オブジェクト指向ソフトウェアテストのための カラーペトリネットスライシング技術

渡辺晴美†

オブジェクト指向ソフトウェアは，生成，削除，継承，動的束縛など実行時に決定される性質を持つ．これらの性質は状態爆発問題を起こすため，ソフトウェアの挙動に関するテストや解析を困難にしている．これらの性質を持つソフトウェアを効率的にテストするために，我々は状態図とクラス図で記述した仕様をカラーペトリネット（CPN）へ変換するテスト方法を提案した．CPN は拡張ペトリネットの１つであり，状態爆発問題を解決する形式技術として知られている．しかし，この技術を利用しても大規模システムに対しては十分に効果的にテストを行うことは難しい．既存の単体テストはメソッドやクラス単位であり，複数のクラス間やメソッドによって実現された機能を状態図レベルでテストする方法はない．本論文では，CPN のためのスタティックスライシング技術を提案することで，機能ごとにテストすることを可能にする．機能ごとにテスト可能にすることで，従来よりも効率的にそして開発初期段階でのテストを可能にする．

A Static Slicing Technique of Coloured Petri Nets for Object-oriented Software Testing

Harumi Watanabe†

Object-oriented software essentially has dynamic property such as object creation, deletion, inheritance, dynamic binding. This property makes behavior analysis and tests of object-oriented software significantly complicated. To enable efficient testing of the above property, we proposed a method which uses Colored Petri Nets (CPN), CPN is an extended version of Petri Nets, one of formal techniques allowing to avoid the net-explosion problem. However, it is difficult to test efficiently by this method in a large-scale system. The existing unit testing is performed for every method, instance, or class. There is no method of testing a function over between classes. In this paper, we propose a static slicing technique for the CPN. It becomes possible to test the function implemented between classes by this method. Furthermore we can efficiently detect errors peculiar to object-oriented software, in the early stage of development.

1. はじめに

ソフトウェア開発においてテスト，デバッグ，解析は不可欠な工程であり，莫大なコストが必要である．リリース後に発見した誤りを修正するコストは，設計時に発見した場合と比較し 4 倍もかかるといわれており[23]，なるべく早い開発段階で発見できることが望まれている．

昨今のソフトウェアの大規模化，複雑化にともない，オブジェクト指向開発法やオブジェクト指向モデリング言語，プログラミング言語などを利用したオブジェクト指向開発が普及しつつある．本論文では，オブジェクト指向開発によって得られるプロダクトをオブジェクト指向ソフトウェアと呼ぶ．オブジェクト指向ソフトウェアは，継承，動的束縛，並行性など実行時に決定される性質を持つため，状態爆発問題から網羅的に挙動をテストすることは困難である．我々は，この問題を解決するために，Jensen によって提案されたカラーペトリネット（Coloured Petri Nets: CPN）[13]を用いたテスト方法を提案してきた[1),5)]．

CPN は並行性をテスト検証するペトリネット[22]を拡張したネット指向言語であり，階層化の概念，カラーと呼ばれる型付きの値を持つトークン（以下，カラートークン）の概念，そして時間の概念が拡張されている[13),14)]．これらの拡張された概念により，CPN は様々な性質を持つソフトウェアを記述でき，記述したモデル，シミュレーション，検証は様々な産業界の分

† 立命館大学理工学部情報学科
Department of Computer Science, Ritsumeikan University

野で用いられている[15]．

CPN は，カラートークンをオブジェクトと対応付けることができ，CPN のトークンの生成，削除機能をオブジェクトの生成，削除に関連付けることができる．さらに，カラートークンを利用することで，継承や動的束縛のメカニズムを表現することも可能である．

オブジェクト指向モデリング言語として広く用いられている UML[9]のステートチャートでは，並行性やオブジェクトの生成，削除を記述できるが，挙動に関する規則が十分に定められていない．したがって，オブジェクト間にわたる挙動，すなわちメソッドやオブジェクトごとに記述したステートチャートが協調して動作する場合の挙動に関するテストを行うことはできない．我々は，ステートチャートを CPN へ変換し，テストを行う方法を提案してきた[1),5)]．本論文では，この変換した CPN を OO-CPN と呼ぶことにする．

CPN を用いて挙動のテスト検証を行う際，到達グラフと呼ばれる実行の軌跡を状態図で表現し，可達解析を行うが，オブジェクト指向ソフトウェアではその状態図が実行時に決定される性質のために，状態爆発を起こすという問題が起きる．CPN は，この問題を同値仕様という方法で解決している．しかし，同値仕様の記述はテスト施行者が行うため，適切に記述することは困難であり，到達グラフを十分小さくできる同値仕様が存在することも保証できない．機能別に部分テストを行う場合でも，到達グラフを生成する必要があるため，同値仕様を上手く記述できなければ，効率的なテストは困難である．

本論文では，カラーペトリネットのスタティックスライシング方法を提案することで上記の問題を解決する．この方法は，プログラミング言語のプログラムスライシング[24)]をカラーペトリネットに応用した方法である．プログラムスライシング技術は，プログラミング言語の世界では，オブジェクト指向プログラミング言語にも適用され[19)]，テスト，デバッグ，保守に広く利用されているが，状態図やペトリネットへの適用は提案されていない．プログラムスライシングはプログラムスライスを求める技法のことであり，プログラムスライスとは，ある命令の実行に影響を与える可能性のあるすべての命令の集合のことである．プログラムスライスを得ること，すなわち，ある命令の実行に影響を与える部分プログラム得ることは，プログラムの中からある機能を実現している部分のみを取り出すことが可能であるといえる．この技術により，プログラムを機能別に理解することが可能になる．したがって，スライシングにより機能に着目したテストが可能になる．

近年のプログラムスライシング技術では，System Dependence Graph（SDG）や Class Depedence Graph（CDG）を利用した方法が一般的である[12),19)]．OO-CPN スライシングを得るのに特徴となる箇所は，データ依存と制御依存を求める箇所である．本論文では，データ依存と制御依存を求める方法について議論し，これらのグラフの構築については扱わない．

