ソフトウェア工学 138−8
（２００２．７．１１）

組み込みシステムにおける優先度付きタスクの抽出方法

渡辺 晴美
立命館大学 理工学部
E-mail: harumi@selab.cs.ritsumei.ac.jp

組込みシステムにおいて、プライオリティは、システムの安全性、性能を左右し、タスクを構築の基準のひとつである。プライオリティは、機能の重要さに加えタスクの周期、実行時間をもとに決定する。そのプロセスは複雑であり、大きなコストを伴う。さらに、ハードウエアの選択にも影響を与えるため、開発の早い段階での決定が望まれている。本稿では、上記の問題を解決するために、カラーペトリネットを利用したプライオリティに基づいたタスク構築支援方法について提案する。

A Task Constructing Method for Embedded Systems Based on Priority View

Harumi Watanabe
Department of Computer Science, Ritsumeikan University

In embedded systems, the priority of a task influences safety and a performance, and is one of the criteria of construction of a task. In addition to the urgency of a function, a priority is determined based on the period and execution time of a task. The process is complicated and large cost is required for it. Furthermore, since selection of hardware is also affected, determination of priorities in the early stage of development is desired. In this article, to cope with this problem, we propose a task construction support using Coloured Petri Nets for embedded systems based on priority View.

1 はじめに

組込みシステムにおいて、プライオリティは、システムの安全性、性能を左右する重要な要素であり、タスクを構築する際に考慮すべき非常に重要な要素である。[16]では、プライオリティはタスク構築クライテリアの一つである。

ハードリアルタイム性を保障するスケジューラーとして最もよく利用されている Rate Monotonic[19] は、短い周期ほどプライオリティが高い。従って、最も重要な役割を果たす機能は、最も短い周期のタスクに割り当てる工夫が必要となる。タスクの周期は、サンプリング周期にも影響するため、ハードウエアの選択に大きく影響する。Rate Monotonic を利用しない場合でも、ハードウエアのサンプリング周期は、デッドラインの一要素となり得るため、プライオリティと関連する。

ソフトリアルタイムシステムはハードリアルタイムシステムの設計よりも困難であると言われている [13]。ハードリアルタイムシステムの場合、時間とタスクの有効性の関係を表した有効性関数 (utility function)[14] が 1 か 0、すなわちデッドラインまでは 1 であり、それを過ぎれば、0 である。ソフトリアルタイムシステムの場合、ゆるやかに 0 近くへ推移するといった曲線を描く。従って、デッドラインが全く存在しないのではない。従って、周期とプライオリティに無関係であるわけではない。ハードウエアの選定は、開発の早い段階で行われるため、早い段階でのプライオリティを考慮したタスク抽出は重要であり、タスク抽出を支援する方法は開発効率と性能を上げるために必要である。

ところで、プライオリティのタスク割当ては、機能の重要度や緊急度のみに基づいて行うような単純作業ではない。一つの機能は、複数のイベントによって達成される場合は多く、重要な機能の起床に対応したイベントのみを優先しても、その機能に求められているリアルタイム性を保証することはできない。重要な機能を実現するのに必要なイベントのプライオリティを上げる必要があるかもしれない。

大規模なシステムでは、ある機能に着目した際にプラ

イオリティが低くても良いオブジェクトが、他の機能に着目した際に、プライオリティが高い必要があるかもしれない。例えば、ナビゲーションシステムで、最適経路を得る機能を実現することに着目した場合、移動路の状態 (交通量や路面状況) の情報を獲得するタスクは、車両情報 (故障情報など) を獲得するタスクよりも高いかもしれないが、移動車両間で経路情報を得る機能に着目した場合、逆かもしれない。

開発の分析段階で得たタスクの多くが、非周期的になる場合がある。組込みシステムでは、タスクを起動するイベントの種類 (イベント到着パターン) を考慮し、周期タスクに割当てることで、プライオリティと関連付ける。しかし、必ずどのタスクよりも優先させる必要があるが、非常通信のように非周期的に一年に一度起こるかどうか分からないようなイベントもある。このようなイベントは高い周期でポーリングし、プライオリティを高くするといった仕組みが必要になる。さらに、共有資源のアクセスではプライオリティインバージョンについても考慮する必要がある。

