2018年08月31日　セッションs5d

頻出パターンから学ぶUML状態マシン図

ソフトウェアに変換可能な
ソフトウェア仕様書を書けるようになろう！

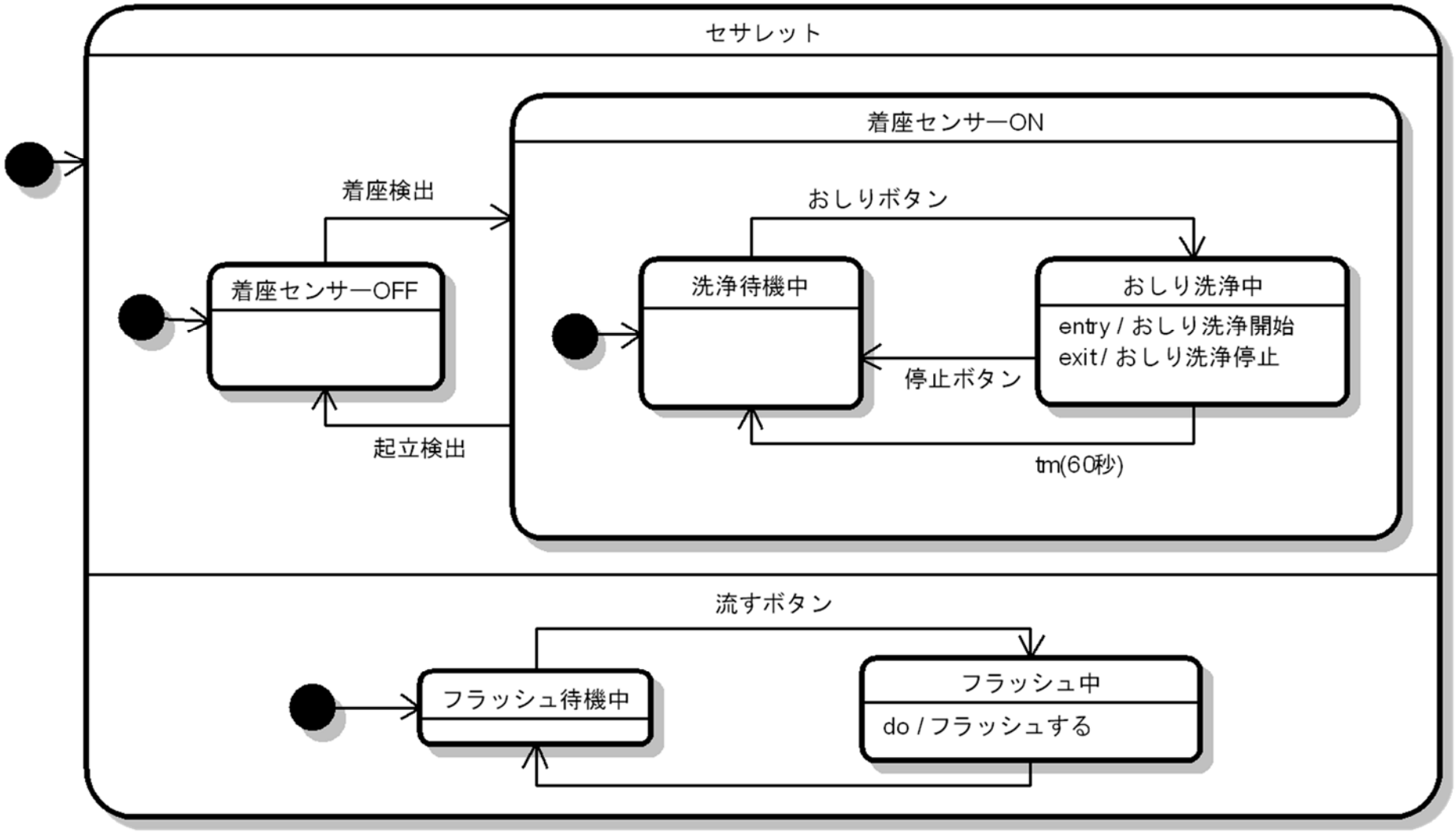
NPO法人
組込みソフトウェア管理者・技術者育成研究会
http://www.sessame.jp/

SESSAME
島　敏博

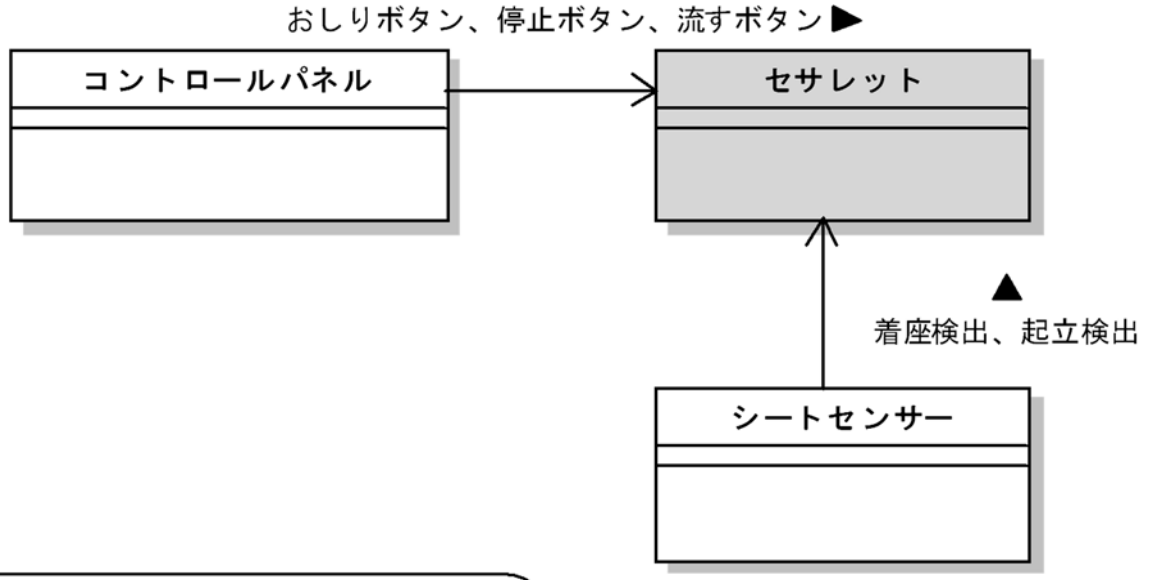
今日の内容

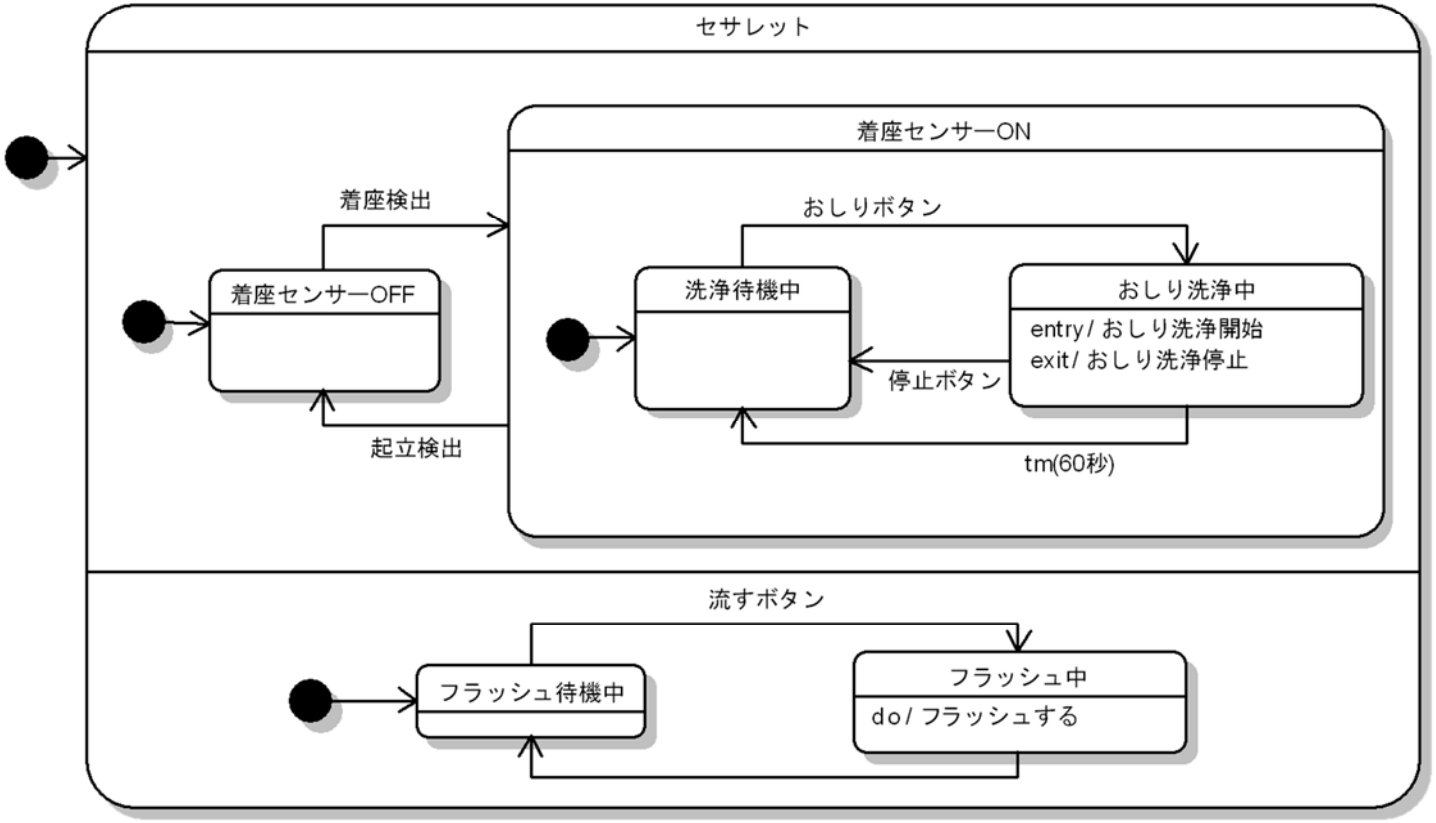
- みなさん、ソフトウェアの動的なふるまいを表現する仕様書はどのようにして書かれているでしょうか。UML状態マシン図はその目的にぴったりです。
- 昨年ご好評いただきました状態マシン図セミナーが今年も帰ってきました。昨年は表記法の詳細な解説をおこないましたが、今年は頻出パターンから 状態マシン図を学んでゆきます。
- このセッションは初心者の方から参加いただける内容にはなっていますが、決して初心者セミナーではありません。ベテランさんにも新たな気づきがあるような深い内容をわかりやすく解説します。さまざまな状態マシン図をじっくり読んでゆきましょう。

