記述された表の内容とプロダクトプログラムを結びつけるためのフィクスチャを開発する．図 8 内の「認証チェックを行う」に対応するフィクスチャの場合，userName や password という名前の変数を持ち，userName と password を用いて valid() メソッドでテストしたい機能を実行するコードを記述しておく．

（3）　顧客はテスト記述ファイルを FIT に入力して受け入れテストを実行し，ランナが出力するテスト結果ファイルを閲覧することにより，受け入れテストの結果を確認する．

2.4　ユースケースを用いた受け入れテストの問題点

ユースケースから実行フローを識別してテストシナリオを作成する既存手法として，文献 8)，9) などがあげられる．文献 8)，9) では，ユースケースの実行フローを人手で識別し，識別された実行フローをもとにテストシナリオを作成している．そのため，以下の問題がある．

問題 1　ユースケースから識別されたテストシナリオを用いた受け入れテストを行う際，広く受け入れられた網羅性の定義がない．そのため，たとえば，すべてのユースケースを網羅するテストシナリオの作成を指示された場合に，ある人はすべてのステップが 1 回は含まれるようなテストシナリオ群をあげるが，別の人はすべての分岐が 1 回は含まれるようなテストシナリオ群をあげてしまう可能性がある．つまり，テスト終了の判定に属人性が生じる可能性があり，受け入れテストの信頼性を損なうおそれがある．

問題 2　ユースケース間の関係が複雑な場合，実行フローの識別に漏れが発生するおそれがある．大規模な開発では，ユースケースの数が数十，数百になることもありうるため，それにともなってユースケース間の関係も複雑になる可能性がある．また，通常のユースケースだけでなく，セキュリティに関する要求が高まっていることから，ミスユースケースやセキュリティユースケースなども考慮することが必要な場合もある．その場合は，通常のユースケースのみを考える場合より実行フローが複雑になるため，実行フローが漏れてしまう可能性が高くなる．

3.　ユースケースの実行フローに基づく網羅性の定義

2.4 節で述べた問題に対し，次の解決を提案する．

問題 1 の解決　ユースケース記述から識別したテストシナリオを対象とする受け入れテストに関して，網羅性の定義．同定義に基づくことで，テスト終了の判定に属人性を排除することができる．

問題 2 の解決　特定のテスト・フレームワーク形式の受け入れテスト環境の雛形生成システムによる，ユースケースの実行フローの機械的な識別．本稿では，多言語対応や普及度を考慮して，テスト・フレームワークを FIT 形式にした．ユースケースの実行フローを機械的に識別することで，テストシナリオの識別の漏れや誤りを排除することができる．

本章では，問題 1 の解決を述べ，次章において問題 2 の解決を述べる．

3.1　ユースケースの全実行フローの識別

ユースケースの実行フローのグラフを作成し，そのグラフに基づいて網羅性を定義する．以下，ユースケースの実行フローのグラフの作成手順を述べる．

すべてのステップの集合を S，ステップ間の遷移の集合を E とすると，ユースケース uc は以下のとおりになる．

$S = \{s_1, s_2, \ldots\}$

$E \subseteq S \times S$

$uc = (S, E)$

n を定義されているステップの数と定義する．$|X|$ を集合 X の要素数，X_i をステップ s_i を移行元とするエッジの集合とすると，X_i の要素数が 2 以上あるとき，X_i が s_i からの分岐の集合となるので，分岐の集合 B は以下のとおりになる．

$$X_i = \left\{ (s_i, s_j) \middle| s_j \in (S - \{s_i\}) \wedge (s_i, s_j) \in E \right\}$$

$$B = \left\{ \bigcup_{i=1}^{n} X_i \;\middle|\; |X_i| \geq 2 \right\}$$

以下のアルゴリズムにより生成されるもの，すなわち，すべての実行フローの集合を F とする．

関数名　generate_ allFlows

概要　あるユースケースのすべての実行フローの集合を得る

入力　ユースケース

出力　入力したユースケースのすべての実行フローの集合

```
generate_allFlows(uc : ユースケース)  {
    s1 := uc の最初に実行されるステップ;
    F := φ;     //全実行フローを表す変数
    Oneflow := φ;     //実行フローを表す変数
    traverse(s1, Oneflow, F);     //全実行フローを再帰関数で作成
    return F;
}
```

関数名　traverse

概要　あるステップを入力して，入力されたステップから実行されうるフローの集合を求める

入力　あるステップとあるユースケースの実行フローを表したもの，および，あるユースケースの全実行フローの集合

出力　なし

```
traverse(s : ステップ, Oneflow : 実行フロー, All : 実行フローの集合)　{
  nextSteps := {s_i | (s, s_i) ∈ E }; //あるステップの次に実行されるステップの集合
  if (nextSteps ≠ φ){　//次に実行されるステップがない場合
    All := All ⋃ {Oneflow};　//実行フローを実行フローの集合に追加
  } else if {　//次に実行されるステップがある場合
    for each (s_j ∈ nextSteps) {　//次に実行されるステップのそれぞれについて
      Oneflow := Oneflow ⋃ {(s, s_j)}; //次に実行されるステップへの遷移を追加
      traverse(s_j, Oneflow, All); //再帰により実行フローを作成
    }
  }
}
```

図 2 のみを考慮した場合において，図 2 を関数 generate_allFlows に入力すると，S，E，F はそれぞれ以下のようになる．ただし，ステップを内容ではなく番号で記述する．

$S = (s_1,\ s_2,\ s_3,\ s_4,\ s_5,\ s_6,\ s_7,\ s_{A1.1})$

$E = \{(s_1,\ s_2),\ (s_2,\ s_3),\ (s_3,\ s_4),\ (s_4,\ s_5),\ (s_5,\ s_6),\ (s_6,\ s_7),\ (s_4,\ s_{A1.1})\}$

$F = \{\{(s_1,\ s_2),\ (s_2,\ s_3),\ (s_3,\ s_4),\ (s_4,\ s_5),\ (s_5,\ s_6),\ (s_6,\ s_7)\},\ \{(s_1,\ s_2),\ (s_2,\ s_3),\ (s_3,\ s_4),\ (s_4,\ s_{A1.1})\}\}$[⋆1]

また，ユースケース間の関係による実行フローの変化の識別手順を**図 9** に示す．図 1 のユースケース群を例として用いる．入力されたユースケース記述から，ユースケースの実行フローのグラフを作成する．

(1)　もし，汎化しているユースケースが存在すれば，汎化しているユースケースの実行フローに，特化しているユースケースの実行フローをユースケース・ステップ単位で上書きする．この例における，汎化による変化が終わった後のグラフを**図 10** に示す．

⋆1 図 38 のみをユースケース間の関係を考慮せずに入力した場合の S，E，F は以下のとおり．

$S = (s_1,\ s_2,\ s_3,\ s_4,\ s_{A1.1},\ s_{A1.2},\ s_{A1.3})$

$E = \{(s_1,\ s_2),\ (s_2,\ s_3),\ (s_3,\ s_4),\ (s_2,\ s_{A1.1}),\ (s_{A1.1},\ s_{A1.2}),\ (s_{A1.2},\ s_{A1.3}),\ (s_{A1.3},\ s_3)\}$

$F = \{\{(s_1,\ s_2),\ (s_2,\ s_3),\ (s_3,\ s_4)\},\ \{(s_1,\ s_2),\ (s_2,\ s_{A1.1}),\ (s_{A1.1},\ s_{A1.2}),\ (s_{A1.2},\ s_{A1.3}),\ (s_{A1.3},\ s_3),\ (s_3,\ s_4)\}\}$

```
作成対象の UC のみに作成されているフローのグラフを作成;
if(汎化関係がある) {
  汎化している UC の実行フローと作成対象
の UC の実行フローの差分をステップ単位で上書き;
}
if(包含関係がある) {
  包含対象を参照しているステップを
    包含対象の UC の実行フローで上書き;
}
if(拡張関係がある) {
  拡張する側の UC の実行フローを拡張点に追加;
}
```

図 9　ユースケース間の関係による実行フローの変化の識別手順
Fig. 9　Identification procedure of execution flow.

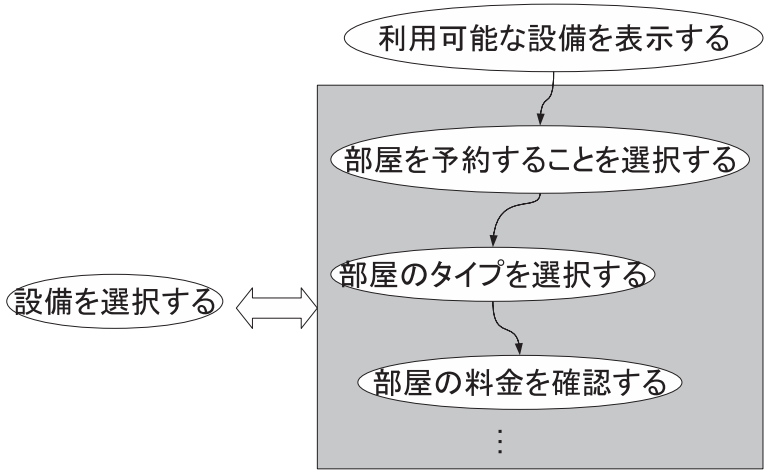


図 10　汎化によるフローの変化
Fig. 10　Flow change by generalization.

灰色の枠は汎化により変化した部分である．結果，「施設を予約する」ユースケースで定義されていた「設備を選択する」ステップが，「部屋を予約する」ユースケースに定義されている「部屋を予約することを選択する」「部屋のタイプを選択する」「部屋の料金を確認する」ステップに置き換わる．

(2)　もし，包含するユースケースが存在すれば，包含対象を参照するステップを，包含さ

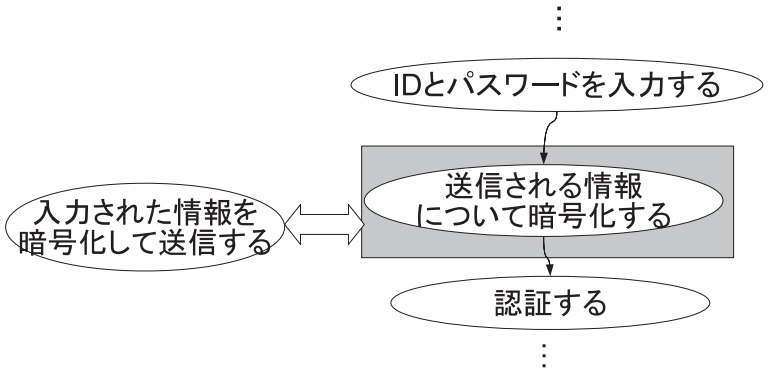


図 11　包含によるフローの変化
Fig. 11　Flow change by include.

