- 4章までに学んだ開発方法を使って、ロボットに別の機能を追加してみましょう
- 部品とモデルを使ったソフトウェア開発を実際に体験してみましょう

5. 演習１）システムに新しい機能を追加する

5-1. 演習課題と進め方

- 例題1で開発したロボットに別の機能を追加します
  - ユースケース「経路に沿って走行する」にスタート機能を加えます
    - 電源ONで起動しただけでは走行を開始しません
    - ユーザがタッチセンサを押した時点で、走行を開始します
    - いったん走行を開始したら、タッチセンサが押されても無視します
- 例題1で紹介した「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

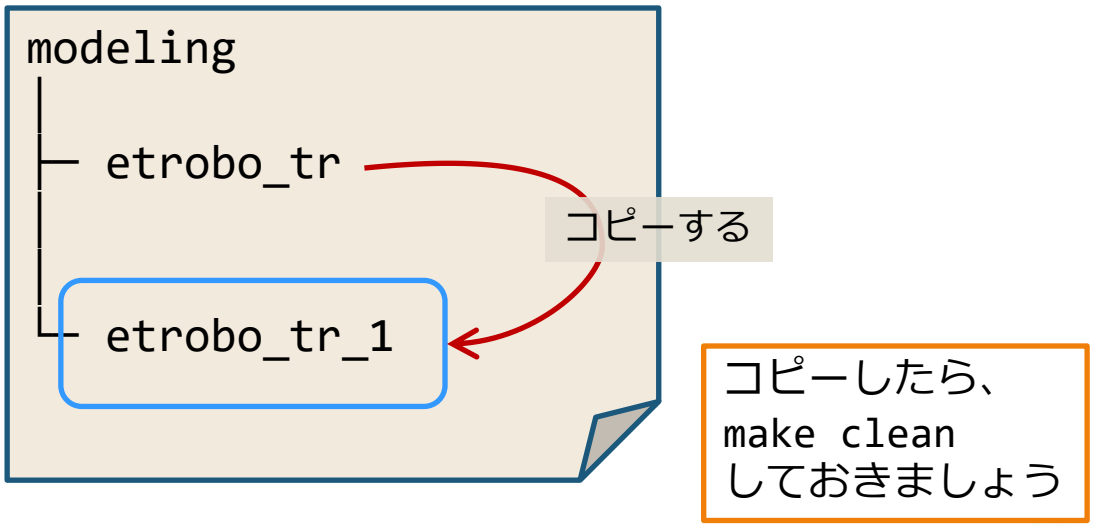
5-2. 演習の準備（１）

- 5章から７章で使う配布ファイル
  - ファイル名：chap-05-07.zip
  - 5章から７章のテキストが含まれています
  - コードやモデルの解答例は含まれていません
- 圧縮ファイルchap-05-07.zipを展開します
  - 展開したディレクトリにテキストが含まれていることを確認しましょう

```
chap-05-07
└── pdf                          テキストのディレクトリ
    ├── chap-05.pdf                5章のテキスト
    ├── chap-06.pdf                6章のテキスト
    └── chap-07.pdf                7章のテキスト
```

5-2. 演習の準備（2）

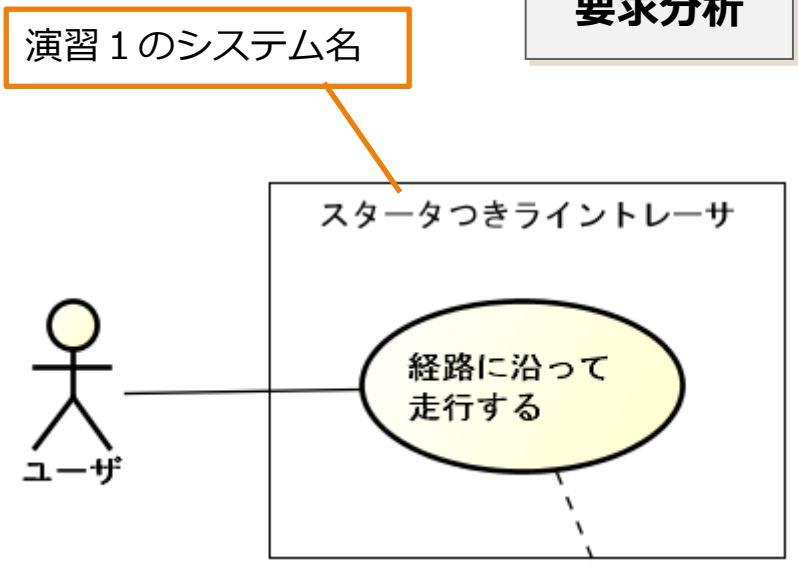
- モデル図の準備
  - 例題1で使ったモデル図のファイル「etrobo_tr.asta」をコピーして、「etrobo_tr_1.asta」を作ります
- ソースコードの準備
  - 「modeling」ディレクトリ中の「etrobo_tr」ディレクトリを複製します
  - 複製したディレクトリの名前を「etrobo_tr_1」に変えます
  - make cleanしておきましょう



5-3. システムが提供する機能を決める（１）

- 何が変わるか考えましょう
- ユーザに提供している機能は何だったでしょうか
  - その機能自体が変わっていますか
  - それとも機能を実現する方法が変わっていますか
- スタート機能の追加はどこに記述したらよいでしょうか
  - 新しいユースケースを追加しますか
  - 既存のユースケースを修正しますか

要求分析



スタータつきライントレーサの機能
（ユースケース図）

要求モデル・機能「2輪倒立振子ロボットの機能」を開いて、図の名前を「スタータつきライントレーサの機能」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）

5-3. システムが提供する機能を決める（２）

- ユーザとユーザに提供する機能は変わりません
  - ユースケース図はシステムの名称を「スタータつきライントレーサ」に変更しましょう
- ユーザに機能を提供する手順や動作は修正が必要です
  - ユースケースの記述を追加された処理に応じて修正しましょう

