Vol.2012-SE-178 No.4
2012/11/1

モデル検査技術を用いたユースケース記述分析手法の提案

川井彬央[†1]　　岸知二[†1]

ソフトウェア開発において，要求定義の工程で欠陥を発見することは非常に重要である．要求定義において様々な手法があるが，その中でもユースケース記述は，ユーザーとシステムとのやりとりや状況などを具体的に記述することで要求の把握に用いられる．1 つのシステムにおいてユースケースは，外界に対するサービスのタイプに応じてそれぞれ作成されるので，複数のユースケースが作成される．しかし，ユースケース中の系列記述はそのユースケースの単位で書かれることが多く，他のユースケースとのサービスレベルの競合までは必ずしも考慮されていない．そのため，複数のユースケース記述に基づいて仕様定義をする際には，ユースケース間のサービス競合を考慮しながら競合のない 1 つの仕様へと洗練する必要がある．そこで，本稿では複数のユースケース記述からモデル検査技術を使ってサービス競合を発見する手法を提案する．

Analysis Method of Use Case Descriptions using Model Checking Techniques

AKIHISA KAWAI[†1]　TOMOJI KISHI[†1]

In software development, it is very important to find a defect with the request process. There are various methods in the request process, use case descriptions are used to understand the request by describing specifically the interaction or status with the user and the system. Because use cases are created, depending on the type of service for each of the outside world in the system, multiple use case descriptions are created. However, the sequence descriptions of use case are often written in units of the use case, the level of service to compete with other use cases are not necessarily taken into account. Consequently, must be refined and gradually to the specification in consideration of services conflicts between use cases when the specification of one together multiple use cases. In this paper, we propose a method to detect the service conflicts multiple use case descriptions using model checking techniques.

1. はじめに

ソフトウェア開発において，計画通りにプロジェクトを遂行することは難しく，不具合や納期遅れなどの問題が起きることが多い．品質，コスト，納期という点で見ると，これらをすべて満足して終えることができるのは 6 割で，その他の 4 割は何らかの不具合が発生し[1]，場合によっては品質，コスト，納期すべてが満足しないこともある．特に，開発の途中に何らかの不具合や欠陥が発生すると修正する必要があるが，場合によっては前工程に戻る必要が出てきてしまい，非常に手間がかかる．

ソフトウェア開発において，最も欠陥が発生する工程は要求定義（50%）であり，設計（26%），実装，テストと続く（図 1）．また，コストにおいても早い段階で修正しない表 1 のように莫大な数値になってしまう．従って，品質や開発コストを改善するには，要求定義の工程で欠陥を発見することが重要となる．

要求定義には様々な手法があるが，その中でもユースケース記述は，ユーザーとシステムとのやりとりや状況などを具体的に記述することで要求の把握に用いられる．ユースケースは様々な要求に応じてそれぞれ作成されるので，複数のユースケースが作成されるが，それらは他のユースケースの状態などを必ずしも考慮しているわけではない．そのため，複数のユースケース記述に基づいて 1 つの仕様定義をする際には，ユースケース間での競合を解消するように洗練する必要がある．

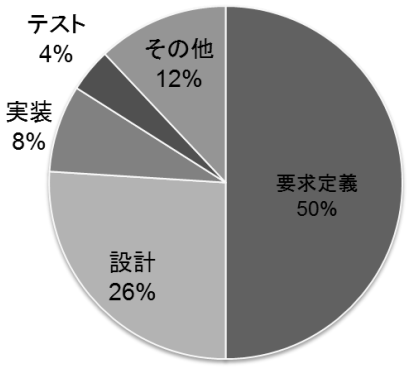

図 1　欠陥が発生する工程[2]
Figure 1　Process of defect occurs.

表 1　修正コストの比率[3]（要求定義を 1 とする）
Table 1　The ratio of the cost of fixing.

