コンテキストを考慮した組込みシステムのアスペクト指向アーキテクチャに関する研究

―湯沸かしポットを題材にして―

2012SE113 小林純大　2012SE156 宮木華奈
2013SE052 井上航汰
指導教員：野呂昌満

1　はじめに

外部状況に応じて振る舞いを変化させる，コンテキストアウェアな組込みシステムが広く普及している．一方で，組込みシステムに対する非機能要求として，耐故障制，リアルタイム性，省電力化等が重要視されている [1]．組込みソフトウェア開発において，オブジェクト指向をプライマリコンサーンとしたとき，コンテキストに応じた振る舞いに関する記述は横断的関心事となる．

コンポーネント間の関係が複雑化することで，コードが理解しにくくなり，その結果，保守性の低下を招く．この問題を解決することも視野に入れ，本研究室では，組込みシステムのためのアスペクト指向アーキテクチャ [2] を提案している．

本研究の目的は，提案しているアーキテクチャに基づいて記述したコードが理解しやすくなることを検証することである．提案するアーキテクチャに基づいて省電力化を実現し，オブジェクト指向に基づく実現と比較することにより，コードが理解しやすくなったことを明らかにする．

我々は，組込みシステムのためのアスペクト指向アーキテクチャに基づき，省電力化に関連する記述をコンテキストとレイヤに分離した．その後，それらの省電力コンサーンをアスペクトとしてモジュール化した．プログラミング言語独立を目的とし，提案しているアーキテクチャに基づく省電力化の実現には，デザインパターンを用いる．事例として湯沸かし保温制御可能な電気ポット（以下，電気ポット）を取り上げ，提案しているアーキテクチャに基づき記述したコードとオブジェクト指向に基づき記述したコードについて，循環複雑度が低いコードは読み手から理解されやすいという仮定の下，それを用いて定量的に比較する．

それらのプログラムを比較した結果，オブジェクト指向で同じシステムを実現したものと比べ，循環的複雑度（Cyclomatic Complexity）が減少したので，アスペクト指向を適用したことにより，柔軟性の高いプログラムを実現することが確認できた．

2　背景技術

本研究において背景技術となる，コンテキスト指向，アスペクト指向，Cyclomatic Complexity，組込みシステムのためのアーキテクチャについて述べる．

2.1　コンテキスト指向

コンテキスト指向プログラミングとは，プログラムの実行時に，外部環境に応じた振る舞いのモジュール化を実現するために考えられたプログラミング手法である．

コンテキスト指向プログラミング言語は主に以下の言語要素を提供する [3].

- レイヤ…外部環境に応じた特定の振る舞いを「アスペクト」として分離し，モジュール化したもの
- 部分メソッド…モジュール化したクラスに記述される具体メソッド
- 層活性…層を動的に活性化，非活性化させ，現在活性な層の部分のメソッドのみを実行させる機構

2.2　アスペクト指向プログラミング

アスペクト指向プログラミングとは，ソフトウェアの特定の振る舞いを「アスペクト」として分離し，モジュール化するプログラミング方法である．オブジェクト指向プログラミングでは，属性（データ）と操作（メソッド）の集合であるオブジェクトをソフトウェアの分解単位として扱うが，オブジェクトとしてうまく分解できないものが存在し，複数のオブジェクト間にまたがる操作となる．これを「横断的関心事」と呼ぶ．横断的関心事はコード中に散在するため，全てを把握し管理することが難しい．アスペクト指向プログラミングでは，この横断的関心事を「アスペクト」としてモジュール化し分離することで，把握・管理・変更を容易にする．アスペクト指向を導入することにより，既存のコードに手を加えなくてもコード中に散在する機能を持った部分を書き換えることができる．[4].

