Vol.2016-EMB-42 No.1
2016/8/25

コンテキスト指向ソフトウェアのための ペトリネット・シミュレータ構築の課題

渡辺晴美[†1]　小倉信彦[†2]　菅谷みどり[†3]　久住憲嗣[†4]

概要：コンテキスト指向プログラミング(COP)技術は、環境の変化すなわちコンテキスト(文脈)に応じてシステム全体をレイヤにより実行時に書き換えが可能である。本稿では、RT-COA と呼ぶアーキテクチャに基づいた COP のカラーペトリネット(CP-nets)のフレームワークについて紹介する。RT-COA は、レイヤ切り替えが即時に実行できない場合の問題について、オペレーティングと類似した方法で対処することが可能である。我々は、これまで RT-COA を中心に C#で実装した COP、レイヤ構造図、レイヤ相互作用図を提案した。本稿では、RT-COA のフォーマル化、および、シミュレータの構築を目指し、CP-nets によるモデリングについて議論する。

キーワード：コンテキスト指向プログラミング，カラーペトリネット，ソフトウェアアーキテクチャ

Issues for Realizing a Petri-Nets Modeling to Simulate Context-Oriented Software

HARUMI WATANABE[†1]　NOBUHIKO OGURA[†2]
MIDORI SUGAYA[†3]　KENJI HISAZUMI[†4]

Abstract: Context -Oriented Programming(COP) enables to reconstruct a software by layer at runtime on receiving a change of the surrounding environments. The article introduces a modeling framework of Colored Petri Nets for our COP architecture called RT-COA. This architecture solves the layer interaction problems, such as delay time for switching layers, by an OS-like mechanism. In previous works, we have proposed a C# COP, a layer structure, and a layer interaction diagram based on RT-COA. Towards a realization to formalize RT-COA and construct its simulator, we discuss the CP-nets modeling in this article.

Keywords: Context-Oriented Programming, Coloured Petri Nets, Software Architecture

1. はじめに

環境の変化に応じて、システムがサービスを提供することは、IoT の重要な役割の一つである。スマートフォンでは、位置情報やバッテリー残量に応じてサービスを変更することは、一般的な機能である。一方、現在のロボットシステムは、介護、見守り、災害救助など単一であり、環境の変化すなわちコンテキスト(文脈)に応じたサービスの提供、すなわち、スマートロボットの実現は難しい。その理由について、我々は、ソフトウェアの基礎である**「横断的関心事、状態、変数」**の 3 つの視点から考察し解決を目指している。

この問題解決に期待できる技術に、コンテキスト指向プログラミング(COP: Context-Oriented Programming)があり、コンテキストに応じて、ソフトウェア全体を実行時に書き換え可能である。代表的な COP に[1-7]がある。特に、レイヤと呼ぶ単位でクラス群を実行時に書き換え可能な COP は、我々が着目している横断的関心事の問題に効果的であると考える。我々は、スマートロボットに適した 3 つの視点を考慮したアーキテクチャ(RT-COA: Real-Time Context-Oriented Architecture)を提案した[8-9]。これまで、RT-COA を基にコンテキスト指向方法論 RT-COM(Real-Time Context-Oriented Methodology)を目指し、レイヤ構造図、レイヤ相互作用図[10][11]、C#の COP[8]を提案してきた。これらの特徴は、レイヤ部分とレイヤ切り替えを管理する部分を分離し、レイヤの状態をオペレーティングシステムと類似した機構で管理することにある。本機構により、3 つの視点に貢献できると考えている。

以上の技術の難しさは、レイヤ相互作用の問題を扱うレイヤ管理部分にある。本稿では、レイヤ管理部分を構築可能にするための構成とレイヤ活性化の処理に関する流れについてカラーペトリネット(CP-nets: Coloured Petri Nets)[12]によるモデリングした。本モデリングの目的は、フォーマル化および、RT-COM で開発したアプリケーションをシミュレート可能にすることを目指した議論の土台を作ることである。また、新規性は、レイヤの管理部分について CP-nets によりモデリングする点にある。

†1 東海大学
Tokai University
†2 東京都市大学
Tokyo City University
†3 芝浦工業大学
Shibaura Institute of Technology
†4 九州大学
Kyusyu University

以下、2 章では、まずスマートロボットに関する問題を 3 つの視点で整理する。これらの問題を解決するために提案したアーキテクチャ RT-COA、RT-COM について紹介し、本稿で紹介するモデリングの位置付けを明らかにする。3 章では、本稿の主題であるレイヤ相互作用問題について述べる。4 章では、準備として CP-nets について概説し、CP-nets を利用する理由を明らかにする。5 章では CP-nets よるコンテキスト指向ソフトウェアについて紹介する。6 章では、達成事項、今後の課題、新規性、有用性、信頼性について議論する。

2. 問題・アーキテクチャ・方法論

本章では、まず最終目標となるスマートロボットの問題を整理する。次に、その課題を解くために想定しているコンテキスト指向アーキテクチャおよび方法論について述べ、本稿で紹介するペトリネットによるシミュレータの位置付けを明らかにする。

