プリエンプティブスケジューリングにより リソースを共有する複数タスク動作仕様の性能検証

百々　太　市[†]　中　田　明　夫[†]

性能とリソースの制約が共に厳しく，高い信頼性を要求される組込みソフトウェア開発においては，設計モデルの段階で時間制約やリソース制約を考慮した最悪時性能の検証を行うことが有用である．我々は従来，ノンプリエンプティブスケジューリングでリソースを共有し，分岐や並列などの制御構造を持つ複数タスク動作仕様から，優先権付き時間ペトリネットに変換し，スループット性能の検証を行う手法を提案しているが，プリエンプティブなスケジューリング方式を表現できない問題があった．本研究では，優先権付きストップウォッチ時間ペトリネットを用いることにより，プリエンプティブスケジューリングでリソースを共有する複数タスク動作仕様の性能検証を行う手法を提案する．

Performance Verification of Multi-Task Specification with Resources Shared by Preemptive Scheduling

Taichi Dodo[†] and Akio Nakata[†]

In the development of embedded software, where performance and resource requirements must be highly optimized and high-level reliability is required, it is useful to verify the worst case performance considering time and resource constraints in the early phase of the design process. We formerly proposed a method to translate a multi-task specification, which has a control structure such as choice or parallel and shares resources with nonpreemptive scheduling, into a Prioritized Time Petri Net(PrTPN), to verify its throughput performance. However, it cannot handle preemptive scheduling. In this paper, we propose a performance verification method of multi-task specification with preemptive resource scheduling, using Prioritized Stopwatch Petri Nets(PrSwPN).

1. はじめに

性能とリソースの制約が共に厳しく，高い信頼性を要求される組込みソフトウェア開発においては，設計モデルの段階で時間制約やリソース制約を考慮した最悪時の性能検証を行うことが有用である．性能検証とはソフトウェアシステムがユーザの性能要求を満たすか否かを性能検証モデルに基づき検証することを指す．性能検証を行う手法としてはソフトウェア仕様モデルからソフトウェア性能検証モデルへ変換し，性能検証を行う手法が提案されている[1)]．ソフトウェア性能検証モデルは，待ち行列ネットワークなどの確率的モデル[2)]と時間ペトリネット[3)]などの確定的モデルに分類される．確率的モデルはシステムの平均性能を考慮するのに対して，時間ペトリネットは最悪時（および最良時）の性能を考慮する．性能とリソース制約が共に厳しく，高い信頼性を要求される組込みソフトウェアを対象とする場合，最悪時に性能要求を満たせるか否かの検証が重要となる．

文献 4) などでは時間オートマトンを用いて，プリエンプティブスケジューリングでプロセッサリソースを共有する複数タスク動作仕様の検証を行う手法が提案されている．しかし，プロセッサ以外のさまざまなリソース（バス，ネットワーク，共有メモリ等）をさまざまなスケジューリング方式で共有するシステムを統一的に扱う手法は我々の知る限りでは提案されていない．

我々は従来，ノンプリエンプティブスケジューリングでリソースを共有し，分岐や並列などの制御構造を持つ複数タスク動作仕様から，優先権付き時間ペトリネットによる性能検証モデルに変換し，スループット性能の検証を行う手法を提案している[5),6)]．しかし，従来手法により生成された性能検証モデルはプリエンプティブなスケジューリング方式を表現できないという問題があった．

† 広島市立大学　大学院情報科学研究科　システム工学専攻
Department of Systems Engineering, Graduate School of Information Sciences, Hiroshima City University

本研究では，従来手法[6] を拡張し，優先権付きストップウォッチペトリネット (PrSwPN)[7),8)] を用いることにより，プリエンプティブなリソーススケジューリング[9)] を考慮した性能検証モデルを生成する手法を提案する．時間ペトリネットを用いることにより複数のリソースをそれぞれ異なるスケジューリング方式で共有するシステムを統一的なモデリング方式で簡潔に扱うことが可能となる．また，経過時間の計測の中断や再開が指定可能で，動作の優先関係を指定可能な優先権付きストップウォッチペトリネットを用いることにより，プリエンプティブな固定優先度スケジューリングを扱うことが可能となる．生成した PrSwPN を既存の検証ツール TINA[10)] によって検証することにより，プリエンプティブスケジューリングによりリソース共有を行う複数タスクから構成されるシステムに対して，設計初期段階における最悪時を考慮した性能の検証が可能となる．提案手法の評価のため，例題に提案手法を適用して生成した性能検証モデルに対して，単位時間に処理できるデータ数を表すスループット要求を満たすかどうかの検証を行った．

