Vol.2010-SE-167 No.32
2010/3/19

形式手法を用いたシステム設計検証技術

市原利浩[†]　上野浩一郎[†]　磯田 誠[†]　大貫智洋[†]

ソフトウェア・システムの大規模化・高機能化に伴い，従来の人手によるレビューやテストの実践だけでは，設計で混入した不具合を完全に除去することが難しくなっている．そのため，近年，形式手法の一つであるモデル検査による検証技術が注目を集めている．しかし，モデル検査技術を広く展開するには，現場の開発者にとってモデル検査ツールに与える仕様モデルの定義や出力される結果の解析が困難という課題がある．
これに対して我々は，特定の分野に特化した記法を用いて仕様モデルを定義することで，仕様モデル定義の負荷の軽減を狙うこと，モデル検査器が出力した検証結果をモデル変換することで，開発者にとって解析しやすいモデルの出力を実施するモデル検査器フロントエンドを適用することを検討している．本稿では，このアプローチによるシステム設計検証技術について報告する

Using Formal Methods for System Design Verification Technology

Toshihiro Ichihara[†]　and Ueno Koichiro[†]
and Makoto Isoda[†]　and Tomohiro Onuki[†]

While sof tware systems b ecome l arger an d more c omplex, the ve rification of the software s ystems' spec ification be comes h arder. In r ecent, th erefore, model checking technology is foc used as a method to ex tract defects of design ed sp ecifications in su ch large software intensive systems. However, for those practical software developers, it is hard to rewirte their specifications for model checking tools, and also hard to understand the output of the checkers.
In this pap er, we pr opose an approach to overco me these issues by co mbining 2 approaches: Do main-oriented modeling la nguage, and Model C onversion between ordinal UML and M odel ch ecking tools . Domain-oriented modeling langu age he lps developers to specify their systems more easily, and Model Conversion mechanism helps those d evelopers to under stand th e r esult of those m odel checking tools on th eir own domain-oriented modeling languages. We are now developing the method and supporting tools for our model checking method.

1. はじめに

近年，社会基盤を支える分野にソフトウェア・システムが広く適用されるに伴い，ソフトウェア・システムの不具合が社会的な問題として取り上げられることが増えている．一方で，ソフトウェア・システムの大規模化・高機能化は，設計レビューやテストと言った従来からの手法で，これらの不具合を完全に排除することを難しくしている．特に，高信頼・高可用性要求を目的通りに実現していることを，有限時間で実施するテストの中で全て検出することは難しく，またテストフェーズでこれらの品質要求に関わる不具合を検出した場合も，その不具合修正に要するコストはしばしば大きなものになることが多い．

こうした中，近年，形式手法による仕様検証技術が注目を集めている．例えば，経済産業省が公表した「情報システムの信頼性向上に関するガイドライン」[1]で，形式手法の適用が推奨されている．また，宇宙航空研究開発機構による宇宙機の制御ソフトウェア開発への適用も報告されている[2]. 形式手法の中でも特に脚光を浴びているのがモデル検査手法である．これは，モデル検査が，数学の専門的な知識が余りない技術者でも，ソフトウェア開発の中で利用することが可能であること[3]，モデル検査を実装した様々なツール[4]が既に開発されていることが理由と考えられる．

しかし，これらのモデル検査手法も，まだ広くソフトウェア・システム開発現場へ導入されているとは言えない．この要因は，いくつか考えられるが，我々が考える主な要因は，モデル検査ツールに対する入力となる仕様モデル定義が困難であること，また出力される反例の意味するところを理解することが難しいからである．

我々は，これらの課題を解決する方策として，仕様モデルの定義を簡便化するための記法をメタモデルとして定義するアプローチを採用し，また，この記法とモデル変換技術を組み合わせることにより，反例解析を容易に行えるツールを開発している．

本稿では，我々の提案する手法の概要を紹介する．なお，モデル検証ツールとしては，SPIN（Simple Promela INterpreter）[5]を用いる．以下，２章ではモデル検査によるシステム設計検証の実用化の課題を示す．３章では，我々が提案するシステム設計検証手法について示す．４章では，サンプル題材を用いた本手法の適用事例を示す．５章では，本手法の評価結果を示す．６章では，本稿のまとめを示す．

2. モデル検査によるシステム設計検証

2.1 形式手法とモデル検査

形式手法とは，計算機科学における論理学や離散数学に基づいて，システムの仕様を記述し検証する技術の総称である．形式手法の代表的なものは，演繹手法とモデル

[†] 三菱電機（株）　情報技術総合研究所
Inform ation Technology R&D Center,　Mitsubishi Electric Corporation

検査手法がある．演繹手法は，検証したい性質が仕様によって満たされることを論理的形式に則って証明する方法であり，人手による対話的な証明のガイドを必要とする．一方，モデル検査とは，振舞いの仕様をモデルとして定義し，その振舞いに対して保障すべき性質を命題として与えると，検査対象の状態を網羅的に探索し，各動作パターンにおいて命題の充足関係を調べることにより，システム動作を検証する方法である．モデル検査は，網羅的検証をツールにより自動化できる点が実用化するうえで大きなメリットとなっている[3].