2.1 スマートロボット実現に向けた問題

我々が着目しているスマートロボット実現に向けた問題について述べる。

(1) **横断的関心事の問題**：環境の変化は、サービスと直行する関心事である。ロボットを構成するセンサやアクチュエータは、一般的に変化しないが、コンテキストの変化に応じて全ての振る舞いを変化させる必要がある。すなわち、**一つのサービスを実現するために、**センサやアクチュエータごとにオブジェクトを構築したとする。コンテキストの変化に応じて各々のオブジェクトの振る舞いが変化する。特に、ロボット等の組込みシステムは非正常系とよぶ処理を多数有し、ソフトウェアを複雑化しているが、これらの非正常系の多くは、環境の変化に応じて発生するものが多いことからコンテキストである。

(2) **状態の問題**：ロボットでは振る舞いを突然の変更することが難しい。従って、レイヤの切り替えを即座に行うことができず、待ち状態が発生する問題が発生する。例えば、掃除機の吸引を途中で中断してしまうと、ゴミを巻き散らかしてしまうかもしれない。また、処理の開始にも、アクチュエータやセンサの初期化をはじめ多くの処理が必要となり、すぐに処理をはじめられない。以上から、コンテキストに応じたサービスを適用するために、レイヤの活性、不活性には準備状態が必要となる。さらに、コンテキストに応じて排他的にサービスを提供するような場合、すぐにサービスを切り替えられず待ち状態を生じる場合がある。

(3) **変数の問題**：レイヤに応じた変数の割り当てに加え、ロボットの振る舞い変更には、制御系の変更が必要な場合がある。優先度を考慮した変数値の受け渡しが発生する。

2.2 RT-COA

図 1 に我々が提案したアーキテクチャ RT-COA(Real-Time Context-Oriented Architecture)を示す。RT-COA は、レイヤ管理部分(Layer Manager)とレイヤ部分(Layers)からなる。レイヤ管理部分とレイヤ部分を分離することで、コンテキストに応じるための関心事と、レイヤで提供するサービスに関する関心事を明確に分離している。

これらの分離を実現するために、レイヤ管理部分は、**コンテキスト管理(Context Manager)**、**レイヤスケジューラ(Layer Scheduler)**を含み、オペレーティングシステムがタスクコントロールブロックによりタスクを管理するのと同様の仕組みで、**レイヤコントロールブロック(Layer Control Block)**により各レイヤ管理している。尚、レイヤ受信部(Layer Receiver)は、実行時に新たにサービスを追加するための部分である。

一方、上記の関心事の分離を強化するために、各レイヤは、**レイヤ監視部分(Layer Observer)**を備えている。これは、コンテキストの変化を感知するセンサは、レイヤが提供する通常のサービスにおいても利用しているためである。

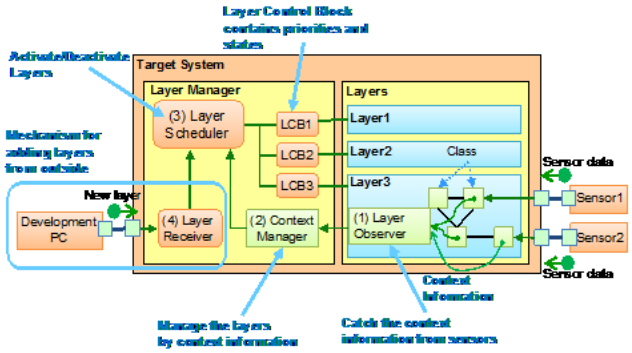


図 1　コンテキスト指向アーキテクチャ：RT-COA

Figure 1　RT-COM

本アーキテクチャは 2.1 で述べた問題解決に下記のとおり貢献する。

(1) **横断的関心事の問題**： RT-COA では、レイヤ部分とレイヤ管理部分に分離している。各々は、サービス内容を実装する部分と、環境の変化すなわちコンテキスト情報を管理する部分を分離できる。分離を実現するためにあるのが、レイヤ監視部分とコンテキスト管理部分である。

(2) **状態の問題**： RT-COA のレイヤスケジューラにより、レイヤの状態をオペレーティングシステムと同様の方法でレイヤの待ち状態や排他制御を管理する。

(3) **変数の問題**：排他制御を可能にすることで、オペレーティングシステムのタスク間通信と同様に、優先度を考慮した変数の授受を可能にする。

2.3 RT−COM

図 2 に我々が構築を目指している方法論 RT-COM(Real-Time Context-Oriented Methodology)について示す。図に示すとおり、我々の方法では、フィーチャモデルに基づき、レイヤ構造図、レイヤ相互作用図、レイヤ状態図を構築する。レイヤ内については、UML によりモデリングする。これらのモデルをもとに RT-COS と呼ぶ C#の COP を記述し、RT-

