- これまで開発したロボットの提供する機能を部品として、新たな機能を提供するロボットを開発します
- 既存の部品をうまく使って開発してみましょう

7. 演習3）これまでの部品を使って新たな機能を実現する

7-1. 演習課題と進め方

- 演習1、2で作った機能を使って、不特定の時間で「ライントレース」と「シナリオトレース」をランダムに切り替えて走行する「ランダムウォーカー」を作りましょう
  - 走行体は、演習1、2と同様のものを使います
  - 切り替えは、乱数を使った不特定の時間を使って行います
    - ライン上でない場所でライントレースに切り替った場合は、ラインを見つけることを期待して、そのまま動作させてください

- 演習1、2と同様に「モデルと部品を使った開発」の流れに沿って開発します
  - モデルを作成します
  - ソースコードを実装します
  - 実機での動作を確認します

7-2. 演習の準備（1）

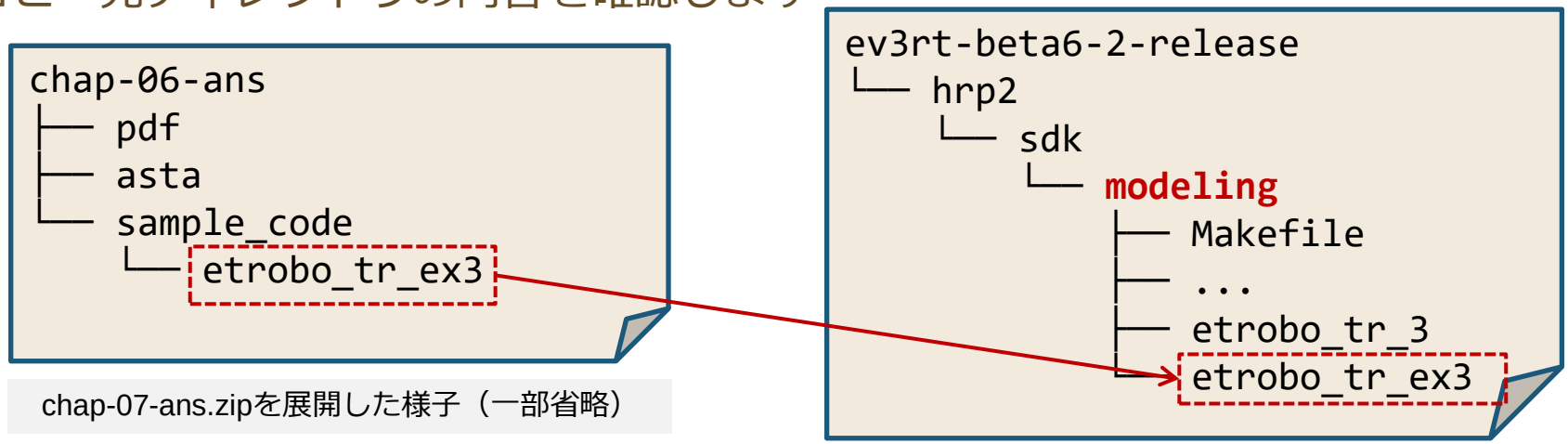
- 7章の解凍用パスワードを取得します
  - 圧縮ファイル解凍用のパスワードを取得してください
- 7章の演習後に使う配布ファイル
  - ファイル名：chap-07-ans.zip
  - 7章の演習のモデルとコードの解答例が含まれています
- 圧縮ファイルchap-07-ans.zipを展開します
  - 展開したディレクトリの中身を確認しましょう

```
chap-07-ans
├── pdf                          テキストのディレクトリ
│   └── chap-07-opened.pdf           7章のテキスト（解答例つき）
├── asta                         モデル図のディレクトリ
│   └── etrobo_tr_ex3.asta           演習3のサンプルモデル
└── sample_code                  サンプルコードのディレクトリ
    └── etrobo_tr_ex3/*              演習3のサンプルコード
```

「etrobo_tr_2」に対応した解答例が
「etrobo_tr_ex2」です

7-2. 演習の準備（2）

- モデル図の確認
  - 展開したディレクトリの「asta」ディレクトリにある「etrobo_tr_ex3.asta」が解答例のモデル図です
- ソースコードの確認
  - 展開したディレクトリの「sample_code」ディレクトリにある「etrobo_tr_ex3」が解答例のコードです
  - EV3RTをインストールしたディレクトリの中に「hrp2¥sdk」の中の「modeling」ディレクトリの中へ「etrobo_tr_ex3」をコピーします
  - コピー先ディレクトリの内容を確認します

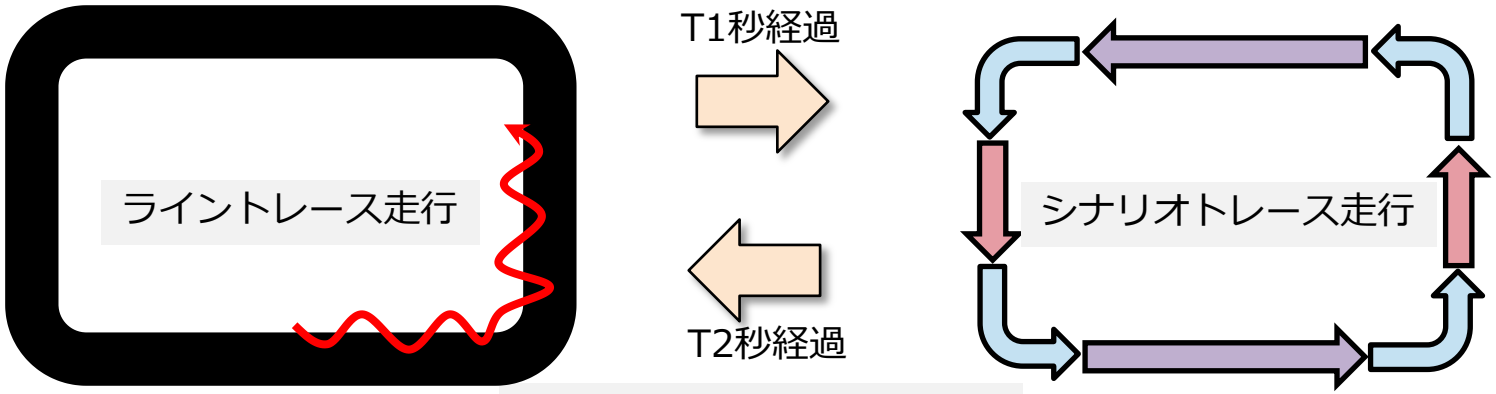


chap-07-ans.zipを展開した様子（一部省略）

7-2. ランダムウォーカーの動作

- 「ランダムウォーカー」は、「ライントレース」と「シナリオトレース」を不特定の時間で切り替えるロボットです
- ランダムウォーカーの走行手順
  - スタートボタンが押されるまで待つ
  - デバイスやライブラリを初期化する
  - 最初は、ライントレースを開始する
    - 乱数を使って次の切り替え時間を設定する
  - 切り替え時間が経過してたら、シナリオトレースを開始する
    - 乱数を使って次の切り替え時間を設定する
  - 切り替え時間が経過したら、手順3の動作に戻り、以降手順3～4を繰り返す

