情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

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

百々 太市[†] 山脇 弘[††,‡] 中田 明夫[††]

UML MARTE の記法で書かれた振る舞い仕様(シーケンス図)とリソースの割り当て情報から優先権付き時間ペトリネットによるリソーススケジューリングを考慮した性能検証モデルを生成し，組込みシステムなど性能制約の厳しいシステム設計において，設計初期段階における性能検証を行う手法を提案する．提案手法により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い，提案手法の有効性を示す．

Performance Verification of UML MARTE Behavioral Specification with Resource Scheduling

Taichi Dodo[†], Hiromu Yamawaki[††,‡] and Akio Nakata[††]

We propose a method to synthesize a formal model written in Prioritized Time Petri Nets from a behavioral specification written in a UML MARTE sequence diagram and resource allocation information, and to verify its performance requirement in an early phase of system design process. We also show that throughput requirement can be verified effectively for two small UML MARTE sequence diagram examples by analyzing the synthesized model using an existing model checking tool.

†広島市立大学 情報科学部
Faculty of Information Sciences, Hiroshima City University
††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
‡現在, NTT ソフトウェア(株)勤務
Presently with NTT Software Corp.

1. まえがき

伝統的なソフトウェア開発手法においてはソフトウェア機能の正しさに焦点が置かれており，設計の途中で性能が大きく変化することを考慮していない．そのような開発手法においては性能問題が開発プロセスの後半で見つかることが多く，設計の見直しなど手戻りが生じる．そこで，開発プロセスの早期の段階で，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予測し，検証することを指す．最近では，機器に組み込まれるソフトウェアである組込みソフトウェアの需要も増えている．特に，組込みソフトウェアにおいては実時間制約を考慮した性能検証[1]を行うことが望まれている．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．ソフトウェア仕様モデルは，標準化されている UML[2]で書かれていることが多い． UML に対しては UML MARTE[3]など組込みソフトウェア用の拡張も提案されており，組込みソフトウェアでの性能検証手法の開発も課題となっている．ソフトウェア性能検証モデルには，待ち行列ネットワークなどの確率的モデル[1]と時間ペトリネット[4][5]などの確定的モデルに分類される．確率的モデルは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の性能を考慮する．時間制約やリソース制約が厳しい組込みソフトウェアを対象とする場合，最悪時の考慮が重要となる．

本研究では UML MARTE の記法で書かれた振る舞い仕様(シーケンス図)とリソースの割り当てから，優先権付き時間ペトリネット(PrTPN)[6]を用いてリソーススケジューリング[7][8]を考慮した性能検証モデルを生成する手法を提案する．生成した PrTPN を既存の検証ツールによって検証することにより，最悪時を考慮した性能の検証を行うことが可能になる．本研究では，遠隔制御ロボットシステムおよび書籍発注システムの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検証を行うことによって，提案手法の有効性を示す．

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

2. UML MARTE

2.1 UML

UML はソフトウェアの成果物を仕様化，図式化するときに使用される言語である．UML には以下の 3 つの特徴が挙げられる．

① 表現力が高く理解が容易，多様な表記法を使用することにより曖昧さが排除されている．

② すべての工程で用いる表記法が一貫している．

③ 世界中で普及している共通の言語である．

UML には対象の構造に着目してモデリングする構造図とモデリングする対象の振る舞いに着目してモデリングする振る舞い図の 2 種類がある．構造図にはクラス図，オブジェクト図，コンポーネント図，配置図などがある．振る舞い図には，ユースケース図，状態マシン図，シーケンス図などがある．振る舞いとはモデリングの対象の動的な仕様を定義したものを指す．本研究では振る舞い図の中でシーケンス図を扱う．

2.2 UML MARTE

UML MARTE とは UML2.0 を組込み用に拡張したプロファイルである．UML MARTE は組込みソフトウェアだけでなく，ハードウェアを含めた組込みソフトウェアのモデル駆動開発を対象としており，両方の仕様と設計を扱うことができる．UML MARTE のシーケンス図の例を図 1 で挙げる．

