E-AoSAS++ における例外処理記述と検証に関する研究

2009SE319 横田純也　2010SE243 冨田慶悟
指導教員：張漢明

1　はじめに

組込みソフトウェアのソフトウェア開発では，システムを停止することなく安全に動作させるために，例外処理の記述と検証が重要である．しかしプログラムの記述において例外処理の記述は膨大なものになりつつあり，プログラムレベルでの例外処理の検証は困難である．南山大学の野呂・沢田・張研究室では，E-AoSAS++(Aspect-oriented Software Architecture Style for Embedded software)[5] を用いたアスペクト指向ソフトウェアアーキテクチャに関する研究をおこなっている．E-AoSAS++ では，アスペクト指向技術における横断的関心事をモジュール化して記述することができる．本研究では，例外処理を横断的関心事として，モジュール化して記述と検証することを考える．

本研究の目的はアーキテクチャ記述における E-AoSAS++ を用いた例外処理の記述と検証の実用化である．研究課題は，E-AoSAS++ において例外処理をアスペクトとして分離した際の妥当性と，検証の容易性の確認することである．

本研究の基本的なアイデアは，例外処理の事例検証をおこない，例外処理の妥当性の分析および系統的な検証手順を提示することである．代表的な例外処理として，時間監視と故障監視の記述と検証をおこなうことを考える．系統的な検証手順は，本研究室で提案している「同時性を考慮した振舞いの検証手法」[3] の適用を試みる．検証にはモデル検査ツールである FDR[2] を用いた．
本稿では，事例として自動販売機システムに例外処理の記述を追加し，系統的な検証をおこなった．

2　背景技術

2.1　E-AoSAS++

組込みソフトウェアのためのアスペクト指向アーキテクチャスタイル (以後，E-AoSAS++) は，組込みソフトウェアのアーキテクチャを構築するための，系統的な記述方法を規定したものである．E-AoSAS++ における計算モデルは，状態遷移機械がイベントを受理し，対応したアクションを実行することにより振舞いを表現する．E-AoSAS++ では，横断的関心事のモジュール化をすることが可能であり，UML を用いた図式表現で記述される．図式表現には，以下の要素が存在する．

- View
  - Aspect
  - IAD
    - AspectCoodinator
    - WeavingPolicy

View はアスペクトを分割する際の関心事で対象となる複数の Aspect と，その Aspect 間を横断する関心事を記述する IAD が存在する．例外処理に複数の項目が存在する場合，複数の例外処理を一つの View に記述することができる．IAD には非機能処理が記述される AspectCoodinator と，結合点が記述される WeavingPolicy が存在する．WeavingPolicy で記述された特定のイベントが受理されることにより，AspectCoordinator の処理が開始する．WeavingPolicy は結合点の数により複数存在し，WeavingPolicy に対応する AspectCoordinator が 1 つ存在する．本研究では，組込みソフトウェアの横断的関心事として，例外処理をモジュール化している．

2.2　CSP

CSP[1] とは，並行システムにおける相互作用のパターンを記述する形式仕様言語である．組み込みソフトウェアの FDR における検証では元のシステムと例外処理のイベント送受信を振舞いとして記述する．CSP では Refinement を調べることができる．Refinement は与えられた仕様と，システムの振舞いの詳細化関係を調べている．

2.3　FDR

本研究での検証にはモデル検査を採用した．モデル検査器には FDR を使用し，入力には CSP を利用する．モデル検査が検証できる性質には安全性と活性，ライブロックがある．それぞれトレースモデル，安定失敗モデル，失敗/発散モデルを調べることにより検証が可能である．これらの性質を用いて検証することで，並行処理特有の問題であるデットロック等を事前に検出することが可能である．

2.3.1　検証

対象のシステムが外部仕様を満たしている検証をおこなうには，以下の検証が成り立てばよい．

外部仕様 = システム\{外部仕様以外のイベント}

CSP で検証をおこなうには外部仕様と対象のシステムが双方に詳細化関係が成り立てばよい．

3　例外処理の記述方法と検証方法

例外処理を合成する前のシステムと例外処理をアスペクトとして分離し，アスペクト間のメッセージのやり取りを IAD に記述する．また例外の処理の振舞いは AspectCoodinator に状態遷移図として記述される．WeavingPolicy では，元のシステムと例外処理で特定のイベントを受理した場合の例外処理の開始と検知することを状態遷移図を用いて表現する．