れるユースケースの実行フローで上書きする．包含により変化したグラフ例を**図 11** に示す．包含により変化した部分は灰色の枠の部分である．これにより，「ログインする」ユースケースの「入力された情報を暗号化して送信する」ステップが，「ログインする」ユースケースが参照していた「暗号化する」ユースケースに定義されている「暗号化する」フローに置き換わる．これは，「送信される情報について暗号化する」ステップを実行することで，万が一，情報を盗聴されても悪用される可能性を緩和していることを意味する．

(3)　もし，拡張するユースケースが存在すれば，拡張するユースケースの実行フローを拡張点に指定されているステップに追加する．拡張による変化が終わった後のグラフを**図 12** に示す．拡張により変化した部分は灰色の枠の部分である．これにより「盗聴する」ユースケースの拡張点に指定されていた,「ログインする」ユースケースの基本フローの「入力された情報を暗号化して送信する」ステップの後に「盗聴する」フローが追加される．これにより，顧客がログインする際に送信される情報を，攻撃者が盗聴を試みるという脅威を表している．同様に「ログインする」ユースケースの拡張点に指定されていた「施設を予約する」ユースケースの基本フローの「利用可能な設備を表示する」ステップの前に「ログインする」フローが追加される．

なお，脅威の関係は UML2.0 で定義されている拡張の特化版として解釈できるので[5]，実行フローの特定時には脅威の関係は拡張の関係として扱う．また，緩和は脅威の影響が緩和

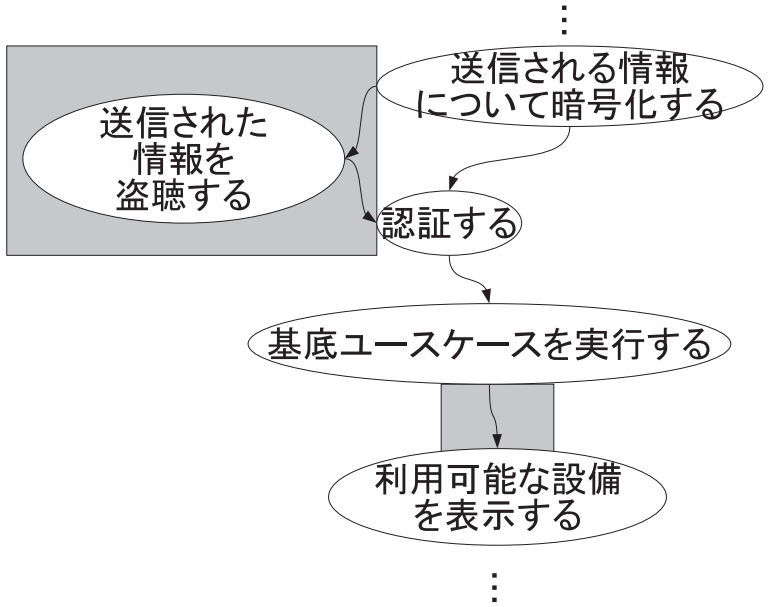


図 12　拡張によるフローの変化
Fig. 12　Flow change by extend.

された結果を表し，実行フローに影響を与えないので，実行フローの特定時には緩和の関係は存在しないものとして扱う．

3.2　ユースケースを用いた受け入れテストの網羅性

ユースケースから識別されたテストシナリオに基づく受け入れテストの網羅性について，3.1 節により作成された実行フローのグラフを用いて次のとおりに定義する．ユースケース記述中のフローとプログラムフローの類似性に着目し，プログラムフローのテスト網羅性の定義を参考として，ステップ網羅，分岐網羅，全実行フロー網羅の 3 種を提案する．ループはプログラムの制御フローと同様に 1 つの分岐として扱う．ただし，1 回の繰返しのみを扱い，2 回目以降は考慮しない．

定義 1　ステップ網羅 C_0'

ステップ網羅 C_0' はすべてのユースケース・ステップの中で，いくつのステップが実行されたかを百分率で表す．テストケースの集合 T が含むテストケース t により処理されたス

ユースケース：リソースへのアクセスを待たせる
基本フロー：リソースへのアクセスを待たせる
1. リソースロックは，サービスクライアントの
プロセスを一時中断させる
2. サービスクライアントは再開されるのを待つ
3. サービスクライアントのプロセスが再開される
例外フロー
A1. 待ち時間のタイマー切れ
基本フローのステップ 2 において，
待ち時間のタイマーの期限が切れたとき
1. 要求されたアクセスが許可できないことを
サービスクライアントにシグナルで知らせる
2. ユースケースが終了する

図 13「リソースへのアクセスを待たせる」ユースケース記述
Fig. 13　Use case describing awaiting for the access for the resource.

テップの集合を S_t として，ステップ網羅 C'_0 を次のように定義する．

$$C'_0 = \frac{|S_t|}{|S|} \times 100$$

定義 2　分岐網羅 C'_1

分岐網羅 C'_1 はすべてのフローの分岐の中で，いくつの分岐が実行されたかを百分率で表す．テストケースの集合 T が含むテストケース t により処理された分岐の集合を B_t として分岐網羅 C'_1 を次のように定義する．

$$C'_1 = \frac{|B_t|}{|B|} \times 100$$

定義 3　全実行フロー網羅 C'_∞

全実行フロー網羅 C'_∞ はすべてのフローの組合せの中で，いくつのフローの組合せが実行されたかを百分率で表す．テストケースの集合 T が含むテストケース t により処理された実行フローの集合を F_t として，全実行フロー網羅 C'_∞ を次のように定義する．

$$C'_\infty = \frac{|F_t|}{|F|} \times 100$$

実際に，**図 13** のユースケースに対して，C'_1 の算出例を示す（文献 2) より一部変更して抜粋）．実行フローをグラフ化したものを**図 14** に示す．実際にテストされた分岐を $\{c_2\}$ として，分岐網羅 C'_1 は以下となる．

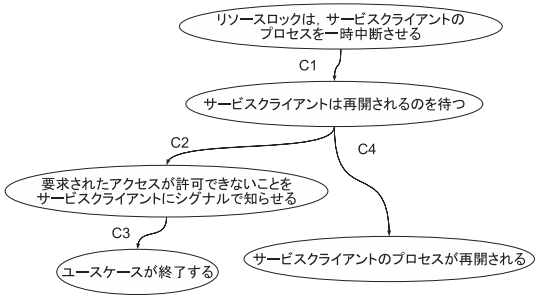


図 14「リソースへのアクセスを待たせる」ユースケースの実行可能フロー
Fig. 14　Execution flows of use case of awaiting for the access for the resource.

$$\begin{aligned} C'_1 &= \frac{|\{c_2\}|}{|\{c_2, c_4\}|} \times 100 \\ &= 1/2 \times 100 \\ &= 50\% \end{aligned}$$

ちなみに，ステップ網羅 C'_0 は 80%，全実行フロー網羅 C'_∞ は 50%となる．

4. テスト環境生成システム

人手による実行フローの識別では，漏れや誤りが混入するおそれがあった．本稿では，特定のテスト・フレームワーク形式の受け入れテスト環境の雛形生成システム（以下，提案システムと呼ぶ）を提案する．提案システムによりユースケースの実行フローを機械的に識別することで，テストシナリオの漏れや誤りを排除することができる．本稿では，多言語対応や普及度を考慮してテスト・フレームワークの FIT 形式にした．

4.1　提案システムの構成および提案システムを用いたテスト方法

ユースケースの実行フローを人手で識別すると，漏れが発生するおそれがある．ユースケース間の関係が複雑な場合はさらにその可能性が高まる．解決策として，ユースケース記述とドメインモデルおよび網羅性を入力として，実行フローの識別を自動で行い特定のテスト・フレームワーク形式の受け入れテスト環境の雛形を生成するシステムを提案する．提案システムの概要を**図 15** に示す．提案システムのスクリーンショットを**図 16** に示す．提案システムを用いた受け入れテストは，(1)「提案システムへの入力フェーズ」，(2)「提案シ

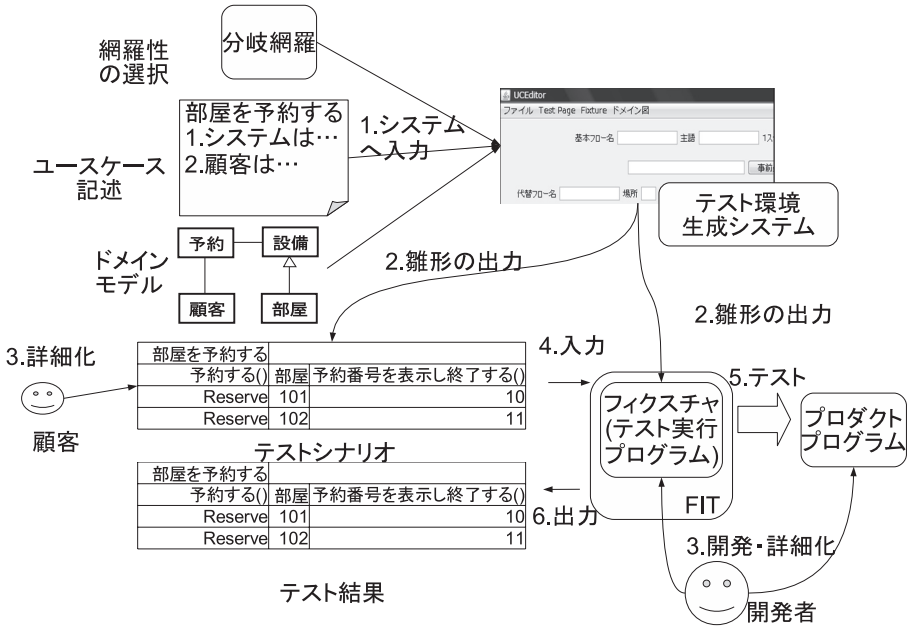


図 15　提案システムの概要
Fig. 15　Outline of proposal system.