2.2 モデル検査によるシステム設計検証

モデル検査によるシステム設計検証は，システム開発のライフサイクルプロセスの中で，設計段階にて仕様のバグを検出し，上流バグが下流に流出するのを未然に防ぐための手法である．この作業の位置づけを図 1 に示す．

モデル検査を行う場合，開発者はシステム仕様書から検証対象とする部分の仕様を抽出し，状態遷移図を作成する．その仕様モデルを基に検査プログラムを生成してモデル検証器に入力する．モデル検証器が何も出力しない場合は，仕様モデルに不具合が無いため，検証プロセスはそのまま終わる．仕様モデルに不具合がある場合はモデル検証器が反例を出力するため，反例の意味解析を行い，該当箇所の仕様書を修正する．この検証プロセスを，反例が検出されなくなるまで，繰り返し実行する．

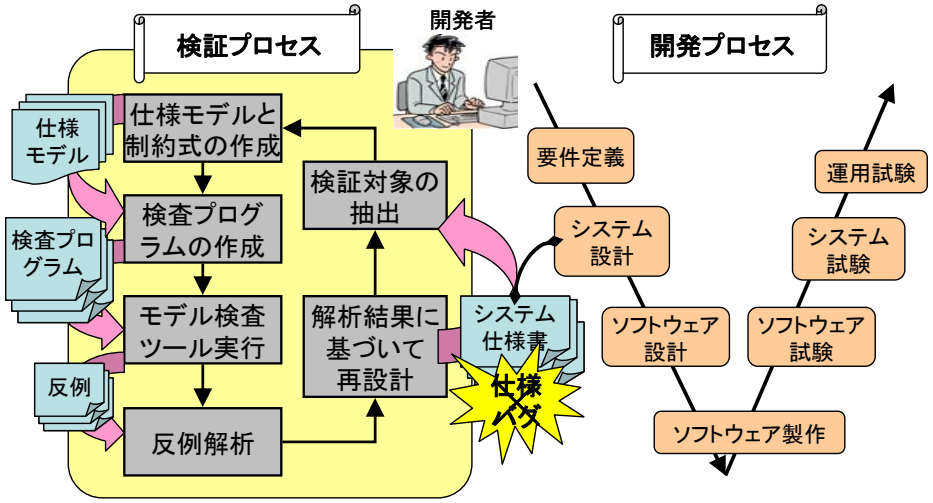

図 1　モデル検査によるシステム設計検証

2.3 モデル検査の現場適用に向けた課題

2.1 節で述べたように，モデル検査技術は，入力として与える仕様モデルに対し，状態遷移空間を網羅的に探索するため，微妙なタイミングを伴うなどの複雑な仕様に対して強力な手法として知られており，国内での適用事例も報告されている[2]．しかし，モデル検査技術には，下記 (1)~ (4)のような課題があるため，開発現場への浸透が難しい[6].

(1) **仕様モデルの定義**

モデル検査を行うには，開発対象ソフトウェアの振舞い（動作仕様）を専用の仕様記述言語を使って，厳密にコード化することが求められる．しかし，一般の開発者が新たな仕様記述言語を習得してコードを記述することは大きな負担である．近年では，UML（Unified Model Language）[7]などで定義した仕様モデルからモデル検査用の仕様記述言語を生成するツールも提唱されている[8]．しかし，UML は汎用的過ぎるため，ある特定分野において部品化できる共通的な記述をその都度最初から記述しなければならなく，開発者による定義を軽減できる余地がある．

(2) **反例モデルの意味解析**

モデル検証器（SPIN）は，与えた制約式を満たさない全ての動作シナリオを反例として出力する．XSPIN などモデル検証器が出力する検証結果（反例）をビジュアルに表示するツール[9]も存在するが，これらの検証結果には検査プログラムのコードや反例として意味を持たない情報が含まれる．結果的に，現状のモデル検証器が出力する反例は，開発者が理解できる仕様モデルとの隔たりが大きく，モデル検査に精通しない開発者にとって反例の解析が困難となっている．

(3) **検証項目の選定と制約式の定義**

モデル検査を行うには，振舞いの仕様と満たすべき性質を定義した制約式を与える必要がある．例えば，モデル検査ツール SPIN の場合，検証可能な検証項目は，①到達可能性(初期状態から特定状態に到達できる)，②安全性(デッドロックや無限ループがない)，③活性(いつか必ず特定状況が発生する)，④公平性(各々の並行処理が必ず進む)の 4 種類である．これらの検証項目は，検査を行う設計者が検査内容に応じて選定して，それに応じた制約式を記述する必要がある．すなわち，モデル検査の実施者が，適切な検証項目の選定や制約式の定義のための知識を持つことが前提となる．

(4) **状態爆発による無回答**

仕様モデルや検証項目が複雑になると，モデル検証器が探索する状態遷移パスが爆発的に増大し，検証結果が即座に得られない状態になりやすい．検証時間が膨大となる場合は，仕様モデルの見直しや検証項目の再設定を行うとされる．しかし，モデル検証器には検証処理の進行度を伝える機能がないため，検証処理を即座に得られない場合，処理を打ち切るかどうかの判断が難しい．

3. システム設計検証手法

今回我々は，2.3節における課題の中で， (1)(2)について取り組んだ．本章では，我々が取り組んだモデル検査によるシステム設計検証手法について述べる．

3.1 我々のアプローチ

我々が提案するシステム設計手法の概要について図 3 に示す．また，対比として従来手法の概要を図 3 に示す．

我々の提案する検証手法では，モデル検査，モデル駆動，ドメイン特化言語の考え方を組み合わせた手法である．本手法の特徴を以下に示す．なお，以降，モデル検証器が出力するテキストベースの反例を検証結果と呼び，この検証結果をモデル変換して開発者に出力するモデルを反例モデルと呼ぶ．

