Vol.2017-EMB-45 No.3
2017/6/30

イベントに着目したカラーペトリネットによるコンテキスト指向ソフトウェアの分析

渡辺晴美†

概要：本稿では，コンテキスト指向ソフトウェア(COS)の振る舞いを確認するために，カラーペトリネット(CPN)に基づいた分析方法のアイデアについて紹介する．COS は，レイヤによりシステム全体を書き換えることが可能であるため，環境の変化に応じてサービスを提供することに適している．一方，レイヤにより衝突や競合等の問題を引き起こす可能性がある．CPN は，並行システムの衝突や競合を分析することに適したモデリング言語であり，レイヤ間の問題分析にも期待できる．また，COS の特徴であるレイヤの活性・不活性をトークンとトークンゲームにより表現することができる．一方，CPN を用いても，実際の環境は複合的で複雑であるため，振る舞いの誤りの原因を分析することは容易ではない．すなわち，COS の振る舞いモデルにおいて，変化するレイヤのどのイベントが失敗したのかを発見し再現することは容易ではない．この課題を解決するために，イベントに着目した CPN による分析方法のアイデアについて紹介する．

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

Event based Analysis on Coloured Petri Nets for Context-Oriented Software

HARUMI WATANABE†

Abstract: The article introduces an analysis based on Coloured Petri Nets (CPN) for conforming behaviour Context-Oriented Software (COS). The token of CPN and the token game are suitable for representing the layer of COS and its (de-)activation. Especially, we expect CPN can detect mutual exclusion faults related to the layer interactions. Even though, if CPN finds a fault on COS, we cannot easily find where is the cause of the fault in the software since actual environments are compounded. Thus, in a behavior model of COS, we cannot find easily which event fails in changing layers. In this paper, we focus on two issues that are called compound environments, and context events. To overcome these issues, firstly, we clarify the advantages of CPN to state machines. Secondly, the paper introduces an analysis based on these advantages. In this testing environment, the state space graph of CPN contributes to finding a cause why fault occurred.
.

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 時代のシステムの特徴とも言える．コンテキスト指向プログラミング言語(COP: Context-Oriented Programming)は，この特徴に着目した言語である．COP の特徴は，環境の変化，すなわちコンテキストの変化に応じてシステム全体の振る舞いを変更することを支援することにあり，様々な言語が提案され，議論されている[1-13]．本稿では，環境に応じて，レイヤと呼ぶ単位でシステム全体を書き換える COP の概念に基づいたソフトウェアをコンテキスト指向ソフトウェア(COS: Context-Oriented Software)と呼ぶ．

実際の環境は複雑であるため，本質的に，適切に環境の判断，すなわちコンテキストを判断することは容易ではない．加えて，COS では，実行時にレイヤによりソフトウェアを書き換えるため，誤りの分析が容易ではないという問題がある．この問題を解決するために，これまでカラーペトリネット(CPN: Coloured Petri Nets)[16,18]を用いた方法を提案してきた[14-15]．本方法は，レイヤに応じたメソッド呼び出しを CPN で表現することで，意図通り振る舞いが変更したかどうかについて確認することについて貢献した．本方法を用いても，どのようなコンテキストの判断により，レイヤが選ばれたのかを分析することはできない．本稿では，複合環境とイベントに着目することにより，本問題について解決するアプローチについて紹介する．

†1 東海大学
Tokai University

以下，2 章では，まず適用例である協調ロボットの例題について説明する．3 章では，本稿で扱う問題について明らかにする．

2. 例：協調掃除機ロボット

本稿で扱う課題，および紹介する方法を説明するために，本章では，図 1 に示す協調掃除機ロボットについて述べる．協調掃除機ロボットは，図 1 の左側に示すとおり，複数の自動掃除機がネットワークに繋がっている．右側に示すとおり，各掃除機は異なる種類のフロア上で様々な種類の汚れを，効率良く，掃除することを想定している．