3.1 例外処理

例外処理 [4] は，プログラムが実行中の場合に想定されていない入力などが与えられた場合に実行される処理である．例外に対し適切な処理がおこなえなかった場合はより致命的な異常動作を続けてしまう可能性がある．
本研究において例外の仕様としては以下に分類する．

- システムを初期状態へ戻し，継続する
- システムをすべて止めて停止させる

例外よってはシステムを継続させることができる場合もある．例外の影響が少なく，システムを初期の状態へ戻すことでリセットし，通常通りシステムの継続を維持できる場合には初期状態へ戻す処理のみを行うことでなるべく例外が発生した場合にシステムへ与える影響を減らすことが重要だと考える．発生した例外の影響が非常に大きくシステムの継続が不可能だと考えられる場合，速やかにシステムを停止させなければならない．

3.2 記述の方法

E-AoSAS++ におけるアーキテクチャ記述を行う際に分類される記述のパターンを提示する．例外処理の記述に必要な要素として，例外処理の開始，終了，処理，検知がある．この 4 つの要素を IAD にて記述をおこなう．

3.2.1 例外の開始

IAD における記述の要素に開始がある．例外処理の開始を分離して記述する為に元のシステムの起点となるイベントを元のシステムから決定する．例外が発生した時にシステムから送信されてくる開始イベントと状態を図 2 のように状態と状態間の遷移に記述されるイベントを分離して記述し，AspectCoordinator に開始されたことを通知するアクションを記述する．例外処理の開始イベントを例外処理が受理することにより例外処理が開始される．

図 1 例外処理の開始

3.2.2 例外の終了

終了も開始と同様に，元のシステムの起点となるイベントを決定し，処理が終了した際の終了イベントと状態を図 **??**のように記述し，Aspectcoordinator に開始されたことを通知するアクションを記述する．

図 2 例外処理の終了

3.2.3 例外の処理

例外が発生した場合に行われる例外に対応した処理を例外の処理とする．例外の開始が発生したというイベントを受理した後に AspectCoordinator にて実行されるアクションを例外の処理全体を図 3 のように記述する．3 章 1 節で説明した仕様により，システムを通常処理状態へ戻す処理を行う．

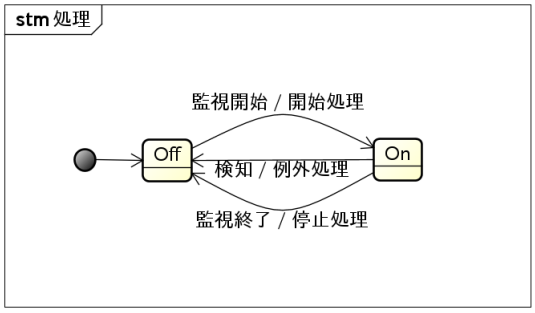

図 3 例外処理の処理

3.2.4 例外の検知

例外の処理が終了した後，処理が完了したというメッセージをシステムに例外の検知イベントとして例外処理から送信される．例外の検知を WeavingPolicy にて分離して記述する場合には例外処理に存在する状態と状態間の遷移を WeavingPolicy に記述し，処理が完了したというイベントに対応するアクションとして AspectCoordinator に通知するアクションを図 4 のように記述する．

3.3 検証項目の作成

E-AoSAS++ における例外処理記述を検証することを考える．例外処理が合成される前の基本システムおよび例外処理のシステムを分け，本研究が提案する検証手順を説明する．
本研究では以下の 2 つの検証方法を考えた．