2.3　循環複雑度

循環複雑度 (Cyclomatic Complexity) とは，プログラムの制御の流れをグラフで表現し，グラフのもつ要素に基づいてプログラムの複雑性を示す尺度 (メトリクス) である [5]．あるプログラムに対し，1 つの入り口と 1 つの出口をもつグラフを対応づける．グラフの節点 (ノード) は制御が逐次に流れる連続した箇所であり，グラフの枝 (エッジ) は制御の分岐を表現している．グラフ理論によると，一般的に有向グラフがどの節点からも他の任意の節点に到達可能な場合，グラフに含まれる 1 次独立な閉路の数 V(G) は，

エッジの数 e とノードの数 n を用いて

$$V(G) = e - n + 1 \qquad (1)$$

と表現できる．しかし，プログラムの制御の流れを表現するグラフは，一般的に有向グラフがどの節点からも他の任意の節点に到達可能でないので，出口から入り口へ向かう仮想的なエッジを 1 つグラフに追加して計算する．これにより，グラフに含まれる一次独立な閉路の数 V(G) は (1) にエッジを 1 つ加えたものとなるので，

$$V(G) = (e - n + 1) + 1 = e - n + 2 \qquad (2)$$

となる．

2.4 組込みシステムのためのアスペクト指向アーキテクチャ

本研究室で提案している組込みシステムのための共通アーキテクチャ [2] を図 1 に示す．

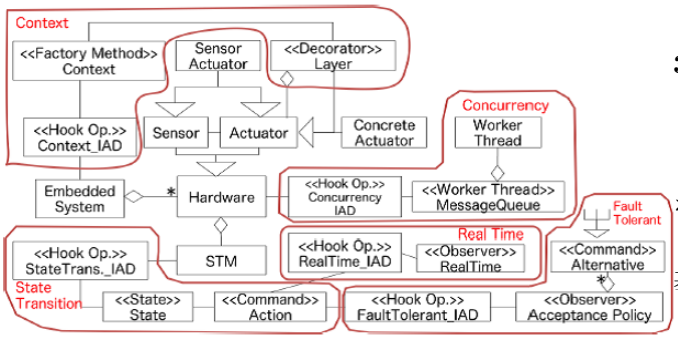

図 1　本研究室で提案しているアーキテクチャ

組込みシステムのハードウェアをセンサとアクチュエータに分類し，多相型として定義している．また，組込みシステムのハードウェアの振る舞いを状態遷移機械として実現している．

コンテキストは，システムの状態に応じて特定の振る舞いを変えることから，コンテキストに応じた振る舞いに関する記述を横断的関心事であるとし，内部状態をコンテキストとして Context の中にアドバイス記述として実現している．また，横断的関心事をアスペクトとして Layer にモジュール化している．これにより，コンテキストと横断的関心事の対応付けができ，統一的な取り扱いを実現している．

3 オブジェクト指向, アスペクト指向に基づく電気ポットの設計および実装

3.1 ポットの構造

電気ポットには以下のモジュールが付属するものとする．

- 温度を測るセンサ
- 水量を図るセンサ
- 蓋の開閉を確認するセンサ
- 給湯ボタン
- 予約時間を計るタイマー
- 水を温めるヒータ

電気ポットの機能を示す．

- 蓋が開いているときは，ヒータを停止する
- 沸騰するまで水を温めた後は，自動的に保温する
- 水が入っていないときは，空焚き防止のためヒータが停止する

電気ポットの省電力化実現方法として，水温と水量をコンテキストとし，Heater の振る舞いを Pot が制御する．次に振る舞いを変化させる条件と，その振る舞いを示す．

- 水量が多く，水温が高いとき，弱火で長時間温める
- 水量が多く，水温が低いとき，強火で長時間温める
- 水量が少なく，水温が高いとき，弱火で短時間温める
- 水量が少なく，水温が低いとき，強火で短時間温める