単純状態、合成状態、直交合成状態

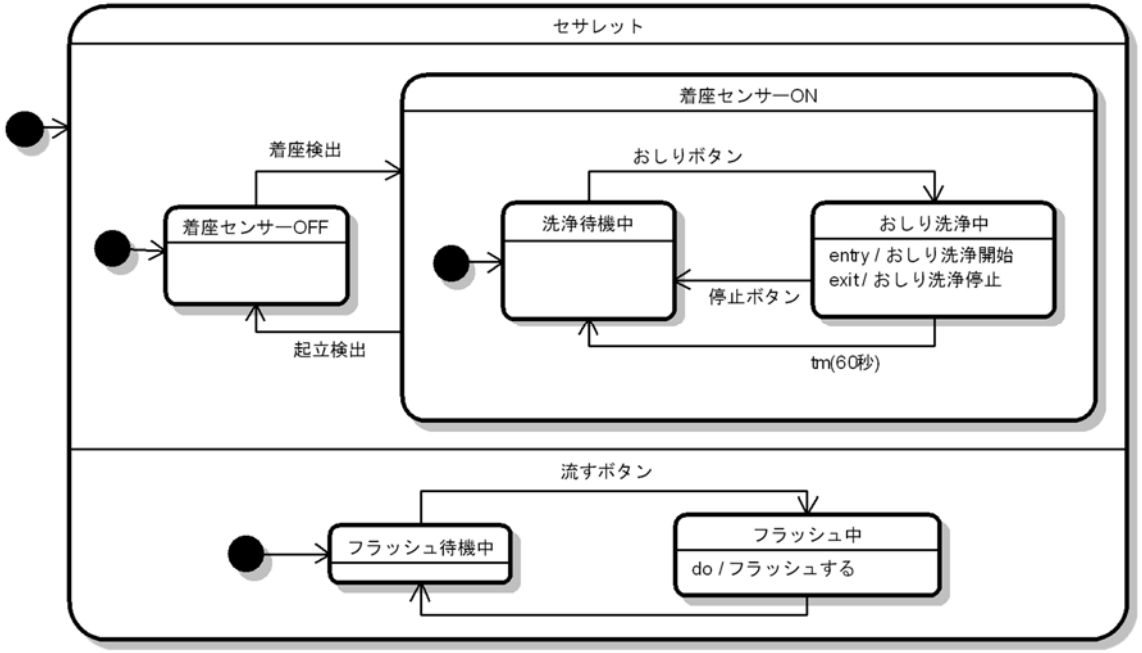


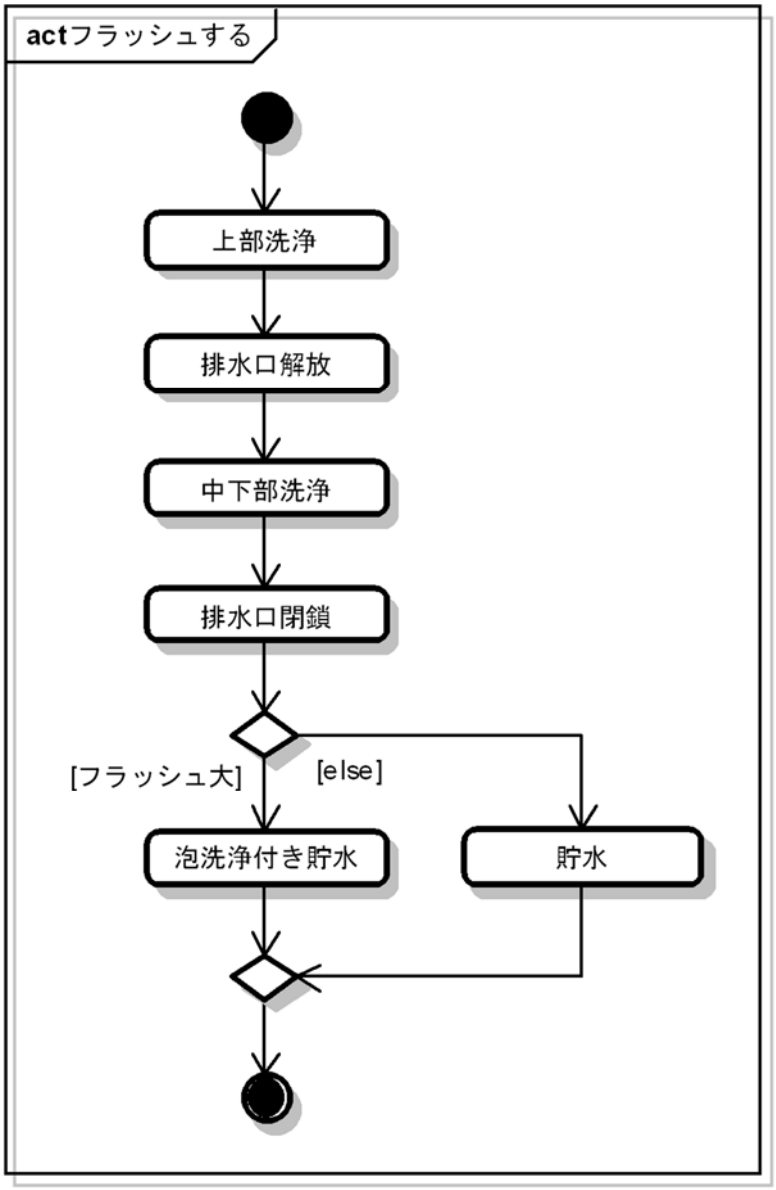
状態マシンは クラスに定義される





do / フラッシュする





アクティビティ図

ワンショットタイマーと繰り返しタイマー

図1

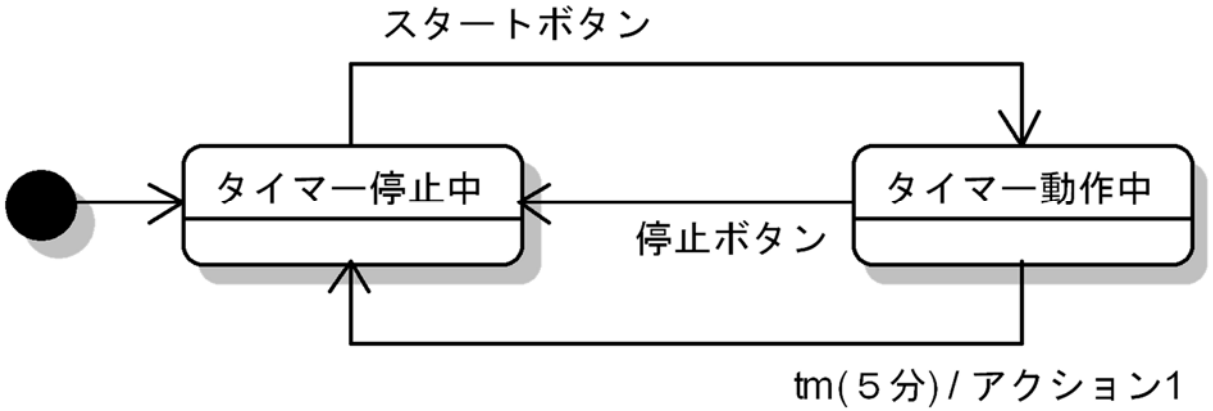
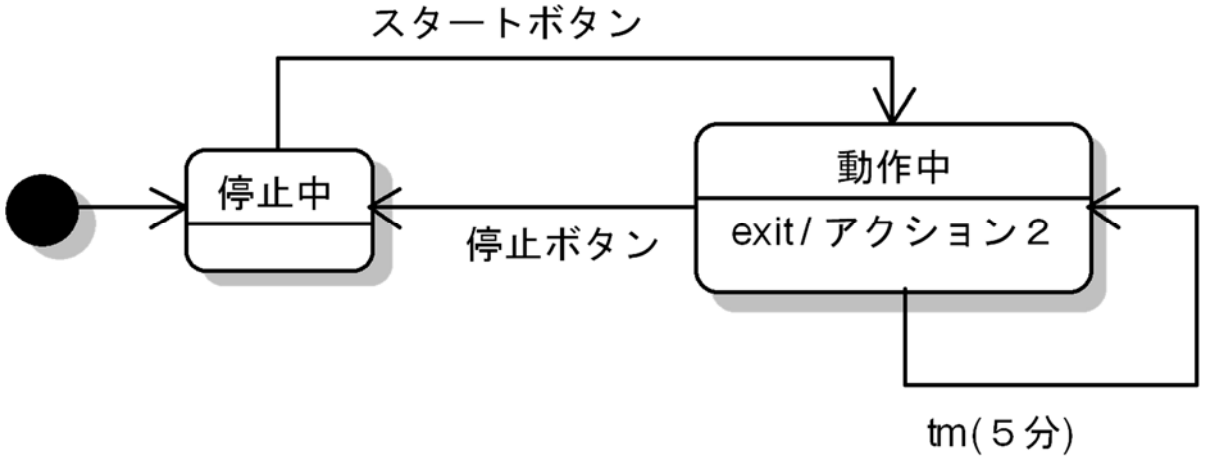
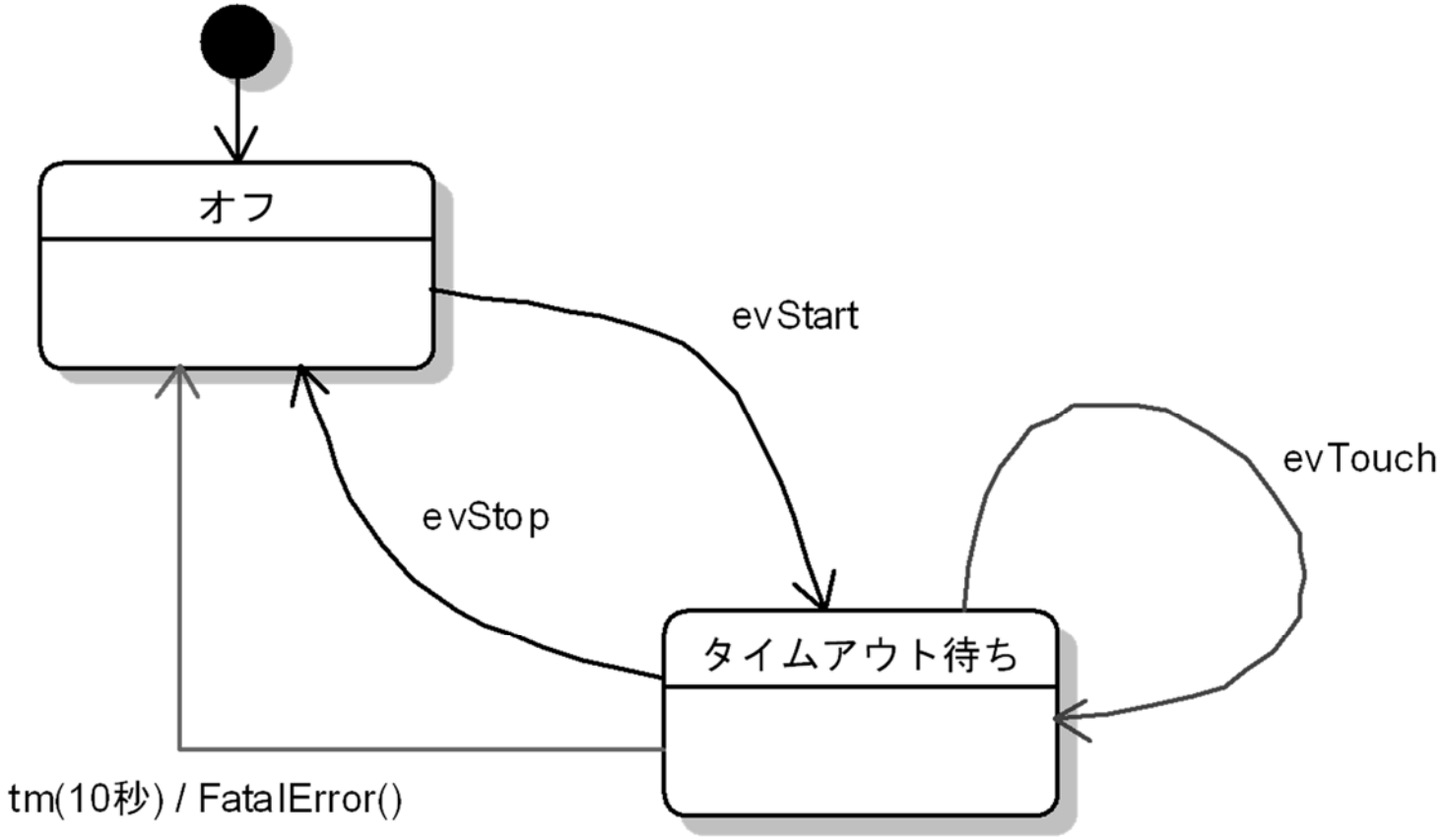


