行動履歴を利用したコンテキスト指向組込みシステムのソフトウェアアーキテクチャに関する研究

2015SE009 長谷川結佳　2015SE045 間瀬友美子
指導教員：沢田篤史

1　はじめに

近年，組込みシステムの高機能化により，コンテキストに応じてシステムの振舞いを変化させるコンテキストアウェアな処理が可能となった．さらに最近ではコンテキストの一つとして行動履歴が利用されるようになってきた．

組込みシステムにおいて，センサ情報だけでなく行動履歴をコンテキストとして扱うことにより，ユーザに対する振舞いの種類を多くさせることができるが，それによって振舞いに関する記述も多くなる．振舞いに関する記述が多くなると，記述の変更や追加が容易でなくなる．また，行動履歴のデータの種類が多くなると，コンテキストの条件式も多くなり複雑になるので保守性が低下する．

本研究の目的は，行動履歴を利用した組込みシステムの保守性を向上させることである．コンテキストの条件式が複雑になってしまうと振舞いの追加や変更がしにくいという問題を解決するためにコンテキストの条件式とコンテキストに応じた振舞いの記述を局所化する必要がある．これにより振舞いの追加や変更を容易にする．

我々は，組込みシステムの振舞いを動的に変更するために，江坂らが提案した自己適応のための PBR パターン [1] を適用し，アーキテクチャを設計する．そのアーキテクチャに基づいて，アプリケーション設計を行う．そのさい，スマートベッドを題材とし，コンテキストに行動履歴を用いる．アプリケーション設計に基づいて，システムのプログラムを Java で作成し，アーキテクチャを使用しない場合のコードと比較することによって保守性の観点から考察する．

2　行動履歴を利用したコンテキスト指向組込みシステム

近年，各種機器に組み込まれた制御を行うためのコンピュータシステムである組込みシステムが普及している．その高機能化や多機能化により，多数のセンサを使用したシステムが登場してきた．このようなシステムでは，周辺の環境や人，物の状況の変化を正確に取得できるようになった．システムの振舞いを変化させる外部環境の情報をコンテキストという．上記の情報をコンテキストとして，コンテキストに応じてシステムの振舞いを柔軟に変化させるコンテキストアウェアな処理が実現可能となった．

さらに最近ではより高度なコンテキストアウェアな処理を実現するためにコンテキストの一つとして行動履歴が利用されるようになってきた．行動履歴とは，ユーザの振舞いの変化や，過去にシステムがユーザに対してどのような振舞いをしたかを蓄積したデータである．リアルタイムの値を利用するだけでなく，過去の状況とその状況に対してのシステムや人の行動，行動をした結果を利用することによって，ユーザの嗜好に沿ったコンテキストの組み合わせが増える．つねに変化する要求に柔軟に応じるための振舞いを多く表現できるようになるので，ユーザの嗜好に合わせたサービスを提供することができる．

3　行動履歴を利用するスマートベッドのアーキテクチャ設計

3.1　背景技術

3.1.1　コンテキスト指向

コンテキスト指向は文脈に依存する振舞いをモジュール化するためのプログラミングの方法である．コンテキストとは，プログラムから観測することのできる外部環境やシステムの内部状態で，時間や場所とともに変化し，それがプログラムの様々な実態の実行に影響を与えるものを指す [2]．一般的にコンテキストに依存した振舞いはシステムの支配的分割に対する横断的関心事となる．

3.1.2　PBR パターン [1]

PBR(Policy-Based Reconfiguration) パターンとは，江坂らが提案している静的および動的に再構成を行う自己適応のためのアーキテクチャパターンである．

3.2　センサ情報と行動履歴を用いた制御

組込みシステムにコンテキスト指向を適用するとさまざまな情報からコンテキストを作成することができる．その中でも本研究で扱うのはセンサ情報と行動履歴である．センサの情報をそのままコンテキストとして扱うと，リアルタイムの値をもとにシステムがユーザに対して動作する．それにより，ユーザに対する振舞いはシステムにあらかじめ組込まれた単調なものになってしまう．行動履歴はユーザの振舞いの変化や過去にユーザに対してどのような振舞いをしたかを蓄積したデータである．それをもとにユーザに対してシステムが動作するので，ユーザに対する振舞いはそのユーザに合わせたものに変化する．よってセンサ情報を使うだけではなく行動履歴も使うことによって，ユーザに適応した振舞いが提供できるようになる．