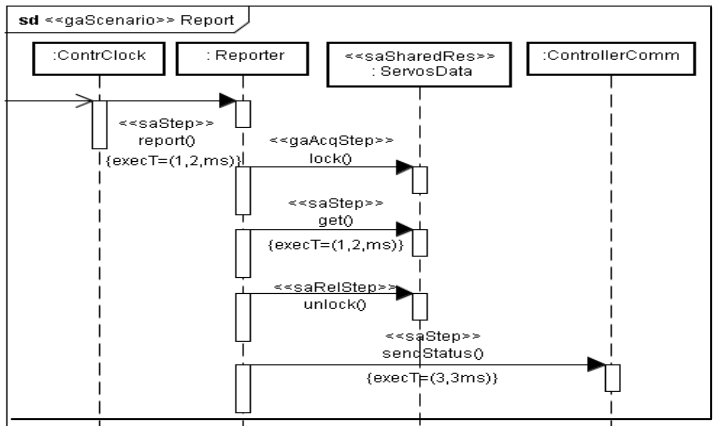


図 1 UML MARTE の記法で書かれた振る舞い仕様の例

UML MARTE では，メッセージ，オブジェクトなどの UML 構成要素に対して，UML2.0 で定義されているステレオタイプ属性を付加し，リソースとの関連や時間制約を表現する．図 1 の例では，オブジェクト ServosData が共有リソース，メッセージ report が計算リソースを使った単位処理（ステップ），lock がリソース獲得処理を表していることを示している．また，各メッセージの時間制約が execT で表されている．

3. 優先権付き時間ペトリネット(PrTPN)

ペトリネット(PN)は並列的・非同期的・分散的なシステムを表すための数学的モデルである．PN に確定的な時間の概念を導入したものが時間ペトリネット(TPN)という．図 2 に例を示す．

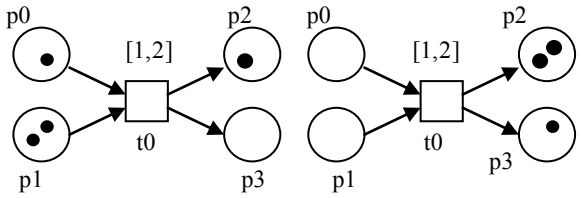


図 2 TPN の例(左：発火前，右：発火後)

プレースは○，トランジションは□，アークは→，トークンは●で表される．p0 と p1 がトランジション t0 の入力プレースで，p2 と p3 が出力プレースである．アークには，重み（正の整数）がつけられ，重みが書いていない場合は 1 とし，数字は省略する．ペトリネットの実行は発火で表現することができ，各入力プレースのトークンの数がアークの重み以上であれば，発火することができる．TPN の場合，トランジションに[1,2]のような時間情報を付加することができ，この例の場合，発火可能となってから 1 秒以上 2 秒以内に発火しなければならない．発火すると，各入力プレースのトークンがアークの重みだけ抜き取られ，各出力プレースにアークの重みだけのトークンが加えられる．それによりシステムの振る舞いを表すことができる．

優先権付きペトリネット(PrPN)とは任意のトランジションの組(t1,t2)に対して，優先順位を設定可能であり，t1 と t2 が同時に発火可能な場合には，より優先度の低いトランジションが発火できないような拡張ペトリネットのことをいう．TPN に優先権の概念を導入したもの PrTPN という[6]

4. リアルタイムシステム

リアルタイムシステムとは，設定された時間通りに動作しなければならないコンピ

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

ュータシステムのことをいう．通常，リアルタイムシステムではイベント(event)の発生ごとにタスクがリリースされ，定められたサービスを実行しなければならない．そしてある時刻までにサービスを完了しなければならない．イベントがリアルタイム処理系に発生すると，このイベントに関連づけられたタスクが実行を行う．また，タスクの実行のためには実行に必要となるリソースが全て使用可能であることが必須である．図 3 にタスクの実行状況を示す．

図 3 タスクの実行状況

イベントが発生すると，そのイベントに関連づけられたタスクが到着(arrive)という．このことをタスクがリリース(release)されると呼び，この時点を矢印“↑”で示す．四角“■”の部分はタスクの処理を表している．start はタスクの処理の開始時刻，final はタスクの処理の完了時刻を表す．

タスクの最悪実行時間とはタスクが実行を開始してから実行を完了するまでにかかる一番長い時間のことをいう，図 3 においては start から final までの時間が最悪実行時間に対応する．release から deadline までの時間のことを相対デッドラインという．一方，現在時刻にタスクの相対デッドラインを加えたものをタスクの絶対デッドラインという．タスクの処理の完了時刻 final がデッドライン以下であるとき，タスクはデッドラインを満たすという．