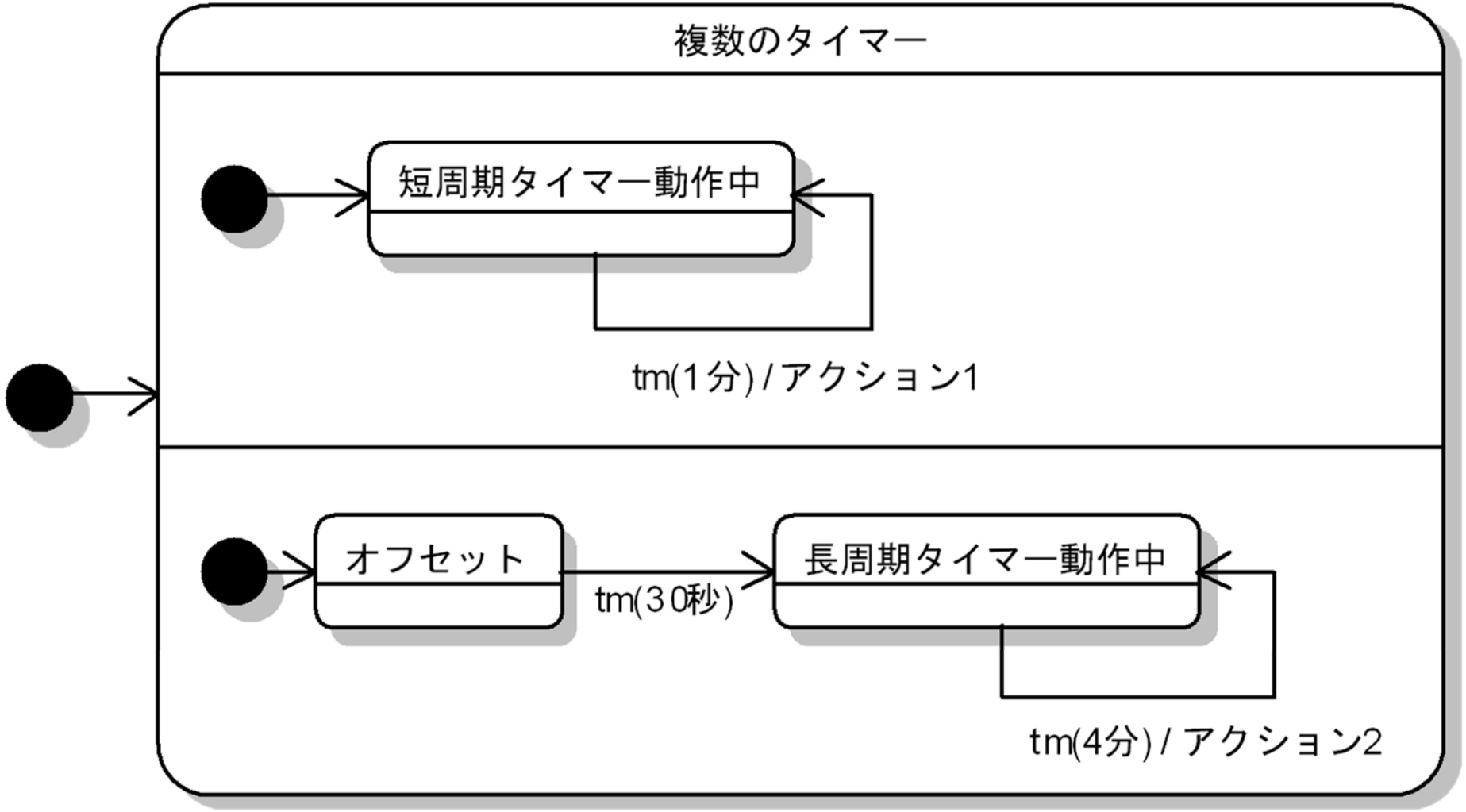
図2



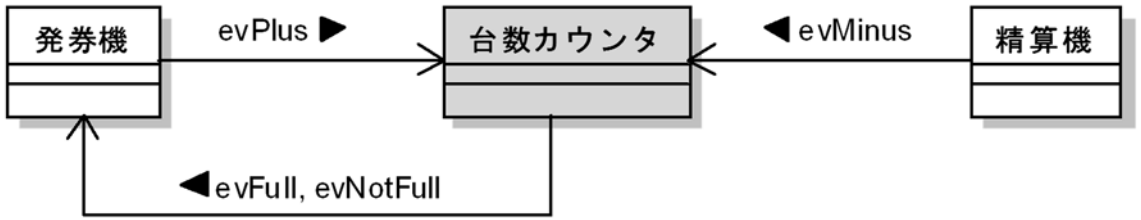
Watchdogパターン [Bru2009]

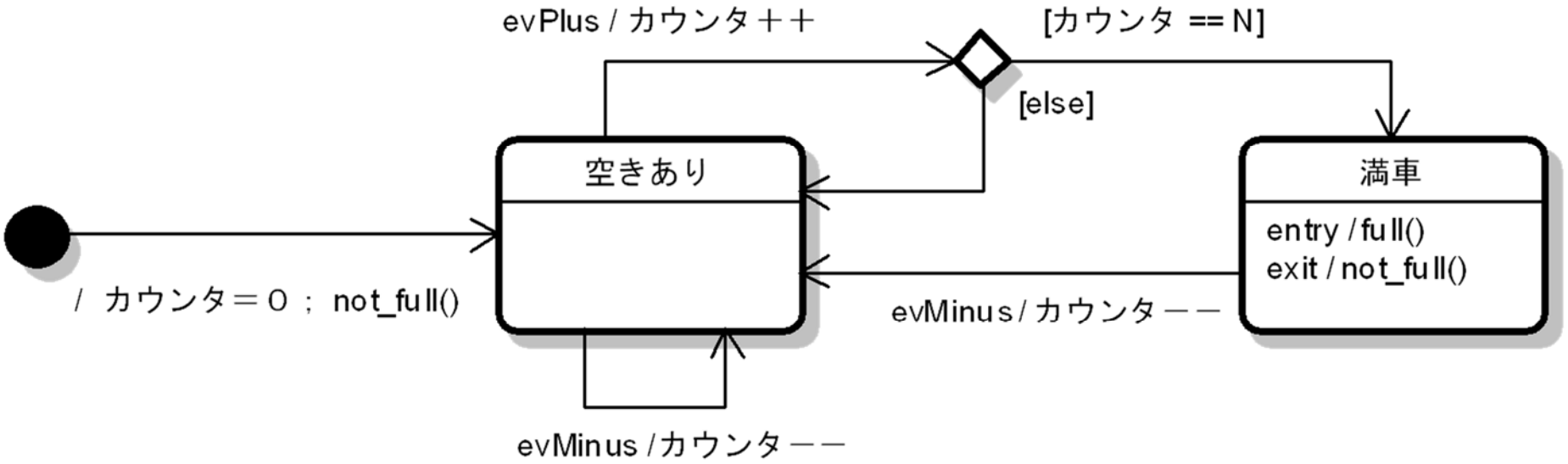


複数のタイムアウトを待つ



数を数える

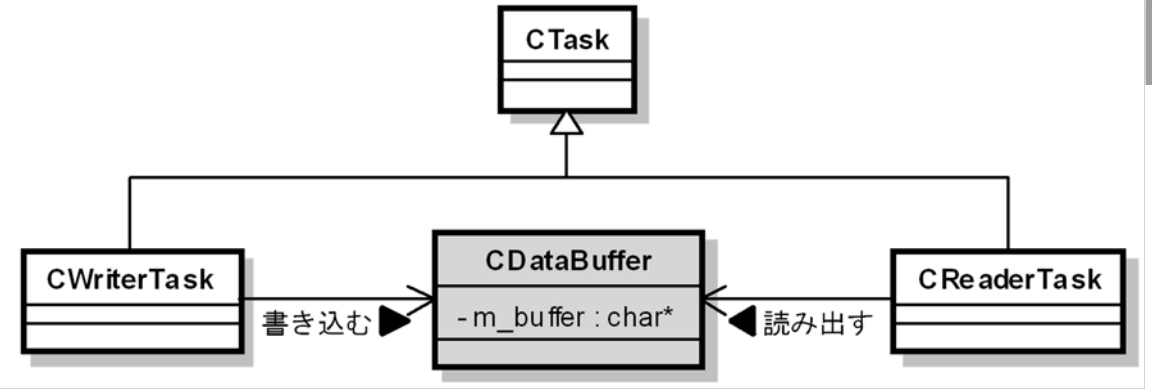


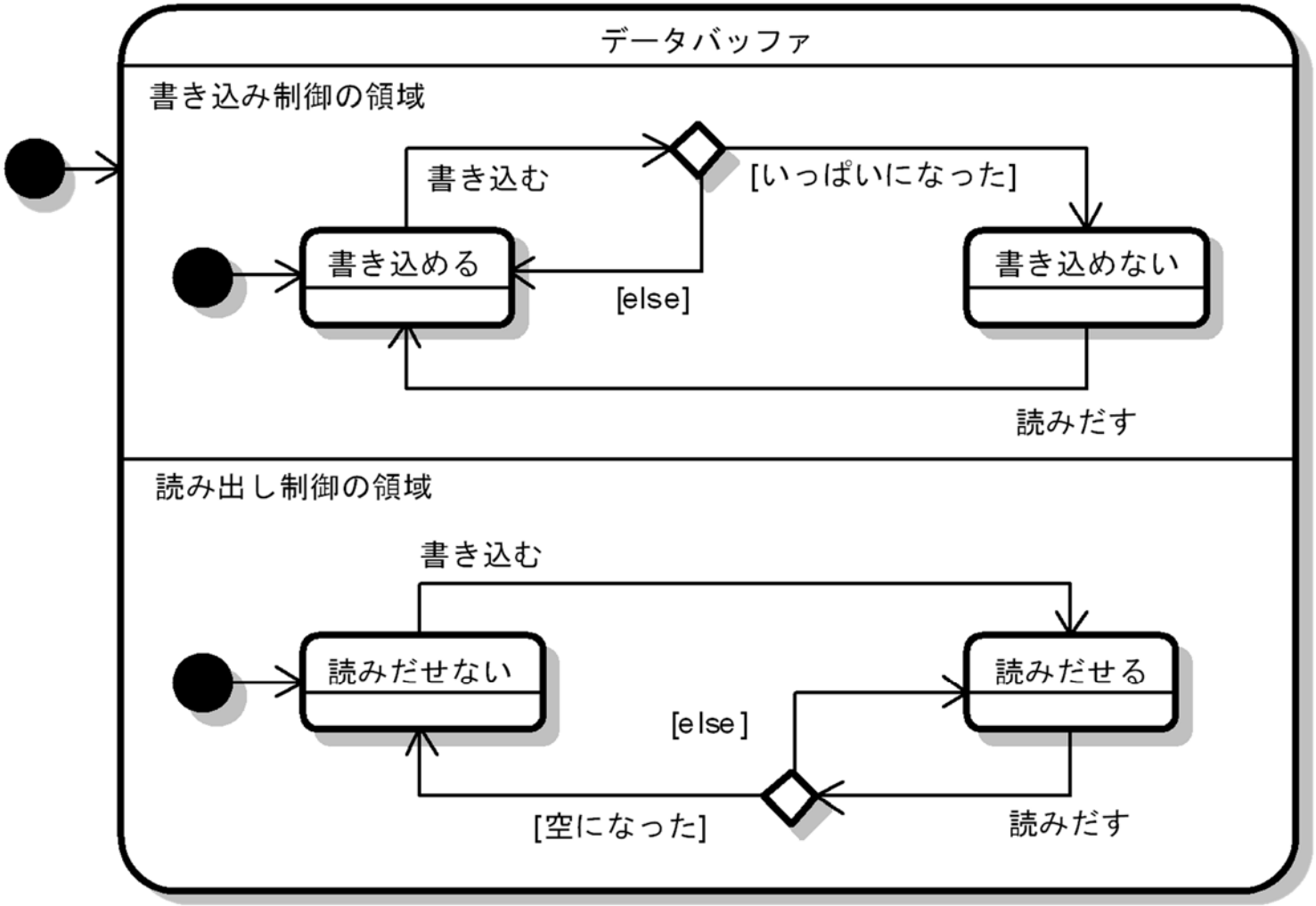


車がN台駐車できる有料駐車場の台数カウンタの状態マシン図です。

外部から届くイベント	evPlus	車が1台入った
	evMinus	車が1台出た
実行してほしいアクション	not_full()	空きあり表示する (発券機にevNotFullを送信する)
	full()	満車表示する　(発券機にevFullを送信する)