要求定義	設計	実装	テスト	出荷後
1	5	10	20	200

†1 早稲田大学
Waseda University

そこで本稿では，複数ユースケース間に発生する競合を発見する手法を提案する．

要求定義において，要求を検証する研究が行われているが，人手による手法が一般的である．そこで，ソフトウェアの要求定義の検証において，形式手法による検証が注目されている．形式手法とは，数理論理学などに基づく言語や，証明技法からなる設計手法などに関係する総合的な技術である．形式手法を用いることで，記述に不具合がないことを数学的に証明，厳密な言語により仕様の明確化，記述中の誤りや不具合を早期に発見などの効果がある[4]．また，形式手法の方法によってはツールによる支援が可能となり，半自動的に行うことができる．

そこで本稿では，ユースケース記述を形式手法の 1 つであるモデル検査技術を用いて，ユースケース間の競合を発見する検証手法を提案する．

本稿の構成について，2 章ではソフトウェア開発の要求定義プロセスと本稿の対象である要求分析の検証について説明する．3 章では関連研究について説明する．4 章では提案手法について説明し，5 章ではその手法を用いた検証例を述べる．6 章では提案手法や検証例に対する考察を述べ，7 章ではまとめと今後の予定について述べる．

2. 要求定義プロセスにおける要件分析

2.1 要求定義プロセス

ソフトウェア開発において，要求定義，設計，実装，テストといった工程があるが，その中の要求定義においても段階がある．要求定義プロセスは図 2 のようになる[5] [6].

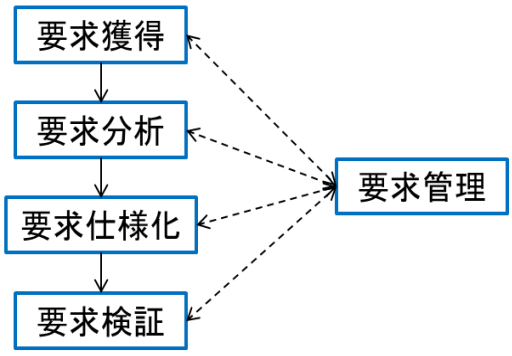

図 2　要求定義プロセス
Figure 2　Requirements definition process.

要求定義プロセスは要求獲得から始まり，要求分析，要求仕様化，要求検証と続く．また，後の変更や再利用に対応する要求管理がある．

要求獲得では，システム開発を依頼する顧客の現状の問題発見, 要求の抽出, ステークホルダーの識別などを行う．それを要求分析において，異なる要求同士の競合の調整や見落とされた要求の発見，代替案などを分析する．要求仕様化では，要求分析したものを仕様記述や図，UML などでモデル化，仕様化する．その仕様化したものを要求検証において，レビュー，シミュレーション，テストなどによって解析する．要求管理では，要求が変更したときの追跡や要求の再利用を行う．

2.2 要求分析の検証

要求おけるシナリオは，特定の局面から具体的，断片的に記述したものである．その一方，仕様は一般的現象をシステムの観点から抽象的，網羅的に記述したものであり，シナリオとはお互いを補完し合う関係ということもできる[5]．具体的な作業やシステムの動作を記述し，それを分析することで要求を抽出する．これがシナリオ分析である．その他にも, 具体的な記述であるシナリオにより, 顧客側, 開発側とも内容の理解や両者のコミュニケーションが容易になる．

また，要求分析の一手法であるユースケース分析法は，ユーザーからの要求の集合（ユースケース）によってシステムをモデル化し，ユースケース記述を使って具体的に表現し，それを分析する手法である．その意味ではシナリオ分析の一種とも言える[5].

