オブジェクト指向ペトリネットによる業務プロセスモデリングと時間制約の検証

秦 良平[1,a)]　飯島 正[2,b)]

概要：業務プロセスから業務ルールを取り出し，管理する業務ルール管理 (BRM; Business Rule Management) は，業務プロセス管理 (BPM; Business Process Management) の効率化に有効である．業務ルールの中でも，「時間制約に関するルール」は重要な要素の一つであり，本研究では，業務プロセスにおいて「時間制約に関するルール」をモデル検査手法を用いて検証することを目指す．著者らが提案する業務プロセス表記法に対して，時間表現を導入し，時間制約を含めたモデル検査ツール UPPAAL で検査可能なモデルである時間オートマトンに自動変換する仕組みを提供することで，それを可能にする．

Business Process Modeling by Object Oriented Petri-Net and Verification of Temporal Constraints

1. はじめに

企業における情報システムを用いた業務改善のために，業務プロセス管理 (BPM; Business Process Management) は効果が大きい．近年では，これに加え，システムに実装される業務ルールを業務プロセスから取り出し管理することで，システムの開発・運用・保守を効率化する業務ルール管理 (BRM; Business Rule Management) が重視されている．本研究では，業務プロセスから抽出した業務ルールが，プロセスに対して矛盾を含むものではないかをモデル検査手法を用いて検証することを目標としており，業務システムの自動化において，タイムアウト処理等を実現する「時間制約に関する業務ルール」を対象とする．

近年の企業活動の広域化に伴い，業務プロセスは巨大化・煩雑化しており，またその業務プロセスが複数の組織間にまたがることが多くなってきている．この現状に対し，これまで著者らは，組織間の協調と相互作用を扱うこと，ならびにアクター（実行者/サブシステム）を資源として扱うこと等を目的に，業務プロセスの表記法として「オブジェクト指向ペトリネット」を導入してきた．このモデルを用いることで，プロセスを構成要素の役割・機能ごとに分割してモジュール性を持たせ，それらの各要素同士の相互作用の記述のしやすさを示してきた．さらに，業務プロセスから抽出した業務ルールを「ドメイン固有言語 (DSL; Domain Specific Language) [1] により表現し，それを著者らの提案する業務プロセスの表記法と統合することで，フローの構造変換 [6]，リソースの再配分 [7]，プロセスを流れるパラメータの外部化 [7] [8] を実現してきた．

本研究では，業務プロセスにおける時間制約に関する業務ルールについて，著者らが提案する業務プロセスの表記法である「オブジェクト指向ペトリネット」に対して，時間制約を含めたモデル検査ツールに適用できるように自動変換を行う方式の実現可能性の確認を試みる．

続く第 2 節で「業務システムにおける業務ルール」について，業務システムを設計・開発する上で，どのような業務要件が業務ルールとして抽出されるかを述べる．さらに，第 3 節で著者らが提案する「オブジェクト指向ペトリネット」の概要と，それによる業務プロセスの表現に関して記述する．第 4 節では形式的に記述されたモデルの論理的な正しさを証明するモデル検査の概要と，そのモデル検査のためのツールとして，Aalborg 大学と Uppsala 大学で共同開発された UPPAAL[9][10] について紹介する．そし

[1] 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology,Keio Univ.
Yokohama, Kanagawa, 223–8522 Japan
[2] 慶應義塾大学理工学部
Department of Science andTechnology,Keio Univ.
Yokohama, Kanagawa, 223–8522 Japan
[a)] r_hata@ae.keio.ac.jp
[b)] iijima@ae.keio.ac.jp

て，第 5 節でオブジェクト指向ペトリネットによる業務プロセスのモデルを UPPAAL で検証可能とするため，時間オートマトンへの自動変換の方法およびそのための設計・実装について述べる．第 6 節では実装したモデル変換のシステムの適用事例を示す．

2. 業務システムにおける業務ルール

近年，業務プロセスから業務ルールを取り出して，業務プロセス管理の効率化を行う業務ルール管理が重視されている．業務ルールとは，「業務権限の範囲内のルール」であり，業務ルール管理を実施するにあたり，業務プロセスから何を業務ルールとして抽出するかという判断も重要になってくる．全てのルールは「定義付けのルール（構造ルール）」または「行動ルール（運用ルール）」のどちらかに分類され [2]，本研究では業務プロセスからそれらに該当するものを業務ルールとして取り扱う．以下に，「定義付けのルール」と「行動ルール」の概要を示す．

