2009－SE－163 (16)
2009／3／18

E-AoSAS++に基づく開発支援環境 －実行前検査ツールの提案－

加藤大地†，蜂巣吉成‡，沢田篤史‡，野呂昌満‡

† 南山大学大学院数理情報研究科　　‡ 南山大学数理情報学部

本稿ではアスペクト指向設計されたソフトウェアの実行前検査の方法を提案する．
われわれは組込みソフトウェアを並行状態遷移機械の集合ととらえて，組込みソフトウェアのため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，アスペクトが織り込まれたソフトウェアの動作を把握することが困難であることが挙げられる．個々のアスペクトに欠陥がなくとも織り込み後のプログラムには欠陥が含まれる可能性がある．
本稿では，アスペクト指向ソフトウェアの検査をモデル検査に帰着させて，設計段階における実行前検査の方法を提案する．E-AoSAS++ に基づいて記述されたアーキテクチャを対象に，アスペクト指向ソフトウェアにおける問題を整理する．モデル検査を行うために，アーキテクチャを記述したUML 図と CSP 記述との対応関係を考察し，UML 図から CSP 記述への変換を行うツールを試作した．提案した方法がアスペクト指向ソフトウェアにおける問題の解決を支援できることを示す．

Software Development Environment based on E-AoSAS++ - Design of the Pre-Execution Check Tool -

Daichi Kato†, Yoshinari Hachisu‡, Atsushi Sawada‡, Masami Noro†

† Graduate School of Mathematical Sciences and Information Engineering, Nanzan University

‡ Faculty of Mathematical Sciences and Information Engineering, Nanzan University

We propose methods for pre-execution check of aspect-oriented software.
We have proposed the aspect-oriented software architecture style E-AoSAS++ (Aspect-oriented Software Architecture Style for Embedded systems) in which a set of concurrent state transition machines organizes a software. In aspect-oriented software, it is difficult to understand behavior of software woven with aspects. Though software without aspects is designed correctly, woven software may fail on execution.
In this article, we propose pre-execution checking methods for aspect-oriented software using model checking techniques in a design phase. We summarize problems on architecture based on E-AoSAS++ and translation rules between UML diagrams and CSP codes for model checking. We implemented a tool generating CSP codes from UML design. We show that our methods are effective to detect defects on aspect-oriented software.

1. はじめに

ソフトウェアのモジュール化技法の一つとして，アスペクト指向技術が提案されている [1]．アスペクト指向では，ある特性や機能により規定されたコンサーンによって定義されたソフトウェアの一部分をアスペクトしてモジュール化する．複数のアスペクトが合成されて (織り込み，weave という)，ソフトウェア全体の機能を実現する．われわれは組込みソフトウェアを並行状態遷移機械の集合ととらえて，組込みソフトウェアのため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 [2]．

アスペクト指向の問題点として，合成されたソフトウェアの動作の把握が困難であることが挙げられる．個々のアスペクトには欠陥がなくとも合成後のプログラムに欠陥が含まれる可能性がある．

本研究の目的は，アスペクト指向ソフトウェアの挙動の検査をモデル検査に帰着させて，設計段階における実行前検査の方法を提案することである．E-AoSAS++ に基づいて記述されたアーキテクチャを対象に，アスペクト指向ソフトウェアにおける問題を整理する．モデル検査を行うために，アーキテクチャを記述した UML 図と CSP 記述との対応関係を考察し，UML 図から CSP 記述への変換を行うツールを試作した．提案した方法でアスペクト指向ソフトウェアにおける問題の解決を支援できることを示す．

2. E-AoSAS++

2.1 概　　要

E-AoSAS++は，我々が提案するソフトウェアアーキテクチャのスタイルであり，特に組込みシステムのソフトウェアアーキテクチャを記述する目的で設計されている．一般に組込みソフトウェアでは，実時間処理に対する要求，耐故障性ハードウェア制約などの横断的な関心事が存在する．E-AoSAS++ではこれらの関心事を適切にモジュール化するために，アーキテクチャ設計にアスペクト指向の考え方を導入している．