- 入力する仕様モデルの記法を開発者が担当する分野に特化した記法によって定義する．これにより，冗長的な記述を省き，負荷の削減を狙う．以降，このような特定分野向けの記法をドメイン特化記法と呼ぶ．
- モデル逆変換により，検証結果を開発者が解釈可能なレベルのモデルに戻すことで，反例モデルの解析を行い易くする．
- 検証目的に応じて適切なモデル検証器の選択が必要である．分野によって異なるドメイン特化記法及び，検証目的に応じた検証器をそれぞれプラグイン可能とする仕組みを導入するために，従来のモデル変換の間に，汎用モデル（UML モデル）への変換を行う．

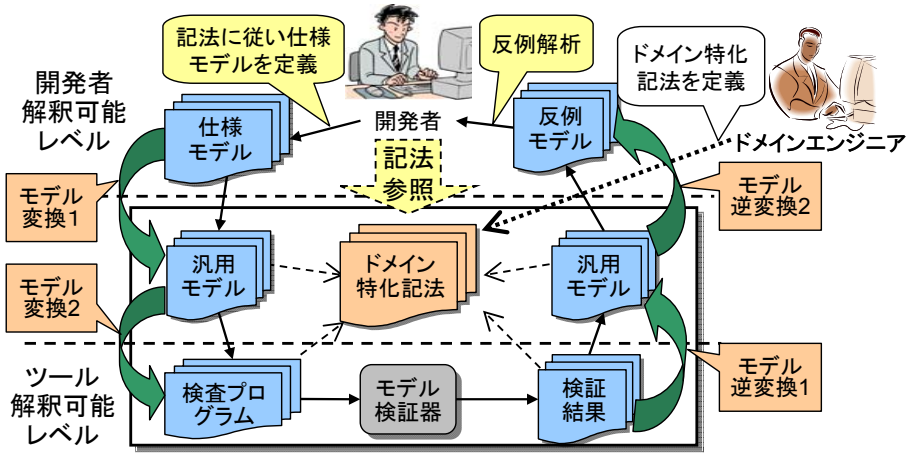

図 2 我々の検証手法

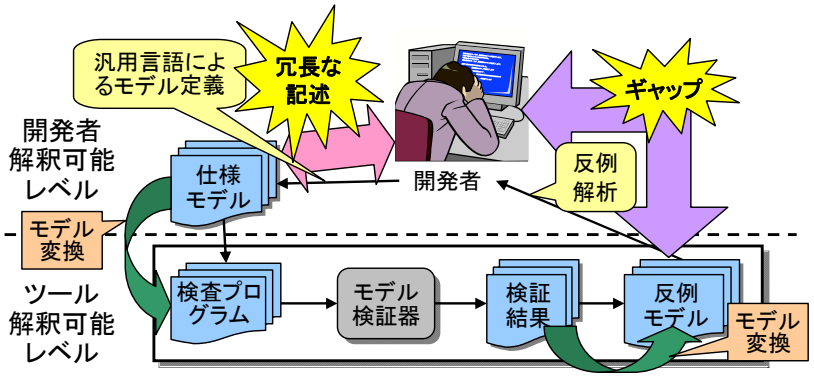

図 3 従来の検証手法

3.2 ドメイン特化記法

ドメイン特化記法は，UML を拡張し，特定分野における共通的な記述を部品化した記法であり，ドメインエンジニアが予め定義する． 開発者はこのドメイン特化記法に従って仕様モデルを定義することで，同様記述の繰り返す手間を省き，仕様モデル定義の負荷軽減を狙う．

ドメイン特化記法のメタモデルを図 4 に示す．ドメイン特化記法は，SE 記述要素とドメイン共通要素，検証結果要素の 3 つの要素から成る．各要素についての説明を以下に示す．

(1) SE 記述要素

開発者が仕様モデルを記述する際に使用する言語要素である．この要素は，汎用モデルの言語要素（UML）と，ある特定分野において共通的な記述を部品化した言語要素から成る．SE 記述要素は，仕様モデルの構成を定義する SE 記述構成要素と，振舞いを定義する SE 記述振舞い要素から成る．

(2) ドメイン共通要素

SE 記述要素において部品化した要素を，分解して汎用モデル言語（UML）で定義した要素であり，開発者が仕様モデルを記述する際には用いない要素である．仕様モデルから汎用モデルに変換する際に，部品化した要素を本要素に置換する．ドメイン共通要素も，構成を定義するドメイン共通構成要素と，振舞いを定義するドメイン共通振舞い要素から成る．