2.1 定義付けのルール

定義付けのルールは概念を分類体系化して定義するものであり，それに基づき「計算」を引き起こすことが多い．たとえば，運賃計算を行う業務プロセスにおいて，特定区間に対する運賃は，「東京名古屋間の運賃は，18000 円である」というルールによって定義できる．このような定義付けのルールは，条件と結果の決定表 (Decision Table) や決定木 (Decision Tree) を利用すると，理解しやすく，管理しやすい形で表現できることがよくあり，多くの業務ルール管理システムに導入されている．著者らはこれらの定義付けのルールに対し，if-then ルール形式の DSL により表現し，さらにそれに加え，決定表を導入することで，業務ルールというドメインの知識の共有を実現するとともに，業務プロセス管理の効率化を実現してきた [8]．

2.2 行動ルール

2.2.1 行動ルールの概要

行動ルールは，行為や行動に関する制約を表現することが多く，必要に応じて，判断や行動に関する評価をも含む．たとえば，「インターネットでチケットを予約した時点から 48 時間以内に購入を完了しなければならない.」といった当事者の行動を規制する制約である．

2.2.2 時間制約に関する行動ルール

当事者の行為や行動を規制する制約条件には，様々な種類のものが考えられるが，業務システムにおける業務ルールの中で，時間制約は重要である．人を含む業務プロセスを情報システム化し，自動運用するためには，タイムアウト処理はほぼ不可欠といえるからである．そこで本研究では，時間制約に関する業務ルールを対象とし，著者らが取り組んできたオブジェクト指向ペトリネットによる業務プロセス表現に統合することを試みている．

3. オブジェクト指向ペトリネット

オブジェクト指向ペトリネットを業務プロセスを表現するモデルとして用いる．オブジェクト指向ペトリネットとは，オブジェクト指向の概念に基づいてモジュール性を取り入れたペトリネットの拡張モデルであり，著者らは，nets-in-nets 意味論に基づく参照ネット (Reference Net) [3] の考え方を採用している．ペトリネットで業務プロセスを記述する先行事例に YAWL [4] があるが，本研究は，この nets-in-nets 意味論に基づくモジュール性の導入に特徴がある．このモデルの例を示す (図 1)．

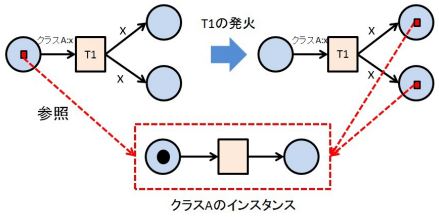

図 1 オブジェクト指向ペトリネットのモデルの例

nets-in-nets 意味論 [3] とその拡張に基づくオブジェクト指向ペトリネットは次の 2 種類のトークンを持つ．

- 単純トークン (Black Token)
  通常の P/T ネット (Place/Transition ネット) におけるトークンのことである．プレースに存在する単純トークンは，トークン数で示す．
- 参照トークン (Reference Token)
  別のサブシステムを表現するサブネットへの参照 (reference) を持つトークンのことである．

図 1 の例では，クラス A のインスタンスへの参照を持っているトークンが参照トークンであり，クラス A のインスタンスを表現するペトリネットのプレース上に存在するトークンが単純トークンである．

この例で，参照トークンを持つ側のネットをシステムネットと呼び，参照トークンが参照している先のサブネットをオブジェクトネットと呼ぶ．このオブジェクトネットは，一つのオブジェクト（インスタンス）として識別される．こうして参照を通して階層構造が形成されるが，この階層構造は 2 階層に限定されるものではなく多層性をも持ちうる (参照に依って形成されるので相互参照による複雑な構造も形成可能ではあるが，　実用性の観点から，いわゆる part-of 関係を想定して欲しい)．つまり，オブジェクトネットが，また別のサブネットへの参照を持つことで，そのサブネットにとってのシステムネットに相当することもありうる．すなわち，システムネットとオブジェクトネットの関係は，あくまで相対的なものといえる（システムネット側がオブジェクトネット群を協調制御していると

みなすとわかりやすい)．そこで，最上位のシステムネットを，どこからも参照されていない特別なオブジェクトネットとみなすことで，以降はシステムネットも含めてオブジェクトネットと称し，相対的な関係性を強調する必要のあるときのみシステムネットという用語を用いる．

参照トークンも単純トークンと同様，ネットワーク中を遷移する．参照トークンがトランジションの発火によって複数個にコピーされるときは，オブジェクトネットの実体ではなく，それへの参照がコピーされる点に注意して欲しい．

また，オブジェクトネットとシステムネットで，トランジションを同期発火 (interaction) させることも可能である．この同期発火が行われるには，全ての同期トランジションが発火可能な状態であることが条件である．nets-in-nets 意味論 [3] とその拡張に基づくオブジェクト指向ペトリネットにおいては，次の 3 通りの発火則が考えられる．