COA を内包する RT-COLE(Real-Time Context-Oriented Layer Engine)で動作させる。将来的には、RT-COS をモデルから自動生成する。

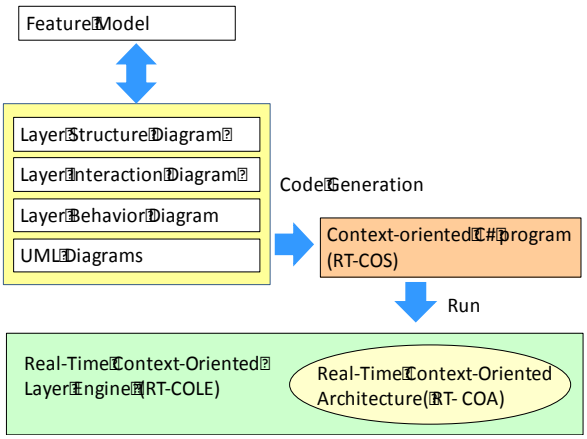


図 2　コンテキスト指向方法論の概要：RT-COM

Figure 2　Outline of RT-COM

2.4 位置づけと目的

本稿で紹介する CP-nets によるモデリングは、RT-COA に基づいたレイヤ切り替え部分であり、レイヤ相互作用図で記述したレイヤの切り替えをシミュレートする位置づけにある。ただし、本稿のモデリングは、RT-COA の形式化および RT-COLE 上で動作するアプリケーションのシミュレータ構築を将来実現するための準備段階であり、議論の土台作りを目的としている。

3. レイヤ相互作用問題

本章では、本稿で対象とするレイヤ相互作用問題について述べる。まず、説明の準備として、掃除機ロボットの例について紹介する。次に、本問題を可視化するために提案したレイヤ相互作用図について概説する。最後に、本稿で課題としているレイヤ相互作用問題について述べる。

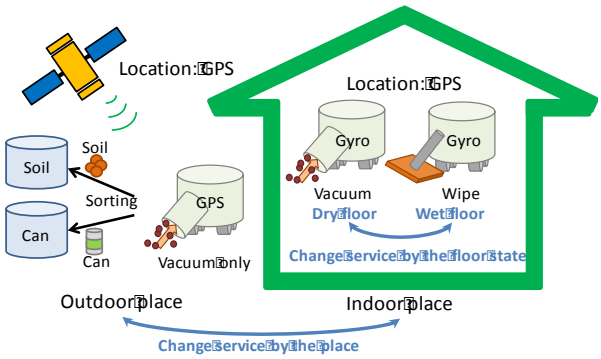


図 3　コンテキスト指向方法論の概要：RT-COM

Figure 3　Multi-service cleaner robot.

3.1 例

本節では、レイヤ相互作用問題を明らかにするために、図 3 の掃除機ロボットの例を提示する。本稿で関連する掃除機のサービスは以下の通りである。掃除機は屋内と屋外の掃除が可能である。屋内では、乾いた床では吸引掃除、濡れた床では拭き掃除を行う。一方、屋外では、吸引掃除のみである。

3.2 レイヤ相互作用図の概要

図 4 に示したレイヤ相互作用図は、レイヤの活性・不活性をモデリングするための図である。図 4 に示すとおり、レイヤ間の関係を表す横棒の左側にレイヤ名を記し、レイヤ名の下に排他的なレイヤを付す。活性・不活性に関するメッセージ、および、レイヤも要素については、レイヤ間の関係を表す横棒の左側と下側に記す。

3.3 レイヤ相互作用図の特徴

レイヤ相互作用図は、シーケンス図やタイミングチャートと外見が類似しているが、下記の点で異なる。

(1) 不活性準備状態と活性後状態の保持：これらが、レイヤ相互作用問題について特に重要な役割を果たすため、敢えて、要素に加えている。

(2) 表現目的(振る舞い)の違い：シーケンス図やタイミングチャートはオブジェクト、スレッド、タスクなどの並行に動作するモジュールをモデリングするが、レイヤ相互作用図の場合は、後に呼ばれたレイヤが前のレイヤを上書きするため、その挙動が大きく異なる。見た目は類似しているが、全く異なる振る舞いをモデリングしている。

3.4 レイヤ相互作用に関する問題

前節の例題について発生するレイヤ相互作用問題について考える。本問題は図 4 の×印の周辺に関する問題である。まず、レイヤとなる要素は、環境要因であることから、屋内と屋外、濡れた床と乾いた床である。従って、屋内、屋外、濡れた床、乾いた床がレイヤであり、図 4 の Outdoor, Indoor, Wet, Dry が各々のレイヤに対応している。掃除機は、前節のとおり屋外では拭き掃除ができない。ところが、屋内で拭き掃除をしている最中に掃除機が屋外に出てしまうことは容易に想像がつく。その際は、拭き掃除である Wet レイヤを中断させ、吸引掃除である Dry レイヤに移行しなければ、屋外レイヤである Outdoor レイヤに移動できない。そこで、この非正常系と言える状況を、エラー処理として×印を付し、モデリングしたのが図 4 である。ペトリネットでモデリングする際も、以上の問題をシミュレーション可能である必要がある。