2. 優先権付きストップウォッチペトリネット

$\mathbb{N}$ を 0 を含む自然数全体の集合，$\mathbb{R}^+$ を非負実数全体の集合とする．

定義 1 時間ペトリネット (Time Petri Nets)[3)] は，プレースと呼ばれるノード（中が白い丸で記述される），トランジションと呼ばれるノード（四角で記述される），およびプレースとトランジション間を結ぶアークと呼ばれる有向枝を持つ有向 2 部グラフであり，各トランジション t に 2 つの属性 $d_{min}(t), d_{max}(t)$ を付加したものである．各アークは重みと呼ばれる正整数を属性として持ち，各プレースはトークン（黒丸で記述される）を一般に複数個持つことができる．プレース p からトランジション t へ入るアークがあるとき，p を t の入力プレースと呼び，トランジション t からプレース p へ出るアークがあるとき，p を t の出力プレースと呼ぶ．t の全ての入力プレースに対応するアークの重み以上の個数のトークンが存在するならば，t は発火可能であると呼ぶ．t が発火するとは，t の全ての入力プレースから対応するアークの重み分だけトークンを取り除き，全ての出力プレースの対応するアークの重みだけトークンを置く動作のことであると定義する．発火可能なトランジション t は発火可能になってからの経過時間 $\theta(t)$ を保持し，発火不能にならない限り $d_{min}(t) \leq \theta(t) \leq d_{max}(t)$ を満たすいずれかの時刻に発火するものとする． □

定義 2 時間ペトリネットにトランジション間の優先権 $\succ \subseteq T \times T$ を付与したものを優先権付き時間ペトリネット (PrTPN)[7)] と定義する．優先権付き時間ペトリネットでは，2 つのトランジション $t_1, t_2 \in T$ が同時に発火可能なとき，もし $t_1 \succ t_2$ ならば，t_1 のみが発火できるものとする． □

定義 3 優先権付き時間ペトリネットに，ストップウォッチアーク $Sw \subseteq T \times P$ およびその重み付け関数 $W_{sw} : Sw \mapsto \mathbb{N}$ を付与したものを優先権付きストップウォッチペトリネット (PrSwPN)[8)] と定義する．優先権付きストップウォッチペトリネットでは，マーキング M においてトランジション t の全てのストップウォッチアーク (t,p) に対して $M(p) \geq W_{sw}(t,p)$ ならば，t は最後に発火可能になってからの時間 $\theta(t)$ を更新し続ける．さもなければ，t は $\theta(t)$ の値の更新を停止する． □

3. 複数タスク動作仕様

本研究では，周期的に到着する入力イベントに応じて起動するタスクと，他のタスクの終了に応じて起動するなど，一定の制御構造に従って起動するタスクが混在したマルチタスクシステムの動作仕様を扱う．

複数タスクの順序関係や並列・選択などの制御構造は，一般に次に定義されるタスクグラフ TG で定義される．

定義 4 タスクグラフ TG は 9 つ組 $TG = (T, R, C, \rightarrow, a, c, d, rr, sc)$ である．ここで，T はタスクの有限集合，R はリソースの有限集合，C は制御ノードの有限集合，$\rightarrow \subseteq (T \times C) \cup (C \times T)$ は依存関係，$a : C \mapsto \{input_\lambda, par, join, choice, endchoice, output\}$ (ただし，$\lambda \in \mathbb{R}^+$) は制御ノードの種類を返す関数，$c : T \mapsto \mathbb{R}^+$ は各タスクの最悪実行時間を返す関数，$d : T \mapsto \mathbb{R}^+$ は各タスクの相対デッドラインを返す関数，$rr : T \mapsto 2^R$ は各タスクの実行に必要なリソースの集合を返す関数，$sc : R \mapsto \{\mathrm{FIFO}, \mathrm{FP}_\gamma\}$ は各リソースのスケジューリング方式，$\gamma : T \mapsto \mathbb{N}$ は各タスクへの優先度割り当て関数である． □