要求分析

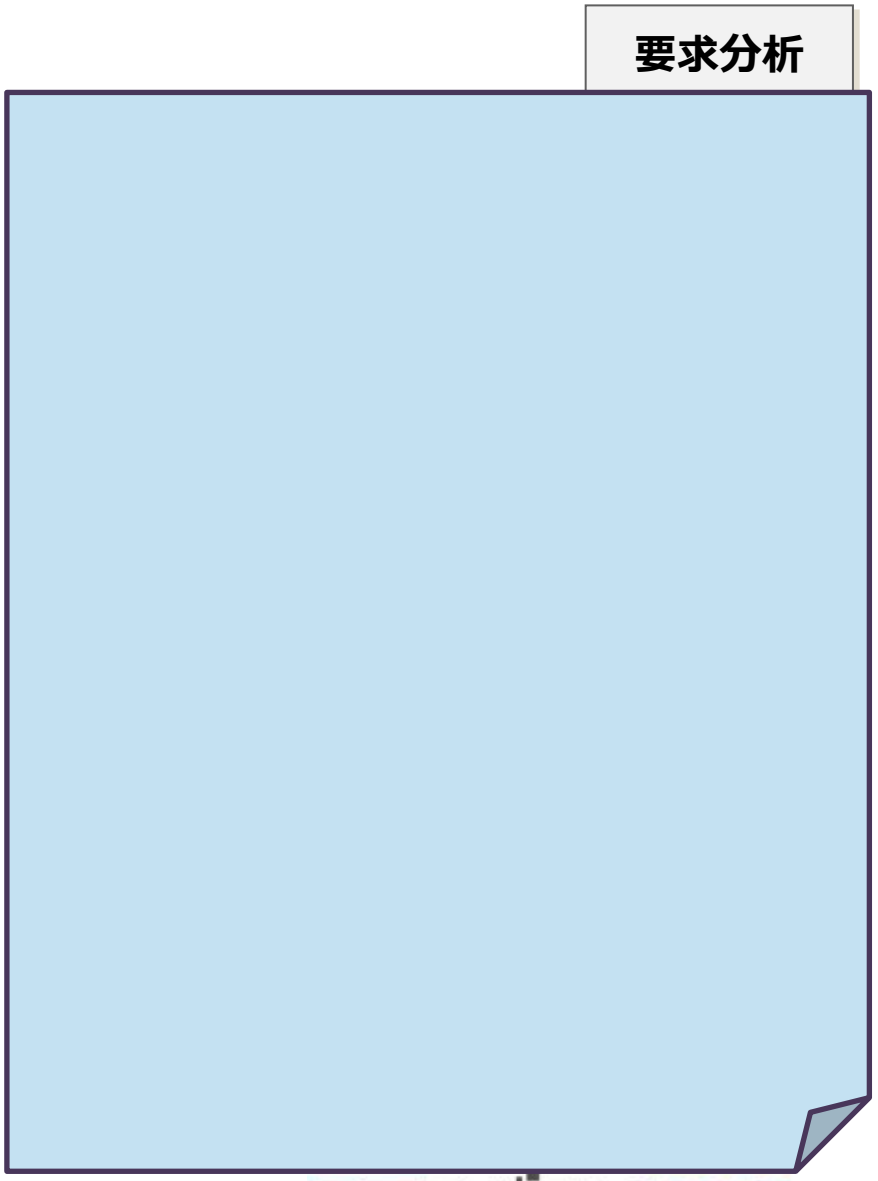
「経路に沿って走行する」(変更前)の基本ユースケース

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する
3. ロボットは、経路上にいるか外れているかを調べ、走行する向きを計算する
4. ロボットは決定した向きに従って倒立走行する
5. 3~4を繰り返すことで経路に沿って走行する

要求モデル・機能「2輪倒立振子ロボットの機能」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

5-3. システムが提供する機能を決める（３）

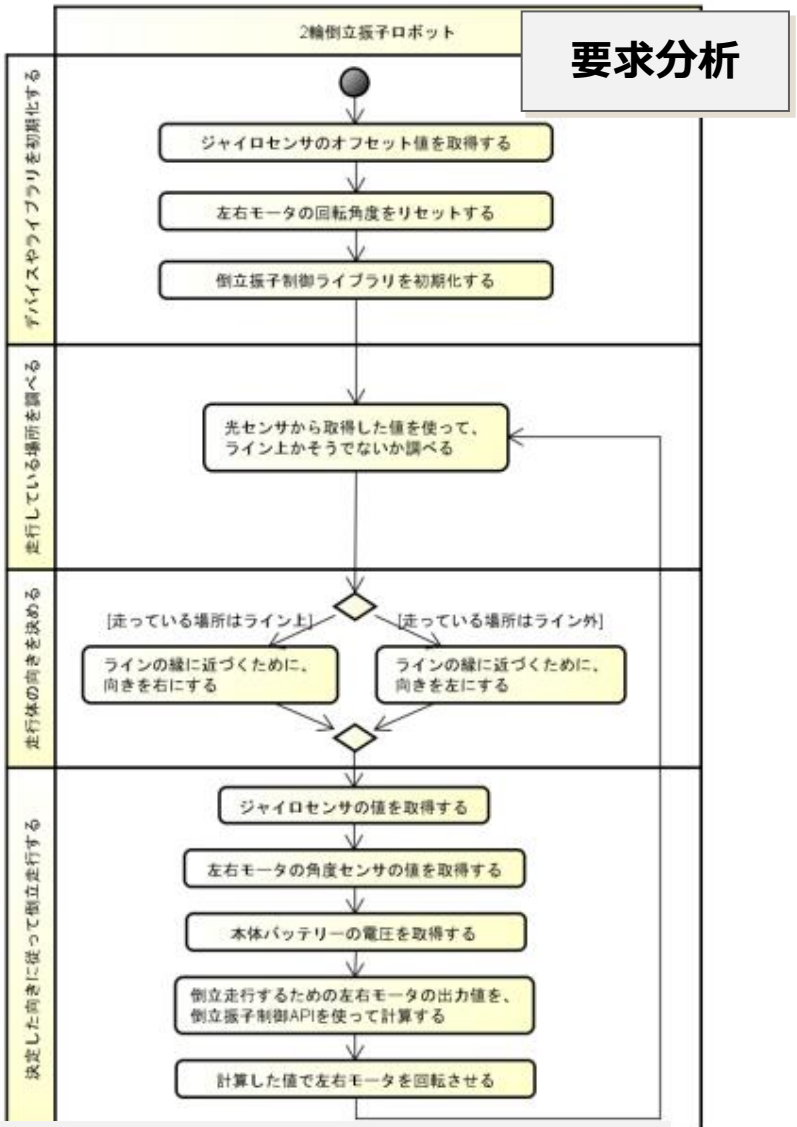
- 修正したユースケース記述は、右のようになりました
  - 赤字のところが追加したところです



要求モデル・機能「2輪倒立振子ロボットの機能」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

5-4.　機能の実現方法を考える（１）

- 処理の流れには修正が必要です
  - ユースケース記述が変わったのに合わせて修正しましょう
  - 「スタート機能」に関わる処理が追加されています
- アクティビティ図を使って処理の流れを検討します
  - 例題1のアクティビティ図を修正して、「スタート機能」を実現するための制御フローを書き加えてみましょう