以上に関し、本稿では、プライオリティを抽出するために、機能やイベントの重要度とイベント到着パターン、機能やイベントの波及効果、共有資源について着目する。提案する方法では、開発者は [20] で決められた、プライオリティ、イベント到着パターンをタグとして UML のクラス図と状態図に添付した仕様を記述する。このクラス図と状態図をカラーペトリネットへ変換し解析を行う。解析には、[3] で提案したカラーペトリネットのスタティックスライシングとカラーペトリネットのシミュレーションによる到達グラフを利用する。

スライシングをもとに、データ依存関係から共有資源の抽出、スライスの最低実行時間、重要なイベントの波及効果を得る。スライシングはタグに付けられたプライオリティとイベント到着パターンを参考に、プライオリティが高いもの、あるいは低いものを対象として行う。スライスから着目したイベントの最低実行時間とイベントとの関わりからタスクの周期について見積もり、タグがついてないイベントや、非周期イベントについて周期を割当てる。最後に、カラーペトリネットを動かすことで、シミュレーションを行い性能を評価する。さらに、デッドロックの検出、公平性など並行性に関係する誤りについて解析を行う。

本稿では、以下 2 章では、本稿で提案する方法の基礎となる技術について紹介する。3 章では、プライオリティ抽出を支援する方法について提案する。4 章では、リアルタイムシステムへの適用例について評価し、5 章では、関連研究と比較する。6 章で議論し、最後に 7 章でまとめる。

2　カラーペトリネット

本論文で提案する方法は、クラス図と状態図で記述された仕様を K.Jensen が提案したカラーペトリネット [8] [9] [10] へ変換し解析を行う。また、変換したカラーペトリネットのスタティックスライシングも解析に利用する。以下、カラーペトリネット、カラーペトリネットの変換、カラーペトリネットのスタティックスライシング

2.1　カラーペトリネット

図 1 にカラーペトリネットの例を示す。カラーペトリネットには、左上の基本要素に加えて融合プレースという概念がある。このプレースを利用することにより、複雑な図を整理して記述したり、大域変数を表したりすることが可能である。

カラーペトリネットでは、システムの挙動を発火と呼ばれる仕組みにより表す。トランジションが発火することで、トークンはアークの方向に従いプレースからプレースへ移動する。発火は、発火可能なトランジションのうち一つが任意に起きる。発火可能とは、トランジションに入力しているアーク式を満たすようにアークの始点となるプレースにトークンがあり、さらにアークを通ってきたトークンの値がガードを満たす場合のことである。

カラーペトリネットの解析は到達グラフを用いて行う。図 1 のノードは、マーキングの状態を示し、上のノードは、トランジション Operation が発火する前の状態を表し、下のノード発火後を示している。

2.2　オブジェクト指向ソフトウエアの変換

図 2 にオブジェクト指向ソフトウエアのカラーペトリネットへの変換の概要を示す。この方法は、我々が [5] で提案した方法である。この方法では、開発者はクラス図とメソッド毎に状態図を記述する。メソッドは図 2 の右上のようなフォーマットのカラーペトリネットへ変換される。演算は図 3 の右上の形式に変換される。この変換の特徴は、図 3 の右側に示したように制御トークンとデータトークンの 2 種類が用意されていることと、図 2 の右下のメソッド呼出を行うためのカラーペトリネットにある。右下のネットは、制御トークンがどのインスタンスかによってオブジェクト指向特有の実行時の束縛を実現している。これら二つの特徴により、グラフの書き換えを行うことなしに、オブジェクト指向ソフトウエアの挙動をテストすることができる。オブジェクト指向ソフトウエアで利用されている状態図をテストに利用する場合、状態図では動的束縛はグラフの書き換えを行うことにより行うのが一般的である。グラフの書き換えを行う場合、網羅的にテストケースを生成する解析を行うことはできない。我々は、テストケースを生成する解析方法についても提案している [6]。