本論文で提案する方法により，プログラムスライシングの場合と同様，CPN からある機能に着目した部分ネットを抽出することを可能にする．OO-CPN をスライシングすることにより，オブジェクト間にまたがる機能をテストすることを可能にする．これにより，同値仕様のみに依存していた状態爆発問題を解決する．さらに，本方法は CPN を実行せずにスライスを得るため，開発途中であっても，記述が完了した機能に関してはテストを行うことができる．これにより，従来よりも開発早期の段階に誤りを発見することが可能になる．

以下，2 章では OO-CPN スライシングを用いたテスト方法の概要について述べる．3 章では，CPN と，オブジェクト指向ソフトウェアから CPN へ変換する方法について概説する．4 章で，OO-CPN スライシング方法について提案する．5 章では，提案した方法を例題に適用し評価を行う．6 章では，関連研究との比較を行う．最後に 7 章で本論文のまとめと，今後の展望について述べる．

2. OO-CPN スライシングを用いたテスト方法

本章では，OO-CPN スライシングを用いたテスト方法について概説する．**図 1** は，テスト方法のプロセスであり，以下はその説明である．

（1）開発者はステートチャートとクラス図を記述する．ステートチャートはメソッドごとに記述する．

（2）開発者はステートチャートの遷移にテスト基準を印付けする．すなわちテスト基準となる遷移のイベントに影響するイベント列についてテスト可能にする．本論文では，このテスト基準の実現を機能の実現と見なしている．

（3）テストマシン（OO-CPN テスタ）は，ステートチャートとクラス図を OO-CPN へ変換する．この変換方法は，すでに我々がで提案した方法[1),5)]を用いる．

（4）OO-CPN テスタは，テスト基準を OO-CPN 上のスライシング基準へ対応付ける．

(5) OO-CPN テスタは，OO-CPN スライスを計算する．

(6) Desing/CPN は到達グラフをスライスから生成する．Desing/CPN は CPN を記述し解析できる既存のツールである[13]．

(7) 可達解析を行う場合，Design/CPN は到達グラフを利用して可達解析を行う．この解析により，デッドロックの検出を行うことができる．

(8) テストケースを利用したテストを行う場合，OO-CPN テスタは，到達グラフからテストケースを生成する．テストケースの生成は，我々が提案した方法[2),6)]を利用する．

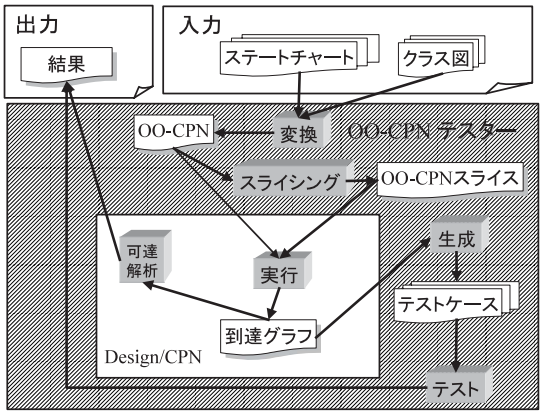

図 1 OO-CPN のテスト方法
Fig. 1 Testing process of OO-CPN.

3. CPN と OO-CPN

本章ではカラーペトリネット（Coloured Petri Nets: CPN）について紹介し，オブジェクト指向ソフトウェアを CPN へ変換する方法について概説する．

3.1 CPN

図 2 に CPN の例を示す．CPN は，図 2-1 中の長方形内に記述したカラー宣言（color declaration）部分とグラフ部分からなる．カラー宣言部分では，カラーと呼ばれる型と変数の宣言を行う．図 2-1 では，int 型のカラー Int が宣言され，Int 型の変数 arg1，arg2 の宣言を行っている．

グラフ部分は図 2-1 に示すとおり，プレース，カラー，トランジション，ガード，アーク，アーク式，時間からなる．CPN 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．図 2-1 のトランジション Operation が発火すると，プレース arg1 と arg2 にあるトークンはプレース Result に移動する（図 2-2 参照）．

発火は，発火可能なトランジションのうち 1 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のアーク式を満たすように，アークの始点となるプレースがトークンを有し，そのトークンの値がトランジションに添付されているガードも満たす場合のことである．

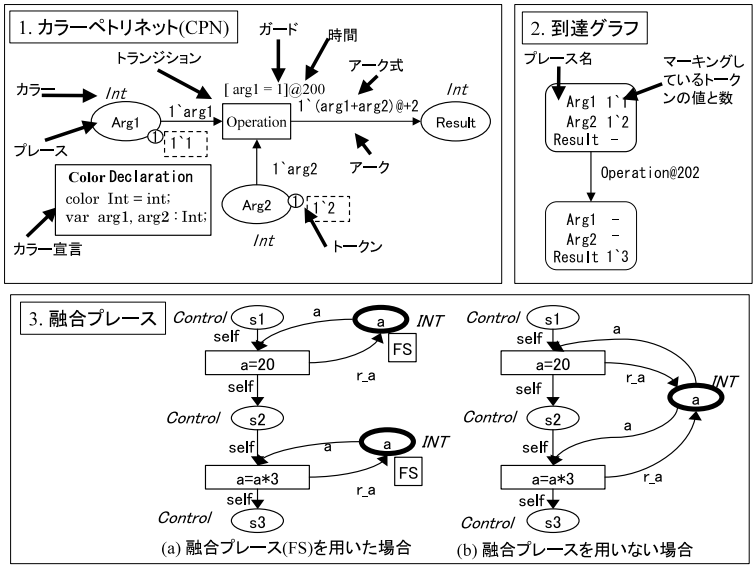

図 2 CPN，到達グラフ，融合プレースの例
Fig. 2 An example of CPN, occurence graph and fusion places.

