コンテキストに注目したゴールモデルのモデル検査に関する研究

本田 大雅[1]　小島 英春[1]　中川 博之[1]　土屋 達弘[1]

概要：ゴール指向要求モデルは，ソフトウェア開発において顧客の要求を分析して仕様化するのに用いられる．そのゴールモデルに対して矛盾はないか，必要な要求は満たされているかなどを開発の初期段階で検証することで，開発工程の無駄な手戻りを防ぐことが期待できる．本研究では，ゴールモデルからステートマシン図に変換し，それをモデル検査器 SPIN で扱うことのできるモデルに変換することでモデル検査を行う．モデル検査を行うにあたっては，ゴールモデルで表現されたシステムのエージェントやアクタを 1 つのプロセスとして扱う．検査対象のエージェントやアクタが増加すれば，それに伴ってプロセス数も増加するため，状態数は爆発的に増加する．そのような状態爆発を緩和するために，対象とするシステムの周囲の環境を考慮して検証する機能を限定する．モデル検査では全ての状態を網羅し特性を検証するが，ある環境では必要とはされない機能が存在することが考えられ，それに対応する状態については検証する必要はない．そこで，システムの周囲の環境を表現するコンテキストに注目し，それを用いて機能の限定を行う．本研究では，ゴールモデルからモデル検査器用のモデルを作成し，モデル検査器 SPIN を用いてコンテキストを適用した場合としない場合について検証を行う．状態数，実行時間と利用メモリ量を比較し，全ての項目においてコンテキストを適用した場合はしない場合に比べ削減できることが確認できた．

A Study of Model checking for Contextual Goal Models

1. はじめに

システム開発工程は一般に，企画，要件定義，設計，開発，テスト，運用・保守といった流れで進む．その要件定義において，システムの要求仕様を開発の初期段階で確定しておくことは重要である．そこで用いられる手法の 1 つがゴール指向要求分析手法である．この手法を用いて要求を抽出・分析し，文書化して仕様化を行い，要求の一貫性や完全性を調べて，開発の全ての工程において要求の管理を続ける．顧客が望む要求を正確に定義し仕様化することで，要求仕様が不完全や曖昧なため顧客が望むソフトウェアを実現できなかったり，要求仕様通りであるが顧客の望んだものではなかったというようなソフトウェア開発の失敗を防ぐ．また，要求の評価や管理を行うことで，開発途中で要求定義が不十分なために開発作業をやり直すといった開発工程の無駄な手戻りを防ぐことを避けることができる．ゴール指向要求分析で用いられるゴールモデルとして KAOS[1]，i*[2]，NFR[3]，Tropos[4] が挙げられる．作成されたゴールモデルを精査して，対象とするシステムがゴールを達成できるか，ゴールの矛盾がないかを様々な方法で調べることによって，要求仕様の検証や評価を行う．しかし，このモデルは抽象的なものであるため，形式的に検査を行うことができない．そこで，ゴールモデルからモデル検査器で用いるモデルへと変換することで，モデルの検査を形式的に自動で行うことを考える．

本研究では，モデル検査器として SPIN を用いる．このためには，まずは，ゴールモデルを SPIN で用いることができるモデルに変換することが必要であるため，既存手法のゴールモデルからステートマシン図に変換する手法を用いる．ステートマシン図はそのシステムの動作を状態遷移として表現しているため，変換されたステートマシン図を用いることで，モデル検査器で扱うことのできるモデルを作成することが可能である．

次に，SPIN ではプロセスが並行に動作する際にシステムとして特性を満たすかどうかを検証することが可能である．そこで，ゴールモデルからどのようにプロセスを抽出

[1] 大阪大学大学院情報科学研究科
大阪府吹田市山田丘 1 − 5

するかが必要となる．通常，ゴールモデルには，システムが満たすべきゴールやあるゴールを達成するために達成すべきサブゴールが記述されている．さらに，それらのゴールを達成するために必要な操作やタスクも定義されている．ゴールモデルに示されているゴール，操作やタスクはシステム内の何が責任を持って行うか，誰が責任を持って行うかが示されている．この何が，誰がを 1 つのプロセスとして扱うことで，SPIN のモデルに変換することが可能となる．

