自己適応計算の CSP を用いた検証方法の提案

2013SE214 田中文菜　2014SE088 澤木結香
指導教員：張漢明

1　はじめに

ソフトウェアの大規模化に伴い，自己適応を前提とした構成が要求される [5] とともに，開発において手戻りの多さが課題となっている．本研究室では自己適応構成法として PBR パターン [4] を提案している．PBR パターンは動的再構成を目的とした自己適応のためのアーキテクチャパターンである．一方で，再構成前にはデッドロックが起こらないソフトウェアが，再構成後にはデッドロックを呼び込む可能性がある．これは動的にしか発見することができず，デッドロックが起きた場合，手戻りになる．

本研究の目的は，PBR パターンを適用したソフトウェアの振舞いを，形式手法で記述し，モデル検査器で実行前検査することで，手戻りを減少させることである．組込みシステムではハードウェアが並行に動作し，イベントの授受によって協調することから，その振舞いを CSP(Communicating Sequential Process)[1, 7] で記述し検証する．モデル検査には FDR(Failures Divergence Refinement）[2] を利用する．モデル検査を使用し，デッドロックが起きた場合，トレースが反例として表示されるが，その原因となる誤りを特定する作業は困難である．そこで，記述の誤りをパターンとして提示することにより，検証作業の軽減を目指す．

本研究では紙幣を搬送・管理するシステム (以下， 紙幣搬送システム) を事例として CSP を用いた検証方法を考察する．

2　関連技術

2.1　PBR パターン

PBR(Policy-Based Reconfiguration) パターンとは，本研究室が提案している動的再構成を目的とした自己適応のためのアーキテクチャパターンである．PBR パターンの静的構造と動的振舞いを図 1(a)，(b) に示す．

ポリシーと再構成の仕組みをそれぞれコンポーネント化し，動的再構成を実現する．コンテキストの変化を含むポリシー (Policy)，ポリシーに応じて変化する再構成後のオブジェクト群を代表にするアスペクトオブジェクト (AspectObject)，再構成の仕組みとして，このアスペクトオブジェクトのインスタンス生成を行うファクトリ (Factory) から構成した．ポリシーがその記述に従ってファクトリを起動する関係となり，ファクトリはアスペクトオブジェクトを生成する関係となる．オブジェクト間のメッセージを横取りし，ポリシーでファクトリにアスペクトオブジェクトのインスタンスを生成させ，このインスタンスにメッセージを送る．複数のコンポーネントに分割したことで，それぞれのコンポーネントの役割の汎用性が高くなる．

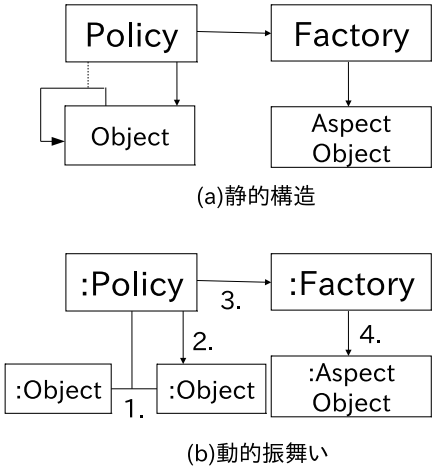

図 1　PBR パターン

2.2　CSP

CSP(Communicating Sequential Process) は並行システムの振舞いを形式的に記述し，解析するための理論であるプロセス代数の一つであり，イベントの集合としてシステムをモデル化する．イベントは瞬間的で基本的な動作であり，プロセスはイベント列として記述する．CSP では複数のイベントの実行順序を並行合成することで並行システムを表すことができる．補助的に図を用いて構図や動作を表現することで，より理解しやすくなる．本研究では，提唱者である Hoare の記法を用いる．

2.3　FDR

FDR(Failures Divergence Refinement) は CSP でモデル化されたプロセスの等価関係や詳細化関係を自動的に判定するためのツールである．

FDR ではトレース詳細化，失敗詳細化，失敗発散詳細化が検証できる．プロセス P を詳細化したプロセス Q を P $\sqsubseteq$ Q と示すことができ，以下に 3 つの詳細化関係の説明をする．

- トレース詳細化 (P $\sqsubseteq_T$ Q)
  Q は P のトレース詳細化関係である．プロセス Q は元のプロセス P よりも実行可能なトレース (イベントの実行列) が少ないことを表す．詳細化の過程で安全性を検証するために使うことが可能．
- 失敗詳細化 (P $\sqsubseteq_F$ Q)
  Q は P の失敗詳細化関係である．トレースの条件に加え，各状態の失敗集合 (その状態で実行できないイ