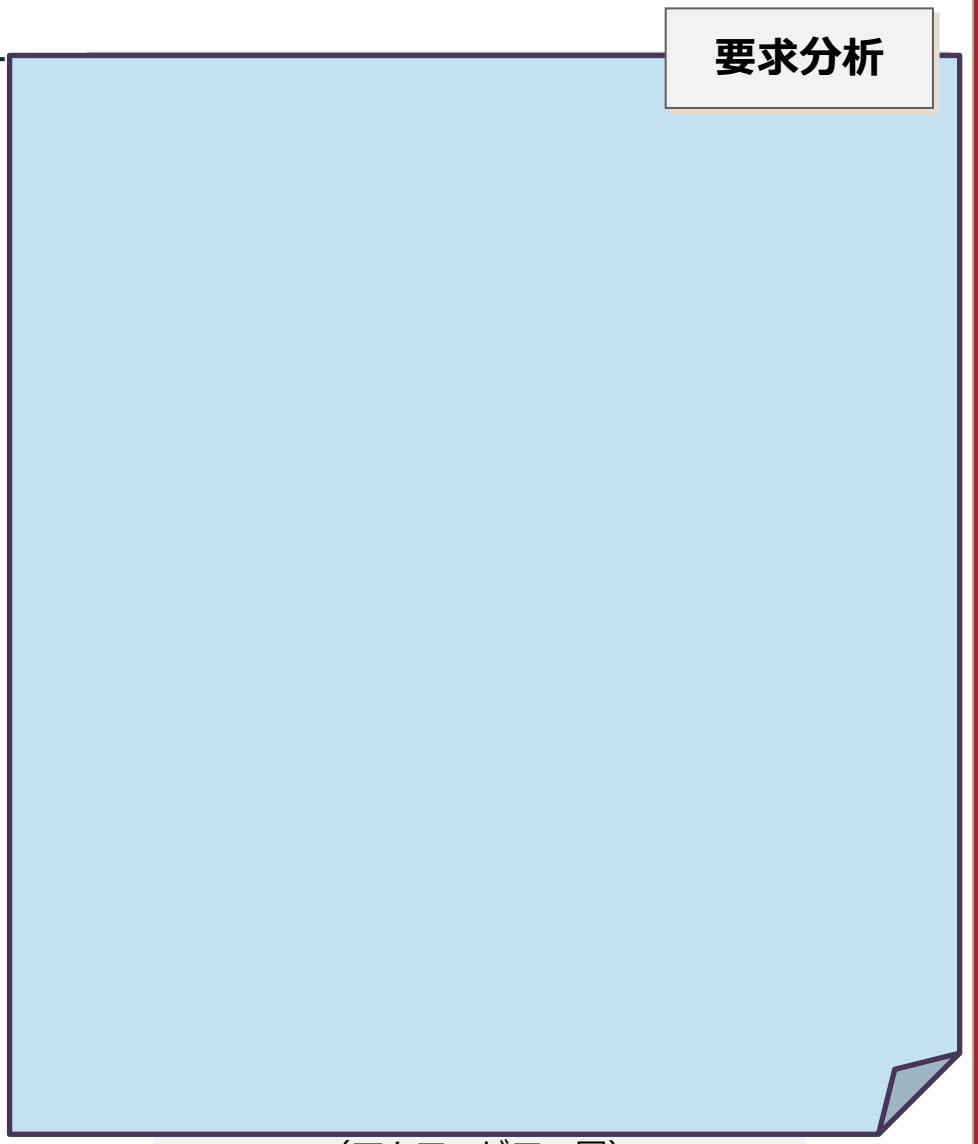


例題1の「経路に沿って走行する」の振舞い（アクティビティ図）

5-4.　機能の実現方法を考える（2）

- パーティションを追加します
  - 「ユーザーの指示を待つ」というパーティションを追加しましょう
- 処理の流れを追加します
  - 追加したパーティションに、スタートボタンが押されるまで待つような制御フローを記述します
- 修正したアクティビティ図は、右のようになりました
  - 水色で囲んだ部分が追加したところです

要求モデル・振舞い「経路に沿って走行する」を開いて、アクティビティ図を書き直しましょう



（アクティビティ図）

5-5. 機能実現に必要な部品を見つける（１）

基本設計

- 部品の候補を考えます
  - アクティビティ図に追加した制御フローに着目し、「働き」や「情報」を探します
- 例題1で紹介した表を参考に、新しい表を作ってみましょう

「働き」や「情報」	部品の候補
経路に沿って走行する	2輪倒立振子ロボット
デバイスやライブラリを初期化する	2輪倒立振子ロボット
走行している場所（ライン上かどうか）を調べる	2輪倒立振子ロボット
光センサの値を取得する	光センサ（EV3ではカラーセンサを使う）
走行体の向きを決める	2輪倒立振子ロボット
ジャイロセンサのオフセット値を取得する	ジャイロセンサ
ジャイロセンサの値を取得する	
本体バッテリーの電圧を取得する	本体（インテリジェントブロック）のバッテリー
左右モータの角度センサの値を取得する	左モータ、右モータ （モータを回転させるとロボットは走行する）
左右モータを回転角度をリセットする	
左右モータを回転させる	
倒立振子ライブラリを初期化する	倒立振子制御ライブラリ
倒立走行のための左右モータの出力値を計算する	
決定した向きに従って倒立走行する	2輪倒立振子ロボット