E-AoSAS++におけるアーキテクチャの基本構成要素であるモジュールは，並行状態遷移機械（CSTM）である．複数CSTMの階層的な包含関係によってCSTMの複合構造を表現し，CSTM間の通信によって使用や参照などの依存関係を表現する．個々のCSTMの振舞いは状態遷移論理により表現されており，複数のCSTMにまたがる振舞いはCSTM間のメッセージ通信とその系列によって表現される．これにより，主たる関心事を実現するCSTMと，実時間処理や耐故障性などの関心事を実現するCSTMを明確に分離して記述できる．

E-AoSAS++では，これらのCSTMに共通する関心事として，並行処理および状態遷移のための論理を，それぞれアスペクトとして記述する．並行処理アスペクトは，他のCSTMからのイベントを受信し，CSTMの起動，中断などの同期処理を行う．非同期に他のCSTMとイベントの送受信を行うためにキューが利用される．並行処理アスペクトがキューからイベントを取り出すと，IADにより状態遷移アスペクトにイベントが送られ，状態が遷移する（**図1**）．各CSTMに共通する論理をアスペクトとし，CSTM独自のアプリケーション論理から独立させ，アスペクト間記述（IAD: Inter Aspect Description）を介して疎結合させることで，柔軟で再利用性の高いアーキテクチャ設計を可能とする．

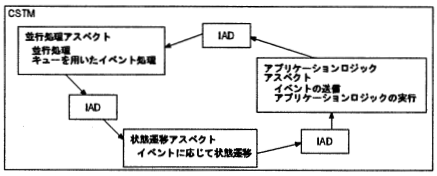

図1　E-AoSAS++の概略

2.2 ソフトウェア開発プロセス

E-AoSAS++を用いてアーキテクチャを設計し，そのアーキテクチャに基づいてプロダクトを開発する．E-AoSAS++では概念アーキテクチャおよび実装アーキテクチャと呼ぶ二種類のアーキテクチャを設計する．

概念アーキテクチャは，アプリケーションの概念的な構造を表現するものであり，対象ドメインの概念構造を分析した結果を集約モデルとして表現する．この段階ではアスペクトへの分割は意識しない．

実装アーキテクチャは，共通する機能のアスペクトへの分割，利用可能なハードウェア構成などを想定して作成するアスペクト指向モデルである．詳細な記述方法は2.3節で述べるが，アスペクトとして抽出されるCSTMの間の静的構造および依存関係を表現したシステム静的構造モデル，各CSTMの状態遷移による振舞いを表したモジュール振舞いモデル，イベントの送受信によるCSTM間のやりとりを表したモジュール間通信モデル，CSTMの物理的な配置について記述されたモジュール配置モデルの四種類のモデルを用いて表現される．

実装アーキテクチャに対して，実行前検査を行う．アスペクト指向ソフトウェアの動作に対して検査を行い，誤りを発見したいので，モジュール振舞いモデルとモジュール配置モデルに対して検査を行う．実行前検査によって誤りが発見された場合はアーキテクチャを修正する．その後，実装アーキテクチャにおける各モジュールの内部論理をプログラムコードに変換する．

2.3 実装アーキテクチャの文書化

E-AoSAS++の実装アーキテクチャはUMLに基づいて記述される．UMLを採用した理由は，開発者間の共通の理解が得やすく，編集のためのUMLツールが一般に利用可能であるからである．

表1にE-AoSAS++の実装アーキテクチャ記述とUML図との対応づけの方針，アーキテクチャ文書化のスタイル“Views and Beyond”（V&B）[3]との関係を示す．V&Bでは，アーキテクチャ記述の視点をモジュール視点，処理（コンポーネント・コネクタ）視点，配置視点に大別しているが，実装アーキテクチャではこれらの視点をカバーしている．

表1　実装アーキテクチャの記述法

文書名	UMLの記法	V&B視点
システム静的構造	コンポーネント図	モジュール
	クラス図	モジュール
モジュール振舞い	ステートマシン図	処理
	シーケンス図	処理
モジュール間通信	シーケンス図	処理
モジュール配置	オブジェクト図	配置