$N = C \cup T$ とし，$n \in N$ をタスクグラフ TG のノードと呼ぶ．$G = (N, \rightarrow)$ は DAG(閉路の無い有向グラフ) となっているものとする．$n_1, n_2 \in N$ のとき，$(n_1, n_2) \in \rightarrow$ を $n_1 \rightarrow n_2$ と書き，n_1 は n_2 に先行する，n_2 は n_1 に後続すると呼ぶことにする．

任意の制御ノード $c \in C$ の動作意味は表 1 に示すとおりと定義する．

任意のタスク $\tau \in T$ の動作は次の通りである．まず初期状態では起動待ち状態であり，起動されると，τ

表 1 制御ノードの動作意味

$a(c)$	先行タスクの待ち条件	後続タスクの起動方法
$input_\lambda$	なし	一定周期 λ
$choice$	一つが終了するまで待つ	いずれか一つを選択
par	一つが終了するまで待つ	全てを起動
$endchoice$	いずれかが終了するまで待つ	一つ
$join$	すべてが終了するまで待つ	一つ
$output$	一つが終了するまで待つ	なし

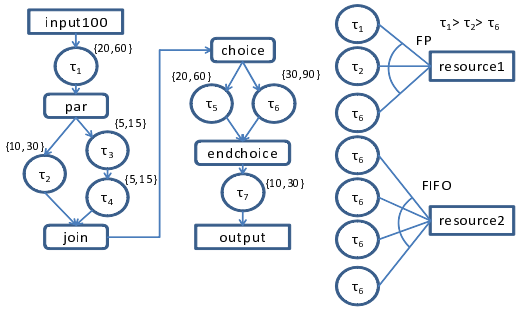


図 1 例題のタスクグラフ
Fig. 1 Example of Task Graph

の実行に必要な全てのリソース $r \in rr(\tau)$ の獲得待ち状態となる．各リソース $r \in R$ 獲得の際には，(一般にプリエンプティブな) スケジューリング方式 $sc(r)$ に従って他のタスクとのリソース競合を解決し，リソースを獲得できたら，実行状態に遷移する．実行状態において，もし他のタスクに $rr(\tau)$ に属するいずれかのリソースを横取りされたならば，中断状態に遷移し，再度必要リソースをすべて獲得できたならば実行状態に戻る．実行状態に滞在した時間の合計が最悪実行時間 $c(\tau)$ に達したらタスクの実行を終了し，制御構造に指定された後続のタスクを起動し，起動待ち状態に戻る．相対デッドライン $d(\tau)$ は，タスク τ が起動してから終了するまでの時間が $d(\tau)$ を超えてはならないという時間制約を表している．

任意のリソース r に対して，リソーススケジューリング方式 $sc(r) = \mathrm{FIFO}$ ならば，リソース r は早い者勝ちスケジューリング，すなわち，先にリソースを要求したタスクがリソースを獲得するものとし，$sc(r) = \mathrm{FP}_\gamma$ ならば，各タスク τ について優先度 $\gamma(\tau)$ を割り当てた固定優先度プリエンプティブスケジューリングで最も優先度 $\gamma(\tau)$ の大きいタスク τ がリソース r を獲得するものとする．この際，もし既に r を獲得していた別のタスク τ' があり，かつ $\gamma(\tau') < \gamma(\tau)$ ならば，τ' はリソースをいったん解放して中断状態とし，τ がリソースを獲得する (プリエンプション)．