しかし，これまでのゴールモデルからステートマシン図への変換手法 [5][6][7][8] では，ゴールモデルを 1 つのステートマシン図へ変換しているため，そのままでは，複数のプロセスを生成することができない．そこで，本研究では，ゴールモデルに現れる，何が，誰がを表現するエージェントやアクタを 1 つのプロセスとして扱うことで SPIN で用いることのできるモデルを作成する．また，ゴールモデルでは，1 つのゴールを満たす前に満たすべきゴールが存在し，各ゴールにはそれを満たすために必要な振舞いが記述されている．その振舞いが複数ある場合にエージェントやアクタがどのような順序で行うかについて記述がないため，ステートマシン図への変換時にその実行順序を決定しなければならない．複数の実行順序が可能な場合，モデル検査では，全ての可能な状態遷移を走査する必要がある．対象とするシステムでエージェントやアクタの数が増加することや複数の実行順序をモデル検査器で検証する場合，状態数は爆発的に増加する．この状態数の爆発を抑制するために，本研究ではコンテキストを考慮したゴールモデルに注目する．

あるシステムが様々な場所で実行される場合，ある環境では必要とされていても別の環境では必要とはされない機能が存在することが考えられる．その場合，それに対応する状態については検証する必要はない．そこで，ゴールモデルで表現されたシステムが利用される環境を考慮して，その状況に適切なゴールと振舞いをどの様に選択するかを示すコンテキストゴールモデル [9] に注目する．コンテキストを用いることで環境に合わせてゴールや振舞いを抜き出し，元のゴールモデルのサブセットを用いて環境にあったゴールモデルを作成することが可能となる．

ゴールモデルで表されたシステムが実行される環境を考慮したコンテキストを用いて，改めてゴールモデルを作成し，それを対象にモデル検査を行うことで，不必要なゴールや振舞いを削除することができ，そこから変換されるステートマシン図の状態数や遷移数の削減が見込まれる．その結果，モデル検査時の状態数も削減することが可能となる．

本研究では，コンテキストを適用したゴールモデルから SPIN 用のモデルを作成する方法を示すとともに，システムの全てのゴールと全ての振舞いが表されたゴールモデルを対象にモデル検査する場合と，システムが実行する環境を考慮したコンテキストを適用したゴールモデルをモデル検査する場合の比較を行う．

2. 関連研究

ゴールモデルから状態遷移図を導出することは，システム開発の初期段階で様々な分析を行うのに役立つため，このような研究は重要視されている．[5] では，要件モデルとアーキテクチャ間の中間の拡張ゴールモデルに対して一連の改良を加えることで，ゴールモデルからステートマシン図として表現される動作モデルを導出するプロセスを提案している．本手法では，この手法を利用するため，詳細な説明は 4 節で行う．[6] では，与えられた i*モデルからそれに対応する可能性として全ての有限状態モデルを生成する Naive アルゴリズムを説明し，Naive アルゴリズムの欠点であるモデル空間の指数関数的成長を解決する方法として，Semantic Implosion アルゴリズムを提案している．i*モデルは実行順序を定めずに作成することができ，ゴールタスク間の部分的な順序関係を持たない．そのため，時間特性仕様に関して i*モデルをモデル検査器で検証することができず，一般的には拡張有限状態モデルを導出する必要がある．Formal Tropos [10] を i*に対し拡張して，ゴール，タスク，資源のモデル構成を Not Created，Created Not Fulfilled，Fulfilled の 3 状態に分類することができる．これらの状態遷移を総当たりで全て置換して得られる可能性のあるすべての有限状態モデルを作成する方法が Naive アルゴリズムである．しかし，これは状態空間の爆発を引き起こす．i*モデルは時間に依存しないが，時間的な観点を持つ特徴やモデル構成が存在する．Semantic Implosion アルゴリズムは，それらを用いて有限状態モデルの可能性を減らしモデル空間の成長を抑える．この手法により，時間に依存しないモデルと標準化されたモデル検査との間の橋渡しとなり，システム開発における要件分析の段階でモデル検査を実行することができる．[8] では，ゴール集合を満たす機械仕様の導出を自動化することにより，システムが必要とする動作の理解と，代替システムの設計の手助けを目指している．この手法では，許可と義務の概念を取り入れた拡張入出力オートマトンと，その導出をサポートする形式仕様の自動合成手法を提案している．まず，状態ベースの同期性とイベントベースの非同期性がゴールからイベントベースの動作モデルの効率的な生成を妨げるため，イベントの実行の許可と義務を区別した deontic IO LTS を提案している．次に，ゴールを満たす最も制約の少ない機械仕様，あるいは，ゴールが実現不可能であることを示す，deontic IO LTS 上での自動の仕様合成技術を提示している．最後に，実現可能性の確認と仕様の合成を利用して代替要件モデルを探し出す例を紹介している．この技術はモデル作成者がドメイン内でゴールが実現できない場合を早