COS と関連し，本例題での環境は，汚れ方，床，そして自分以外の掃除機である．これらに応じて，掃除機は振る舞いを変更する．例えば，同じ領域に，多くのロボットが集まってしまった場合は，ロボットが少ない領域を探して移動する．別な例では，汚れが酷い場所があれば，応援を呼ぶ．

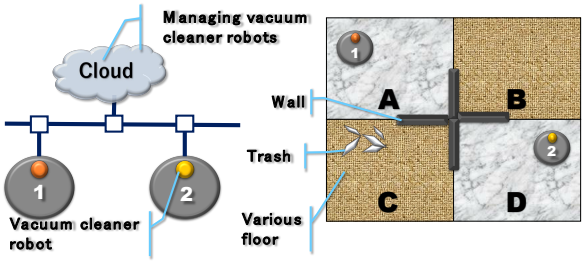


図 1　協調掃除機ロボット

3. 課題

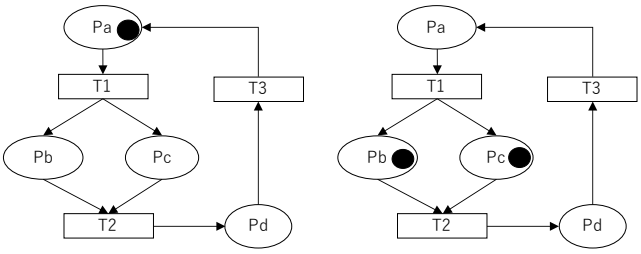
本章では，本稿で扱う課題について述べる．本稿で扱う課題は，「コンテキストの判断の分析」にある．その難しさとして，複合環境の再現，コンテキストイベントの原因分析に着目し，以下で述べる．

3.1 複合環境の再現

実際のシステムを取り巻く環境は，述べるまでもなく複雑である．前述の掃除機においても，実際の自動掃除機は，障害物を避け，様々な種類の床で，様々な種類の汚れを掃除しなければならない．障害物は椅子や机などの固定されたものばかりではなく，人や動物など動く障害物もある．システムに取り付けられるセンサは限られており，適切な複合環境を識別することは容易ではない．本問題について着目した研究[20]もあるが，本稿では，こうした環境の再現性について着目する．上記のとおり複雑であるため，再現することも容易ではない．

3.2 コンテキストイベントの原因分析

環境が変化した際に，適切な振る舞いに変更しない，すなわち，想定した通りにレイヤが(不)活性化されない場合，その原因を探ることは容易ではないと考える．レイヤの(不)活性化は，環境の変化，すなわちコンテキストの変化を識別してから行われる．コンテキストを識別するには，通常，複数のセンサからの入力を集め，複数のメソッドが処理することで判断する．加えて，レイヤの(不)活性化が途中過程に加わると，その解析は容易ではない．



(a) T1 発火前　　(b) T1 発火後

図 2　ペトリネット

4. カラーペトリネット(CPN)

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(CPN: Coloured Petri Nets)について概説する． CPN は，K. Jensen が提案したグラフ形式モデリング手法である[16, 18]．C. A. Petri のペトリネット [17] 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，CPN について概説し，状態遷移図に対する CPN の特徴について述べる．本特徴を利用し，3 章の課題を解決する．

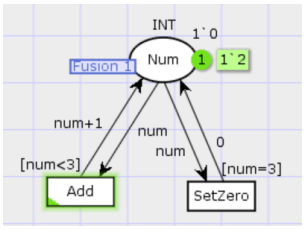
4.1 ペトリネット

ペトリネットを図 2 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 2(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 2(a)の T1 および(b)の T2 が発火可能である．図 2(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．

