E-AoSAS++ における実行前検査に関する研究

2006MI020 長谷川 裕記　　2006MI057 神谷 浩翔　　2006MI135 岡嶋 智史
指導教員　野呂 昌満

1　はじめに

本研究室では，組込みシステムの開発を目的としたアスペクト指向ソフトウェアアーキテクチャスタイル E-AoSAS++(Aspect Oriented Software Architecture Style for Embedded systems)[5] を提案している．E-AoSAS++ では，システムを並行に動作する状態遷移機械の集合と捉え，アーキテクチャの設計段階でシステムの振舞いをイベント通信として記述する．ソフトウェア開発におけるアスペクト指向技術の適用は，複数のモジュールに関わる横断的な関心事を分離することができる反面，全体の振舞いを把握することが難しくなる．

本研究の目的は，E-AoSAS++ におけるアーキテクチャのフォールト (Fault) を分析し，フォールトがないことを系統的に検証する方法を考察することである．アーキテクチャのフォールトと，検査で表れるフェーリア (Failure) の関係を明示することにより，アーキテクチャの検証が可能になると考えた．

本研究では，アーキテクチャの振舞い記述を CSP[1] 記述に変換し，CSP の代表的なモデル検査ツールである FDR[2] を用いて検証を試みた．モデル検査とは，システムの振舞いを網羅的に走査して，システムが満たすべき性質を自動的に検査する手法である．

E-AoSAS++ におけるシステムの検査をおこなう際に，本研究で提案する段階的検証法を適用した．段階的検証法を用いることで，「イベント実行順序の逆転」のフォールトを系統的に検査することができた．段階的検証法を基に，フォールトの特定を容易にする開発手順について考察した．

2　E-AoSAS++ の計算モデルの特徴

E-AoSAS++ に基づくシステムの振舞いを，(1) キューを介した非同期通信，(2) アスペクトの織り込み，の二つの観点から説明する．

2.1　キューを介した非同期通信

E-AoSAS++ に基づいて設計されたシステムは，複数の並行に動作する状態遷移機械 (Concurrent State Transition Machine, 以下 CSTM と呼ぶ) からなる．各 CSTM は，図 1 のように，キューを介して互いにイベントを非同期に通信することで並行動作を実現している．キューは各 CSTM ごとに存在する．

2.2　アスペクトの織込み

E-AoSAS++ では，横断的な関心事をアスペクト間記述を用いて記述する．アスペクト間記述では，ジョインポイントに対してアドバイスを記述することができる．ジョインポイントは，状態遷移機械の状態とイベントの組みで表され，アドバイスには，図 2 に示すように，アクションの実行前に実行される before アドバイスと，アクションの実行後に実行される after アドバイスがある．

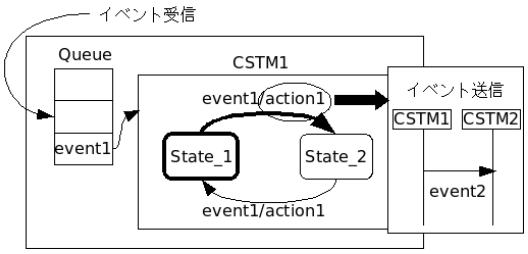

図 1　キューを介したイベントの送受信

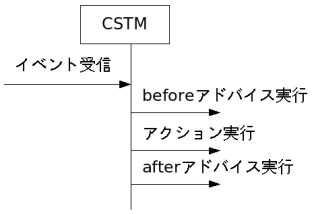

図 2　アドバイス実行のタイミング

3　アーキテクチャのフォールトの分析

E-AoSAS++ に基づいて設計したアーキテクチャ上に存在するフォールトとフォールトに起因するフェーリアについて (1) 決定的な振舞い，(2) 非決定的な振舞い，(3) アスペクトの合成，の三つの観点から分析した．

3.1　決定的な振舞い

決定的な振舞いとは，着目するイベントの系列が一意に決定される振舞いである．自動販売機を例にすると，コインを投入した後にボタンを押下すると商品が排出される，といった決定的なイベント系列を対象とする．

決定的な振舞いに起因するフォールトとして，システム記述の誤りや仕様 (期待するイベントの系列) が間違っている場合が考えられる．フォールトに起因するフェーリアとして，(1) 仕様を満たさない，(2) デッドロック，(3) ライブロック，が考えられる．これらのフェーリアからフォールトを特定する方法は，フェーリアが発生するまでのイベント系列を追うことである．イベント系列と仕様を比較することで，フォールト箇所の特定は容易にできる．

3.2　非決定的な振舞い

非決定的な振舞いとは，複数のイベントが並行実行することで，イベントの系列が一意に決定されない振舞いのことである．並行実行における一般的な問題として，「相互排除の問題」・「生産者と消費者の問題」[3] に代表されるプロセスの実行順序が非決定的であることによる問題がある．本研究では，イベント実行順序に着目して