「ライントレース」と「シナリオトレース」を切替えることを、「走行モードの切替え」と呼ぶことにします。

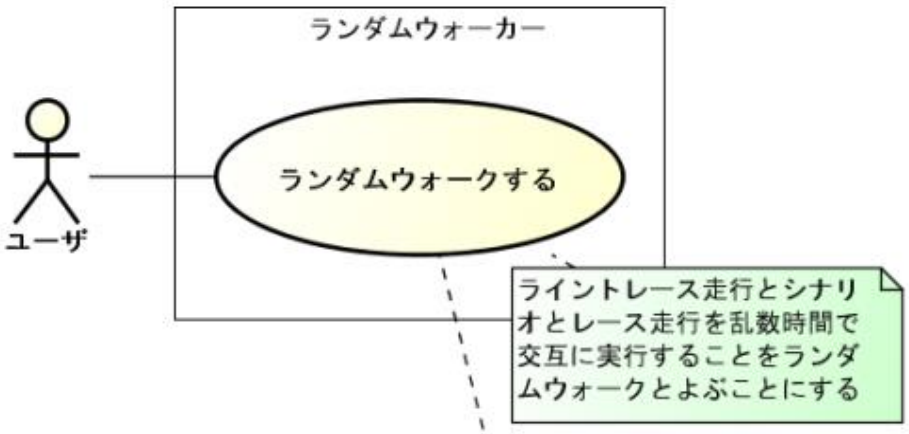


ランダムウォーカーの動作の概要

7-3. システムが提供する機能を決める（１）

要求分析

- アクタはユーザのままです
- ユーザに提供する機能は変わりますね
- 演習2のユースケース図を元に修正してみましょう
  - ユーザに提供する機能は演習2とは別のものになります
  - ユースケース名を「シナリオ走行とイントレース走行を交互に実行する」としましょう
- 作成したユースケース図は右のようになりました



ランダムウォーカーの機能
（ユースケース図）

要求モデル・機能「シナリオトレーサの機能」を開いて、図の名前を「ランダムウォーカーの機能」に変更し、ユースケース図を書き直しましょう（システムの名前が変わっています）

7-3. システムが提供する機能を決める（2）

要求分析

- ユースケースの記述を要求に合わせて記述してみましょう
- ユースケース記述も見なおしてみましょう
  - ユーザに機能を提供するための手順を整理します
  - 一部の動作は、演習1や演習2で作成したものが使えそうです
  - シナリオを使うための手順や動作は新しく作成する必要があります
- 作成したユースケース記述は、右のようになりました

要求モデル・機能「ランダムウォーカーの機能」を開いて、ユースケース図に書いてあるユースケース記述を書き直しましょう

ユースケース「ランダムウォークする」の基本ユースケース

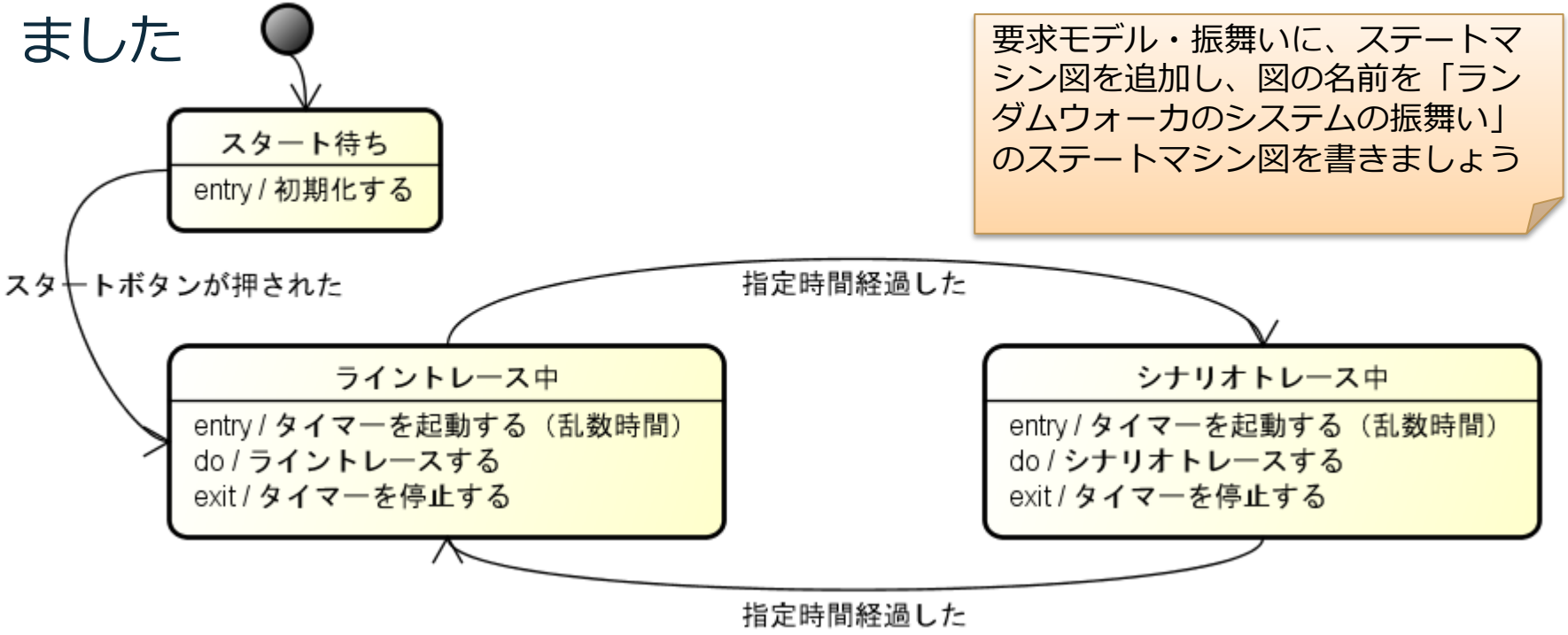
1. ユーザがロボットを起動すると、ロボットは動作を開始する
2. ロボットは動作を開始すると、デバイスやライブラリを初期化し、走行開始の指示を待つ
3. ユーザは、スタートボタン（タッチセンサ）を押してロボットに走行開始を指示する
4. ロボットは、ボタンが押されるとライントレース走行する
5. ロボットは、ライントレース走行し、ランダムな時間経過するとシナリオトレース走行に切り替える
6. ロボットは、シナリオトレース走行し、ランダムな時間経過するとライントレース走行に切り替える
7. 手順5～6を繰り返すことでランダムウォークする

