組込みソフトウェアを対象としたコード自動生成に関する研究
－意味制約を用いたコード生成ツールの高機能化－

2010SE037 羽根あずさ　2010SE158 野々村友花
指導教員： 沢田篤史

1　はじめに

近年，ハードウェアの高性能化に伴い，組込みソフトウェアの構造は複雑化し，その規模は拡大している．その一方で，開発工程の短縮や生産性の向上が課題となっている．組込みソフトウェアにおいて，並行に動作するハードウェアを状態遷移機械でモデル化することが一般的に行なわれており，本研究室ではその考え方に基づき，組込みシステム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（以下，E-AoSAS++）を提案している [1]．E-AoSAS++ では，組込みソフトウェアを並行に動作するミーリ型の状態遷移機械の集合と規定している．ミーリ型は，イベント入力部分に動作（アクション）を記述する型であり，イベントが入力されるとアクションを実行し，状態が遷移する．E-AoSAS++ を用いた開発において，コード生成を行なうためにモデル駆動型アーキテクチャ（以下，MDA）[2] の概念を用いたコード生成ツールが実現されている．アクションのプログラムコードの自動生成においては，イベントの発生列に関するメッセージ送信コードの自動生成しか行なわれていない．アプリケーションごとに異なる要求が存在するので，イベントの発生に関連するプログラムコード以外を完全に自動生成することは難しい問題である．

本研究の目的は，アクションの例外処理コードのパターンを定義することで，例外処理関連コードを自動生成する可能性を確認することである．例外処理を記述する上で必要不可欠である記述を意味制約とし，例外処理の意味制約のパターン化を行なった．一般に組込みシステムのソフトウェアで例外処理は重要な要素であり，アクションコード内に多数存在する．例外処理関連コードは定型的に記述されることが多いので，我々はパターンを考えることによって自動生成可能になると考えた．

我々は，例外処理のパターン定義を行なうために，Java の例外処理構文に着目した．構文上の例外発生部，受取る例外，例外処理部の三つの要素に着目し，それぞれについて整理した．整理するために，我々が開発した自律走行ロボットの制御ソフトウェアと Java で記述された Apache の前文検索用オープンソースコードを調査した．その結果，それぞれの要素についていくつかのパターンができることを確認し，定義した．例外処理の自動生成に必要な情報を付加するために，コード生成ツールの入力となるアーキテクチャ記述の拡張を行なった．定義したパターンの情報を記述するために，アーキテクチャに記述するさいのパターンの構文定義を行なった．必要な情報を記述するためのテンプレートを作成した．提案したパターンを用いてアクションにおける例外処理のコード生成が可能であるかを，自律走行ロボットの制御ソフトウェアの開発を通して検証した．成果として，アクションの例外処理コードのパターンを定義することで，例外処理関連コードの自動生成の可能性を確認することができた．

2　背景技術

2.1　モデル駆動型アーキテクチャ (MDA)

MDA はモデルを中心としたソフトウェア開発の手法であり，プラットフォームに依存しないソフトウェア開発を目的とする．MDA ではプラットフォームに独立モデルである PIM の設計を行い，プラットフォームに依存したモデルである PSM への変換を行なう．複数のプラットフォームに対応したプログラムコードの自動生成が可能となる．

2.2　E-AoSAS++

2.2.1　概要

E-AoSAS++ は，本研究室で提案されている組込みシステム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である．E-AoSAS++ は，組込みシステムのソフトウェアアーキテクチャを構築するためのコンポーネント・コネクタスタイルに基づく枠組みである．アプリケーション論理を実装するオブジェクトと，その状態遷移アスペクトを実現する並行状態遷移機械（以下，CSTM）を基本的な構成要素とする．CSTM は受理したイベントに応じて状態を遷移し，状態遷移時に他の CSTM にイベントを通知する．この各 CSTM の協調動作によって，組込みシステムの機能を実現する．E-AoSAS++ に基づくアーキテクチャ記述は UML を用いる．

2.2.2　コード生成ツール

E-AoSAS++ に基づく開発支援環境の一つに，MDA の概念を用いたコード生成ツールがある [4]．コード生成ツールでは，UML を拡張した E-AoSAS++ のアーキテクチャ記述を入力とする．PIM，PSM の 2 段階のモデル変換を行なうことで，各プラットフォームに対応したプログラムコードを自動生成することが可能である．コード生成ツールは開発期間の短縮化を実現している．コード生成ツールによって生成されるコードは，CSTM の並行処理，状態遷移，CSTM のインスタンス処理について実現するコードと，CSTM の状態遷移に伴うアクションを記述するためのテンプレートコードである．