(3) 検証結果要素

反例モデルの記述のみに使用する要素である．なお，反例モデルは，本要素と SE 記述要素，ドメイン共通要素により構成する．

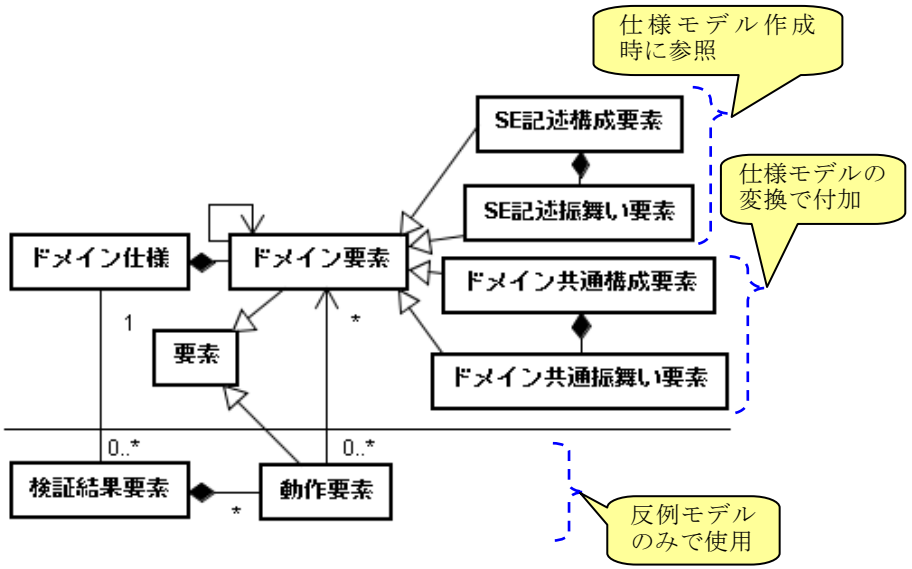

図 4　ドメイン特化記法のメタモデル

3.3　モデル変換

本節では，我々の手法で採用しているモデル変換について説明する．なお，本手法のモデル変換で利用するマッピングルールは，ドメインエンジニアがドメイン特化記法の定義に伴い作成するものである．一連のモデル変換で同一のドメイン特化記法に則ったマッピングルールによる変換を行うことで，入力した仕様モデルの要素と出力される反例モデルの要素が対応する．

(1)　モデル変換１

まず，モデル変換により仕様モデルから汎用モデルを生成する．モデル変換の手順は，図 5 に示す通り，マッピングルールに従って仕様モデルの各要素を汎用モデルの要素に置換する．このマッピングルールは，置換する仕様モデルのドメイン特化記法要素と汎用モデル要素の対応表である．次に，置換した汎用モデルに対し，ドメイン共通要素の付加を行う．具体的には，ドメイン特化記法において，ドメイン共通要素と直接関連する SE 記述要素を抽出する．次に，抽出した SE 記述要素を基にドメイン特化記法からドメイン共通要素を抽出し，汎用モデルに付加する．

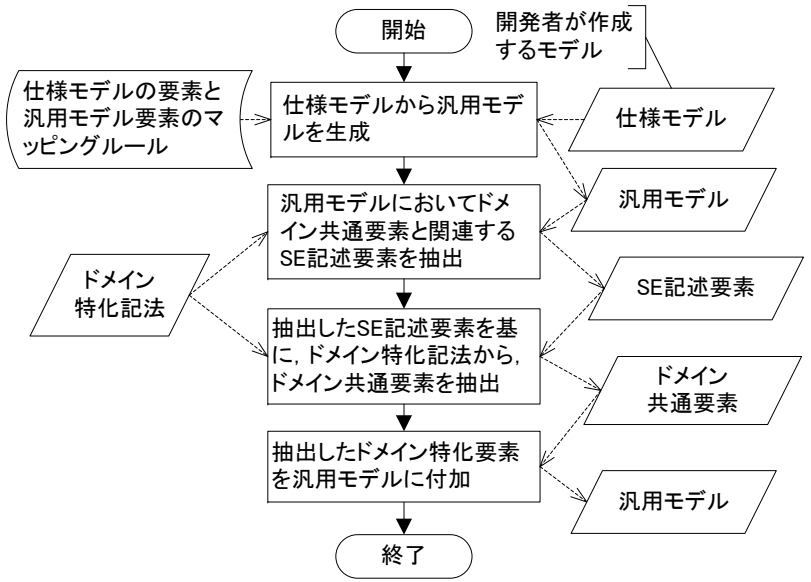

図 5　モデル変換１（仕様モデルの変換）のアルゴリズム

(2)　モデル変換２（検査プログラムの生成）

モデル変換により汎用モデルから検査プログラムを生成する．SPIN の検査プログラムは，Promela（PROtocol/PROcess MEtaLAnguage）と呼ばれる仕様記述言語により記述される．この汎用モデルから検査プログラムへの変換ルールは，既存の変換ルール[3]を用いる．

(3)　モデル逆変換１

モデル変換によりテキストベースの検証結果から汎用モデルを生成する．モデル変換の手順は，図 6 に示す通り，反例モデル（例えば，シーケンスモデル）に必要な要素を予め定義しておき，その要素（例えば，シーケンス図のライフライン名やメッセージ名など）を，検証結果を字句解析して抽出する．

次に，マッピングルールに従い，抽出したデータから汎用モデルを生成する．このマッピングルールは，検証結果要素とドメイン特化記法要素の対応表である．

さらに，生成した汎用モデルに不足する要素の付加を行う．具体的には，生成した汎用モデルと予め定義された反例モデルに必要な要素の比較を行い，汎用モデルの中で不足している要素を特定する．ドメイン特化記法を参照し，この不足要素を抽出し，汎用モデルに付加する．