しかし，単に獲得した要求に応じてユースケースやシナリオを記述するだけでは他の要求のことを考慮しているとは必ずしも限らない．そのため，他の要求との兼ね合いや競合などを分析しそれを反映させる必要がある．よって，要求プロセスの流れとして，要求分析において獲得した要求に対し分析を行い, 要求同士の競合などを回避した後に，それを要求仕様化するという流れとなる．

そこで本稿では，図 2 の要求定義プロセスにおいて要求分析の段階で検証を行う．要求分析におけるユースケース記述において，競合を発見する．それを今までは人手で行われていたが，形式手法を用いることにより，競合を発見し易くする．これにより，要求分析の次の段階である要求仕様化や設計よりも早い段階で競合を発見し，それを解消することで品質の向上が期待される．

3. 関連研究

形式手法のモデル検査技術を用いて，ユースケース記述（あるいはユースケース）を検証している研究として[7] [8]がある．どちらの研究も自然言語で書かれたユースケースでは，曖昧さや記述漏れがありそのままではモデル検査ツールで検証しにくいという点から, ユースケース記述（あるいはユースケースそのもの）について議論している．

[7]はユースケース記述について，論理演算子を用いて論理式のように表現することで明確性を上げている． [8]はユースケースの曖昧さを減らすように，ユースケースを拡張した構造化ユースケースを提案している．検証内容は，ユーザーの要求を満たすかどうかの安全性や，事前事後条件が成立するか，決められたアクターのみが実行できるか

などが行われている．

また，ユースケースの競合ではなく，フィーチャインタラクションにおける競合の検出をモデル検査技術で行った研究として[9] [10]などの多くの研究がある．フィーチャインタラクションとは，通信システムなどにおける既存のサービスに新しいサービスを追加したときに発生する競合である．[9]では電話通信システム，[10]では電話通信システムとホームネットワークについて，モデル検査技術で競合を検出している．

本稿ではモデル検査技術を用いて，複数ユースケース間に関わる競合を検出する手法を提案する．

4. 提案手法

4.1 提案アプローチ

ユースケース記述では，要求に応じて様々なユースケースが作成され，それをより詳細に系列記述している．このとき，ユースケースの系列記述はユースケース単位で書かれるため，他のユースケースとの兼ね合いを考慮していないことが多く，競合が発生する可能性がある．それを従来は人手で行われてきた．

そのため，本稿ではユースケース記述における競合を発見するための手法を提案する．競合とは，2 つ以上の事柄が同時に重なることで互いに干渉することにより，想定した結果を得られないことである[11]．ユースケースレベルで起きる競合として，機能競合と環境競合がある[10] [11].

機能競合は，同一機器において複数の機能が実行された時の競合で，例えば携帯電話で“アラームを鳴らす”と“着信音を鳴らす”という記述が別々のユースケースにあるとき同時に実行すると，どちらが起きるかわからない．その一方，環境競合は異なる機器の機能がそれらの周囲の環境に対して間接的に起きる競合である．例えば“エアコンで部屋を暖める”と“換気扇で空気を入れ替える”という記述が別々のユースケース記述のとき，同時に実行すると，外が寒ければエアコンの効率が悪くなってしまう．これは，空気という機器周囲の環境に対して，間接的に競合している．これらの競合のうち，本稿では機能競合について検証を行う．

複数ユースケースが合わさったとき，ユースケース間における機能競合が起きないか検証するために，本稿では形式手法の 1 つであるモデル検査技術を用いて検証を行う．モデル検査技術は，対象の状態遷移の動きを計算機上で模倣し，その中で与えられた性質が成り立つかどうか網羅的に検査することができる．複数のユースケースはそれぞれ独立して動いており，それぞれの動作や状況をすべて考慮すると非常に複雑になり，競合の発見が難しくなる．そこで，あらゆる場面を網羅的に検査できるモデル検査技術を用いる．また，ツールにより検査し易くなる．

そこで，複数ユースケースにおいてユースケース記述内に登場するリソースに注目する．それをまとめて状態遷移モデルを作成し，モデル検査技術を用いて全体の振る舞いを網羅的に検査することで競合を発見する．