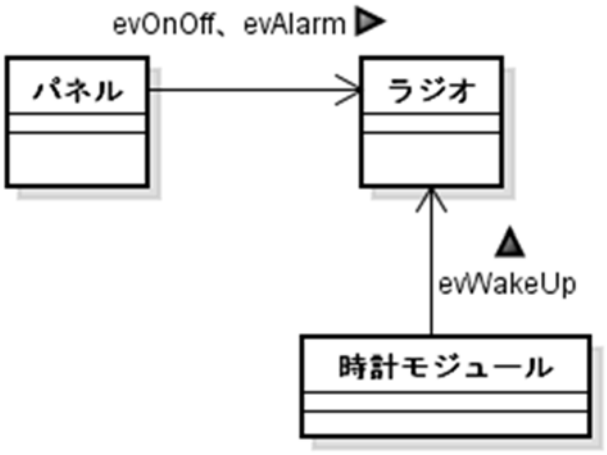
データ送受信

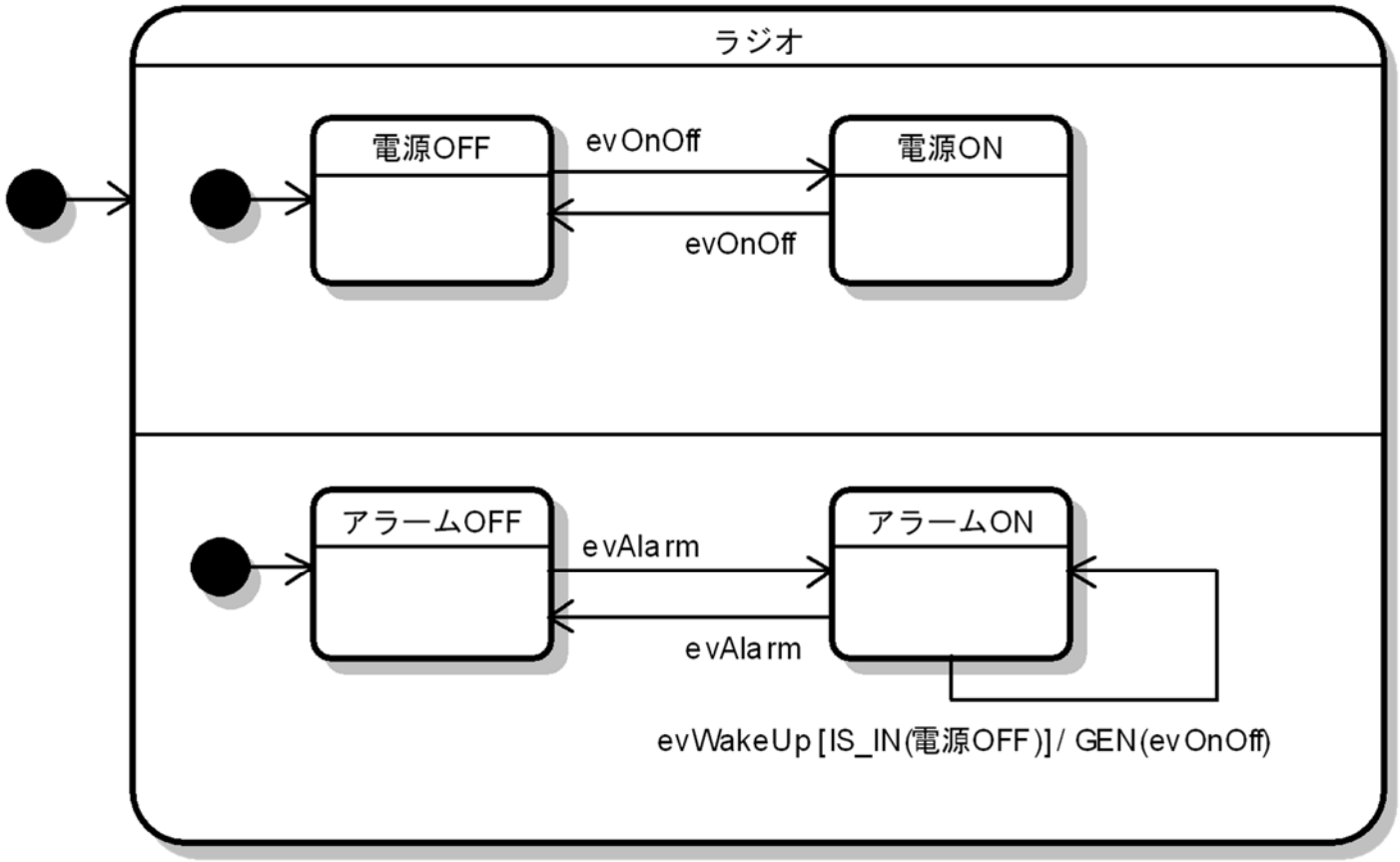

CTask
CWriterTask
書き込む
CDataBuffer
- m_buffer : char*
読み出す
CReaderTask


データバッファ
書き込み制御の領域
書き込む
[いっぱいになった]
書き込める
書き込めない
[else]
読みだす
読み出し制御の領域
書き込む
読みだせない
読みだせる
[else]
[空になった]
読みだす

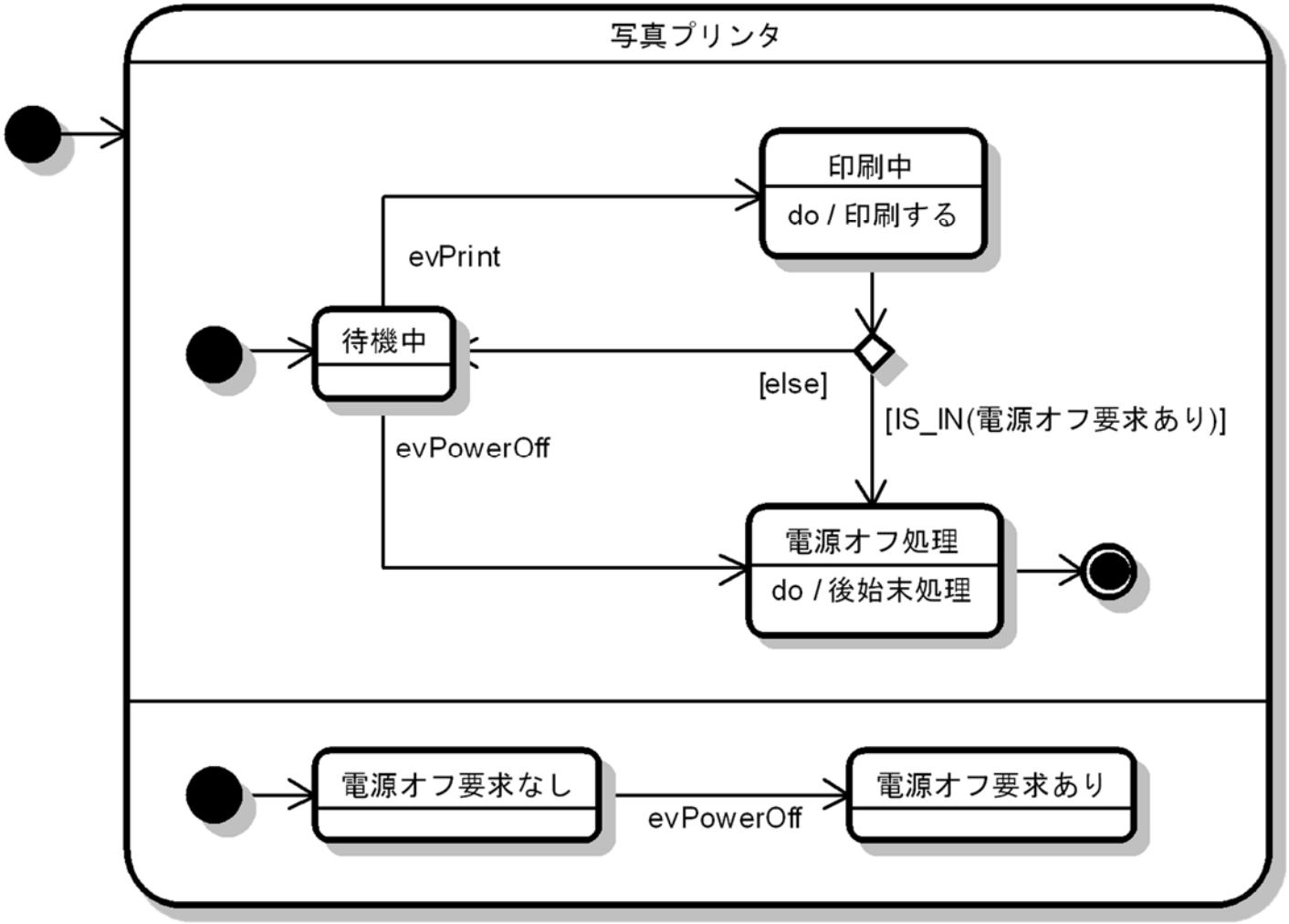
IS_INマクロとGENマクロ (非UML)

- ある状態にいるかどうか判定する IS_IN(状態名)
- 状態マシンにイベントを投入する GEN(イベント名)