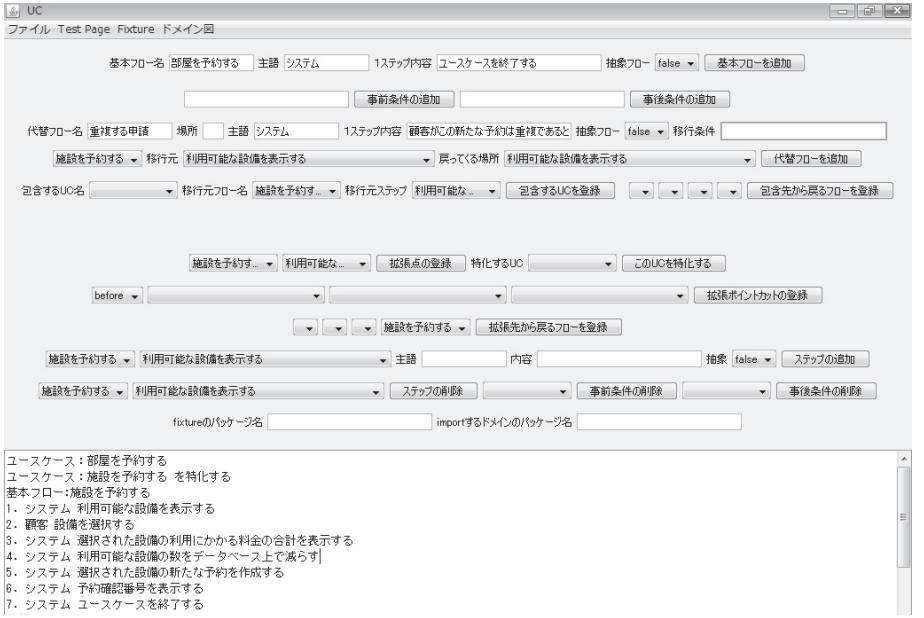


図 16　提案システムの実行画面
Fig. 16　Screen-shot of proposal system.

ステムの出力を詳細化するフェーズ」，(3)「FIT による受け入れテストフェーズ」の 3 段階に分けられる．

(1) 提案システムへの入力フェーズ

提案システムに以下の 3 種を入力する．

- 網羅性の選択
- ユースケース記述
- ドメインモデル

提案システムにドメインモデルを入力することで，テストの入力値となる要素を抽出することができる．たとえば，「部屋を予約する」ユースケースに関して，「顧客」である斎藤さんが 3022 の「部屋」を予約するというテストシナリオを記述する際に，「顧客」や「部屋」といった要素を抽出することができる．提案システムは入力を受け取ると以下の 2 種を出力する．

- テストシナリオの雛形（HTML の表形式）
- フィクスチャ（テストシナリオの内容のテスト実行プログラム）の雛形

(2) 提案システムの出力を詳細化するフェーズ

提案システムにより出力されたテストシナリオにはドメインモデル内のテスト入力項目やステップが記述されている．しかし，テストシナリオには「顧客」や「部屋」などのドメインモデル内のテスト入力項目は記述されているが，「顧客」の項目に対し「斉藤」，「部屋」の項目に対し「3022」などのテスト入力項目の具体的な値は記述されていない．同様に，フィクスチャはステップと同名の操作が定義されているがテストしたい機能の呼び出しなどが記述されていない．以上のテスト項目の詳細をテストシナリオに，テストしたい機能の呼び出しをフィクスチャに追記する必要がある．このフェーズにより以下のファイルをテスト可能な状態に詳細化する．

- テストシナリオの雛形．顧客が修正する．

- フィクスチャの雛形．テスト担当者が修正する．

(3) FIT による受け入れテストフェーズ

人手によるテストシナリオ，フィクスチャの詳細化の後に FIT を用いた受け入れテストを行う．

4.2　実行フローのグラフを用いたテストシナリオの雛形の生成

本節は以下の 2 点を述べる．

- 提案システムへの入力フェーズで得られるテストシナリオの雛形の生成
- 提案システムの出力を詳細化するフェーズで行うテストシナリオの詳細化

以下の 2 点は次節で述べる．

- 提案システムへの入力フェーズで得られるフィクスチャの雛形の生成
- 提案システムの出力を詳細化するフェーズで行うフィクスチャの詳細化

3.1 節により作成された実行フローのグラフからテストシナリオの雛形を生成する．3.2 節で定義された 3 種類の網羅性の選択に基づいて実行フローのグラフから出力されるテストシナリオを選択する．以下，出力されるテストシナリオ選択アルゴリズムを述べる．

関数名　generate_ testScenarios

概要　特定の網羅性に基づくユースケースの実行フローのグラフを得る

入力　網羅性とユースケース

出力　入力された網羅性に基づくユースケースの実行フローの集合

```
generate_testScenarios(criteria : 網羅基準 , uc : ユースケース)  {
  Flows := generate_allFlows(uc); //ユースケースの全実行フローのグラフを作成
  if(criteria = 全実行フロー網羅) {//全実行フロー網羅の判定
    return Flows;
  }
  coverageFlows := φ; //網羅基準に基づく実行フローのグラフ
  for each (f ∈ Flows) {//各実行フローについて
    coverageFlows := coverageFlows ⋃ {f}; //実行フローを coverageFlows に追加;
    Edges := getEdges(coverageFlows) ; //ステップの遷移の集合を得る;
    if (criteria = 分岐網羅) {
      if(E = Edges){//分岐網羅の判定
        return coverageFlows;
      }
    } else if (criteria = ステップ網羅) {
      if(S = getSteps(Edges)){//ステップ網羅の判定
        return coverageFlows;
      }
    }
  }
}
```

関数名　getEdges

概要　実行フローのグラフの集合から，ステップの遷移の集合を抽出する．たとえば，{{(s_1, s_2), (s_2, s_3)}, {(s_1, s_A)}} を入力すると {(s_1, s_2), (s_2, s_3), (s_1, s_A)} を得る．

入力　実行フローのグラフの集合

出力　入力された実行フロー群のグラフ内に出てくるステップの遷移の集合

```
getEdges(Flows : 実行フローのグラフの集合)  {
  Edges := φ; //ステップの遷移の集合を保存する変数
  for each (f ∈ Flows) {//すべてのフローについて
    for each (e ∈ f) {//f 内に出てくるすべてのステップの遷移について
      Edges := Edge ⋃ {e}; //Edges にステップの遷移を追加
    }
  }
  return Edges;
}
```

関数名　getSteps

概要　ステップの遷移の集合から，ステップを抽出する．たとえば，{(s_1, s_2), (s_2, s_3), (s_1, s_A)} を入力すると {s_1, s_2, s_3, s_A} を得る．

入力　ステップの遷移の集合

出力　ステップの遷移の集合内に出てくるステップの集合

```
getSteps(Edges : ステップの遷移の集合)  {
  Steps := φ; //ステップを保存する変数
  for each (e | e ∈ Edges, e = (s_i , s_j)) {//すべてのステップの遷移について
```

表 1　生成された「部屋を予約する」ユースケースに対応するテストシナリオファイル（その 1）（詳細化前）
Table 1　Generated test scenario 1 corresponding to use case of reserving room without detalils.

部屋を予約する				
システムは利用可能な…	部屋	顧客は部屋を予約…	≈	システムはユースケースを…
pass	*null*	pass	≈	pass
pass	*null*	pass	≈	pass

$Steps := Steps \bigcup \{s_i\}$; //*Steps* にステップを追加

$Steps := Steps \bigcup \{s_j\}$; //*Steps* にステップを追加

}

return Steps;

}

たとえば，図 2 をステップ網羅，分岐網羅，全実行フロー網羅を指定して，generate_testScenarios に作成される実行フローはすべて同じであり，以下のとおりである．

$F = \{\{(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_5), (s_5, s_6), (s_6, s_7)\}, \{(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_{A1.1})\}\}$[⋆1]

以上のアルゴリズムからテストシナリオを選択し，以下のように出力する．出力例を**表 1** に示す．

- テスト項目（FIT に対する入出力となる変数名を定義したもの）の行に，ステップを実行順に記述する．たとえば，表 1 では，「システムは利用可能な設備を表示する」「顧客は部屋を予約することを選択する」…「システムはユースケースを終了する」の各ステップをテスト項目の行（ここでは，2 行目にあたる）に記述する．
- その際，ステップの中にドメインモデルに含まれるクラスの名前が記述されていれば，含まれるクラス名のうち，提案システムの利用者が指定したクラス名をテスト項目の列内の該当するステップの前に記述する．たとえば，ステップが「顧客は部屋を予約することを選択する」の場合，ドメインモデルのクラス名の中に「顧客」，「部屋」，「予約」の 3 つがあれば，「顧客」と「部屋」と「予約」の 3 つがテスト項目の候補になる．このうち，提案システムの利用者が「部屋」のみを選択すると，「部屋」と「顧客は部屋を予約することを選択する」の 2 つがテスト項目の行に記述される．
- 選択された網羅性に応じて出力されるテストシナリオ数が変化する．たとえば，提案システムへの入力時にステップ網羅を選択すれば，出力されるテストシナリオ群は各ステップが少なくとも 1 回はテストされる最低限の数だけテストシナリオ群を出力する．つまり，生成されるテストシナリオ数は以下の関係が成り立つ．
  ステップ網羅選択時 ≤ 分岐網羅選択時 ≤ 全実行フロー網羅選択時．
- 3 行目以降は，2 行目の列がステップであれば pass を，ドメインモデルの要素であれば null を記述する．提案システムが出力するのはテストシナリオの雛形にすぎないからである．たとえば，「システムは利用可能な設備を表示する」と同じ列には pass が，「部屋」と同じ列には null を記述する．提案システムの出力を詳細化するフェーズにおいて，pass や null の値を修正する必要がある．

⋆1 図 38 のみをユースケース間の関係を考慮せずに，ステップ網羅と分岐網羅を指定して generate _ testScenairos に入力した場合の実行フローのグラフはそれぞれ以下のとおり．
ステップ網羅指定時 $\{\{(s_1, s_2), (s_2, s_{A1.1}), (s_{A1.1}, s_{A1.2}), (s_{A1.2}, s_{A1.3}), (s_{A1.3}, s_3), (s_3, s_4)\}\}$
分岐網羅指定時 $\{\{(s_1, s_2), (s_2, s_3), (s_3, s_4)\}, \{(s_1, s_2), (s_2, s_{A1.1}), (s_{A1.1}, s_{A1.2}), (s_{A1.2}, s_{A1.3}), (s_{A1.3}, s_3), (s_3, s_4)\}\}$
ステップ網羅指定時のほうがグラフが小さいのは，1 の分岐によりすべてのステップが含まれるからである．

表 2　生成された「部屋を予約する」ユースケースに対応するテストシナリオファイル（その 2）（詳細化前）
Table 2　Generated test scenario 2 corresponding to use case of reserving room without detalils.

部屋を予約する				
システムは利用可能な…	部屋	顧客は部屋を予約…	≈	システムは顧客がこの新たな…
pass	*null*	pass	≈	pass
pass	*null*	pass	≈	pass

表 3　表 1 を人手で詳細化した例
Table 3　Example of detailed test scenario 1 corresponding to use case of reserving room.