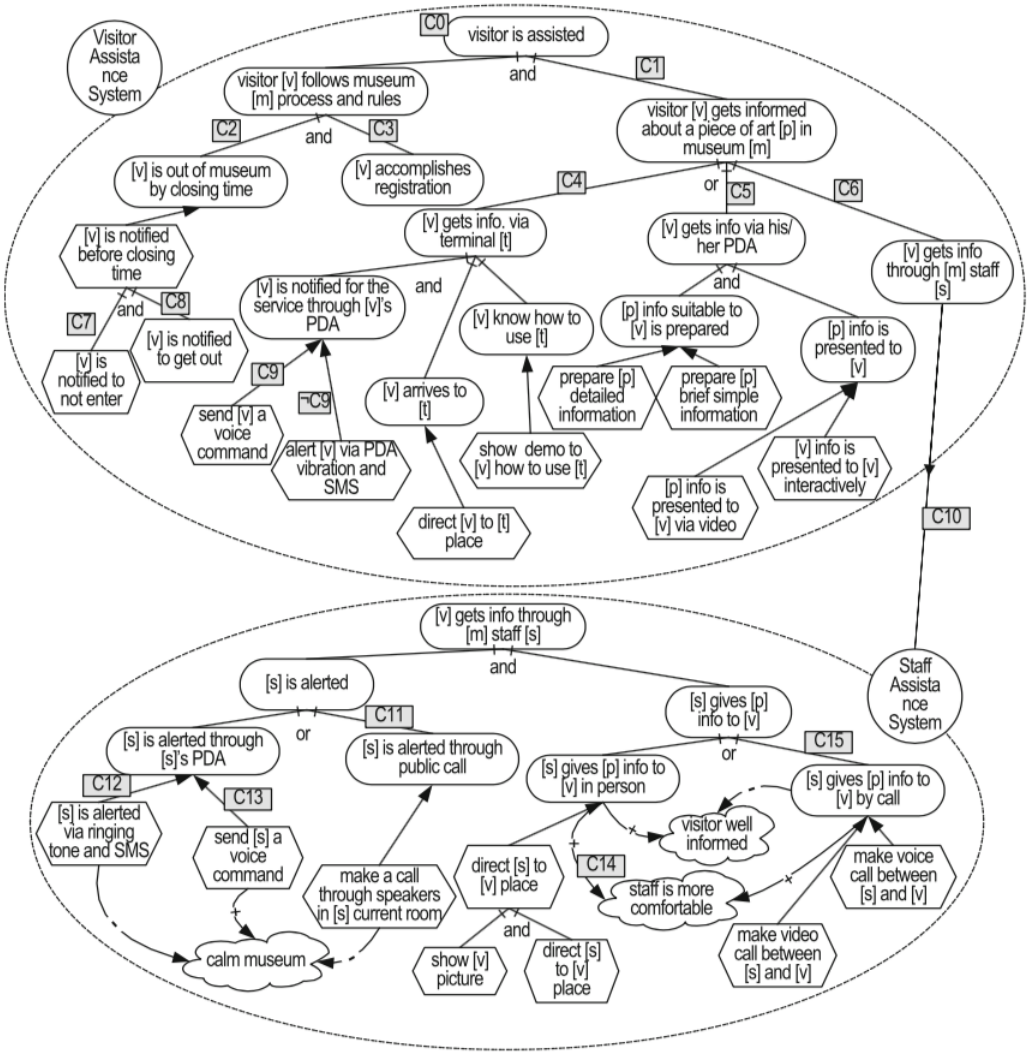

図 1 [9] で用いられている美術館案内システムのゴールモデル

期に特定し，代替モデルの探索を行うのに役立つ．

3. コンテキストゴールモデル

本研究では，[9] で示される美術館案内モバイル情報システムをケーススタディとして用いる．このシステムは来訪者に適切なタイミングで美術館の規則をアナウンスし，また来訪者が美術品に興味を持っているかどうかを把握し，興味があるならばその美術品についての説明も行う．美術品の説明方法には，設置型端末で伝える，来訪者に与えられる携帯情報端末で伝える，スタッフが伝えるという方法がある．システムは主に来訪者やスタッフなどの美術館内の状況を把握するための監視コンポーネントと，システムによるスタッフの呼び出しや美術品の説明を実行する機能コンポーネントの 2 つからなる．図 1 は，美術館案内システムのコンテキストゴールモデルである．