4. カラーペトリネット(CP-nets)

本章では、本稿で紹介する形式モデリング手法であるカラーペトリネット(CP-nets:Coloured Petri Nets)について概説し、なぜ、CP-nets を利用するのかについて述べる。

4.1 CP-nets の概説

カラーペトリネットは、K. Jensen が提案したグラフ形式モデリング手法である[12]。C. A. Petri のペトリネット[13]を、型で色付けしたトークンと階層化で拡張し、複雑なシステムをモデリング可能にしている。以下、ペトリネット、CP-nets について述べる。

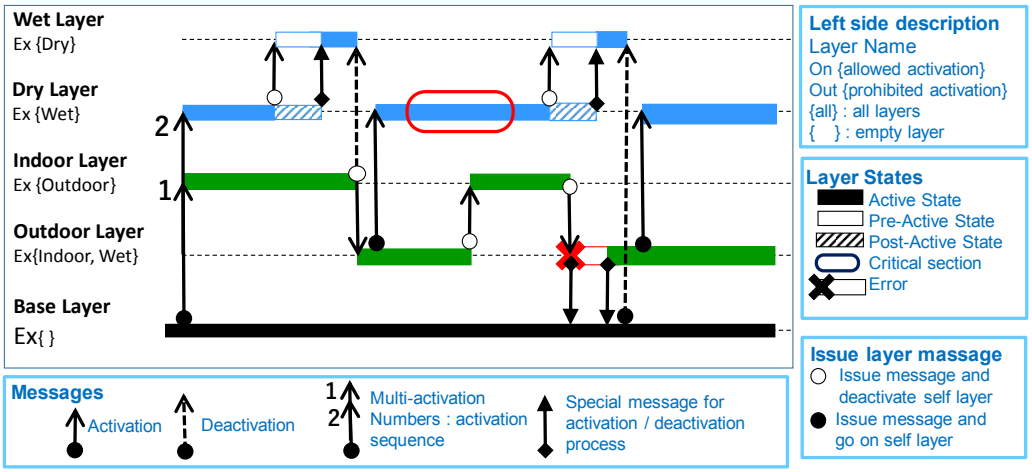


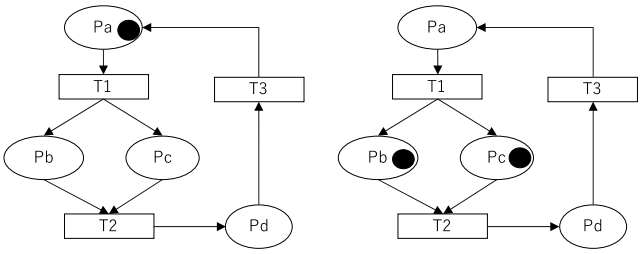
図 4　レイヤ相互作用図

Figure 4　RT-COM

(1) ペトリネット

ペトリネットを図 5 に示す。ペトリネットは楕円で示されたプレースと四角で記されたトランジションからなり、各々は矢印で結びつける。図 5(a)のプレース Pa の黒丸をトークンと呼び、トークンで印付けることをマーキングと呼ぶ。トランジションに入力しているプレースが全てマーキングされている場合、発火可能状態と呼ぶ。図 5(a)の T1 および(b)の T2 が発火可能である。図 5(a)の T1 が発火すると(b)の状態になる。発火可能な状態のトランジションが複数ある場合は、その一つ選ばれ発火する。

ペトリネットの目的は、デッドロック等の並行システムの問題発見にある。発火によるシミュレーションに加え、シミュレーション結果を表した到達グラフ、プレースとトランジションの接続行列を作成し解析する方法がある。



(a) T1 発火前　　(b) T1 発火後

図 5　ペトリネット

Figure 5　Petri Nets.

(2) CP-nets

図 6 に CP-nets の例を示し、カラー宣言を図 7 に記す。(a)～(c)は、0～3 を数え上げ、数え上げた数をリストにするソフトウェアをモデリングしている。サブネット A は、0～3 をカウントし、3 になるとゼロに戻る。サブネット B は、主要関数にあたる部分で、サブネット A と C を関連付ける。サブネット C は、0～3 の数のリストを作成する。

前述の通り CP-nets はペトリネットのトークンに型による色付けと階層化を特徴としている。以下、トークンカラー、階層化について概説する。

- **トークンカラー**：**「トークン」**はカラーと呼ぶ型の値を持つことができる。図 6 のサブネット A のプレース Num は INT 型のトークン 2 を 1 つ持つ。サブネット C のプレース S1 は UNIT 型のトークン e を持ち、S3 は[1, 0, 3, 2,1,0]というリスト型のトークンを持つ。

  「プレース」は、受け取れるトークンのカラーで限定する。サブネット A のプレース Num に INT 修飾子が付いている。これは、この Num が INT カラーのトークンのみを受け取ることを意味している。