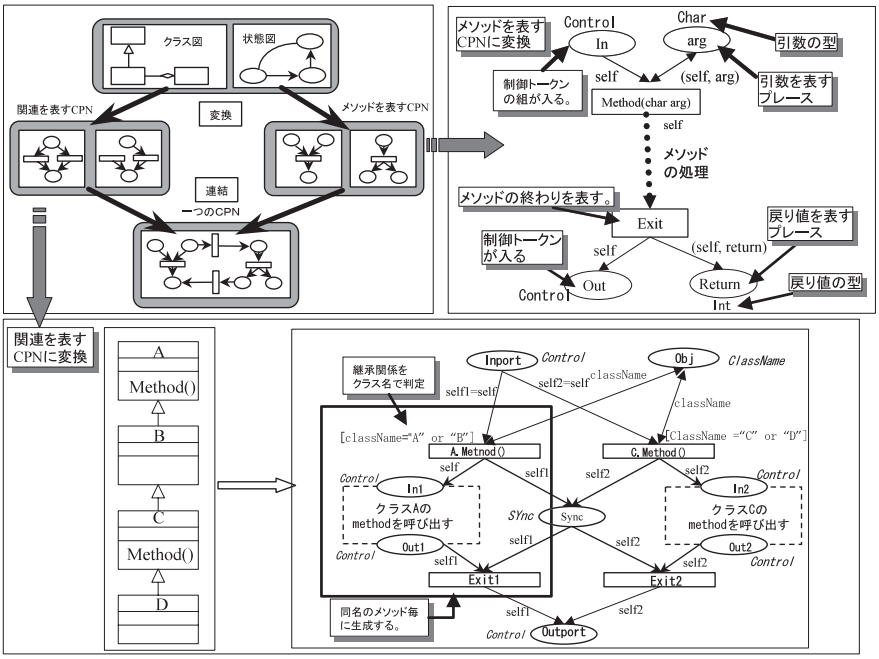

図 3 オブジェクト指向ソフトウェアの CPN への変換
Fig. 3 Translation of OO software into CPN.

CPN には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．融合プレース（FS: Fusion Sets place）の例を図 2-3 に示す．図 2-3 (a) と 図 2-3 (b) は同じ意味である．FS が添付された同じ名前のプレースは融合プレースであり，表現上は 2 個のプレースであるが，実際は 1 つのプレースである．

CPN の解析は到達グラフを用いて行う．図 2-1 に示した CPN の到達グラフを図 2-2 に示す．図 2-2 の到達グラフのノードは，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表す．アークは発火したトランジションを表し，遷移に 202 タイム経過したことを示している．遷移時間は，図 2-2 のトランジションとアーク式に添付された@マーク後の数値が計算される．

3.2 オブジェクト指向ソフトウェアから CPN への変換

図 3 にオブジェクト指向ソフトウェアから CPN への変換概要を示す．変換の方法は以下のとおりである（詳細は文献 1)，5) を参照）．

(1) クラス中のメソッドを CPN へ変換する．メソッドごとにステートチャートが記述され，メソッドとクラス間の関係がクラス図によって明らかになっているものとする．

(2) 関連を CPN へ変換する．関連は，メソッド呼び出しとしてステートチャート上に実現されているものとする．メソッド呼び出しは，図 3 の右下に示した CPN へ変換する．イベントとアクションはトランジションに変換され，変数はプレース，値はデータトークンになる（**図 4**-2 参照）．

(3) 最初のステップで得たメソッドごとのメソッド処理を表す CPN を次のステップで得たの関連を表す CPN により連結する．

OO-CPN（オブジェクト指向ソフトウェアから変換した CPN）のトークンは，データトークンとコントロールトークンの 2 種類である．図 4-1，図 4-2 に各々のトークンの構成について示す．

データトークンは，データを表すトークンであり，変数を表すプレースに置かれる．データトークンの値が変数の値となる．コントロールトークンは Control プレースに置かれ，2 つの役割を持つ．1 つ目はプログラムカウンタのような役割であり，処理を 1 つずつ進める働きを持つ．データトークンの前半部分にはコントロールトークンと同様の値を持ち，どのオブジェクトの値を変更したかを区別している．コントロールトークンはクラス名とインスタンス番号を持つため，これらによりオブジェクトを識別する．

実際のコントロールトークンは，図 4-1 に示すとお

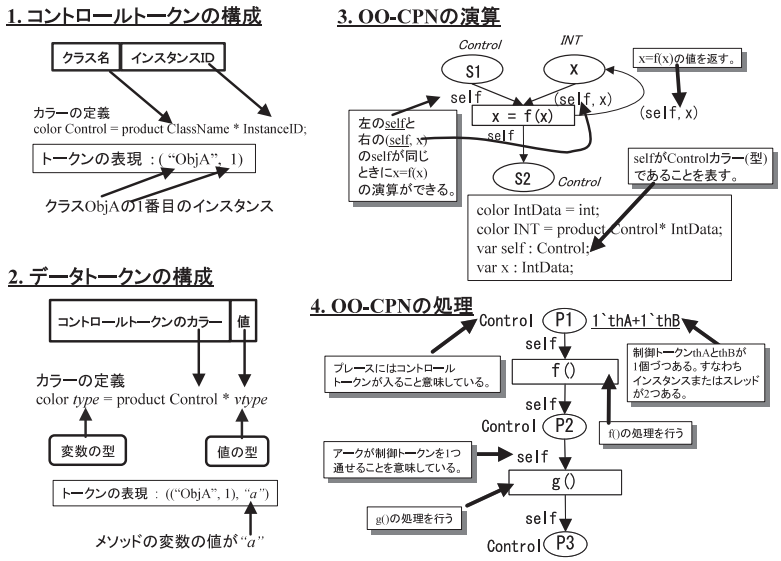

図 4 コントロートークン，データトークン，処理
Fig. 4 Control token, datatoken, and calculation.