コンテキストとはシステムの周囲の環境を表現したものであり，システムに影響を及ぼすだけでなく，ステークホルダのゴールやタスクの選択に影響を与える可能性がある．変化する周囲の状況を監視しそれに適応するようなシステムの要求分析に用いられるのが，コンテキストゴールモデルである．

図 1 に現れる [s]，[v]，[p]，[m]，[t]，PDA は，それぞれ，スタッフ，来訪者，美術品，美術館，設置型端末，携帯情報端末を表している．この図では，2 つのアクタ (Visitor Assistance System と Staff Assistance System) があり，それぞれの振舞いは破線の楕円で囲まれている内部に示されている．破線の内の楕円はゴールを示し，6 角形はタスクを表す．また，雲型のものは品質要求を示している．そこへ向けての破線矢印に+や-がついているが，それらはその矢印元のタスクが品質に与える影響を表している．

例えば，「calm museum」については，タスク「send [s] a voice command」が品質要求に良い影響を与え，タスク「make a call through speakers in [s] current room」が，に悪い影響を与えていることが分かる．ゴール間を繋ぐ and 分解や or 分解は，そのゴールにを満たすために必要なゴールを示している．and 分解の場合，サブゴールはすべて満

表 1 美術館案内システムのコンテキスト

C	内容
C0	[v] は [m] にいる．かつ，[v] はシステムの支援を受け入れる．
C1	[v] は [m] にいる．かつ，[v] は [p] に興味がある．
C2	閉館時間が近づいている．
C3	[v] が [m] に入った．
C4	[t] は [v] に近く利用可能状態．かつ，[v] は [t] を扱える．
C5	[p] の情報が難しくない．かつ，[v] は PDA を扱える．
C6	[v] は PDA と [t] を扱えない．または，[v] は重要人物．
C7	[v] は [m] に入る途中である．
C8	[v] は [m] にいる．かつ，出口へと向かっていない．
C9	[v] はヘッドホンをしている．かつ，PDA を使用していない．
C10	[v] と同じ言語を使え，[p] を説明できる [s] が空いている．
C11	[s] のいる部屋には音声芸術品がない．
C12	[s] は電話中ではない．
C13	[s] は電話中ではない．かつ，ヘッドホンをしていない．
C14	[v] は [s] に近い．
C15	[s] の PDA は利用可能．かつ，[v] の PDA は利用可能．

たされなければならないが，or 分解の場合はどれか 1 つのゴールが満たされていればよい．ゴールとタスクを繋ぐ終端が矢印である線はゴールを満たすために実行されるタスクの関係を表しており，中央に矢印がある線はアクタが他のアクタに依存している関係を表す．依存関係ではアクタは他のアクタにゴールを移譲することによってそのゴールを達成する．また，C1，C2，...，C15 はコンテキストを表しており，このシステムが動作する環境において，コンテキストが乗っている矢印元のゴールを満たすべきか，タスクを行う必要があるかを表している．コンテキストの内容を表 1 に示す．

システムが来訪者に美術品の説明を行う場合を例にして説明する．システムが来訪者に美術品の説明を行うゴールは「visitor [v] gets informed about a piece of art [p] in museum [m]」であり，これを満たすために 3 つのサブゴールのいずれかが選ばれる．コンテキスト C4「[t] は [v] に近く利用可能状態．かつ，[v] は [t] を扱える．」が成り立つとき，設置型端末を用いる方法，すなわち「[v] gets info via terminal [t]」を選択することができる．このゴールを満たすために，システムは来訪者にそのサービスが受けられることを伝え，その場所まで誘導してそれの使い方を教えるといったタスクが実行される．また，コンテキスト C15「[s] の PDA は利用可能．かつ，[v] の PDA は利用可能．」が成り立つとき，スタッフが伝える方法，すなわち「[v] gets info through [m] staff [s]」が選択することができ，それを満たすためにシステムはスタッフに連絡してゴールの達成を移譲する．移譲されたアクタ Staff Assistance System はゴール「[v] gets info through [m] staff [s]」を満たすために起動し，タスクを選択し実行する．

4. ゴールモデルからステートマシン図への変換

モデル検査器 SPIN で検証するためには，ゴールモデルから実行時のシステム動作を表現するステートマシン図を生成し，それをモデリング言語 PROMELA でモデル化する必要がある．そのため，本章ではゴールモデルからステートマシン図を生成する手法を説明する．ゴールモデルからステートマシン図の生成に関しては [5] の手法を利用して，次の 2 つの手順を行う．