5. スケジューリング方式

スケジューリング方式とは，一般に少数のリソースを共有する複数のタスク群において，すべてのタスクがデッドラインを満たすように，各タスクをどのような順番でいつからいつまで実行するかを決定する方式である．ここでリソースとしては，計算リソース（プロセッサ）のみならず，通信リソース（ネットワークや共有変数など）も含めて考慮する．本論文では以下の 3 つのスケジューリング方式を扱う．

① 早い者勝ち

早い者勝ち(FCFS: First Come First Served)とは先に到着したタスクが即座にリソースを獲得して実行するというスケジューリング方式である．

② 固定優先度スケジューリング

固定優先度スケジューリング(FP: Fixed Priority scheduling)とは，各タスクに固定の優先度を割り当て，複数のタスクがリソースについて競合した場合には割り当てられた固定優先度の大きいタスクがリソースを獲得し実行するというスケジューリング方式である．

③ EDF スケジューリング

EDF(EDF: Earliest Deadline First)スケジューリングとはタスクの優先度が時間の経過と共に動的に変化するスケジューリング方式の一つである．絶対デッドラインが最も早く訪れるタスクに高い優先度を割り当てるスケジューリング方式のことを EDF スケジューリングという．

なお，実行中の他のタスクからのリソースの横取り（プリエンプション）を許すか否か(プリエンプティブ/ノンプリエンプティブ)によるスケジューリング方式の分類が可能であるが，本論文で扱うスケジューリング方式はすべてノンプリエンプティブであるとする．

性能検証モデルを生成する際，モデルの制約により時間経過に応じて動的に優先度を変化させるスケジューリング方式を記述出来ない．本研究では，ノンプリエイティブな EDF スケジューリング方式を近似するスケジューリング方式を提案し，それをモデル化する．

6. UML から優先権付き時間ペトリネット(PrTPN)への変換

6.1 変換への前提条件

変換前に必要なものは UML MARTE の記法で記述されたシーケンス図とリソース割り当て情報である．これらより，TPN の性能評価モデルを作成する．以下の条件を満たすような UML MARTE の記法で記述されたシーケンス図を対象とする．

① 全体の動作シーケンスは単一の外部入力により起動する
② オブジェクト，リソースの総数が静的に決まっている．
③ 再帰呼び出し処理がない．
④ 値が動的に変わる変数を使用していない．

オブジェクトのリソース割り当て情報は図 4 のように記述する．リソース割り当て情報に関しては，UML MARTE の記法で書かれてなくても，どのオブジェクトがどのリソースを利用するのかの情報があればよい．

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

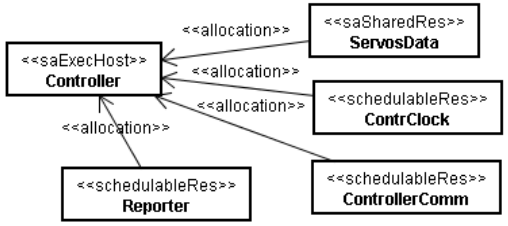


図 4　UML MARTE の記法によるリソース割り当て情報

6.2 変換方針

冗長性は持っていても,機械的に変換できるようにするという変換方針で UML MARTE およびリソース割り当て情報から PrTPN への変換を行う．まず，各オブジェクトのメッセージの送信前，送信後，受信前，受信後などの状態に対応するプレースを導入する．トークンがこれらのプレースを移動することによりオブジェクトの状態遷移を表す．次に，計算リソース，通信リソース，共有リソースなどのリソースに対応したプレースを導入し，これらのプレースにトークンが有れば，対応するリソースが利用可能であることを表す．最後にシーケンス図の中の動作はトランジションを導入することによって表現する．大まかな変換手順は以下の通りである：

① 各オブジェクトの状態をプレースで表す
② 必要リソースに対応するプレースで表す
③ メッセージ（動作）に対応するトランジションで表す．実行時間がある場合は，トランジションに時間情報を追加する
④ メッセージ送信直前の状態を表すプレースが入力プレース，メッセージ送信直後の状態が出力プレースになるように結ぶ
⑤ メッセージを送信するのに必要なリソースが入力プレース，メッセージを受信後，解放するリソースが出力プレースになるように結ぶ．
⑥ プレースにトークンを配置する．
（ア）最初から生成されているオブジェクトの初期状態にトークンを置く．
（イ）リソースに対応するプレースにトークンを置く．
（ウ）オブジェクトが生成される場合はスレッドリソースにトークンを置く．