り，複数のパラメータを持つが，図を簡略するために，今後は単に self と呼ぶラベルで表す（図 4-3，4 参照）．図 4-3 で，`x=f(x)` トランジションが発火すると，コントロールトークンは，プレース `S1` から `S2` へ移動値をする．`self` に対応するトークンに対応するデータトークン `(self, x)` の `x` の値が `f(x)` になる．

コントロールトークンのもう 1 つの役目は，トークンの値にオブジェクト名を持つことにより，実行時に決定される性質を表現することである．図 3 のように継承関係があるメソッド呼び出しがある場合，図 3 の右下の CPN に変換され，トークンの値が示すオブジェクトに対応するメソッドを呼び出す．

4. OO-CPN のスタティックスライシング

本章では，OO-CPN のスタティックスライシング方法について提案する．まずプログラムスライシングに関してであるが，プログラムポイント p における変数 x に関するプログラムスライスは，プログラムポイント p の変数 x に影響を与えるプログラムのすべての文である[12]．プログラムポイントはスライシング基準として知られている[24]．Weiser は，スライシング基準を以下のとおり定義している．プログラム P の文が i であり，V が P の変数の部分集合である場合，プログラム P のスライシング基準を $< i, V >$ とする．

以上にならい，OO-CPN のためのスタティックスライシングのスライシング基準は以下のとおりである．OO-CPN N のスライシング基準は，N の任意のトランジションを t とし，D を N のデータプレースの部分集合とした場合，$< t, D >$ である．このトランジションを基準トランジションと呼ぶ．データプレースは属性または変数を表す．**図 5**-1 において，プレース “a” と “b” はデータプレースである．

OO-CPN のスライスは，プログラムスライスを得る場合と同様，データ依存と制御依存を求めることによって得られる．データ依存と制御依存関係を図 5-1, 2 に示す．スライスは以下の手順によって得る．

(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命令トランジション（開始トランジション）と，スライシング基準となる命令トランジション（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を抽出する．このプレースをデータ依存プレースと呼ぶ．

(b) データ依存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を選択する．このトランジションをデータ依存トランジションと呼ぶ．

(c) データ依存トランジションにを基準トランジションとしてデータ依存関係を調べる．

スライスの構成要素は，データ依存プレース，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コントロールプレースと出力するコントロールプレースである．

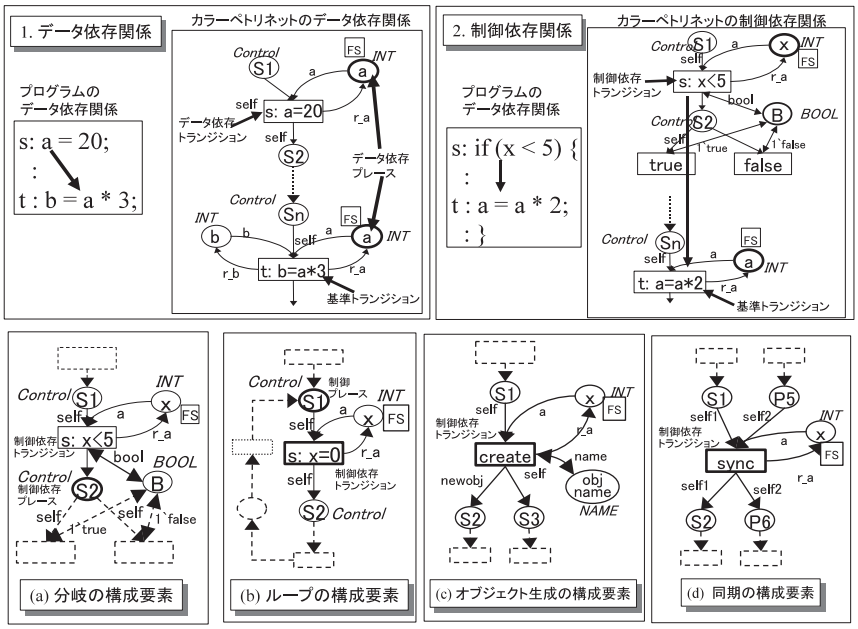

図 5 OO-CPN のスタティックスライシング
Fig. 5 Static slicing of OO-CPN.

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．

データ依存関係が他のメソッドに依存している場合も，上記とほぼ同様の方法によりデータ依存関係を得ることができる．共有変数や大域変数を表す変数は，メソッド間であっても同じ融合プレースによって表されるため，上記と同じ方法を利用することができる．引数やメッセージに関しては，引数プレースによって表され，関連を表す CPN（図 3 の右下，コントロールトークンにより，メソッドを選択する CPN）内で関連付けられ，メソッド呼び出し側と呼び出される側が関連付けられている．したがって，引数プレースに関して上記の方法を適用することで引数に関するデータ依存関係を求めることができる．

（3）制御依存関係を求める．

（a）制御依存プレース，制御依存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の方向に各アークを逆順にたどり，複数の入力アークまたは出力アークを持つコントロールプレースを抽出する．このコントロールプレースを制御依存プレースと呼ぶ．制御依存プレースへ入力するトランジションは，分岐の条件判定式，ループの開始または終了の条件判定式である．これらのトランジションを制御依存トランジションと呼ぶ．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トランジションを抽出する．これらのトランジションも制御依存を発生させるトランジションであるため，制御依存トランジションと呼ぶ．コントロールプレースは，プログラムカウンタのような役割を果たすため，制御依存関係を求める計算が，CPN のループ構造により阻害されることはない．

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 5 (a)，(b)，(c)，(d) の実線部分になる．

（b）制御依存トランジションを新たなスライシング基準とし，データ依存関係，制御依存関係を求める．

メソッド呼び出しがある場合，関連を表す CPN（図 3 の右下）の CPN により，呼び出し側と呼び出される側が関連付けられる．この際，呼び出されるメソッドにデータ依存プレースがない場合，関連を表す CPN からそのメソッドを呼び出す部分を削除する．