3.3　スマートベッドのアーキテクチャ設計

本研究では，行動履歴を利用したアプリケーションとしてスマートベッドを設計する．スマートベッドに PBR パターンを適用し，コンテキストに応じて起こし方を動的に

再構成するアーキテクチャを設計する．

本研究で設計したアーキテクチャの静的構造と動的振舞いを図 1，図 2 に示す．

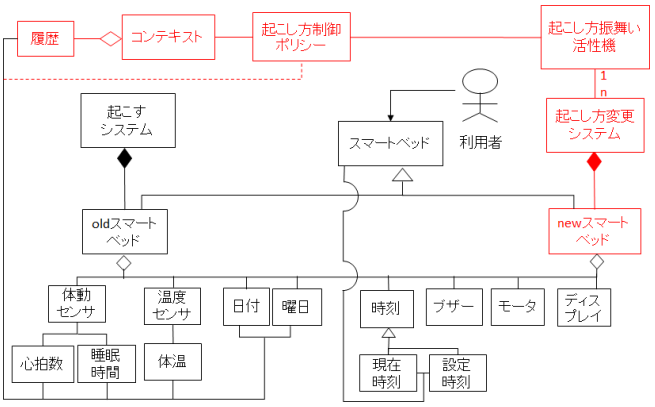

図 1　アーキテクチャ静的構造

- 体動センサ，温度センサ，ブザー，モータ，ディスプレイ
  スマートベッドに付属しているコンポーネント
- 履歴
  体動センサから取得した心拍数と睡眠時間，温度センサから取得した体温，日付，曜日を履歴に蓄積する
- old スマートベッド
  再構成前のコンポーネント
- コンテキスト
  履歴に蓄積されているデータを照らしあわせ，どの起こし方をするのかを決定する
- 起こすシステム
  再構成前のコンポーネントの情報を保持する
- 起こし方変更システム
  再構成後のコンポーネントの情報を保持する
- 起こし方制御ポリシー
  コンテキストに応じて起こし方振舞い活性機を生成し，メッセージを送る
- 起こし方振舞い活性機
  起こし方制御ポリシーに応じて new スマートベッドと起こし方変更システムを活性化する
- new スマートベッド
  再構成後のコンポーネント

日付，曜日，スマートベッドに付属している温度センサと体動センサから検知したバイタルデータの状態からなる行動履歴をコンテキストと定義した．ユーザはスマートベッドに対し起床時刻を設定する (図 2 の 1，図 2 の 2)．起床時刻になり (図 2 の 3)，温度センサと体動センサが起動し (図 2 の 4) 体温, 心拍数, 睡眠時間の値を取得する．取得したこれらのデータをデータベースに蓄積する (図 2 の 5)．データを蓄積した後，起こし方制御ポリシーがメッセージ通信を横取りし (図 2 の 6) データベースに蓄積されている履歴をコンテキストが照らしあわせてどの起こし方をするか決定する (図 2 の 7)．起こし方制御ポリシーがコンテキストに基づいて起こし方の再構成を行うように起こし方振舞い活性機を生成し，メッセージを送る (図 2 の 8)．起こし方振舞い活性機は，コンテキストに応じた起こし方変更システムを生成し (図 2 の 9)，new スマートベッドを生成する (図 2 の 10)．生成された new スマートベッドに記述されている詳細に従ってブザー，モータ，ディスプレイを活性化させる (図 2 の 11)．