6.3 変換ルール

6.3.1 タスクの変換

本研究では，シーケンス図におけるメッセージ送信により行われる動作をリアルタイムタスクに対応させる．タスクには処理時間があり，それはタスクがリリースされてからデッドラインを迎えるまでに処理を完了させなければならない．ここでのタスクのリリースはメッセージ送信直前の状態とする．また，タスクの処理に必要なリソースは処理完了後，リソースを解放することとする．このとき，各タスクを次の方針で PrTPN によってモデル化する．

(1) メッセージ送信側，受信側それぞれのオブジェクトについて，メッセージ送信直前の状態，および，メッセージ送信直後の状態を表すプレースを導入する．すべてのオブジェクトのメッセージ送信直前の状態を表すプレースにトークンが存在するときにタスクがリリースされ，タスクの実行が終了するとメッセージ送信直後の状態を表す各プレースにトークンが移動するものとする．
(2) タスクの各状態（リリース中，実行中，デッドライン超過）を表すプレースを導入する．トークンがこれらのプレースを移動することにより，タスクの状態遷移を表現する．
(3) タスクがリリースされてからの経過時間を測定するタイマーを起動するプレースを導入する．時間ペトリネットの各トランジションで計測される時間は入力プレースのトークンの移動によりリセットされるため，タスクの状態遷移と独立してデッドラインまでの時間を計測するためにこのようなプレースを導入する．
(4) タスクの状態遷移（リリース，実行開始，実行完了，デッドライン超過）を表すトランジションを導入する．タスクの到着を表すトランジションは入力プレースとしてメッセージ送信直前の状態を表すプレースをもち，出力プレースとしてタスクのタイマーを表すプレースとタスクのリリースを表すプレースをもつ．トランジションの発火によってタスクの到着を表す．タスクの処理開始を表すトランジションは入力プレースとしてタスクのリリースを表すプレースとタスクの実行に必要な各リソースを表すプレースをもち，出力プレースとしてタスクの実行中を表すプレースをもつ．トランジションの発火によってタスクの実行開始を表す．タスクの実行完了を表すトランジションは入力プレースとしてタスクの実行中を表すプレースをもち，出力プレースとしてリソースを表すプレースとメッセージ送信直後の状態を表すプレースをもつ．トランジションの発火によってタスクが処理を完了したことを表す．タスクの実行がデッドラインを超過したことを表すトランジションは入力プレースとしてタスクのタイマーを起動するプレースをもち，出力プレースとしてタスクのデッドラインを表すプレースをもつ．トランジションの発火により，タスクがデッドラインを迎えたことを表す．

上記で構成された性能検証モデルを図 5 に示す．なお，以降ではタスクのモデルは図 5 の右のように表現する．

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

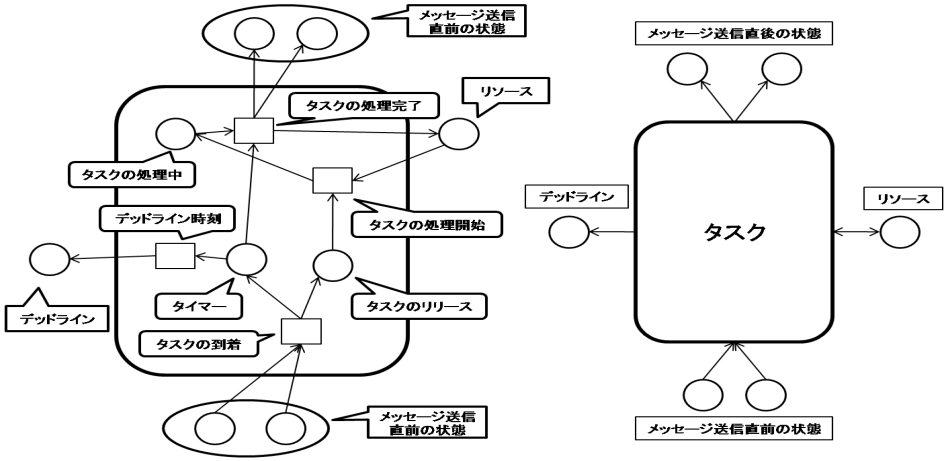


図 5 タスクの性能検証モデル

6.3.2 リソースの変換

リソースの表現に関して詳細に述べる．各オブジェクトが計算リソース，通信リソース，共有リソース(データ)，スレッドリソースを利用するので，リソースの表現が必要となる．計算リソースには CPU，プロセッサなどがあり，通信リソースには，バスなどがある．リソースはプレースで表し，リソースは 1 つのトークンを持っているとする．トークンがある場合は，処理でリソースを使用していて，リソースは利用不可の状態である．トークンがない場合は，リソースは利用されておらず，利用可能の状態であることを示している．