(a) interaction(相互作用)
システムネットとオブジェクトネットが同期して遷移する場合である．

(b) transport(移送)
システムネットの発火に際し同期して発火するオブジェクトネットがない場合である．

(c) autonomous(自律)
システムネットとは独立に，オブジェクトネット内のトランジションが発火する場合である．

これらの特徴から，オブジェクト指向ペトリネットを業務プロセス表現を用いることで，以下が可能となる．

- 組織間にまたがる業務プロセス (inter-organizational business process) における組織間の協調表現
- アクター（実行者/サブシステム）を資源としてモデルに表現すること，

4. モデル検査手法

4.1 モデル検査

近年，組み込みソフトウェアも大規模化，複雑化が進んでおり，それらが正しく処理を実行し，安全に動作する高い品質と信頼性が求められている．この高品質と高信頼を示すために，あらゆる全ての動作パターンを人手でテストするには，膨大な工数が必要となり，現実的ではない．そこで，注目されているのがモデル検査 (Model Checking) である [9]．モデル検査とは，システムのモデルを設計した時点およびその範囲で，そのシステム上で起こり得る状態を網羅的に調べ，「システムが満たすべき動作を完了する」あるいは「満たすべきでない状態にならない」といった検証項目を確認する手法である．

モデル検査手法は，実際に製品を作ってテストを繰り返し行うにはコストが嵩んでしまう組み込みソフトウェアの検証において多く用いられている．将来的に大規模化，複雑化していくと考えられる業務システムの開発においても，このモデル検査手法を導入することで，システムの不具合の早期発見につながると期待できる．本研究では，そのような流れを汲み，著者らが提案する業務プロセス表記法でモデル検査を実施するためにモデル変換機能を導入する．

4.2 UPPAAL によるモデル検査

モデル検査ツールとして，SPIN[5] を代表に，数多くのものが存在する．しかし，それらの多くは動作ロジックに関する性質を検証するに留まり，時間制約に関する性質を検証する機能が含まれていない．

そこで，時間制約の性質についても検証可能なモデル検査手法の研究が行われてきた．その手法として，時間制約を表現することができる時間オートマトンに基づいたものがある．そのためのツールとして UPPAAL[10] が，デンマークの Aalborg 大学とスウェーデンの Uppsala 大学の共同で開発された．本論文では，この UPPAAL を採用している．

5. 時間オートマトンへのモデル変換

オブジェクト指向ペトリネットにより記述した業務プロセスのモデルに対して，時間制約に関するルールについてその性質の検証を実施するために，UPPAAL で検査可能なモデルである時間オートマトンに自動変換する．

本方式で前提とするのは，業務プロセス全体が一つの有界ペトリネットで記述できることである．有界性は，オートマトンへの変換の際に，状態数を有限に抑えるために不可欠である．この有界性は，単純トークンだけではなく，参照トークンにも求められる．すなわち，同じクラスのオブジェクトネット (インスタンス) は複数存在しても構わないが，その個数には上限がある．さらに，もう一つ強い条件として，実行途中に新たなオブジェクトネット (インスタンス) を生成させず，その上限個数のオブジェクトネット (インスタンス) を事前に全て生成しておき，全体で一つの有界ペトリネットに合成できるものとする (本論文の例題では，図版のサイズの制約上，上限個数は 1 個のケースしか示していない)．この条件は，実用上，現実と必ずしもそぐわないが，検証をすることが目的であるということからすれば支障はないと考えている．

時間オートマトンに変換できるように，まず有界なペトリネットに時間表現を導入する．プロセスを構成する各オブジェクトの振る舞いを記述し，さらに時間表現が導入されたオブジェクトネット同士を合成し，一つのペトリネットを得る．その一つのペトリネットを一つの時間オートマトンに変換することにより，その一つの時間オートマトンを対象としたモデル検査を可能とさせる．

本来，オブジェクト指向ペトリネットのモデル検査のためには，ペトリネット（オブジェクトネット）を合成する代

わりに，個々のオブジェクトネットを，それぞれ同期しあう並行プロセス（時間オートマトン）に変換した上で，そのままモデル検査を行えばよい．もしくは，中間的な方法として，それらの時間オートマトンを一つのオートマトンに合成した上で，モデル検査を行なう方法も（その必要があるかどうかは別として）ありうる．しかし，本論文の執筆時点では，オブジェクト指向ペトリネットに時間表現を付与するにあたり，まずは時間制約に関するビジネスルールとして，どのような表現力を与えるべきか模索しつつ，それをペトリネット上で表現し，さらに使用するモデル検査器 (今回は UPPAAL) で検証できるプロセス表現 (時間オートマトン) に変換できることを確認する，という「摺り合わせ」作業のための実験環境の整備も目的であった．そこで，各オブジェクトネットを一つのペトリネットに合成して，時間オートマトンに変換する方法を採用し，まず第一段階としての有用性の確認を試みた．