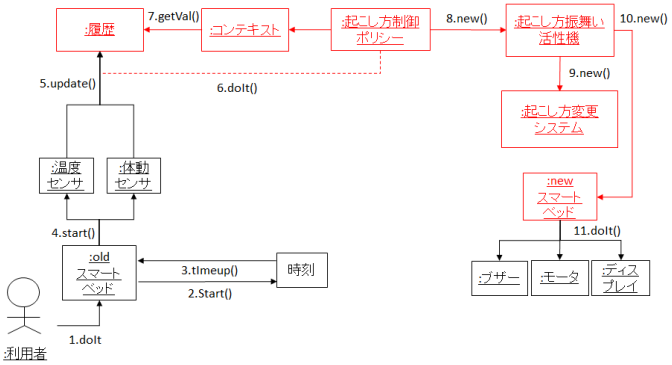

図 2　アーキテクチャ動的振舞い

4　行動履歴を利用するスマートベッドのアプリケーション設計

3 章のアーキテクチャに基づく具体的なアプリケーションの設計について述べる．本研究では，バイタルデータを体温・心拍数とし，体温は設定した自身の平均体温 +1 度未満，心拍数は設定した自身の平均心拍数 +10 回未満は正常であり，それを超えると異常であるとする．

スマートベッドシステムの動作の詳細を以下に示す．

平日の場合

- **バイタルデータが正常であった時**
  前日と二日前のバイタルデータがそれぞれ正常・異常のどちらであっても，設定された時間通りに起こす振舞いをする．
  - ブザーを鳴らす．
  - ディスプレイに’Good Morning’ と表示する．
  - モータを起動する．

- **バイタルデータが異常であった時**
  データベースを活性化させ，バイタルデータに関する行動履歴のデータを照らし合わせることで振舞いが変わる．
  - **前日のバイタルデータが正常だった時**
    二日前のバイタルデータがが正常・異常のどちらであっても，寝るか起きるかを選択させる．
    * ブザーを弱く鳴らす．
    * ディスプレイに休むか起きるかの選択画面を表示する．

– **前日のバイタルデータも異常だった時**
二日前のバイタルデータが正常か異常かで振舞いが変わる．

＊ **二日前のバイタルデータが正常であった時**
寝るか起きるかを選択させる．
· ブザーを弱く鳴らす．
· ディスプレイに休むか起きるかの選択画面を表示する．

＊ **二日前のバイタルデータも異常であった時**
病院に行くように促す．
· ブザーを弱く鳴らす．
· ディスプレイに病院に行くように警告画面を表示する．

休日の場合

- **バイタルデータが正常**
  – **前日の睡眠時間が自分の平均睡眠時間より長かった時**
  設定された時間通りに起こす．
    ＊ ブザーを鳴らす．
    ＊ ディスプレイに’Good Morning’ と表示する．
    ＊ モータを起動する．
  – **前日の睡眠時間が自分の平均睡眠時間より短かった時**
  時間をずらして起こす．
    ＊ ブザーを鳴らす．
    ＊ ディスプレイに’Good Morning’ と表示する．
    ＊ モータを起動する．
- **バイタルデータが異常だった時**
  起こさない．
  – ディスプレイに体温，心拍数を表示する．

5 考察

5.1 センサ情報のみとの比較

我々は，センサ情報だけでなく，行動履歴もコンテキストとして利用した．ここでは，コンテキストをセンサ情報のみとした場合との比較を行い，二つの観点から考察する．

一つは振舞いに関する考察である．コンテキストをセンサ情報のみとした場合と行動履歴も利用する場合とではシステムの振舞いに差が出る．例として，コンテキストをセンサ情報と行動履歴とした場合，当日が平日でバイタルデータが正常であった時は普通に起こす振舞いをし，異常であった時は前日と二日前のデータを照らし合わせ，選択させる振舞いか病院に行くことを促す振舞いをするように定義した．これがコンテキストをセンサ情報のみとした場合では，前日と二日前のデータを扱うことができないので，当日が平日である時，当日のバイタルデータが正常か異常かで二つの振舞いしか定義できない．これによって，行動履歴もコンテキストとして利用することにより，振舞いをよりユーザに合わせたものにすることが出来る．

もう一つは保守性に関する考察である．コンテキストをセンサ情報のみとした場合に比べて，行動履歴も利用した場合では，行動履歴を利用した分だけコンテキストの条件式が増えるので保守性が低下する．この問題を本研究ではアーキテクチャに PBR パターンを適用し，コンテキストの条件式と振舞いに関する記述を局所化することによって解決した．

