アスペクト指向ソフトウェアの実行前検査に関する研究

2005MT032 井上 敬登　　2005MT058 久野 公彰　　2005MT061 眞野 佑規
指導教員　野呂 昌満

1　はじめに

近年，ソフトウェアが大規模化している中，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．アスペクト指向は横断的関心事をアスペクトとして分離して設計し，アスペクトを織り込むことで合成しソフトウェアを作成する．我々の研究室は，組込みソフトウェアの開発をアスペクト指向を用いて開発するアーキテクチャスタイルとして，組込みソフトウェアの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++) を提案している．アスペクト指向の問題点として，合成後の動作を把握することが困難で，個々のアスペクトに欠陥がなくとも合成後のプログラムに欠陥が発生する可能性がある．設計上の不具合が実行時に発生した場合，ソフトウェアの開発において大きな手戻りが発生する．

設計段階でモデル検査を行うことで設計の不具合を発見し，手戻りを防ぐことができる．しかし，モデル検査を行うにはモデル検査言語の習得が必要となる．モデル検査言語はプログラムや設計とは異なる考えに基づいているので，習得と記述に大きな手間がかかる．

本研究の目的は，E-AoSAS++ に基づいた設計に対して行う実行前検査の手法を提案することである．設計時に実行前検査を行うことで，開発の早期に問題を発見し手戻りを防ぎ，開発期間の短縮化やコストの削減ができる．また，モデル検査言語の習得と手間がかかる問題を解決するために，設計からモデル検査言語への自動変換ツールを考える．

本研究では，はじめにモデル検査，既存の研究の理解を行い，E-AoSAS++ で設計したソフトウェアに起こる問題を整理する．次に，設計したソフトウェアとモデル検査言語との対応関係を考察し，モデル検査ツールを用いた E-AoSAS++ の検査手法の提案を行う．最後に，モデル検査言語への変換を容易にするために自動変換ツールの作成を行う．

2　モデル検査

モデル検査とは検査対象の各状態を網羅的に探索する検査方法である．設計段階でモデル検査を行うことで，ソフトウェアの問題を開発の早期に見つける事ができる．並行ソフトウェアをモデル化する手法として CSP(Communicating Sequential Processes)[2] がある．アスペクト指向では，アスペクトを織り込むことによりソフトウェアを作成する．アスペクトの織り込みを，並行処理のメッセージの通信で表現することで，アスペクト指向ソフトウェアを CSP で表現することができると考えた．

CSP においてモデルはイベント，イベントの集合を表現したプロセスを用いて表現される．CSP に基づいた検査ツールとして FDR(Failures Divergence Refinement)[3] がある．FDR では，デッドロックやスタベーションの検査が可能である．デッドロックの検査ではソフトウェアのすべてのプロセスがそれ以上遷移出来ない状態を検知する．スタベーションの検査では，あるプロセスの集合に対して，集合の内部の通信を内部イベント，外部との通信を外部イベントにわける．内部イベントが永遠に続く可能性があることを検知する．

3　E-AoSAS++

3.1　E-AoSAS++ の概要

E-AoSAS++[4] は，組込みソフトウェアを CSTM の集合と定義する．ソフトウェアに散在する関心事を適切にアスペクトとしてモジュール化することにより，柔軟性，再利用性に優れたソフトウェアを設計することができる．CSTM は相互にキューを介してイベントを送受信することで連携動作する．送受信の方法には，メッセージ通信 (MP) によるものとアスペクト間記述 (IAD) によるものの 2 種類がある．アスペクト内の送受信には MP，アスペクト外との送受信には IAD を利用する．MP による送受信では，送信イベントの順番が一意に決まる．IAD による送受信のイベントの順序は，開発者により定義されない．

次に E-AoSAS++ の開発プロセスを述べる．

1. 開発するソフトウェアの機能特性，
   非機能特性の決定
2. オブジェクト指向に基づいてソフトウェアの
   設計・検査
3. 2を基にアスペクト指向に基づいて
   ソフトウェアの設計・検査
4. アーキテクチャからコードの自動生成