ベントの集合) が増加しないことを表す．詳細化の過程でデッドロックが入り込まないことやイベントの実行可能性を検証するために使うことが可能．

- 失敗発散詳細化 (P $\sqsubseteq_{FD}$ Q)
  Q は P の失敗発散詳細化関係である．トレースと失敗の条件に加え，発散トレース集合 (内部の無限ループに至るトレースの集合) が増加しないことを表す．デッドロックフリーとライブロックフリーを保証する詳細化に使うことが可能．

3 デッドロックパターンの定義

3.1 デッドロック

デッドロックとは，2 つ以上のプロセスが互いの処理終了を待ち，結果としてどの処理も先に進めなくなってしまうことである．デッドロックになったプロセスは外部処理からデッドロックを解除されない限り，永久的に待機を続ける．

3.2 デッドロックパターン

我々は，さまざまな例題を試し，デッドロックが起こる場合のデータを集めた．多くの場合，「交差パターン」「つまりパターン」の 2 つのパターンが多いことが分かった．よって，「交差パターン」「つまりパターン」をデッドロックが起こる一般的なパターンとし，デッドロックパターンと定義した．それぞれの概要を以下に示す．

交差パターンは，並行合成する 2 つ以上のプロセスの中から任意の 2 つのプロセスを取り出したとき，任意の 2 つのイベントが交差していた場合デッドロックが起きることである．CSP 記述を以下に示す．A はイベント集合，e.m，e.n は任意のイベントである．

パターン 1

```
PA = [] x : A @ x -> PA [] SKIP
P = PA; e.m -> PA; e.n -> P
Q = PA; e.n -> PA; e.m -> Q
R = P [ Palpha || Qalpha ] Q
```

つまりパターンは，並行合成する 2 つ以上のプロセスの中の 1 つのプロセスが SKIP によって正常終了してしまい，次の工程へ進めない場合デッドロックが起きることである．CSP 記述を以下に示す．A はイベント集合，e.m，e.n は任意のイベントである．

パターン 2

```
PA = [] x : A @ x -> PA [] SKIP
P = PA; e.m -> PA; e.n -> P
Q = PA; e.m -> SKIP
R = P [ Palpha || Qalpha ] Q
```

以上の 2 パターンをデッドロックパターンとして定義する．システムの振舞いを CSP を用いて記述した場合，このパターンと比較し実行前に訂正を行う．

4 紙幣搬送システムへの適用

4.1 紙幣搬送システム

紙幣搬送システムの仕様は以下のとおりである．

- 搬送路
  搬送対象の経路
- 金庫 (1000 円用，5000 円用)
- モータ (正モータ，負モータ)
  搬送対象を搬送する
- 搬入口センサ
  搬送対象を検知する
- 判別センサ
  搬送対象である紙幣の種類を判別する

搬入口センサが搬送対象を検知した時，モータは正方向に回転する．判別センサで紙幣の種類を判別し，1000 円札を検知した時，モータは正方向に回転し 1000 円金庫に紙幣が格納される．5000 円を検知した時，モータは負方向に回転し 5000 円金庫に紙幣が格納される．図 2 に設計を示し，図 3 にはハードウェアの順序関係を整理し，状態遷移図として設計したものを示す．図 4 には，1000 円が投入された際のシーケンス図を示す．