2.3　スタティックスライシング

カラーペトリネットのスタティックスライシングは、著者が [3] で提案した方法である。[3] は、オブジェクト指向ソフトウエアのスタティックスライシングではないが、上記の方法によって変換したカラーペトリネットは、理解しやすさのために、融合プレースを利用したりしているが、実際には、一つの基本的なカラーペトリネット (図 1 の右上に示した基本要素のみで構成されたカラーペトリネット) に変換される。従って、オブジェクト指向プログラムをスライスする際のような特別な工夫を行うことなしに、スライスを得ることが可能である。

スライシングは、以下の 4 つのステップから求める。

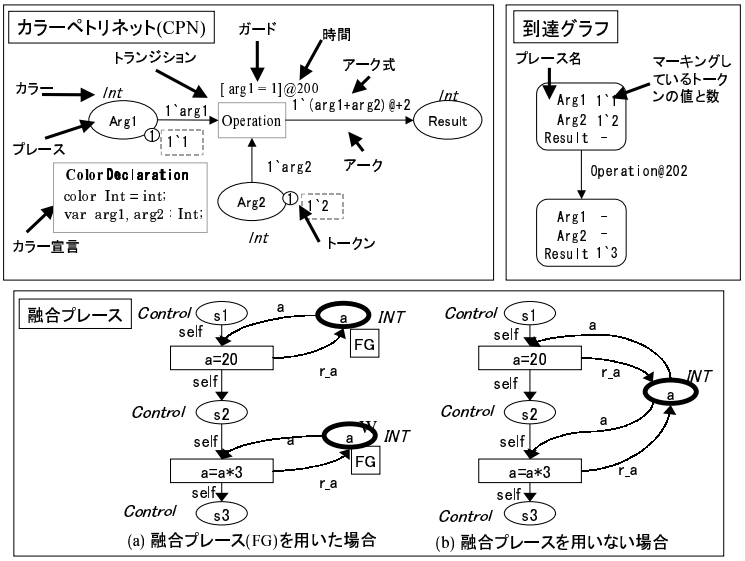

図 1: カラーペトリネット・到達グラフ・融合プレースの例

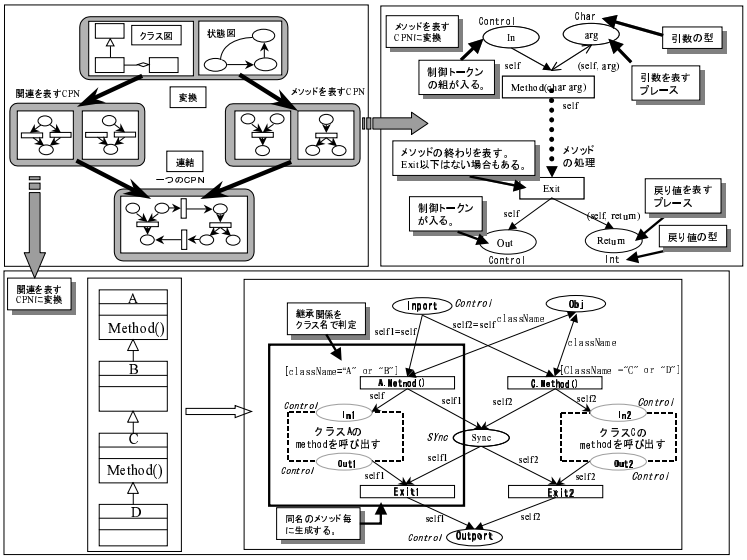

図 2: オブジェクト指向ソフトウエアのカラーペトリネットへの変換

データ依存関係と制御依存関係について、図 4 の上半分に示す。

1. スライシング基準と開始を選択する。
   スライシングを開始するトランジション (開始トランジション) と、スライシング基準となるトランジション (基準トランジション) を選択する。
2. データ依存関係を求める。
   (a) 基準トランジションに入力している変数プレース (データ依存プレース) を抽出する。
   (b) データ依存変数プレースに連結しているトランジションを抽出する。そのうち、開始トランジションから基準トランジションの間にあるトランジション (データ依存トランジション) を選択する。
   (c) スライスの構成要素は、データ依存プレース、