3.2 オブジェクト指向アーキテクチャに基づく設計および実装

オブジェクト指向に基づき，省電力を考慮した湯沸かしポットの設計および実装を行なった．

電気ポットに付属するモジュールをオブジェクト指向に基づき図 2 のように表現した．

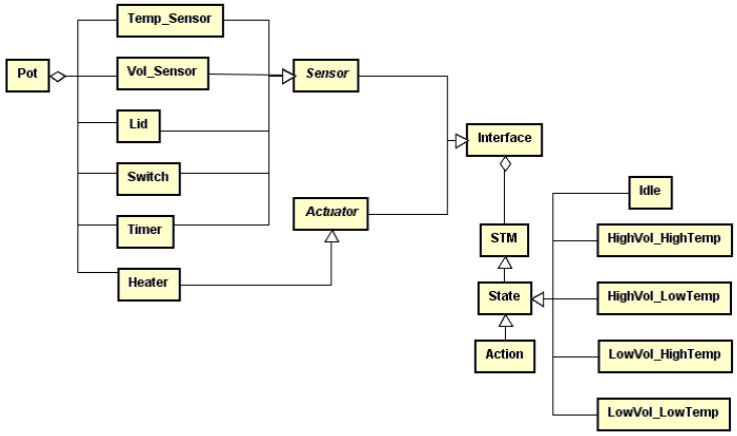

図 2　オブジェクト指向による省電力を考慮した湯沸かしポットの構造

我々は，電気ポットのアーキテクチャ設計において多相型を導入し，Interface のサブクラスとして抽象クラスを定義した．抽象クラス Actuator のサブクラスとして Pot からメッセージを受け取りヒータを動作させる Heater を定義した．抽象クラス Sensor のサブクラスとして，温度を測り Pot に渡す Temp_Sensor，水量を測り Pot に渡す Vol_Sensor，蓋の開閉を確認し Pot に渡す Lid，給湯ボタンの状況を確認し Pot に渡す Switch，予約された時間を計り Pot に渡す Timer をそれぞれ定義した．Pot は，Heater に定義された振る舞いを制御する役割を持つこと

から，Heater のそれぞれの振る舞いに対応する状態をインタフェースが持っていると考えられる．したがって我々は，State にインタフェースの持ちうる状態を定義し，Pot が Sensor 群から渡された値の応じて状態を変化させ，その状態から Heater の制御の判断を行なわせることが適切であると判断した．State に定義された状態に応じて Heater を制御する Pot のコードを図 3 に示す．

```
public static void StateAnalyze(double v,double t,State s){
    if(v==0){   //空焚き防止は最優先
        s = State.Idle;
    }else if(/*水量が多い*/ && /*水温が高い*/){
        s = State.HighVol_HighTemp;
    }else if(/*水量が多い*/ && /*水温が低い*/){
        s = State.HighVol_LowTemp;
    }else if(/*水量が少ない*/ && /*水温が高い*/){
        s = State.LowVol_HighTemp;
    }else if(/*水量が少ない*/ && /*水温が低い*/){
        s = State.LowVol_LowTemp;
    }
}
public static void doIt(State s){
    switch(s){
    case HighVol_HighTemp:      //水量が多く水温が高い
        Heater.LongTime_LowHeat();
        break;
    case HighVol_LowTemp:       //水量が多く水温が低い
        Heater.LongTime_HighHeat();
        break;
    case LowVol_HighTemp:       //水量が少なく水温が高い
        Heater.ShortTime_LowHeat();
        break;
    case LowVol_LowTemp:        //水量が少なく水温が低い
        Heater.ShortTime_HighHeat();
        break;
    }
}
```

図 3　Pot の制御プログラム

Heater の持ちうる状態及び，状態の遷移を引き起こすイベントの関係を図 4 に示す．

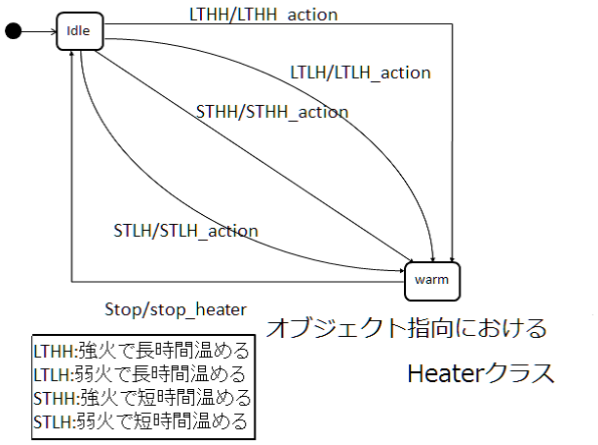

図 4　遷移を引き起こすイベントと Heater の状態の関係

図 4 において，LTHH，LTLH，STHH，STLH は，Pot から渡されるメッセージである．

Heater は，Idle 状態において，Pot から温めの指示を受けとると，warm 状態へ遷移し，指示内容に応じてヒータを動作させる．warm 状態において，Pot から停止の指示を受け取ると，Idle 状態へ遷移し，ヒータを停止させる．