- 基本システムに例外処理を織り込んでも基本システムの仕様に影響を与えない
- 例外が発生した場合に正しく処理がおこなわれている

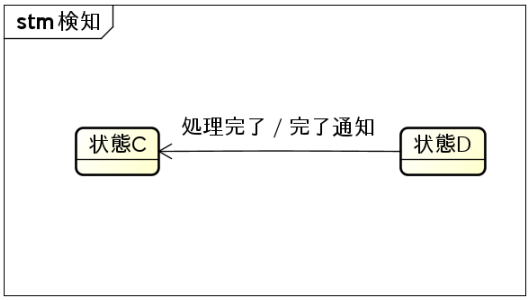

図 4 例外処理の検知

以上の検証をおこなうことにより例外処理の検証が可能である．

3.4 CSP 記述

基本システムに例外処理を織り込んでも基本システムの仕様に影響を与えないことの検証と，例外が発生した場合に正しく処理がおこなわれていることの検証は 2 段階で以下の制約式を用いておこなわれる．

SEQ_SECTION(s) = []event:s @ event -> SKIP

PL_SECTION(occured, not_occured) = (||| (st, end, ev)+occured @ st -> ev -> end -> SKIP) ||| (||| (st, end, ev):not_occued @ st -> end -> SKIP)

4 事例：自動販売機

本章では，前章で書かれた記述方法と検証方法を単純な自動販売機を例に適用する．自動販売機の振舞いは以下の通りである．

- コインを投入した後，ボタン押すと商品がでる．
- コインを投入した後，レバーを引くとコインが返金される．

単純な自動販売機ではボタンは一つで商品に種類が無く，コインに金額の種類も無い．コインの後にユーザーが行えるのはボタンを押すかレバーを引くことの 2 種類とする．この単純な自動販売機システムに例外処理として以下の例外処理の合成をおこなう．

- 時間監視：コインを投入した後，ボタンが押されない場合は返金されて待機状態に戻る．
- 故障監視：Controller で異常が発生した場合は動作を停止する．

以上の例外処理をそれぞれ記述する．

4.1 時間監視

時間監視をおこなうために，元のシステムに存在するコンポーネントとは別に時間をカウントするコンポーネントとして Timer を用意する．元のシステまずは通常処理をおこないかつ，時間が監視される Aspect として Terget を用意する．Terget には Model である VM が存在する．また時間を監視する Aspect として Timer を記述する．そして二つの Aspect 間のメッセージのやり取りをボタンの時間を監視する IAD を記述する．

4.1.1 WeavingPolicy

WeavingPolicy では例外処理の開始，検知，終了を記述する．Timer では元のシステムから時間監視をスタートさせる start イベントを受理し，timeout を発生した場合はムへ timeout というイベントを図 5 のように送る，timeout が発生する前に元のシステムから時間監視のキャンセルをさせる end イベントを受理した場合には時間監視を図 6 のように停止する記述をする．

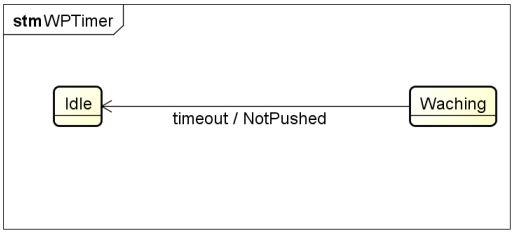

図 5 Timer の WeavingPolicy

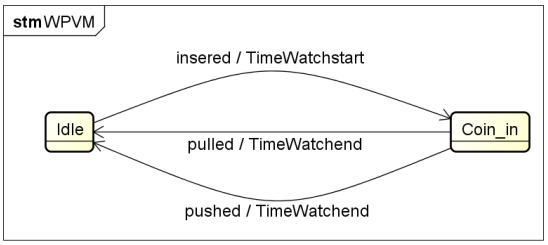

図 6 VM の WeavingPolicy

4.1.2 AspectCoodinator

WeavingPolicy から送信されてきたイベントを受理し，固有の処理をおこなう．時間監視では，時間監視の開始と，通常処理をおこなった場合の時間監視終了と，ボタンが押されなかった場合の返金のアクションが図 7 のように記述される．