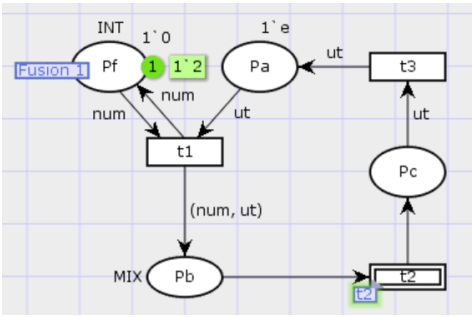
ペトリネットの目的は，デッドロック等の並行システムの問題発見にある．発火によるシミュレーションに加え，シミュレーション結果を表した到達グラフ，プレースとトランジションの接続行列を作成し解析する方法がある．

4.2 CPN

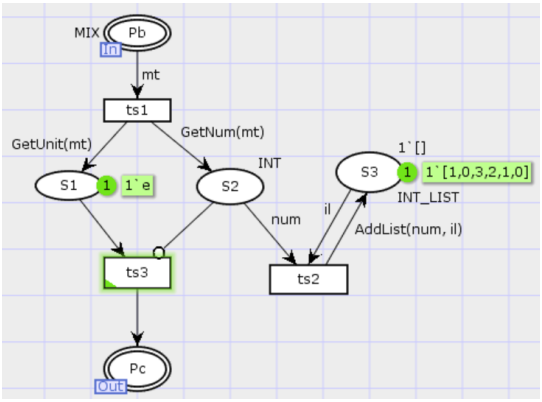
図 3 に CPN の例を示し，カラー宣言を図 4 に記す．(a)～(c)は，0～3 を数え上げ，数え上げた数をリストにするソフトウェアをモデリングしている．サブネット A は，0～3 をカウントし，3 になるとゼロに戻る．サブネット B は，主要関数にあたる部分で，サブネット A と C を関連付ける．サブネット C は，0～3 の数のリストを作成する．前述の通り CPN はペトリネットのトークンに型による色付けと階層化を特徴としている．以下，トークンカラー，階層化について概説する．



(a) サブネット A



(b) サブネット B



(c) サブネット C

図 3　カラーペトリネット

```
▼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(I, Il) = I::Il;
▼fun GetNum(I:MIX) = #1 I;
▼fun GetUnit(I:MIX) = #2 I;
```

図 4　図 3 のカラー宣言

- **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 3 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**：CPN 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 3 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．

4.3 CPN と状態遷移図

下記で，本稿と関係のある CPN の特徴について述べる．

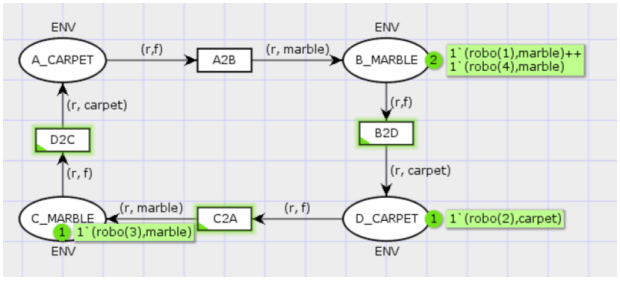
(1) カラートークン

CPN は同じ処理を有する複数のシステムが協調動作をする際に，一つの CPN モデルに，各システムをトークンとして配置することができる．その様子を図 5 に示す．図 5 は図 1 の例題を表しており，カーペットと大理石の床を 4 台の掃除機が動作していることを表す．B_MARBLE に 2 台の掃除機があり，D_CARPET と C_MARBLE に 1 台ずつ掃除機が配置されている様子を表している．このモデルでは床ごとに異なる掃除を表すことを示している．

(2) 複数の関心事の表現

状態遷移図の場合，関心事が状態のみにあるが，カラーペトリネットでは，カラー宣言により，着目する関心事を明示的に表すことが可能である．図 6 は，状態に加えてイベントにも着目した様子を示している．

Vol.2017-EMB-45 No.3
2017/6/30




```
colset ROBOT = index robo with 1..5;
colset FLOOR = with carpet | marble;
colset ENV = product ROBOT * FLOOR;
var r:ROBOT;
var f:FLOOR;
```