5.1 有界ペトリネットから有限オートマトンへの変換

まず，モデル変換の基礎として有界な (時間制約のない) ペトリネットからオートマトンへ変換することを考える．ペトリネットでは，全てのプレース上におけるトークン数をベクトルで表現したマーキングをシステムの一つの状態と見なすことができる．また，マーキングの変化の要因となるトランジションの発火を「状態間の遷移」として対応付けることができる．つまり，ペトリネットの全ての取り得るマーキングとその際の発火可能なトランジションを調べ上げることにより，有界ペトリネットから有限オートマトンへの変換が可能である．図 2 および図 3 は，この操作による有界ペトリネットから有限オートマトンへの変換の例である．

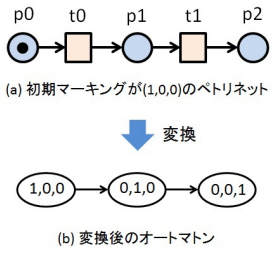

図 2 初期マーキングが (1,0,0) のペトリネットの変換

図 2 および図 3 の変換前のペトリネットは，プレース，トランジション共に同様の構造を持つものであるが，初期マーキングの違いから，取り得る全てのマーキングの集合は異なるものとなり，それぞれ変換後のオートマトンも異なる．

ここまで，ペトリネットのトークンが単純トークン（ブラックトークン）の場合を扱ってきたが，オブジェクト指向ペトリネットの場合，トークンが参照トークン（リファレンストークン）の場合がある．参照トークンの場合，マーキングは単にトークン数で表現することができず，どのオブジェクトネットへの参照を持つかを区別する必要がある．このことから，変換後のオートマトンの状態数は組合せ的に増える場合がある．図 3(a) におけるペトリネットのトークンは単純トークンなので，図 3(b) のオートマトンに変換できる．一方，図 3(a) におけるペトリネットのトークンが図 4 のように参照を持っていた場合，それらのトークンは識別されるので，変換後のオートマトンの状態は図 5 のように増える．

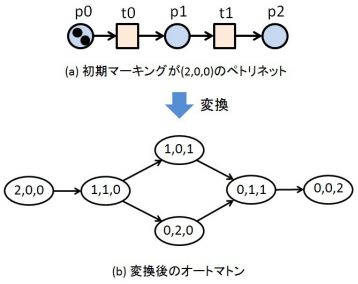

図 3 初期マーキングが (2,0,0) のペトリネットの変換

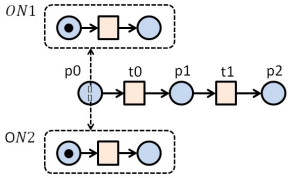

図 4 変換前のオブジェクト指向ペトリネット

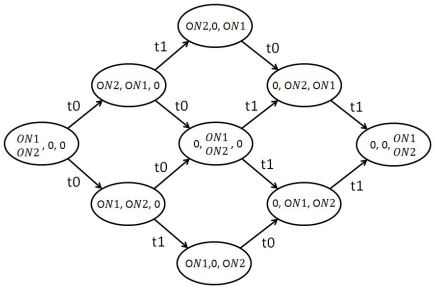

図 5 変換後のオートマトン

5.2 ペトリネットへの時間表現の導入

オブジェクト指向ペトリネットを時間オートマトンに変換するためには，時間オートマトンのモデル上で定義される時間表現を変換前のモデルにも定義しておく必要がある．変換の方法として，最下層のオブジェクトネットに時間表現を導入し，それらを一つのペトリネットに合成して，時間オートマトンに変換する方針であるため，単純なペトリネットへの時間表現の導入を考える．時間オートマトンで

定義される時間表現で，変換前のモデルにも定義したものは，図 6 の時間オートマトンのモデルの例にも示される以下の 3 つである．

(1) **guard（ガード）**
プロセスが遷移するときに満たさなければならない条件である．図 6 の例では，時間変数 t が 5 以上であれば遷移可能であることを意味する．

(2) **invariant（不変式）**
プロセスのある状態に滞在可能な条件である．図 6 の例では，時間変数 t が 10 未満の間，ロケーション $L0$ に滞在可能であることを意味する．

(3) **assignment（更新）**
変数の代入や初期化を行う式である．図 6 の例では，状態が遷移する際に．時間変数 t を 0 に更新することを意味する．

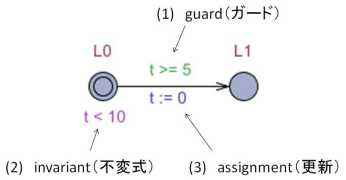

図 6 時間オートマトンモデルの時間表現

図 6 で記述される時間オートマトンにペトリネットを変換するために，ペトリネットに時間表現を導入する．導入の方法として，ペトリネットで状態遷移の要因となるトランジションへ入力アークに先の 3 つの時間表現を導入した．これは，状態が遷移する際に条件を判定したり，変数の更新を行うためである．ペトリネットに時間表現を導入したものが，図 7 におけるトランジション t0 への入力アーク上の条件式で表現される．上から順に，guard（ガード），invariant（不変式），assignment（更新）を表現している．なお，この例では，マーキング $\mu = (1, 0)$ が図 6 の時間オートマトンの状態 $L0$ を，マーキング $\mu = (0, 1)$ が状態 $L1$ を意味する．