各アーキテクチャに対し実行前検査を行うことで，開発の手戻りを防ぐことができる．本研究ではアスペクト指向に基づいた設計に対する実行前検査手法を提案する．

E-AoSAS++ に基づく実行前検査では UML のオブジェクト図，ステートマシン図，シーケンス図を用いて行う．これらの UML は E-AoSAS++ 基づく開発プロセスの設計段階で記述される．

・オブジェクト図:
　複合 CSTM 内の CSTM 名，インスタンス名，
　インスタンスの数，インスタンス間の関連を表現
・ステートマシン図:
　CSTM の各状態と，状態遷移のイベント，
　アクションを表現
・シーケンス図:

ステートマシン図における各アクションにおいて
他の CSTM に対し送信するイベントを表現

3.2 E-AoSAS++ の実行前検査に関する既存の研究

既存の研究 [1] により，UML のステートマシン図，シーケンス図から CSP 記述への対応関係が考えられている．図 1 のように E-AoSAS++ の CSTM の状態，アクションを CSP のプロセスとして表し，アクションの開始を示すイベントを同期することにより CSTM を表現する．

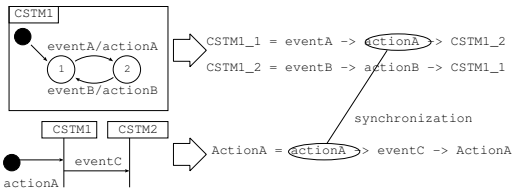

図 1　ステートマシン図，シーケンス図と CSP との対応

4 E-AoSAS++ の実行前検査手法の提案

4.1 検査概要

本研究ではモデル検査ツール FDR を利用した実行前検査を提案する．E-AoSAS++ に基づいた設計を CSP 記述へ変換し，E-AoSAS++ に基づくソフトウェアで発生し得る問題をデッドロック，スタベーションとして検出する．図 2 に実行前検査の概要を示す．本節では，E-AoSAS++ に基づくソフトウェアで発生し得る問題について整理し，CSTM の構成要素と CSP の対応関係，E-AoSAS++ に基づいたソフトウェアで発生する可能性のある問題の検査方法を考察する．

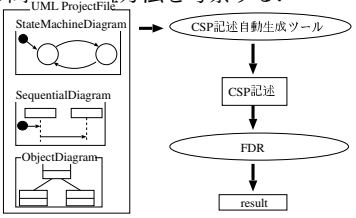

図 2　実行前検査の概要

4.2 問題の整理

E-AoSAS++ に基づき設計を行った際，ステートマシン図に記述されているイベント名が，アクションを表現するシーケンス図に該当するイベント通知が記述されていなかったり，イベント名が間違って記述されている場合がある．この誤りは設計を行った時点で発見することができるが，気付かなかった場合実行時に不具合が起こる．イベント名の記述間違いの場合，各 CSTM は本来受け取らないはずの，遷移できないイベントを受理することになる．本来記述されているべきイベントが記述されていなかった場合，遷移されない状態が存在する可能性がある．

設計段階で記述の誤りがなくとも，実行時に問題が発見される可能性がある．E-AoSAS++ では CSTM を組み合わせてソフトウェアを構築する．IAD によるイベント通知の際にはイベント通知の順番が一意に決定しないので，動作の把握が困難である．また，E-AoSAS++ に基づいたソフトウェアは複数の CSTM が並行に動作するので，実行時の動作の把握が更に困難になる．

組合せ次第では CSTM の各状態で本来受け取ると定義されていないイベントを受理する可能性がある．その時にソフトウェアがどのような動作をするかは不明確である．動作に不明確な点があることは，適切な設計でない．例を図 3 を用いて示す．

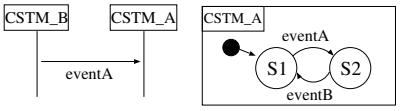

図 3　遷移できないイベントを受け取る問題の例

図 3 の場合 CSTM_A が S1 の状態の時に eventA を受理すれば問題ないが，S2 の状態の時に受理した場合，状態遷移を行えず，その際の動作が不明瞭である．