例題1のライントレーサの働きや部品の候補

5-5. 機能実現に必要な部品を見つける（２）

基本設計

- 部品の候補を考えて表を作ってみましょう

追加する機能に必要な働きや情報をリストアップ

- 新しく作った表は、下のような表になりました

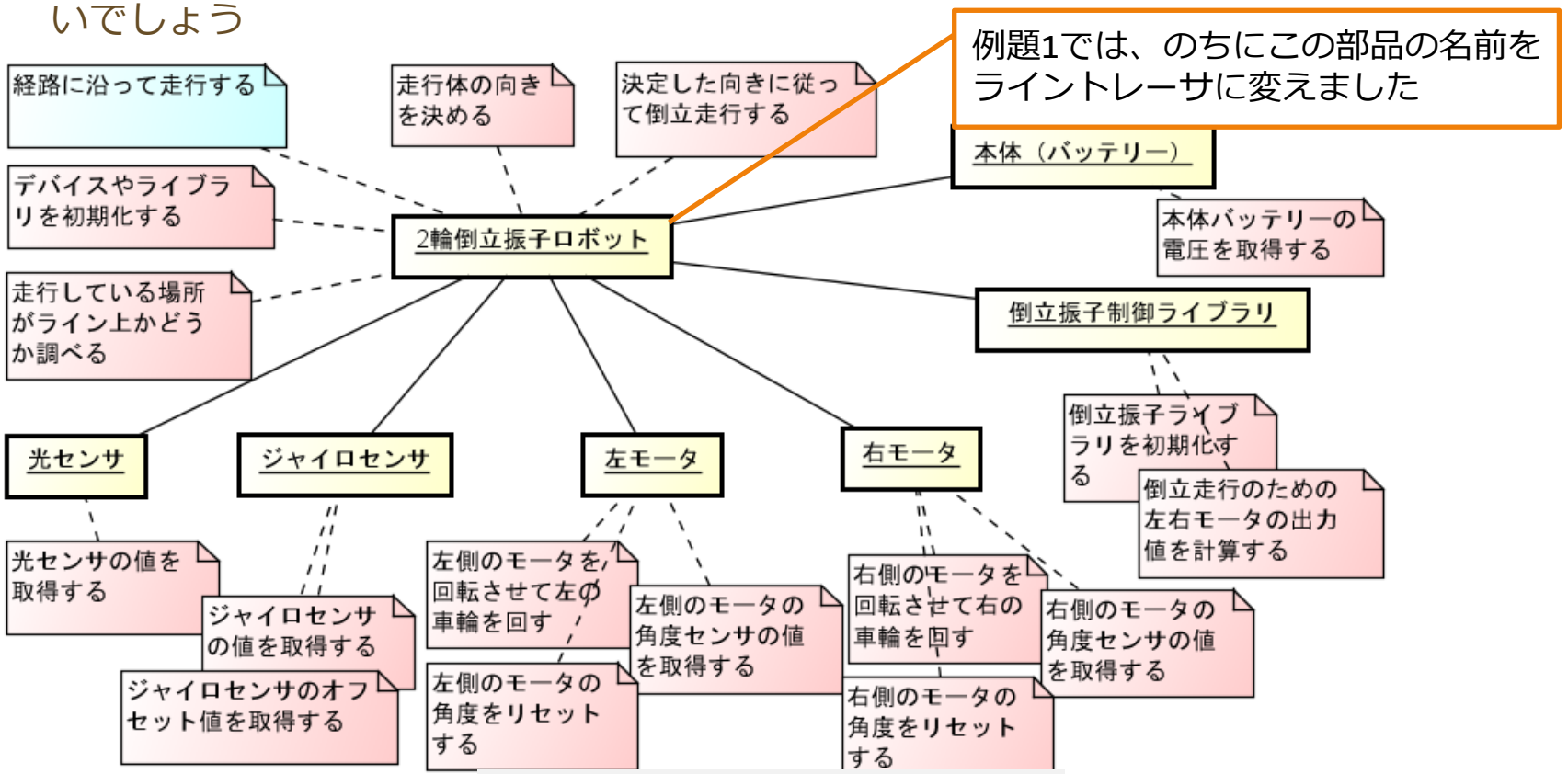
「働き」や「情報」	部品の候補

働きや情報から部品の候補を抽出

5-6. 新しい部品を定義する（1）

基本設計

- スタータつきライントレーサを構成する要素を考えましょう
- 例題1で紹介したオブジェクト図を参考に、新しい図を書いてみましょう
  - スタート機能が検討したいので、再利用するライントレーサの詳細は隠してもかまわないでしょう



例題1のライントレーサの部品のつながり
（オブジェクト図）

5-6. 新しい部品を定義する（2）

基本設計

- 例題1のライントレーサの詳細を隠します
  - ライントレーサとタスクだけ残して他は削除しましょう
    - 残しておいたほうが分かりやすい人は、残しておいてもかまいません
- 「スタート機能」を実現するために必要な部品を書きます
  - 5.5で抽出した部品候補から抽出した部品
- 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

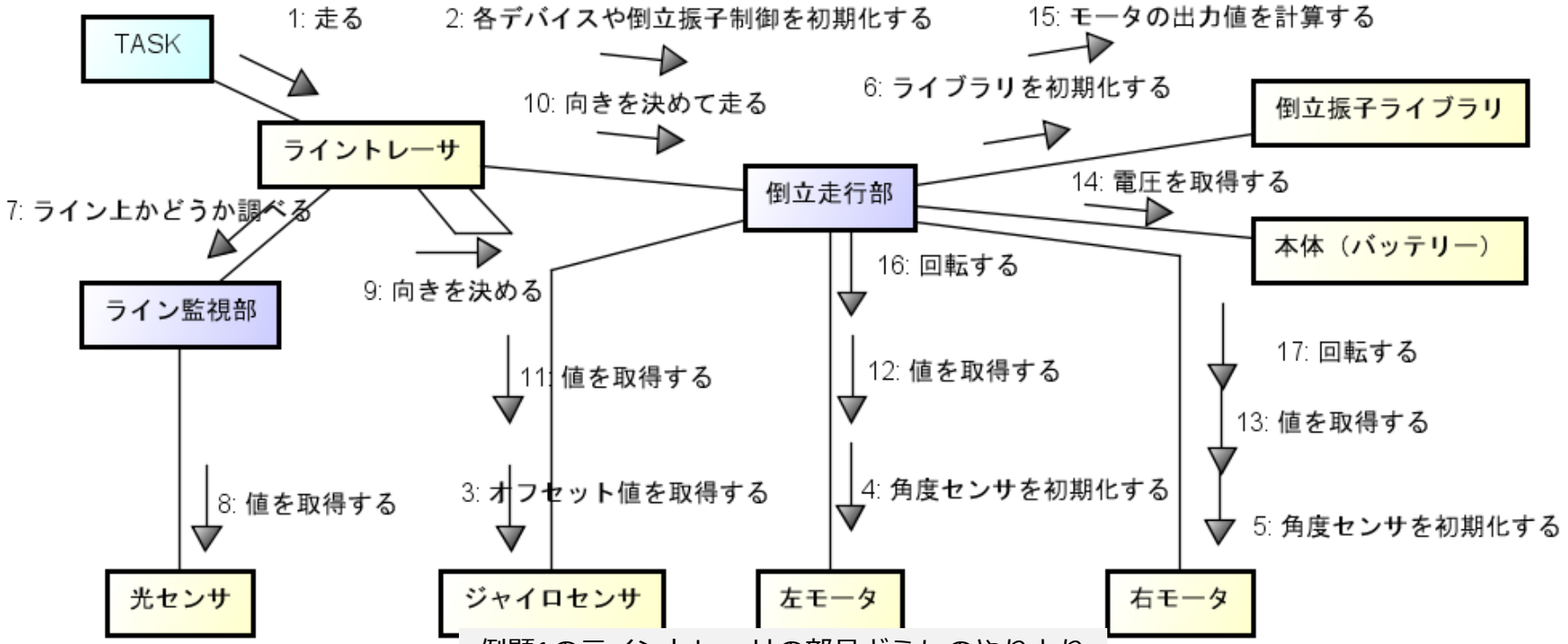
設計モデル・構造「ロボットの部品の候補のつながり」を開いて、図の名前を「スタータつきライントレーサの部品の候補のつながり」に変更し、オブジェクト図を書き直しましょう