4.2 提案手法の流れ

本稿における，モデル検査技術を用いたユースケース記述の検証手法の流れは図 3 のようになる．

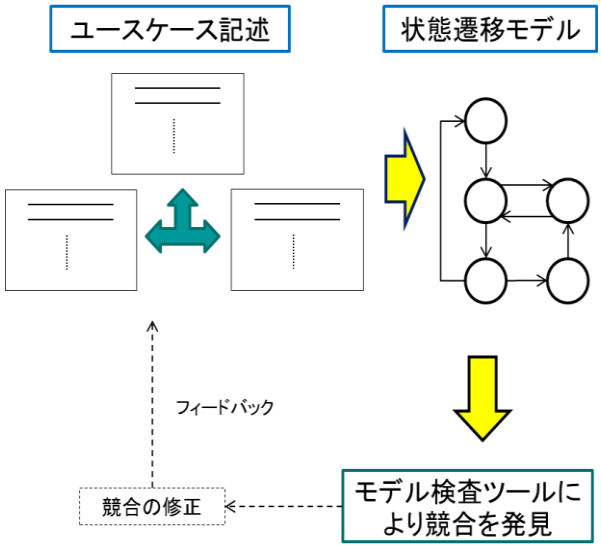

図 3　提案手法の流れ
Figure 3　Order of the proposed method.

まず，要求獲得で抽出した要求とその要求を満たすために，システムに必要なサービスや流れなどをユースケース記述で書く．1 つのシステムにおいて様々な要求があるため，複数のユースケースが作成される．この時点での系列記述は，ユースケース単位のものであり他のユースケースのことを必ずしも考慮しているとは限らないので競合が発生する可能性がある．そこで，モデル検査技術を用いて競合を発見する．

そのため，ユースケース記述から情報を抽出して状態遷移モデルを作成する．ユースケースは複数あるので，状態遷移モデルも複数となる．これらを合わせて 1 つの状態遷移モデルとする．これをモデル検査ツールに入力する．そして，モデル検査ツールで競合状態となるかどうか検証を行う．競合状態になる事を確認したら，それを修正する．修正方法は単にユースケースの条件設定など開発側だけではなく，ハードウェアや顧客側との相談で決めることも出てくる．それによって競合を修正し，ユースケース記述に反映させ再びモデル検査ツールを使用することにより，競合が修正されたことが確認される．

競合の修正は顧客側にも関わる事なので，本稿ではユースケース記述と，ユースケース記述から情報を抽出し状態遷移モデルを作成するところを考案する．

4.3 ユースケース記述と競合

ユースケース記述の記述法はフォーマットなどが存在するが，特に書き方は決まっていない．関連研究[7][8]や[12]などのユースケースで書かれている項目について多いものをまとめると，例えば表 2 のようなユースケーステンプレートができる．表 2 の項目以外ではユースケースの目的，要約（あるいは概要），レベル，拡張など様々な書き方があるが要求レベルの段階などによって異なる[12].

ユースケーステンプレートに沿って，獲得してきた要求を満たすようにユースケース記述で表現する．このとき，例えば携帯電話のユースケースとして図 4 と図 5 があったとき, 2 つのユースケースにおいて競合が発生する. なお，このユースケースではユースケース名，アクター名，基本系列のみ記述し，事前条件などは省略している．

表 2　ユースケーステンプレートの例
Table 2　Examples of the use case template.

項目	内容
ユースケース名	ユースケースを一意に識別できるもの
アクター名	ユースケースを実際に実行する人
事前条件	そのユースケースが開始される時点の成立している条件
事後条件	そのユースケースを終えた時に成立している条件
基本系列（基本フロー）	ユースケース実行時の一般的な流れ
代替系列（代替フロー）	基本フローにおける別の処理が発生したときの流れ

ユースケース名 : 充電する
アクター名 : ユーザー
1. ユーザーは携帯電話に充電器をさす
2. システムはランプを光らせる
3. システムはランプを消す

図 4　“充電する”ユースケース
Figure 4　"Charging the battery" use case.