  プレースとトランジションを結ぶ**「アーク」**には式または変数を付ける。これは、アークを通るトークンの型を明示的にするとともに、トークンの値に処理を加えることができる。

  「トランジション」には、アークに付した変数を利用して、**「ガード条件」**を付けることができる。例えば、サブネット A において、トランジション A が発火するのは、変数 num の値が 3 の未満の場合である。すなわち、プレース Num にマーキングされているトークンの値が 3 未満の場合である。

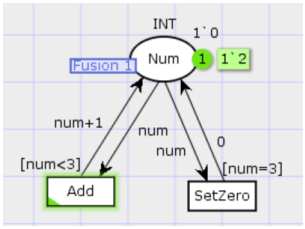
  トークン／プレースの型、アーク／トランジションに付す変数と関数は「**カラー宣言**」で宣言する。
- **階層化**：CP-net では、トランジションの階層化が可能であり、融合プレースという特別なプレースを持つことが可能である。

  「トランジションの階層化」に関し、サブネット C はサブネット B のトランジション t2 を階層化している。トランジション t2 に入力している Pb と出力している Pc は**「ポートプレース」**と呼び、呼び出すネットにトークンを引き渡すことができる。

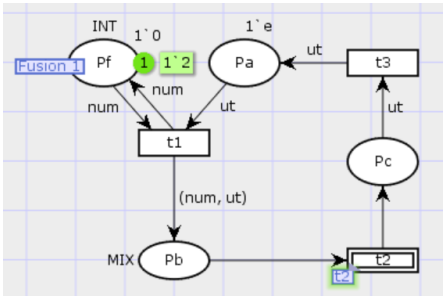
  「融合プレース」は、サブネット A の Num とサブネット B の Pf である。これらのプレースは外見上 2 つ

であるが、一つの同一プレースとしてみなすことができる。従って、Num と Pf は常に同じ値、同じ数のトークンによりマーキングされる。図 6 の例のように、サブネットを分割した場合のメッセージ通信に使うことができる。

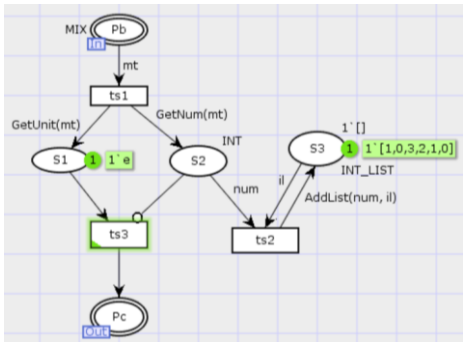
融合プレースのみで、階層化が実現できそうであるが、トランジションの階層化は、プログラミング言語の関数呼び出しが戻る場所をスタックしているのと同様に、戻る際のトランジションを把握している。



(a) サブネット A



(b) サブネット B



(c) サブネット C

図 6　カラーペトリネット

Figure 6　Cp-nets.

```
colset MIX
  = product INT * UNIT;
colset INT_LIST = list INT;
var num:INT;
var mt:MIX;
var ut:UNIT;
var il:INT_LIST;
fun AddList(l, ll) = l::ll;
fun GetNum(l:MIX) = #1 l;
fun GetUnit(l:MIX) = #2 l;
```

図 7　カラー宣言

Figure 7　Color declaration

4.2 CP-nets とコンテキスト指向ソフトウェア

本節では、CP-nets でモデリングするための課題を整理し CP-nets を選択した理由について述べる。

(1) 課題

以下に、2 章で述べた我々の最終目標である横断的関心事、状態、変数と関連し、CP-nets でモデリングする際の課題をいかに示す。

- **構成：**図 1 に示した我々アーキテクチャ RT-COA の構成要素を表現できること。すなわち、レイヤ、レイヤ監視部分、コンテキスト監視部分、レイヤスケジューラ、レイヤコントロールブロックをモデリングできること。
- **振る舞い：**上記の構成要素が協調し、レイヤの活性・不活性を表現できること。活性化されたレイヤに伴い、同じ名前の変数・メソッドが変化する。多重にレイヤが活性化された場合、その活性化の順序に従い、メソッドや変数を適切に実行させなければならない。その際、RT-COA の特徴であるレイヤの状態を考慮できること。
- **レイヤ相互作用問題：**3.4 で述べた排他的なレイヤの活性化に関する問題を表現でき、問題を検出できること。

(2) 理由

CP-nets を選択した理由は、上記の課題を実現できる可能性を持っているためである。特に、レイヤに応じたメソッド、変数の選択をシミュレートできることが、重要であり、CP-net のカラートークンは、この課題の実現に寄与できると考える。

加えて、CP-nets は。インタープリタ言語のように動作させながら構築できるところに魅力がある。本研究の目的であるフォーマル化およびシミュレータの構築は、現在、実現に向けた課題を抽出する段階であり、動作させながら構築できることは助けとなる。