例題１のオブジェクト図に加える要素の構造

5-7. 部品による機能実現を確認する（１）

基本設計

- 「スタート機能」が実現できるか確認しましょう
- 例題1で紹介したコミュニケーション図を参考に、新しい図を書いてみましょう
  - スタート機能が検討したいので、再利用するライントレーサの内部のやりとりは隠してもかまわないでしょう



例題1のライントレーサの部品どうしのやりとり（コミュニケーション図）

5-7. 部品による機能実現を確認する（2）

基本設計

- 例題1のライントレーサの詳細を隠します
  - ライントレーサとタスクだけ残して他は削除しましょう
    - 残しておいたほうが分かりやすい人は残しておいても構いません
- 「スタート機能」を実現するのに必要なやりとりを書きます
  - 5.5で抽出した部品候補から抽出した部品を追加する
  - 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「経路に沿って走行するときの部品間のやりとり」を開いて、図の名前を「スタータつきライントレーサ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

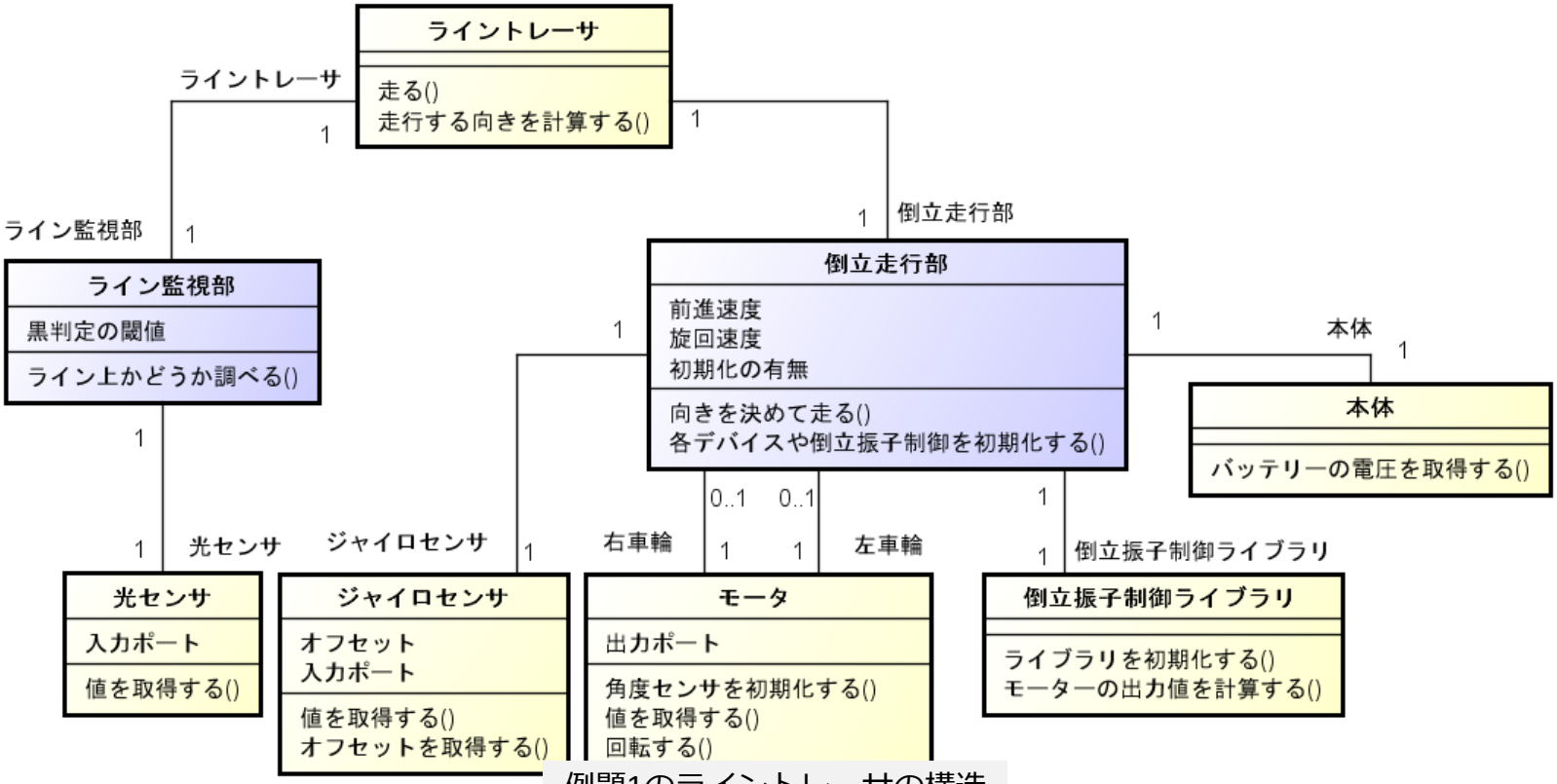
例題１のコミュニケーション図
に加える要素の振舞い

（コミュニケーション図）

5-8. システムの構造を決定する（1）

基本設計

- スタータつきライントレーサの構造を決めましょう
- クラス図を使って構造の図を書いてみましょう
  - 例題1のクラス図に、これまでに作成したオブジェクト図とコミュニケーション図を参考にして、新しい図を書いてみましょう



例題1のライントレーサの構造
（クラス図）

5-8. システムの構造を決定する（2）

基本設計

- 例題1のライントレーサの詳細を隠します
  - ライントレーサだけ残して他は削除しましょう
    - 残しておいたほうが分かりやすい場合は消さなくてもよいです
- 「スタート機能」を実現するのに必要なクラスを書きます
  - 5.5で抽出した部品候補から抽出した部品を追加する
  - コミュニケーション図で使ったやりとりを操作に追加しましょう
- 新しく作ったクラス図は、下のような図になりました