3.3　アスペクト指向アーキテクチャに基づく設計および実装

提案しているアスペクト指向アーキテクチャに基づき，省電力を考慮した湯沸かしポットの設計および実装を行なった．

アスペクト指向に基づく湯沸かしポットの構造を図 5 に示す．

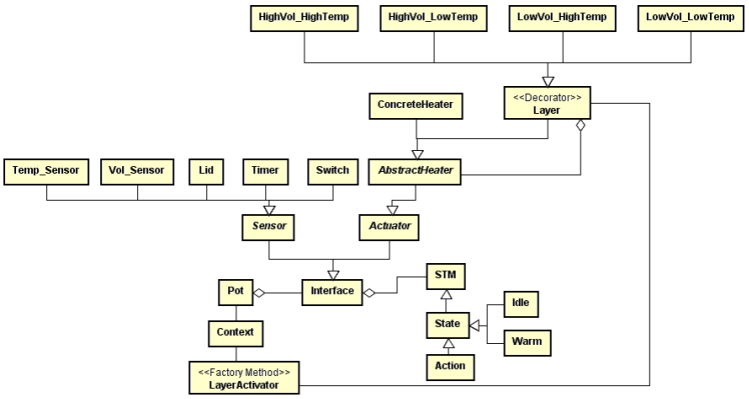

図 5　アスペクト指向による省電力を考慮した湯沸かしポットの構造

センサの取得したコンテキストと，それに関わる省電力コンサーンを統一的に扱うために，Heater の振る舞いに関する記述をアスペクトとし，Layer としてモジュール化した．このアーキテクチャでは，コンテキストに応じて必要な Layer を活性化させることで，Heater の振る舞いの切り替えを実現している．Layer の活性化を司る LayerActivator は，Layer の活性化に関わるイベントを Context から受け取り，対応する Layer の活性化を行なう．

Layer の活性化に関わる LayerActivator のコードを図 6 に示す．

```
public class LayerActivator{
    public static void LayerActivate(Event ev,AbstractHeater h){
        switch(ev){
        case LongTime_LowHeat:      //水量が多く水温が高い
            h = new HighVol_HighTemp();
            break;
        case LongTime_HighHeat:     //水量が多く水温が低い
            h = new HighVol_LowTemp();
            break;
        case ShortTime_LowHeat:     //水量が少なく水温が高い
            h = new LowVol_HighTemp();
            break;
        case ShortTime_HighHeat:        //水量が少なく水温が低い
            h = new LowVol_LowTemp();
            break;
        }
    }
}
```

図 6　LayerActivator の制御プログラム

4　オブジェクト指向設計とアスペクト指向設計の比較

3 章で作成した湯沸かしポットのオブジェクト指向設計とアスペクト指向設計の比較を行なう．

4.1　状態遷移機械の比較

状態遷移機械における Heater 部分の比較を図 7 に示す．

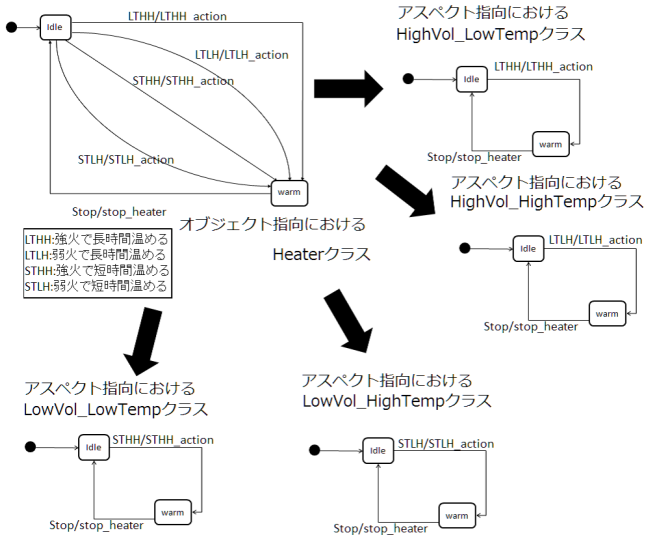

図 7　状態遷移機械の比較

図 7 に示した状態遷移機械において，オブジェクト指向では，Heater に水温と水量に応じた 4 つの振る舞いを定義した．

- LongTime_HighHeat
- LongTime_LowHeat
- ShortTime_HighHeat
- ShortTime_LowHeat

それに対し，アスペクト指向では，Heater の 4 つの振る舞いを，Layer のサブクラスとして，4 つに分離した．それにより，コンテキストに応じた Heater のそれぞれの振る舞いを別々のクラスで扱うことができるようになった．