例 1 タスクグラフの例を**図 1** に示す．図 1 において，白い丸はタスク（内側にタスク名の表記），角の丸い四角は制御ノード (内側に選択 (choice)，並列 (par)，選択合流 (endchoice)，並列合流 (join) のいずれかの表記)，黒い二重丸は入力ノード，黒い丸は出力ノード，タスク τ と制御ノード c を結ぶ矢印は依存関係を表す．2 つのタスク τ_1, τ_2 の間に矢印がある場合は，$\tau_1 \to c$ かつ $c \to \tau_2$ なる制御ノード c(ただし，$a(c) = choice$) が省略されているものとする．各タスク τ には最悪実行時間と相対デッドラインの対 $(c(\tau), d(\tau))$ が属性として付与されている．角のとがった四角はリソース (内側にリソース名の表記)，タスク τ とリソース r を結ぶ線は $r \in rr(\tau)$ である関係を表し，各リソース r には属性としてスケジューリング方式 (FIFO または FP) が付与されており，FP の場合はさらに各タスクの固定優先度が併記されている．
図 1 の例は次のように動作する．まず，一定周期 λ で到着する入力によってタスク τ_1 が起動する．τ_1 が終了すると，τ_2 と τ_3 が並列に起動する．τ_3 が終了すると τ_4 が起動する．τ_2 と τ_4 が共に終了すると，τ_5 と τ_6 のいずれか一方が選択されて起動する．τ_5 と τ_6 のいずれかが終了すると τ_7 が起動する．τ_7 が終了すると外部出力を行う．タスク τ_1, τ_2, τ_6 は $\tau_1 > \tau_2 > \tau_6$ の順の固定優先度プリエンプティブスケジューリングでリソース r_1 を共有し，$\tau_3, \tau_4, \tau_5, \tau_7$ は早い者勝ちスケジューリングでリソース r_2 を共有する．各タスクの最悪実行時間はそれぞれ $c(\tau_1) = 20, c(\tau_2) = 10, c(\tau_3) = 5, c(\tau_4) = 5, c(\tau_5) = 20, c(\tau_6) = 30, c(\tau_7) = 10$ であり，相対デッドラインはそれぞれ $d(\tau_1) = 60, d(\tau_2) = 30, d(\tau_3) = 15, d(\tau_4) = 15, d(\tau_5) = 60, d(\tau_6) = 90, d(\tau_7) = 30$ が指定されている． □

4. 複数タスク動作仕様から PrSwPN への変換

4.1 問題定義および変換の方針

与えられたタスクグラフ TG から，TG の各動作（任意のタスク τ_i のリリース，実行開始，実行中断，実行再開，実行終了，リソースの獲得・解放）の実行系列の集合（時間付きトレース集合）と，対応するトランジションの実行時刻も含めた発火系列の集合が等しいような PrSwPN に変換する問題を考える．この変換を，まず，タスク，スケジューラなどのシステムの構成要素それぞれに対して，対応する PrSwPN による部分モデルを構成し，それらを結合することにより全体の動作を表現するモデルに変換する方針で行うことを考える．

4.2 各タスクの PrSwPN への変換

タスクグラフの各タスク $\tau_i = (c(\tau_i), d(\tau_i))$ に対して，図 2(a) のような構造の PrSwPN を対応させる．

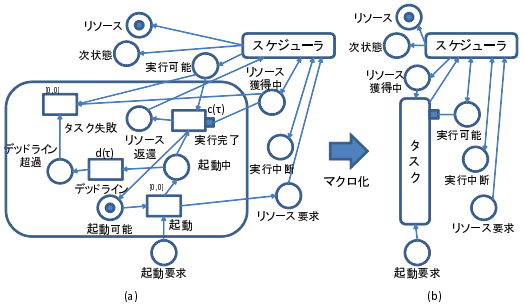


図 2 PrSwPN で表したタスク
Fig. 2 Task expressed by PrSwPN

図 2(a) の PrSwPN は次の様に動作する．

タスクの起動は起動トランジションが発火することによって表現される．起動トランジションの入力プレースが起動要求プレース，起動可能プレースで出力プレースが起動中プレース，リソース要求プレースになる．

タスクの実行はスケジューラによってリソース要求プレースから実行可能プレースとリソース獲得中プレースにトークンを移動することによって表現される．タスクの実行完了は実行完了トランジションが発火することによって表現される．実行完了トランジションの入力プレースが実行中プレースと実行可能プレースと起動中プレースで出力プレースがリソース返還プレースになる．スケジューラによって，リソース返還プレースから次状態プレースとリソースプレースへトークンを移動させることによって，リソースの返還と次状態への遷移が表現される．