手順 1： ゴールモデルに対してシステム動作の流れを考慮することでゴール達成とタスク実行の順序を決定する．

システム全体を一度に考えるのではなく，洗練された各要素に対してその子要素の振舞いを決定する．実行順序は拡張された正規表現であるフロー式を用いゴール，タスクを要素として記述する．例えば，g1 がゴールならばフロー式における記号 g1 はゴール g1 が満たされた状態を表す．演算子に関しては，連接 g1g2 は g1 が満たされその後 g2 が満たされることを表し，g1*は g1 が 0 回以上満たされることを表す．このように正規表現演算子を用いて実行順序を表すフロー式を構成する．その他の演算子は，代替を表す縦棒”|” ，0 回または 1 回の実行を表す疑問符”?”，0 回以上の実行を表す星印”*”，1 回以上の実行を表す”+”などがある．”|” は二項演算子で，その他は単項演算子である．図 2 は図 1 のゴール「[v] gets info via his/her PDA」以下に対し実行順序を決定したものである．この例ではゴール「[v] gets info via his/her PDA」は，子要素の「[p] info suitable to [v] is prepared」が実行され達成されたのち，「[p] info is presented to [v]」が実行され達成されると満たされることを，実行順序 g18 g21 で表現している．

手順 2： 実行順序に対して図 3 に示す導出パターンを適用することでステートマシン図を生成する．

フロー式が ABC である場合，状態 A に入り，その後状態 B に入り続いて状態 C へと流れる．これをステートマシン図上で表現すると，図 3 の上から 1 つ目の図のようになる．またフロー式 A | B | C は，状態 A，状態 B，あるいは状態 C のいずれかの状態に入ることを表す．これはステートマシン図上では，図 3 の上から 2 つ目の図のように表せる．

あるゴール以下のステートマシン図を求めるとき，サブゴールまたはサブタスクの実行順序と導出パターンからそのサブゴールを状態とするステートマシン図ができる．また，そのサブゴールについて同様に行うことで，そのサブゴールの子であるサブゴールまたはサブタスクを状態とするステートマシン図ができる．これを繰り返すことにより，最終的に求めたいゴールのステートマシン図ができる．図 2 に対しこの導出パターンを適用して得られるステートマシン図を図 4 に示す．