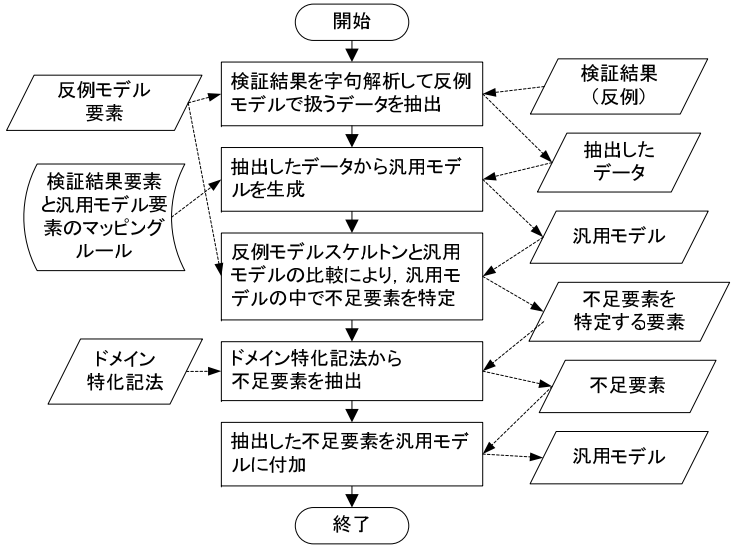

図 6　モデル逆変換 1（検証結果の変換）のアルゴリズム

(4) **モデル逆変換 2**

モデル変換により，汎用モデルから反例モデルを生成する．モデル変換は，汎用モデル要素と反例モデル要素のマッピングルールに従って行う．

4. サンプル題材の適用事例

本章では，2 名の作業者がプリンターとスキャナーの資源を使用する仕様をサンプル題材として，3 章で説明した性能検証手法の適用例を示す．本例題の仕様を下記とする．

- 作業者 A は，プリンター，スキャナーの順に資源を獲得し，2 つの資源を連続して利用した後，プリンター，スキャナーの順に資源を解放する．
- 作業者 B は，スキャナー，プリンターの順に資源を獲得し，2 つの資源を連続して利用した後，プリンター，スキャナーの順に資源を解放する．
- 一方が獲得しようとした資源を既に他方が獲得している場合は，その資源がリリースされるまで待つ．

4.1 ドメイン特化記法

本例題のドメインを共有資源獲得とし，共有資源獲得仕様のドメイン特化記法を図 7，図 8 に示す．このドメイン特化記法における SE 記述要素，ドメイン共通要素，検証結果要素についての説明を (1)～ (3)に示す．

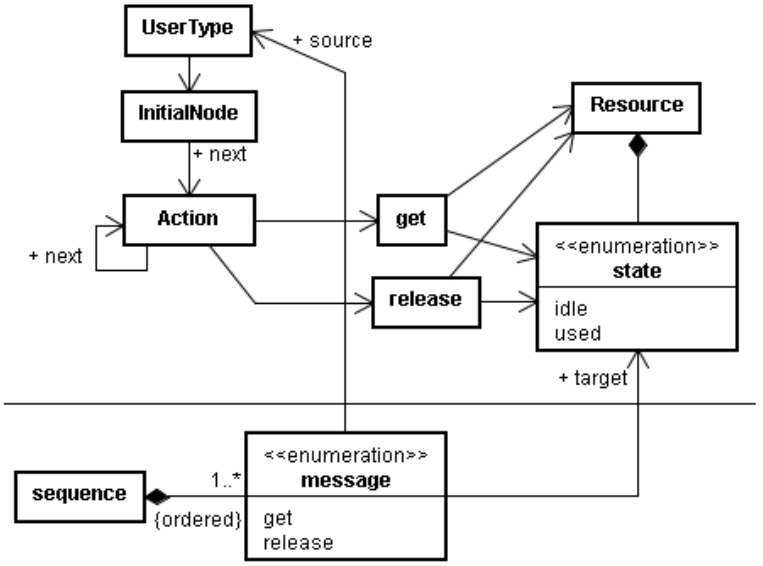

図 7　共有資源獲得仕様のドメイン特化記法 1

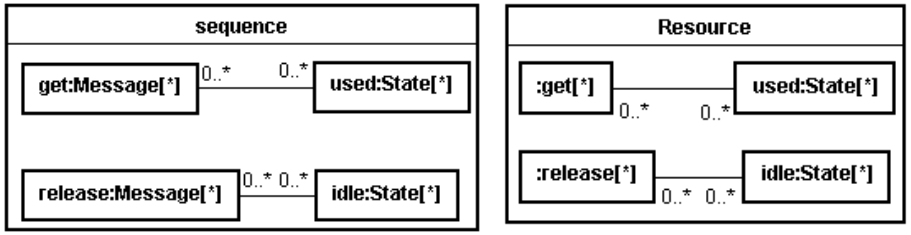

図 8　共有資源獲得仕様のドメイン特化記法 2

(1) **SE 記述要素**

SE 記述構成要素を“作業者”と“資源”とし，SE 記述振舞い要素をプリンターとスキャナーの“獲得”及び“解放”とする．図 7 の記法において，UserType クラス，Resource クラスが SE 記述構成要素，Action クラスと get オブジェクト，release オブジェクトが SE 記述振舞い要素である．なお，本記法による仕様モデルは，アクティビティ図による作業者の振舞いの定義と，オブジェクト図による資源構成の定義を合わせた図を用いる．

(2) **ドメイン共通要素**

共有資源獲得のドメインにおいて，作業者の“獲得”や“解放”の振舞いに応じて資源のステータスが“使用中”の状態や“空き”の状態に変わることは，全ての資源に関して共通していると考える．そこで，プリンターとスキャナーなどの，資源の利用状況（ステータス）の定義に関わる要素をドメイン共通要素とする．ドメイン共通構成要素が図 7 の state オブジェクトであり，ドメイン共通振舞い要素が図 8 のコンポジット構造図である．共有資源獲得のドメインでは，排他制御を定義する仕様であると考え，資源の状態の定義に，図 8 に示すコンポジット構造図を用いている．