図 5　メソッドの切り替え

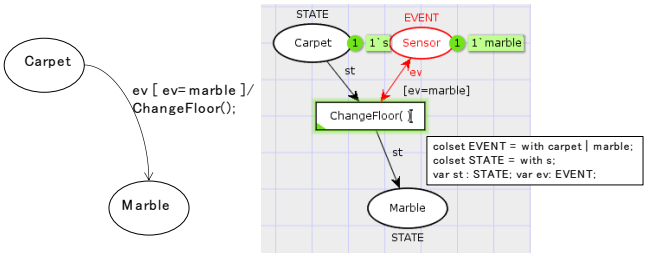


図 6 状態遷移図と CPN

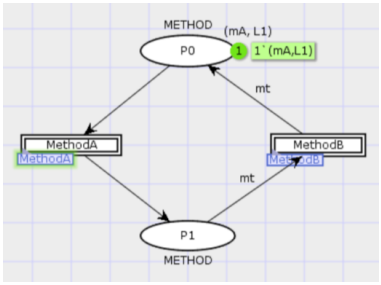
5. これまでの取り組み

本章では，これまでの取り組みについて紹介する．COS では，活性化されたレイヤにより振る舞いを変更する．これまで，レイヤの切り替えに応じて，メソッドが切り替わる CPN をモデリングした[15]．以下，図 7～8 を用い，その仕組みについて述べる．

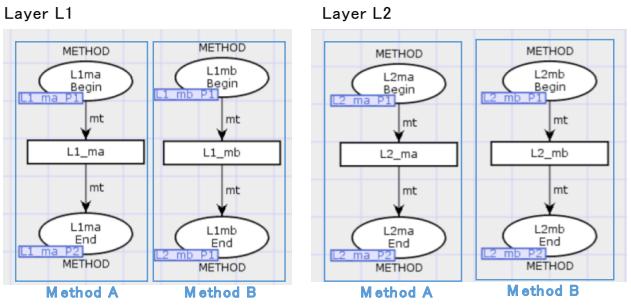
図 7(a)の CPN は，MethodA()と MethodB()を繰り返し実行する．これらのメソッドは階層化されており，メソッドの本体が，図 7(b)である．(b)の CPN は，L1 と L2 の 2 つのレイヤを持つ．各々のレイヤは MethodA()と MethodB()の両方を持ち，図 7(a)は現在活性化されているレイヤのメソッドを実行する．

通常の CPN の階層化では，図 7(a)の MethodA()は，メソッド本体を表すネットと直接結び付く．我々の方法では，図 8 に示す CPN により現在のレイヤを判断し，対応するメソッドを呼び出す．我々の方法では，全てのメソッド呼び出しは，メソッド毎に，図 8 に示すようなレイヤに対応したメソッド呼び出しを判別する部分ネットを経由し，メソッドボディと結びつく．

レイヤの判断は図 8 の Address プレースにあるトークンの値により行う．Address プレースのトークンが L1 であれば，左側の L1Begin トランジションが発火し，L1 にあるメソッドボディが実行される．尚，Address プレースは融合プレースであり，レイヤの活性不活性を行うネットと関連付いている．Address プレースは，ADDRESS カラーのトークンを受け取ることができる．図 7 のカラー宣言をに示すとおり ADDRESS はレイヤを表すトークン L1 と L2 を持つカラーである．実際には，アドレスの決定にはクラス情報や活性不活性に関する情報等も必要であり，カラー宣言はこれらの直積で表される．ここでは説明を簡単にするために，レイヤ名のみのトークンを定義するカラーとしている．



(a) メソッド呼び出し



(b) 各レイヤのメソッド本体

```
colset ADDRESS = with L1 | L2;
colset METHOD_NAME = with mA | mB;
colset METHOD = product METHOD_NAME * ADDRESS;
var mt:METHOD;
var new_mt:METHOD;
fun new_addr(mt:METHOD, addr:ADDRESS)
 = (#1 mt, addr);
```