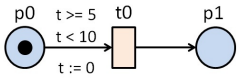

図 7 ペトリネットへの時間表現の導入

5.3 時間表現を導入したペトリネットの合成

前節までにオブジェクト指向ペトリネットのモデルにおける最下層のオブジェクトネットに時間表現を導入することまで述べた．続いて，それら各オブジェクトネットを一つのペトリネットに合成する方法を述べる．たとえば，図 8 のようなオブジェクト指向ペトリネットを考える．オブジェクトネット $ON1$ とオブジェクトネット $ON2$ のトランジション $t0$ が同期して発火するものとする．$ON1$ の $t0$ の入力アークに時間表現を取り入れている．

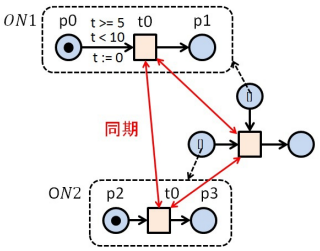

図 8 オブジェクト指向ペトリネットの例

図 8 で記述される時間表現を導入したオブジェクト指向ペトリネットのモデルを，一つのペトリネットに合成する (図 9)．今回はトランジション間の同期制約の定義を簡単化するため，各オブジェクトネットの同期トランジションには同じ名前を与えており，合成の際には同名のトランジションを全て同一化する．名前を異なるものにしておくと，合成後はそのトランジションを保持したまま合成される．同様に，各オブジェクトネットのプレースにおいても，他のオブジェクトネット上のプレースと同じ名前を与えると，合成後に一つにまとめられ，名前が異なればそのまま保持される仕組みを取っている．

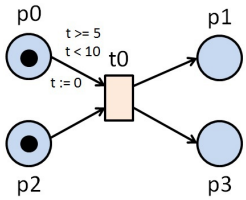

図 9 合成後のペトリネット

オブジェクト指向ペトリネットのオブジェクトネットに時間表現を導入し，それらを一つに合成することができたので，前節で述べた通り，時間オートマトンに変換することができる．図 9 の時間ペトリネットもマーキング $\mu = (p0, p1, p2, p3)$ とすると，$\mu = (1, 0, 1, 0)$ を図 6 の $L0$，$\mu = (0, 1, 0, 1)$ を $L1$ と見なすことで，図 6 の時間オートマトンに変換することができる．

5.4 モデリング・エディタによるモデル変換

前節までに述べた，オブジェクトネットに時間表現を導入し，オブジェクトネットを一つのペトリネットに合成して，時間オートマトンへモデル変換を行う図形エディタを実装した（図 10，図 11）．

変換後の時間オートマトンの状態名の名前に関して，初期状態の場合は，「before_(その状態の後に起こる遷移名)」

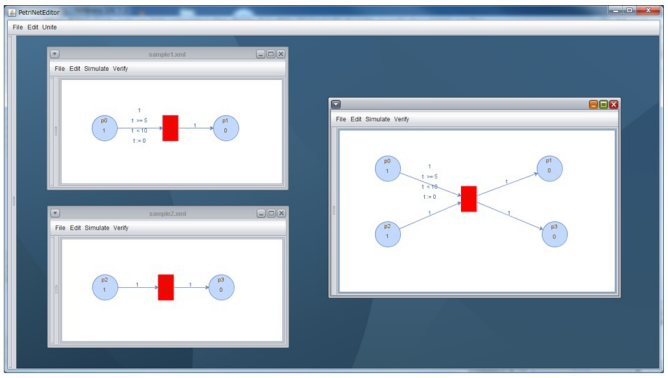

図 10　モデリング・エディタによるペトリネットの合成

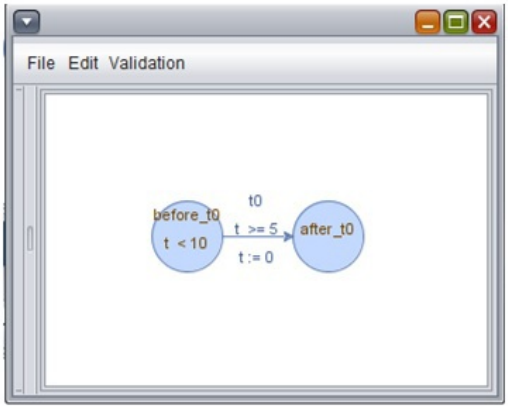

図 11　モデリング・エディタによる変換後の時間オートマトン

を，その他の状態は，「after_(その状態に至る要因となった遷移名)」という規則に沿って定義するようにしている．

6. ケーススタディ

時間表現を導入したオブジェクト指向ペトリネットで業務プロセスをモデル化し，そのモデルを時間オートマトンに変換し，時間制約に関する業務ルールを UPPAAL で検証する事例を示す．事例として，企業における社員の出張旅費申請のプロセスを挙げる．