3　例外処理パターンの定義

アクションのプログラムコードを完全に自動生成することは，アプリケーションごとに異なる要求が存在することから不可能である．特定の目的のために書かれるプログラムコードの書き方には，一定の規則性が存在する場合がある．一定の規則性をパターンとして定義することができれば，それに基づきコードを生成することが可能になる．我々の組込みシステムに関する開発経験から，例外処理は定型的に行なわれることが多く，例外処理をパターン化することで自動生成による省力化が望めることに着目した．

3.1　例外処理の整理

例外処理関連コードを自動生成する可能性を確認するために，例外処理のコードを定義する．Java の例外処理構文を調査し，いくかのパターンに定型化した．一般的に例外処理記述は例外発生部分と発生した例外を処理する部分に分かれている．発生した例外を処理する部分は，受取る例外を列挙した部分と例外に対する処理内容の部分に分かれる．我々は，この構文に注目して，構文上の例外発生部，受取る例外，例外処理部の三つの要素に着目する．三つの要素についていくつかのパターンができることを確認できた．三つの要素から分類することで全ての例外処理記述を try-catch 文の形で記述できることがわかった．それらは我々の組込みシステムに関する開発経験と，Java で記述されたオープンソースコードの Apache の前文検索用ソースコードを調査した結果得られた．三つの要素の関係を図 1 に示す．

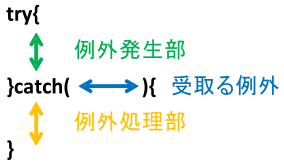

図 1　三つの要素の関係

例外発生部は try 節の中に発生させたい例外を記述する要素である．受取る例外は発生しうる例外を受取る例外名を記入する要素である．例外処理部は発生した例外に対する処理を記述する要素である．各要素における記述内容をパターンとし，そのパターンに対してコード片を適切に対応付けることができれば，例外処理関連コードを自動生成できると考える．各要素の記述内容を表 1 にまとめる．

表 1　三つの要素と例外処理記述の関係

	例外発生部	受取る例外	例外処理部
場所	try 節	catch 節	
記述内容	例外を発生させる	例外が一つ	再通知
	例外処理記述	例外が複数	その他
	その他		

あらゆる例外処理記述は，三つの要素と各要素における記述内容の組合せで記述することができる．

3.2　例外処理のパターンの定義

パターンは着目する要素ごとに存在する．例外発生部には三つのパターン，受取る例外には二つのパターン，例外処理部には二つのパターンを定義することができる．各要素のパターンの種類は，表 1 の記述内容である．

これらの各パターンを組合わせることで，try-catch 構文で記述された全ての例外処理記述を記述することができる．我々が調査したプログラムコードにおいては，提案したパターンで全ての例外処理記述を記述することができた．パターンの中には，組合わせても意味のないものも存在する．例外発生部のその他のパターンは，事前事後条件検査，例外解釈のパターンと組合わせても生成されるソースコードが事前事後条件検査，例外解釈だけのパターンと同一であり，意味のない組合わせとなる．

3.2.1　例外発生部のパターンの分類

例外発生部は三つのパターンを定義することができる．三つのパターンを次に示す．

例外発生部のパターン

パターン 1　事前・事後条件検査

```
try{
    if (条件) throw new Exception 名 ();
}
```

パターン 2　例外解釈

```
try{
    try{
    } catch(Exception 名 e) {
    }
}
```

パターン 3　その他

```
try{

}
```

パターン 1 の事前・事後条件検査とは，事前条件や事後条件を if 文記述し，事前条件や事後条件に対応させる例外を発生させるものである．パターン 2 の例外解釈とは，try-catch 文で記述し，例外処理をよみかえて新たな例外を発生させるものである．

3.2.2　受取る例外のパターンの分類

受取る例外は二つのパターンを定義することができる．二つのパターンを次に示す．

受取る例外のパターン

パターン 1　単一例外
　catch(Exception 名 e)
パターン 2　複数例外
　catch(Exception 名 1 | Exception 名 2 e)