図 7　レイヤと振る舞い

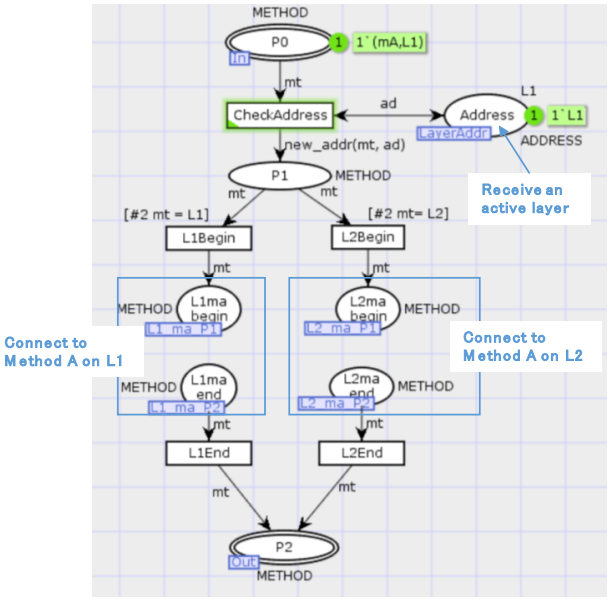


図 8　メソッドの切り替え

6. 環境の再現とイベントの分析

本章では，3 章で述べた課題を解決するために，CPN による複合環境の再現およびコンテキストイベントの分析方法について考察する．図 9 に示すとおり，本方法は，(1) CPN と状態マシンを関連づけ，(2) CPN の連結，(3) シミュレーションからなる．以下，各々について述べる．

6.1 CPN と状態遷移図

状態遷移図を設計時に記述することは，一般的であり，多くの技術者が記述できる．一方，CPN は一般的とは言えず，発火可能性とカラーに注意を払うという煩雑さも増える．これらの煩雑さにより，ある程度，パターンとして整理していなければ，CPN の記述は難しい．状態遷移図の組み合わせ，解釈により，CPN を構築することは可能であるため，本方法では，状態遷移図から CPN へ変換する．また，同様の理由から UML の状態マシンから CPN を変換する方法は[19]に代表されるように多数提案されている．

変換の際に，3 章で述べたイベントの分析を可能にするために，4.3(2)の特徴を活かし，状態遷移図のイベントから，プレースとそのカラーを抽出する．CPN の各要素は下記の手順で抽出する．

(1) プレース： 状態をプレースと関連づける．また，上記の通り，イベントと関連したプレースを関連付ける．
(2) カラー：図 6 のカラー宣言に示すとおり，状態を表すカラーENV_STATE，イベントを表すカラーEVENT を宣言する．プレースのカラーも，状態を表すプレース，イベントを表すプレースに，各々STATE と EVENT を関連づける．
(3) トランジション：状態遷移図の遷移に基づいてトランジションを抽出し，ガード条件を付随させる．
(4) アーク：プレースとトランジションをアークで結びつける．また，アークには変数が必要である．この変数はトークンを渡す役割とともに，トークンを加工する役割を持つ．アークの変数は，結びつけたプレースのカラーで変数宣言する．