6.3.3 スケジューリング方式の変換

タスクはリソース競合が起こった際はスケジューリングすることによって問題を解決する．PrTPN の性能検証モデルにおけるスケジューリングの表現方法について詳細に述べる．早い者勝ちはペトリネットにおけるトランジションの発火がそのまま早い者勝ちを満たしているため通常の状態で早い者勝ちを満たしている．

固定優先度スケジューリングはタスクの実行開始を表すトランジションに固定の優先権を割り当てることによって表現する．優先権をタスクの性能検証モデルに割り振ったモデルを図 6 に示す．

EDF スケジューリングは，離散的な時間の経過を表すトランジションとプレースをそれぞれ用意し，それぞれに実行開始を表すトランジションを対応させ，それらに優先権を割り振っていくことによって動的に優先度が変化する動きを離散的に近似するスケジューリング方式を用いる．タスクの到着時間からデッドラインまでの時間を一定な周期(以下，スケジューリング周期)で区切り，デッドラインが遠くなるほど優先度を低く割り当てる．また，タスクの到着時刻はスケジューリング周期によって切り上げ，またデッドラインはスケジューリング周期によって切り捨てることとする．上記のスケジューリング方式の例を図 7 に示す．この例では，到着時刻が 7 単位時間，相対デッドラインが 34 単位時間，スケジューリング周期が 10 であったとき，タスクの到着時刻を 10 単位時間，デッドラインの時刻を 40 とし，10～20 単位時間区間，20～30 単位時間区間そして 30～40 単位時間区間にそれぞれデッドラインが遠くなるほど低く優先度を割り当てている．

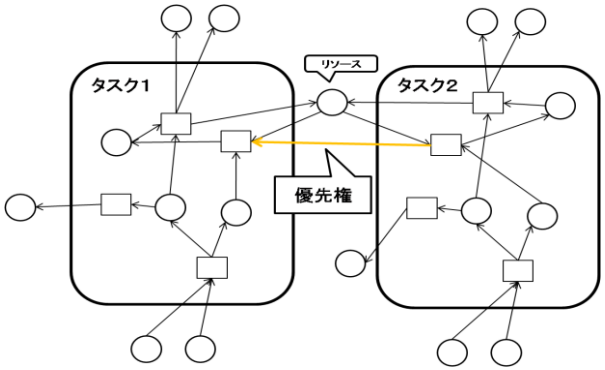


図 6 固定優先度スケジューリングのモデル化

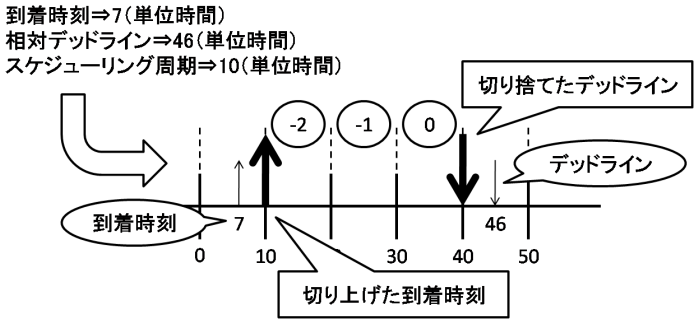


図 7 EDF スケジューリングの離散近似例

次に，モデル化の手順について説明する．タスクの到着時刻からデッドラインまでの時刻を，スケジューリング周期で発火するトランジションで区切り，各区間に離散的な時間経過を表すプレースを用意する．タスクの到着を表すトークンを各区間に離

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

散的な時間経過を表すプレースに移動させていくことによってデッドラインまでの時刻を離散的に得ることとする．そして，実行開始を表すトランジションを各区間に離散的な時間経過を表すプレースに対応させ，それらに優先度を割り当てることによって動的に優先度を変化させるモデルを得る.また,タスクの切り上げを表現するため，スケジューリング周期の時刻のときにだけ，プレースにトークンが存在するようなモデルを生成し，タスクの到着を表すトランジションに付加させる．図 7 の例をモデル化したものが図 8 になる．