並行実行による問題を考えた．

イベントの実行順序の逆転

イベントの実行順序の逆転とは，複数のイベントを決まった順番に送信した時に，CSTM がイベントを受理してアクション実行する順番が，送信したイベントの順番と逆になることである．

イベント実行順序の逆転を図 3 を例にして示す．`eventA` → `eventB` の順番にイベントを送信しても，`CSTM2` → `CSTM1` の順番にイベントを受理すると，`eventA`，`eventB` の実行順序はイベント送信した順番とは逆になってしまう．

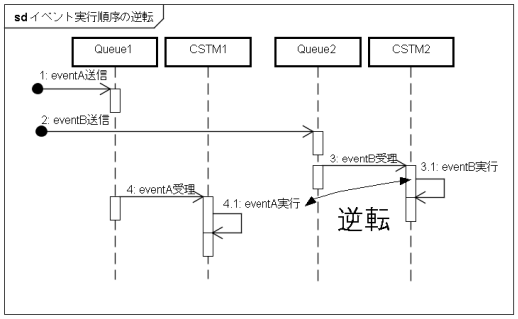

図 3　イベント実行順序が逆転する例

このフォールトを検出するのは困難である．なぜなら，検査でフェーリアが出現した場合，逆転が起こった箇所はどこなのか特定するのは困難だからである．また，逆転が起こったか否かを調べるのも困難である．

3.3　アスペクトの合成

アスペクトの合成による問題としてアスペクトの干渉 [4] がある．アスペクトの干渉とは，同一ジョインポイントに同一実行タイミングのアドバイスが複数存在する場合に，その複数の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．

アスペクトの干渉の一つとして，合成した二つのアスペクトの相互依存がある．E-AoSAS++ の計算モデルにおいて，アドバイスの実行におけるイベント送信先 CSTM に同一の CSTM を含む場合にこれが発生する．

このフォールトを検出するのは困難である．なぜなら，検査で表れるフェーリアは，イベント送信先 CSTM の処理に依存するので一意に特定できないからである．

4　CSP による記述と検証

E-AoSAS++ に基づくアーキテクチャの記述を，CSP を用いて検証する方法について議論する．CSP でアーキテクチャと検証方法を記述することで，FDR で自動的に検証することができる．

4.1　CSP 記述への変換

E-AoSAS++ に基づくアーキテクチャの振舞いを CSP で記述する方法 [6] について説明する．CSTM のキューを介した通信や外部環境とシステムとの並行実行はライブラリとして汎用的に表現する．システムは `SYSTEM = ||| STM [|送受信イベント|] ||| Queue` と記述し，外部環境とシステムとの並行実行は `ENV || SYSTEM` と記述する．ライブラリを用いることにより，検査の利用者は状態遷移図で示されている状態遷移とイベント送受信を記述するだけで，アーキテクチャの振舞いを CSP で表現できる．

アスペクトの織込みは，ジョインポイントでのアドバイス実行によるイベント送信として図 4 のように表現する．同一の実行タイミングのアドバイスが複数存在する場合，インターリーブを用いて記述することで，アドバイス実行順序が非決定的であることを示す．

```
CSTM = recieve_event -> beforeAdvice ->
    Action -> afterAdvice -> CSTM
beforeAdvice = afterAdvice
    = |||e:sendEvents@(e->SKIP)
```

図 4　アドバイス記述の概要

4.2　CSP における検証

CSP で記述されたシステムに対して FDR で以下の項目を検査できる．

- デッドロック (Deadlock)
- ライブロック (Livelock)
- リファインメント (Refinement:詳細化関係)

E-AoSAS++ に基づくアーキテクチャに対して上記項目を検査するには，仕様で想定する外部環境の振舞いを CSP で記述する必要がある．

FDR で検証できるリファインメントと，リファインメントで検証できる性質を表 1 に示した．

表 1　リファインメントで検証できる性質

リファインメント	検証できる性質
Trace refinement	安全性
Failure refinement	活性，デッドロックフリー性
Failure divergence refinement	ライブロックフリー性

検査対象を外部環境とシステムの並行動作で表現し，仕様に対してリファインメントが成立することを以下の式で表現する．

仕様 [= 外部環境 || システム

4.3　CSP による検証

3 章で分析したフォールトごとに，リファインメントを検証できるかを分析する．

4.3.1　決定的な振舞い

決定的な振舞いにおいて，リファインメントが満たされることの検査方法を述べる．決定的な振舞いにおいて入力と出力を外部環境の振舞いとし，システムと並行動作させ，システム全体の振舞いとする．リファインメント検査では，システム全体の振舞いから注目するイベント以外を隠ぺいする必要がある．隠ぺいされていないイベントにおいて，リファインメントを考え検査をおこなうことで，入力と出力に注目して検査をすることができる．