6.2 CPN の連結

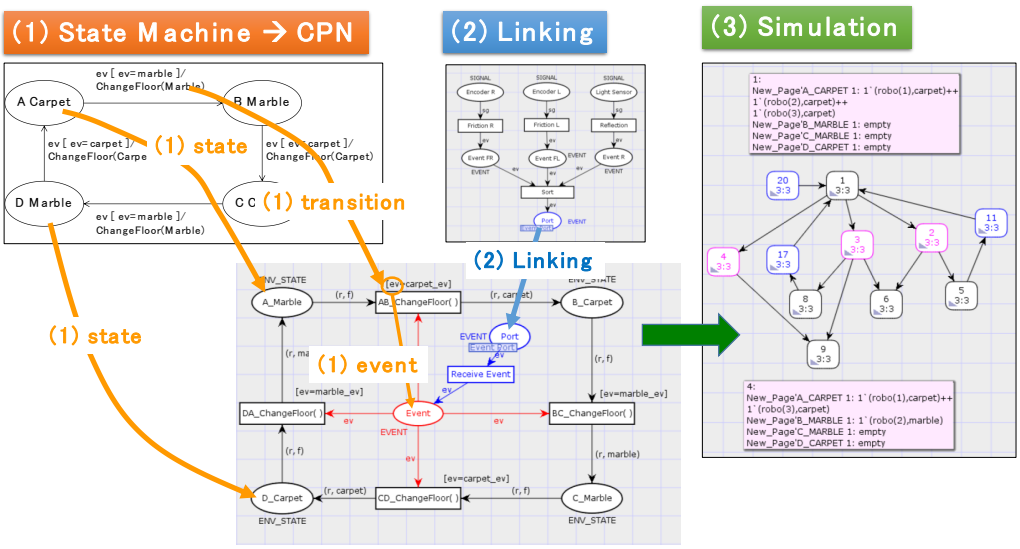
想定どおりにレイヤの(不)活性，すなわち想定どおりの振る舞いにならなかった場合の分析には，レイヤの(不)活性の時点から，過去にさかのぼる必要がある．レイヤの(不)活性は，センサが獲得した値を処理，判断し，環境が変化した，すなわち，コンテキストが変化したという判断結果に基づき行われる．従って，レイヤが想定どおりに(不)活性化しない場合，この処理をさかのぼって分析できる必要がある．本稿では，このレイヤの(不)活性に関わるイベントのみをイベントと呼んできたが，このセンサからイベントに至る振る舞いについてモデリングし，6.1 で述べた CPN と関連づける．

6.3 シミュレーション

6.1 と 6.2 の CPN により，レイヤの(不)活性と関連したイベントと状態を表現した．振る舞いの分析するためには，シミュレーション，すなわち，トークンを動かした結果が必要である．CPN では，図 9(3) Simulation に示す状態空間グラフ(State Space Graph)を生成することができる．状態空間グラフは，発火グラフ(Occurrence Graph)あるいは到達(可達)グラフ(Reachability Graph)とも呼ばれている．このグラフをもとに分析を行う．

状態空間グラフのノードは，ある瞬間のマーキング，すなわち，どのようなトークンがどのプレースに置かれているかということを表している．

図 9(3)に示すとおり，状態空間グラフの各ノードは，3 つの数字を持つ．各ノードの上側に 1 つだけ記された数字は，ノードの ID であり，下側に記されたコロンで区切られた 2 つの数字は，そのノードに至る可能性のあるノードの数と，ノードの状態から発火して次に移行可能なノードの数である．各ノードはマーキングの詳細な情報を有している．図 9(3)では，ノード 1 とノード 4 の詳細情報を示している．ノード 1 は，A_CARPET プレースに 3 つのトークンがあり，他のプレースが空である状態を表している．ノード 4 は，A_CARPET プレースに 2 つのトークンがあり，B_MARBLE プレースに 1 つのトークンがあることを示している．問題が起きたノードを起点に，関係のある EVENT カラーのプレースについて，状態空間グラフを逆方向に辿っていく．




```
colset ROBOT = index robo with 1..5;
colset FLOOR = with carpet | marble;
colset ENV_STATE = product ROBOT * FLOOR;
colset EVENT = with carpet_ev | marble_ev
var ev:EVENT;
var s: ENV_STATE;
var r:ROBOT;
var f:FLOOR;
```

図 9 CPN によるイベント分析方法

7. おわりに

本稿では，コンテキスト指向ソフトウェアの振る舞いを確認するために, CPN に基づいた分析方法のアイデアについて紹介した．CPN を用いることで，レイヤ間の衝突や競合等の問題を発見し，さらに，状態空間グラフを用いた分析を行うことで，誤りの原因分析を可能にすることを目標としている．