7-4. 機能の実現方法を考える

要求分析

- この演習では、システム全体の振舞いを表すためにステートマシン図を使ってみましょう
  - ランダムな時間で走行方法を切り替える動作に着目して、「何が起きるのを待って」動作するのかを考えて整理するとよいでしょう
- 作成したシステム全体のステートマシン図は右のようになりました

要求モデル・振舞いに、ステートマシン図を追加し、図の名前を「ランダムウォーカのシステムの振舞い」のステートマシン図を書きましょう



ランダムウォーカーのシステムの振舞い（ステートマシン図）

7-5. 機能実現に必要な部品を見つける

基本設計

- 部品の候補を考えて表を作ってみましょう
  - ライントレーサとシナリオトレーサが使われている
  - 指定時間経過したことを調べる
  - 指定時間は固定時間ではなくランダムな時間を設定する
  - 演習1や演習2で使用したスタータが使えそう

- 新しく作った表は、下のような表になりました

「働き」や「情報」	部品の候補
ライントレース走行する	ライントレーサ
シナリオトレース走行する	シナリオトレーサ（スタータなし）
スタート機能	スタータ
指定時間経過したことを調べる	経過時間を計測する（タイマー）
ランダムな値を得る	乱数発生関数（ライブラリ関数）

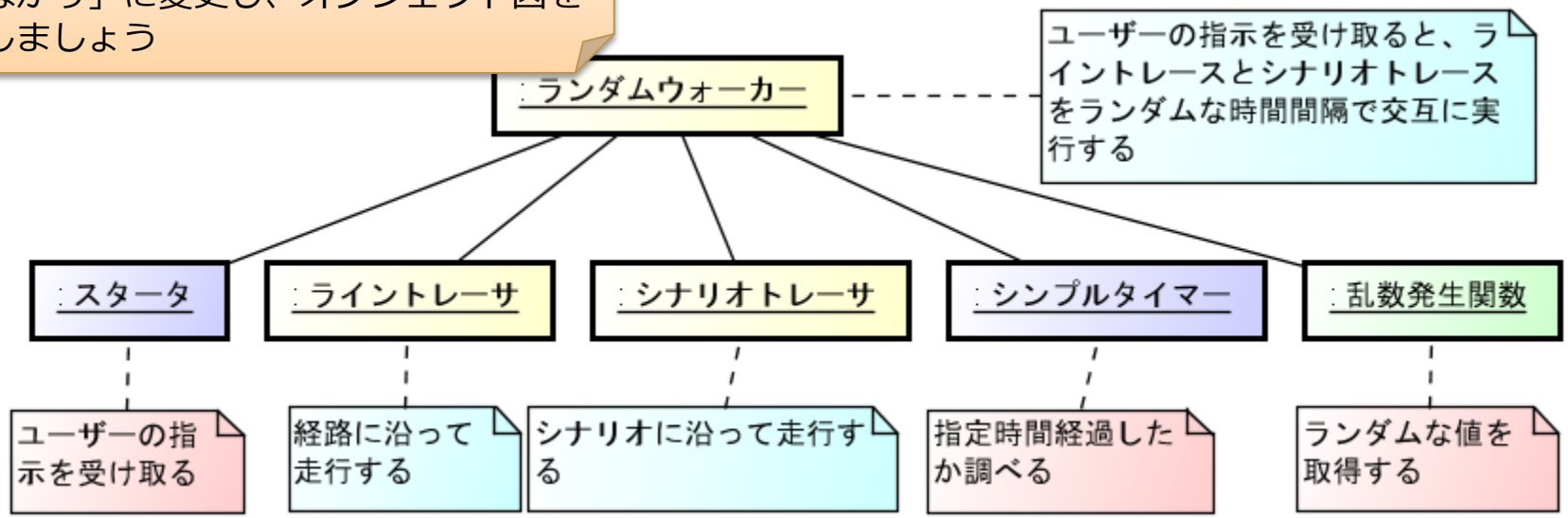
ランダムウォーカーの働きや部品の候補

7-6. 新しい部品を定義する

基本設計

- 演習1、演習2で作成した部品を使って「ランダムウォーク機能」を実現するために必要な部品を組合わせます
  - ライントレーサ、シナリオトレーサ、スタータ、タイマーを追加しましょう
  - 乱数発生関数を追加します
- 新しく作ったオブジェクト図は、下のような図になりました

設計モデル・構造「スタータつきシナリオトレーサの部品の候補のつながり」を開いて、図の名前を「ランダムウォーカーの部品の候補のつながり」に変更し、オブジェクト図を書き直しましょう



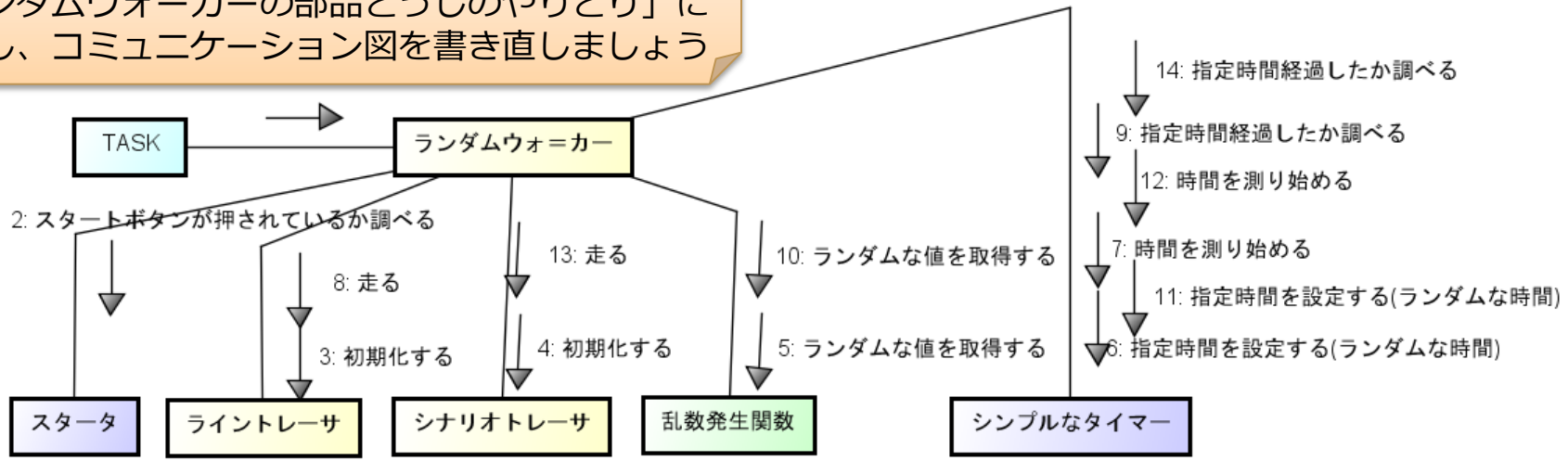
ランダムウォーカーの部品の候補のつながり
（オブジェクト図）

7-7. 部品による機能実現を確認する

基本設計

- 「ランダムウォーク機能」を実現するのに必要なやりとりを書きます
  - ライントレーサ、シナリオトレーサ、スタータ、タイマーを追加しましょう
  - 乱数発生関数を追加します
  - 候補を探したときに得た働きを参考にやりとりに使うメッセージを追加しましょう
- 新しく作ったコミュニケーション図は、下のような図になりました