また，イベントが通知される順番次第では状態遷移を行われない状態が存在する場合がある．例を図 4 を用いて示す．

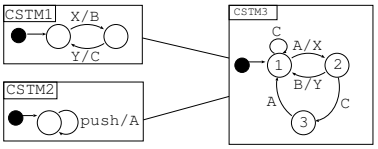

図 4　遷移されない状態が存在する問題の例

図 4 では CSTM3 は，CSTM1 と CSTM2 からのみイベントを受け取るとする．この時，イベントは常に push → A → X → B → Y → C の順に行われるので，3 の状態に遷移しない．3 の状態に必ず遷移しなくてはならないと開発者が意図していた場合，スタベーションとなる．これらを踏まえた結果，実行時の検査をする際に発見しなくてはいけない問題として，次の 2 つが存在すると考えた．

・各状態において遷移できないイベントを受け取る問題
・CSTM に遷移されない状態が存在する問題

CSTM が本来受け取らないはずのイベントを受け取ることは，各状態において受け取らないはずのイベントを受け取る問題に帰着できると考えた．

4.3 CSP 記述でのソフトウェアの表現方法

E-AoSAS++ に基づいたソフトウェアの設計を CSP を用いて表現する．各 CSTM を CSP のプロセスで表現し，CSTM を表現するプロセスを組み合わせることで，E-AoSAS++ に基づくソフトウェアを CSP のプロセスで表現する．これにより，アスペクト指向ソフトウェアの特徴をモデル検査言語で表現できると考えた．

4.3.1 考慮すべき E-AoSAS++ の要件

E-AoSAS++ に基づいたソフトウェアを CSP でモデル化する際には，下記の要件を満たす必要がある．

・アクションが終了した後，次の状態に遷移する
・CSTM がキューを介してイベントの送受信を行う
・複数インスタンスがある場合にも対応する
・IAD と MP の違い (イベントの順序) に対応する

既存の研究の対応関係では，これらの要件を考慮していないので，新しい対応関係を提案する．

4.3.2 CSTM の設計と CSP 記述の対応関係

E-AoSAS++ の設計において，CSTM の振る舞いはステートマシン図と，アクションを表すシーケンス図で表現される．設計段階では CSTM の振る舞いは状態遷移とアクションの側面が考えられるので，CSTM の振る舞いををモデル検査言語で表現するには，状態遷移とアクションの両側面を考慮してモデル化を行う必要がある．

状態遷移機械の表現方法

CSP のプロセスで状態遷移機械を表現する方法の，既存の研究からの変更点は次の 2 点である．

・アクションの開始と終了をイベントで表現
・インスタンスを変数 (i) で表現

アクションの開始と終了を表現することで，アクション終了後に次の状態に遷移することを表現した．
インスタンスを変数で表現することで，一つの CSTM に対し複数のインスタンスがある場合に対応することができ，同じプロセスで表現することができる．プロセス内のイベントも，変数 (i) を付加することで，インスタンスごとのイベントを区別する．
状態遷移機械の表現方法の例を図 5 で示す．

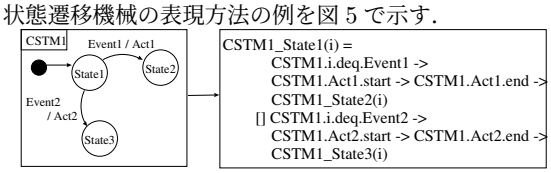

図 5　状態遷移機械を表現した CSP のプロセスの例

アクションの表現方法

CSP のプロセスでアクションを表現する方法の，既存の研究からの変更点は次の 3 点である．

・アクションの開始と終了をイベントで表現
・オブジェクト図からインスタンス間の関係を取得し，表現
・IAD，MP を考慮した CSP 記述の表現

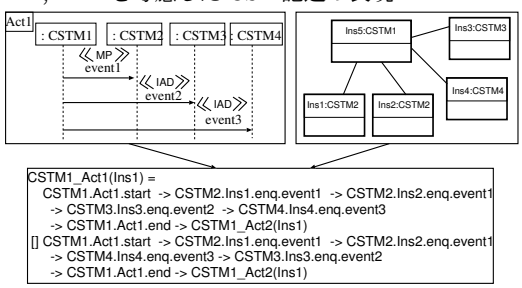

図 6　アクションを表現した CSP のプロセスの例

アクションの開始と終了を表現することで，アクションが終了した後に次の状態に遷移することを表現した．
オブジェクト図からインスタンス間の関係を取得し，対応するプロセス，イベントにインスタンス名を付加することで，インスタンスを区別した．CSTM 間の関係において，MP の場合は記述されている順番のみを記述する．IAD の場合は，全ての順序の組み合わせを選択で記述する．図 6 に設計と CSP 記述の対応の例を示す．

