アスペクト指向プログラミングの CSP による検証方法の提案

2014SE093 杉山拓希　2014SE108 山田祥太
2014SE115 横井広大
指導教員：張漢明

1　はじめに

ソフトウェアの大規模化，複雑化に伴いソフトウェアの保守性，拡張性が重要視されてきている．これらを高める方法として，ソフトウェアのモジュール化技法の一つとしてアスペクト指向が研究されている．アスペクト指向とは，横断的関心事をアスペクトとして，アスペクト記述言語を用いて分離しプログラムに柔軟性を持たせることである．アスペクト指向を用いることで将来の変化や進化に強いソフトウェアが開発できる．

アスペクト指向開発の問題点として，各モジュールの合成後の動作が把握しづらいことが挙げられる．個々のモジュールに欠陥がなくても，合成後のプログラムに欠陥が発生する場合がある．これらの欠陥が実行時に発生した場合，問題点の特定が困難であり，手戻りが発生し必要以上にコストがかかる．加えて，作業効率の低下にも繋がる．設計段階で実行前検査を行うことで設計の不具合を発見し早期に修正し，手戻りを少なくする．

本研究の目的はアスペクト指向プログラムの実行前検査をすることで早期に問題を発見することである．プログラムの構文の正しさやデータ型の正しさを検証することではなく，プログラムの振舞いが正しいことを実行前に検証する．加えて，デッドロックパターンの一般形を明示し分類することで，検証段階での発見が容易になると考えた．

本研究では，アスペクト指向を用いた組込みシステムを実行前検査するために，設計段階で作成される UML 図から CSP 記述を行い検証ツール FDR を用いて検証する．また検査から検出されるデッドロックについて，パターンを分類し，一般化することでデッドロックの発生頻度の軽減を行う．

2　背景技術

2.1　実行前検査

実行前検査では，デッドロックやスタベーションなどの欠陥を設計段階で検出する．設計書からモデル検査言語を用い状態遷移モデルの動作パターンを網羅的に探索することで実行前に検査する．実行前検査を行うことで，早期に問題発見することができ，手戻りを減らすことができる．検査手順の概要を図 1 に示す．

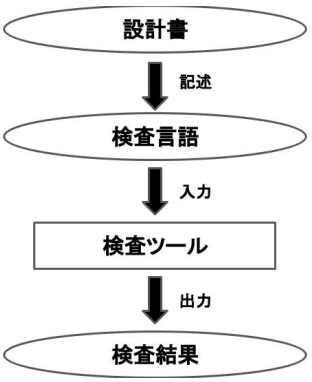

図 1　実行前検査手順

2.2　CSP

CSP(Communication Sequential Processes)[1] では，並行システムをモデル化し，各プロセスをイベントの集合として表現する．本研究ではアスペクト指向を用いた組込みシステムが並行動作していると捉えモデル化する．

2.3　FDR

FDR(Failure-Divergence Refinement)[2] では，CSP でモデル化したものを検査し，ソフトウェアにおける欠陥を検出する．検出できる欠陥としては以下があげられる．

- デッドロック
- スタベーション
- 非決定性

3　アスペクト指向ソフトウェア

3.1　アスペクト指向の基本概念

アスペクト指向プログラミング (Aspect Oriented Programming: AOP)[5] は関心事の分離を実現するために，特定の処理を実行させる条件を，クラスとは独立してアスペクトに記述する．アスペクトを導入することで，オブジェクト指向プログラミングにおいてクラス間にまたがる関心事を，クラスから分離してモジュール内に閉じ込めることが可能である．その反面，アスペクトの定義を見なければ，クラスの振舞いを正確に把握することができないという性質により，互いに影響を及ぼしあう複数のアスペクトを，開発者が無意識に記述してしまう可能性がある．

3.2　アスペクトの干渉

アドバイスの実行順序に起因する副作用として，衝突 (conflict) と干渉 (interference) がある．衝突とは，同一ジョインポイントに対して複数のアドバイスが記述されていることである．干渉とは，衝突したアドバイスの実行順序の違いによって，実行結果に違いが生じることである．図 2 は，アスペクト干渉の例である．$x = 1$ という値を持つオブジェクトに対して $x+ = 2$ を行なう AspectA と，

$x* = 2$ を行なう．AspectB が存在する．AspectA が先に織込まれる場合，実行結果は $x = 6$ となる．AspectB が先に織込まれる場合，実行結果は $x = 4$ となり，織込み順序によって実行結果が異なることが確認できる．