（4）スライスを得る．

（a）スライスの構成要素を上記のデータ依存関係，

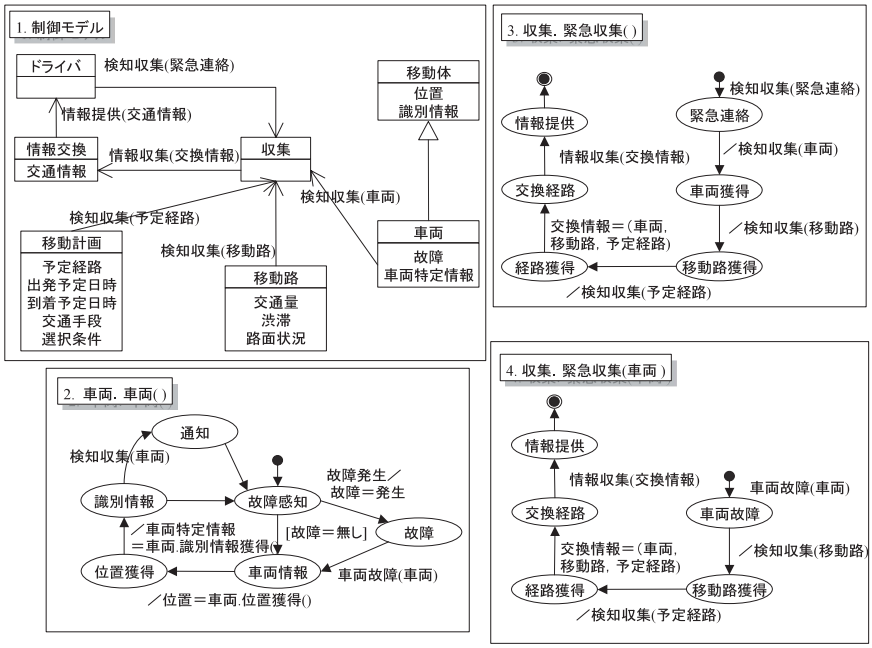

図 6 移動車両間の経路情報の交換の制御モデル
Fig. 6 A control model of exchange of the course information between vehicles.

制御依存関係を求める作業から集める．

(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，同じコントロールプレース（‘‘Control’’ カラーのプレース）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．

上記において，OO-CPN はメソッドごとに独立した複数の CPN で表されているのではなく，関連を表す CPN を通してメソッド間は連結され，1 つの CPN で表されている．上記のデータ依存関係を求める際に述べたが，メソッド間，オブジェクト間にまたがる変数は同じプレースに，引数は直接関連付けてられているため，オブジェクト間にまたがるスライスを得ることができる．

5. 適 用 例

本章では，OO-CPN スライシングを利用したテスト方法を例に適用する．以下，まずクラス図とステートチャートの例について説明する．次に，4 章の方法を例に適用しスライス得る様子について述べ，Design/CPN を利用し，OO-CPN スライスから到達グラフを得，テストケースを到達ラフから得る様子について述べる．

5.1 クラス図，ステートチャート，CPN

図 6 は ITS にかかわるシステムアーキテクチャ[3]の「移動車両間の経路情報の交換の制御モデル」をもとに変更を加えた例である．

この例において，ドライバに事故が起きると，“ドライバ” インスタンスから，“収集” インスタンスへ緊急連絡が “検知収集 (緊急連絡)” により通知され，“収集” インスタンスは，“車両” インスタンスから “車両故障 (車両)” メソッドにより故障情報を得る．図 6-2 “車両. 車両 ()” メソッドと図 6-3 に示した “収集. 緊急収集 ()“メソッド” を変換した CPN を各々**図 7**-1, 図 7-2 に示す．

“移動体” クラスと “車両” クラスは同じメソッド “識別情報獲得 ()” を持つとする．“移動体” インスタンスは “識別情報獲得 ()” により “識別情報” を得，“車両” インスタンスは “車両特定情報” を得る．“識別情報獲得 ()” メソッド呼び出しを表す CPN を**図 8** に示す．この図はコントロールトークンのクラス名により，適したメソッドを選択する．図 8 は，OO-CPN 生成時にクラス間の関連から生成する．図 8 のメソッド呼び出しを決定する CPN により，オブジェクト指向特有の実行時に決定される性質の表現を可能にする．図 6 で，“車両” クラスは “移動体” クラスを継承し，各々 “識別情報獲得 ()” メソッドを持つ．“車両” インスタンスのメソッドから呼ばれた場合，“車両” クラスで

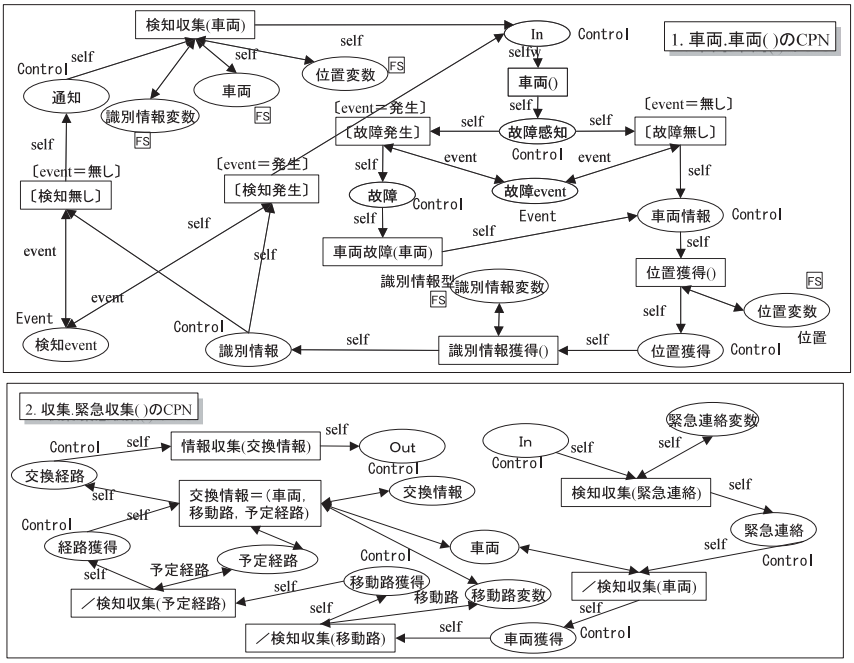

図 7 制御モデルの CPN
Fig. 7 CPN of control model.