(3) **検証結果要素**

今回我々は反例モデルを，シーケンス図に“ステータス”の要素を追加したモデルで表すこととした．そこで，シーケンス図の要素と“ステータス”要素から，仕様モデルの定義で扱う要素を省いた要素を検証結果要素とする．図 7 において，message オブジェクトが検証結果要素である．

4.2 仕様モデルの定義とモデル変換，反例モデル

(1) **仕様モデル**

4.1 節で定義した記法を用いて定義した本例題の仕様モデル（一部）を図 9 に示す．本例題における仕様モデルは，作業者 A と作業者 B の構成をクラス図として定義し，作業者 A と作業者 B の振舞い及びプリンター，スキャナーの構成をアクティビティ図とオブジェクト図を合わせた図として定義する．

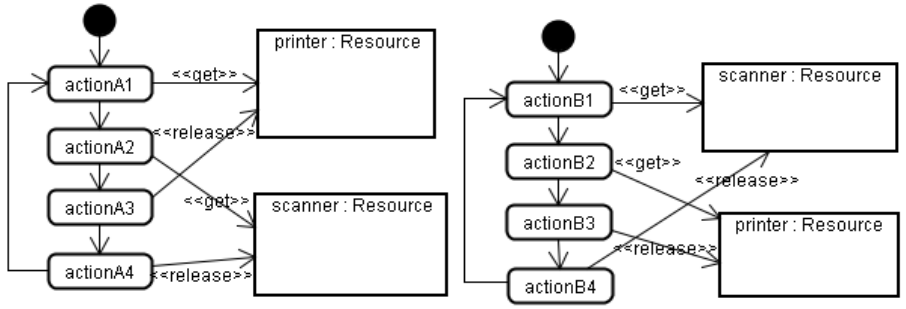

図 9　サンプル題材の仕様モデル

(2) **仕様モデルに対するモデル変換**

本例題の仕様モデルをマッピングルールに従って汎用モデルへ置換し，ドメイン共通要素を付加した汎用モデルの例を図 10 に示す．付加するドメイン共通要素は，プリンターとスキャナーに関するステータス定義の要素であり，具体的には，“used”と“idle”の state オブジェクトある．この要素は，作業者の振舞い（“get”と“release”）によって特定できる．“get”の振舞いの場合，プリンターやスキャナーのステータスは“used”であり，“release”の場合は“idle”となる．

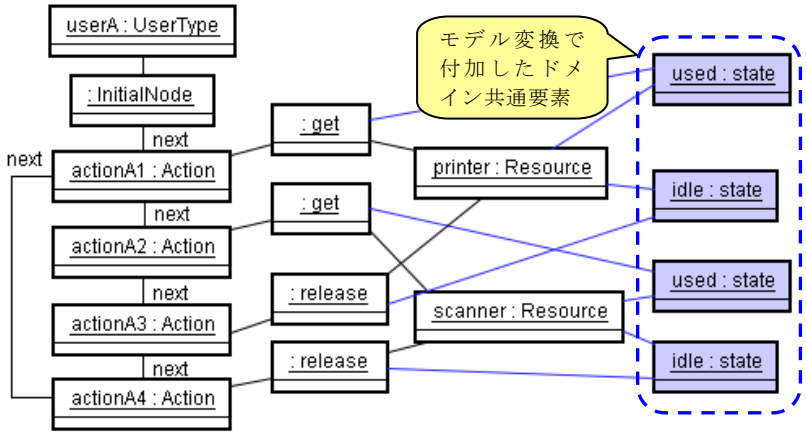

図 10　仕様モデルから変換した汎用モデルの例

(3) **検証結果に対するモデル変換**

検証器が出力した検証結果をマッピングルールに従って汎用モデルへ置換し，不足する要素を追加したモデルの例を図 11 に示す．SPIN が出力する検証結果には状態の要素（インスタンス名）がないため，ドメイン特化記法を参照し，不足する要素を追加する．本例題では，資源の状態名として state オブジェクトのインスタンス名が不足する．“get”の振舞いに対する資源のステータスは“used”であることから，state オブジェクトのインスタンス名として“used”を追加する．

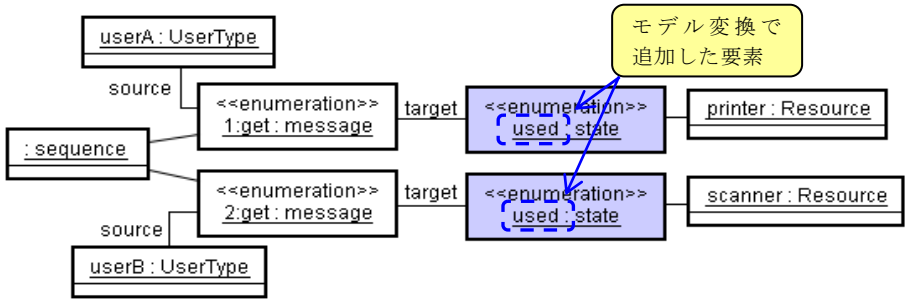

図 11　検証結果から変換した汎用モデルの例

(4) **反例モデル**

図 11 に示す汎用モデルをモデル変換して，出力した反例モデルを図 12 に示す．本例題では，開発者 A と B がそれぞれ別々の資源を同時に獲得し，お互いに，もう一方の資源が解放されるのを待つ状態となり，デッドロックが生じる反例が数パターン出力される．今回の題材では，検証結果のモデル変換でステータスの要素を付加しており，反例モデルのシーケンス図に状態不変式を用いて表示する．