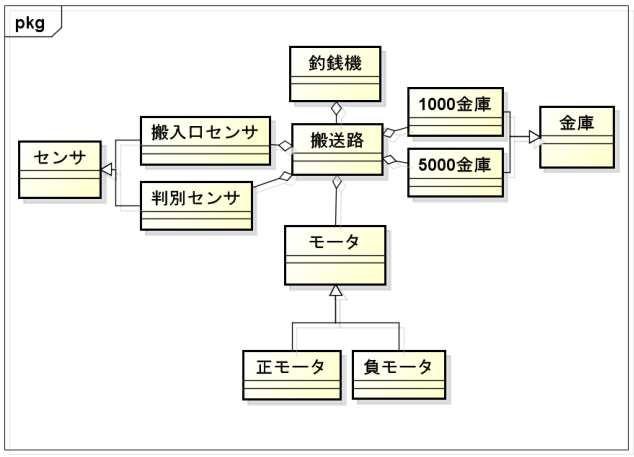

図 2 設計

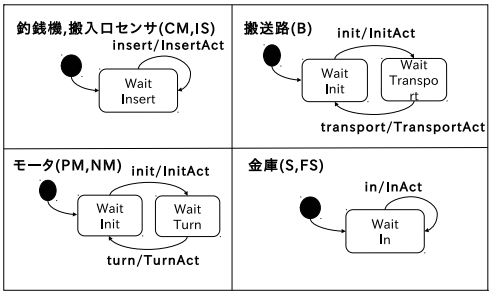

図 3 HW の振舞い

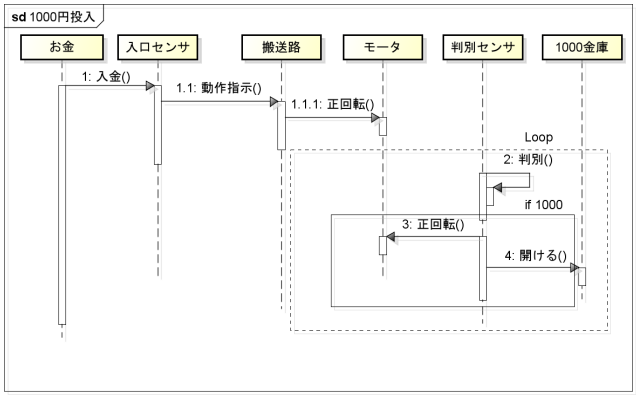

図 4　1000 円投入時の振舞い

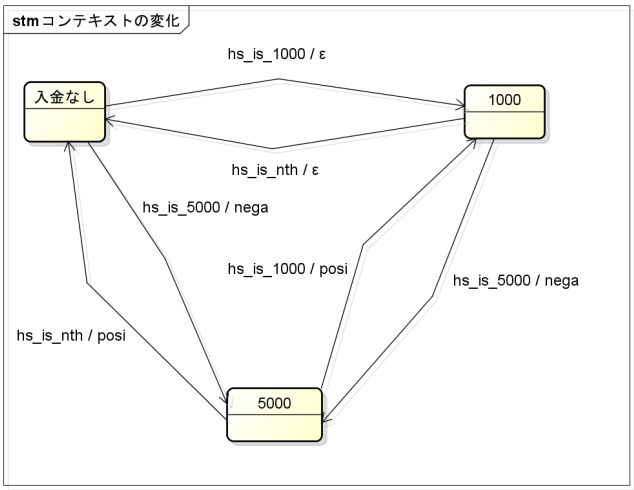

図 6　コンテキストの変化

4.2　PBR パターンの適用

紙幣搬送システムに PBR パターンを適用する．PBR パターンを適用し，コンテキストに関連する記述を分離する．コンテキストとこれに応じた振舞いを活性化する手続きを分離し，独立に変更できるようにする．この動的振舞いを図 5 に示す．

ポリシーをコンテキスト，ファクトリを振舞い活性化手続きとした．搬入口センサと判別センサのメッセージ通信を横取りし，コンテキストの状態を変化させる．振舞い活性化手続きは，コンテキストの振舞いが変化した際に，モータに対してこれに応じた振舞いを活性化させる．

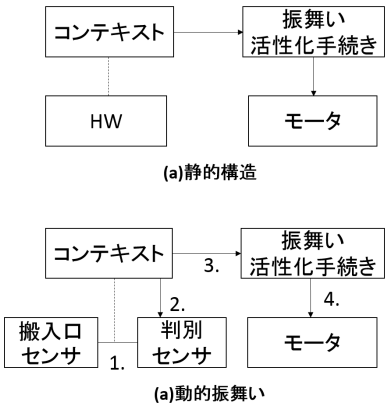

図 5　PBR パターン

挿入紙幣種類の変化をコンテキストと定義する．コンテキストの変化を図 6 に状態遷移図で示す．

ファクトリの振舞いはコンテキストとアスペクトオブジェクトの関係で記述する．表 1 にその関係を示す．入金が確認できない場合は C1’，1000 円が投入された場合は C2’ が実行されモータは正方向回転する．5000 円が投入された場合は C3’ が実行されモータは負方向回転する．

4.3　CSP による振舞いのモデル化

前述の状態遷移図とシーケンス図に基づいて，振舞いを CSP を用いてモデル化する．

表 1　ファクトリの振舞いの関係

コンテキスト	アスペクトオブジェクト
nth	C1’
1000	C2’
5000	C3’

HW の振舞いを設計に基づいて記述する．コンテキストは紙幣の種類に応じて変化する．Context1 は入金なし，Context2 は 1000 円入金，Context3 は 5000 円入金を表す．ファクトリではコンテキストの変化に応じてインスタンスを再生成する．入金がない場合 C1 を選択，1000 円が入金された場合 C2 を選択，5000 円が入金された場合は C3 を選択する．ハードウェア間の協調は同期イベント (e1，e2，e3，e4，e5) により実現している．e2 により，搬送路が搬送対象を搬送し，モータが正回転する．e4 も同様である．e3 により，モータによって搬送された搬送対象が 1000 円金庫に格納することを示している．e5 も同様である．結果として，紙幣搬送システムにおける振舞いを形式的に記述することができた．