部屋を予約する				
システムは利用可能な…	部屋	顧客は部屋を予約…	≈	システムはユースケースを…
List	*101*	Reserve	≈	exit
List	*201*	Reserve	≈	exit

実際に全実行フロー網羅を指定して出力されたテストシナリオ群の一部を表 1，**表 2** に，表 1 を人手で詳細化したものを**表 3** に，表 2 を人手で詳細化したものを**表 4** に示す．これらの詳細化は一例である．入力したドメインモデルは図 7 である．

4.3　フィクスチャの生成

本節は，提案システムへの入力フェーズで得られるフィクスチャの生成についてと，提案システムの出力を詳細化するフェーズで行うフィクスチャの詳細化を述べる．

表 4　表 2 を人手で詳細化した例
Table 4　Example of detailed test scenario 2 corresponding to use case of reserving room.

部屋を予約する				
システムは利用可能な…	部屋	顧客は部屋を予約…	≈	システムは顧客がこの新たな…
List	*101*	Reserve	≈	error
List	*201*	Reserve	≈	error

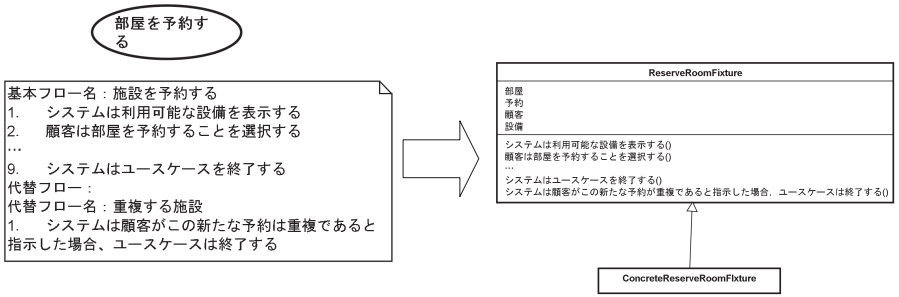


図 17　フィクスチャの生成例（詳細化前）
Fig. 17　Example of generated fixture before detailed.

提案システムによるフィクスチャの生成例を**図 17** に示す．ユースケース A に対し出力するとき，

- AFixture という名前のフィクスチャのみ
- AFixture という名前のフィクスチャとそれを継承する ConcreteAFixture という名前のフィクスチャ

のどちらかを提案システムの利用者が指定して生成する．開発者は，テストコードの一例として ConcreteAFixture を実装し，ConcreteAFixture を用いてテストする．選択式にしたのは，開発者が加えたフィクスチャの詳細をユースケースが変更となり，フィクスチャを再生成するときに上書きされるのを防ぐためである．また，ユースケース A に定義されている 1 つのステップにつき，ユースケース A に対応する親クラスとなるフィクスチャ AFixture は，該当のステップと同名の操作を所有する．同様に，ユースケース A のユースケース記述の中にドメインモデル中のクラス名を含むならば，ユースケース A に対応する親クラスとなるフィクスチャAFixture は，含まれるクラスを属性として所有する．図 17 内の ReserveRoomFixture のコード断片を**図 18** に示す．

```
public class ReserveRoomFixture
                extends ColumnFixture{
    public 部屋 部屋;
    …
    public String システムは利用可能な…() {
        return ”pass”;
    }
    …
```

図 18　「部屋を予約する」に対応するフィクスチャの出力例の一部（詳細化前）
Fig. 18　Part of fixture corresponding to use case of reserving room before detailed.

図 17 では「部屋を予約する」ユースケースにステップ「システムは利用可能な設備を表示する」,「顧客は部屋を予約することを選択する」, …,「システムはユースケースを終了する」,「システムは顧客がこの新たな予約は重複であると指示した場合，ユースケースは終了する」があるので，ReserveRoomFixture は「システムは利用可能な設備を表示する」,「顧客は部屋を予約することを選択する」, …,「システムはユースケースを終了する」,「システムは顧客がこの新たな予約は重複であると指示した場合，ユースケースは終了する」という名前の操作を所有し，入力されたドメインモデル内に「顧客」,「部屋」,「予約」,「設備」があるので AFixture は「顧客」,「部屋」,「予約」,「設備」を属性として所有する．提案システムにより出力されたフィクスチャを人手により詳細化した一例を，**図 19**，**図 20** に示す．システムは利用可能な設備を表示する () メソッドにおいて，実際にテストしたい機能である App クラスの listFacilities() メソッドを呼び出してテストする．

図 19 では，表 3 内の「システムは利用可能な…」の列が図 19 の「システムは利用可能な…」メソッドに対応し，表 3 内の「部屋」の列が図 19 の「部屋」の変数に対応している．表 3 の 3 行目では，順番に,「システムは利用可能な…」メソッドを実行してその結果が「List」と一致するか,「部屋」に 101 を代入し,「顧客は部屋を予約…」メソッドを実行しその結果が「Reserve」と一致するか,「システムはユースケースを…」メソッドを実行し「exit」と一致するか，テストすることを表す．

4.4　ユースケース間の関係に基づくフィクスチャの生成

ユースケース間の関係については，該当するフィクスチャ間の関係を，アスペクト指向プログラミング（横断的関心事をモジュール化する技術[11]）によるインタータイプ宣言（他のクラスが持つべきメソッド・フィールドを記述する機構[12]）で定義する．ユースケース間の関係をアスペクトで定義することにより，ユースケース間の関係が変化した場合に，ア

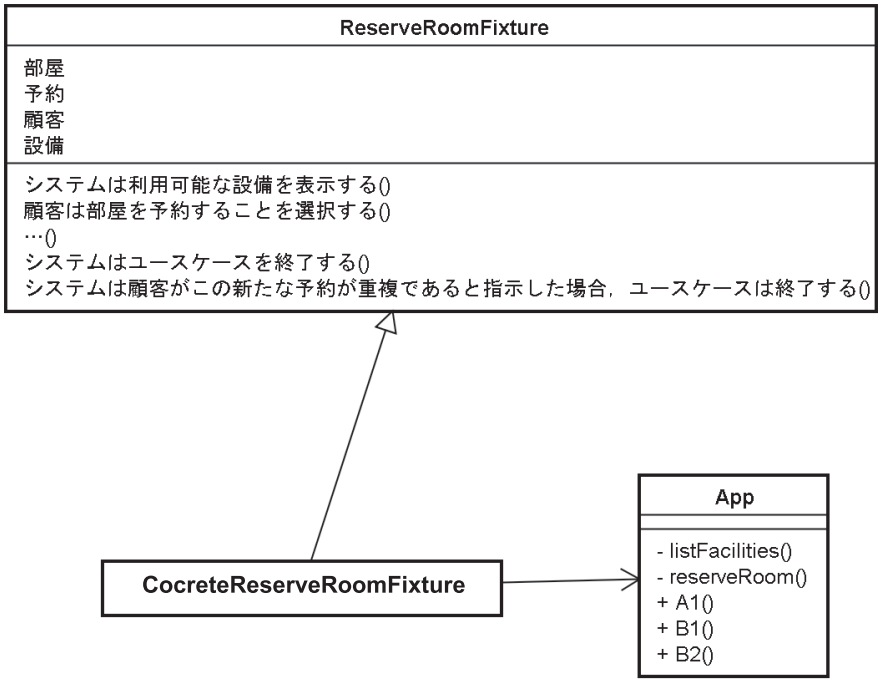


図 19　フィクスチャの生成例（詳細化後）
Fig. 19　Example of generated fixture after detailed.

```
public class ReserveRoomFixture
                extends ColumnFixture{
    public 部屋 部屋;
    …
    public String システムは利用可能な設備を表示する () {
        return App.listFacilities();
    }
    …
```

図 20「部屋を予約する」に対応するフィクスチャの出力例の一部（詳細化後）
Fig. 20　Part of fixture corresponding to use case of reserving room after detailed.

```
if(包含関係がある) {
  包含対象の参照と包含先の操作および
    同名の操作を追加するアスペクトを作成;
}
if(汎化関係がある) {
  該当するフィクスチャ間を
    継承関係にするアスペクトを作成;
}
if(拡張関係がある) {
  拡張する側への参照と拡張する側の操作および
    同名の操作を追加するアスペクトを作成;
}
```

図 21　アスペクト作成の手順
Fig. 21　Procedure of generating aspect.

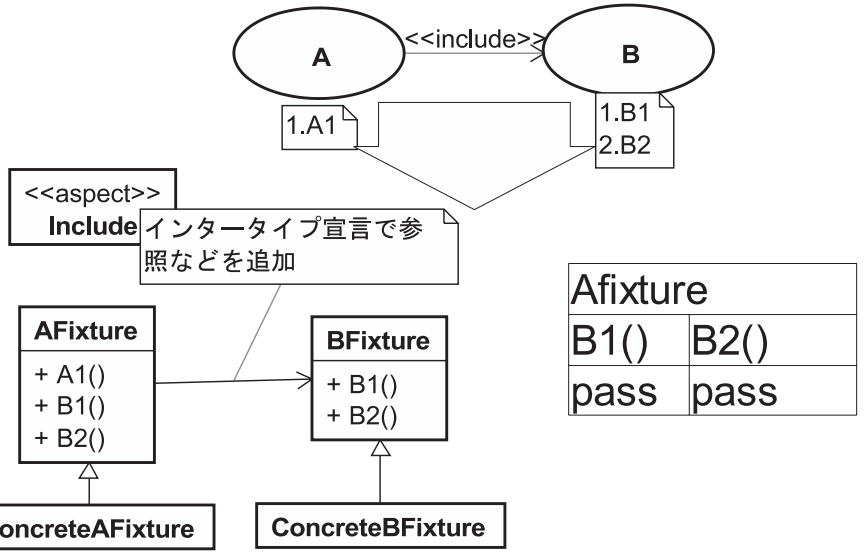


図 22　包含におけるフィクスチャの出力例（詳細化前）
Fig. 22　Example of generation of fixture in include before detailed.