システム静的構造モデルはコンポーネント図を用い

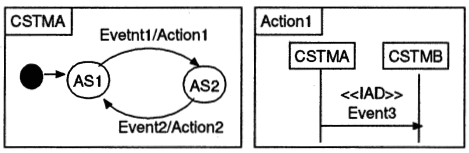

図 2 モジュール振舞いモデルの表現

て，モジュールの間の複合構成関係と依存関係を表現する．E-AoSAS++はアスペクト指向アーキテクチャであるので，複数のモジュールに横断するアスペクトの関係を記述できなければならない．クラス図では素直にこの関係を表現できないので，コンポーネント図により表現する．アスペクトの複合構成をコンポーネントの階層関係で表現し，横断的な依存関係を提供インタフェースと利用インタフェースの共有関係で表現する．ただし，コンポーネント図におけるコンポーネントやインタフェースは，それ自身が実体化される対象ではないので，これらの要素の詳細構造については，実体化可能なクラスとしてクラス図で表現する．

モジュール振舞いモデルでは，アーキテクチャを構成するモジュールがどのように動作するかを示す必要がある．E-AoSAS++のモジュールは CSTM であるので，ステートマシン図とシーケンス図を用い，イベント受け取りに伴う状態遷移とアクションの仕様を記述する．CSTM 間のやりとりは，アスペクト間記述を用いた動的で柔軟な通信方式を表現する必要がある．そこで，本モデルには，CSTM の間のやりとりがアスペクト間記述を参照することで動的に行なわれるのか，固定的に相手を指定したメッセージ通信によって行なわれるのかを示すステレオタイプ（<<IAD>>および<<MP>>）を導入した．

図 2 にモジュール振舞いモデルの記述例を示す．ここでは，CSTMA が Event1 を受け取ると，Action1 を実行して状態を変化させる．その際，Event3 をアスペクト間記述で CSTMB に送信することが示されている．

モジュール間通信モデルでは，モジュール間にどのようにメッセージがやりとりされるか，その結果システム全体としてどのような振舞いが実現されるかを，シーケンス図を用いて記述する．

ここまで記述した実装アーキテクチャのモデルは，いずれもモジュールの型の間の関係を示すものである．これらのアーキテクチャにしたがって構築されたアプリケーションが動作する時には，型定義にしたがって実体化されたモジュールのインスタンスがハードウェアや OS プロセスなどの計算主体に割り当てられることになる．一つのモジュール型が複数のインスタンスを持つ場合には，個々のインスタンスがどのような計算主体に割り当てられ，他のモジュールとどのように関連するのかを示す必要がある．モジュール配置モデルでは，オブジェクト図を利用してこの関係を示す．

3. モデル検査

モデル検査とは有限状態並行システムの状態を網羅的に探索して，正当性を自動的に検査するための方法である [4]．プロセス代数の一つである CSP[5] によりソフトウェアを並行プロセスの集合として記述し，検査ツール FDR[6] を用いて検査する方法や，チャネル通信オートマトンに基づいた言語 Promela によりソフトウェアを記述して検査ツール SPIN[7] で検査する方法などが知られている．

CSP はイベントの有限集合としてオブジェクトの挙動を記述する論理体系である．イベント列をプロセスと呼び，プロセスによりオブジェクト群の挙動を記述する．実装アーキテクチャのモジュール振舞いモデルにおける状態遷移とアクションは CSP のプロセスとして表現できる．CSTM 間で送受信されるイベントを CSP のイベントとし，アクションの起動を CSP のイベントで表現することで，それらのイベント列として状態遷移とアクションを記述できる．例えば，図 2 のモジュール振る舞いモデルは次のように記述できる☆．`Action1Trigger` が CSP におけるアクションの起動のイベントである．

```
AS1 = Event1-> Action1Trigger -> AS2
AS2 = Event2-> Action2Trigger -> AS1
Action1 = Action1Trigger -> Event3 -> Action1
```

SPIN はチャネル通信オートマトンに基づいた Promela と呼ばれる専用言語によりモデルを記述する．モジュール振舞いモデルにおける状態遷移は Promela によりオートマトンとして自然に記述できる．しかし，アクションにおけるイベント列をオートマトンで記述した場合，記述が繁雑になる恐れがある．

以上より，本研究ではモデル検査に CSP と FDR を用いることにする．

4. E-AoSAS++の実行前検査方法の提案

4.1 概　　要

本研究では E-AoSAS++に基づいた実装アーキテクチャに対する実行前検査の方法を提案する．実行前検査手順の概要を**図 3** に示す．CSP 記述自動生成ツールを用いて，実装アーキテクチャを CSP 記述へ変換し，モデル検査ツール FDR を利用して，アーキテク