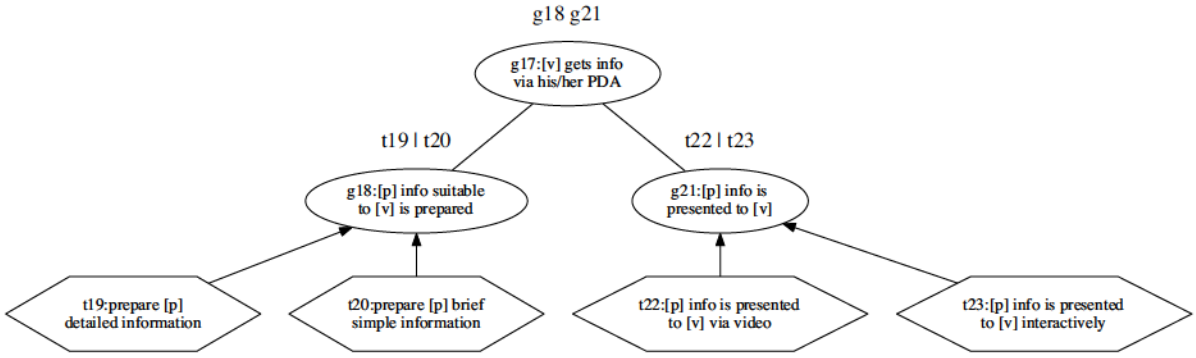

図 2　実行順序をつけた美術館案内システムのゴールモデルの一部

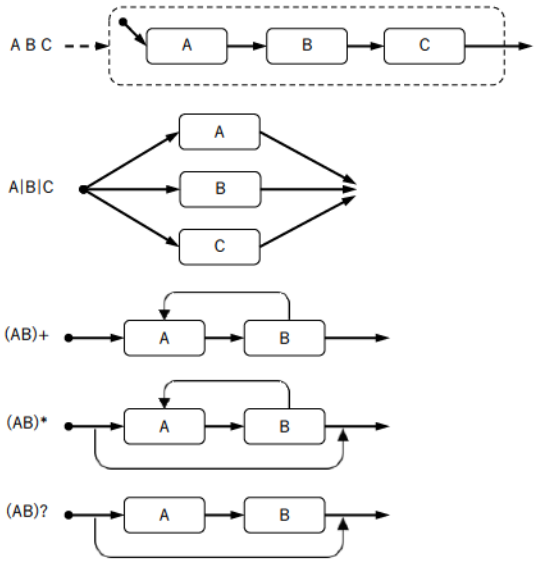

図 3　導出パターン

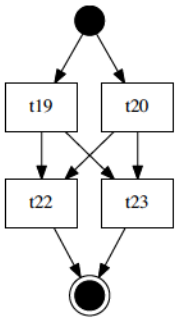

図 4　図 2 のステートマシン図

図 1 のルートゴールの右側部分以降に対し実行順序をつけ導出パターンを適用した結果，得られるステートマシン図を図 5 に示す．このステートマシン図を PROMELA でモデル化し SPIN 用モデルを作成する．

5. コンテキストに注目したステートマシン図の生成

コンテキストとはシステム稼動時の周りのものなどの環境の具現化として定義される [11]．そのようなコンテキストを定義し検証するために，コンテキストを取り込んだ

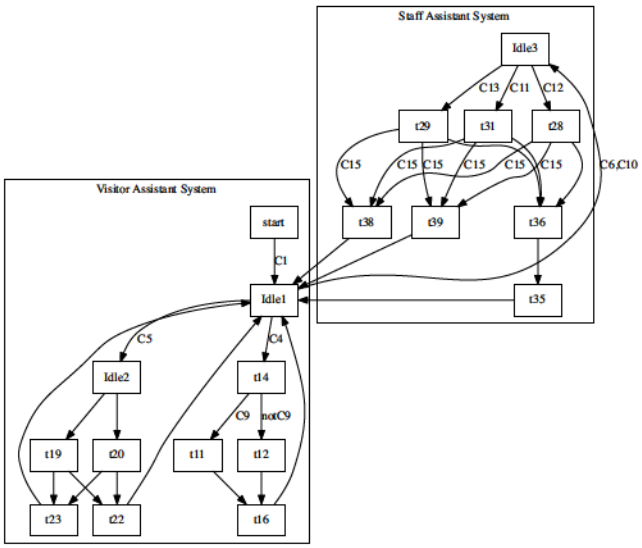

図 5　美術館案内システムの実行時の振舞いを表すステートマシン図

ゴールモデリング技術 [9] を利用する．ここでは，コンテキストはゴールモデルの AND/OR 分解，目的手段関係，アクタ間の依存関係，ルートゴール，ソフトゴールへの貢献の 6 つの点に定義される．この提案の中では，そのままでは検証することができないコンテキストを分析し，ファクトと呼ばれる真理値として計算可能な検証することのできる式の代替集合として表す技術も紹介されている．コンテキストは分析により 2 種類の述語の論理関係式で表すことができ，アクタにとって検証することができる述語をファクト，できない述語をステートメントと呼ぶ．ステートメントはさらに分析することによりファクトとステートメントに洗練され，それを繰り返すことで最終的にコンテキストはファクトの式で表現することができる．例えば，「来訪者は設置型端末の扱いを知っている」というコンテキストはそれ自体では検証することはできないが，コンテキスト分析により「来訪者は設置型端末の操作が遅い，または失敗している」というような検証することのできるファクトの関係式で表すことができる．この手法を用いて，検証可能なコンテキストを含めたゴールモデルを構築することができる．