設計モデル・振舞い「スタータつきシナリオトレーサの部品どうしのやりとり」を開いて、図の名前を「ランダムウォーカーの部品どうしのやりとり」に変更し、コミュニケーション図を書き直しましょう



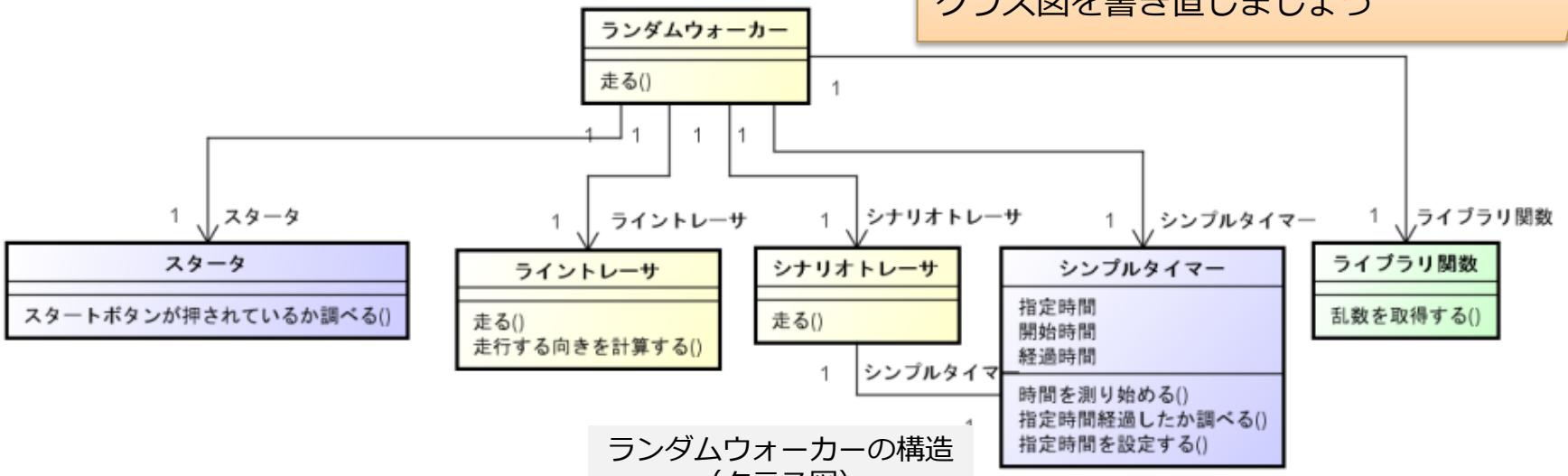
ランダムウォーカーの部品どうしのやりとり
（コミュニケーション図）

7-8. システムの構造を決定する

基本設計

- 「ランダムウォーク機能」を実現するのに必要なクラスを書きます
  - ライントレーサ、シナリオトレーサ、スタータ、タイマーを追加しましょう
  - 乱数発生関数を追加します（OSのライブラリ関数としましょう）
  - 関連、多重度、関連端名をつけましょう
- 新しく作ったクラス図は、下のような図になりました

設計モデル・構造「スタータつきシナリオトレーサの構造」を開いて、図の名前を「ランダムウォーカーの構造」に変更し、クラス図を書き直しましょう

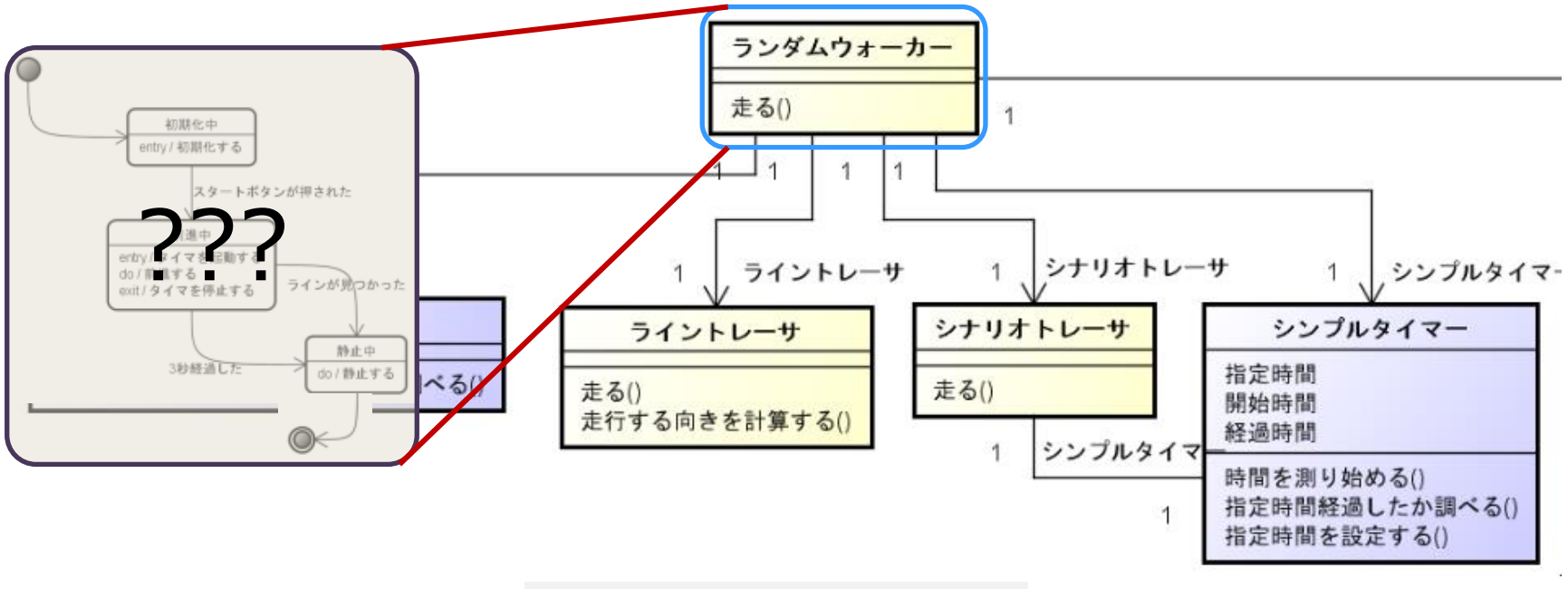


ランダムウォーカーの構造（クラス図）

7-9. システムの振舞いを決定する（1）

基本設計

- ランダムウォーカークラスの振舞いを書いてみましょう
- 演習1、演習2で作成したステートマシン図を参考に、ステートマシン図を書いてみましょう
  - 7-5で作成した表などを参考に、起きるのを待っているできごとを探して、状態の候補を見つけましょう