パターン 1 の単一例外とは，例外が発生する箇所で発生した一つの例外を受取るものである．パターン 2 の複数例外とは，例外が発生する箇所で発生した例外を同じ例外ハンドラで複数受取るものである．

3.2.3　例外処理部のパターンの分類

例外処理部は二つのパターンを定義することができる．二つのパターンを次に示す．

例外処理部のパターン

パターン 1　例外統合
　{
　throw new Exception 名 ();
　}
パターン 2　その他
　{

　}

パターン 1 の例外統合とは，複数の例外を一つにし，メッセージ送信元にプロパゲートするものである．

4　アーキテクチャ記述法の拡張

シーケンス図に 3.2 節で定義したパターンを付加することができれば，コードの一部が自動生成可能になると考えた．例外処理の自動生成に必要な情報を付加するために，コード生成ツールの入力となるアーキテクチャ記述の拡張を行なった．

4.1　パターンの構文定義

3.2 節で定義したパターンをアーキテクチャに記述する規則として，バッカス・ナウア記法 (以下，BNF) を用いてパターンの記述規則の定義を行なった．各パターンの記述方法を BNF で表現する．
例外発生部のパターンを式 (1) に示す．

$$\begin{aligned} \text{事前・事後条件検査} ::= & \text{“}if\text{”“(条件)”} \\ & \text{“}Exception\ \text{名”“;”} \\ \text{例外解釈} ::= & \text{“}try\text{”“}Exception\ \text{名”“;”} \\ \text{その他} ::= & \text{“;”} \end{aligned} \tag{1}$$

受取る例外のパターンを式 (2) に示す．

$$\begin{aligned} \text{単一例外} ::= & \text{“}Exception\ \text{名”“;”} \\ \text{複数例外} ::= & \text{“}merge\text{”“}Exception\ \text{名”} \\ & \text{“,”” ...”“;”} \end{aligned} \tag{2}$$

例外処理部のパターンを式 (3) に示す．

$$\begin{aligned} \text{例外統合} ::= & \text{“}throw\text{”“}Exception\ \text{名”“;”} \\ \text{その他} ::= & \text{“;”} \end{aligned} \tag{3}$$

4.2　アーキテクチャ記述の拡張方法

E-AoSAS++ の開発体系に提供されているコード生成ツールのアクション部分は，シーケンス図を基に自動生成されている．我々はシーケンス図に 4.1 節で定義した構文を付加する．シーケンス図に例外処理に関する情報を記述するためのテンプレートを作成する．

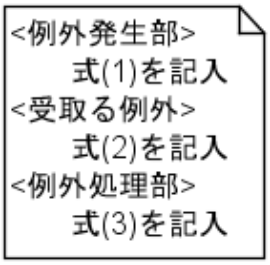

図 2　テンプレート

図 2 は我々が用意したテンプレートである．式 (1)(2)(3) を用いて生成させたい例外処理記述を図 2 のテンプレートに記述する．テンプレートの < 例外発生部 > に式 (1)，< 受取る例外 > には式 (2)，< 例外処理部 > には式 (3) を記述をする．複数の記述が必要な場合は続けて記述する．

例えば IOException を受取るか，ArgumentNullException と ArgumentExcepton を同じハンドラで受取り RuntimeExcepton を再通知する．try 節の中の例外処理記述では IOException を受取る例外処理記述を記述したい場合は図 3 のように記述する．

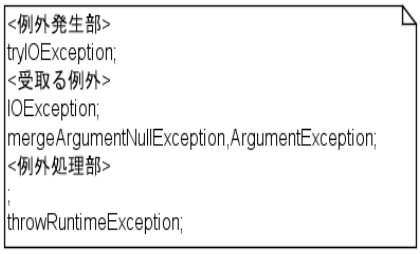

図 3　記述方法の例

シーケンス図のノートにテンプレートを付加することで，例外処理関連コードを自動生成することが可能になる．

5　事例検証

本章では，第 3 章で定義した例外処理パターンの事例検証を行なう．

5.1 事例の概要

ESS ロボットチャレンジ 2013 の仕様 [3] に基づいた自律走行ロボット（以下，Kobuki）の制御ソフトウェアを事例とし，定義した例外処理パターンの妥当性を示す．