例えば，スタッフの PDA が常に使用できない美術館で

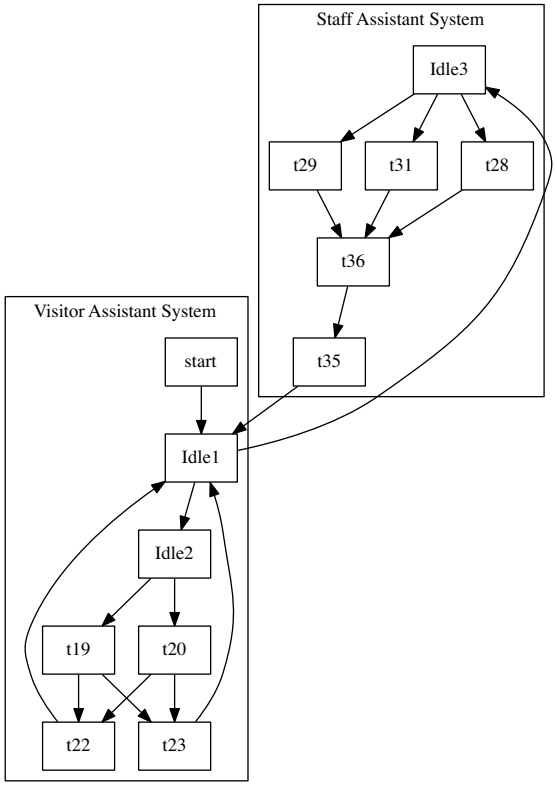

図 6 C15，C4 が偽の美術館案内システムの実行時の振舞いを表すステートマシン図

この美術館案内システムを適用するために検証を行うとする．この場合，コンテキスト C15 は常に偽となり，ゴール「[s] gives [p] info to [v] by call」以下のタスクは必要としない機能となるため，ゴールモデルから削除することができる．また，設置型端末がない美術館であった場合なら，コンテキスト C4 は常に偽となりゴール「[v] gets info via [t]」以下のゴールおよびタスクは削除できる．実行例として C15 および C4 が常に偽である環境でのゴールモデルに対して検証を行う．ゴールモデルからステートマシン図の変換手法を適用した結果，得られるステートマシン図は図 6 のとおりである．

6. ステートマシン図から Promela コードの生成

前節で生成したステートマシン図は，Staff Assistant System と Visitor Assistant System が 1 つのステートマシン図として表されている．このステートマシン図をそのまま Promela コードに変換すると，1 つのプロセスしか生成されない．しかし，スタッフと来訪者は独立してこのシステムを利用するため，Staff Assistant System と Visitor Assistant System は，Promela 上では別のプロセスとして生成する必要がある．図 5 の Idle1 から Idle3 への遷移は，来訪者がスタッフから情報を得る際にスタッフを呼び出すことを表す遷移である．そのため，来訪者はスタッフから情報を得るまで待機する必要がある．Visitor Assistant System の「[v] gets info through [m] staff [s]」のゴールを満たすために，Staff Assistant System に振舞いを移譲している．ゴールモデルに移譲がある場合，アクタ間のやり取りが必要になる．そこで，ゴールモデルにアクタ間のやり取りがある場合，これを Promela コードでは 2 つのプロセス間をチャネルでつなぐことで対応する．来訪者のプロセスとスタッフのプロセスをチャネルを用いてメッセージのやり取りをすることで，来訪者のプロセスがスタッフのプロセスからの情報を得るまで待機するようにする．図 7 は図 5 をゴールモデルの移譲を考慮して生成された Promela コードの一部である．`VisitorAS` は Visitor Assistant System を表すプロセス，`StaffAS` は Staff Assistant System を表すプロセスである．また，`chV` は来訪者の受付チャネル，`chS` はスタッフの受付チャネルである．`L_i1` は Idle1，`L_t14` などは図 5 の状態を示している．`VisitorAS` の 55 行目，56 行目，57 行目では，非決定的にスタッフ 3 人のうち誰かにメッセージを送っている．`StaffAS` では，`chS[id]?s;` により，メッセージを受け取ると，処理を行い，メッセージを送ってきた VisitorAS にメッセージを返信する．

この Promela コードでは，複数の来訪者，3 人のスタッフが存在する場合を表しており，来訪者のプロセスは 3 人のスタッフのうち誰を呼び出してもよいようになっている．

7. 実行結果

実際に Promela コードを生成し，それを SPIN を用いてモデル検査を行った．1 つは，図 5 をもとに，すべての状態を網羅するモデルである．もう 1 つは，図 6 をもとに，設置型端末がなく，スタッフは PDA を持たない環境の美術館での美術館案内システムのモデルである．実行環境は次に示すとおりである．

- OS: CentOS 7.2
- CPU Xeon E5-2667
- メモリ: 512 GB
- ツール: SPIN 6.4.8

モデル検査を実行した結果を表 2，表 3，表 4 に示す．表中の SnVn は，実行するプロセスがスタッフ n プロセス，来訪者 n プロセスを表している．どの値においても，コンテキストを適用して作成したものが全網羅より削減されていることが分かる．特に，S3V3 では，メモリ使用量で約 24%強，状態数で約 22%，実行時間で約 29%の削減となっている．網羅すべき状態が削減されているため，全網羅より削減されることは自明ではあるが，対象とするシステムが運用される環境を考慮して，適切にコンテキストを適用することにより，ゴールモデルをモデル検査する際に状態数やメモリ使用量，実行時間の削減が可能であることを示した．

```
1:   chan chV[V] =[1] of {byte};
2:   chan chS[S] =[1] of {byte};
3:   proctype VisitorAS(byte id){
4:   byte v;
5:   L_i1:
6:   if
7:   ::  goto L_t14
8:   ::  goto L_i2
9:   ::  goto L_i3
10:  fi;
...
53:  L_i3:
54:  if
55:  ::chS[0]!id; chV[id]?v;
56:  ::chS[1]!id; chV[id]?v;
57:  ::chS[2]!id; chV[id]?v;
58:  fi;
59:  goto L_i1;
...
64:  proctype StaffAS(byte id){
65:  byte s;
66:  L_start:  chS[id]?s;
67:  L_i3:
68:  if
69:  ::  goto L_t28
70:  ::  goto L_t29
71:  ::  goto L_t31
72:  fi;
...
```