スペクトのみを修正すればよく，フィクスチャ自体を変更しなくてすむため，フィクスチャの再利用性が高まる．本稿では AspectJ 形式のアスペクトを生成する．アスペクト作成の手順を**図 21** に示す．

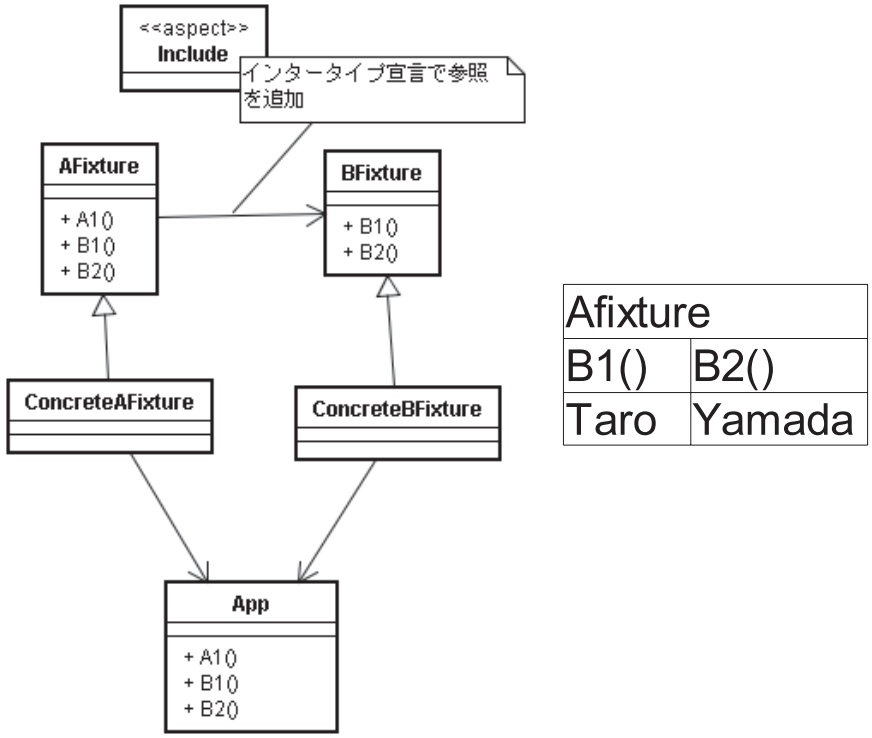


図 23　包含におけるフィクスチャの出力例（詳細化後）
Fig. 23　Example of generation of fixture in include after detailed.

包含　2 つのユースケース間が包含関係であれば，包含するユースケースに対応するフィクスチャに対し，包含されるユースケースに対応するフィクスチャを参照し，包含されるユースケースに対応するフィクスチャの操作と同名の操作を持つように変更するアスペクトを生成する．つまり，ユースケース A がユースケース B を包含しているとき，**図 22** のようなフィクスチャが出力される．アスペクトによるインタータイプ宣言で AFixture に，B1，B2 の操作と，BFixture への参照が追加される．図 22 を人手により詳細化した一例を**図 23** に示す．ConcreteAFixture と ConcreteBFixture にテストしたい App クラスへの参照を追記する．また，テストシナリオを詳細化し「Taro」，「Yamada」といった具体的な値を追記する．

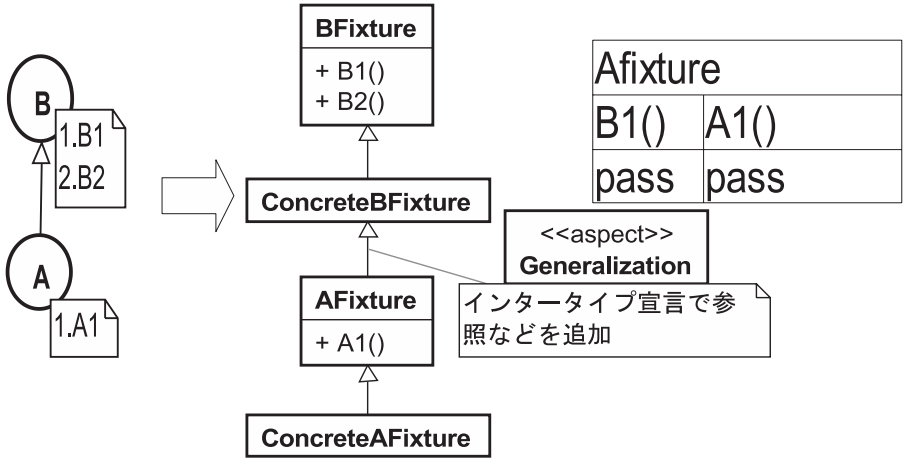


図 24　汎化におけるフィクスチャの出力例（詳細化前）
Fig. 24　Example of generation of fixture in generalization before detailed.

汎化　2 つのユースケース間が汎化関係であれば，当該のフィクスチャ間を継承関係にするアスペクトを生成する．つまり，ユースケース A がユースケース B を特化するとき，**図 24** のようなフィクスチャが出力される．アスペクトによるインタータイプ宣言で ConcreteBFixture と AFixture の間に継承関係を追加する．図 24 を人手により詳細化した一例を**図 25** に示す．ConcreteAFixture と ConcreteBFixture にテストしたい App クラスへの参照を追加する．また，テストシナリオを詳細化し「Taro」，「Yamamoto」といった具体的な値を追記している．

拡張　2 つのユースケース間が拡張関係であれば，拡張される側のユースケースに対応するフィクスチャに対し，拡張する側のユースケースに対応するフィクスチャを参照し，拡張する側のユースケースに対応するフィクスチャの操作と同名の操作を持つように追加するアスペクトを生成する．つまり，ユースケース A がユースケース B に拡張されているとき，出力されるフィクスチャのクラス構造は**図 26** のようになる．アスペクトによるインタータイプ宣言で AFixture に，B1，B2 の操作と，BFixture への参照が追加される．図 26 を人手により詳細化した一例を**図 27** に示す．ConcreteAFixture と ConcreteBFixture にテストしたい App クラスへの参照を追加する．また，テストシナリオを詳細化し「Yamamoto」，「Taro」，「Yamada」といった具体的な値を追記している．

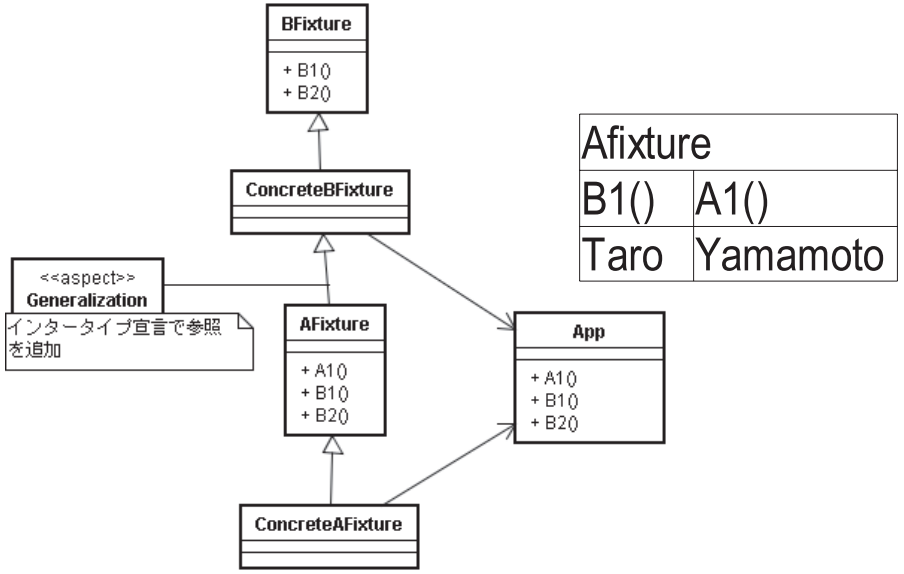


図 25　汎化におけるフィクスチャの出力例（詳細化後）
Fig. 25　Example of generation of fixture in generalization after detailied.

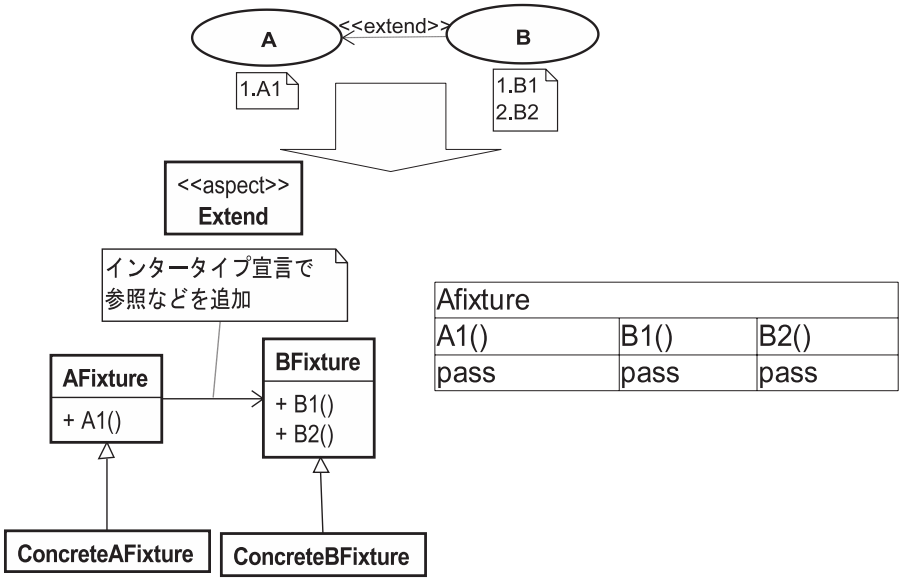


図 26　拡張におけるフィクスチャの出力例（詳細化前）
Fig. 26　Example of generation of fixture in extend before detailed.

以上のルールにより，図 1 のユースケース群に対し提案システムにより生成されたフィクスチャ群を**図 28** に示す．

4.5　提案手法および提案システムの限界

提案手法によるテストだけでは，受け入れテストとして不十分である．提案手法はステップの網羅に着目しており，たとえば，ユーザビリティなどの非機能要求はフローに記述されないため，他のテスト手法で補う必要がある．

また，提案手法は出力されたテストシナリオ群とテストフィクスチャ群に対し人手で詳細化する必要があるため，この工程において欠陥が混入する可能性がある．しかし，従来手法では実行フローの識別およびテストシナリオの実装を手動で行うのに対し，提案手法では実行フローの識別を自動化し，テストシナリオの実装のみを人手で行う．つまり，手動処理と比較して，提案手法は 1 段階減らしているため，欠陥が混入する可能性が少ないと考えられる．また，提案手法ではデータに依存する欠陥を検知できない可能性がある．

しかしながら，提案手法は機能テストのテストシナリオを特定の網羅基準に基づいて機械的に生成するため，機能テストの信頼性を向上させる．

提案システムの実装上の制約は以下のとおりである．しかし，以下の問題は提案手法の枠組み自体には依存していない．

(A)　ユースケース記述は専用のエディタで入力する．

(B)　生成されるフィクスチャは Java 形式である．

(C)　ドメインモデルはモデリングツールである Jude[10] の API を利用して入力するため，ドメインモデルは Jude で書かれているものとする．しかし，提案手法の枠組み自体は特定のモデリングツールに依存していない．また，Jude Professional では XMI 形式のインポート・エクスポートに対応しているため，現実的には大きな問題ではないと考えている．

5. 実　　験

従来手法[8] と提案手法の 2 種類に関して比較実験した．被験者は UML の経験が約 2 年の

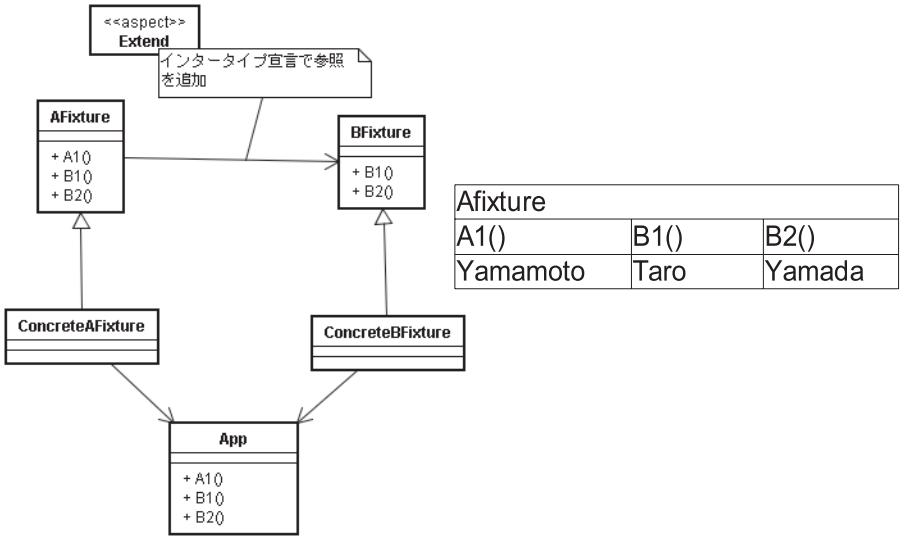


図 27　拡張におけるフィクスチャの出力例（詳細化後）
Fig. 27　Example of generation of fixture in extend after detailed.