図 5 を例に検査に必要な CSP 記述を以下に述べる．

1. 着目するイベントを決める (input と out)
2. 着目するイベントに対して外部環境の振舞いを記述する
3. 着目するイベントでシステムと並行動作する
4. 着目するイベントで安全性の仕様を記述する
5. 着目するイベント以外のイベントを隠ぺいする

```
ENV = input -> out -> ENV
TEST = SYSTEM [|{|input,out|}|] ENV
SPEC = input -> out -> SPEC
assert SPEC [T= TEST (input と out に対して)
```

図 5　決定的なイベント系列における仕様の検査の CSP 記述

4.3.2　非決定的な振舞い

「イベント実行順序の逆転」を例として，フォールトに対してリファインメントで検査する方法を示す．イベント実行順序が逆転しないことは，入力順と出力順が同じになることである．これを検査するには，システムに対する外部環境の入力順とシステムの出力順が仕様として満たされることを，リファインメントを用いて検査する．

CSP 記述による検査の例を図 6 に示す．検査対象のシステムは，in1 を入力すると out1 を出力し，in2 を入力すると out2 を出力するシステムである．外部環境の入力に対して，期待する順番に出力があることを調べることで，イベント実行順序の逆転の検査実現している．

```
ENV = in1 -> in2 -> ENV
TEST = SYSTEM [|{|in1,in2|}|] ENV
SPEC = out1 -> out2 -> SPEC
assert SPEC [T= TEST (out1,out2 に対して)
```

図 6　イベント実行順序の逆転の検査を示す CSP 記述

4.3.3　アスペクトの合成

アスペクトの合成によりフォールトが起こらないことの検査方法を，アスペクト A とアスペクト B の二つを合成する場合を例にして考える．合成前のアスペクト A・B それぞれが仕様を満たすことを検査した後に，二つのアスペクトを合成後にアスペクト A・B それぞれの仕様を満たすことを検査する．合成前の検査でフェーリアが検出されず，合成後の検査でフェーリアが検出されたら，アスペクトの合成によるフォールトを確認できる．

5　段階的検証方法

E-AoSAS++ に基づくシステムの検証方法を提案し，システムの実例を用いて説明する．検証方法として，段階的検証方法を提案する．システムを小さな単位から大きな単位へと段階的に検査をすることで，フォールト箇所を絞り込むことができると考えた．

システムの段階ごとの検査には, 3 章で分析したフォールトを踏まえて検査することを考える．

1. 決定的な振舞い：想定しえる一連の流れを検査する．
2. 非決定的な振舞い：複数の処理が並行に実行する場合を検査する．
3. システム全体：システム全体が受理しえるイベントをランダムに発生させて，デッドロックやライブロックに陥らないことを検査する．

上記の項目において，下の項目になるほどシステムの振舞いが複雑になる．複雑なシステムを検証する場合，フォールト箇所の特定がむずかしいので簡単な振舞いから検証をしていくことで，検証範囲をせばめフォールト箇所を絞り込むことができる．

段階的検証方法の実例

システムの実例として，ランプとモータからなるランプモータシステムを考える．このシステムは，ランプやモータに対してスイッチとボタンで操作する (図 7)．

- スイッチの切替えで，ランプとモータのどちらを操作するかを決める．
- ボタンの押下で，ランプまたはモータを操作する．

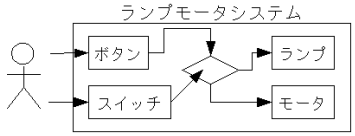

図 7　ランプモータシステムの概要

■決定的な振舞いの検査　決定的な振舞いの検査として，「ランプが仕様を満たすことの検査」，「モータが仕様を満たすことの検査」をおこなう．仕様を満たすことの検査として，外部環境がボタンの入力をおこなった場合，ランプ・モータの仕様が満たされるかを検査する．

ランプの仕様は，ランプに対するボタン入力があった場合ランプが点灯・消灯を繰り返すことである．モータの仕様も，モータに対するボタン入力があった場合モータが動作・停止を繰り返すことである．CSP 記述の例として，ランプ検査の CSP 記述を図 8 に示す．

```
-- ボタンを押し続ける環境の振舞い
ENV = snd.Button.press.env -> ENV
TEST = ENV[| EnvSystemInterface |] SYSTEM
-- ランプが点灯・消灯を繰り返すことの検査
SPEC = output.out_lamp_on -> output.out_lamp_off
        -> SPEC
assert SPEC [F= TEST \
    diff(Events,{|output.out_lamp_on ,
        output.out_lamp_off|})
```