さらに、CP-nets は、デッドロックなどの並行性の問題検出が可能である。これは、シミュレーション結果を到達グラフで表現し、モデル検査ツールと合わせることで実現している。本研究が実現した際には、これらの機能により、レイヤ相互作用の問題解決に期待できる。

5. RT-COA の CP-nets モデル

本章では、RT-COA の CP-nets モデルの概要について紹介し、本稿では、レイヤの表現とコンテキストイベントの識別に関し紹介する。

5.1 概要

図 8 に、図 1 の RT-COA の構成要素に対応付けた CP-nerts モデルの概要を示す。図 1 と図 8 では縦横の配置が代わり、レイヤ部は、図 1 では右側であるが図 8 では上部である。

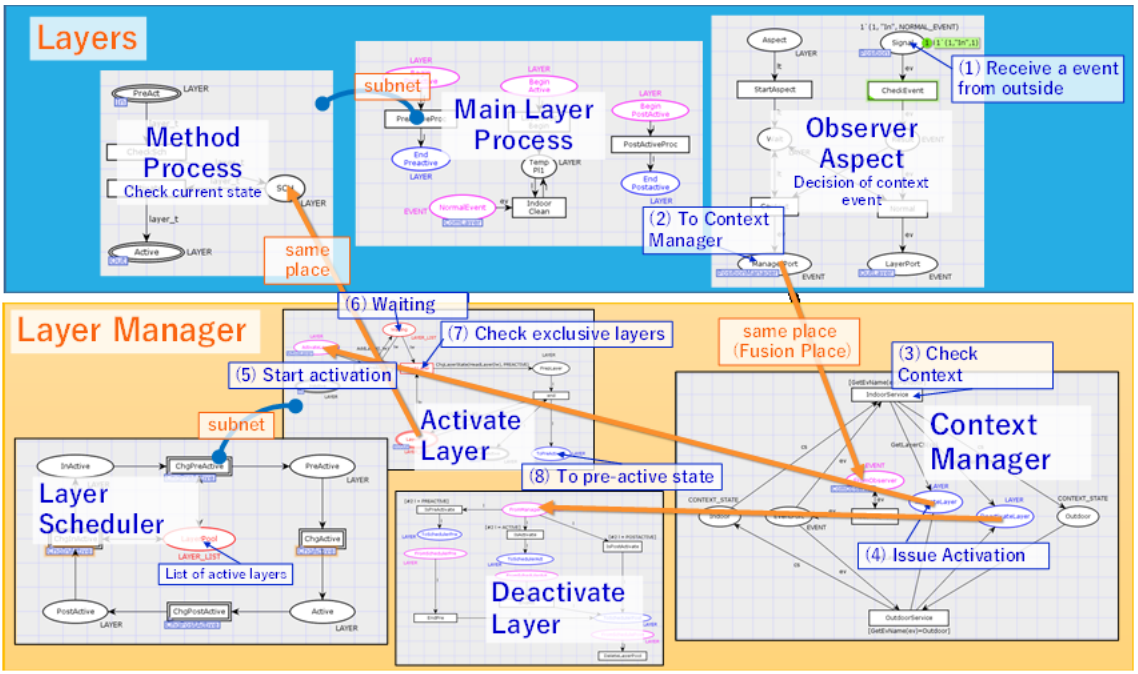


図 8　RT-COA の CP-nets モデルの構成

Figure 8　Color declaration

これらのサブネットは、4 章で述べた CP-nets の階層化と融合プレースにより、トークン授受しながら動作する。下記に、レイヤ活性化の流れについて記す。尚、下記の番号は、図 8 の(1)～(8)に対応している。

(1) イベントの受理：コンテキスト監視部分(Observer)は、外部からのシグナルを受理する。

(2) コンテキストイベントかどうかの判断：レイヤ内でレイヤ活性・非活性と関係のあるイベントかどうかを判断する。通常のイベントであれば、通常のレイヤ処理である Main Layer Process のサブネットに処理を戻す。コンテキストの変化と関係のあるイベントの場合は、コンテキスト管理部分(Layer Manager)に通知する。

(3) コンテキストの確認：レイヤから受診したイベントの位置付け、すなわち、コンテキストを判断する。

(4) レイヤ活性化命令の発行：コンテキストの状況により、レイヤ活性化命令を発行する。

(5) レイヤ活性化命令の処理：図 8 の Activate Layer と Deactivate Layer は、プログラミング言語におけるレイヤ活性命令とレイヤ非活性命令の処理部分にあたる。(4)の命令発行に従い、処理を行う。

(6) 活性化のためのレイヤの待ち：レイヤは多重に活性化する可能性があるため、レイヤを順次活性化するために待たせるための処理を行う。

(7) 排他的なレイヤの処理：3.4、4.2 で述べたレイヤ相互間の問題を処理する。

(8) 活性準備状態：図 8 のサブネット Activate Layer と Deactivate Layer は、Layer Scheduler と関連しており、Active Layer の処理が完了すると、Scheduler は活性準備状態(PREACTIVE)になる。