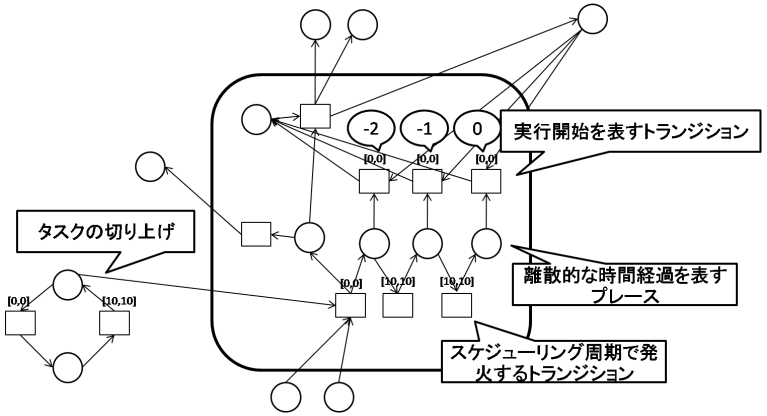


図 8　EDF スケジューリングの離散近似に対する性能検証モデル

6.3.4 変換ルール

UML MARTE シーケンス図の各構成要素の PrTPN への変換ルールについて説明する．紙面の都合により本論文ではオブジェクト間の同期処理についてのみ述べる．他の構成要素に対する変換ルールの詳細については[9]を参照されたい．

同期処理の場合,メッセージ受信後に各オブジェクトの状態が変化するとする.図 9 では A が B に同期メッセージを送っている例になる．この例では，太い黒線で状態が変化する．

変換の手順は以下のようになっている．

(ⅰ)各オブジェクトの状態をプレースで表す.

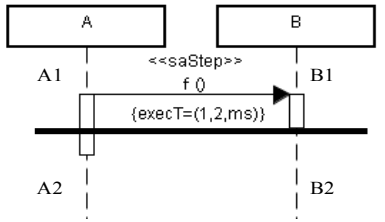
・A は A1，A2 の状態，B は B1，B2 の状態を持つので，プレースを生成

(ⅱ)必要リソースに対応するプレースで表す.

・A で CPU1 を利用し，B が CPU2 を利用するので，CPU1 と CPU2 のリソースを表すプレースを生成

(ⅲ)メッセージに対応するトランジションで表す．実行時間がある場合は，トランジションに時間情報を追加する．

・　同期メッセージ f をトランジションで表す．実行時間が指定されているのでタスクの処理完了を表すトランジションに時間情報(1，2)を付加する．



リソース割り当て
A→CPU1，B→CPU2

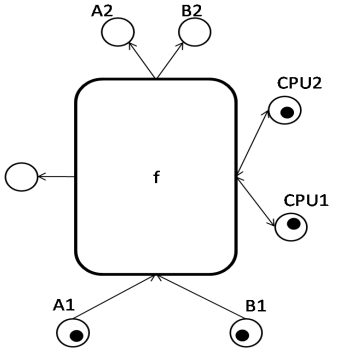


図 9　同期処理の変換

(ⅳ)メッセージ送信前の状態を表すプレースが入力プレース，メッセージを受信直後の状態が出力プレースになるように結ぶ.

・メッセージ送信前の状態を表すのは A1 と B1，メッセージ送信直後の状態を表すのは A2，B2 になるので，A1，B1 が入力プレース，A2，B2 が出力プレースになる.

(ⅴ)メッセージを送信するのに必要なリソースが入力プレース,メッセージを受信後,開放するリソースが出力プレースになるように結ぶ．CPU1 と CPU2 を両矢印で結ぶ.