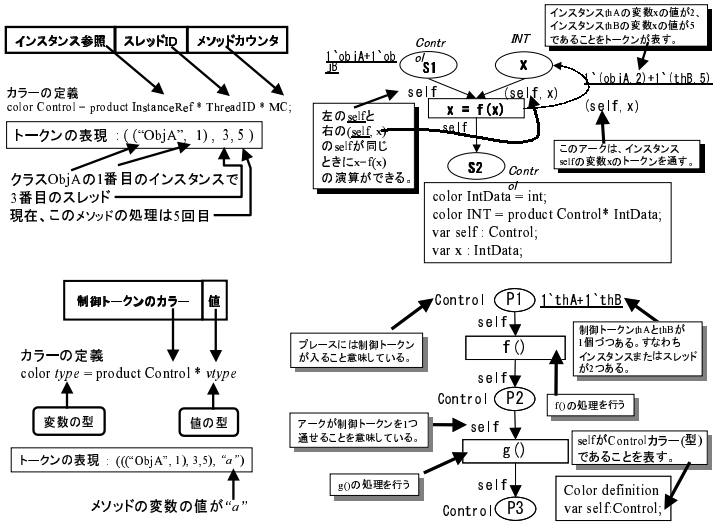

図 3: 制御トークン、データトークン、演算処理

データ依存トランジション、各データ依存トランジションへ入力する制御プレースと出力する制御プレースである。これらを連結するアークとアーク式は元のカラーペトリネットの関係を維持する。また、プレースのカラー、トランジションのガード式についても元のカラーペトリネットと等しい。

3. 制御依存関係を求める。

   (a) 制御依存プレース、トランジションを発見する。基準トランジションから開始トランジションへアークを逆にたどり、制御プレース、すなわち、複数の入力アークまたは出力アークを持つオブジェクトプレースを抽出し、制御トランジションを得る。また同様に、同期トランジション、すなわち、複数の入力アークまたは出力アークを持つ制御トランジションを抽出する。

   (b) スライス構成要素は、分岐、ループ、同期、オブジェクト生成に応じて図 4 の 4 つの構成要素の実線部分になる。

   (c) 制御依存トランジションに関するデータ依存関係を求める。

4. スライスを得る。

   (a) スライスの構成要素をデータ依存関係、制御依存関係から求める。

   (b) 切り離されたスライス構成要素を連結する。各構成要素で、同じオブジェクトプレースを連結する。独立した構成要素は、構成要素の入り口にあたるプレースと出口にあたるプレースを連結していく。

3 タスク構築のための解析

本章では、2 章で述べた技術を利用し、タスク構築に必要な情報、すなわちイベントの波及効果、共有資源についての情報を集め、カラーペトリネットでシミュレーションを行う。

1. 開発者は、UML のクラス図、状態図からタスク図を作成する。タスク図はメソッドの集まりとして定義する。

   タスク図には、[20] に従い重要度に応じてプライオリティのタグを添付する。さらに、イベント到着パターンに応じて Periodic、Irregular、Bursty、Bounded、Bounded average rate、Unbounded のタグを添付する。この段階では、全てにこれらのタグを添付する必要はない。重要なものと、ハードウエアなどの制約からすでに決まっているものについて添付する。

2. カラーペトリネットへ変換する。

   タスク図のタスクは、オブジェクトの集まりである。同じタスク内の制御トークンは図 3 のトークンでスレッド番号を同じ値にする。クラス名から継承関係をたどり、親クラスに含まれるメソッドと属性を参照可能にする。それ以外は、2 章で概要を述べた方法と同様である。