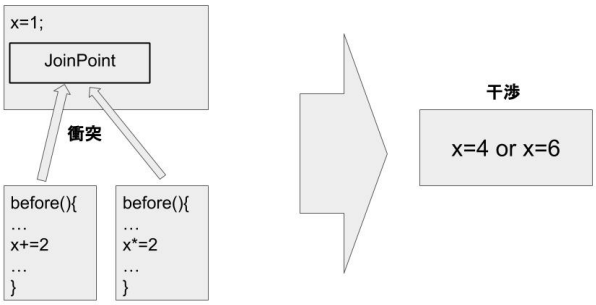

図 2　アスペクト干渉の例

アスペクトの干渉を実行前に検出する方法として，既存研究から以下の 2 つが挙げられる．

- データフロー解析
- プログラムスライス

データフロー解析 [7] とはプログラム内の様々な位置で，取りうる値の集合に関する情報を収集する技術で，データフロー解析を用いてアスペクトの干渉可能性箇所の検出を行い検査することで，アスペクトの干渉が起こった場合の原因箇所の特定作業時間の軽減ができる．プログラムスライス [3] とはプログラムに含まれる手続き呼び出し関係や変数の参照・代入関係などの依存関係を解析し，プログラム内の注目すべき部分だけを抽出，提示する技術で，アスペクトによって動作不良に陥った場合，スライスを用い対象のプログラムを開発者に提示し原因の特定作業の軽減をすることができる．

3.3　先行研究

先行研究では，ステートマシン図とシーケンス図の CSP 記述との対応が提案されている．図 3 に表現方法の例を示す．

4　アスペクト指向ソフトウェアの CSP 記述と検証

アスペクト指向で作成されるソフトウェアは，オブジェクト指向で考えられるクラスとアスペクト指向で考えられるアスペクトの集合である．アスペクト指向を用いて横断的関心事の分離によりモジュール間の関係が変わっても，動的に検証することができれば，構造の正しさも証明できると考えた．本研究では，動的な問題であるデッドロックを扱うので，アスペクト指向プログラムを CSP で動的に表現し，それらを組み合わせることでソフトウェアを検証できると考えた．

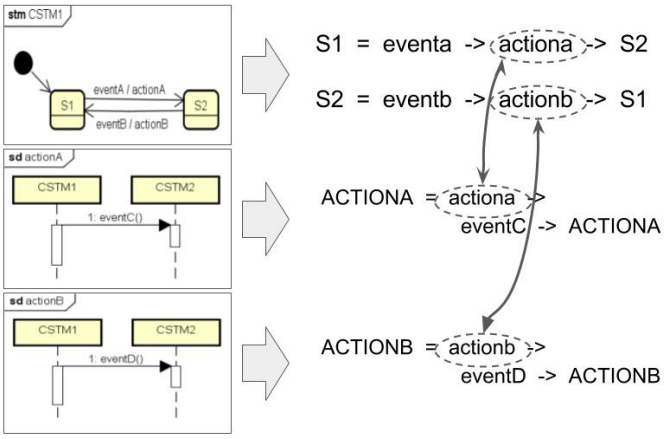

図 3　CSP による UML 図の表現方法の例

4.1　アスペクトのモデル化

アスペクト指向による設計に対して CSP で表現する際，ポイントカットの位置を適切に記述する必要がある．もし間違ったモデル化をしてしまった場合，アドバイスが違うタイミングで実行されるなどの不具合が考えられる．アスペクト間記述のシーケンス図表現方法を図 4 に示す [6].

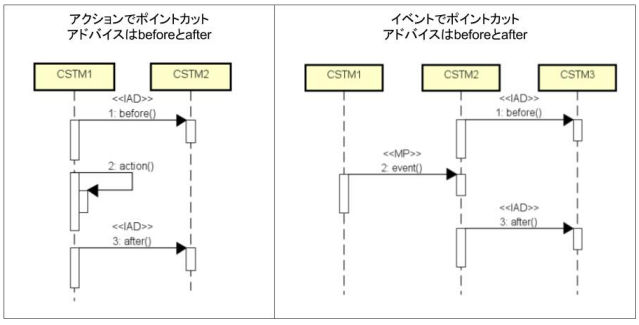

図 4　アスペクト間記述のシーケンス図表現

4.2　UML 図から CSP 記述

状態遷移機械の振舞いを示すステートマシン図とアクションを示すシーケンス図を用いて CSP 記述を行う．

4.2.1　状態遷移機械の表現方法

状態遷移機械をステートマシン図を基に CSP 記述を行う．状態をプロセスで表現し，受信イベントとアクションをイベントで表現する．また遷移時のイベントを開始イベントと終了イベントの 2 つで記述する．終了イベントを記述することで，次の状態へ遷移することを正確に表現できる．また一つの状態からの状態遷移が複数ある場合は，選択を用いて表現する．状態遷移機械を表現する CSP 記述の記述法を図 5 に示す．