図 8　ランプに対するボタン入力の検査

■非決定な振舞いの検査　非決定的な振舞いの検査では，「スイッチとボタンの同時入力の検査」をおこなう．スイッチとボタンが同時に入力された場合の仕様をすべて検査することは難しいので，いくつかのケースを考え検査をする．ここでは，入力としてボタン → スイッチ → ボタンと押した場合に，出力としてランプ点灯 →

モータ動作があることを検査する．CSP 記述を図 9 に示す．

```
ENV = snd.Button.press.env -> snd.Switch.toggle.env
        -> snd.Button.press.env ->STOP
TEST = ENV[| EnvSystemInterface |] SYSTEM
SPEC = output.out_lamp_on -> output.out_motor_move
        -> STOP
assert SPEC [F= TEST \
    diff(Events,{|output.out_lamp_on,
        output.out_motor_move|})
```

図 9　スイッチとボタンの同時入力の検査

■システム全体の検査　システム全体の検査では，どんな入力をしてもデッドロックとならないことを検査する．実例では，スイッチとボタンをどう入力してもデッドロックに陥らないことを検査する．CSP 記述を図 10 に示す．

```
ENV = ENV_1 ||| ENV_2
ENV_1 = snd.Button.press.env -> ENV_1
ENV_2 = snd.Switch.toggle.env -> ENV_2
TEST = ENV[| EnvSystemInterface |] SYSTEM
assert TEST2 :[ deadlock free [F] ]
```

図 10　スイッチとボタンの同時入力の検査

6　考察

6.1　イベント実行順序の逆転を防ぐ設計

5 章で用いた実例の検証をしたところ，「イベント実行順序の逆転」を検出したので，設計においてこれを防ぐ方法を考察する．「イベント実行順序の逆転」の発生を防ぐには，イベント実行順を保証する必要がある．イベント実行順を保証する方法として，イベント入力・イベント出力の制御をする方法が考えられる．

入力の制御は，システムに対して同時に入力があった場合の制御を考える必要がある．出力の制御は，一つの出力の実行中に他の出力は実行しないようにすることで出力順序を制御することである．

入力制御において，二種類の入力が同時に発生した場合に考慮しなければならない仕様として，以下の項目を挙げる．

- 片方のイベントを実行
  - どちらかのイベントを実行
  - 優先度の高いイベントを実行
- 両方のイベントを実行
  - 順序を考慮せず両方のイベントを実行
  - 優先度の高いイベントから実行
- イベント実行しない

システムの設計において，上記項目のうち仕様として適当なものを選択する．入力が三種類以上の場合も，実行イベント数と優先度の組合せから，上記の項目を列挙できる．

6.2　段階的検証におけるスケーラビリティ

段階的検証において，アスペクトごとに段階を設定する．アスペクトごとに検証することで，フォールトが潜む箇所を特定できると考えた．

CSTM を S'' として仕様 S を満たすことを調べる．

$$S \sqsubseteq \sum_{i=0}^{n} S_i''$$

段階的な検査の実現として，アスペクト S' を設定し，これを段階とする．

$$S \sqsubseteq \sum_{j=0}^{m} S_j' \quad , \quad S' \sqsubseteq \sum_{k=0}^{l} S_k''$$

上記の式において，実装 S'' が仕様 S' を満たすことを検査した後に，S' が S を満たすことを検査する．これにより，検査時にフェーリアが発生した場合に，どのアスペクトにフォールトが潜んでいるのか特定できると考えられる．

7　おわりに

本研究では，E-AoSAS++ におけるアーキテクチャのフォールトを分析し，フォールトが発生しないことを段階的検証方法をもちいて系統的に検証する方法を提案した．また，フォールトの一つである「イベント実行順序の逆転」を防ぐ設計を考察した．

今後の課題として，段階的検証方法における段階の設定を考える必要がある．また，「イベント実行順序の逆転」以外のフォールトに対してフォールトを防ぐ系統的な設計について考察する必要がある．

参考文献

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

[2] Formal Systems(Europe), *"FDR2 User Manual,"* http://fsel.com/documentation/fdr2/html/fdr2manual.html, 2005.

[3] M.Ben-Ari, *Principles of Concurrent and Distributed Programming (2nd ed),* Addison-Wesley, 2006.

[4] P. Durr, T. Staijen, L. Bergmans, and M. Aksit,“ Reasoning About Semantic Conflicts Between Aspects, ” in Proc. European Interactive Workshop on Aspects in Software, Sep. 2005.

[5] 加藤大地，蜂巣吉成，沢田篤史，野呂昌満，“ アスペクト指向に基づくソフトウェアアーキテクチャの文書化方式，” 知能ソフトウェア工学研究会 (KBSE), vol.108，no.449，pp55-60，2009.

[6] 張漢明, 蜂巣吉成，沢田篤史，野呂昌満，“ アスペクト指向ソフトウェアアーキテクチャの振る舞い検証に関する考察，” ソフトウェア工学の基礎 XVI, pp267-274，2009.