3. スライシングを行うタスクを決定する。

   スライシングを行う基準は、イベントの重要度、イベント到着パターン [13] から開発者が決定する。

4. スライシングを行い以下について調べる。

   - スライス中のアークやトランジションに添付してある時間をもとにスライスの実行時間を求める。

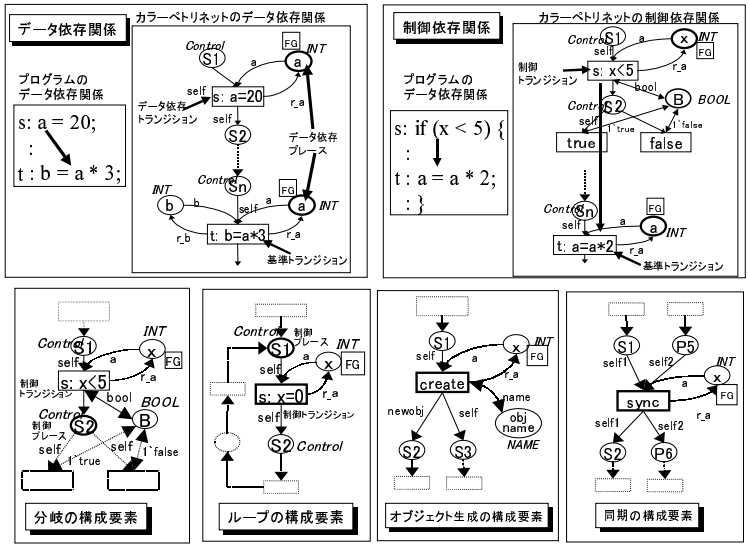

図 4: カラーペトリネットのスタティックスライシング

- データ依存関係から共有資源の有無を調べる。
- 他タスクとの関係を制御依存関係から調べる。

5. 非周期イベントを周期イベントへ対応付ける。

   重要なイベントととの関係からイベントのプライオリティを上下させる。スライスとイベント到着パターンをもとに周期を決定する。

6. シミュレーションを行う。

   カラーペトリネット上でシミュレーションを行い、挙動の確認 (分析した機能が要求に従っているかの確認)、デッドロックの検出、性能解析を行う。結果に応じて、時間がかかるなどの問題が発生した箇所をスライシングするなどして、タスクを構築しなおす。

   ところで、Rate Monotonic などのシミュレーションをカラーペトリネットで行うためには、スケジューラをカラーペトリネットで記述し、[4] で提案したような仕組みが必要である。スケジューラの記述に関しては、紙面の都合上、省略する。

4 適用例

図 5 は ITS に係わるシステムアーキテクチャ[1] の「移動車両間の経路情報の交換の制御モデル」をもとにした例である。クラス図と、収集クラスのメソッドである緊急収集メソッドと車両クラスのメソッドである車両メソッドの状態図、収集・車両間のタスク図について示す。図 6 は、車両メソッド、緊急収集メソッドの状態図を変換したカラーペトリネットである。図 7 に変換したカラーペトリネットの識別情報獲得メソッド呼出部分について示す。識別情報獲得メソッドは、移動体クラスと車両クラスの両方に存在し、車両クラスの方のメソッドが、Control トークンのクラス名に従い選択できるところを示している。図 8 は、情報収集 (交換情報) をスライシング基準検知収集 (緊急連絡) をスライシング開始として、緊急タスクについてスライシングした結果である。

5 関連研究

従来の組込みシステムの開発で用いられているシミュレーションはプログラムに基づいている場合が多い。Rapsody[13] は、UML に基づいたシミュレーションが可能であるがデッドラインの解析や、スライシングによる影響解析はできない。

ペトリネット [17] やカラーペトリネットなどの拡張ペトリネットの多くは、デッドロックの検出と性能解析の両方が解析できるツールとして制御システムの分野で多く用いられている。カラーペトリネットは、ソフトウエアの解析にも適しており、組込みシステムに関しても実際モデルへの適用例がある [10]。しかし、カラーペトリネットは、オブジェクト指向には対応していない。オブジェクト指向に対応した拡張ペトリネットの多くは、時間解析、デッドロックの検出などの点で、解析能力が落ちる。

また、状態遷移図をカラーペトリネットへ変換しない方法についても考えられる。状態遷移図に対してスライシングを行う場合、一度、遷移を制御とデータへ分ける作業が必要であり、このコストはカラーペトリネットへ変換する場合と同じである。シミュレーションによる解析に関してであるが、状態図の場合、実行時に決定される性質に関して、グラフの書き換え規則によって表す。我々が提案してきたカラーペトリネットを用いた方法では、実行時に決定される性質についてもカラーペトリネッ