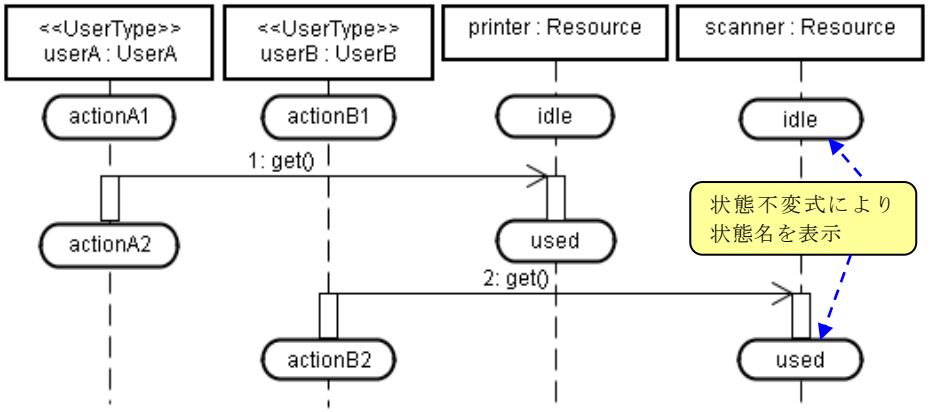

図 12　本手法により出力した反例モデル

4.3 本手法による仕様モデル定義と検証結果

本節では，本手法による仕様モデルと検証結果を，従来手法のモデルと比較する．

(1) **仕様モデルの比較**

従来手法によって定義した本例題の仕様モデル（一部）を図 13 に示す．汎用的な UML で仕様モデルを定義する場合，振舞いが異なる構成要素ごと，それぞれの状態遷移を定義する必要がある．本例題の場合，作業者 A，作業者 B，資源（プリンター，スキャナー）に対して，状態遷移を定義する．

これに対し本手法による仕様モデル（一部）を図 9 に示す．本手法では，共有資源獲得のドメインにおいては，資源の状態遷移は共通的な要素であると考え，ドメイン共通要素としているため，開発者は資源のステータスに関する定義は不要である．

また，従来手法では，各構成要素間のイベントの授受は，例えば Printer.Get のように，イベントを投げる側は“*イベントの送信先*. *イベント名*”の記述形式を状態遷移内に定義する必要があり，文字を主体とした定義を行わなければならない．これに対して本手法では，制御フロー（矢印）による視覚的な定義を主体としている．

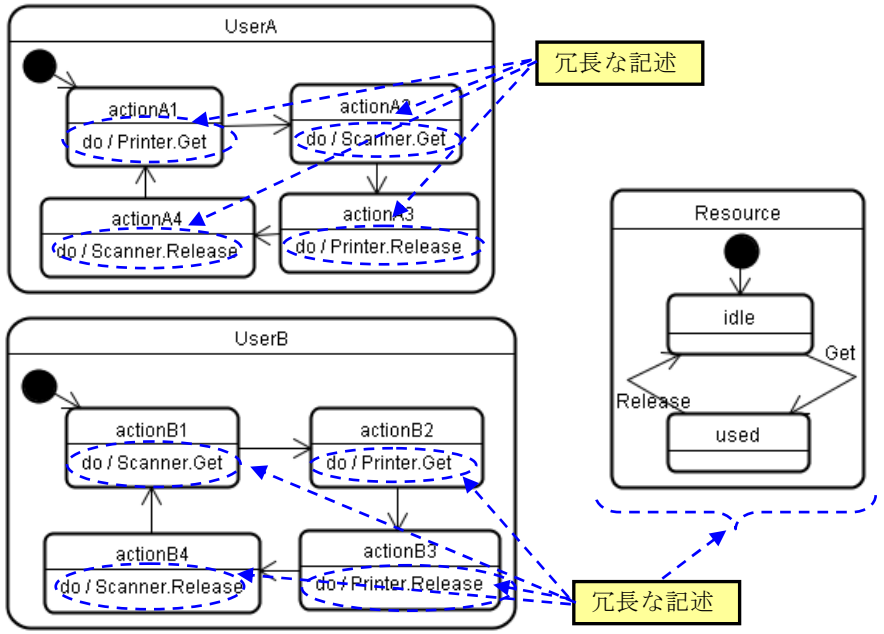

図 13　従来手法による仕様モデル

(2) **反例モデルの比較**

SPIN のフロントエンドツールである XSPIN により出力されるモデルを従来手法による反例モデルとして比較する．本例題の反例モデルの一つを図 14 に示す．SPIN が出力する検証結果には，各構成要素のステータス情報がないため，検証結果をそのまま出力する XSPIN の反例モデルには，ステータスの要素がない．これに対し本手法による反例モデルの一つを図 12 に示す．本手法では，検証結果からモデル変換する際に，反例モデルとして不足する要素を追加することで，開発者に出力する反例モデルにステータスの要素を含めている．

また，従来手法では，検証器内部で扱うデータがそのままの形式で検証結果に含まれるため，反例として意味を持たないライフラインや“scanner ch! Get ”などの Promela コード（仕様記述言語）が表示される．これに対し本手法では，不要な要素は，モデル変換時に切り捨てる方式を取った．更に，クラス名やステレオタイプ名など，仕様モデルに用いた要素と対応させることで，入力モデルと反例モデルの一貫性を保っている．