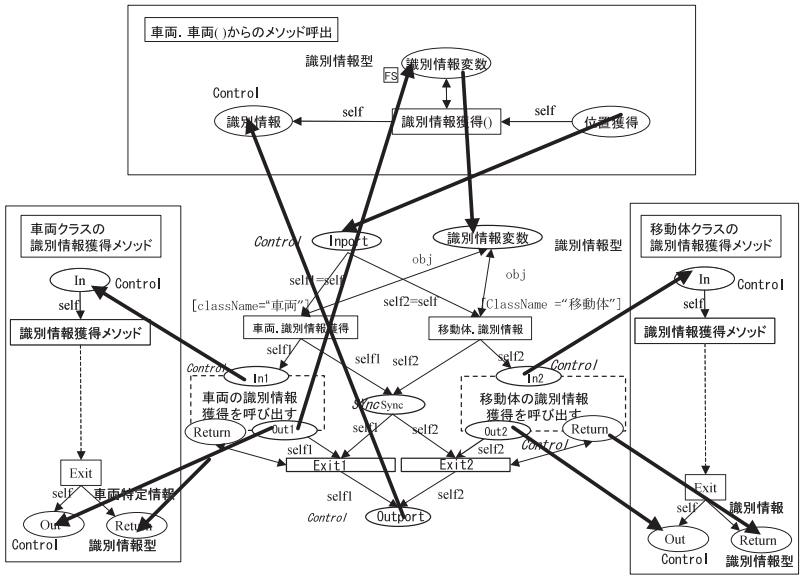

図 8 制御モデルのメソッド呼び出しを表す CPN
Fig. 8 Method call of control model.

定義された “識別情報獲得 ()” メソッドが選択される．この仕組みを図 8 の CPN で表す．

5.2 スライシング

4 章で提案したスライシング技術を図 7 の例に適用する．以下にその過程を示す．

(1) スライシング基準と開始を選択する．“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，すなわちデータ依存プレースを抽出する．この場合，データ依存プレースは，“交

換情報” プレースである．

(b) データ依存プレースに連結しているデータ依存トランジショントランジションを抽出する．データ依存トランジションは，“交換情報=(車両，移動路，予定経路)” トランジションである．

(c) データ依存トランジションにを基準トランジションとしてデータ依存関係を調べる．“交換情報=(車両，移動路，予定経路)” トランジションのデータ依存プレースは，“車両” プレース，“移動路変数” プレース，“予定経路” プレースである．“車両” プレースは “検知収集 (車両)” トランジション，“移動路変数” プレースは，“検知収集 (移動路)” トランジション，“予定経路” プレースは “検知収集 (予定経路)” トランジションと関係している．
さらに，“検知収集 (車両)” トランジションは “車両．車両 ()” の “検知収集 (車両)” トランジションと同期をとっているため，“車両．車両 ()” の ‘検知収集 (車両)” トランジションに関してもデータ依存関係を調べる（本論文では同期処理部分に関しては省略する）．

(3) 制御依存関係を求める．

(a) 制御依存プレース，制御依存トランジションを発見する．制御依存プレースは，“In” プレース，“故障感知” プレース，そして “識別情報” プレースである．制御依存トランジションは，‘条件分岐を表す ‘[検知無し]” トランジションと “[検知発生]” トランジションである．

(b) 制御依存トランジションを新たなスライシング基準とし，データ依存関係，制御依存関係を求める．

(4) スライスを得る．

(a) スライスの構成要素を上記のデータ依存関係，制御依存関係を求める作業から集める．

(b) 切り離されたスライス構成要素を連結する．不連続な構成要素は，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．たとえば，スライス構成要素は，“検知収集 (緊急連絡)” トランジションとその周囲にある “In” プレース，“緊急連絡変数” プレース，“緊急連絡” プレース，そしてこれらを結んだ部分 CPN がスライスの一構成要素である．この構成要素に続く要素は，“検知収集 (車両)” トランジション，“緊急連絡” プレース，“車両” プレース，“車両獲得” プレースの部分 CPN である．“緊急連絡” プレースは，“検知収集 (緊急連絡)” トランジションを中心とした部分 CPN と，“/検知収集 (車両)” トランジションを中心とした部分 CPN 共通のコントロールプレースであるため，この 2 つの部分 CPN は “緊急連絡” プレースによって連結される．もし，構成要素と構成要素が連続していない場合，すなわち格構成要素の入力と出力が共通のコントロールプレースでない場合は，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．

5.3 到達グラフとテスト

2 章で述べたように，テストケースは到達グラフから生成する．図 6 の到達グラフの一部を**図 10**-1 に示す．この到達グラフは，**図 9** の OO-CPN スライスを実行した結果から得たグラフの抜粋である．テストケースは到達グラフから得たトランジション列である．テストケースの例を図 10-2 に示す．

以上から，提案した方法で得たスライスから到達グラフを生成し，テストケースを生成できることを示した．以前の方法では[1]，スライシングを行わない元の到達グラフからテストケースを生成していた．機能別にテストしたい場合でも OO-CPN を網羅的に実行しなければならず，さらに実行に必要な情報が盛り込まれている必要があった．本方法では，一機能のみを記述した段階でスライシングを行いテストケースを得ることが可能であるため，従来よりも開発の早い段階で誤りを検出することが可能となる．また，複数のオブジェクト間にまたがった機能についても，機能に着目したテストが可能となった．