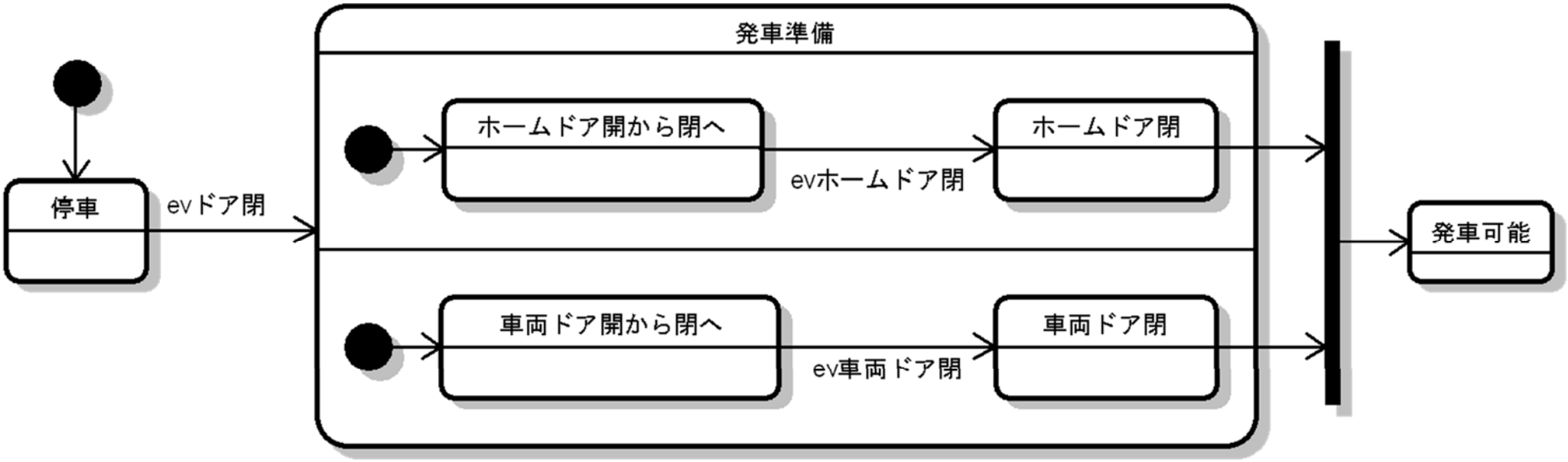




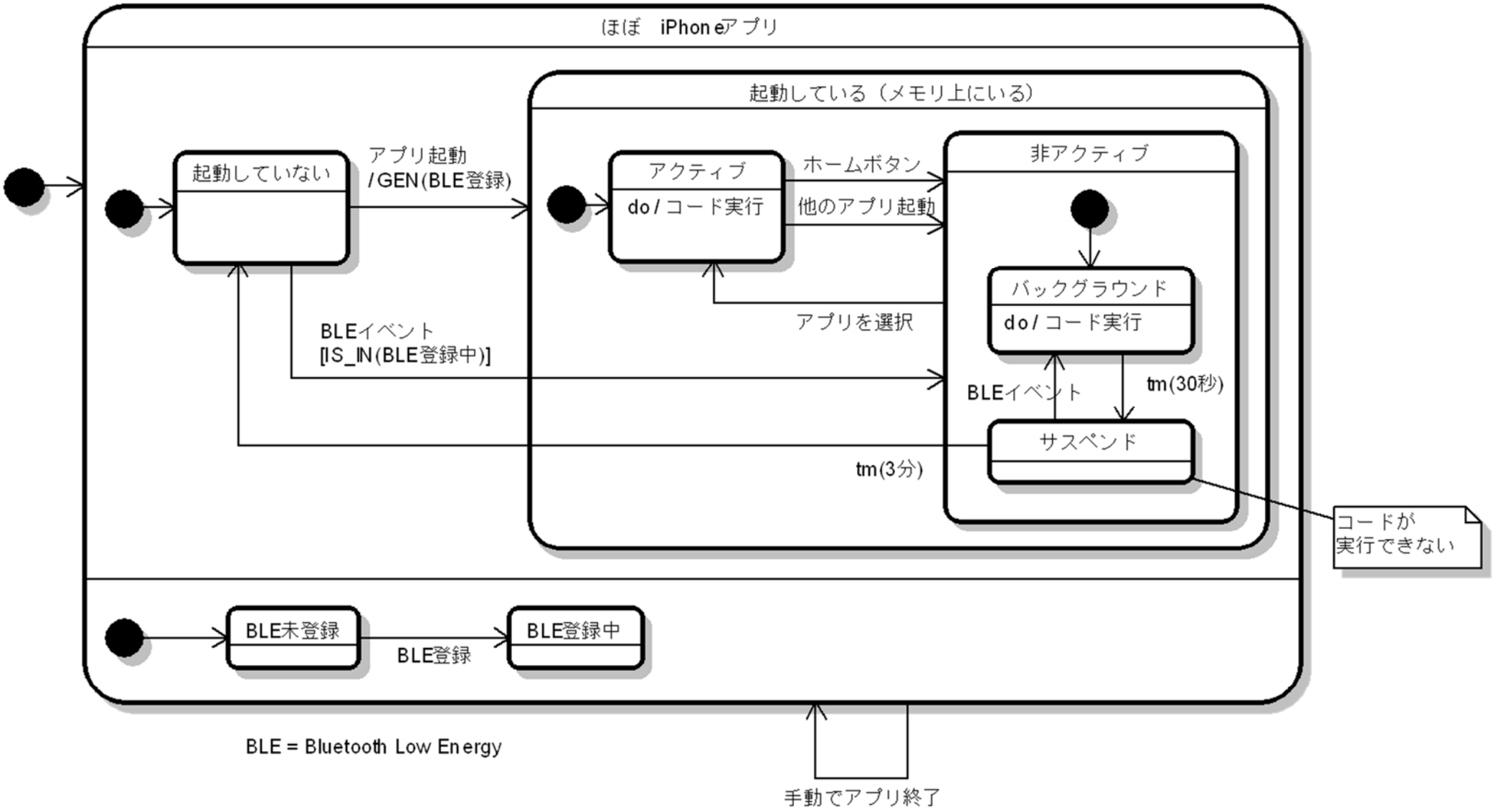
Latch State パターン [Bru2009]



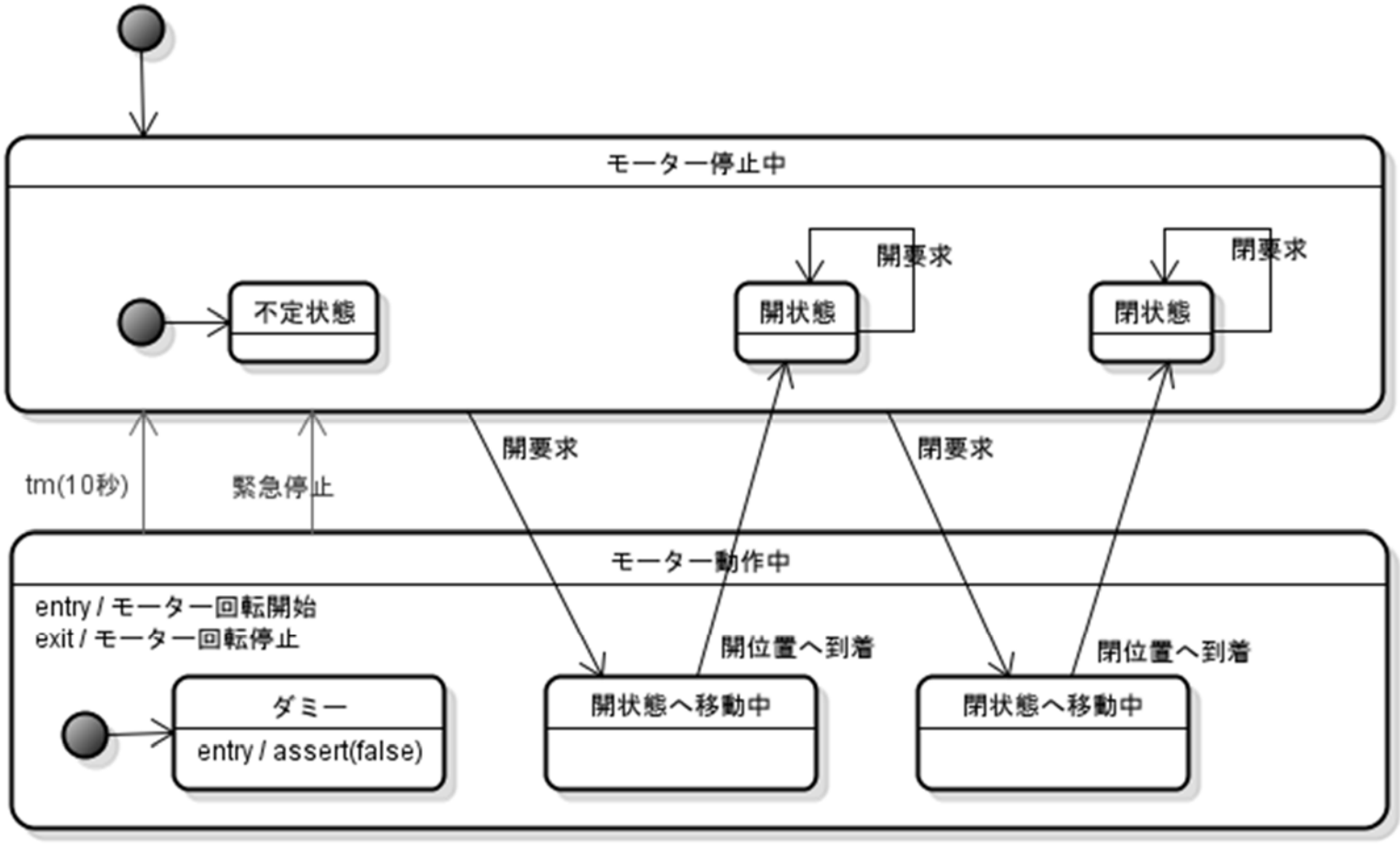
同期をとる、ジョイン疑似状態



ほぼ iPhoneアプリ



ロータリーエンコーダ


モーター停止中
不定状態
開状態
開要求
閉状態
閉要求
tm(10秒)
緊急停止
開要求
閉要求
モーター動作中
entry / モーター回転開始
exit / モーター回転停止
ダミー
entry / assert(false)
開状態へ移動中
開位置へ到着
閉状態へ移動中
閉位置へ到着

履歴を覚える(1)