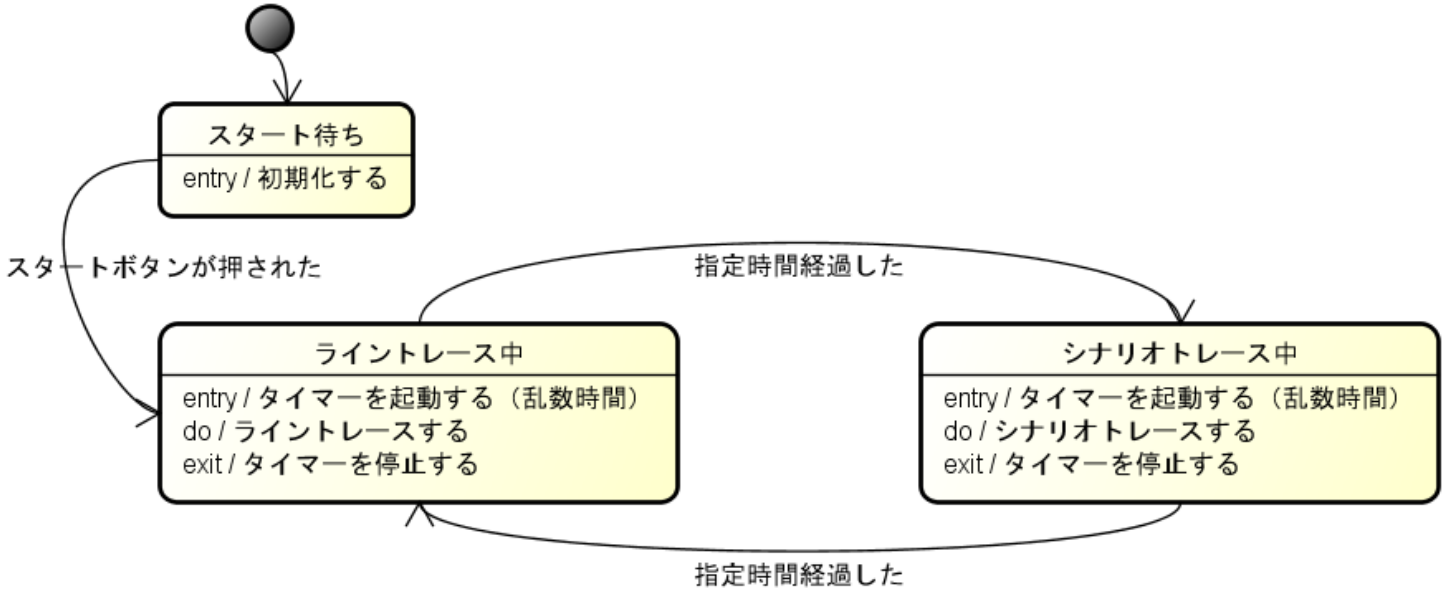


ランダムウォーカーの振舞いを作る
（クラス図とステートマシン図）

7-9. システムの振舞いを決定する（2）

基本設計

- ランダムウォーカーの「走る」の振舞いを作成しましょう
  - ステートマシン図を作成しましょう
  - 詳細な処理はライントレーサ、シナリオトレーサに任せていることから、7-4で作成したステートマシン図とおなじになることが分かるでしょう
- ランダムウォーカーの「走る」の振舞いを表すステートマシン図は、下のような図になりました