キューの表現方法

E-AoSAS++ では各 CSTM はキューを用いてイベントのバッファリングを行うので，キューも CSP で表現する必要がある．キューは CSP のプロセスとして表現し，CSTM ごとに作成する．インスタンス名を変数で表現することでインスタンスごとのキューを表現する．格納しているイベントの種類は変数を用い保持する．キューに格納するイベントはイベント名に enq を付加し，キューから取り出すイベントはイベント名に deq を付加する．上記の表現方法を基に CSP で表現すると，図 7 のような CSP 記述となる．

```
CSTM1_Queue(i,x1,x2) =
    if (x1 == nulle) then CSTM1.i.enq?x -> CSTM1_Queue(i,x,nulle)
    else CSTM1.i.deq!x1 -> CSTM1_Queue(i,x2,nulle)
        [] (CSTM1.enq?x -> (
            if (x2 == nulle) then CSTM1_Queue(i,x1,x)
            else CSTM1_Queue(i,x1,x2) ) )
```

図 7　キューを表現した CSP のプロセスの例

4.3.3 ソフトウェア全体の表現方法

各 CSTM の状態遷移機械を表現したプロセスとアクションを表現したプロセスを同期させて，CSTM を表現したプロセスを作成する．各プロセスのアクションの開始イベントと終了イベントで同期を行うことで，アクションが終了した後に，遷移先の状態へ遷移することを表現する．CSTM を表現するプロセスも変数 (i) を用いてインスタンスを表現する．インスタンスの情報はオブジェクト図から取得する．ソフトウェア内の全ての CSTM を表現したプロセスと，キューのプロセスを同期させ，ソフトウェアを表現したプロセスを作成する．

4.4 E-AoSAS++ で発生し得る問題の検出方法

本節では，4.3 節で説明した CSP 記述での表現方法を基に，4.2 節で挙げた問題の検出方法について考察する．CSP を用いたモデルを検査するには，問題をデッドロックやスタベーションとして扱う必要がある事を踏まえた検査手法を提案する．

4.4.1 受理できないイベントを受け取る問題の検査方法

受理できないイベントを受け取る問題を，デッドロックとして検出する手法を考えた．次に検査手順を示す．

1. すべての CSTM に遷移できない状態を追加する．
2. 受理できないイベントを受け取ったとき，追加した状態への遷移を追加する．追加した遷移をした場合，他の CSTM を止める為に error イベントを全ての CSTM に送信する．error イベントを受信した CSTM は追加した状態に遷移し，それ以上遷移することは出来ない．
3. 全ての CSTM を追加した状態に遷移させることで，作為的にソフトウェアをデッドロックの状態にして，デッドロック検出機能を用いて検出する．

図 8 に状態とイベントの追加の一例を図示する．

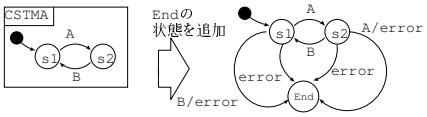

図 8　受理できないイベントを受け取る問題の検査方法の例

4.4.2 遷移しない状態が存在する問題の検査方法

遷移しない状態が存在する問題を，CSP におけるスタベーションとして検出する手法を考えた．必ず遷移しなくてはいけない状態への遷移を外部イベント，他のイベントを内部イベントとして扱うことで，スタベーションとして検出できると考えた．しかし内部イベントとして扱うイベントは隠蔽する必要があるが，CSP の隠蔽はイベント名のみを指定し隠蔽するので，CSTM 内に同じイベントによる遷移が複数ある場合，状態遷移の特定ができない．そこで検査する遷移に新たに検査用のイベント (以下ラベルと表現) を追加し，ラベルを外部イベントとして扱う．ラベル以外のイベントを内部イベントとして隠蔽することでスタベーションとして検出できる．検査は次の手順により行う．

1．検査対象の状態への遷移にラベルを追加
2．ラベル以外のイベントを隠蔽する
3．FDR でスタベーションの検査を行い，ラベルの遷移が行われる可能性があるか検査する．