5.2 パターンが適用可能かどうかの確認

第 3 章で提案したパターンを適用することで，アクションのプログラムコードの例外処理が記述可能であるかを，事例検証を行ない確認する．パターンを適用せずに記述されたプログラムコードと，適用して記述したプログラムコードを比較することで検証を行なう．Kobuki の制御ソフトウェアには六つの例外処理記述が存在し，それぞれの例外処理記述に対してパターンを適用した．結果として，提案したパターンを用いてパターンを適用しない場合と同様のプログラムコードが記述可能であることを確認できた．

6 考察

本章では，次の二つの視点から例外処理関連コードの自動生成の可能性に関する考察を行なう．

6.1 パターン定義による自動生成の妥当性

本研究では，アクションの例外処理関連コードを自動生成するために，パターン定義を行なった．パターンを定義することによって，アーキテクチャに例外処理の情報を付加することができ，アーキテクチャとプログラムコードを対応付けることができた．

第 5 章で示した通り，定義したパターンを用いて自律走行ロボットの制御ソフトウェアにおける例外処理記述を全て記述できることを確認した．自律走行ロボットの制御ソフトウェアにログの機能を付け加えた場合，ArrayIndexOutOfBoundsException が発生すると考えられる．この例外は，4.1 節で定義した構文を用いて図 4 のように記述することができる．

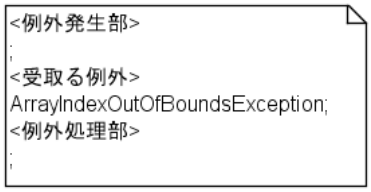

図 4 想定される例外のノート記述

生成されるコードとパターンとの関係を示したさいに，これらは一対一に対応している．このことから，我々が提案したパターンのパターン分けは独立に分類できており，パターン分けは妥当であったと言える．上記より，パターン定義による自動生成は妥当であると考える．

今回は例外処理に着目しパターン定義を行なったが，例外処理以外の項目についても規則的な記述をパターン定義することでコード生成が可能になると考える．本研究で示したように，パターンを定義し対応するアーキテクチャ記述を作成することで，パターン定義した部分のアクションコードが自動生成可能になると考える．

6.2 アーキテクチャ記述の拡張方法の妥当性

本研究では，定義したパターンを用いて自動生成可能にするためにアーキテクチャ記述を拡張した．我々は，既存のコード生成ツールのアクション部分がシーケンス図を基に生成されていることから，シーケンス図のノートに必要な情報を付加した．ノートにはテンプレートを作成し，生成したいパターンと必要な情報を書き込むだけでよい．見やすさ，分かりやすさの観点から拡張方法は妥当であったと考える．また，既存のコード生成ツールの入力にはノートを用いておらず，シーケンス図の構文に影響を与えない．既存の自動生成に支障がなく，UML 本来の意味を保つことが出来ている．このことからも，拡張方法は妥当であったと考える．

例外処理以外の項目でパターン化が可能な場合には，同様にアーキテクチャに記述することで自動生成が可能になると考える．例として，判定条件とそれに対応する実行動作の自動生成が挙げられる．

7 おわりに

本研究では，アクションの例外処理関連コードを自動生成可能にするために例外処理コードのパターンを定義し，アーキテクチャの記述方法を提案した．例外処理の調査，整理を行ない，パターン定義を行なった．抽象化したパターンを用いてアーキテクチャ記述を変更した．

今後の課題として，本研究で提案したアーキテクチャ記述を用いるために，E-AoSAS++ の開発体系に提供されているコード生成ツールの拡張を行なう必要がある．また，例外処理以外の項目についてもパターン定義が行なえるかの考察を行なう必要がある．

参考文献

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

[2] Object Management Group, “*Model Driven Architecture,*” http://www.omg.org/mda/, 2001.

[3] ESS ロボットチャレンジ実行委員会，“*ESS* ロボットチャレンジ *2013* 競技ルール *Ver.1*，” http://www.qito.kyushu-u.ac.jp/ess/2013/rule2013_Ver.1.pdf，2013.

[4] 長大介，加藤大地，蜂巣吉成，沢田篤史，野呂昌満，“E-AoSAS++ に基づく開発支援環境コード生成ツールの提案，”情報処理学会研究報告. ソフトウェア工学研究会報告，vol.2009，no.31，pp.113-120，2009.