6. 関 連 研 究

本章では，既存の技術と提案した方法とをペトリネット，状態機械，プログラミング言語の 3 つの視点から比較する．

6.1 ペトリネット

ペトリネットのスライシング技術は，半順序展開技術[10),17)]と関連がある．これらの技術では，可達解析を行うことでペトリネットを循環のない木構造へ展開する．可達解析は，ペトリネットの実行結果である到達グラフを用いて行われる．これらの技術を用いれば，展開したグラフから機能に着目した CPN のサブグラフを得ることが可能である．このプロセスは，実行結果からスライシングを得るための木を得るという点で，プログラミング言語のダイナミックスライシン

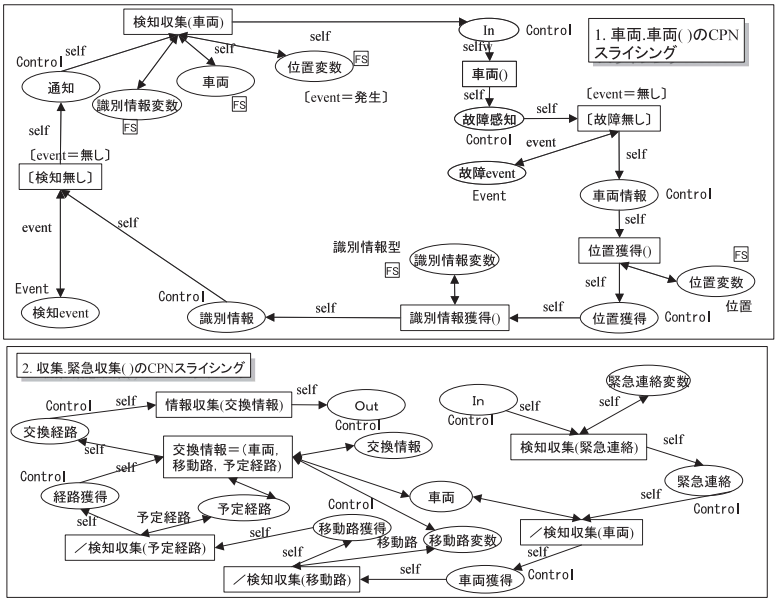

図 9 制御モデルのスライス
Fig. 9 Slice of control model.

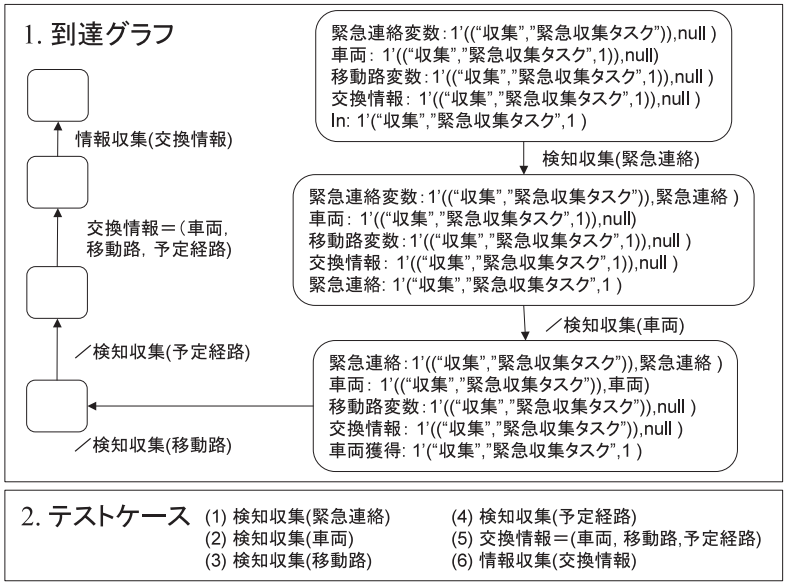

図 10 到達グラフとテストケース
Fig. 10 Occurence graph and test case.

グ[7]に似ている．しかし実行による方法では，実行に必要な仕様の記述が網羅されていなければならない．したがって，一機能は記述したが，他の箇所は記述途中である場合にテストを行うことができない．また，1 章で問題提起したように，記述したペトリネットを網羅するように到達グラフを得るには，膨大なコストが必要となる．

本論文で提案した方法では，OO-CPN 全体に対して到達グラフを得る必要はないため，到達グラフを得るためのコストは削減され，スライシング基準に影響する機能さえ記述してあれば，テストすることも可能である．

6.2 状態機械

状態機械のスライスを得るためには，データ依存と

制御依存を抽出し，SDG[12]や CDG[19]を構築しなければならない．OO-CPN では，コントロールトークンとデータトークンの 2 種類にトークンを分類していることから，データ依存，制御依存を求めることができる．コントロールトークンの概念により，データ依存関係を求めることが可能であるため，直接，状態機械からスライシングを求めることは困難である．さらに，オブジェクト指向ソフトウェア特有の実行時に決定される性質（動的性質）が記述されている場合でも，動作するモデルがなければオブジェクト間にまたがるテストを行うことはできない．OO-CPN へ変換することにより，動作するモデルを得ることができる．

状態機械のいくつかは，動的性質が記述されていても動作させることが可能である．これらの多くは，実行時にグラフ書き換え技術を適用し，動的に状態図を書き換える[20]．したがってこれらの状態図を静的に解析することはできない．

6.3 プログラミング言語

プログラミング言語のためのスライシング技術は，手続き呼び出し，ポインタ，オブジェクト指向などプログラム言語の様々な性質に対応してきた[7),12),16),18),19),24)]．したがって，記述したステートチャートをプログラムへ変換し，スライスするという方法も考えられる．しかし，プログラムではデッドロック，衝突，公平性などの並行に関する誤りを発見することはできない．プログラムへ対応付けるためには，仕様であるステートチャートを必要以上に詳細に書かなければならないという問題がある．そのため仕様のまま動作可能であり，並行性に関する誤りを発見できるようなモデルも必要である．すでに述べたように，OO-CPN は動作可能であり，並行性に関する誤りを発見できる．