タスクのデッドラインの超過はデッドライントランジションが発火することによって表現される．デッドライントランジションの入力プレースが，起動中プレースで出力プレースがデッドライン超過プレースになる．タスクの失敗トランジションが発火するとタスクが処理に失敗したことが表現される．入力プレースがデッドライン超過プレース，実行可能プレース，リソース獲得中プレースである．

4.3 スケジューラのモデル化

タスクグラフの各リソース R に対して，スケジューラ $sc(r)$ を PrSwPN に変換する．

$sc(r)$ =FIFO の場合はリソース要求が早く到着した順に，タスクを実行中にさせる必要がある．そのため，トランジションとリソースの組によって EDF キューを表現し，これを用いて早く到着した順にタスクを実行中状態にさせる．また，先に到着したタスクを追い越さないため，追い越し禁止を表すプレースも用意する．上記の方針で変換したモデルが**図 3** である．

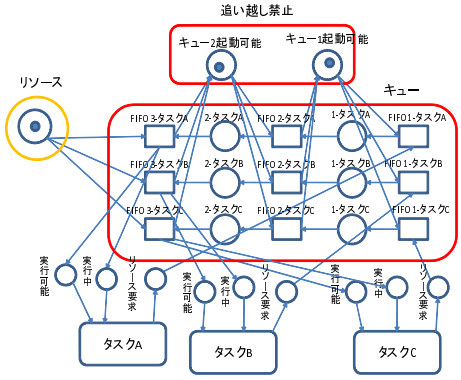
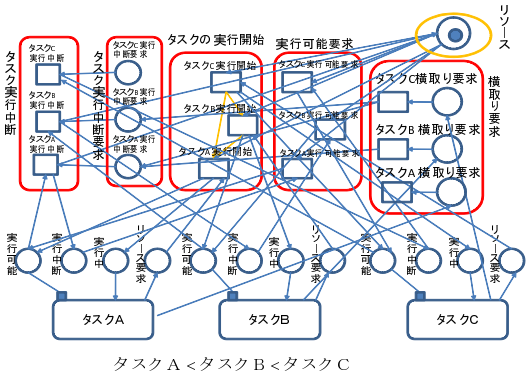


図 3 PrSwPN で表した FIFO スケジューラ
Fig. 3 FIFO scheduler expressed by PrSwTPN



タスク A < タスク B < タスク C

図 4 PrSwPN で表した FP スケジューラ
Fig. 4 FP scheduler expressed by PrSwPN

$sc(r)$ =FP の場合は**図 4** のように表現する．横取り要求プレースは，トークンがある場合，優先度の低いタスクへ横取り要求をしていることを表すプレースである．このプレースはタスクの起動トランジションの出力プレースであるものとする．横取り要求トランジションをそれぞれのタスクに対して導入する．入力プレースは横取り要求プレース，出力プレースは実行中断要求プレースである．タスクの実行中断要求プレースはトークンがある場合，タスクの実行中断を表すプレースである．各タスクを実行中断にするトランジションをそれぞれのタスクに対して導入する．入力プレースは実行中断要求プレースと実行可能プレースであり，出力プレースは実行中断プレースである．実行可能トランジションはタスクの実行中断から実行可能にすることを表している．入力プレースは実行中断プレースとリソースプレースであり，出力プレースは実行可能プレースである．実行開始トランジションはタスクの実行開始を表している．入力プレースはリソー

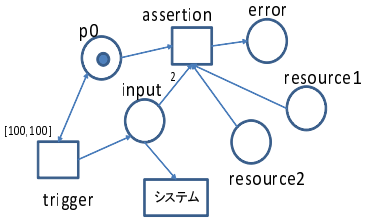


図 6 スループット検証を行うための PrSwPN 表現
Fig. 6 PrSwPN representation for verifying throughput requirement