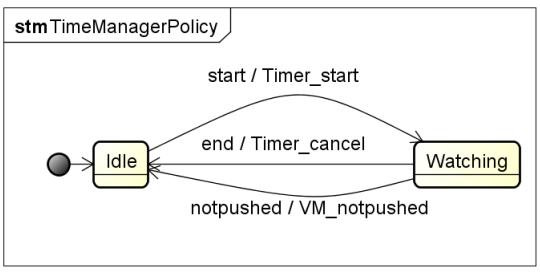

図 7 時間監視の AspectCoodinator

4.2 検証

- 基本仕様の検証自走販売機の基本仕様としては，コインが投入された後にボタンが押されると商品が出る，またはレバーが引かれるとコインが返却されるのどちらかの繰り返しである．システムに例外処理を合成した後に着目するイベントである外部イベント以外の振る舞いを隠ぺいした場合に仕様を満たしているかを検証する．基本仕様での検証は以下の式でおこなわれる．

  SPEC [= (SYSTEM||EXCEPTION)
  \(EVENTS-{coin, button, item, lever, return})

- 例外が発生した場合自動販売機の例外仕様では時間切れが発生した場合にコインの返却をおこなうというものである．基本仕様と同様に着目するイベントを外部イベントの timeout としてすべてのイベントから timeout 以外を隠ぺいした場合の検証をおこなう．例外仕様の検証は以下の式となる．

  SPEC [= (SYSTEM||EXCEPTION)
  \(Events - {timeout，return})

4.3 故障監視

故障監視では，MVC 分割でのコントローラーのコンポーネントが自身を監視する．故障を検知した場合は，金をおこなってからシステムを停止することを考えるが，返金箇所に異常がある場合は返金を行うことができない．振舞いの記述は，通常処理を行い故障が監視する Target とそれぞれの Controller である Hardware を Aspect として分離する．その返アスペクト間のメッセージのやりとりを IAD として HardwareError に記述する．

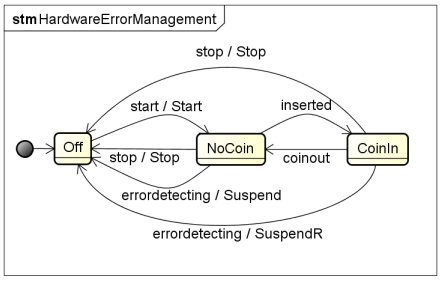

図 8 故障監視の AspectCoodinator

図 8 は，故障監視の AspectCoodinator である．時間監視での TimeManagerPolicy では実行状態が active 状態な状態は Watching 状態のみであったが，故障監視では，返金が可能である状態と可能でない状態の 2 つの状態が必要である．

5 考察

5.1 故障監視の記述

故障が発生した場合にはシステムを停止させる必要があるが，停止を記述には停止状態とその遷移を追加する必要がある．これらの停止状態やイベントは，動作モデルに存在するメタな状態やイベントを利用することで表現できる．

5.2 故障監視の検証

故障監視では自動販売機の故障した箇所と状態によって仕様が決まり，時間監視と同様に基本仕様と例外仕様の検証を段階的に検証をおこなうことができる．1 段階目の検証式は時間監視と同様の記述を用いればよい．2 段階目は以下のようになる．

SPEC [= (SYSTEM||EXCEPTION)
\(Events - {error，return，sleep})

5.3 例外処理の複数発生

例外処理を複数記述した場合は例外処理が複数発生することが考えられる．元のシステムに複数の例外処理を合成し，本研究室で提案されている同時性の段階的検証方法を用いて優先度の概念を導入することにより，例外処理の複数発生の検証も可能であると考える．

6 おわりに

本研究は E-AoSAS++ を利用し，例外処理の記述と検証について整理し，例外処理記述のフレームワークを提案した. 今後の課題として，同時性の検証の実現があげられる．

参考文献

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

[2] Formal Systems (Europe), “Formal Systems (Europe) Ltd,” http://www.fsel.com/, 2010.

[3] 井上 啓佑，寺西 祐斗，友松 良輔“同時性を考慮したアーキテクチャ検証手法に関する研究”南山大学卒業論文，2015.

[4] 片山 卓也，土井 範久，鳥居 宏次，監訳“ソフトウェア工学大辞典”ISBN:4-254-12123-7，pp1038，朝倉書店，1998.

[5] M.Noro，A.Sawada，Y.Hachisu，M.Banno “E-AoSAS++ and its Software Development Environment”，Proceedings of the 14th Asia-Pacific Software Engineering Conference (APSEC2007), pp.206-213，Dec．2007.