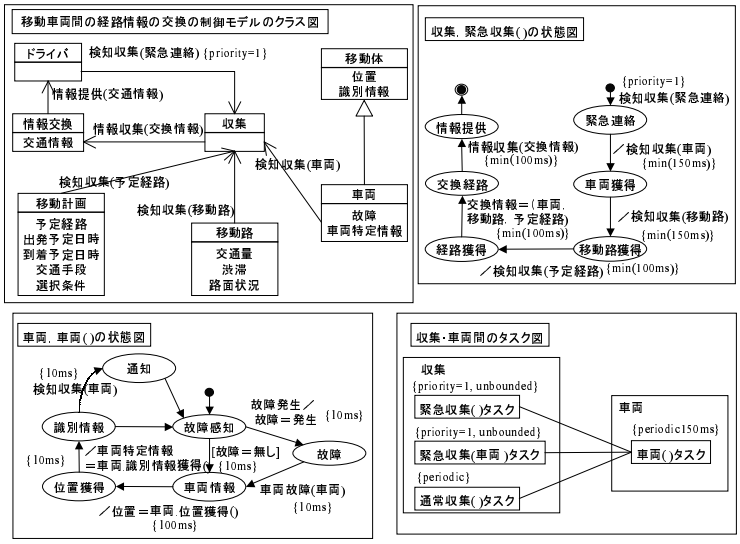

図 5: 移動車両間の経路情報の交換の制御モデル

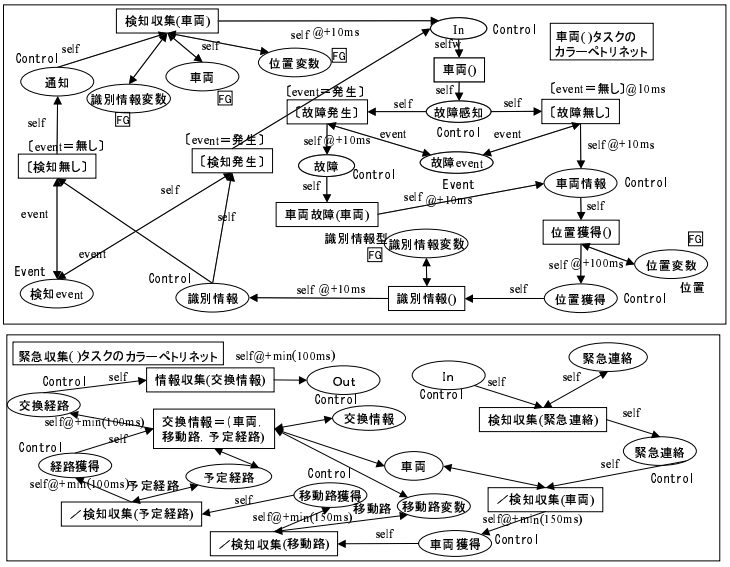

図 6: 制御モデルのカラーペトリネット

トでトークンを選択する方法として表現している。従って、継承関係がある場合でもスタティックにスライシングを得ることが可能である。

ペトリネットに関してのスライシングの研究は、半順序展開技術 (Partial-order verification tecnique) [7] と関連している。これらの研究は、ある状態からある状態に到達可能であるかを調べる可達解析を行い、循環を含むもとのペトリネットを循環構造を含まない木構造グラフへ展開する方法について研究を行っている。カラーペトリネットをはじめ多くの拡張ネットの半順序展開技術は、ペトリネットの網羅的な実行により到達グラフを獲得し、半順序展開を行う。この技術を利用することでも、機能に着目した部分ペトリネット、すなわちスライスを得ることは可能である。これは、プログラミング言語でのダイナミックスライシングが実行した結果から木を構築し解析することと同様のプロセスであり、ペトリネッ