6.1 事例を用いたモデル変換

企業における社員の出張旅費申請のプロセスにおいて，プロセスを構成する実体となるオブジェクトについて，社員，申請書，課長，財務部を定義した．それぞれをオブジェクトネットとして定義し，オブジェクト指向ペトリネットでモデル化すると，図 12 のように定義できる．時間表現は申請書のモデルに記述している．今回申請書に時間表現を記述したのは，後に，ある時間において申請書が満たすべき状態を決める業務ルールの検証を行うことを目的としているためである．

この事例のプロセスの概要を以下に示す．

(1) 社員が出張旅費の申請書を作成する．
(2) 社員は申請書の必要事項を記入した後，その申請書を課長に提出し，課長が受け付ける．
(3) 課長は社員から提出された申請書の承認を行い，財務部へその申請書を提出し，財務部が受け付ける．
(4) 財務部は提出された申請書の内容を審査する．
(5) 財務部は申請書の審査の結果を，提出した社員へ通知し，社員が受信する．

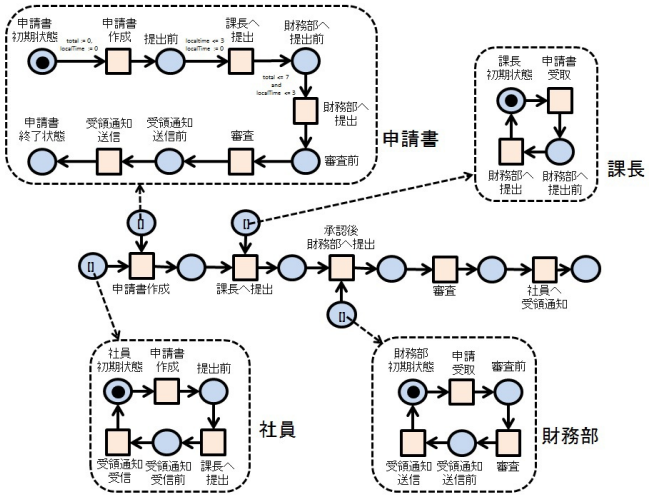

図 12　出張旅費申請のプロセス

前節の時間オートマトンへの変換方法に従い，図 12 の時間表現付きオブジェクト指向ペトリネットを一つの時間ペトリネットに合成すると，図 13 のモデルが得られる．

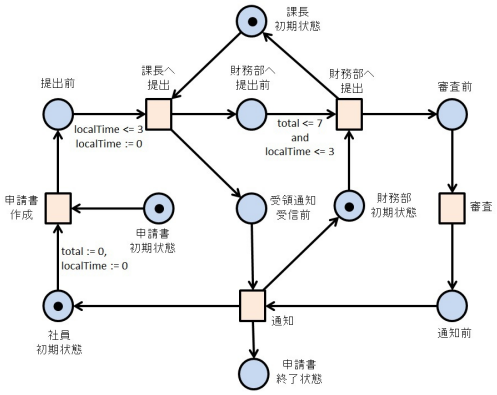

図 13　出張旅費申請のプロセスの合成したペトリネット

このモデル変換の一連の流れを実装し，モデリング・エディタの中で合成機能を実施した様子を，図 14，図 15，図 16 に示す．エディタにより合成したペトリネットのノードの配置は，変換操作の際，自動的にレイアウトを行うため，図 13 とは見た目が異なる．