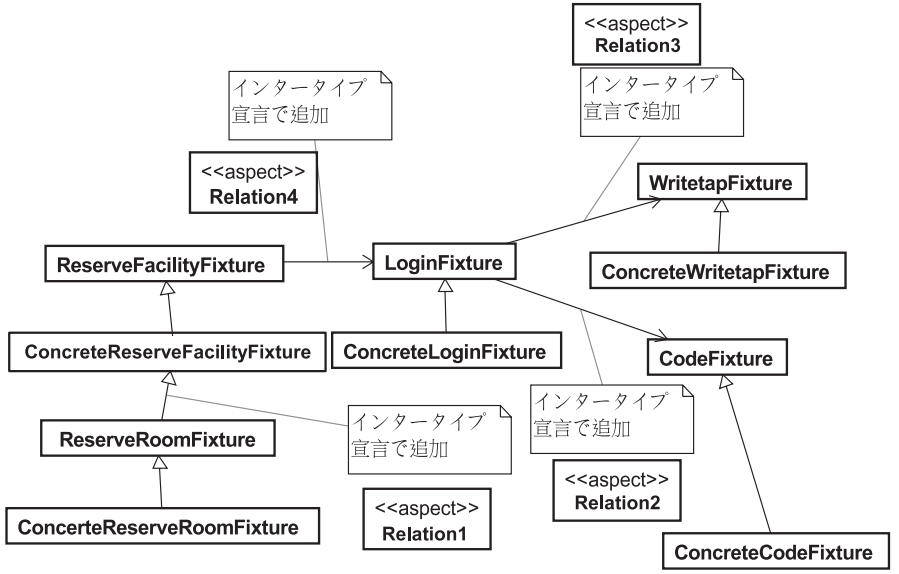


図 28　生成されたフィクスチャのクラス図
Fig. 28　Class diagram of generated fixture.

学生 6 名である．文献 13) 内にある**図 29** のユースケース図と付録 A.1 のユースケース記述を被験者に与えて，「衣装販売の履歴を確認する」ユースケースの実行されうるすべての実行フローについて識別を行い，分岐網羅になるようにテストシナリオをあげる．ユースケース数は 10，ユースケース記述数は 10，ユースケース間の関係数は 10（包含 9，拡張 1），ユースケースステップの合計は 47 ステップ，分岐数は 12，全フロー数は 32 である．その際に実行フローの誤りや漏れの数を比較した．提案手法や提案システムがその効果を発揮するであろうユースケースの規模は任意であるが，大規模であり，かつ，ユースケース間の関係が複雑であるほど，本手法は有効であると考える．そのため，この程度の関係の複雑さでも，従来手法[8]と提案手法の比較実験として妥当であると考える．この実験の目的は，以下の 2 点である．

- ユースケース間の関係が複雑なとき，人手での実行フローの識別に漏れや誤りが混入する可能性があることを示す．
- 人手での識別により誤りや漏れがある可能性がある場合において，提案システムによるユースケースの実行フローの自動識別により解決できるかどうかを示す．

被験者 A，B，C は，最初に従来手法による実行フローの識別および実行フローの列挙を行う．その後，従来手法の出力とは無関係に，同じ被験者がユースケース記述群を提案システムへ入力し，自動生成されたテストシナリオ群を得る．被験者 D，E，F は，最初にユースケース記述群を提案システムへ入力し，自動生成されたテストシナリオ群を得る．その後，提案手法の出力とは無関係に，同じ被験者がユースケース記述群から従来手法による実行フローの識別，および，実行フローの列挙を行う．結果を**表 5** に示す．

従来手法の場合，誤りや漏れの数は，被験者 A の誤りは 2 カ所，被験者 B，C，E，F の誤りは 1 カ所であった．被験者 D が従来手法を用いた場合と提案手法を用いた場合には誤りはなかった．従来手法において，被験者 A の誤りは，「反復される面会日時をスケジュールする」ユースケースの拡張による変化を忘れていたことと「衣装をデザインする」ユースケースを代替フローへ移行するステップを誤ったことである．同様に，被験者 B，E，F も「衣装をデザインする」ユースケースを代替フローへ移行するステップを誤っていた．また，被験者 C は「反復される面会日時をスケジュールする」ユースケースを拡張するステップを誤っていた．

これらの誤りは，分岐網羅および全実行フロー網羅の網羅率を低下させる．提案手法では

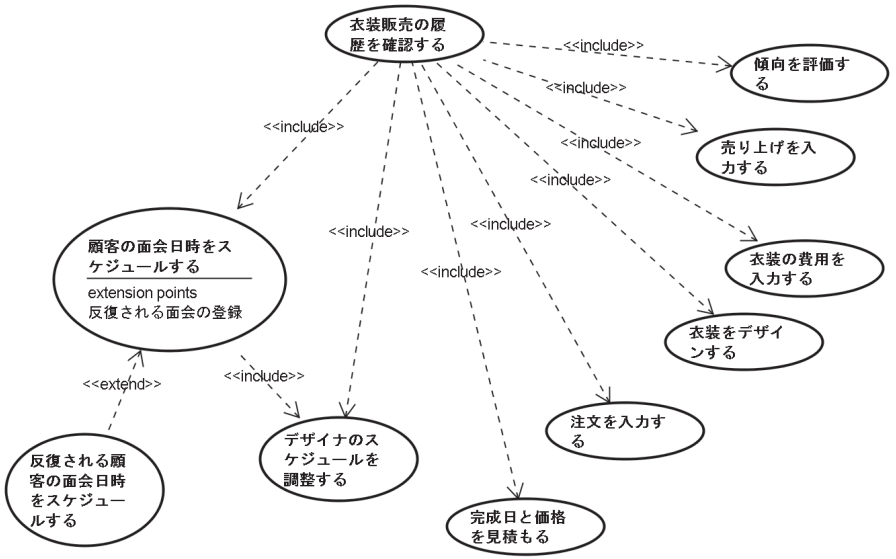


図 29　実験時に使ったユースケース図
Fig. 29　Use case diagram used when experimenting.

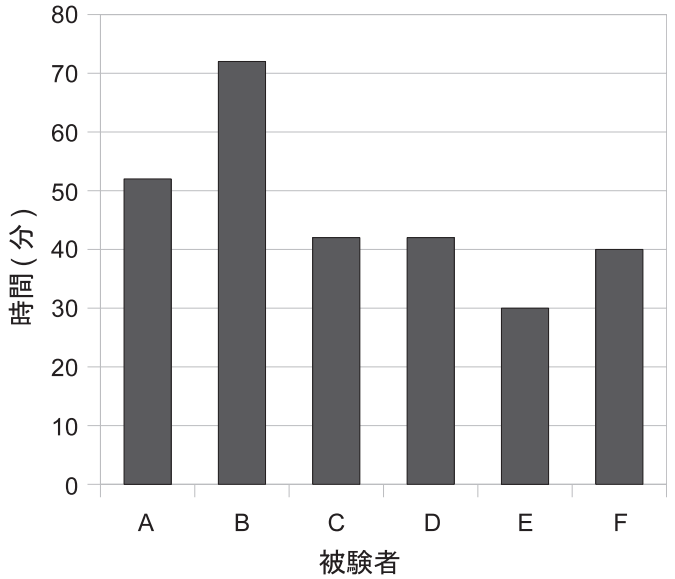


図 30　従来手法による識別の所要時間
Fig. 30　The time required of identification by technique in the past.

表 5　被験者のあげた実行フローの誤りの数
Table 5　Number of mistakes of execution flow listed by testee.

	被験者 A	被験者 B	被験者 C	被験者 D	被験者 E	被験者 F
先に実験した手法	従来手法	従来手法	従来手法	提案手法	提案手法	提案手法
従来手法	2	1	1	0	1	1
提案手法	0	0	0	0	0	0

実行フローを機械的に識別するため，誤りや漏れがなくなる．実験により，人手で識別する際に実行フローの識別に誤りが入る可能性があることを確認できた．この実験の規模では大きな影響は受けていないが，ユースケース数が数十，数百となる開発時は危険性は高くなる．提案手法を用いることで実行フローの漏れや誤りがなくなり，受け入れテストの網羅性が向上するので，提案手法は有用であると考える．また，従来手法による実行フローの識別にかかる時間を**図 30** に示す．最小値は 30 分，最大値は 72 分，平均値は 46.3 分，中央値は 42 分である．従来手法の場合において，実行フローの識別にこの程度時間がかかり，提案手法ではその時間を削減できる．

6. 関 連 研 究

Zielczynski は，ユースケースからテストケースへの追跡可能性について述べた[8]．Zielczynski の手法では，ユースケースからすべてのシナリオを人手で確認し，その中から妥当なテストシナリオを選択している．人手で識別するため，実行フローの識別に漏れがでるおそれがある．また，ユースケース間の関係は考慮していない．一方，提案手法では実行フローの識別を自動で行うため漏れがなくなり，ユースケース間の関係を考慮した網羅的なテストシナリオを作成する．

Some らは，テキスト形式のユースケースをもとにしたシステムレベルのテストシナリオ生成について述べた[14]．Some らの手法は，ユースケースからシステム全体の機能をテストするテストシナリオを生成し，ユースケースの早期検証を実現する手法である．しかし，ユースケース自体の検証が目的であるため，実行可能なテストシナリオは生成しない．一方，

ユースケース：ビデオを貸し出す
基本フロー：ビデオを貸し出す
1. 顧客は借りたいビデオと顧客の情報を入力する
2. システムは貸出処理をする
3. システムはユースケースを終了する
代替フロー：ID が存在しないとき
基本フローのステップ 1 の後に移行する
A1. システムは何らかの原因により，貸出しができないことを表示して終了する

図 31「ビデオを貸し出す」ユースケース記述
Fig. 31　Use case describing renting video.