ランダムウォーカーの「走る」の振舞い
（ステートマシン図）

7-10. 実装モデルに変換する

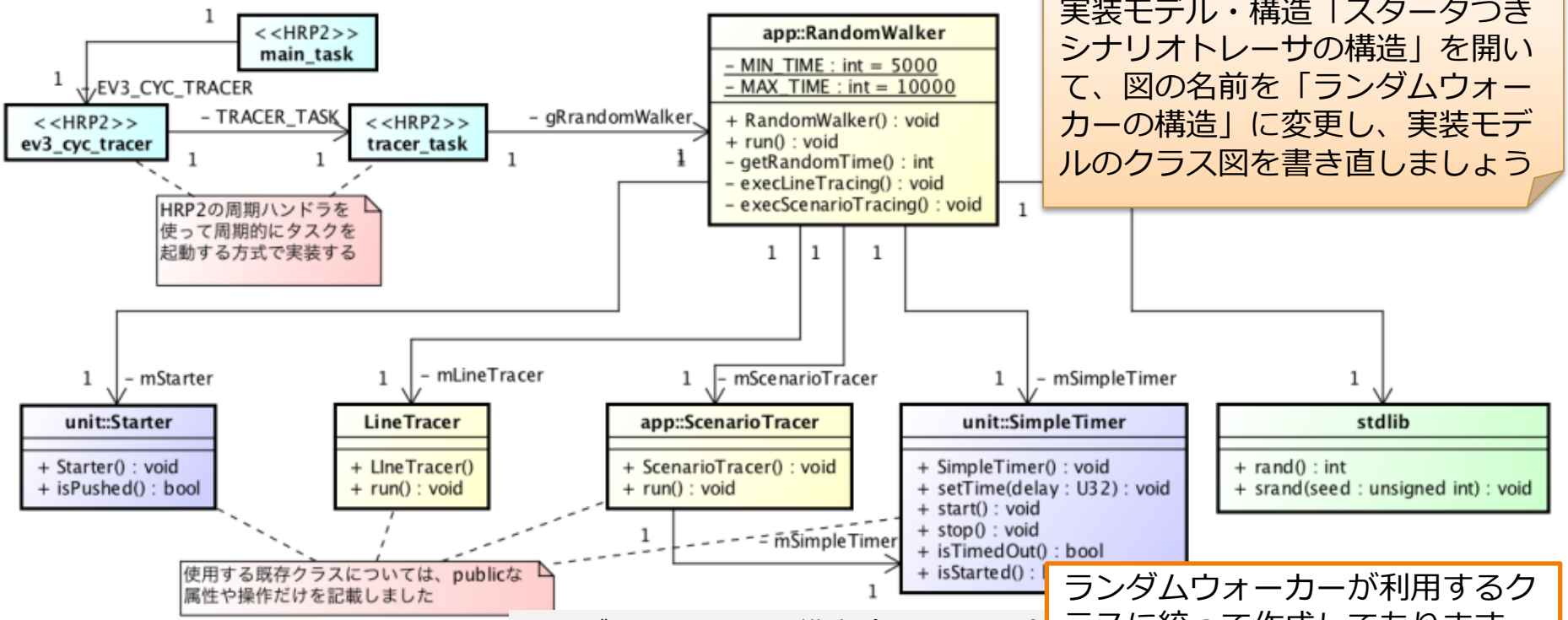
詳細設計

- 構造に関するアーキテクチャを整理します
  - app（ランダムウォーカー、ライントレーサ、シナリオトレーサなど）、unit（スタータ、倒立走行部、シンプルタイマーなど）、platform（OSやEV3RT C++など）に分け、パッケージとして扱いましょう
- 今回使うタスクやハンドラの設定を決定します
  - 今回も、アプリケーション部分は1つのタスクで構成しましょう
  - アプリケーション内には繰り返し構造を持たせず、1周期分の処理を書きます
- 提供されたライブラリのクラスやAPIに読み替えます
  - ランダムな数値を得るには、ライブラリ関数のrand関数とsrand関数（乱数ライブラリの初期化関数）を使います
  - ランダムな時間を得るために、rand関数で得られる乱数を5秒から15秒の間の任意の値に変換して使います
- 自分たちで案出したクラスをプログラミング言語で使える表現に直します
  - ランダムウォーカー：RandomWalker

7-11. 実装モデルを作成する（1）

詳細設計

- 変換の結果をまとめて実装モデルを作成します
  - 設計モデルの構造＋構造のアーキテクチャ＋実装モデルへの変換の結果を組合わせると、実装用の構造のモデルができます
- 新しく作ったクラス図は、下のような図になりました



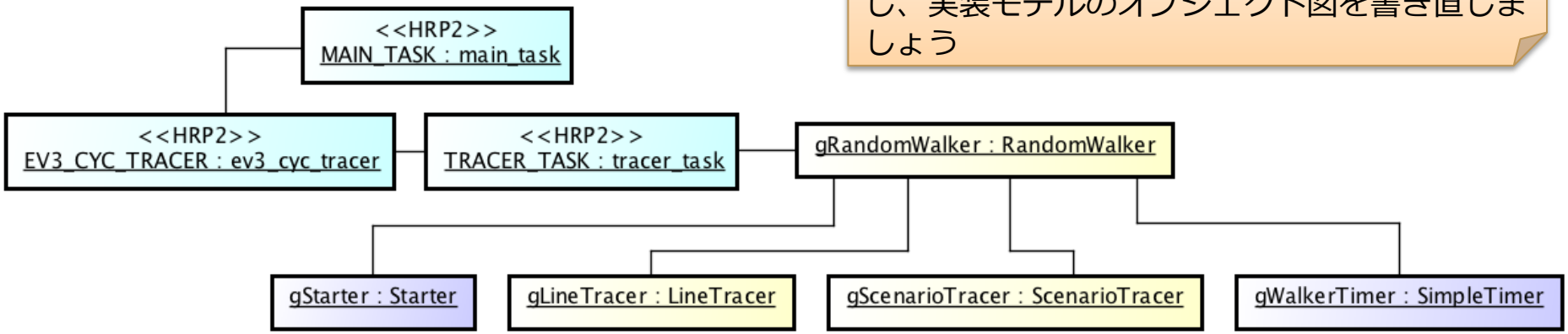
ランダムウォーカーの構造（EV3RTの場合）
（クラス図）

7-11. 実装モデルを作成する（2）

詳細設計

- 実装モデルのオブジェクト図を作成します
  - 設計モデルのオブジェクト図に登場することばを、実装モデルのクラス図で用意したことばで置き換えます
  - オブジェクト名は、実装するときにオブジェクトを作成するときに使う名前になります
- 新しく作ったオブジェクト図は、下のような図になりました

実装モデル・構造「スタータつきシナリオトレーサのオブジェクト」を開いて、図の名前を「ランダムウォーカーのオブジェクト」に変更し、実装モデルのオブジェクト図を書き直しましょう