☆ 実際の変換規則はこれより複雑である．4 節で詳細を述べる．

チャ記述の誤りをデッドロック，ライブロックとして検出する．

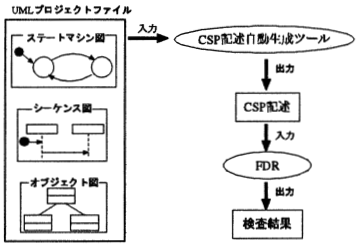

図 3　実行前検査手順の概要

本節では，E-AoSAS++に基づく実装アーキテクチャにおける記述の誤りについて整理し，CSTM の構成要素と CSP の対応関係，実装アーキテクチャにおける記述の誤りの検査方法について提案する．

4.2　実装アーキテクチャ記述における問題の整理

アスペクト指向ソフトウェアでは個々のアスペクトには欠陥がなくとも，織り込み後のプログラムに欠陥が含まれることがある．これは，アスペクトが織り込まれたソフトウェアの動作把握が困難であるというアスペクト指向における問題に起因する．

E-AoSAS++における実装アーキテクチャにおいても，同様の問題が起こる．実装アーキテクチャにおけるモジュールは CSTM であり，モジュール振舞いモデルでは CSTM の状態遷移とアクションの仕様を記述する．個々の CSTM の記述には誤りはなくても，複数の CSTM を連携させた場合に開発者の意図しない動作を CSTM が行う可能性がある．開発者の意図しない動作として，次の 2 つが挙げられる．

(1)　CSTM が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

(2)　CSTM に遷移されない状態遷移が存在する

(1) の例を**図 4** に示す．図 4 では，CSTM_A から CSTM_C にはイベント e1 と e3 を送信する．これらのイベントは 2.3 節で述べたように，アスペクト間記述を参照して動的に行われるので，イベントの送信順序が常に同じとは限らない．e1, e3 の順序で CSTM_C にイベントが送信されたときは遷移できるが，e3, e1 の順序で送信された場合は遷移ができない．

(2) の例を**図 5** に示す．CSTM3 は，CSTM1 と CSTM2 からのみイベントを受け取るとする．この時，イベントが push → evA → evX → evB → evY → evC の順に行われると，CSTM3 は S33 の状態には決して遷移しない．開発者が S33 の状態へ遷移することを意図していた場合，これは意図とは異なる動作となる．

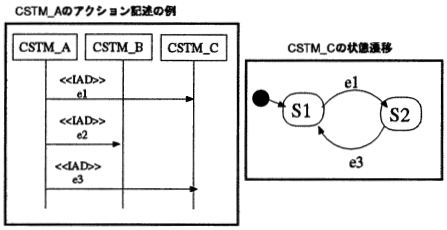

図 4　遷移できないイベントを受け取る問題の例

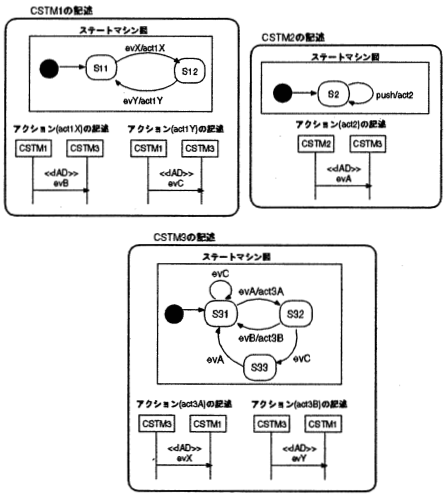

図 5　遷移されない状態が存在する問題の例

本稿ではこれらの問題を検出できる実行前検査の方法を提案する．

4.3　実装アーキテクチャの実行前検査における要件

E-AoSAS++に基づいた実装アーキテクチャを実行前検査するには，実装アーキテクチャの意味を CSP で記述しなければならない．CSP で記述する際の要件を次に示す．

- MP と IAD によるイベント送信の違いを記述する．IAD によるイベント送信では順序を一意に特定できないので，CSP でもこの性質を記述する必要がある．
- CSTM のキューを介したイベントの送受信を記述する．E-AoSAS++ではイベントの送受信にキューを用いている．キューの動作を CSP で表現することで，キューが原因となる記述の誤りも実行前検査で検出できる．
- 同一の CSTM のインスタンスが複数ある場合に，各インスタンスへのイベントを区別する．
- 状態遷移では，アクションが終了した後に次の状態に遷移する．