(ⅵ) トークンを配置する

最初から生成されている A，B の初期状態 A1，A2 にトークンを置く．リソースに対応する CPU1，CPU2 にトークンを置く．

7. 振る舞い仕様の性能検証

7.1 性能検証問題

一般的な性能検証問題は，性能要求と性能モデルから性能検証を行い，性能要求を満たすかどうかの情報を得る．この場合入力が，性能要求と性能モデルになり，出力は，性能要求を満たすどうか?(yes/no)になる．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

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

うかを性能検証問題と定義する（図 11）．スループット要求と，6 章で変換した PrTPN 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．

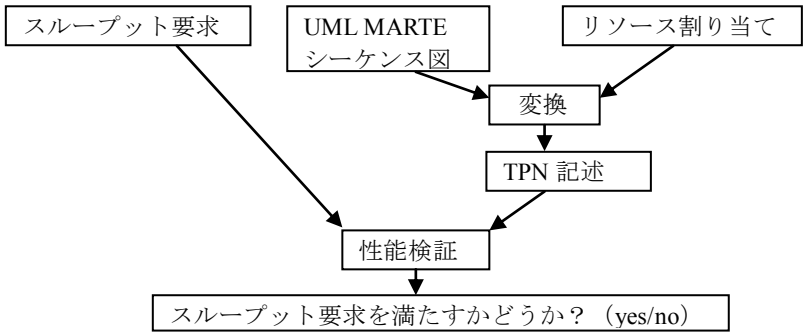


図 10　本研究での性能検証問題

7.2 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つける．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．具体例を図 13 に示す．Error プレースはエラーの状態を表すプレースで，トークンがある場合，現在エラー状態であることを示す．assertion トランジションはエラー処理を行うトランジションを表している．入力プレースが input，Resource1，Resource2，p0 で出力プレースが error になる．この例では，input プレースに trigger から 100ms ごとにトークンが入ってくる．つまり，スループットは 10 単位データ/秒である．ここに入ってくるトークンが 100ms 以内にトークンがシナリオの最後まで到達すれば問題はない．もし，input プレースにトークンが 2 つ以上たまれば，100ms 以内に処理できていないことになる．その場合 input プレースにトークンが 2 個以上たまるので，assertion トランジションにより即座に発火し，入力プレースと各リソースからトークンを抜き取り，error プレースにトークンが移る．

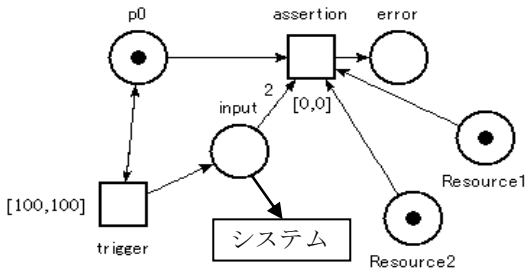


図 11　スループット検証を行うための PrTPN 表現

8. 検証例

8.1 検証対象とした例題

遠隔ロボットの例題と発注管理システムの例題を用いて変換を行った．遠隔ロボットの例題はステーション，ネットワーク(CAN バス)，コントローラ，ロボットアームの 4 つのハードウェアから構成されており，ステーションから遠隔にあるロボットを操作するシステムとなっている．ステーションは CAN バスを通して，コントローラと通信を行い，コントローラは内部バスを通してロボットアームと通信を行っている．検証に用いたシーケンス図は，CAN バスを通じてロボットアームの状態をステーションにレポートするシナリオを記述している．リソース割り当てとしては，コントローラとステーションにそれぞれ 1 つずつ CPU リソースが割り当てられているとする．

書籍発注システムの例題は，顧客が書籍を端末で欲しい書籍情報を取得して，欲しいものがあれば，本を注文する処理を行うシステムである．ここでの顧客オブジェクトは顧客の端末を表している．大まかな流れを説明する．顧客端末はまず注文画面を表示する前に，商品リストの在庫個数が他から更新されないようにロックをかける．注文画面を表示した後，欲しい商品情報を取得し，注文画面に表示する．その後，他からもアクセスできるようにアンロックをかける．注文の工程に移る．在庫があれば，注文し，注文明細に商品を追加する．もし，在庫がない場合はエラーを表示する．もし，カード払いの場合は手続きをする．リソース割り当ては，顧客端末とその他のオブジェクト群にそれぞれ 1 つずつ CPU リソースを割り当てる状況を仮定する．

8.2 検証環境

生成した検証モデルの検証には PrTPN 検証ツール TINA[10]を使用する．TINA は TPN をグラフィカルに描写することができ，作成した TPN 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，

Vol.2010-SLDM-144 No.35
Vol.2010-EMB-16 No.35
Vol.2010-MBL-53 No.35
Vol.2010-UBI-25 No.35
2010/3/27