4.2　オブジェクト指向とアスペクト指向のコードの比較

オブジェクト指向に基づき記述したコードとアスペクト指向に基づき記述したコードの理解しやすさを比較する．循環複雑度を用い，定量的に比較を行なう．本研究室では，SourceMonitor を使用し，循環複雑度を収集した．SourceMonitor から収集されたメトリクスのうち，循環複雑度（AvgComplexity）を用いてコードの理解しやすさを比較する．

オブジェクト指向に基づき記述したコードとアスペクト指向に基づき記述したコードに対して SourceMonitor を使用し，それぞれのメトリクスを収集した結果を図 8，図 9 に示す．

Java Checkpoints In Project 'object'

Checkpoint Name	Files	Lines	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Baseline	2	133	8	8	4.26	2.20

図 8　オブジェクト指向の循環複雑度

Java Checkpoints In Project 'aspect'

Checkpoint Name	Files	Lines	Max Complexity	Max Depth	Avg Depth	Avg Complexity
Baseline	10	216	8	8	2.80	1.63

図 9　アスペクト指向の循環複雑度

図 8，図 9 より，オブジェクト指向の循環複雑度は 2.20，アスペクト指向の循環複雑度は 1.63 となった．オブジェクト指向とアスペクト指向に基づくコード全体の循環複雑度は変わらないが，コンポーネントごとに循環複雑度を平均した場合，アスペクト指向に基づくコードの循環複雑度（AvgComplexity）の方がオブジェクト指向に基づくコードより小さいことがわかった．また，提案しているアーキテクチャにより，コンポーネントの役割が明確になり，意味の断片化を補うことが可能になった．

5　まとめ

本研究において我々は，省電力を考慮した電気ポットを事例とし，オブジェクト指向とアスペクト指向それぞれに基づきコードを記述し，循環複雑度を用いて比較を行なった．

コンテキストアウェア組込みシステムでは，コンテキストに応じた振る舞いに関する記述は複数のコンポーネントに散在する．それらの散在した振る舞いに関する記述を，コードを理解しやすくすること及びプログラミング言語独立を目的とし，統一的に分離した．これにより，Pot クラスに記述されていた，クラス名からは判断できない制御を，アスペクト指向に基づき分離することで，クラス名から役割を汲み取れるようになり，クラス名から制御の役割を明確にすることができた．省電力コンサーンを横断的関心事としたプログラムにおいて，提案しているアーキテクチャに基づきコードを記述することにより，オブジェクト指向に基づきコードを記述する場合と比べると，コンポーネントごとの循環複雑度が小さくなり，コードを理解しやすくすることができる．

参考文献

[1] 新屋敷泰史，三瀬敏郎，江浦洋平，畑中久典，橋本正明，鵜林尚靖，片峯恵一，中谷多哉子：“組込みソフトウェア非正常系の概念モデル” 情報処理学会，pp105-112, 2004

[2] 江坂篤侍，野呂昌満，沢田篤史，繁田雅信，谷口弘一：“コンテキストアウェアネスを考慮した組込みシステムのためのアスペクト指向アーキテクチャの適用と実現” ソフトウェア工学の基礎ワークショップ（FOSE2016）論文集，vol.23，pp.175-180，2016.

[3] 坂本寛幸，井垣宏，中村匡秀：“コンテキストアウェアアプリケーションの開発を容易化するセンササービス基盤” 電子情報通信学会，2009

[4] 伊藤広大，亀山怜希，横山脩二：“アスペクト指向に基づく SOA の考察 ” 南山大学　数理情報学部　ソフトウェア工学科　卒業論文（2011）

[5] 山田茂，高橋宗雄：“ソフトウェアマネジメントモデルの入門” pp.117-122，1993