前節で述べた状態遷移図から CPN への自動変換を可能にするためには，まず，想定した COS の動作通りに CPN を動かすことを確保しなければならない．振る舞いの比較は状態空間グラフで弱双模倣性を調べれば良いが，同じように振る舞うように振る舞うモデルを構築する必要がある．これには，発火規則を考慮しながらモデリングすることにより，状態遷移モデルにはない要素が出現し冗長なモデルになる．本研究は，オリジナルの CPN で現在モデリングを行なっているが，リフレクションを有したペトリネットによるモデリング方法もあり，このようなペトリネットを用いた場合，冗長なモデルになることを防ぐ効果が期待できる一方，静的な解析が難しくあることが予測される．

また，CPN の連結で述べたモデリングについては，コンテキスト判断と関連するため，状態遷移図ではないモデル等を検討する必要がある．シミュレーションについては，今後，関係のある要素のみを自動的に抽出する方法について検討していく予定である．

参考文献

[1] G. Salvaneschia, C. Ghezzia, M. Pradellaa: Context-oriented Pro-gramming: A Software Engineering Perspective, Journal of Systems and Software archive, Vol.85 Issue 8, pp. 1801-1817, 2012.
[2] R. Hirschfeld, P. Costanza, and O. Nierstrasz: Context-oriented Programming, Journal of Object Technology, Vol. 7, No. 3, pp. 125-151, 2008.
[3] M. Appeltauer, R. Hirschfeld, and J. Lincke: Declarative Layer Composition with the JCop Programming Language, Journal of Object Technology, Vol. 12, No. 4, 2013.
[4] 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.
[5] R. Hirschfeld, P. Ccostanza, 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.
[6] M. Appeltauer, R. Hirschfeld, M. Haupt, and H. Masuhara. ContextJ: Context-oriented programming with Java. Information and Media Technologies, 6(2):399-419, 2011.
[7] 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, 2011.
[8] 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.
[9] 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.

[10] 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.
[11] 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.
[12] N. Cardozo, S. González, K. Mens, R.Van Der Straeten, J. Vallejos, T. D'Hondt: Consistent Activation Semantics of Context-Oriented Sys-tems, Journal of Information and Software Technology (JIST). Else-vier, 58 - 2015, pp. 71--94, 2015.
[13] N. Cardozo, S. Gonzalez, K. Mens, T. D'Hondt: Uniting Global and Local Context Behavior with Context Petri Nets,Proceedings of the Workshop on Context-Oriented Programming (COP) 2012, ECOOP 2012, 2012.
[14] H. Watanabe, M. Sugaya, I. Tanigawa, N. Ogura, and K. Hisazumi: A Study of Context-Oriented Programming for Applying to Robot De-velopment, Proceedings of the Workshop on Context-oriented Pro-gramming (COP) 2015, ECOOP 2015,2015.
[15] H. Watanabe, I. Tanigawa, N. Ogura, M. Sugaya, K. Hisazumi and A. Fukuda: Coloured Petri-Nets Framework for Simulating Method Invocations on Context-Oriented Software, Proceedings of the Workshop on Meta-Programming Techniques and Reflection (META) 2016, SPLASH 2016,2016.
[16] K.Jensen, L. M. Kristensen: Coloured Petri Nets: Modelling and Validation of Concurrent Systems, Springer, 2009.
[17] W.Reisig. Petri Nets: An Introduction, Vol.4 of EATCS Monographs on Theoretical Computer Science. Springer, 1985.
[18] CPN tool web page: http://cpntools.org/
[19] D. Buchs, L. Pedro, and L. L ´ ucio, Formal Test Generation from UML Models, Dependable Systems, LNCS 4028, pp. 145-171, 2006.
[20] T. Petricek, D. Orchard, A. Mycroft: Coeffects: A calculus of context-dependent computation, ICFP '14, Volume 49, Issue 9, pp. 123-135, 2014.