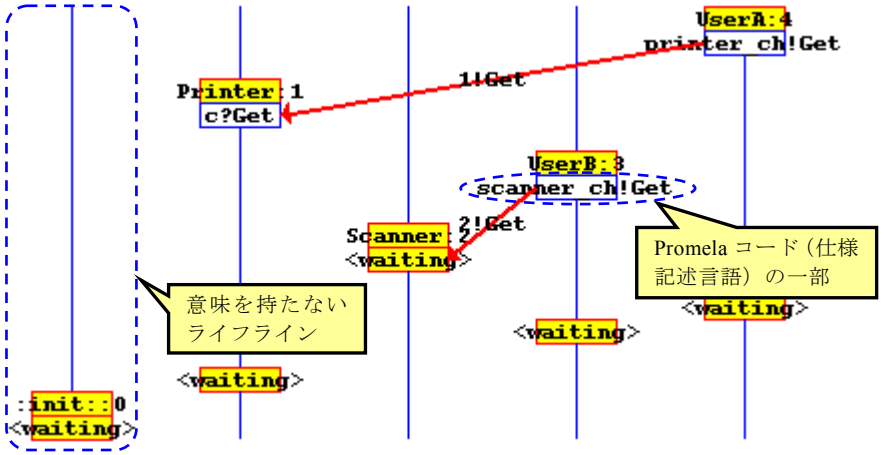

図 14　従来の反例モデル（XSPIN による出力例）

5. システム設計手法の評価

本章では，今回我々が提案するドメイン特化記法とモデル変換の評価結果を示す．

(1) ドメイン特化記法

ドメイン特化記法は，開発者による定義が必要な要素と開発者による定義が不要な要素（ドメイン共通要素），反例出力用の要素から成ると考え，ドメイン特化メタモデルを，SE 記述要素，ドメイン共通要素，検証結果要素から成るモデルとして定義した．開発者による仕様モデル定義では，この SE 記述要素のみを用いることとし，ドメイン内で共通的な要素をモデル変換時に付加することで，冗長的な記述を減らすことができた．

今後は，実システムの開発において，本手法を適用し，我々が提案するドメイン特化メタモデルのフィージビリティと実開発における効果の確認に取り組む予定である．また，必要に応じてドメイン特化メタモデルの拡張を検討する．

(2) モデル変換

仕様モデル定義，及び仕様モデルの入力から反例モデルの出力までの一連のモデル変換にて，同一ドメインにおいて同一の記法を参照してモデル変換する．これにより，仕様モデルと反例モデルの要素（クラス名，インスタンス名，ステレオタイプ名など）が一致することを確認した．さらに，モデル検証器から出力される検証結果のモデル変換時に，反例モデルに不要な要素は切り落として変換することで，意味のない要素が反例モデルに出力されないことも確認した．これらにより，反例モデルの解析が容易になったと考える．

今回シーケンス図を拡張した記法によって反例モデルを出力した．今後は，状態遷移図やアクティビティ図による反例モデルの出力を検討する．また，SPIN 以外のモデル検証器を用いた本手法の適用も検討する．

6. おわりに

本稿では，我々が取り組んでいるシステム設計検証のアプローチについて，モデル検査技術とモデル駆動技術，及びドメイン特化記法を組み合わせた検証手法を提案した．具体的には，ドメインエンジニアによって定義されたドメイン特化記法とマッピングルールを用いて，仕様モデルから２段階のモデル変換による検査プログラムの生成，さらに，検証器が出力した検証結果から２段階のモデル逆変換による反例モデルの出力を行う手法である．この手法により，プリンター・スキャナーの共有資源獲得仕様をサンプル題材として，仕様モデルの定義と反例モデル解析の容易化が図れたと考える．

この評価結果を踏まえ，我々が提案したシステム設計検証手法を実用化するために，今後新規システム開発の現場に適用し，本手法の有効性の検証に取り組む予定である．また，今回取り上げなかった検証項目の選定と制約式の定義，及び状態爆発による無回答の課題にも取り組む予定である．

参考文献

1)　経済産業省：情報システムの信頼性向上に関するガイドライン第 2 版（2009）
2)　宇宙航空研究開発機構：「ソフトウェア独立検証と有効性確認」技術の研究 http://stage.tksc.jaxa.jp/jxithp/
3)　吉岡信和，青木利晃，田原康之，SPIN による設計モデル検証，近代科学社(2008)
4)　B.Berard，M. Bidoit，A. Finkel，F. Laroussinie，A. Petit，L. Petrucci，Ph. Schnoebelen with P. McKenzie，Systems and Software Verification : Model-Checking Techniquesand Tools，Springer，2001
5)　Gerard J. Holzmann,“The Model Checker SPIN”, IEEE Trans. Soft. Engin., Vol.23, No.5, pp.279–295, 1997.
6)　篠崎 孝一，太田 弘，早水 公二，星野 光勇，今村 哲典，吉田 雅昭，モデル検査の実用化課題と支援ソフトウェアの開発，第三回 システム検証の科学技術シンポジウム，November 2006
7)　OMG Unified Modeling LanguageTM (OMG UML)， Su perstructure， V ersion 2.2， OM G formal/2009-02-02， Object Management Group (2009)
8)　A.Knapp,S.Merz and C.Rauh:”Model Checking Timed UML State Machines and Collaborations,”Formal Techniques in Real-Time and Fault-Tolerant System,pp.395-414,Springer-Verlag,2002.
9)　M.Ben-Ari: Principles of the Spin Model Checker,Springer,2008.