例題１のクラス図に加える構造

設計モデル・構造「ラインントレーサの構造」を開いて、図の名前を「スタータつきラインントレーサの部品どうしのやりとり」に変更し、クラス図を書き直しましょう

（クラス図）

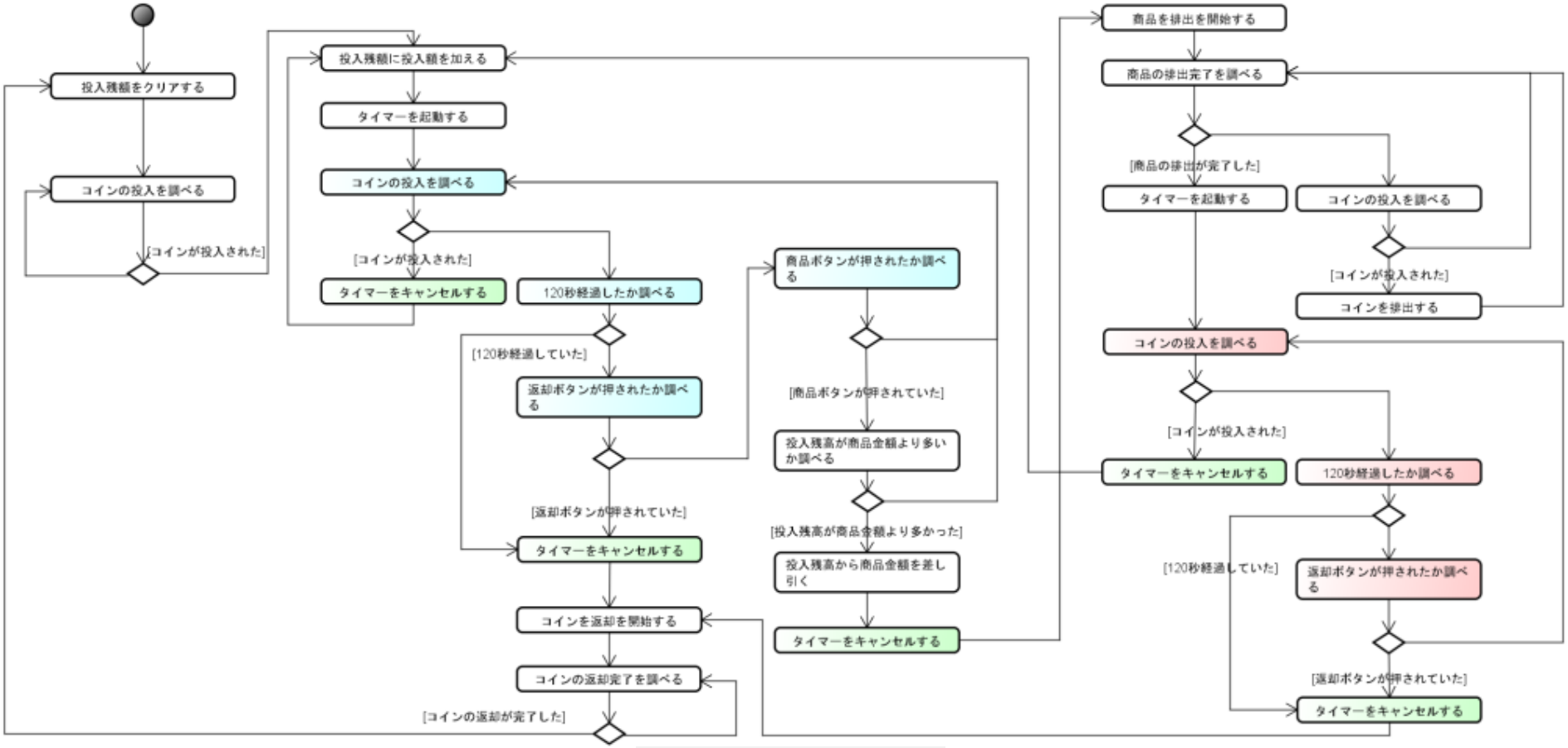
5-9. 振舞いの作り方を検討しよう（1）

- 次のような自動販売機AXの振舞いを考えてみましょう
  - コインが投入されると投入残額に加算されます
  - 投入残額が商品購入額以上なら、商品ボタンを押すと、投入残額から商品金額分が差し引かれ、商品が排出されます
    - 残額が不足なら、商品ボタンを押しても何も起きません
  - 商品を排出している間は、コインを投入しても排出して戻されます
  - 商品排出後にコインが投入されれば、引き続き購入できます
  - コインの投入中に返却ボタンを押すとコインが返却されます
  - 商品排出後に返却ボタンを押すとコインが返却されます
  - コインの投入中に何も操作しないで120秒経過するとコインが返却されます
  - 商品排出後に何も操作しないで120秒経過するとコインが返却されます
  - コインの投入はハードでチェックし、投入されたことと投入額が通知されます
  - コインの返却はハードが実行し、返却作業が完了すると通知されます
  - 商品の排出はハードが担当し、排出が完了すると通知されます