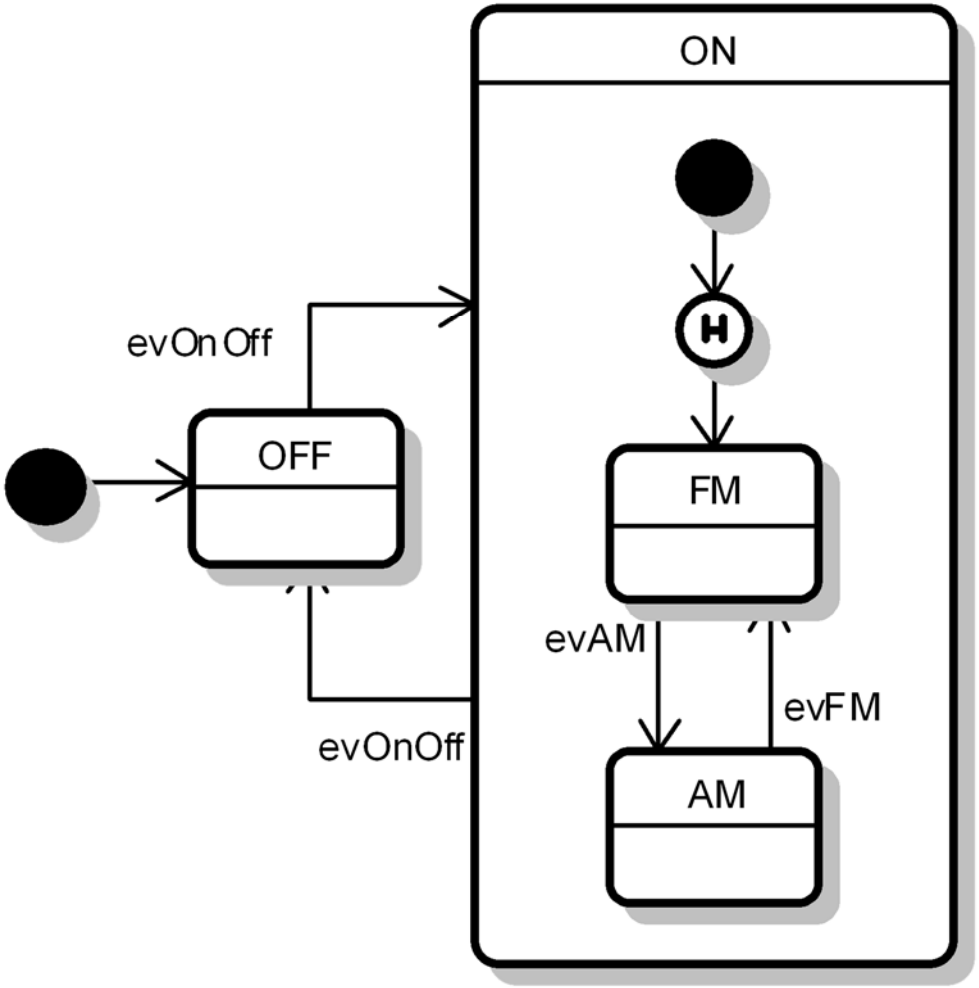


図8

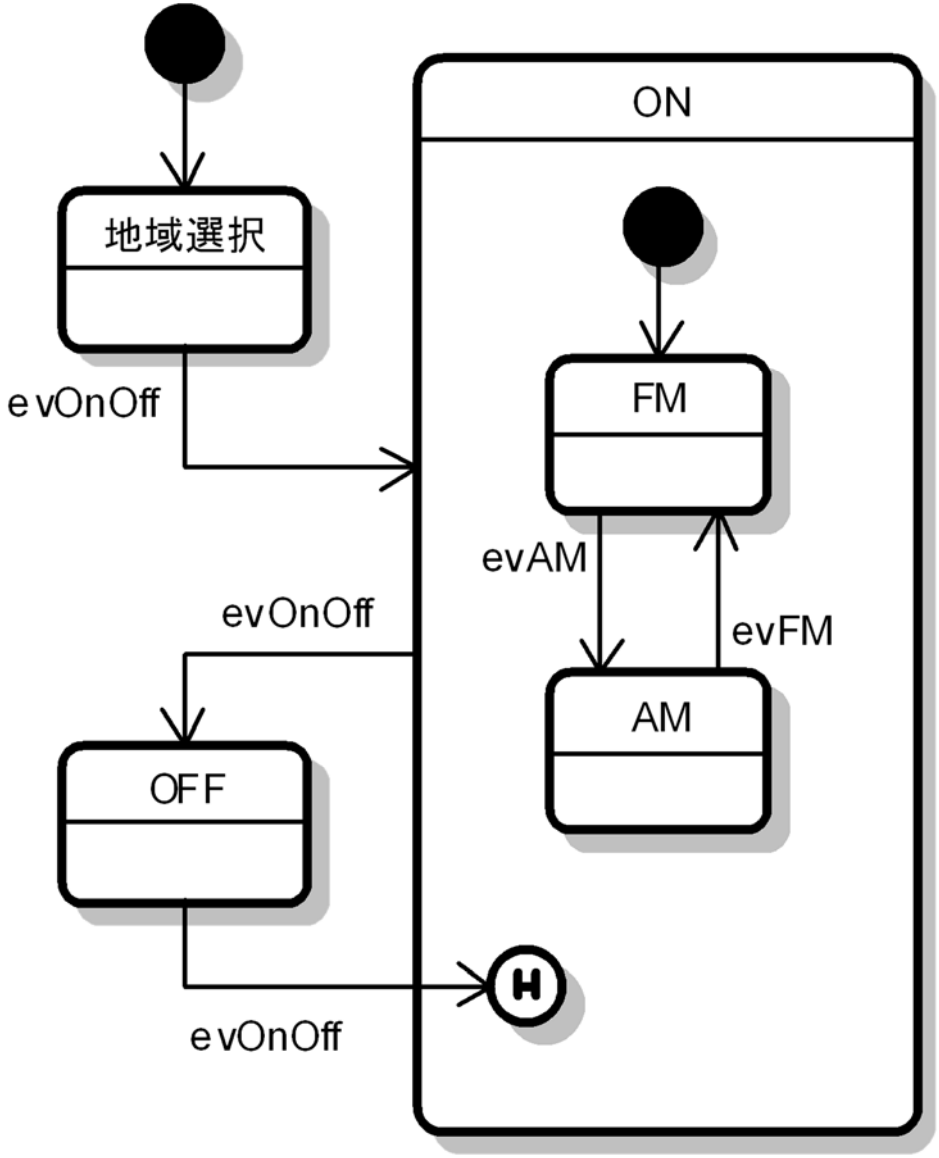
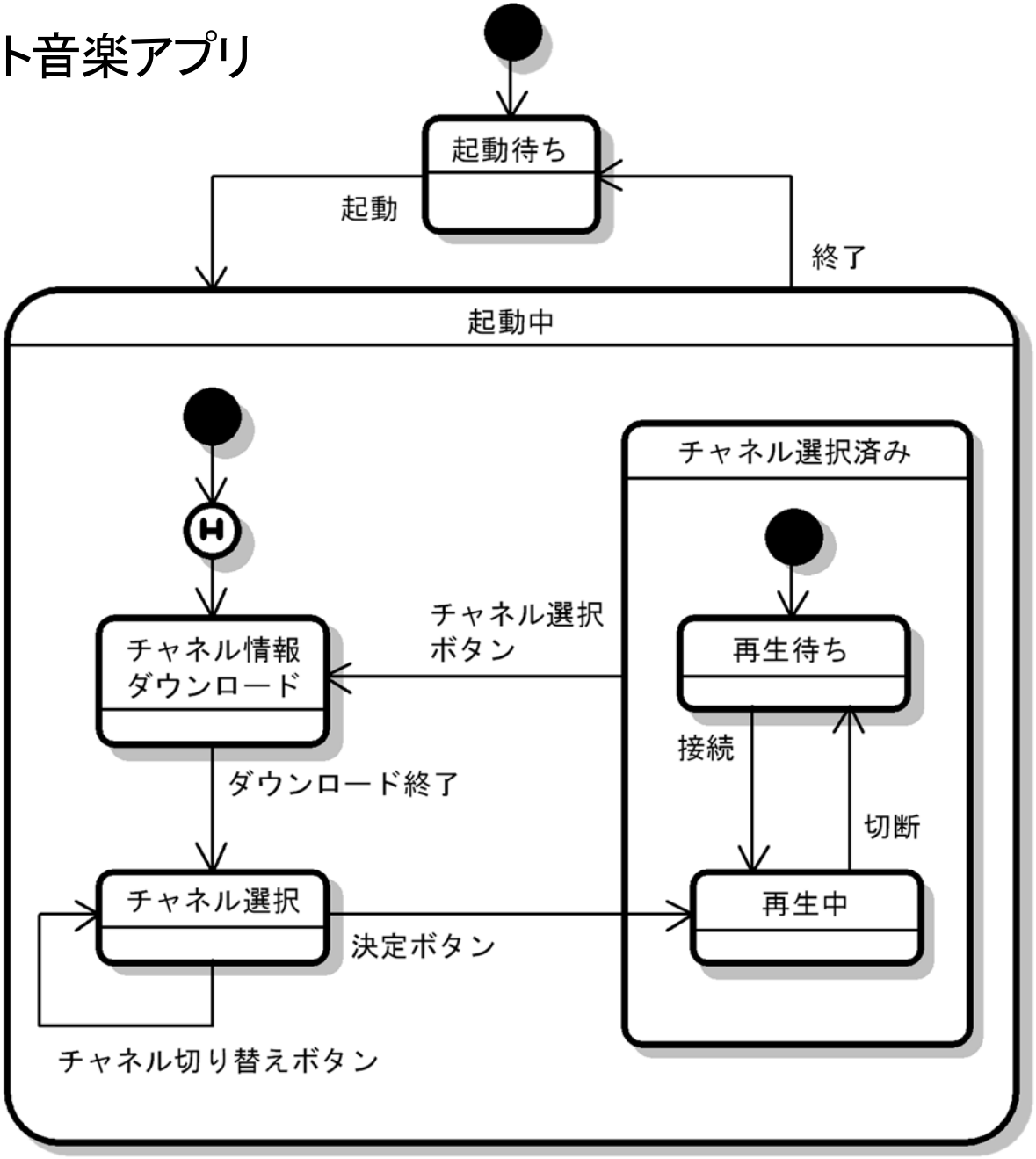


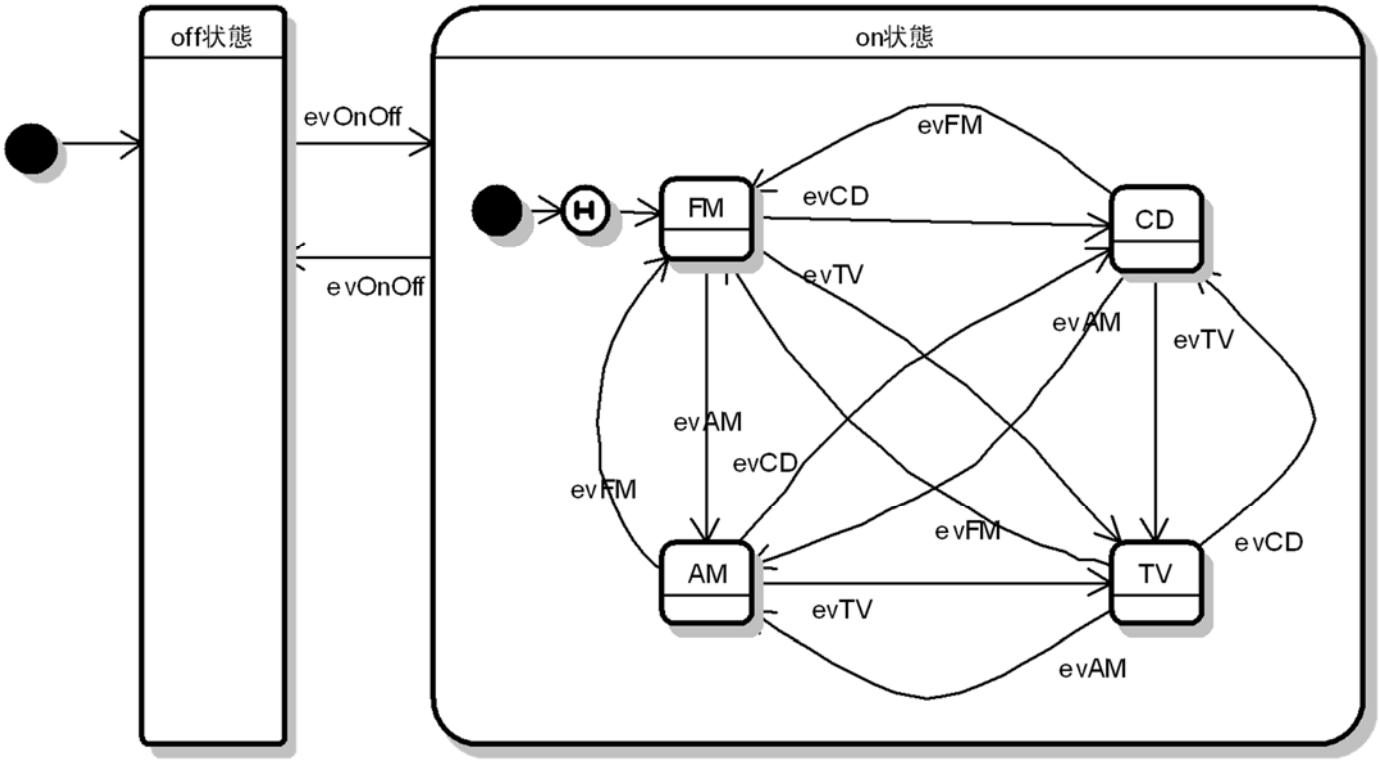
図9

履歴を覚える(2)