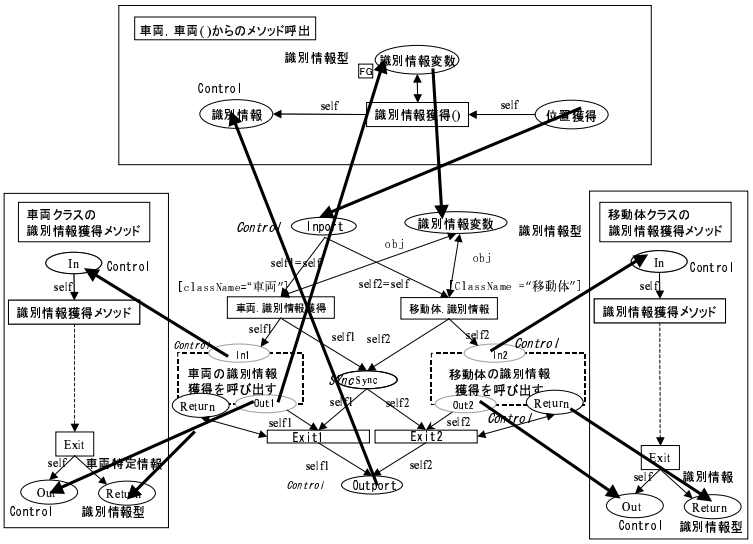

図 7: 制御モデルのメソッド呼び出し部分

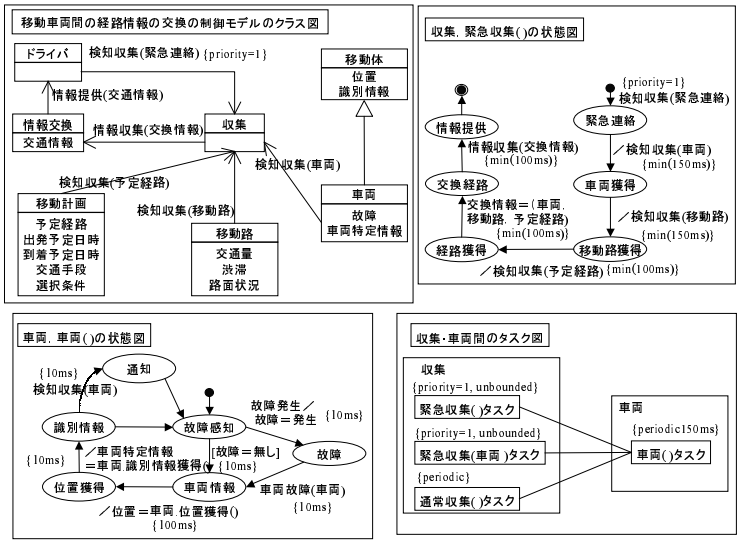

図 8: 制御モデルのスライシング

トのダイナミックスライシングとも言える。しかし、ペトリネットはプログラミング言語の場合と比べ、プレースやトランジションなどの構成要素について決まったセマンティクスはなく、記述者が決定するため、本稿で利用した静的な解析を汎用的に行うことは難しい。

6 議論

本稿では、カラーペトリネットのスタティックスライシングとシミュレーションを利用したタスクのプライオリティ付けを支援する方法について提案した。また、[3] で提案したカラーペトリネットのスタティックスライシングを利用することで、[3] が有益であることを示すことができた。本稿で提案した方法は、5 章で述べたように、

継承関係があるオブジェクトをタスクの要素とした場合も、共有資源、イベントの波及効果解析、さらに時間の見積もりが可能である点で新しい。また、その有効性を 4 章で示した。

[13] ではタスク図をオブジェクトの集まりとして表しているが、分析で得たクラスは多くの属性を持ち、それぞれお別なセンサーによって値を獲得するものも多く、異なるハードウエアによって実装される。従って、異なるサンプリング周期のものを同じタスク上に構築することになり、良い性能を得ることができない。また、スケジューラの中には、run-to-completion 型のものや、ノンプリエンプティブなスケジューラもある。このようなスケジューラは、非常に小さなタスクにする必要がある。そのため、本稿では、メソッド単位で行った。「収集クラス」をオブジェクト単位でタスク化した場合、一つのタスクになるが、「緊急連絡」用のタスクと「通常」用のタスクは別に実装した方が良い。従って、4 章からメソッド単位の方がオブジェクト単位よりもタスク構築により適していることを示すことができた。

上記のような利点をふまえても、本稿で提案した支援のみでは、プライオリティ付けには、開発者の直感に依存するところが多く、開発者に対して大きな負担が存在しる。非周期タスクを周期タスクに割当てる方法に、ポーリングタスクに割当てる方法、スポラディックサーバー (sporadic server) を利用する方法があるが [13]、現段階では、人手により記述する必要がある。すでに、様々な挙動に関するパターンが提案されているが、イベント到着パターンから挙動に関するパターンを生成するプロセスについての研究はなされていない。ソフトリアルタイムシステムでは、プライオリティの決定に有効関数を考慮する必要があり、さらに複雑なプロセスとなる。さらに、イベント到着パターンを開発者が記述していくことは、開発者に大きな負担となる。