検査方法の例として，図 9 に例を挙げる．状態 C の状態に遷移するかを検査したいとき，状態 C に遷移する EventC にラベルを付加し他のイベントを隠蔽しスタベーションの検査で判断する．

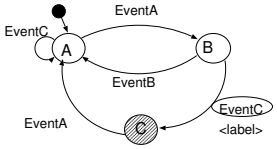

図 9　遷移されない状態が存在する検査方法の例

4.5 実行前検査支援ツール

モデルの作成や検査言語の習得の手間を軽減するために，UML による図式表現から CSP 記述を自動生成するツールの作成を行った．今回作成したツールでは，UML でソフトウェアを図式表現し，作成した図式を入力として，CSP 記述を出力する．プログラムは約 1400 行の Java のコードで実現した．

5 考察

5.1 E-AoSAS++ の要件

本研究では，次の要件を考慮した対応関係を提案した．アクションと状態遷移の順番については，アクションの開始と終了時に同期させることにより対応した．CSTM 間の通信方法については，キューを CSP で表現することにより対応した．複数インスタンスがある場合には，オブジェクト図からイベントの宛先を読み取ることにより対応した．IAD，MP については，イベントの順序を考慮しモデル化を行うことで対応した．

5.2 アスペクト指向ソフトウェアのモデル化

並行ソフトウェアを検査する理論を用いてアスペクト指向ソフトウェアをモデル化し，実行前検査手法を提案した．モデル化の際に，アスペクト指向ソフトウェアの各アスペクトを CSP における各プロセスに対応させる事を考えた．アスペクト間のつながりを，CSP のプロセス間のイベント通信に対応させる事を考えた．また，アスペクト合成時の織り込みの順番の非決定性を，全ての可能性を選択を用いて表現する方法を考えた．CSP における各プロセスを合成することでソフトウェアを表現し，アスペクト合成時の非決定性を表現したことで，アスペクト指向の特徴であるアスペクトの合成を CSP で表現できた．よって，本研究で提案したアスペクト指向ソフトウェアのモデル化は妥当であるといえる．

5.3 検査の妥当性

本研究では E-AoSAS++ に基づいたソフトウェアで発生し得る問題をデッドロックやスタベーションとして検出する方法を提案した．本来受理しないイベントを受け取る問題は，問題が発生した際に強制的にデッドロックの状態にすることでデッドロックとして検出できると考えた．遷移されない状態が存在する問題は，本来外部イベントや内部イベントなどの区別を行わないイベントに対して, 外部イベントとして扱うイベントを追加し, 他の全てのイベントを内部イベントとして扱う事でスタベーションとして検出できると考えた．実際に問題が発生する設計に対し検査を行ったところ，検出することができた．E-AoSAS++ で発生し得る問題を CSP で発見できる問題に帰着させたので，この検査法は E-AoSAS++ で発生し得る問題を発見できると考える．

6 おわりに

本研究では E-AoSAS++ の設計と CSP 記述との対応づけを行い，E-AoSAS++ に基づいたソフトウェアの検査手法を提案した．設計から FDR 記述への自動変換ツールを作成した．自動変換ツールを使用することにより，CSP を記述する手間が省け，開発に実行前検査を取り入れる事が容易になると考える．実行前検査を開発に取り入れることにより手戻りを防ぐことができる．

今後の課題として検査結果の UML へのフィードバックツールの作成が挙げられる．エラー箇所を UML で表現するフィードバックツールを作成することにより，エラー箇所の特定に FDR の知識が不要となり，デバッグを行うことがが容易になる．

参考文献

[1] 安孫子正康, “ソフトウェアの実行前検査に関する研究,” 南山大学大学院数理情報専攻数理情報研究科修士論文, Feb. 2008.

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

[3] Formal Systems(Europe) Ltd, *FDR2 User Manual*, 2005; www.fesl.com/documentation/fdr2/ html/fdr2manual.html.

[4] M. Noro, A. Sawada, Y. Hachisu, and M. Banno, “E-AoSAS++ and its Software Development Environment,” *Proceeding of the 14th Asia-Pacific Software Engineering Conference(APSEC 07)*, Dec. 2007, pp. 206-213.