5.2 実装の比較による保守性の考察

本研究では、アーキテクチャを使用しない場合のプログラムと 4 章でのアプリケーション設計をもとに PBR パターンを適用させる場合のプログラムの実装を Java で行った．

アーキテクチャを使用せずに実装した場合の詳細を述べる．コンテキストに応じて AlarmSystem を制御する SmartBed クラスのコードを図 3，振舞いの詳細が記述してある AlarmSystem クラスのコードを図 4 に示す．

```
public static void Context() {
  if(/*今日が平日*/){
    if(/*今日のバイタルデータが正常*/) {
      s = "GetupNormal";
    }
    else if(/*今日のバイタルデータが異常 */&&/*昨日のバイタルデータが正業*/) {
      s = "Getupgentlyselect";
    }
    else if(/*今日のバイタルデータが異常*/ && /*昨日のバイタルデータが異常*/
          && /*一昨日のバイタルデータが正常*/){
      s = "Getupgentlyselect";
    }
    else if(/*今日のバイタルデータが異常*/ && /*昨日のバイタルデータが異常*/
          && /*一昨日のバイタルデータが異常*/){
      s = "Getupgentlywarning";
    }
  }else{
    if(/*今日のバイタルデータが正常*/){
      if(/*今日の睡眠時間*/ < /*設定睡眠時間 */){
        s = "GetupShifttime";
      }else{
      s = "GetupNormal";
      }
    }else{
      s = "NoGetup";
    }
  }
}
public static void Behavior() {
  switch(s) {
  case "GetupNormal":
    alarmsystem.GetupNormal();//普通に起こす
    break;
  case "Getupgentlyselect":
    alarmsystem.Getupgentlyselect();//選択させる
    break;
  case "Getupgentlywarning":
    alarmsystem.Getupgentlywarning();//病院に行くことを促す
    break;
  case "GetupShifttime":
    alarmsystem.GetupShifttime();//時間をずらして起こす
    break;
  case "NoGetup":
    alarmsystem.NoGetup();//起こさない
    break;
  }
}
```

コンテキストに応じた振舞いを決定するメソッドと振舞いを決定したメソッドに基づいてAlarmSystemにメッセージを送るメソッドが一つのクラスに書かれている

図 3 SmartBed クラスのコード

コンテキストに応じた振舞いを決定するメソッドと振舞いを決定したメソッドに基づいて AlarmSystem クラスにメッセージを送るメソッドを SmartBed クラスに記述し

```
public class AlarmSystem {
   public void GetupNormal() {
      //＜普通に起こす振舞い＞
      //ディスプレイに'Good Morning'と表示する
      //ブザーを鳴らす
      //モータを動かす
   }
   public void Getupgentlyselect() {
      //＜選択させる振舞い＞
      //ディスプレイに選択させる画面を表示する
      //ブザーを小さく鳴らす
   }
   public void Getupgentlywarning() {
      //＜病院へ行くことを促す振舞い＞
      //ディスプレイに病院に行くことを促す画面を表示する
      //ブザーを小さく鳴らす
   }
   public void GetupShifttime() {
      //＜時間ずらして起こす振舞い＞
      //時間をずらしてディスプレイに'Good Morning'と表示する
      //時間をずらしてブザーを鳴らす
      //時間をずらしてモータを動かす
   }
   public void NoGetup() {
      //＜起こさない振舞い＞
      //ディスプレイに体調が悪いことを知らせる画面を表示する
   }
}
```

複数の振舞いの詳細が一つのクラスにすべて書かれている

図 4　AlarmSystem クラスのコード

た．SmartBed クラスにはこれらの他に様々な記述があるので振舞いの追加や変更を行うさい、該当する記述を探すのが困難になる．また，コンテキストに応じた振舞いの詳細は AlarmSystem クラスにすべて記述した．