TPN 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．

8.3 検証結果

遠隔ロボットと書籍発注システムの例題を用いて検証を行った．遠隔ロボットの例題において，スループット要求は 1(入力／s)とし，CPU リソースについてノンプリエンプティブ固定優先度スケジューリングを行うとして PrTPN への変換を行った．変換された PrTPN の規模は，79 プレース，65 トランジションとなった．この例題に対し，スループット要求 10Mbps を満たすか否か検証を行った結果が表 1 で示される．なお，「クラス数」は検証ツール TINA により同じ動作を行う複数の状態を一つの状態集合（クラス）に縮約した際の状態クラスの数であり，検証モデルの複雑度の尺度である．

表 1　CAN バス 10Mbps の検証結果（固定優先度スケジューリング）

CAN バスの転送速度	スループット要求を満たすか否か？	検証時間(s)	クラス数
10Mbps	Yes	0.391	11712

書籍発注システムの例については，各 CPU リソースが前節で述べた動的優先度スケジューリング（ノンプリエンプティブ EDF スケジューリングの離散近似）でスケジュールされるとして PrTPN への変換を行った．変換した PrTPN の規模は 135 プレース，155 トランジションである． 表 2 が検証結果となる．

表 2　書籍発注シナリオの検証結果（動的優先度スケジューリング）

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(s)	クラス数
10	Yes	0.266	10883
11.1	Yes	0.281	10863
12.5	Yes	0.266	10843
14.2	No	4.422	133107

8.4 考察

表 1，表 2 の結果より，固定優先度スケジューリングの方は 1 秒未満の検証時間であった．一方，動的優先度スケジューリングについては，変換されたモデルがより複雑になるため，より検証時間が大きくなったが，それでも数秒以内に検証が終了した．従って，本実験で扱った程度の例題規模であれば，提案手法により効率よく検証が行えるといえる．

9. あとがき

UML MARTE の振る舞い仕様とリソース割り当てから性能評価モデルである PrTPN へ変換手法を提案し，作成した PrTPN を用いてさまざまなスケジューリング方式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．UML MARTE から PrTPN への変換手法では，冗長ではあるが，機械的な変換がしやすいように工夫し，検証可能なモデルになるようなモデル化を行った．また，スループットの検証を行うため，既存の TPN ツール TINA を用いて，デッドロック検証問題へと帰着させた．提案手法により確定的な時間を扱う PrTPN を用いるため，最悪の場合の性能を検証することが可能であり，組込みソフトウェアなどの設計初期段階での性能検証に役立つと思われる．

今後の課題としては，提案した変換手法のツールへの実装やプリエンプティブスケジューリングへの対応などが挙げられる．

参考文献

[1] S.Balsamo, A.D.Marco, P.Inverardi, and M.Simeori, “Model-Based Performance Prediction in Software Development: A Survey”, IEEE Trans. on Softw. Eng, vol. 30, No.5, pp.295-310, 2004

[2] 株式会社オージス総研 オブジェクトの広場編集部(著)，山内亭和（ 監修)，その場でつかえるしっかり学べる UML2.0，2006

[3] Object Management Group MG, “A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems， Beta 2”, http://www.omg.marte.org, OMG Document Number ptc/2008-06-09

[4] J.L.ピータースン（著)，市川惇信，小林重信（訳)，ペトリネット入門，1984

[5] 青山　幹雄，内平直志，平石邦彦（著)， システム制御情報学会（編集)，ペトリネットの理論と実践，1995

[6] B.Berthomieu, F.Peres, and F.Vernadat, “Model Checking Bounded Prioritized Time Petri Nets”, Proc. of ATVA 2007, LNCS, vol.4762, pp.523-532, 2007

[7] C.L. Liu and J. W. Layland, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment”, Journal of the ACM, vol. 20, No1. 1, pp. 46-61, 1973

[8] A.Hamann, M.Jersak, K.Richter, and R.Ernst, “A framework for modular analysis and exploration of heterogeneous embedded systems”, Real-Time Sysetms,Vol.33, No.1-3, pp.101-137, 2006

[9] 山脇弘，「時間ペトリネットを用いた UML MARTE 振る舞い仕様の性能検証」，広島市立大学大学院情報科学研究科修士論文，2009.

[10] B. Berthomieu and F. Vernadat, ”TINA Time Petri Nets Analyzer ”, http://www.laas.fr/tina/.