7 おわりに

本稿では、カラーペトリネットのスタティックスライシングとシミュレーションを利用したタスクのプライオリティ付けを支援する方法について提案した。提案した方法は、継承関係があるオブジェクトをタスクの要素とした場合も、共有資源、イベントの波及効果解析、さらに時間の見積もりが可能である点で新しい。プライオリティの抽出については、6 章で述べたように、様々な課題が残っている。今後は、これらについて取り組んでいきたい。

参考文献

[1] “高度道路交通システム (ITS) に係わるシステムアーキテクチャ”, http://www.its.go.jp/ITS/j-html/SAview/index.htm

[2] 下村隆夫, “プログラムスライシング”, 共立出版, 1995

[3] 渡辺 晴美, ”カラーペトリネットスライシングによるリアルタイム開発支援”, 情報処理学会ソフトウエア工学研究報告,2001-S E-134, 2001, 10

[4] 伊藤恵, 渡辺晴美, 西田雅彦, 片山卓也, “オブジェクト指向リアルタイムシステムのテスト支援環境”, 情報処理学会オブジェクト指向 2000 シンポジウム論文集, pp.49-56,2000, 9

[5] 渡辺 晴美, 徳岡 宏樹, Wu Wenxin, 佐伯 元司, “カラーペトリネットによるオブジェクト指向ソフトウエアのテストと解析方法”, 電子情報通信学会和文誌 D-I, Vol.J82-D-I, No.3, 1999, 3

[6] 渡辺 晴美, 工藤 知宏, “カラーペトリネットを用いた仕様記述からのテストケース生成方法”, 電子情報通信学会和文誌 A, Vol.J80-A, no.7, pp.1073-1080, July 1997

[7] A. Kondratyev, M. Kishinevsky, A. Taubin, S. Ten, “A Structual Approach for the Analysis of Petri Nets by Reduced Unfoldings”, pp356-365

[8] K. Jensen, “COLOURED PETRI NETS – Basic Concepts, Analysis Methods and Practical Use – Volume 1”, Springer-Verlag, 1992

[9] K. Jensen, “COLOURED PETRI NETS – Basic Concepts, Analysis Methods and Practical Use – Volume 2”, Springer-Verlag, 1994

[10] K. Jensen, “COLOURED PETRI NETS – Basic Concepts, Analysis Methods and Practical Use – Volume 3”, Springer-Verlag, 1997

[11] J.D. McGregor and D.M. Dyer, “A Note on Inheritance and State Machine”, ACM Press, Software Engineering Notes, vol.18, pp.61-69, Oct. 1993

[12] G.Booch, J.Rumbaugh, I.Jacobson, “The Unified Modeling Language User Guide”,Addison Wesley, 1999

[13] B.P.Douglass, “Doing Hard Time – Developing Real-Time Systems with UML, OBJECTS, FRAMEWORKS, And Patterns”

[14] J. Douglas, “Real-Time Manifesto”, http://www.real-time.org

[15] B.P.Douglass, “Real-Time UML 2nd Edition Developing Efficient Objects for Embedded Systems”, Addison Wesley, 1999

[16] H. Gomaa, “Design Concurrent, Distributed, and Real-Time Applications with UML”, Addison Wesley, 2000

[17] J.L.Peterson, “Petri Net Theory and the Modeling of Systems”, Englewood Cliffs, New Jersey, Prentice Hall Inc., 1981

[18] B. Selic, G. Gullekson, P. T. Ward, “REAL-TIME OBJECT-ORIENTED MODELING”, Wiley, 1994

[19] J.A.Stankovic, K.Ramamritham, “Hard Real-Time Systems”, IEEE, 1988

[20] “UML Profile for Schedulability, Performance, and Time Specification”, http://www.omg.org/technology/documents/modeling_spec_catalog.htm, March 2002