PBR パターンを適用して実装した場合の詳細を述べる．再構成の仕方を定義している起こし方制御ポリシーである Policy クラスのコードを図 5 に示す．

```
public class Policy {
   private Context c = new Context();
   private int getup;
   public void doIt() {
      getup = c.context();
      if (getup == 1) {
         new GetupNormalDirector(builder).construct();//普通に起こす
      } else if (getup == 2) {
         new GetupgentlyselectDirector (builder).construct();//選択させる
      } else if (getup == 3) {
         new GetupgentlywarningDirector (builder).construct();//病院に行くことを促す
      } else if (getup == 4) {
         new GetupShifttimeDirector(builder).construct();//時間をずらして起こす
      } else {
         new NoGetupDirector(builder).construct();//起こさない
      }
   }
}
```

振舞いごとにクラスを分ける

図 5　Policy クラスのコード

Context クラスではコンテキストに応じた振舞いを決定し，起こし方のタイプとして番号を割り当てている．その Context クラスで割り当てられた番号に基づいて Policy クラスでは該当する ConfigurationBuilder クラスのサブクラスである Director クラスのインスタンスを生成し，メッセージを送っている．これにより，起こし方制御ポリシーとコンテキストの組み合わせに関する記述を変更するだけで，機能の変更や追加，削除を容易に行うことができる．また，振舞いの詳細の記述を AlarmSystem のサブクラスとして分離する．これにより振舞いの詳細の記述も変更しやすくなる．

システムの実行結果を図 6 に示す．

```
Console
<terminated> Main (2) [Java Application] C:¥Program Files¥Java¥jre1.8.0_31¥bin¥javav
アラームをセットしました
時間になりました

今日の日付:2018/12/11　今日の曜日:火曜日
計測した体温:37.8　計測した心拍数:98　計測した睡眠時間:7.45
をデータベースに追加しました。

＜選択させる振舞い＞
休むか起きるかを選択させる画面を表示しました
ブザーを小さく鳴らしました
```

図 6　実行結果

システムを実行した当日を 2018 年 12 月 11 日（火）とし，ユーザ平均体温，ユーザ平均心拍数，設定睡眠時間が 36 度，80 回，7 時間とした．また，前日である 2018 年 12 月 10 日（月）の体温，心拍数，睡眠時間は 36.5 度，85 回，7 時間 13 分であり普通に起こす振舞いをしていたとする．当日に測定した体温，心拍数，睡眠時間が 37.8 度，98 回，7 時間 45 分であり，当日の日付，曜日と一緒にデータベースに蓄積する．蓄積した履歴を照らし合わせて当日は平日でありバイタルデータが異常，前日はバイタルデータが正常であると判断し，起こし方を普通に起こす振舞いから動的に再構成し，選択させて起こす振舞いに変更した．これにより，再構成が適切に行われたと確認できた．

6　おわりに

組込みシステムの高機能化により, コンテキストアウェアな処理が可能となった. そこでコンテキストの一つとして行動履歴が利用されるようになってきた．組込みシステムにおいて，行動履歴をコンテキストとすると条件式と振舞いに対する記述も多くなり，保守性が低下すると考えられる．本研究では, 行動履歴を利用した組込みシステムの保守性を向上させるために，PBR パターンを適用したアーキテクチャ設計とアプリケーション設計をし，システムの実装を行った．実装したプログラムとアーキテクチャを使用しないプログラムを比較することで保守性の観点から考察を行った．

組込みシステムの開発では，非機能特性も設計し，実現しなければならない．非機能特性も横断的関心事になりうるので，適切にモジュール化する必要がある．よって非機能特性にも PBR パターンを適用させることが挙げられる．また，本研究では時刻に関する処理の部分の実装が不完全であるので，適切に実装する必要がある．

参考文献

[1] 江坂篤侍, 野呂昌満, 沢田篤史：インタラクティブシステムのための共通アーキテクチャの設計, コンピュータソフトウェア, Vol. 35, No. 4 (2018), pp. 3-15.

[2] 紙名哲生：文脈指向プログラミングの要素技術と展望, コンピュータソフトウェア, Vol. 31, No. 1 (2014), pp. 3-13.