5.2 トークンカラー

本節では RT-COM の特徴を表すカラーについて紹介する。

(1) レイヤの状態

```
colset STATE = with PREACTIVE | ACTIVE |
                    POSTACTIVE|INACTIVE | WAIT;
```

上記は、レイヤが活性準備状態 PREACTIVE、活性化状態 ACTIVE 、活性後状態 POSTACTIVE 、不活性状態 INACTIVE、待ち状態 WAIT を持つことを表す。

(2) コンテキストのイベント

```
colset EVENT =
        product EVENT_NAME * EVENT_ ATTRIVUTE;
```

コンテキストは、イベントを表す名前と属性からなる。EVENT_ ATTRIVUTE の値により、コンテキストの変化と関係したイベントかどうかを判断する。

(3) レイヤ

```
colset LAYER_NAME = STRING;
colset LAYER_NAME_LIST = list LAYER_NAME;
colset LAYER = product
LAYER_NAME * STATE * LAYER_NAME_LIST;
```

上記、3 つめのカラー宣言がレイヤの宣言であり、RT-COA のレイヤコントロールブロックに対応する。レイヤを表すトークンは、レイヤ名(LAYER_NAME)、状態(STATE)、レイヤ名のリスト(LAYER_NME_LIST)からなる。最初の宣言より、レイヤ名は文字型である。レイヤ名のリストは、排他的なレイヤを表すのに用いる。

6. おわりに

本稿では、我々が提案してきた RT-COA を核としたコンテキスト指向ソフトウェアの形式化およびシミュレータ構築を目指し、CP-nets によりモデリングについて紹介した。以下、以下、課題の達成事項、技術的貢献に関する新規性・有用性・信頼性、今後の課題について議論する。

6.1 達成事項

我々の最終目標である横断的関心事、状態、変数の問題について、CP-nets でモデリングする際の課題を 4.2 節において、構成、振る舞い、レイヤ相互作用問題に着目し整理した。これらの課題について、5.1 および 5.2 に述べたカラーより、構成、振る舞いについて単純な内容についてモデリングできることを確認した。また、5.1 のレイヤ活性化の流れに記した「(6) 活性化のためのレイヤの待ち」、「(7) 排他的なレイヤの処理」で述べたとおり、レイヤ相互作用問題が扱えることを確認した。

6.2 新規性・有用性・実現性

COP と関連した方法論は、紙名らのユースケース図と関連ン付けた方法[14]があるが、OS と類似した構造を取り入れたアーキテクチャを採用している方法論は、独自の方法である。レイヤ相互問題に関し、幾つかの研究がある[17-18]。

特に、N. Cardozo らは、複数の活性化されたレイヤを有するレイヤを不活性化する場合の問題について着目し、我々と同様に、活性準備状態と不活性準備状態について述べている。また、ペトリネットは、リフレクション等の実行時書き換えのメカニズムを有していないため、独自のペトリネットを提案し、シミュレーション可能にしている。我々の方法との違いは下記である。

- レイヤ活性・不活性部分のペトリネット化: N. Cardozo らは、プログラムにより隠蔽しているが、我々は、RT-COA に従った仕組みを CP-nets により明示している点である。
- 可読性 : N. Cardozo らはペトリネットの可読性については触れていない。ただし、我々のレイヤ相互作用図の記述と類似している。我々のレイヤ相互作用図は、彼らのペトリネットの可読性を高めるのに役立つ。

以上が、本研究の新規性といえる点である。

有用性と実用性に関して、本研究では、レイヤ活性・不活性部分についてもペトリネット化しているため、実現すれば、様々なレイヤ相互作用問題を扱うことが可能になると期待できる。従って、フォーマル化に関しては、成果が期待できる。一方で、次節で述べるとおり、様々な課題を有しているため、現状の方法では、シミュレーションの内容や性質を限定しなければ、実現は難しいと考える。以上と関連し、N. Cardozo らの着目点は一つの有用な解である。

6.3 今後の課題

以下に、課題について述べる。

(1) **複雑さ** : CP-nets でモデリングする際に課題となるのはモデルが複雑である点である。

a. リフレクションの問題 : CP-nets は実行時書き換えに関する機能を有していない。図 8 に示した Method Process を全てのメソッド呼出しに付す必要がある。
b. 上記と関連し、トークンはレイヤそのものを有することができない。これは、次に述べる検査性を保証するためである。
c. 階層化のための融合プレースとポートプレースは、CP-nets の本質的な性質上、同じプレースにすることができない。これにより、余分なトランジションが追加され、さらに、振る舞いが変化してしまう。

(2) **適合性と検査性** : CP-nets によるモデリングは、プログラミングや UML などのモデルとは大きく異なるため、適合性についての基準を決めていく必要がある。また、検査性に関し、CP-nets は本来のペトリネットに展開できる範囲でのみ、拡張を行っているため、検査可能である。拡張方法が不適切である場合、この性質を損なう危険性がある。