ユースケース名 : 着信する
アクター名 : ユーザー
1. システムは電波を受信する
2. システムは音を鳴らす
3. システムはランプを光らせる
4. ユーザーは通話ボタンを押す

図 5　“着信する”ユースケース
Figure 5　"Incoming call" use case.

図 4 では携帯電話を充電するときの状況，図 5 では電話がかかってきた状況を表している．このとき，図 4 の 2 番目のプロセスと図 5 の 3 番目のプロセスにおいてランプを光らせるという状況がある．このランプは 1 つしかなく，例えば図 4 では赤色，図 5 では緑色に光らせる場合において，充電中に電話がかかってきたときに何色に光らせるべきかこの記述ではわからない．そのため，1 つのランプにおいて機能競合が発生する．

動作する機器自体や入出力装置など，サービスを行う際に必要なものを資源というが，本稿ではそれを含めサービスとして本質的に必要とされるものも資源とする．また，機能競合のうち今回のランプのように同一資源を奪い合う状態を資源の競合という．本稿において，このような記述内における競合を発見する方法を考案する．

4.4 情報の抽出と検証方法

上記のような競合を発見するために，ユースケース記述から必要な情報を抽出し，状態遷移モデルを生成する．今回のモデル検査で必要な情報は，装置やサービスを行う際に必要となる先述の資源と，資源の状態遷移，そしてユースケース記述の基本系列の状態遷移である．これをユースケース記述から抽出する．必要な情報を抽出し，状態遷移モデルを生成してモデル検査するまでの手順は次の通りとなる．

(1) ユースケース記述の基本系列から資源を見つける
(2) 基本系列の動詞に注目し，資源の状態と遷移を定め，これを資源の状態遷移とする
(3) 基本系列の状態遷移により，資源の状態遷移が起きるようにし，これをユースケースの状態遷移モデルとする
(4) 他のユースケースにおいても(1),(2),(3)を行い，ユースケースごとに状態遷移モデルを作成する
(5) ユースケースの状態遷移モデルで矛盾などが発生したら普通状態から競合状態に遷移するというモデルを作成する
(6) 上記の作成したモデルを合成し，1 つの状態遷移モデルとして，これをモデル検査ツールに入力する
(7) “競合状態に遷移する”をプロパティとしてモデル検査する

ユースケース記述の基本系列において図 4 や図 5 のように，“システムは○○を□□する”や“ユーザーは○○を□□する”という記述が多い．本稿では，基本系列がこのように書かれているとする．この記述の中から資源を見つける. ランプなどの資源は,“○○”の部分に当てはまるので，これを抽出する．

次に記述中の動詞に注目し，資源の状態遷移を定める．動詞は“□□する”という表現であり，“システムは○○を□□する”とあるようにシステムは資源をその動詞の通り

に実行することになる．すなわち，資源がその動作を行っている状態になるということになる．逆に“システムは○○を□□する”の前の段階では資源がその動作を行っていない状態とみることができる．そこで，抽出した資源が動作を“行っている”状態と，動作を“行っていない”状態と定める．これをすべての基本系列に当てはめ，それを“行っていない”状態から“行っている”状態に遷移する．この遷移が，資源の状態遷移となる．図 4，図 5 のユースケースにおいて，ランプを例とすると，まずランプという資源を抽出する．また，“光らせる”という動詞に注目し，光らせるということはその動詞を行っていない状態である“光っていない”という状態があることになる．すなわち，ランプという資源は“光っていない”状態から“光らせる”状態に遷移するという状態遷移を持つ．