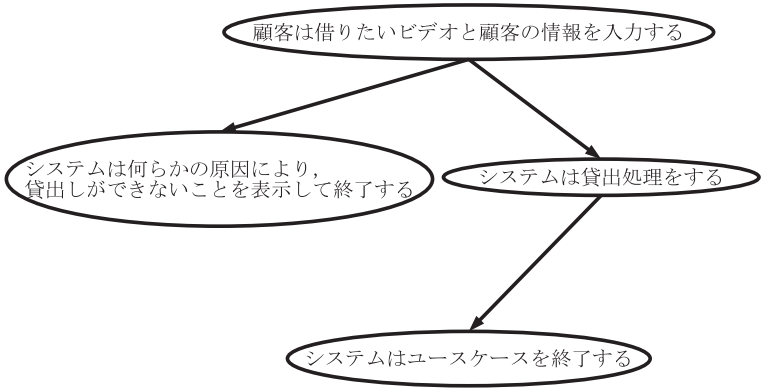


図 32　提案手法の定義によるユースケースのグラフ
Fig. 32　Graph of use case based on our proposal.

提案手法は雛形ではあるが，テストシナリオとテストプログラムを生成する．また，Some らの手法と提案手法とでは，考慮しているユースケースの関係が異なる．この手法では包含と事前条件および事後条件を考慮しており，提案手法では包含と汎化および拡張を考慮して，テストシナリオを生成する．また，Some らの手法と提案手法ではユースケースのグラフの定義が異なる．提案手法ではステップをノードとして定義している．一方，Some らの手法はステップはエッジとして定義し，ステップ実行前後のシステムの状態をノードとして定義している．一例として，「ビデオを貸し出す」ユースケース記述を**図 31** に示し，「ビデオを貸し出す」ユースケースを Some らの手法と提案手法のそれぞれのユースケースのグラフ定義に基づいてグラフ化したものを以下**図 32**，**図 33** に示す．

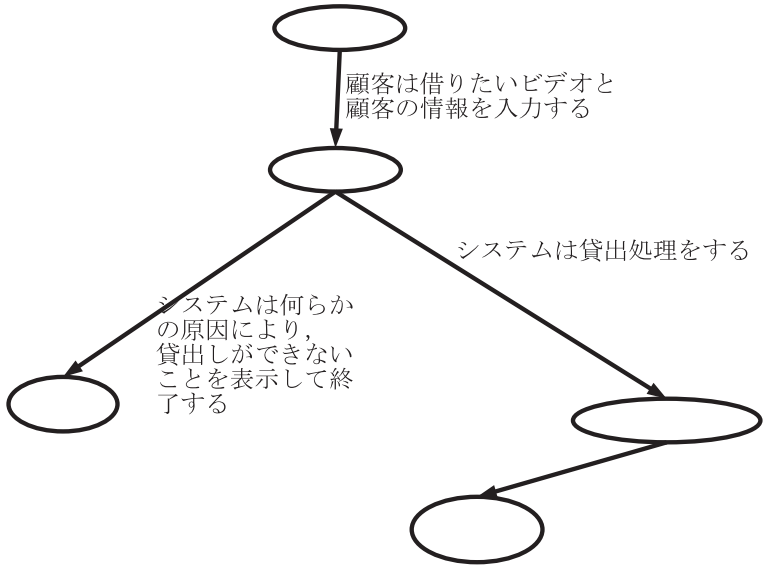


図 33　Some らの手法の定義によるユースケースのグラフ
Fig. 33　Graph of use case based on proposal by Some, et al.

表 6　Some らの手法と提案手法の網羅性の定義
Table 6　Definition of coverage of our proposal and proposal by Some, et al.

網羅性の定義	Some らの手法	提案手法
状態網羅	○	×
ステップ網羅	○	○
分岐網羅	×	○
全実行フロー網羅	○	○

Some らの手法と提案手法とではユースケースのグラフの定義が異なるため，網羅性の定義も異なる．それぞれの定義を**表 6** に示す．状態網羅とは，Some らの手法におけるユースケースのグラフにおいて，すべてのノード，つまりすべての状態が少なくとも 1 回はテストされるような網羅性のことである．各手法において定義されている場合は○，定義されていない場合を×としている．Some らの手法は状態網羅，ステップ網羅，全実行フロー網羅の 3 種類が定義されており，提案手法ではステップ網羅，分岐網羅，全実行フロー網羅の 3 種

類が定義されている．テストの観点に基づけば，分岐に関してバグが存在する確率が高いので[3]，分岐網羅を定義している提案手法の方がテストする場合について適している．分岐網羅を設定することにより，分岐条件の記述ミスや期待していないステップやフローへの遷移が原因である欠陥の発見が期待できる．

Amyot らは，ユースケースマップ駆動の WEB アプリケーションのテストを提案した[15]．Amyot らの手法は，ユースケースマップ（シナリオをモデリングするための記法）[16] を用いて受け入れテストの対象となる実行フローを識別し，テストシナリオの識別の正確性を向上させる手法である．しかし，複数のユースケースマップ間の関係を考慮しておらず網羅的なテストシナリオを作成しにくい．一方，提案手法ではユースケース間の関係を考慮した網羅的なテストシナリオを作成できる．

Nebut らは，ユースケース駆動の自動テスト生成について述べた[17]．Nebut らの手法は，ユースケース内の事前・事後条件に形式的な制約を追加してテストシナリオを自動生成する．形式記述を利用してテストシナリオを自動生成できるが，一般に形式記述はコストが高い．また，この手法は事前・事後条件を用いた網羅的なテストシナリオを生成する．一方，提案手法は非形式的記述を適用し，また，複数のユースケース間の関係を考慮するが，事前・事後条件は考慮していない．

7. おわりに

ユースケースから識別されたテストシナリオを用いた受け入れテストを行う際，広く受け入れられた網羅性の定義がない．そのため，テスト終了の判定に属人性が生じる可能性がある．その結果，受け入れテストの信頼性を損なうおそれがある．また，ユースケース間の関係が複雑な場合，人手での実行フローの識別に漏れが発生するおそれがあり，受け入れテストの信頼性を低下させる可能性がある．本稿では，ユースケースから識別されたテストシナリオに基づく受け入れテストに対して，3 種類の網羅性を定義した．また，ユースケース記述とドメインモデルを入力として，指定された網羅性を満たす FIT 形式のテストシナリオとテストプログラムの雛形を自動生成する手法を提案した．提案手法と従来手法について，小規模な実験を行い，提案手法により，実行フローの漏れがなくなったことを確認した．提案手法を用いることで，受け入れテストの網羅性の定義を用いて機械的に実行フローを識別するため，属人性を排してテスト終了を判定できる．また，求めるテスト網羅性を満たす必要十分なテストシナリオを自動生成することにより，受け入れテストの効率の向上が見込める．

今後の課題として，サンプル数を増やした追加実験や Java 以外の言語対応，実行フローだけでなくテストケースの生成，複数のモデリングツールの対応として XMI 形式への入力の対応などがあげられる．

参考文献

1) Kaner, C., Nguyen, H.Q. and Falk, J.（著）：テスト技術者交流会（訳）：基本から学ぶソフトウェアテスト—テストの「プロ」を目指す人のために，日経 BP 社 (2001).
2) Cockburn, A.（著），ウルシステムズ株式会社（訳）：ユースケース実践ガイド—効果的なユースケースの書き方，翔泳社 (2001).
3) Beizer, B.（著），小野間彰，山浦恒央（訳）：ソフトウェアテスト技法，日経 BP 出版センター (1994).
4) Object Management Group: Unified Modeling Language 2.0. http://www.uml.org/
5) Guttorm, S.: Eliciting security requirements with misuse cases, Requirements Engineering, Vol.10, No.1, pp.34–44 (2005).
6) Fowler, M.（著），羽生田栄一（訳）：UML モデリングのエッセンス第 3 版，翔泳社 (2005).
7) Mugridge, R. and Cunningham, W.: *Fit for Developing Software: Framework for Integrated Tests*, Prentice Hall (2005).
8) Zielczynski, P.: Traceability from Use Cases to Test Cases, developerWorks (2006). http://www.ibm.com/developerworks/rational/library/04/r-3217/3217_rm14.pdf
9) Heumann, J.: Generating Test Cases From Use Cases, Rational edge (2001). http://www.ibm.com /developerworks/rational/library/content/RationalEdge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf
10) JUDE Professional. http://jude.change-vision.com/jude-web/index.html
11) Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G.: An Overview of AspectJ, *Proc. 15th European Conference on Object-Oriented Programming* (*ECOOP 2001*), LNCS, Vol.2071, pp.327–354 (2001).
12) 長瀬嘉秀，天野まさひろ，鷲崎弘宜，立堀道昭：AspectJ によるアスペクト指向プログラミング入門，ソフトバンクパブリッシング (2004).
13) Kulak, D., Guiney, E.（著），市川和久（訳）：ユースケース導入ガイド—成功する要求収集テクニック，ピアソンエデュケーション (2002).
14) Some, S.S. and Cheng, X.: An approach for supporting system-level test scenarios generation from textual use cases, *Symposium on Applied Computing: Proc. 2008 ACM Symposium on Applied Computing*, pp.724–729 (2008).
15) Amyot, D., Roy, J.-F. and Weiss, M.: UCM-Driven Testing of Web Applications, *Proc. 12th SDL Forum* (*SDL 2005*), LNCS 3530, pp.247–264, Springer (2005).
16) Buhr, R.J.A., Casselman, R.S.（著），金沢典子，佐藤啓太（訳）：ユースケースマッ

プ—リアルタイムシステム開発へのオブジェクト指向的アプローチ，トッパン (1998).
17) Nebut, C., Fleurey, F. and Le Traon, Y.: Automatic Test Generation: A Use Case Driven Approach, *IEEE Trans. Softw. Eng.*, Vol.32, No.3, pp.140–155 (2006).

付　　録

A.1　実験に用いたユースケース記述群

実験に用いたユースケース記述を**図 34**，**図 35**，**図 36**，**図 37**，**図 38**，**図 39**，**図 40**，**図 41**，**図 42**，**図 43** に示す.