状態遷移機械の表現方法

遷移前の状態 = 受信イベント -> アクション -> 遷移後の状態
(遷移先が一つの場合)

遷移前の状態 = 受信イベント -> アクション -> 遷移後の状態1
[]受信イベント -> アクション -> 遷移後の状態2
(遷移先が複数の場合)

図 5 状態遷移機械の振舞い表現の記述方法

4.2.2 アクションの表現方法

アクションをシーケンス図とオブジェクト図を用いて CSP 記述を行う．アクションは同じ状態遷移機械から作成されたインスタンスであっても，イベントの通知先のインスタンスが異なるので，インスタンスごとで CSP 記述を行う．複数のインスタンスが存在することで，CSP 記述が複雑になり，記述ミスや重複する可能性がある．この問題を解決するためにオブジェクト図からインスタンス間の関係を取得することで対処できると考えた．アスペクト指向で考える際にはイベントの通知方法は MP によるイベント通知とアスペクト間記述によるイベント通知の二種類がある．アクションを表すシーケンス図においてイベント通知は，ステレオタイプを記述することで区別される．アスペクト間記述によるイベント通知は，設計時に順番を定義しないので，順番は非決定的である．デッドロックなどの不具合を検査するには，可能性のあるすべての順番を記述する必要がある．全てのイベント通知を表現するには，CSP 記述では選択を用いて記述する．アクションの CSP での表現方法を図 6 に示す．

アクションの表現方法

アクション名 = アクション開始イベント -> 送信イベントA
-> 送信イベントB -> アクション名
[] アクション開始イベント -> 送信イベントB
-> 送信イベントA -> アクション名
(アスペクト間記述によるイベントが2つの場合)

図 6 アクション表現の記述方法

4.2.3 プロセスのつながりの表現方法

シーケンス図をもとに並行合成，またはアルファベット付き並行合成を用いてソフトウェア全体を記述する．対象のプロセスの同期集合に含まれるイベントで同期しながら並行動作するプロセスを表すことで，全体の記述ができる．同期集合に含まれないイベントは各々独立に実行可能であるが，同期集合に含まれるイベントが実行されるまで次のイベントは実行できないので，逐次プロセスの動作と同じになる．また 3 つ以上のプロセスを並行合成する場合は，複製型アルファベット付並行合成 [4] を用いることで表現することができる．ソフトウェア全体の CSP での表現方法を図 7 に示す．

プロセスのつながりの表現方法

ソフトウェア名 = プロセス1 [| 同期集合 |] プロセス2
(プロセスが2つの場合)

図 7 プロセスのつながり表現の記述方法

5 デッドロックパターン

5.1 概要

デッドロックが発生している箇所を特定するために，様々なデッドロックの事例を分析して，デッドロックの要因となる記述部分を抽象化し，CSP 記述を行う．デッドロックとは，2 つ以上のプロセスが互いの処理待ちにより結果としてどちらの処理も先に進まなくなることであり，デッドロックの要因を分析し，抽象化して CSP 記述したものがデッドロックパターンとなる．デッドロックが発生する要因は大きく以下の 3 つに分類できる．

- アスペクト自身に問題がある
- アスペクト間記述に問題がある
- アスペクトの組み合わせ上，問題がある

5.2 デッドロックの一般形

デッドロックが起こりうるパターンの一般形の記述を行う．パターンとして，任意の 2 つのイベントの交差によりデッドロックが発生する場合がある．一般形は図 8 のように記述することができる．

PA=□x:A@x→PA□SKIP
P(x)=PA; e.0 → PA; e.1 →P(x)
Q(x)=PA; e.1 → PA; e.0 →Q(x)
forall i, j:{0...N} | i !=j @ P(i)[A(i) ‖ A(j)] Q(j)