7. おわりに

本論文の目的は，開発のなるべく早い時期にオブジェクト指向ソフトウェア特有の誤りを効率的に発見することにあった．この目的を達成するために，ステートチャートとクラス図をカラーペトリネットへ変換して得られる OO-CPN のスタティックスライシング方法を提案した．

1 つのスライスを 1 機能と見なした場合，スライスで得た CPN の部分ネットも 1 機能である．この部分ネットを解析，または部分ネットからテストケースを生成することにより，機能に着目した解析とテストを可能にしたといえる．

本方法は CPN を実行せずにスライスを得るため，開発途中であっても記述が完了した機能に関してテストを行うことが可能である．これにより，従来よりも開発早期の段階に誤りを発見する目的を達成した．さらにオブジェクト指向特有の動的な性質による状態爆発問題を機能ごとにテスト可能にすることで解決した．これにより従来よりも効率的に誤りを発見する目的を達成した．

OO-CPN スライシングは，プログラムスライシングと同様，仕様のデバッグ，変更支援，そしてリファクタリングへの応用が期待できる．今後は，OO-CPN スライシングを利用し，これらの技術に取り組んでいく予定である．さらに，プログラムスライシングで利用している system dependence graph（SDG）に相当するグラフを OO-CPN スライシングにも適用し，より効率的にスライスを得る方法を確立したい．

参考文献

1) 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 DI（情報・システム I—コンピュータ），Vol.J82-D-I, No.3, pp.1–19 (1999).
2) 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（基礎・境界），Vol.J80-A, No.7, pp.1073–1086 (1997).
3) 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
4) 下村隆夫：プログラムスライシング，共立出版 (1995).
5) Watanabe, H., Tokuoka, H., Wu, W. and Saeki, M.: A Technique for Analysing and Testing Object-oriented Software Using Coloured Petri Nets, *Asia Pacific Software Engineering Conference '98*, pp.182–190 (Dec. 1998).
6) Watanabe, H. and Kudoh, T.: Test Suite Generation Methods for Concurrent Systems Based on Coloured Petri Nets, *Proc. Asia Pacific Software Engineering Conference '95*, pp.242–251 (Dec. 1995).
7) Agrawal, H. and Horgan, J.R.: Dynamic Program Slicing, *ACM SIGPLAN'90 Conf. Programming Language Design and Implementation*, pp.246–256, ACM Press (1990).
8) Bennett, K.H. and Rajlich, V.T.: Software Maintenance and Evolution: a Roadmap, *The Future of Software Engineering*, pp.75–87, IEEE (July 2000).
9) Booch, G., Rumbaugh, J. and Jacobson, I.: *The Unified Modeling Language User Guide*,

Addison Wesley (1999).
10) Esparza, J., Römer, S., Schröter, C., Schwoon, S. and Wallner, F.: Partial-order verification techniques. http://wwwbrauer.informatik.tu-muenchen.de/gruppen/theorie/pom/#biblio.
11) Harman, M. and Danicic, S.: Using program slicing to simplify testing, Software Testing, *Verification and Reliability*, 5:143-162 (Sept. 1995).
12) Horwitz, S., Reps, T. and Binkley, D.: Interprocedual Slicing Using Dependence Graphs, *ACM Trans. Prog. Lang. Syst.*, Vol.12, No.1, pp.26–60 (1990).
13) Jensen, K.: *COLOURED PETRI NETS Basic Concepts, Analysis Methods and Practical Use*, Volume 1, Springer-Verlag (1992).
14) Jensen, K.: *COLOURED PETRI NETS Basic Concepts, Analysis Methods and Practical Use*, Volume 2, Springer-Verlag (1994).
15) Jensen, K.: *COLOURED PETRI NETS Basic Concepts, Analysis Methods and Practical Use*, Volume 3, Springer-Verlag (1997).
16) Jiang, J., Zhou, X. and Robson, D.J.: Program Slicing For C — The problems In Implementation, *IEEE International Conference on Software Maintenance*, pp.182–190 (1991).
17) Kondratyev, A., Kishinevsky, M., Taubin, A. and Ten, S.: *A Structual Approach for the Analysis of Petri Nets by Reduced Unfoldings*, pp.356–365.
18) Korel, B. and Laski, J.: Dynamic Program Slicing, *Software Merging and Slicing*, pp.54–62, IEEE (1995).
19) Liang, D. and Harrold, M.J.: Slicing objects using the System Dependence Graph, *IEEE International Conference on Software Maintenance*, pp.358–367 (Nov. 1998).
20) Maggiolo-Schettini, A. and Peron, A.: A Graph Rewriting Framework for Statecharts, *Proc. 5th International Workshop on Graph Grammars and their Application to Computer Science*, Springer LNCS, Vol.1073, pp.106–121 (1996).
21) Maruyama, K. and Shima, K.: An Automatic Class Generation Mechanism by Using Method Integration, *IEEE Trans. Soft. Eng.*, Vol.26, No.5, pp.425–440 (2000).
22) Peterson, J.L.: *Petri Net Theory and the Modeling of Systems*, Englewood Cliffs, New Jersey, Prentice Hall Inc. (1981).
23) Pressman, R.S.: *Software Engineering—A Practitioner's Approach*, 3rd Edition, McGraw-Hill (1992).
24) Weiser, M.: Program Slicing, *IEEE Trans. Soft. Eng.*, Vol.SE-10, No.4, pp.352–357 (1984).

(平成 14 年 10 月 3 日受付)
(平成 15 年 2 月 4 日採録)

渡辺 晴美（正会員）

1990 年東京工科大学情報工学科卒業．1990 年から 1993 年まで日本電気マイコンテクノロジー（株）勤務．1995 年東京工科大学大学院工学研究科システム電子工学専攻修士課程修了．1998 年東京工業大学情報理工学研究科計算工学専攻修了．博士（工学）．1998 年から 2000 年まで北陸先端科学技術大学院大学リサーチアソシエイト．2000 年から立命館大学理工学部情報学科助手．ソフトウェアのテスト，組込みシステム開発方法論の研究に従事．日本ソフトウェア科学会会員．