5　考察

5.1　誤りの記述例

紙幣搬送システムの例から，我々は PBR パターンでデッドロックが起こる可能性が高い記述は，ハードウェア間の協調に関する記述であると考えた．これは，PBR パターンの並行合成に，名前変更を用い，それによってデッドロックの可能性を高めるからである．紙幣搬送システムのハードウェア間の協調に関する記述において考えられるすべての記述ミスを取り上げ，中でも設計を誤った場合にデッドロックが起きる可能性がある記述を示す．

(1)B(搬送路) の記述を，再帰ではなく正常終了で記述した

場合

```
B = init -> transport -> SKIP
```

(2)C2' 内で同期イベントの名前変更を間違えた場合
複数のハードウェアを扱う組込みシステムでは，同期イベントの名前を変更する必要がある．その際，誤った記述になる可能性がある．

```
BPM = B[[B.transport <- e2]]
        [| e2 |]PM[[PM.turn <- e2]]
C2' = BPM[[PM.turn <- e3]]
        [| e3 |]S[[S.in < -e3]]
```

(3) 同期イベントを同じ turn とした場合

```
M = init -> turn -> get -> turn -> M
HS = init -> dis -> turn -> open -> HS
```

5.2 デッドロックパターンとの比較

定義したデッドロックパターンと比較し，誤りを訂正する．

(1) はパターン 2 に当てはまる．プロセス B が正常終了するので，プロセス B 内のイベントと同期しているプロセスは同期イベントから先に動くことができなくなる．

```
(訂正)
B = init -> transport -> B
```

(2) はパターン 1 に当てはまる．同期イベントの名前変更を間違えたことによって，2 つの同期イベントが交差になってしまい，デッドロックが発生する．

```
(訂正)
BPM = B[[B.transport <- e2]]
        [| e2 |]PM[[PM.initM <- e2]]
C2' = BPM[[PM.turn <- e3]]
        [| e3 |]S[[S.in < -e3]]
```

(3) はパターン 1 に当てはまる．モータを一つのプロセスで表した場合，入金後の turn と搬送対象を搬送する turn を同じ同期イベントとしてしまい，判別センサとの同期の際に交差が起こり，デッドロックが起きる．

```
(訂正)
M = init -> turn1 -> get -> turn2 -> M
HS = init -> dis -> turn2 -> open -> HS
```

デッドロックパターンとの比較により，デッドロックパターンは誤りを訂正する際，有用であることが分かった．同時に，PBR パターンを適用した紙幣搬送システムにおいてもデッドロックパターンで事前に確認することによって手戻りの軽減ができることがわかった．

6 おわりに

本研究では，PBR パターンを適用したソフトウェアの振舞いを CSP で記述し，モデル検査器 FDR で実行前検査することで手戻りを減少させることを目的とした．

組込みシステムである紙幣搬送システムを例に，PBR パターンを適用した．振舞いを CSP で記述し，モデル検査器 FDR を用いて実行前検査を行った．並行システムを状態遷移機械の集合と捉え，典型的な誤りをデッドロックパターンとして定義した．簡単な例から，PBR パターンを適用したソフトウェアでは，名前変更を用いるハードウェア間の協調記述の部分においてデッドロックが起こる可能性が高いと考え，デッドロックパターンと比較し，検証前に誤りとなり得る箇所を指摘することで，検証作業を削減することができた．

今後の課題として，より複雑なシステムに PBR パターンを適用し，その仕様の記述と検証を行い，複雑なシステムでも有効であることを確認し，デッドロックパターンをより有用なパターンにすることが挙げられる．

7 参考文献

参考文献

[1] C. A. R. Hoare, *Communicating Sequential Processes,* Prentice-Hall, 1985.

[2] 磯部祥尚，並行システムの検証と実装，近代科学社，2012.

[3] 鵜林尚靖，玉井哲雄，“アスペクト指向プログラミングへのモデル検査手法の適用，”情報処理学会論文誌，vol.43，no.6，pp.1598-1609，2002.

[4] 江坂篤侍，野呂昌満，沢田篤史ほか，“コンテキストアウェアネスを考慮した IoT システムのためのアスペクト指向アーキテクチャの設計，”ソフトウェア工学の基礎ワークショップ論文集，vol.24，pp.3-12，2017.

[5] 鄭顕志，清水遼，高橋竜一，石川冬樹，“自己適応ソフトウェアのための自己適応性設計に関する研究動向，”コンピュータソフトウェア，vol.31，no.1，pp.49-59，2014.

[6] 中島震，ソフトウェア工学の道具としての形式手法，National Institute of Informatics，2007.

[7] 山崎利治，虫の付かないフォーマルな開発手法，Design Wave Magazine，2002.

[8] 吉岡信和，田辺良則，田原康之，長谷川哲夫，磯部祥尚，“モデル検査による設計検証，”コンピュータソフトウェア，vol.31，no.4，pp.40-65，2014.