図 8 デッドロックの一般形

6 検証プロセス

6.1 検証プロセスの流れ

CSP で記述されたものを FDR を用いて検証することで，デッドロック，スタベーションなどを検出する．

図 9 に UML の CSP への変換から，検査までの一連の流れを示す．

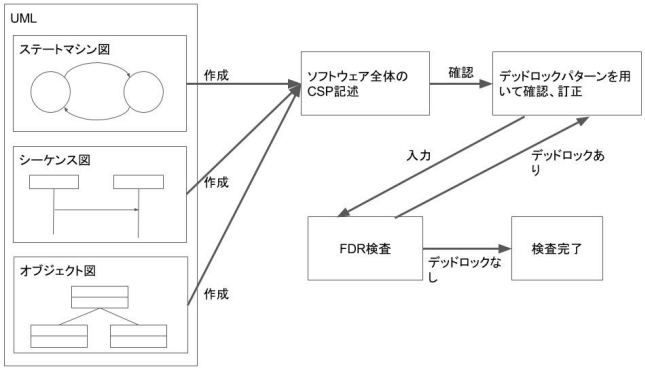

図 9　検証プロセス

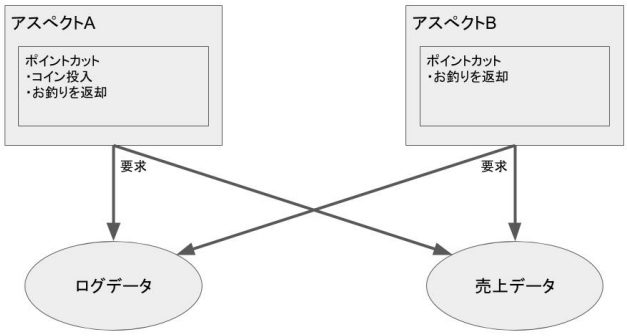

図 10　事例による考察

6.2　事例による考察

図 10 のような，「売り上げを記録」と「ログを記録」という機能が加えられた自動販売機を例にデッドロックパターンの適用方法を示す．アスペクト A は売上データをもとに入れられたコイン，お釣りとして出したコインをログデータとして記録するアドバイスを持ち，アスペクト B はログデータと売上データを取得し，金額が同じことを確かめるチェック機能をアドバイスに持つ．各ポイントカットにおいてアスペクト A はコインを投入，お釣りを返却，アスペクト B はお釣りを返却とする．アスペクト A は売上データ，ログデータの順で取得し，アスペクト B はログデータ，売上データの順で取得する．例を基にした CSP 記述の一部を図 11 に記述する．

```
VM = coin → juice → VM
PERSON = coin → ( juice → PERSON [] return → PERSON )
Wait = get_sales → Have_sales → logging → Wait
Wait = get_log → Have_log → get_sales → check → Wait
```

図 11　事例の CSP 記述

例に対して図 8 のデッドロックパターンを適用し，検証プロセスで利用できるかどうか確かめる．

上記の CSP 記述より get_sales はアスペクト A，B 両方に記述されており，またアスペクト A にログデータを必要とする logging，アスペクト B にログデータを要求する get_log という同じデータを必要とする処理が交差した形で記述されていることが分かる．同じイベントが交差された形で記述された図 8 のデッドロックパターンに当てはまるので，この CSP 記述は FDR 検査した際にデッドロックのが検出されると考えられる．

実際に FDR 検査を用いた時，当てはまる部分でデッドロックが検出されたので，デッドロックパターンによるデッドロックの検出は正しいと言える．

7　おわりに

本研究ではアスペクト指向を用いて設計されたソフトウェアを並行動作しているものと捉え，その振舞いを CSP 記述した．加えて，アスペクト指向において起こり得るデッドロックパターンを明示し，一般形の作成を行った．実際に事例を用いてデッドロックパターンが適用できるか検証することでデッドロックパターンの有用性の証明を行った．

今後の課題としては，他に考えられるデッドロックの要因の抽出とデッドロックパターンの一般形の作成が挙げられる．これらにより，設計段階での検証の精度の向上や，検証作業での負担を軽減することができると考えられる．

参考文献

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

[2] FormalSystemsLtd, *FDR2UserManual*, www.fesl.com/documentation/html/fdr2manumal.html, 2005.

[3] 石尾隆，楠本真二，井上克郎，“プログラムスライスを用いたアスペクト指向プログラムのデバック支援環境，”オブジェクト指向最前線 2003，pp.129-136, Sep. 2003.

[4] 磯部祥尚，並行システムの検証と実装，近代科学社,2012.

[5] 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, June. 2002.

[6] 海津智宏，磯部祥尚，鈴木正人，“SDVerifier:プロセス代数 CSP を用いたシーケンス図検証ツール，”コンピュータソフトウェア，vol.32, no.1, pp.234-252, Feb. 2015.

[7] 四野見秀明，玉井哲雄，“アスペクト指向における織り込みによる影響波及解析，”コンピュータソフトウェア，vol.23, no.3, pp.170-188, July. 2006.