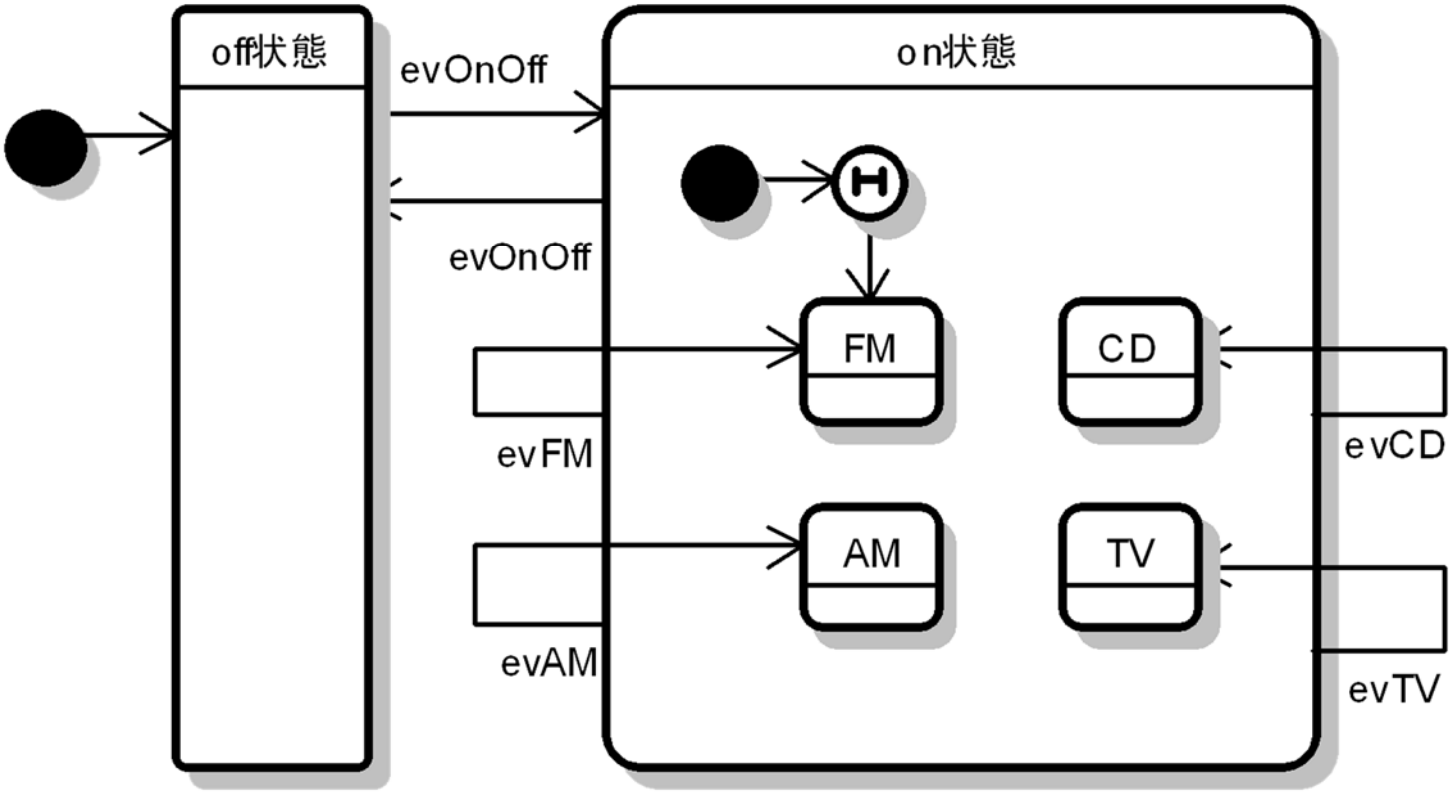
インターネット音楽アプリ



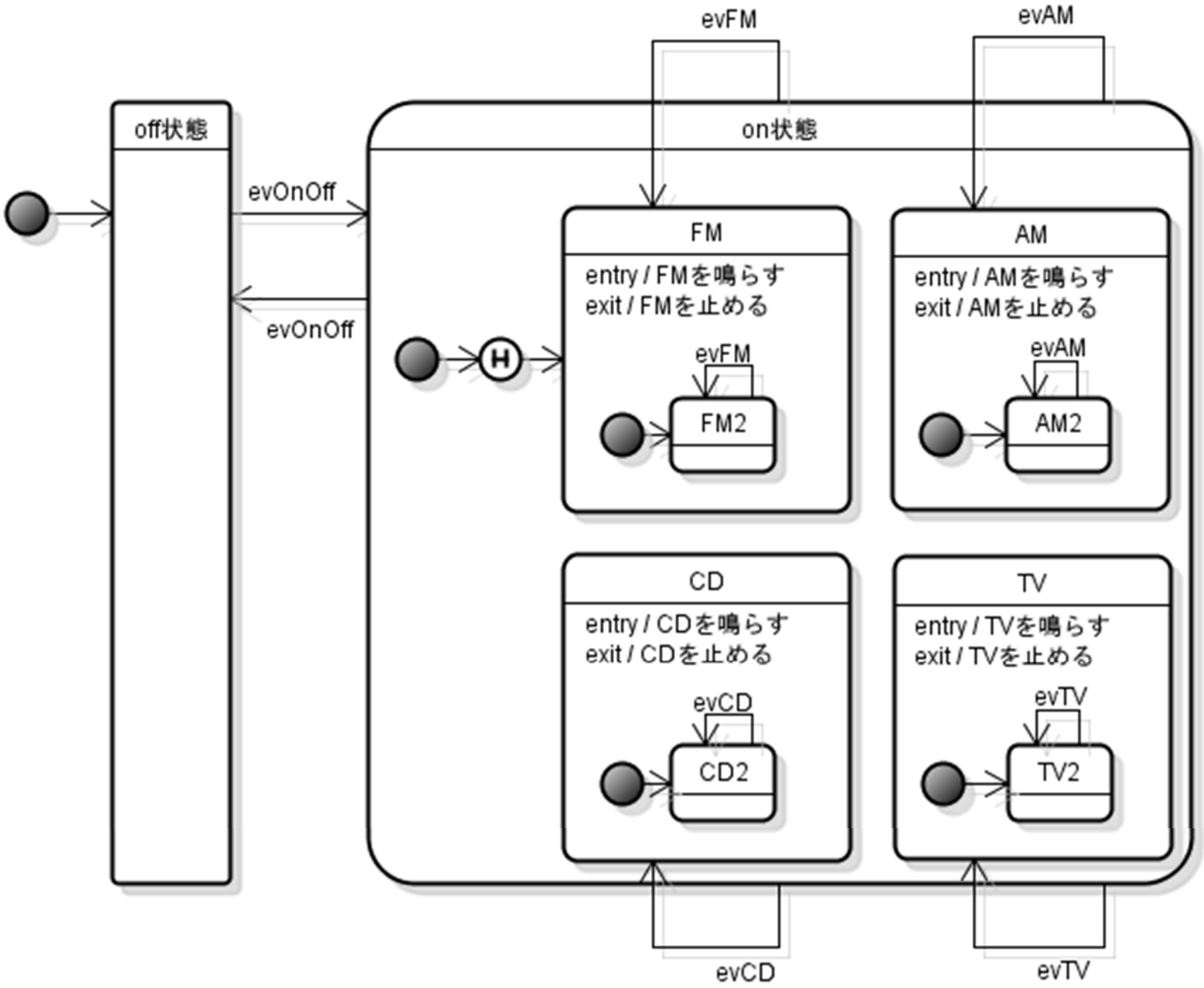
どの音源にも切り替えられるラジオ



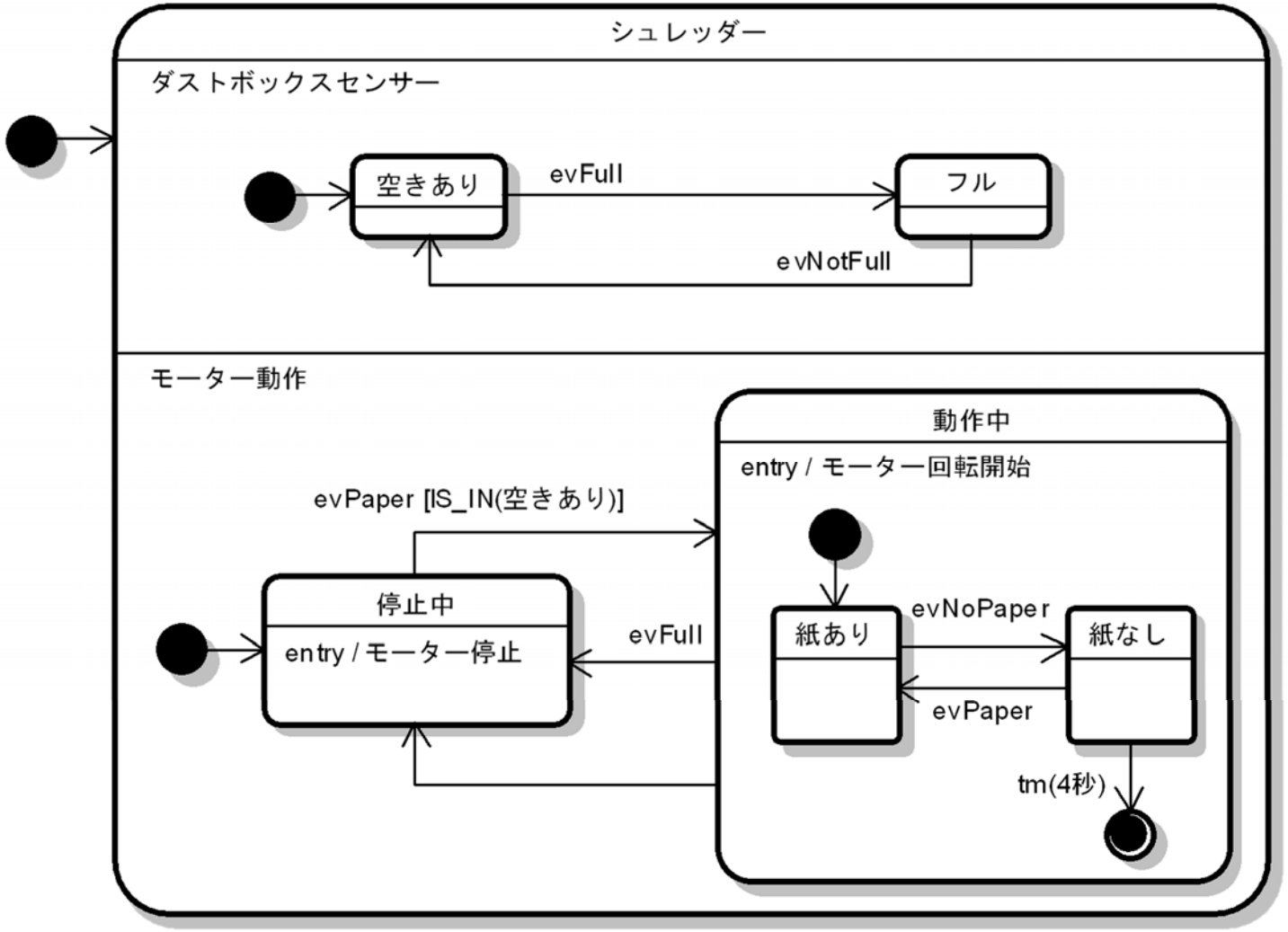
Any Stateパターン [Bru2009]



イベントを無視したい

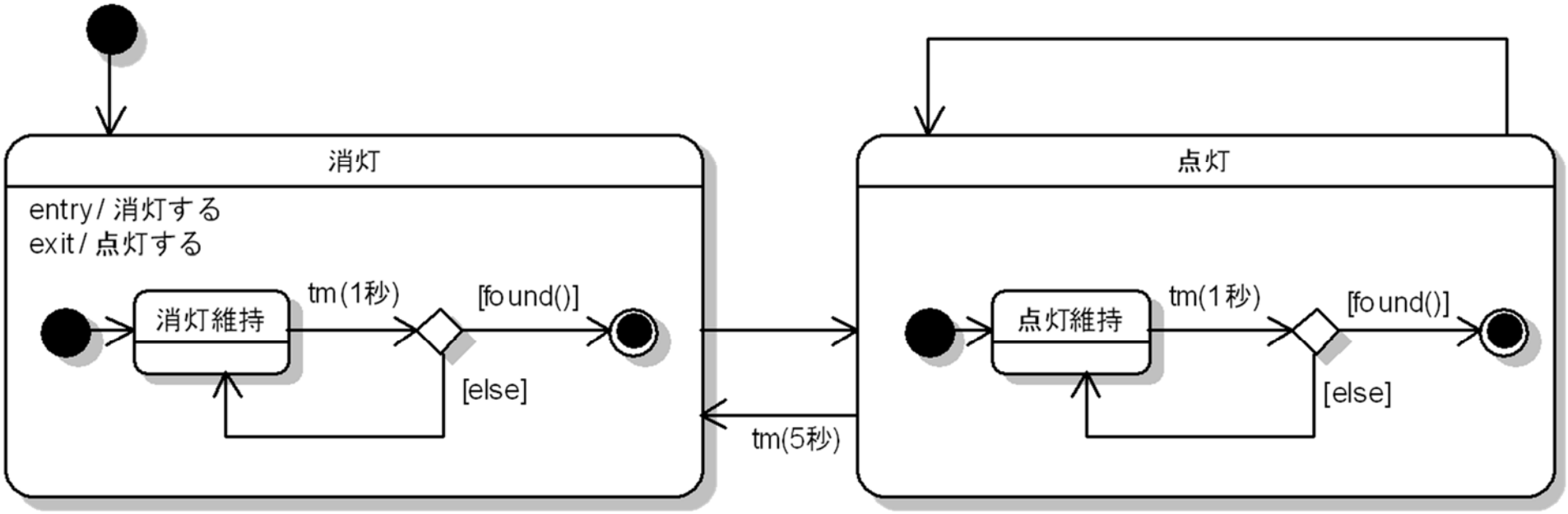

evFM
evAM
off状態
on状態
evOnOff
evOnOff
FM
entry / FMを鳴らす
exit / FMを止める
evFM
FM2
AM
entry / AMを鳴らす
exit / AMを止める
evAM
AM2
H
CD
entry / CDを鳴らす
exit / CDを止める
evCD
CD2
TV
entry / TVを鳴らす
exit / TVを止める
evTV
TV2
evCD
evTV