6.2 時間制約に関する業務ルールの検証

事例に対し，時間制約に関する以下のような業務ルールを定義し，この成立を検証する．

- 出張旅費申請書に関して，社員がそれを作成した時点

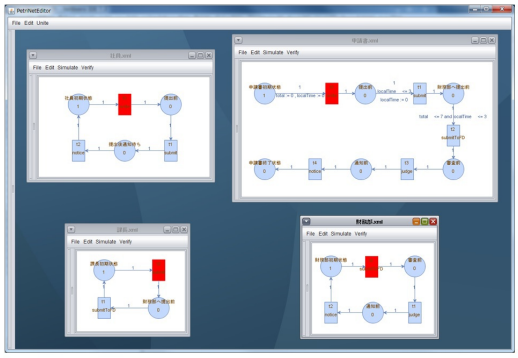

図 14 モデリング・エディタ上での事例のオブジェクトネット

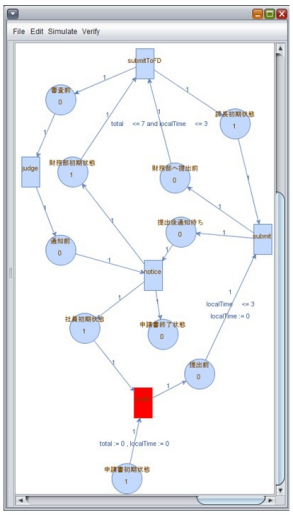

図 15 モデリング・エディタ上での事例の合成ペトリネット

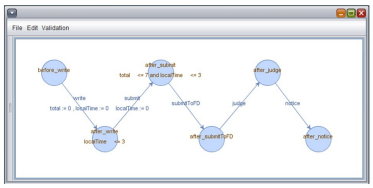

図 16 モデリング・エディタ上での事例の時間オートマトン

から，7 日以内に課長の承認を得て，財務部へ提出しなければならない．

これは検証にあたって以下のように言い換えが可能である．

- 出張旅費申請書が社員により作成された時点から 7 日以上経過しているとき，申請書は課長への提出前の状態または財務部へ提出前の状態にはあってはならない．

UPPAAL を用いて上記の業務ルールを検証するには，下記のように検証項目として時相論理式で記述しなければならない．

- `A[] ( total >= 7 imply not (Process.after_write or Process.after_submit) )`

この検証式は，すべての実行系列で括弧内の性質が成立することを意味する．UPPAAL ツール上で事例の時間オートマトンモデルが展開されている様子と，上記の検証式により検証を行っている様子を示す (図 17, 図 18)．

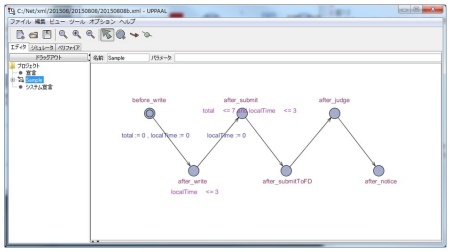

図 17 UPPAAL 上での時間オートマトンモデルの記述

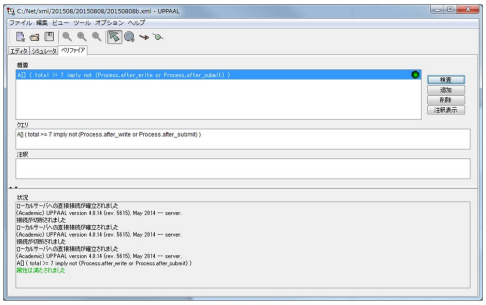

図 18 検証式と検証

7. 評価

実装したエディタにより，時間表現付きペトリネットの合成，および，時間オートマトンへの自動変換，さらに UPPAAL での業務ルールの検証という，オブジェクト指向ペトリネットによる業務プロセスのモデルの設計から，検証までの一連の流れを実行できることを確認できた．モデル記述の際に，初めから巨大なプロセスを記述するよりも，分割してモデルを記述し，それらを自動合成する方式で，モデル記述の効率性は向上したといえる．

しかし，今回は時間表現の記述はオブジェクトネットの中の一つに限定した．同じ時間変数が複数のオブジェクトネット上の時間表現の中で記述されたとき，それらをどのように合成するかには，現在対応していない状況である．

また，業務ルールの検証に関しては，プロセスの状態がどうあるべきか「義務」の様相に関して，検証可能であることが示された．しかし，他に「可能」や「許可」などの様相を用いて表現される業務ルールに対しての検証の可否は確認はできていない．

モデル検査が未だ広く普及していない理由として，検証式の生成の難しさがある．本研究では，検証式の記述は手作業で行っている．本研究の提案をより効果的にするためには，モデルを記述するとともに，いくつかの情報を入力すれば検証すべき性質を検査する検証式を自動生成する仕組みの提案も必要であると言える．

8. 関連研究と本研究の位置づけ

ビジネスプロセスの形式検証には多くの先行研究があり，その中には，BPEL 等のプロセス表現からペトリネットへ変換してペトリネットの検証能力を用いるものもある ([12] の 2.1 節)．また，本論文と同様に BPEL-WS-CDL から時間オートマトンへ変換し，UPPAAL で検証する方式をとるものもある [13]．ペトリネット自体（特にワークフローネットと呼ばれるサブクラス）も，YAWL [4] などビジネスプロセス表現として提唱されているが，時間制約など，従来のペトリネットが備えていない表現に関して，検証を行なう必要も生じている．

時間を取り入れたペトリネット (その一種である Timed Arc Petri Net) を対象に，モデル検査を行なうツールに，Aaalborg 大学で開発された TAPAAL [11] があり，オプションとして，ペトリネットから UPPAAL モデルへ変換する機能もある．しかし，TAPAAL はオブジェクト指向ペトリネットを取り扱うことはできない．