- この自動販売機のアクティビティ図、すぐに書けそうですか？

5-9. 振舞いの作り方を検討しよう（2）

- 自動販売機AXの振舞いをアクティビティ図で描いてみました
  - 見通しはいいでしょうか？思ったより複雑で面倒ではないでしょうか？



自動販売機AXの振舞い
（アクティビティ図）

5-9. 振舞いの作り方を検討しよう（3）

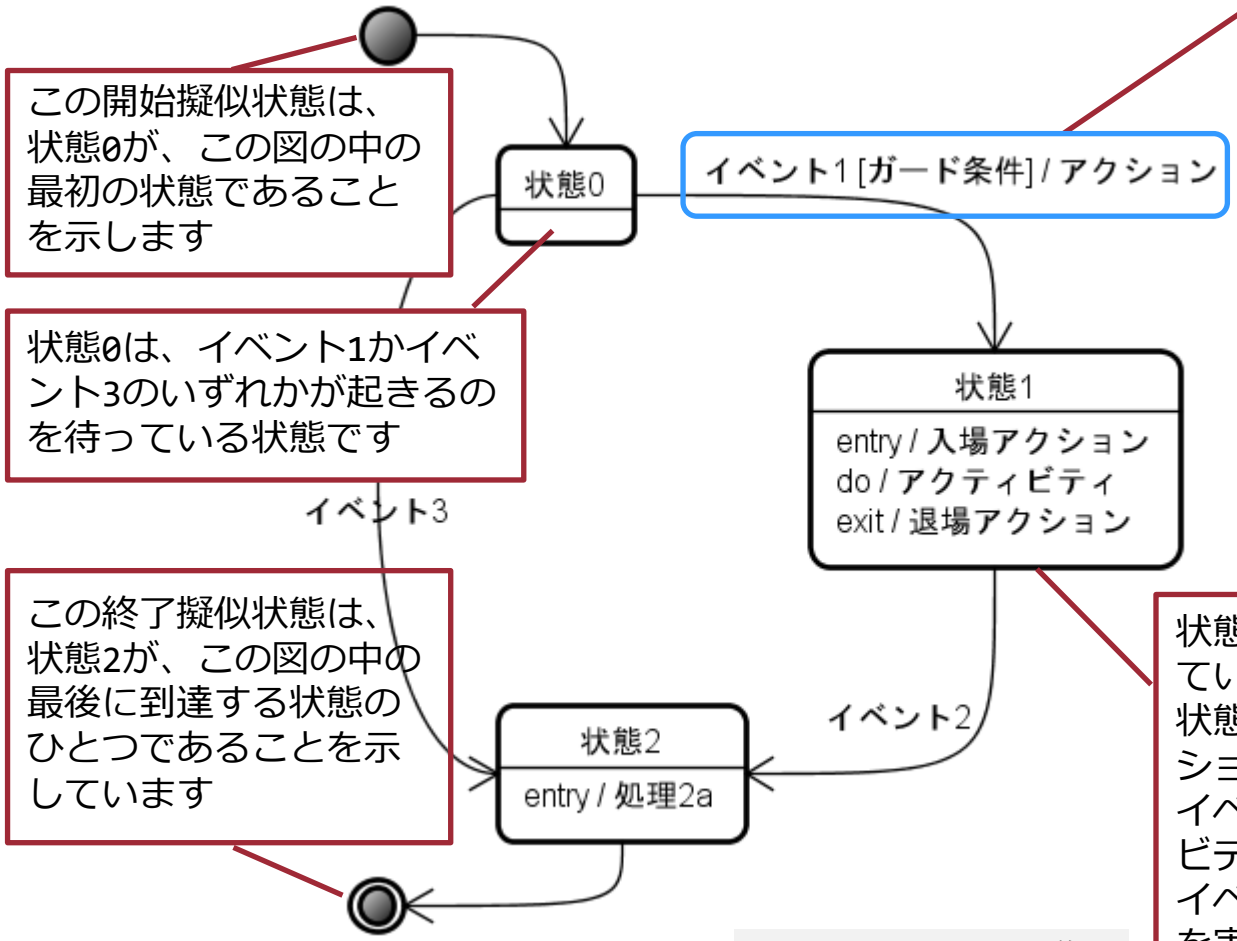
- アクティビティ図やシーケンス図を使うとよいところ
  - 処理の順序を把握しやすい
  - 複数のオブジェクトの間のやりとりが把握しやすい
- アクティビティ図やシーケンス図を使うと面倒な場合
  - 同時にいくつかのできごとを待つような処理
    - コインが投入されるか、返却ボタンが押されるか、商品ボタンが押されるか、120秒経過したか
  - 同じことが起きた時に、状況によって異なる処理をする
    - 商品排出中は、コインが投入されると受け付けずに排出する
  - ある時点までにどんなことが起きていたのかに依存するような処理
    - 覚えさせておくためのフラグや変数が必要になるが、図の記法のサポート外の情報になる
  - 時間の経過を待ちながら、他の処理をしたいとき
    - 経過を待つには、待ち始める時と、待つのを止める時の処理を忘れないようにする必要がある
- どのような図があるとよさそうでしょうか
  - 複数のできごとを待っていることを楽に表現できる
  - 同じことが起きても処理が異なる場面を違う状況として区別できる
  - できごとやできごとが起きたことを調べるのと、その時の処理を別のものとして扱える

5-10. ステートマシン図を使ってみる（１）

- ステートマシン図
  - 待っているできごとや、できごとが起きたときの動作が分かる図です
- 次のような要素を使って振舞いを表します
  - イベント
    - 起きるのを待っているできごと
  - 状態
    - 待っているできごとが異なる状況を分けたもの
  - 状態遷移
    - できごとが起こることで異なる状況（状態）に移ること
  - ガード条件
    - イベントが発生した後に評価し、真のときにだけ遷移が許される条件
  - アクション
    - 遷移と関連づけられ、遷移が起きるときに同時に実行される動作
  - アクティビティ
    - 状態と関連づけられ、状態に入ったときに開始される動作