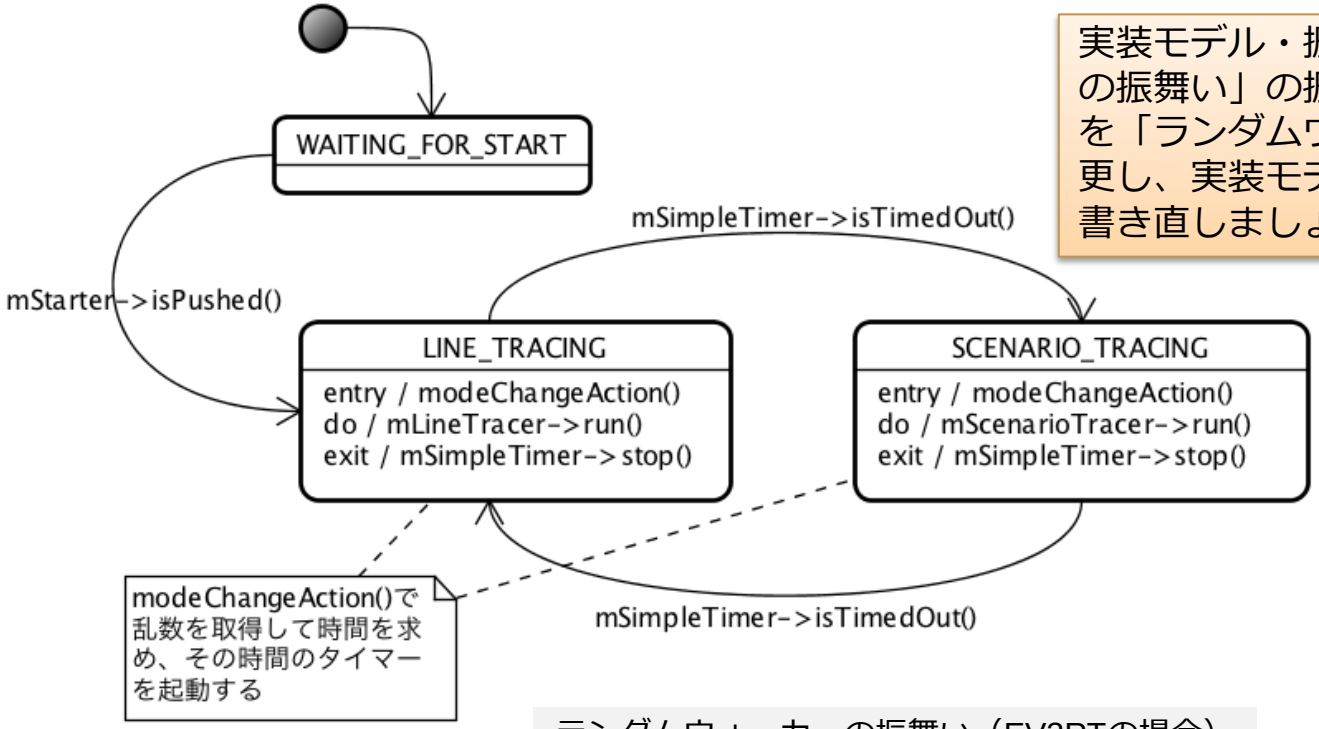
ランダムウォーカーが利用するクラスのオブジェクトに絞って作成してあります

ランダムウォーカーのオブジェクト（EV3RTの場合）
（オブジェクト図）

7-11. 実装モデルを作成する（3）

詳細設計

- 実装モデルのステートマシン図を作成します
  - 設計モデルのオブジェクト図に登場することばを、実装モデルのクラス図で用意したことばで置き換えます
  - オブジェクト名は、実装するときにオブジェクトを作成するときに使う名前になります
- 新しく作ったステートマシン図は、下のような図になりました



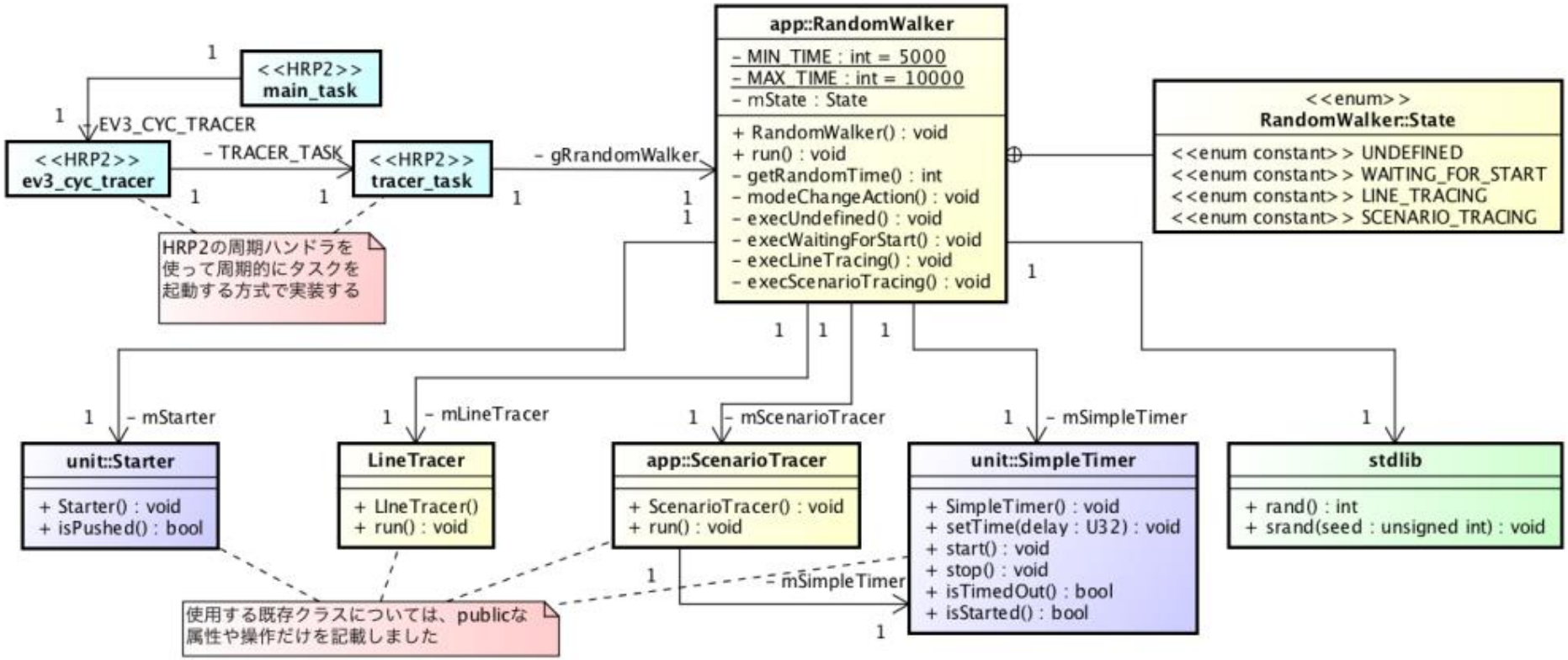
実装モデル・振舞い「シナリオトレーサの振舞い」の振舞い」を開き、図の名前を「ランダムウォーカーの振舞い」に変更し、実装モデルのステートマシン図を書き直しましょう

ランダムウォーカーの振舞い（EV3RTの場合）
（ステートマシン図）

7-11. 実装モデルを作成する（4）

詳細設計

- 状態と状態の処理を追加した後のクラス図は、次のようになりました



振舞いの処理に対応したランダムウォーカーの構造（EV3RTの場合）
（クラス図）

7-12. ソースコードを書く（1）

実装

- 構造のアーキテクチャに従ってクラスを配置するパッケージを決定します
  - RandomWalkerは「app」に配置しましょう
- ひとつのクラスを1組のソースコードに対応づけます
  - 実装モデルのクラス名をもとにヘッダファイル「クラス名.h」と実装ファイル「クラス名.cpp」を作成します
- ヘッダファイルに、属性と操作の宣言を追加します
  - 属性はメンバ変数として、操作はメソッドとして型を合わせて追加します
- 関連するクラスのヘッダファイルをヘッダファイルにインクルードします
  - 直接関連するクラスと内部で使用するライブラリのヘッダだけをインクルードします
- クラスの属性に、関連するクラスのインスタンスへの参照を追加します
- 実装ファイル（cppファイル）に、ヘッダファイルをインクルードします
  - 実装するクラス自身のヘッダファイルをインクルードします
- クラスの操作を参照して、cppファイルに対応するメソッドを追加します
- ヘッダファイルとcppファイルにコンストラクタとデストラクタを追加します