4.4　実装アーキテクチャの CSP による表現方法

E-AoSAS++に基づいた実装アーキテクチャのモ

表 2　UML 図と CSP 記述の対応関係

UML 図	CSP 記述
ステートマシン図	状態遷移を表すプロセスになる．CSTM 名，状態名，イベント名，アクション名を用いる
シーケンス図	アクションを表すプロセスになる．アクション名，イベント名，イベント送信先 CSTM 名を用いる
オブジェクト図	CSTM のインスタンスの記述に使われる．CSTM 名とオブジェクト名を用いる

ジュール振舞いモデルと配置モデルを CSP により表現する．実装アーキテクチャにおける UML 図と CSP 記述の対応関係を**表 2** に示す．

4.4.1　状態遷移機械の表現方法

ステートマシン図における状態遷移を CSP のプロセスで表現し，イベントやアクションの開始は CSP のイベントで表現する．このとき，要件 (3), (4) から次のように CSP での記述を考えた．

- アクションの開始と終了をイベントで表現する
- インスタンスを変数で表現する

アクションの開始と終了をイベントで表現することで，要件 (4) のアクション終了後に次の状態に遷移することを表現できる．アクションの開始のみを CSP のイベントで表現した場合，そのイベントによってアクションを表すプロセスが起動されるが，アクションのプロセスの終了を待たずに，状態遷移のプロセスで次の状態への遷移が行われることがある．アクションの開始と終了をイベントで表現することで，アクション終了のイベントが起きた後で次の状態への遷移を記述することができる (**図 6**).

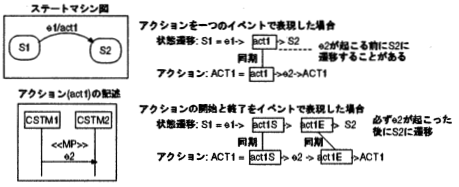

図 6　アクションの開始と終了のイベント

要件 (3) は CSTM の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でも，同じプロセスで表現することができる．変数を使わずに，名前の一部を変更したプロセス (例えば，CSTM1_instance1, CSTM1_instance2) を複数作成することも考えられるが，CSP の記述が繁雑になる．

CSP 記述におけるプロセス名，イベント名を一意にするために次の命名規則を用いる．

- 状態を表すプロセス名
  CSTM 名_状態名 (変数)
- 受信イベントを表すイベント名
  CSTM 名. 変数.deq. イベント名
- 遷移時のアクション開始を表すイベント名
  CSTM 名. アクション名.start
- 遷移時のアクション終了を表すイベント名
  CSTM 名. アクション名.end

初期状態は，

CSTM 名_InitState = 初期状態名

で表現する．

ステートマシン図と CSP 記述の対応関係を**図 7** に示す．

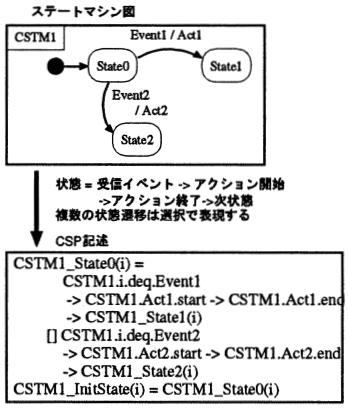

図 7　ステートマシン図と CSP 記述の対応

4.4.2　アクションの表現方法

E-AoSAS++では状態遷移のアクションはシーケンス図で表される．シーケンス図で記述された CSTM 間のイベントは CSP のイベントとして表現する．要件 (1), (3), (4) から次のように CSP での記述を考えた．

(a)　アクションの開始と終了をイベントで表現する
(b)　インスタンスを変数で表現する
(c)　IAD によるイベント送信では，全てのイベント送信の組合せを記述する

(a),(b) については 4.4.1 節で述べた．IAD によるイベント送信は，イベントの送信順序が非決定的であるので，起こりうるイベントの順序の全ての組み合せを CSP で表し，それらの選択として表現する．

CSP 記述におけるプロセス名，イベント名を一意にするために次の命名規則を用いる．

- アクションを表現したプロセス名

CSTM 名_アクション名 (変数)

- 送信イベントを表現したイベント名
  CSTM 名. 変数.enq. イベント名

シーケンス図と CSP の対応関係の例を**図 8** で示す. 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 間のイベント送信において, 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.