ユースケース記述において基本系列は順番に記述されており，プロセスを状態として捉えて順番に遷移しているとみることができる．そしてプロセスが遷移することで，資源の状態遷移も起こる．例えば図 4 では，“ユーザーは携帯電話に充電器をさす”というプロセスが“システムはランプを光らせる”というプロセスに遷移しているとみることができる．そして，“システムはランプを光らせる”プロセスに遷移したら，資源の状態遷移である“光っていない”状態から“光らせている”状態に遷移する．このプロセスの遷移を基本系列の状態遷移とする．基本系列の状態遷移は自律的に行われ，その遷移が起きることで資源の状態遷移が行われる．

これを図 5 に当てはめると，“システムは電波を受信する”状態により“電波”という資源は “受信していない”状態から，“受信している”状態に遷移する．そして，“システムは電波を受信する”状態から“システムは音を鳴らす”状態に遷移する．これにより“音”という資源は “鳴っていない”状態から，“鳴っている”状態に遷移する．続いて，“システムは音を鳴らす”状態から“システムはランプを光らせる”状態に遷移し，これにより“ランプ”という資源は “光らせていない”状態から，“光らせている”状態に遷移する．最後に ,“システムはランプを光らせる”状態から“ユーザーはボタンを押す”状態に遷移し，これにより“通話ボタン”という資源は “押していない”状態から，“押している”状態に遷移する．

図 5 のユースケースに基づき，資源の状態遷移と基本系列の状態遷移を関連づけると，図 6 のような遷移となる．図 6 における細い右矢印は資源の遷移，太い下矢印は基本系列の遷移を表している．各資源の初期状態は“受信していない”などの基本系列の動作を行う前の状態である．基本系列の状態遷移が起きると，資源は初期状態から遷移する．これがユースケースの状態遷移モデルとなる．

図 4 においても同様に，資源の状態遷移と基本系列の状態遷移からユースケースの状態遷移モデルを作る．このとき，“システムはランプを光らせる”は図 5 と同じ状態遷移とする．

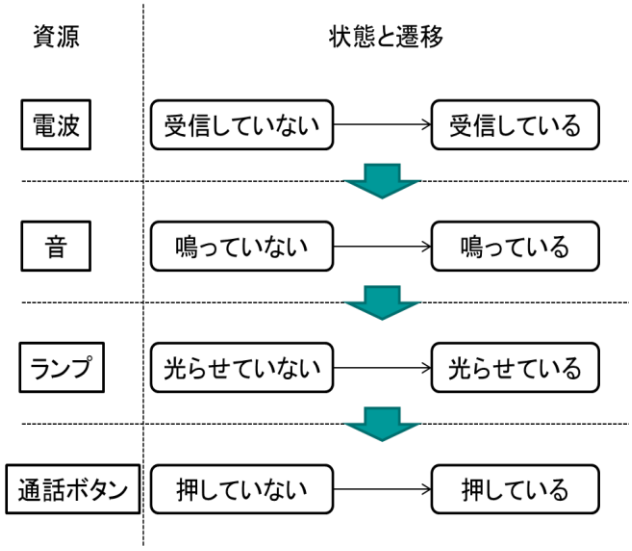

図 6　資源の状態遷移と基本系列の状態遷移
Figure 6　State transition of the basic sequence and state transition of resources.

このユースケースの状態遷移モデルとは別に“競合状態”と競合していない“普通状態”を定める．そしてユースケースの状態遷移モデルにおいて，何か矛盾が発生したら“普通状態”から“競合状態”に遷移するという状態遷移モデルを作成する．本稿における競合とは，2 つ以上のユースケースにおいて同じ資源が“行っている”状態に同じタイミングで遷移したときのことであり，そのような状況になったら普通状態から競合状態に遷移するとなる．

ユースケースの状態遷移モデルと普通状態から競合状態に遷移するという状態遷移モデルを作り終えたら，それらを合成して 1 つのモデルとする．本稿において，複数のユースケースはお互いの状況などは考慮せず，独自に実行される．なので，合成はそれぞれの状態遷移モデルがそれぞれ固有のタイミングで遷移するように，非同期合成で行う．これでモデル検査を行うための状態遷移モデルが完成する．