ス要求プレースであり，出力プレースは実行中プレースと実行中断プレースである．また，このトランジションの組に優先度を割り振る．優先度の割り振りはタスクの優先順位と対応させる．例えば $\gamma(\tau_1) > \gamma(\tau_2)$ ならば，(τ_1 の実行開始トランジション)$\succ$(τ_2 の実行開始トランジション) とする．

4.4 制御ノードのモデル化

制御ノード（*par*, *join*, *choice*, *endchoice*, *input*, *endinput*）を，それぞれ PrSwPN で表したモデルが**図 5** になる．図 5 のように，各タスクのモデルを結合することにより，タスクグラフ全体を表現する PrSwPN を構成する．

5. 複数タスク動作仕様の性能検証

5.1 性能検証問題

性能検証問題とは，性能要求と性能モデルから，性能要求を満たすかどうかの情報を得る問題である．本研究では性能要求として，単位時間に処理できるデータ数を表すスループット要求に着目し，周期的に到着する入力が遅滞なく処理できているかどうかの要求を扱う．そしてスループット要求を満たすかどうかを性能検証問題と定義するスループット要求と，6 章で変換した PrSwPN から性能検証を行い，スループット要求を満たすか否かの情報を得る．

5.2 性能検証手法

提案する性能検証手法では，性能検証モデルに対してスループット検証を行う．本研究ではスループット要求の検証のため，スループット要求が満たされない状態を検出するアサーションを性能検証モデル追加してスループットを検証する．入力が入ってくるプレースに対して 2 個以上プレースがたまったらデッドロック状態になるようなアサーションを付加する．デッドロック状態にする方法としては，リソースのプレースからトークンを抜く方法を採用する．以上の方針で，アサーションを PrSwPN を用いて表したモデルが**図 6** になる．

6. 検 証 例

図 1 の例題について，性能検証モデルへの変換を行い，そのモデルに対して，アサーションを付加させる事によって，スループット検証を行った．また，resource1 に対するタスクの優先度をそれぞれ**表 2** の様に割りあて，どの場合の優先度割りあてが一番良い性能が得られるかについての検証も行った．ペトリネットの規模としては，89 プレース数，68 トランジション数程度のものとなった．

表 2 固定優先度の割り当て
Table 2 Fixed Priority Assignment for Cases 1,2 and 3

ケース 1	$\tau_1 > \tau_2 > \tau_6$
ケース 2	$\tau_2 > \tau_1 > \tau_6$
ケース 3	$\tau_6 > \tau_2 > \tau_1$

6.1 検 証 環 境

生成した検証モデルの検証には PrSWPN 検証ツール TINA[10] を使用する．TINA は PrSWPN をグラフィカルに描写することができ，作成した PrSWPN をテキストファイルに変換することも可能である．ランダムシミュレータでは，トランジションの発火時間をランダムに選んで解析を行うことができる．到達可能性解析では全てのマーキングを調べ，PrSWPN がデッドロックに陥るか否かを解析することができる．本研究ではスループット検証問題をデッドロックの問題に帰着させ，TINA でデッドロック検証を行うことで性能検証を行う．

6.2 検 証 結 果

それぞれのケースについて検証を行った．ケース 1 と 3 についての結果をそれぞれ**表 3**，**表 4** に示す．

表 3 ケース 1 の検証結果
Table 3 Verification result for case1

スループット要求（入力/s）	スループット要求を満たすか否か？	検証時間	状態数
0.6	Yes	0.016	115
0.67	Yes	0.016	115
0.75	Yes	0.016	115
0.86	Yes	0.031	127
0.92	Yes	0.047	119
1	No	0.438	1808

6.3 考 察

表 3，表 4 の結果より，この場合の条件では，ケース 3 にような優先度の割り当てにすれば，より厳しいスループット要求でも満たすことができるため，性能が向上するということが確認できた．