7-12. ソースコードを書く（2）

実装 EV3

複製したソースコードを開いて、変換ルールに従ってソースコードを追加、修正しましょう

- ヘッダファイルを作成しましょう
  - RandomWalkerの例

app/RandomWalker.h

```
#ifndef EV3_APP_RANDOMWALKER_H_
#define EV3_APP_RANDOMWALKER_H_

#include "Starter.h"
#include "SimpleTimer.h"
#include "LineTracer.h"
#include "ScenarioTracer.h"

class RandomWalker {
public:
  RandomWalker(
    LineTracer* lineTracer,
    ScenarioTracer* scenarioTracer,
    const Starter* starter,
    SimpleTimer* simpleTimer);
    void run();
private:
  enum State {
    UNDEFINED, WAITING_FOR_START,
    LINE_TRACING, SCENARIO_TRACING };
    static const int MIN_TIME;
    static const int MAX_TIME;

    LineTracer* mLineTracer;
    ScenarioTracer* mScenarioTracer;
    const Starter* mStarter;
    SimpleTimer* mSimpleTimer;
    State mState;

    int getRandomTime();
    void modeChangeAction();
    void execUndefined();
    void execWaitingForStart();
    void execLineTracing();
    void execScenarioTracing();
};

#endif  // EV3_APP_RANDOMWALKER_H_
```

7-12. ソースコードを書く（3）

実装 EV3

複製したソースコードを開いて、変換ルールに従ってソースコードを追加、修正しましょう

- cppファイルを作成しましょう
  - RandomWalkerの例

app/RandomWalker.cpp

```
#include <stdlib.h>
#include "Clock.h"
#include "RandomWalker.h"

const int RandomWalker::MIN_TIME = 5000;
const int RandomWalker::MAX_TIME = 15000;

RandomWalker::RandomWalker(
  LineTracer* lineTracer,
  ScenarioTracer* scenarioTracer,
  const Starter* starter,
  SimpleTimer* simpleTimer)
  : mLineTracer(lineTracer),
    mScenarioTracer(scenarioTracer),
    mStarter(starter),
    mSimpleTimer(simpleTimer),
    mState(UNDEFINED) {
  // 内蔵時計を使って乱数を初期化する
  ev3api::Clock* clock
    = new ev3api::Clock();
  srand(clock->now());
  delete clock;
}

void RandomWalker::run() {
    switch (mState) {
    case UNDEFINED:
        execUndefined();
        break;
    case WAITING_FOR_START:
        execWaitingForStart();
        break;
    case LINE_TRACING:
        execLineTracing();
        break;
    case SCENARIO_TRACING:
        execScenarioTracing();
        break;
    default:
        break;
    }
}                                    // 続く
```

7-12. ソースコードを書く（4）

実装 EV3

複製したソースコードを開いて、変換ルールに従ってソースコードを追加、修正しましょう

- cppファイルを作成しましょう
  - RandomWalkerの例

app/RandomWalker.cpp

```
// 続き
int RandomWalker::getRandomTime() {
  return MIN_TIME
    + static_cast<int>(
      static_cast<double>(rand())
      * (MAX_TIME - MIN_TIME + 1.0)
    / (1.0 + RAND_MAX));
}

void RandomWalker::modeChangeAction() {
  mSimpleTimer->setTime(getRandomTime());
  mSimpleTimer->start();
}

void RandomWalker::execUndefined() {
  mState = WAITING_FOR_START;
}

void RandomWalker::execWaitingForStart() {
  if (mStarter->isPushed()) {
    mState = LINE_TRACING;
    modeChangeAction();
  }
}

void RandomWalker::execLineTracing() {
  mLineTracer->run();
  if (mSimpleTimer->isTimedOut()) {
    mSimpleTimer->stop();
    mState = SCENARIO_TRACING;
    modeChangeAction();
  }
}

void RandomWalker::execScenarioTracing() {
  mScenarioTracer->run();
  if (mSimpleTimer->isTimedOut()) {
    mSimpleTimer->stop();
    mState = LINE_TRACING;
    modeChangeAction();
  }
}
```

状態ごとのメソッドをステートマシン図に合わせて編集しましょう

7-13. ビルドし走行体を動かす（１）

EV3

テスト

- 演習1、2のソースコードと今回新たに作成したソースコードをいっしょにビルドします
  - 次のページを参照して、コピーした etrobo_tr_ex2用の Makefile の次の箇所を書き換えます
    - ターゲット実行形式ファイル名を、etrobo_tr_ex3 に変更します
    - Cソースファイルに、新たに追加したクラスの実装ファイル名を追加し、不要なファイル名を外します
  - ビルド手順については、4章を参考にしてください
    - 「app」がビルドできればOKです
- 完成したプログラムを走行体にアップロードして実行します
  - アップロード手順については、4章を参考にしてください
  - アップロードできないとき
    - 本体に挿入したSDカードの空きが不足しているかもしれません
      - USBで転送する場所は、本体に挿入したSDカードなのを思い出しましょう
    - これまでの演習で使ったファイルを削除してからアップロードしてみましょう

7-13. ビルドし走行体を動かす（2）

- Makefile.incを修正します
- ビルド手順については、4章を参考にしてください
  - 「app」がビルドできればOKです
  - もし、演習2で、使わなくなったライントレーサ等のファイルを削除していた場合は、etrobo_tr_ex1からヘッダファイルとcppファイルをコピーしてきましょう

LineMonitor.o、
LineTracer.o、
RandomWalker.o
を追加します

テスト

Makefile.inc

```
（途中省略）
APPL_CXXOBJS += ¥
        BalancingWalker.o ¥
        BalancerCpp.o ¥
        Starter.o ¥
        SimpleTimer.o
        ScenarioTracer.o ¥
        Scenario.o ¥
        Scene.cpp ¥
        LineMonitor.o ¥
        LineTracer.o ¥
        RandomWalker.o ¥



（以下省略）
```

理解度チェック（7）

- これまでの部品を使って新たな機能を実現する

1. ランダムウォーカーを開発するため、演習1、演習2の開発との差分を調査しました。どのような差分があったでしょうか？
   a. 機能的側面
   b. 構造的側面
   c. 振舞い的側面
2. 差分を埋めるために、新規開発するもの、改造するもの、再利用するものなどが発見されました。どのようなものがあったでしょうか？
   a. 新しく開発したもの
   b. 改造したもの
   c. 再利用したもの

理解度チェック（7）

3. 演習3のランダムウォーカーには、処理を繰り返せない障害があります
   a. 処理が続けられない原因はなんでしょうか
   b. どうすればライントレーサとシナリオトレーサの切り替えが繰り返せるようになるでしょうか
   c. そのためには、どの工程のどのモデルを見直せばよいでしょうか

4. ある人が次のような改善策を考えていますので、設計の見直しに協力してください
   a. シナリオトレーサから抜けたら、ラインを探索する
   b. ラインを見つけたらライントレースを試みる
      - そこでうまくライントレースしなかった時は諦めることにする
   c. 一定時間ラインを探しても見つからなかったら、停止する