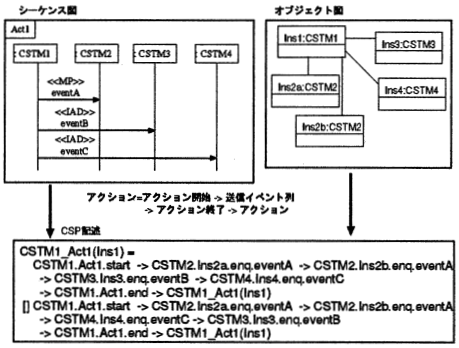

図 8 アクションを表現した CSP のプロセスの例

4.4.3 キューの表現方法

E-AoSAS++では CSTM の通信にはキューを用いる. 要件 (2) より, キューも CSP のプロセスとして記述する. キューの構造は CSTM に依らず同一であるが, CSTM 毎のキューを区別するために, CSTM ごとに固有の名前をつけてキューのプロセスを表す. 要件 (3) はステートマシン図やシーケンス図の場合と同様に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.

キューは次の規則にしたがって記述される.

- インスタンス名は変数 (i) で表現し, CSTM 単位でキューのプロセスを記述する
- キューが保持するイベントは変数 (x) を用い表現する. キューの保持するイベントの最大個数はあらかじめ決めておき, その個数だけ変数を用いる.
- nulle は, イベントを保持していないことを表現する
- キューに挿入するイベントの名前には enq をつけ, 取得するイベントの名前には deq をつける

上記の表現方法を基にキューを CSP で表現すると, **図 9** のような CSP 記述となる. 図 9 では要素数 2 個のキューを表現しており, 変数 x1, x2 がキューが保持しているイベントを表す.

```
CSTM1_Queue(i,x1,x2) =
   if (x1 == nulle) then
       -- キューの先頭が空なので先頭に要素を格納
       CSTM1.i.enq?x -> CSTM1_Queue(i,x,nulle)
   else
       -- キューの先頭が空でないので先頭の要素を取得
       CSTM1.i.deq!x1 -> CSTM1_Queue(i,x2,nulle)
       [] (CSTM1.i.enq?x -> (
         -- キューの 2 番目が空ならば要素を格納
         if (x2 == nulle) then CSTM1_Queue(i,x1,x)
         else CSTM1_Queue(i,x1,x2)))
```

図 9 キューを表現した CSP のプロセスの例

4.4.4 CSTM 全体の表現方法

各 CSTM の状態遷移を表現したプロセス, アクションを表現したプロセス, キューを表現したプロセスを同期させて, CSTM を表現するプロセスを作成する. モジュール配置モデルであるオブジェクト図から各 CSTM のインスタンスの情報を取得し, 変数を用いて対応するプロセスを記述する.

4.5 実装アーキテクチャ記述における問題の検出方法

本節では, 4.4 節で説明した CSP 記述での表現方法を基に, 4.2 節で挙げた問題をデッドロックやライブロックに帰着させて検査を行う方法を提案する.

4.5.1 受理できないイベントを受け取る問題の検査方法

CSTM が受理できないイベントを受け取ったことを検出する方法として, 作為的にデッドロックを起こす方法を考えた. 受理できないイベントを受け取った CSTM はそれ以上遷移を行えず, イベントを生起しないのでライブロックとして検出することはできない. 検査対象内の全てのプロセスが状態遷移を行えなくなったとき, デッドロックとして検出する.

すべての CSTM を停止させるイベント error を作り, CSTM が受理できないイベントを受け取ったときに, error イベントを発生させて, すべての CSTM を停止させて, 作為的にデッドロックを起こす. 具体的な手順は次の通りである.

1. error イベントによって停止する状態遷移の追加
   1-1. 各 CSTM に強制終了状態 END を追加する. END 状態から他の状態への遷移はなく, CSTM は END 状態になるとそれ以上遷移が行えない.
   1-2. 各 CSTM の各状態から error イベントによって END 状態へ遷移する, 状態遷移を追加する.
2. 受理できないイベントを受け取って error イベントを発生する状態遷移の追加
   2-1. 各 CSTM の各状態から, 受理できないイベントを受け取って END 状態へ遷移する, 状態遷移を追加する.