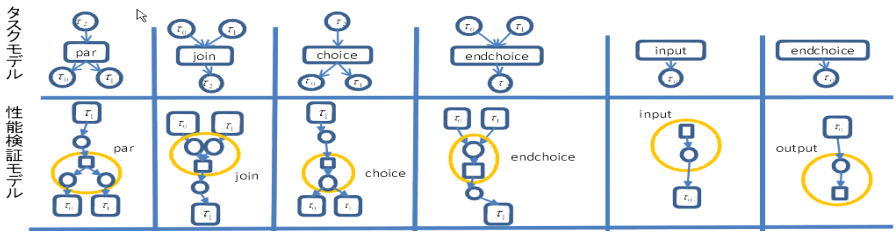


図 5 PrSwPN で表した制御ノード
Fig. 5 Control node with PrSwPN

表 4 ケース 3 の検証結果
Table 4 Verification result for case3

スループット要求（入力/s）	スループット要求を満たすか否か？	検証時間	状態数
0.6	Yes	0.016	115
0.67	Yes	0.016	115
0.75	Yes	0.031	115
0.86	Yes	0.016	130
0.92	Yes	0.016	115
1	Yes	0.031	162
1.09	No	22.594	68148

7. あとがき

プリエンプティブスケジューリングでリソースを共有し，分岐や並列などの制御構造を持つ複数タスク動作仕様から性能評価モデルである PrSwPN への変換手法を提案し，作成した PrSwPN を用いて，さまざまなスケジュールに対してスループットの性能検証を行う手法を提案した．従来の研究ではプロセッサスケジューリングしか考慮していないものがほとんどであったが，提案手法により，さまざまなリソースをさまざまなスケジューリング方式で共有するシステムの検証が可能となる．また，PrSwPN を用いることによりプリエンプティブスケジューリングを扱うことが可能となる．今後の予定としては，ラウンドロビンや EDF といったスケジューリング方式にも対応させることや，タスクの優先度逆転などにも対応させることが考えられる．また，提案した変換手法のツールの実装なども今後の課題に挙げられる．

参 考 文 献

1) Balsamo, S., Di Marco, A., Inverardi, P. and Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey, *IEEE Trans. Softw. Eng.*, Vol. 30, No. 5, pp. 295–310 (2004).
2) Balsamo, S. and Marzolla, M.: Performance evaluation of UML software architectures with multiclass Queueing Network models, *Proc. of 5th Int. Workshop on Software and Performance*, ACM Press, pp. 37–42 (2005).
3) Berthomieu, B. and Diaz, M.: Modeling and Verification of Time Dependent Systems Using Time Petri Nets, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 3, pp. 259–273 (1991).
4) Amnell, T., Fersman, E., Mokrushin, L., Pettersson, P. and Yi, W.: TIMES: a Tool for Schedulability Analysis and Code Generation of Real-Time Systems, *Proc. of the 1st Int. Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, Lecture Notes in Computer Science, Vol.2791, Springer-Verlag, pp. 60–72 (2004).
5) Tanimoto, T., Yamaguchi, S., Nakata, A. and Higashino, T.: A Real-Time Budgeting Method for Module-Level-Pipelined Bus Based System using Bus Scenarios, *Proc. of 43rd Design Automation Conf. (DAC 2006)*, ACM Press, pp. 37–42 (2006).
6) 百々太市, 山脇弘, 中田明夫: リソーススケジューリングを考慮した UML MARTE 振る舞い仕様の性能検証, 情処研報 Vol. 2010-EMB-16, No.35, 情報処理学会 (2010).
7) Berthomieu, B., Peres, F. and Vernadat, F.: Model Checking Bounded Prioritized Time Petri Nets, *Proc. of 5th Int. Symp. on Automated Technology for Verification and Analysis (ATVA 2007)*, Lecture Notes in Computer Science, Vol. 4762, Springer-Verlag, pp. 523–532 (2007).
8) Berthomieu, B., Lime, D., Roux, O. H. and Vernadat, F.: Reachability Problems and Abstract State Spaces for Time Petri Nets with Stopwatches, *Journal of Discrete Event Dynamic Systems*, Vol. 17, pp. 133–158 (2007).
9) Liu, C. L. and Layland, J. W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol. 20, No. 1, pp. 46–61 (1973).
10) Berthomieu, B. and Vernadat, F.: Time Petri Nets Analysis with TINA, *Proc. of 3rd Int. Conf. on the Quantitative Evaluation of Systems (QEST 2006)*, IEEE Computer Society Press (2006).