(3) **困難さ** : CP-nets でモデリングは、コンテキスト指向ソフトウェアであるかとは無関係に下記の問題がある。

a. オブジェクト指向化されていないため、カラー宣言がグローバルであり、規模が大きくなるにつれてモデリングが困難になる。
b. CP-nets は、ML を拡張したプログラミング言語により、カラー宣言、関数を作成することができるが、ペトリネットの基本要素であるプレースとトランジションからなる単純な構成要素を有する。これらの基本要素の意味づけは、モデラーに任されている。以上から、何をプログラミングし、ペトリネットの構成要素にするのかといったことを決める必要がある。
c. 上記 d.と関連し、ペトリネットの定石、パターンをモデラーは考えながらモデリングする必要がある。
d. 上記と関連し、学習が容易でないため、大規模化に応じた人員確保が難しい。

以上から、複雑さの解決に関しては、適合性・検査性を踏まえながら、新たな形式ネットの検討も眼中にいれる必要がある。また、モデリングの困難さは、定石のパターン化や、状態遷移図でモデリング可能な箇所のツール化が必要となる。今後、以上を踏まえ研究を行っていく。

参考文献

[1] R. Hirschfeld, P. Costanza, and O. Nierstrasz: Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp. 125-151, 2008.
[2] M. Appeltauer, R. Hirschfeld, and J. Lincke: Declarative Layer Composition with the JCop Programming Language, Journal of Object Technology, Vol. 12, No. 4, 2013.

[3] M. Appeltauer, R. Hirschfeld, M. Haupt, J. Lincke, and M. Perscheid: A Comparison of Context-oriented Programming Languages, Proceedings of the Workshop on Context-oriented Programming (COP) 2009, ECOOP 2009, pp. 1-6, 2009.
[4] R. Hirschfeld, P. Costanza, and M. Haupt: An Introduction to Context-Oriented Programming with ContextS, In Generative and Transformational Techniques in Software Engineering (GTTSE) II, Springer LNCS 5235, pp. 396-407, 2008.
[5] M. Appeltauer, R. Hirschfeld, M. Haupt, and H. Masuhara. ContextJ: Context-oriented programming with Java. Information and Media Technologies, 6(2):399-419, 2011.
[6] J. Lincke, M. Appeltauer, B. Steinert, and R. Hirschfeld, An open implementation for context-oriented layer composition in ContextJS, Computer Program, Vol. 76, No. 12. (December 2011), pp. 1194-1209.
[7] T. Kamina,T.Aotani, H. Masuhara: EventCJ: EventCJ: A Context-Oriented Programming Language with Declarative Event-based Context Transition, AOSD '11 Proceedings of the tenth international conference on Aspect-oriented software development, pp.253-264, ACM, 2011.
[8] I. Tanigawa, N. Ogura, M. Sugaya, H. Watanabe, and K. Hisazumi: A Structure of A C# Framework ContextCS based on Context-Oriented Programming, MODULARITY Companion'15, pp.21-22, 2015.
[9] H. Watanabe, M. Sugaya, I. Tanigawa, N. Ogura, and K. Hisazumi: A Study of Context-Oriented Programming for Applying to Robot Development, Proceedings of the Workshop on Context-oriented Programming (COP) 2015, ECOOP 2015,2015.
[10] H. Watanabe, I. Tanigawa, M. Sugaya, N. Ogura, H. Kenji: A Layer Structure Diagram and a Layer Interaction Diagram towards a Context-Oriented Development Methodology for Embedded System, The Workshop on Live Adaptation of Software Systems(LASSY’16), 2016.
[11] 森谷大輔, 小川英理, 上條弘貴, 渡辺晴美: コンテキスト指向プログラミングのためのレイヤ相互作用図の考察, 信学技報 115(487), pp.125-130, 2016.
[12] K.Jensen, L. M. Kristensen: Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, 2009.
[13] 村田 忠夫: ペトリネットの解析と応用 (アルゴリズムシリーズ), 近代科学社, 1992.
[14] T. Kamina, T. Aotani, H. Masuhara, and T. Tamai, Context-oriented Software Engineering: A Modularity Vision, Proceedings of the 13th International Conference on Modularity, MODULARITY ’14, pp. 85-98, 2014.
[15]T. Kamina, T. Aotani, and A. Igarashi: On-Demand Layer Activation for Type-Safe Deactivation, Proceedings of the Workshop on Context-oriented Programming (COP) 2014, ECOOP 2014, 2014.
[16]T. Aotani, T. Kamina, and H. Masuhara: Unifying Multiple Layer Activation Mechanisms Using One Event Sequence, Proceedings of the Workshop on Context-oriented Programming (COP) 2012, ECOOP 2012, 2012.
[17] N. Cardozo, S. González, K. Mens, R.Van Der Straeten, J. Vallejos, T. D’Hondt: Consistent Activation Semantics of Context-Oriented Systems, Journal of Information and Software Technology (JIST). Elsevier, 58 - 2015, pp. 71--94, 2015.