ユースケース：衣装販売の履歴を管理する
基本フロー名：衣装販売の履歴を管理する
1.　営業店長が顧客との面会をスケジュールする
2.　デザイナは顧客と相談して価格と完成予定日を見積もり，この情報をシステムに入力する
3.　システムは顧客に示された日までに品物を作成するためにデザイナのスケジュールを調整する
4.　営業店長が注文を入力する
5.　デザイナは顧客と会ったときに顧客のサイズとデザインの好みを入力する
6.　システムは生地と顧客のサイズに合うように型を調整する
7.　デザイナは衣装のデザインと仕立てを行う
8.　デザイナは衣装の費用を入力する
9.　システムは見積もり価格をもとにこの衣装の利益または損失を計算する
10.　営業店長は顧客が最終試着のために到着したときに売上情報を入力する
11.　オーナーが傾向を評価する
12.　オーナーはそこに示された近い将来の傾向を利用してビジネスを変革する
包含：
I1. 1.　「衣装販売の履歴を管理する」フローの「営業店長が顧客との面会をスケジュールする」のステップは，
「顧客の面会日時をスケジュールする」ユースケースを実行し，
「顧客の面会日時をスケジュールする」フローは元のフローに戻る
I2. 1.「衣装販売の履歴を管理する」フローの「デザイナは顧客と相談して価格と完成予定日を見積もり，この情報をシステムに入力する」のステップは，「完成日と価格を見積もる」ユースケースを実行し，
「完成日と価格を見積もる」フローは元のフロー戻る
I3. 1.「衣装販売の履歴を管理する」フローの「システムは顧客に示された日までに品物を作成するためにデザイナのスケジュールを調整する」のステップは，
「デザイナのスケジュールを調整する」ユースケースを実行し，
「デザイナのスケジュールを調整する」フローは元のフローに戻る
I4. 1.「衣装販売の履歴を管理する」フローの「営業店長が注文を入力する」のステップは，
「注文を入力する」ユースケースを実行し，
「注文を入力する」フローは元のフローに戻る
I5. 1.「衣装販売の履歴を管理する」フローの「デザイナは衣装のデザインと仕立てを行う」のステップは，「衣装をデザインする」ユースケースを実行し，
「衣装をデザインする」フローと「材料が足りない」フローは元のフローに戻る
I6. 1.「衣装販売の履歴を管理する」フローの「衣装の費用を入力する」のステップは，
「衣装の費用を入力する」ユースケースを実行し，
「衣装の費用を入力する」フローは元に戻る
I7. 1.「衣装販売の履歴を管理する」フローの「営業店長は顧客が最終試着のために到着したときに売上情報を入力する」のステップは，
「売上を入力する」ユースケースを実行し，
「売上を入力する」フローは元のフローに戻る
I8. 1.「衣装販売の履歴を管理する」フローの「オーナーが傾向を評価する」のステップは，
「傾向を評価する」ユースケースを実行し，
「傾向を評価する」フローは元のフローに戻る

図 34　「衣装販売の履歴を確認する」ユースケース記述
Fig. 34　Use case description of confirming history of the clothes sales.

ユースケース：顧客の面会日時をスケジュールする
基本フロー名：顧客の面会日時をスケジュールする
1.　営業店長が顧客から提示された日時の条件を使ってデザイナと顧客が相談する事のできる時間帯の一覧を要求する
2.　システムは可能な面会時間帯の一覧を示す
3.　営業店長は一覧の中からある時間帯を選択する
4.　システムは面会時間を記録する
5.　システムは面会の予定をデザイナに通知する
代替フロー：
代替フロー名：デザイナのスケジュールの妨げになる
「デザイナのスケジュールの妨げになる」フローは「営業店長は一覧の中からある時間帯を選択する」ステップの後に分岐する
A1．1.　システムは選択された面会時間がデザイナのスケジュールの妨げになる事を警告する
A1．2.　営業店長はデザイナのスケジュールを調整する
A1．3.　「システムは面会時間を記録する」に戻る
包含：
I1．1.「デザイナのスケジュールの妨げになる」フローの「営業店長はデザイナのスケジュールを調整する」のステップは，「デザイナのスケジュールを調整する」ユースケースを実行し，「デザイナのスケジュールを調整する」フローは元のフローに戻る
拡張点：反復される面会の登録
「顧客の面会日時をスケジュールする」フローにおける「システムは面会の予定をデザイナに通知する」ステップ

図 35「顧客の面会日時をスケジュールする」ユースケース記述
Fig. 35　Use case description of scheduling customer's interview date.

ユースケース：反復される顧客の面会日時をスケジュールする
基本フロー名：反復される顧客の面会日時をスケジュールする
1.　システムは営業店長に他にスケジュールしたい予定があるか尋ねる
2.　営業店長に追加したい日時を選択する
拡張：「反復される面会の登録」の前
「反復される顧客の面会日時をスケジュールする」フローは元の戻る

図 36「反復される顧客の面会日時をスケジュールする」ユースケース記述
Fig. 36　Use case description of scheduling interview date of the repeated customer.

ユースケース：デザイナのスケジュールを調整する
基本フロー名：デザイナのスケジュールを調整する
1.　システムはデザイナのスケジュールを表示する
2.　営業店長はデザイナのスケジュールを整理して入力する

図 37「デザイナのスケジュールを調整する」ユースケース記述
Fig. 37　Use case description of adjusting designer's schedule.

ユースケース：完成日と価格を見積もる
基本フロー名：完成日と価格を見積もる
1.　営業店長はデザインに関する情報をシステムに入力する
2.　システムは作業量，価格の見積もりを計算する
3.　営業店長はその価格を承認してそれを顧客に提示する
4.　システムはこの情報を保存してデザイナのスケジュールにこの仕事を組み入れる
代替フロー：
代替フロー名：指定された日時に間に合わない
「指定された日時に間に合わない」フローは「システムは作業量，価格の見積もりを計算する」ステップの後に分岐する
A1．1.　システムは指定された日時に間に合わないことをデザイナ，営業日時に警告する
A1．2.　営業店長はスケジュールの調整を行う
A1．3.　「営業店長はその価格を承認してそれを顧客に提示する」に戻る

図 38「完成日と価格を見積もる」ユースケース記述
Fig. 38　Use case description of estimating completion day and price.

ユースケース：注文を入力する
基本フロー名：注文を入力する
1.　営業店長が価格の見積もり，デザインの好み，完成品の見積もりを含む顧客の注文情報を入力する
2.　システムは注文情報を保存する

図 39「注文を入力する」ユースケース記述
Fig. 39　Use case description of inputting order.

ユースケース：衣装をデザインする
基本フロー名：衣装をデザインする
1.　デザイナは使われる生地と装飾品の種類，および手元にあるそれらの量を入力する
2.　システムはその顧客と衣装に関してそれまで保存されていた情報を表示する
3.　デザイナは衣装のデザインを対話的に作成する
4.　システムはデザイナが生地のカットを始めるのに使うことのできるデザインを制作する
代替フロー：
代替フロー名：デザイン情報が足りない
「デザイン情報が足りない」フローは「システムはその顧客と衣装に関してそれまで保存されていた情報を表示する」ステップの後に分岐する
A1．1.　システムはデザイン情報を追加入力するように促す
A1．2.　「デザイナは衣装のデザインを対話的に作成する」に戻る
代替フロー名：材料が足りない
「材料が足りない」フローは「システムはデザイナが生地のカットを始めるのに使うことのできるデザインを制作する」ステップの後に分岐する
A2．1.　システムは材料が足りない旨を表示し，不足している商品を手配しておく

図 40「衣装をデザインする」ユースケース記述
Fig. 40　Use case description of designing clothes.

ユースケース：衣装の費用を入力する
基本フロー名：衣装の費用を入力する
1.　デザイナは費用の総額をシステムに入力する
2.　システムはこの情報を保存する

図 41「衣装の費用を入力する」ユースケース記述
Fig. 41　Use case description of inputting cost of clothes.

ユースケース：売上を入力する
基本フロー名：売上を入力する
1.　営業店長が顧客名，日時，衣装，売上金額を入力する
2.　システムはこの情報を保存する

図 42「売上を入力する」ユースケース記述
Fig. 42　Use case description of inputting sales.

ユースケース：傾向を評価する
基本フロー名：傾向を評価する
1.　オーナーが傾向情報を要求する
2.　システムは指定された視覚的な傾向情報を表示する
3.　オーナーはビジネスの何らかの側面を変更するのにこの情報を利用する

図 43「傾向を評価する」ユースケース記述
Fig. 43　Use case description of evaluating tendency.

(平成 21 年 4 月 3 日受付)
(平成 21 年 9 月 11 日採録)

雁行　進夢

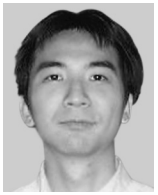
1985 年生．2008 年早稲田大学理工学部卒業．同年早稲田大学大学院基幹理工学研究科情報理工学専攻入学．2009 年テスト技術者振興協会善吾賞．

久保　淳人（正会員）

1981 年生．2009 年早稲田大学大学院博士後期課程修了，博士（工学）．2007 年早稲田大学基幹理工学部助手を経て，2009 年より国立情報学研究所アーキテクチャ研究系特任助教ならびに東洋大学経営学部非常勤講師．ソフトウェアパターンおよびモデル検証を中心としたソフトウェア開発工学の研究に従事．情報処理学会ソフトウェア工学研究会運営委員．2009 年テスト技術者振興協会善吾賞．

鈴木三紀夫（正会員）

1992 年（株）東洋情報システム（現，TIS（株））に入社．顧客向けシステム開発に従事後，2004 年よりソフトウェアテストのプロセス改善活動に従事．2006 年より要件定義のプロセス改善に従事．現在はオフショアプロジェクトのプロセス改善を担当．日本品質管理学会正会員．



鷲崎　弘宜（正会員）

1976 年生．2003 年早稲田大学大学院博士後期課程修了，博士（情報科学）．同大学助手，国立情報学研究所助手を経て，2008 年より同大学理工学術院准教授，および，同研究所客員准教授．再利用と品質保証を中心としたソフトウェア工学の研究と教育に従事．他の活動に情報処理学会代表会員，同学会論文誌編集委員会基盤グループ副査，日科技連 SQiP 研究会運営小委員会委員長，FOSE2009 プログラム委員長等．2004 年 JSSST 高橋奨励賞．2006 年 SES2006 優秀論文賞，2008 年 IPSJ 山下記念研究賞，2008 年船井情報科学奨励賞，2009 年テスト技術者振興協会善吾賞，2009 年 FIT2009 船井ベストペーパー賞．

深澤　良彰（正会員）

1976 年早稲田大学理工学部電気工学科卒業．1983 年同大学大学院博士課程修了．同年相模工業大学工学部情報工学科専任講師．1987 年早稲田大学理工学部助教授．1992 年同教授．工学博士．ソフトウェア再利用技術を中心としてソフトウェア工学の研究に主に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE，ACM 各会員．