2-2. 2-1 の状態遷移のアクションとして，他の CSTM に error イベントを送信するアクション ActErr を追加する．

図 4 の CSTM_C に，デッドロックと作為的に起すための状態遷移とアクションを追加した例を**図 10** に示す．

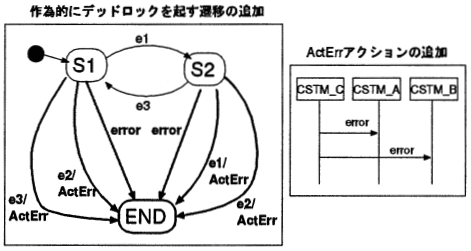

図 10 受理できないイベントを受け取る問題の検査方法の例

4.5.2 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の検出方法として，ライブロックとして検出する方法を考えた．遷移しない状態へのイベントが生起しないので，あるイベントをトリガにして作為的にデッドロックを起すことはできない．CSP では内部イベントが永久に続く場合をライブロックとして扱う．生起しない外部イベントと遷移しない状態を対応づけることができると考えた．

必ず起こらなければならない状態遷移のトリガとなるイベントを外部イベントとして，ライブロックの検査をおこなうことで，この外部イベントが生起するかどうかを検査することができる．しかし，この外部イベントは別の状態遷移で起こる可能性もあるので，この検査方法だけでは十分ではない．本研究では，必ず起こらなければならない状態遷移に新たに検査用のイベント (以下，ラベルと呼ぶ) を追加し，このラベルを外部イベントとしてライブロックの検査を行う．

図 11 に検査方法の例を挙げる．図 5 の状態 S33 への遷移を検査したい場合，S33 への遷移を起すイベント evC にラベルをつける．このラベルは S31 のイベント evC と区別するために使われる．

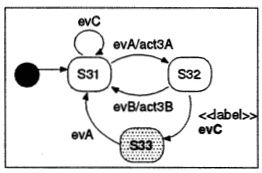

図 11 遷移されない状態が存在する検査方法の例

4.6 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 図から CSP 記述を自動生成するツールを試作した．試作したツールでは，EnterpriseArchitect で作成された UML 図を入力として，CSP 記述を出力する．ツールは約 1400 行の Java コードで実現した．

5. 考　　察

5.1 事例検証

4 節で述べた提案方法で，アーキテクチャ記述の誤りが発見できることを事例を挙げて示す．

図 12 に 2 個のスイッチで 1 個のライトを On/Off するシステムの例を示す．Switch は押される (push) と Switches にイベント push を送る．Switches は 2 個の Switch を管理しており，on,off の状態を持つ．Switches は自状態に合わせて，Light に lightOn, lightOff のイベントを送る．なお，Light の broken 状態は決して遷移されない状態である．

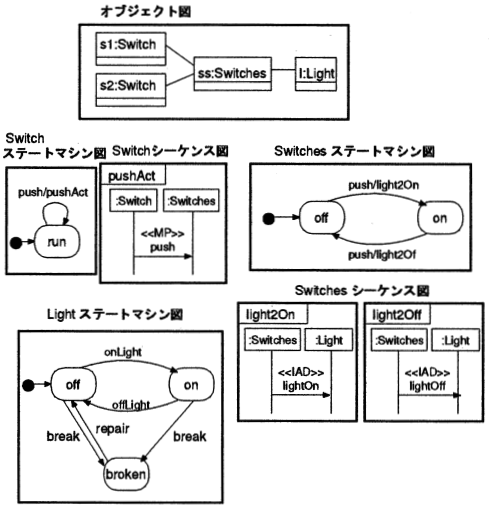

図 12 事例: 2 個のスイッチと 1 個のライト

この UML 図から，CSP 記述を生成して，FDR によりデッドロックの検査をしたところ，デッドロックは検出されなかった． Light のステートマシン図の onLight イベントにラベルをつけて検査をした場合は，ライブロックは検出されなかった．break イベントにつけて検査した場合は，ライブロックが検出された．

図 13 に図 12 の Swtiches のシーケンス図を誤って記述した場合を示す．アクション light2On で lightOff イベントを，アクション light2Off で lightOn イベントを Light に送信している．この UML 図から，CSP 記述を生成して，検査をしたところデッドロックとして検出された．