そして，モデル検査技術を用いて，作成した状態遷移モデルにおいて“競合状態に遷移する”について検証を行う．図 4，図 5 では同じ資源である“ランプ”が“光らせている”状態に両方のユースケースとも遷移し，競合状態となるか検証する．

5. 検証例

提案手法を用いて，ユースケース記述から状態遷移モデルを手動で生成し，検証を行った．本稿ではモデル検査ツール SPIN[13]を用いた．そのため，ユースケース記述から情報を抽出したのち，Promela を用いて状態遷移モデルを

表現する．

Promela を生成する流れとして，ユースケース記述内から資源とその動詞からその資源の状態を定義し，基本系列の状態遷移により資源の状態が遷移するようにする．図 4 と図 5 のユースケース記述においては, 図 7, 図 8 の Promela となる．これを別々のプロセス内に記述する．なお，資源を図 7 では“充電器”と“ランプ”, 図 8 では“電波”，“音”，“ランプ”と少なくして簡略化している．また，普通状態から競合状態へ遷移する状態遷移モデル作成のため，“行っていない”状態を 0 とし，それが“行っている”状態になったら＋1 となるように，変数を定義している．

```
active proctype jyudenn(){
do
 ::jyuden == jyuden_out -> jyuden = jyuden_in; jy = jy + 1
 ::ranpu== ranpu_out -> ranpu = ranpu_in; ra = ra + 1
 ::ranpu == ranpu_in -> jyuden = jyuden_out
   ;ranpu = ranpu_out;jy = 0; ra = 0;
od
}
```

図 7　“充電する”の Promela
Figure 7　Promela of "Charging the battery".

```
active proctype chakushin(){
do
 ::denpa == denpa_out -> denpa = denpa_in; de = de + 1
 ::oto == oto_out -> oto = oto_in; ot = ot + 1
 ::ranpu== ranpu_out -> ranpu = ranpu_in; ra = ra + 1
 ::ranpu == ranpu_in -> denpa = denpa_out;oto = oto_out
   ;ranpu = ranpu_out;de = 0; ot = 0; ra = 0;
od
}
```

図 8　“着信する”の Promela
Figure 8　Promela of "Incoming call".

Promela 記述では，それぞれのユースケース記述を別の active proctype 内で状態が遷移し，do..od 内で繰り返し実行されるようにしている．また，基本系列の最後のプロセスが実行されたら，状態や変数が初期状態に戻るようにしている．

次に普通状態から競合状態に遷移する記述を作る．先述したように，プロセスが実行されたら変数が＋1 となり 0 から 1 となる．もし競合があるとき，同じ資源で“□□している”状態に遷移が起きる．このとき，同じプロセスが二回実行され＋1 が二回行われ，変数は 2 となる．そこで競合状態と普通状態を定義し初期状態を普通状態とする．そして，変数が 2 になったら“競合状態”に遷移するようにする．

普通状態から競合状態に遷移する Promela と図 7，図 8 を合わせて 1 つの記述として, SPIN に入力して検証を行う．“競合状態に遷移する”というプロパティについて，次の LTL 式を用いて検証を行う．

```
! < > state == interaction
```

もし，“競合状態”に遷移する場合はエラーと判定され，判例が表示される．

図 7，図 8 において，ranpu（ランプ）が ranpu_in（光らせる状態）にどちらも遷移した場合，ランプの変数 ra において ra = ra + 1 が二回行われ，ra が 2 となり“競合状態”となる．図 7，図 8 において，SPIN で上記の LTL 式の検証を行ったところエラー判定され，判例のシミュレーションにより競合状態に遷移することを確認できた．

6. 考察

本稿では，モデル検査技術を用いてユースケース記述の基本系列中の機能競合について，発見するための一手法について提案した．また，その手法を例題に適用して SPIN で検証を行った．その結果，ユースケース記述の基本系列における競合を発見することができた．