センサーとアクチュエーター



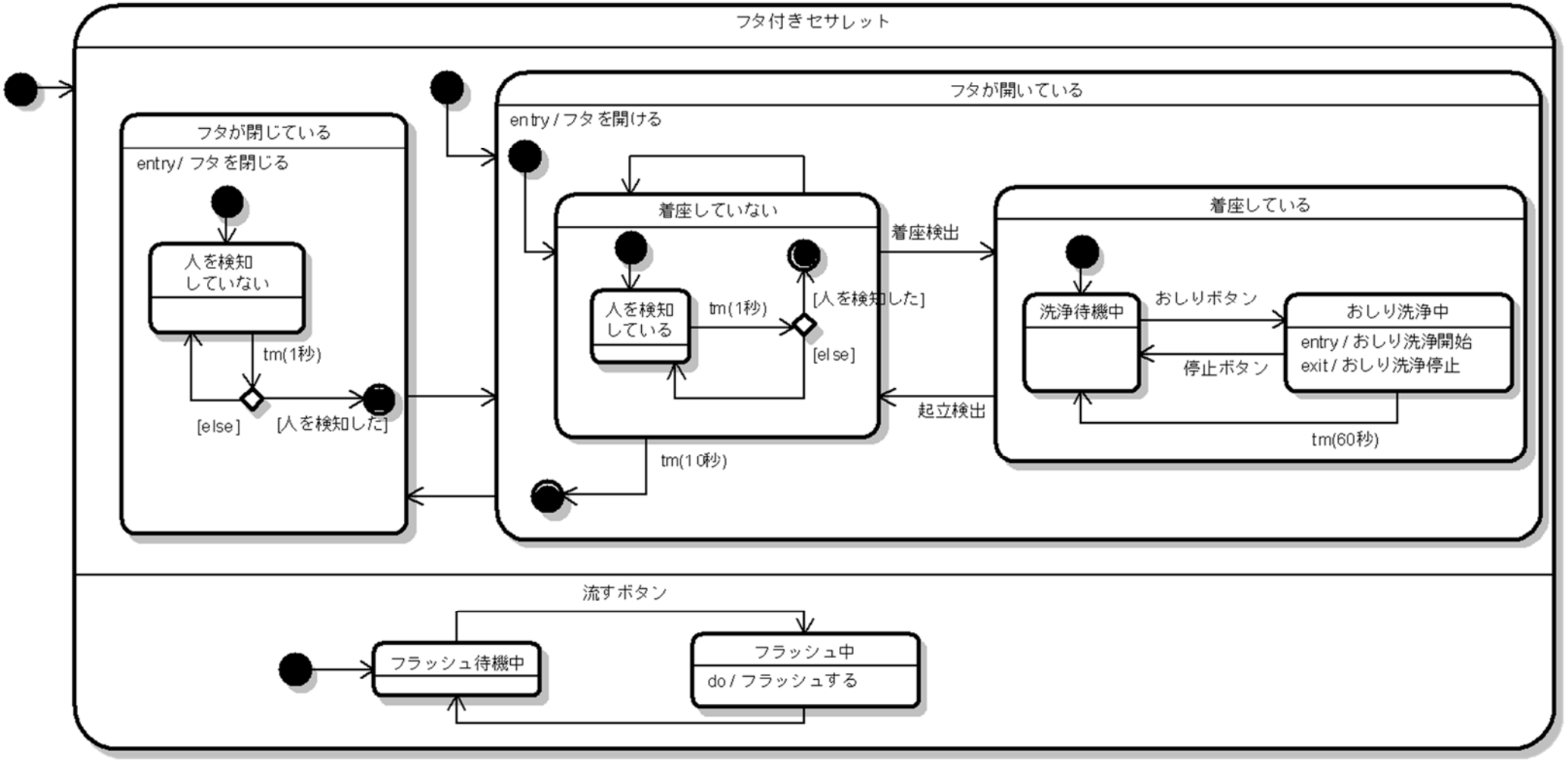
複数のタイマーを同時に待つ

人感センサーライト



・1秒おきに、赤外線を照射して人がいるかどうかを判断する関数 bool found() を呼ぶ
・人がいれば点灯する
・人がいなくなってから5秒たってから消灯する

フタ付きセサレット


フタ付きセサレット
フタが閉じている
entry/ フタを閉じる
人を検知
していない
tm(1秒)
[else]
[人を検知した]
フタが開いている
entry / フタを開ける
着座していない
人を検知
している
tm(1秒)
[人を検知した]
[else]
tm(10秒)
着座検出
起立検出
着座している
洗浄待機中
おしりボタン
おしり洗浄中
entry / おしり洗浄開始
exit / おしり洗浄停止
停止ボタン
tm(60秒)
流すボタン
フラッシュ待機中
フラッシュ中
do / フラッシュする

状態マシンの実装

- 状態マシンは、「今の状態と発生したイベントから、次の状態を決定する」関数のモデル

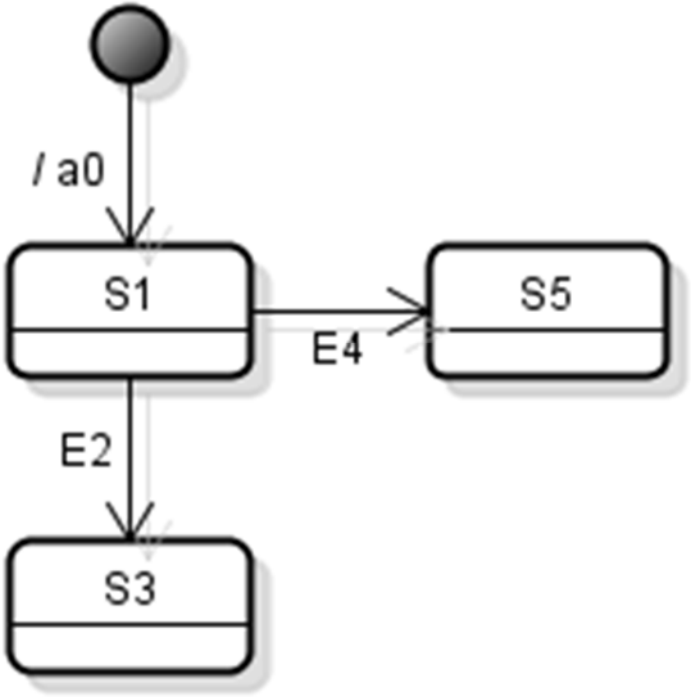
```
次の状態　＝　関数（今の状態，発生したイベント）；
```

- イベントドリブンに動く

```
状態 st = 初期状態;
while(true) {
  イベント e = イベント受信();
  st　＝　イベントハンドラ（st, e）；
}
```

```
typedef enum State { Undef,S1,S2 } State;
static State st = Undef;  // stはスタティック変数にしました。
void main(void) {
  st = S1;
  while(true) {
    Event e = イベント受信();
    HandleEvent(e);
  }
}
```

単純状態の実装

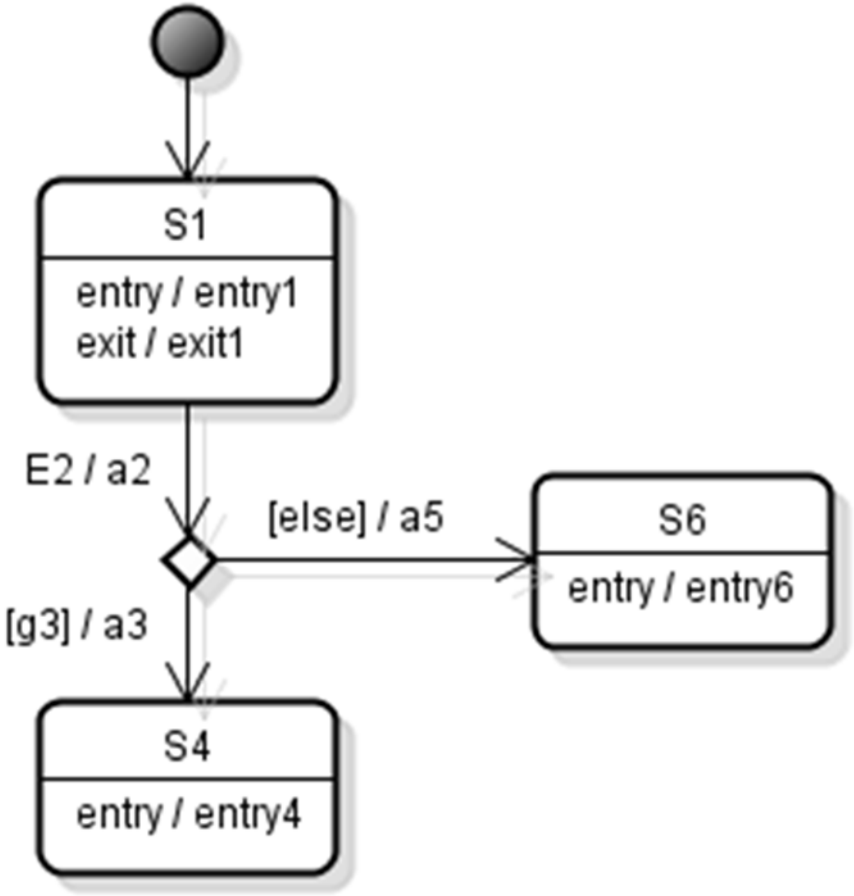



```
typedef enum Event { E2,E4 } Event;
typedef enum State {Undef,S1,S3,S5} State;

static State st = Undef;

void main(void) {
  a0;
  st = S1;
  while(true) {
    Event e = イベント受信();
    HandleEvent(e);
  }
}
void HandleEvent(Event e) {
  switch(st) {
    case S1:
      if (e == E2) { st = S3; }
      else if (e == E4) { st = S5; }
      break;
    case S3: break;
    case S5: break;
  }
}
```

選択疑似状態




```
void HandleEvent(Event e) {
  switch(st) {
    case S1:
      if (e == E2) {
        exit1;
        a2;
        if (g3) {
          a3;
          st = S4;
          entry4;
        } else {
          a5;
          st = S6;
          entry6;
        }
      }
      break;
    case S4: break;
    case S6: break;
  }
}
```

参考文献

- [Shmz2010]「要求を仕様化する技術・表現する技術　改訂第2版」,清水吉男,2010
- [GoF1995]「オブジェクト指向における再利用のためのデザインパターン」,1995
- [Yuki2004]「増補改訂版 Java言語で学ぶデザインパターン入門」,結城浩,2004
- [Dwm2005] 組み込みソフトウェア開発スタートアップ Design Wave Magazine,2005
- [Hanai2013] モダンC言語プログラミング 花井志生,2013
- [Ses2006]「組込みソフトウェア開発のための オブジェクト指向モデリング」,SESSAME WG2,2006
- [Omg2006]「UML2 仕様書 2.1対応」,Object Management Group,2006
- [Ogi2006]「その場でつかえる しっかり学べる UML 2.0」,オブジェクトの広場,2006
- [Ogi2013]「かんたんUML入門」、オージス総研監修,2013
- [Sco2006]「UML 2 スタイルガイドブック」,Scott W.Ambler,2006
- [Bru2001]「リアルタイムUML第2版」,Bruce Douglass,2001
- [Bru2009]「リアルタイムUMLワークショップ」,Bruce Douglass,2009
- [Ino2011]「ダイアグラム別UML徹底活用 第2版」,井上樹,2011
- [Tam2004]「ソフトウェア工学の基礎」,玉井哲雄,2004
- [Tom2014]「アンダースタンディング コンピュテーション 」,Tom Stuart,2014

Shmz2010 では、要求を仕様に分解する手順、状態遷移とリンクした仕様化の話しがでてきます。

GoF1995と Yuki2004による、ステートパターンの議論は必読です。ステートパターンは、状態マシンの実装が困難なところに着目して考え出されたパターンです。その議論を読めば、状態マシンの実装がステートパターンとそれ以外の実装でどう違っていて、何が問題として残っているのかがわかります。

Dwn2005とHanai2013には、C言語による状態マシンの実装の話しが出てきます。

Omg2006は、OMGのUML2.0仕様を和訳したもの。訳は正確だと思います。

Ogi2006と、Ogi2013は、座右においてUMLの表記法をチェックするときに便利に使えます。必携です。

Sco2006は、コーディングに対するコーティングルール本のように、UMLモデル作成に対する描き方ルールが箇条書きで載っている本です。ルール数はちょっと少ないですが、セルフレビューに使えます。

Bru2009は、これらの本の中でもっとも複雑なレベルの状態マシンまで詳しく載っていて学ぶところが多いです。Any StateパターンやLatch Stateパターンはこの本に載っています。

Ino2011は、静的条件分岐を使った遷移と、動的条件分岐を使った遷移の違いが説明されています。状態遷移に伴うアクティビティ、ステートマシンの特化、プロトコルステートマシンにも少しずつふれられています。

Tam2004 は、ソフトウェア工学の基礎を学んだことがないひとにおすすめ。紙の本は絶版ですが、PDFが法政大学の著者のページで公開されています。

Tom2014は、2014年に出た本の中でおすすめの一冊。有限オートマトンをRubyのプログラムを動かしながら理解するというとても面白い本です。学術的な説明はよくわからないけどプログラムを読むのは得意、という人におすすめです。

SESSAME コンテンツのご利用に際して

- 本著作物の著作権は作成者または作成者の所属する組織が所有し、著作権法によって保護されています
- SESSAMEは本著作物に関して著作者から著作物の利用※を許諾されています
- 本著作物はSESSAMEが利用者個人に対して使用許諾を与え、使用を認めています
- SESSAMEから使用許諾を与えられた個人以外の方で本著作物を使用したい場合は query@sessame.jp までお問い合わせください
  - ※ SESSAMEが著作者から許諾されている権利
    著作物の複製・上演・演奏・公衆送信及び送信可能化・口述・展示・上映及び　頒布・貸与・翻訳・翻案・二次的著作物の利用


SESSAME