以上より，アーキテクチャ記述の誤りを，デッドロックやライブロックとして検出できることを確認した．

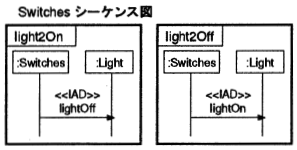

図 13 誤りのある Switches シーケンス図

5.2 関連研究との比較

モデル検査をアスペクト指向ソフトウェアに応用する研究として文献[8]が挙げられる．文献[8]では AspectJ で記述されたプログラムコードに対して，表明方式でモデル検査を適用する方法を示し，そのためのフレームワークを提案している．検査フレームワークはアスペクト指向で実現され，検査したい性質をアスペクトとして記述する．文献[8]は実装時において，アスペクトが織り込まれたソフトウェアを検査する方法である．文献[8]でも述べられているように，アスペクト指向設計を支援する環境は未だ十分に整備されているとは言えず，設計時にモデル検査を適用することは難しい．

これに対し，本研究ではアスペクト指向設計時において検査を行う方法を提案した．われわれは組込みソフトウェアを並行状態遷移機械の集合として規定するアーキテクチャスタイル E-AoSAS++を提案し，それに基づく開発プロセスや開発環境を整備してきた．これらを利用し，E-AoSAS++におけるアーキテクチャの意味を CSP で記述することで，アスペクト指向設計に対する実行前検査が可能となったと考えられる．

6. おわりに

本研究では E-AoSAS++ に基づく実装アーキテクチャと CSP 記述との対応づけを行い，実行前検査の方法を提案した．UML 図から CSP 記述への自動変換ツールを試作し，簡単な例を用いて，提案方法で E-AoSAS++におけるアーキテクチャ記述の誤りを発見できることを示した．

今後の課題を次に示す．

- 実用的なソフトウェアに対する検査の適用
  実際の組込みソフトウェアなどのより大きなシステムに対しして提案方法を適用し，有効性について考察する必要がある．AspectJ のようにアドバイス機構を用いるプログラミング言語では同一の合流点に複数のアドバイスが存在した場合に，アドバイスの実行順序によっては開発者の意図しない結果となることがある．我々の方法ではこの問題は IAD によるイベント送信において生じると考えられる．全てのイベント送信の組合せを記述することで，この問題を検出できると考えるが，確認が必要である．
- モジュール間通信モデルを用いた検査
  E-AoSAS++の実装アーキテクチャでは，システム全体の振舞をモジュール間でのメッセージのやりとりとしてモジュール間通信モデルとして記述する．CSP の軌跡を用いて，各 CSTM の振舞いとモジュール間通信モデルとの整合性の検査を行うことを現在考えている．
- 検査結果を UML 図へフィードバックするツールの設計・実現
  FDR の検査結果と UML 図の対応関係を考察し，誤り箇所を UML 図上で提示するツールを作成することで，CSP や FDR について習熟していなくても UML 図における記述の誤りを特定することができる．

謝 辞 本研究の一部は，平成 20 年度科学研究費補助金（基盤研究 (C)（一般）19500028）および平成 20 年度南山大学パッヘ奨励金 I-A-2 の助成による．

参 考 文 献

1) Elrad, T. and et al: Aspect-Oriented Programming, *CACM*, Vol. 44, No. 10, pp. 28–97 (2001).
2) Noro, M., Sawada, A., Hachisu, Y. and Banno, M.: E-AoSAS++ and its Software Development Environment, *Proceedings of the 14th Asia-Pacific Software Engineering Conference(APSEC 2007)*, pp. 206–213 (2007).
3) Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R. and Stafford, J.: *Documenting Software Architectures: Views and Beyond*, Addison-Wesley (2002).
4) Jr., E. M. C., Grumberg, O. and Peled, D. A.: *Model Checking*, The MIT Press (1997).
5) Hoare, C. A. R.: *Communicating Sequential Processes*, Prentice-Hall (1985).
6) Ltd, F. S. E.: *FDR2*. http://www.fsel.com/.
7) Holzmann, G. J.: The Model Checker SPIN, *IEEE Transaction on Software Engineering*, Vol. 23, No. 5, pp. 279–295 (1997).
8) 尚靖鵜林, 哲雄玉井: 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).