本研究では，nets-in-nets 意味論に基づくオブジェクト指向ペトリネットに対し独自に時間拡張を行い，UPPAAL の時間オートマトンモデルへ変換して，検証を試みている．ここでの時間拡張は，Timed Arc Petri Net(TAPN と略す) と近い考え方を導入している（TAPN には転送アーク等の特化された表現力をもっているが，本モデルの方が表現力に一般性を残しているとはいえる)．TAPN は，トークンが内部に時計 (age) を持っていて，アーク上に，その時計に関する時区間制約を一種のゲート条件として与えるものである．本論文のモデルにおいても，各オブジェクトネットが持っている時間変数を，そのオブジェクトに対してシステムネットに相当するペトリネットのアーク上で，ゲート条件として参照することができる．これは，他のタイプの時間拡張されたペトリネットとは異なる．時間付きペトリネット (Timed Petri Net; TdPN) は，トランジションに発火遅延時間や継続時間を導入したり，プレースに発火可能になるまでの遅延時間を導入するものであり，時間ペトリネット (Time Petri Net; TPN) は，トランジションに発火遅延時間と上限と下限の対を付与するものである．いずれも，TAPN のようにトークン，あるいは，本研究のようにトークンの参照先のオブジェクトが内部に時計を持っているというモデルではない．

本研究の本来の狙いは，個々のオブジェクトネット毎に状態モデルを生成し，互いに制約（同期制約）として機能しあう状態モデル間の相互作用を直接的に扱うことであるが，現時点ではそれに至っていない．今回採用した方法は，ペトリネットのレベルで複数のオブジェクトを一旦合成して，その状態モデルを UPPAAL モデルに変換することである．しかし，次のステップとして個々のオブジェクトネット毎に，状態モデルを生成し，それを合成して UPPAAL を使って検証することを想定しており，その開発に向けて作業を継続的に行っている．

本研究で対象とする時間制約は，本論文で例示したタイムアウト（時間切れ）が代表例であるが，それ以外に，どのような時間制約について取り扱うべきか，頻出パターンの調査も今後進める必要があり，その上で，その内，どの程度の範囲が UPPAAL で検証可能なのか評価を引き続き行う予定である．

9. まとめ

オブジェクト指向ペトリネットによる業務プロセス表現に時間制約に関する業務ルールの表現を導入し，その業務ルールと業務プロセスとの無矛盾性の検証を行うために，時間オートマトンに変換するための仕組みを構築した．

本研究の段階では，まだ対象とするペトリネットに対しても，制約事項が大きく，また状態縮約の手法などを導入していないため取り扱える規模が限られるが，制約を緩和し実用性を向上させることに取り組んでいく．

これと並行して，実用性の観点から，業務ルールの表現や時制論理による検証式の表現をより現場の技術者の感覚に近づけていくことを目標に据えた取り組みを進めていく．

参考文献

[1] Debasish Ghosh: “実践プログラミング DSL 〜 ドメイン特化言語の設計と実装のノウハウ,” 翔泳社, 2012.
[2] ロナルド・G・ロス: “アジャイル経営のためのビジネスルールマネジメント入門,” 日経 BP 社, 2013.
[3] Rüdiger Valk: “Object Petri nets Using the nets-within-nets paradigm, ” LNCS 3098, pp.819–848, Spriner, 2004.
[4] W.M.P. van der Aalst and A.H.M. ter Hofstede: “YAWL: Yet Another Workflow Language,” QUT Technical Report, FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.
[5] “SPIN”, http://spinroot.com/spin/whatispin.html
[6] 飯島 正: “アスペクト指向ワークフロー変換 〜 オブジェクト指向ペトリネットによるワークフロー表現への適用〜,” 知能ソフトウェア工学研究会技報, Vol.112, No.165, pp.1-6, 電子情報通信学会, 2012.
[7] 飯島 正, 片山 輝彦, 金子 良太, 高橋 貴大: “オブジェクト指向論理ペトリネットを使った業務プロセス/業務ルール管理,” 第 8 回 全国大会・研究発表大会, 情報システム学会, 2012
[8] 飯島 正, 秦 良平, 金子 良太: “オブジェクト指向ペトリネットとルールに基づく業務プロセスの理解支援,” 第 9 回 全国大会・研究発表大会, 情報システム学会, 2013
[9] 大須賀昭彦 (監修), 長谷川哲夫，田原康之，磯部祥尚: “UPPAAL による性能モデル検証,” 近代科学社, 2012.
[10] “UPPAAL,” http://www.uppaal.org/
[11] “TAPAAL,” http://www.tapaal.net/
[12] S. Morimoto: ”A Survey of Formal Verification for Business Process Modeling,” in Proc. of ICCS 2008, pp. 514-522, 2008.
[13] Diaz, G., Pardo, J.J., Cambronero, M.-E., Valero, V., Cuartero, F.: ”Automatic Translation of WS-CDL Choreographies to Timed Automata”, in Proc. of EPEW/WS-EM 2005, pp. 230?242, 2005.