図 7 図 5 から移譲を考慮して生成した Promela コード

表 2 実験結果 メモリ量 [MB]

	S1V1	S2V2	S3V3
コンテキスト適用	0.29	4.49	3.85×10^3
全網羅	0.28	5.66	5.07×10^3

表 3 実験結果 状態数

	S1V1	S2V2	S3V3
コンテキスト適用	2.30×10^2	5.14×10^4	3.37×10^7
全網羅	2.56×10^2	5.49×10^4	4.34×10^7

表 4 実験結果 実行時間 [秒]

	S1V1	S2V2	S3V3
コンテキスト適用	0.01	0.06	7.06×10
全網羅	0.01	0.10	9.93×10

8. まとめ

本研究では，コンテキストを適用したゴールモデルからステートチャートを生成することによって SPIN 用のモデルを作成する方法を示した．また，システムの全ての機能を表現したゴールモデルと，システム稼働時の環境を考慮したコンテキストを適用したゴールモデルそれぞれについてモデル検査を行った．モデル検査実行時の状態数，メモリ利用量，実行時間に関して比較を行い，コンテキストを適用したゴールモデルは全ての項目において削減できたことを示し，コンテキストの有用性を示した．

ただし，本研究では 1 つの事例しか対象にしていないため，他の事例にも適用可能かどうかを確かめる必要がある．その際に，移譲以外にもアクタ間のやり取りがある場合に適切に Promela を作成するための方法を確立する必要がある．さらに，KAOS などの他のゴールモデルで記述されたシステムに対してもモデル検査を行うことができるようにすることも考えている．

参考文献

[1] Dardenne, A., van Lamsweerde, A. and Fickas, S.: Goal-directed Requirements Acquisition, *SCIENCE OF COMPUTER PROGRAMMING*, pp. 3–50 (1993).

[2] Yu, E. S. K.: Towards modelling and reasoning support for early-phase requirements engineering, *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pp. 226–235 (1997).

[3] Mylopoulos, J., Chung, L. and Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach, *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, pp. 483–497 (1992).

[4] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. and Mylopoulos, J.: Tropos: An agent-oriented software development methodology, *Autonomous Agents and Multi-Agent Systems*, Vol. 8, No. 3, pp. 203–236 (2004).

[5] Pimentel, J., Castro, J., Mylopoulos, J., Angelopoulos, K. and Souza, V. E. S.: From requirements to statecharts via design refinement, *Symposium on Applied Computing, SAC 2014, Gyeongju, Republic of Korea - March 24 - 28, 2014*, pp. 995–1000 (2014).

[6] Deb, N., Chaki, N. and Ghose, A.: Extracting finite state models from i* models, *Journal of Systems and Software*, Vol. 121, pp. 265–280 (2016).

[7] Deb, N., Chaki, N. and Ghose, A.: i* ToNuSMV: A Prototype for Enabling Model Checking of i* Models, *Requirements Engineering Conference (RE), 2016 IEEE 24th International*, IEEE, pp. 397–398 (2016).

[8] Letier, E. and Heaven, W.: Requirements modelling by synthesis of deontic input-output automata, *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pp. 592–601 (2013).

[9] Ali, R., Dalpiaz, F. and Giorgini, P.: A goal-based framework for contextual requirements modeling and analysis, *Requirements Engineering*, Vol. 15, No. 4, pp. 439–458 (2010).

[10] Fuxman, A., Mylopoulos, J., Pistore, M. and Traverso, P.: Model Checking Early Requirements Specifications in Tropos, *5th IEEE International Symposium on Requirements Engineering (RE 2001), 27-31 August 2001, Toronto, Canada*, pp. 174–181 (2001).

[11] Andrea, A. F. and Savigni, A.: A Framework for Requirements Engineering for Context-Aware Services, *In Proc. of 1 st International Workshop From Software Requirements to Architectures (STRAW 01*, pp. 200–1 (2001).