しかし，提案手法において事前条件や事後条件，代替系列などの記述を考慮した状態遷移モデルの生成や検証を行っていない．もし事前条件を考慮した場合，状態遷移モデルを単に非同期合成することはできず，条件を考慮して合成しなければならない．また，今回の状態遷移は基本系列の動詞から注目し，資源が“行っている”状態か“行っていない”状態の 2 つに分けた．これが“ランプが光っている”状態，“ランプが光っていない”状態を調べる場合には検証できるが，ランプの“明るさ”や温度のような連続量を扱う場合には，それに応じた状態遷移モデルを考案する必要がある．検証のプロパティにおいても，本稿では“競合状態に遷移する”で行ったが，発生してはいけない動作の性質などを用いて検証することも考えられる．その他，今回は 1 つのランプという資源の競合の検証であったが，それ以外の競合や矛盾などを検証できるようにすることも課題として挙げられる．

検証例において今回は 2 つのユースケース記述から検証を行ったが，3 つ以上のユースケースの検証や，複数の競合がある場合はどのような検証結果となるか，携帯電話以外に他のパターンにおけるユースケースの検証などを確認する必要がある．

7. おわりに

本稿では要求定義プロセスの要求分析の工程において，獲得した要求のユースケースにおける系列記述の競合について，モデル検査技術を用いて検証する手法を提案した．また，提案手法を用いて実際にモデル検査ツール SPIN を用いて検証を行った．その結果，ユースケース記述内の競合を発見することができた．

本稿の提案手法により，ユースケース記述の競合を発見し，修正することでそれを仕様化，またはその後の設計に

おける品質の向上につながることができる.

今後の予定として，今回はユースケース記述の基本系列の検証しか行っていないので，事前条件や代替系列などを考慮した検証方法の考案や，今回とは別パターンの事例で検証，資源の競合以外の競合を発見できるようにすることなどが挙げられる.

参考文献

1) 独立行政法人情報処理推進機構 ソフトウェア・エンジニアリング・センター: ソフトウェア開発データ白書 2010-2011, 独立行政法人情報処理推進機構（IPA） (2011)
2) 佐原伸: ～ソフトウェアトラブルを予防する～ 形式手法の技術講座, ソフト・リサーチ・センター(2008)
3) Stuart R. Faulk.: Software Requirements:A Tutorial, Dorfman, M. and Thayer ,R., IEEE Software Engineering, pp82-103 (1996).
4) 石川冬樹, 荒木啓次郎: VDM++による形式手法記述, 近代科学社 (2011)
5) 妻木俊彦, 白銀純子, 大西淳: 要求工学概論, 近代科学社 (2009)
6) 一般社団法人情報サービス産業協会 REBOK 企画 WG: 要求工学知識体系第 1 版, 近代科学社 (2011)
7) Yoshiyuki Shinkawa.: Model Checking for UML Use Cases, Software Engineering Research, Management and Applications Studies in Computational Intelligence, Volume 150, pp233-246 (2008)
8) Ksenia Ryndina, Pieter S. Kritzinger.: Analysis of structured use case models through model checking, South African Computer Journal, Vol.35, pp84-96. (2005)
9) M. Calder, A. Miller.: Using SPIN for Feature Interaction Analysis - A Case Study, Lecture Notes in Computer Science 2057, pp143-162 (2001)
10) Matsuo Takafumi.: Feature Interaction Verification of Telecommunication Services and Home Network Services Using Model Checking, Osaka University doctoral dissertation (2009).
11) 井垣宏, 中村匡秀, 松本健一: 家電機器連携サービスにおけるサービス競合検出システム, 電子情報通信学会技術研究報告. DE, データ工学 104(344), pp11-16 (2004)
12) アリスター・コーバーン: ユースケース実践ガイド, 翔泳社 (2001)
13) G. J. Holzmann.: The model checker SPIN, IEEE Trans. Software Eng., Vol. 23, No.5, pp279–295 (1997)