5-10. ステートマシン図を使ってみる（2）

- ステートマシン図の構成要素と振舞いの説明



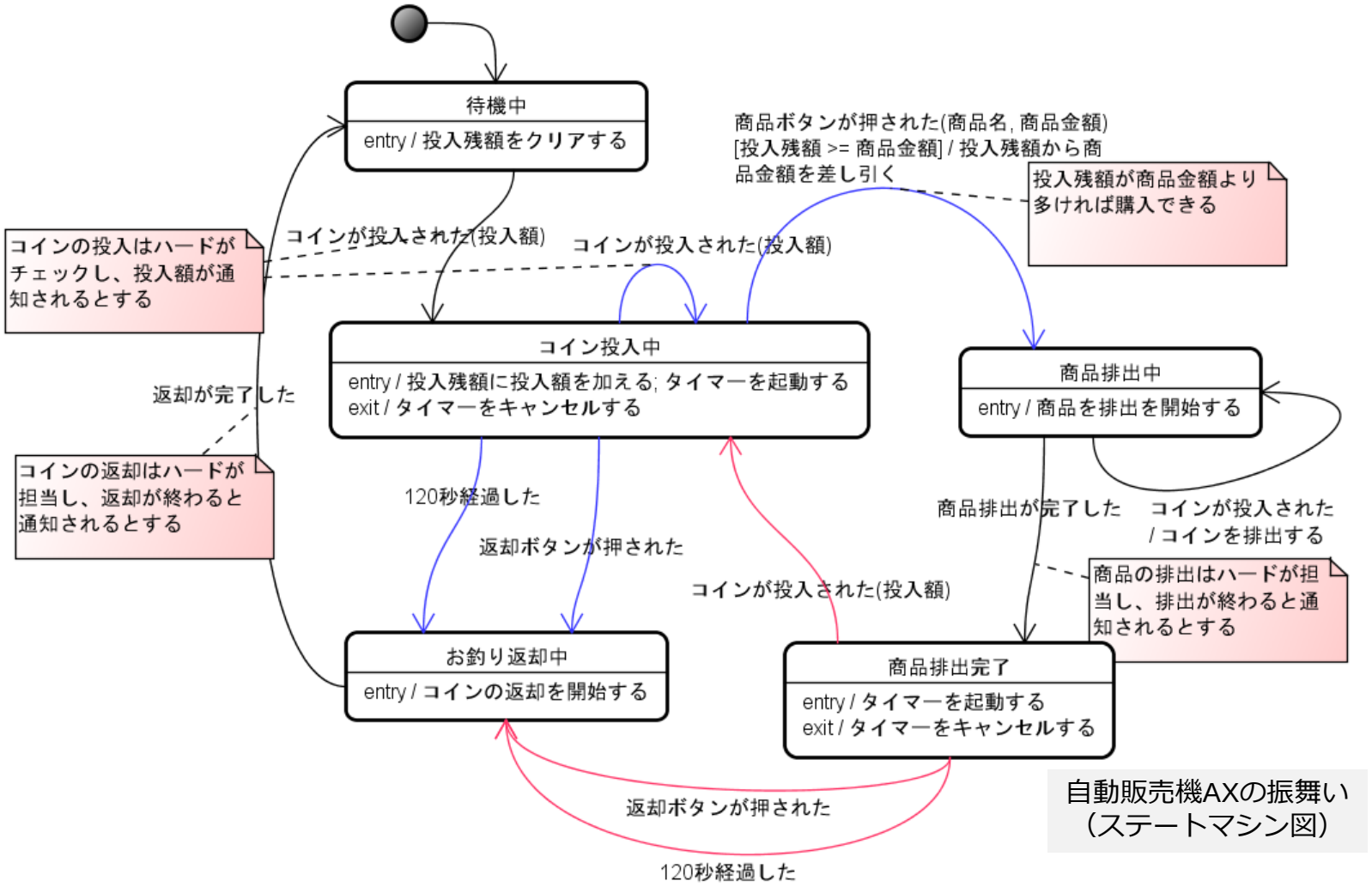
ステートマシン図の説明

5-10. ステートマシン図を使ってみる（3）

- イベントに使えるもの
  - 自分のクラスの処理から調べられるもの
  - 「関連している」他のクラスから調べられるもの
- アクションに書けること
  - 自分のクラスの操作
  - 「関連している」他のクラスの操作
  - 見当たらなければ、自分のクラスに操作を追加して、それを使います
  - 単純に操作を呼び出すだけでは済まない処理は、操作に追加してから追加します
    - その処理が複数の操作の呼び出しや条件判断が必要なら、処理の流れをアクティビティ図などで書いておくとよいでしょう
    - 処理を繰り返す必要がある場合には、その繰り返しの間に待つできごとがないことを確認し、あるなら、状態を分けることを先に検討しましょう
- 処理が同じだけでは状態が同じではないことに気をつけます
  - 待っているできごとが同じでアクションが同じであれば、同じ状態の可能性があります
  - 待っているできごとが違うときは、同じアクションであっても、別の状態の可能性が高いです
    - アクションを共有したいなら、クラスに操作として追加して、それぞれの状態で使います

5-10. ステートマシン図を使ってみる（4）

- 自動販売機AXの振舞いをステートマシン図で描いてみました
  - 待っているできごと、できごとが起きた時の処理の見通しはよくなったでしょうか？



自動販売機AXの振舞い
（ステートマシン図）

5-10. ステートマシン図を使ってみる（５）

- 振舞いを表すモデルには、違いがあることがわかりました
  - アクティビティ図、シーケンス図
    - 処理の順序、オブジェクト間のやりとりを表すのに向いている
  - ステートマシン図
    - あるオブジェクトが、どのようなできごとを待っていて、できごとが起きるとどのように振る舞うのかを表すのに向いている

- 例題ではシーケンス図を活用しました
  - シーケンス図で振舞いモデルを作成し、シーケンス図からソースコードに変換する演習をやりました
- 演習１ではステートマシン図の使い方に慣れましょう
  - あまり複雑な状況になると演習が大変なので、演習1を簡単な題材として利用します
  - ステートマシン図で振舞いのモデルを作ります
  - ステートマシン図からソースコードに変換する方法についても演習しましょう

5-11. システムの振舞いを確認する（１）

- 状態を持つクラス の振舞いを担当する
  「走る」メソッドの動作を書いてみましょう

基本設計

- ステートマシン図を使って書いてみましょう
- 5-5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

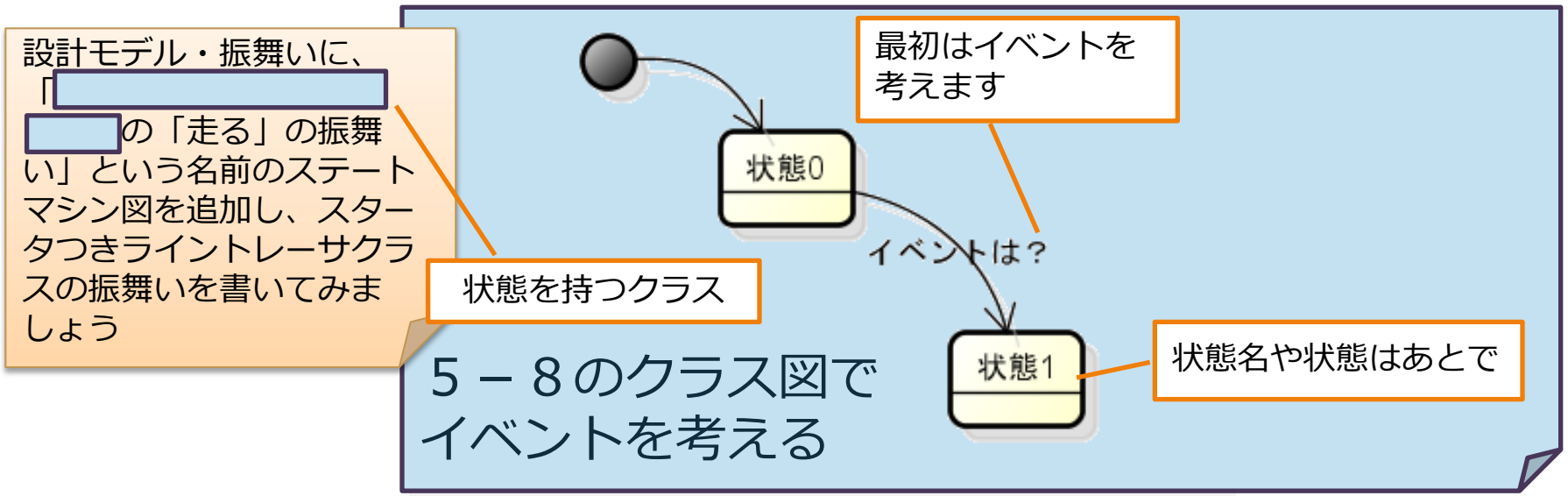
５－８のクラス図で
状態の候補を見つける

（クラス図とステートマシン図）

5-11. システムの振舞いを確認する（２）

基本設計

- ステートマシン図を追加します
  - 開始擬似状態と最初の状態を追加して、状態遷移でつなぎます
  - ２つ目の状態を追加します
- 状態遷移とイベントを書きます
  - ライントレーサは、最初の「待っているできごと」が何かを考えます
  - 最初の状態と２つ目の状態の間に状態遷移を追加します
  - 最初の待っているできごとをこの状態遷移のイベントとします
    - イベントにするときは、起きてほしいできごとが「起きた」と書いておくとよいでしょう



（ステートマシン図）

5-11. システムの振舞いを確認する（３）

基本設計

- アクションを書きます
  - それまでにやっておくことを最初の状態のアクションに書きます
    - 操作の呼び出しを「～をする」と書きます（なければアクションはなしです）
- 状態の名前を決めます
  - 起きるのをまっているできごとからつけるなら「～待ち」とつけます
  - できごとが起きるまでの振舞いから付けるなら「～中」とつけます
- 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

（ステートマシン図）

5-12. 実装モデルに変換する

EV3

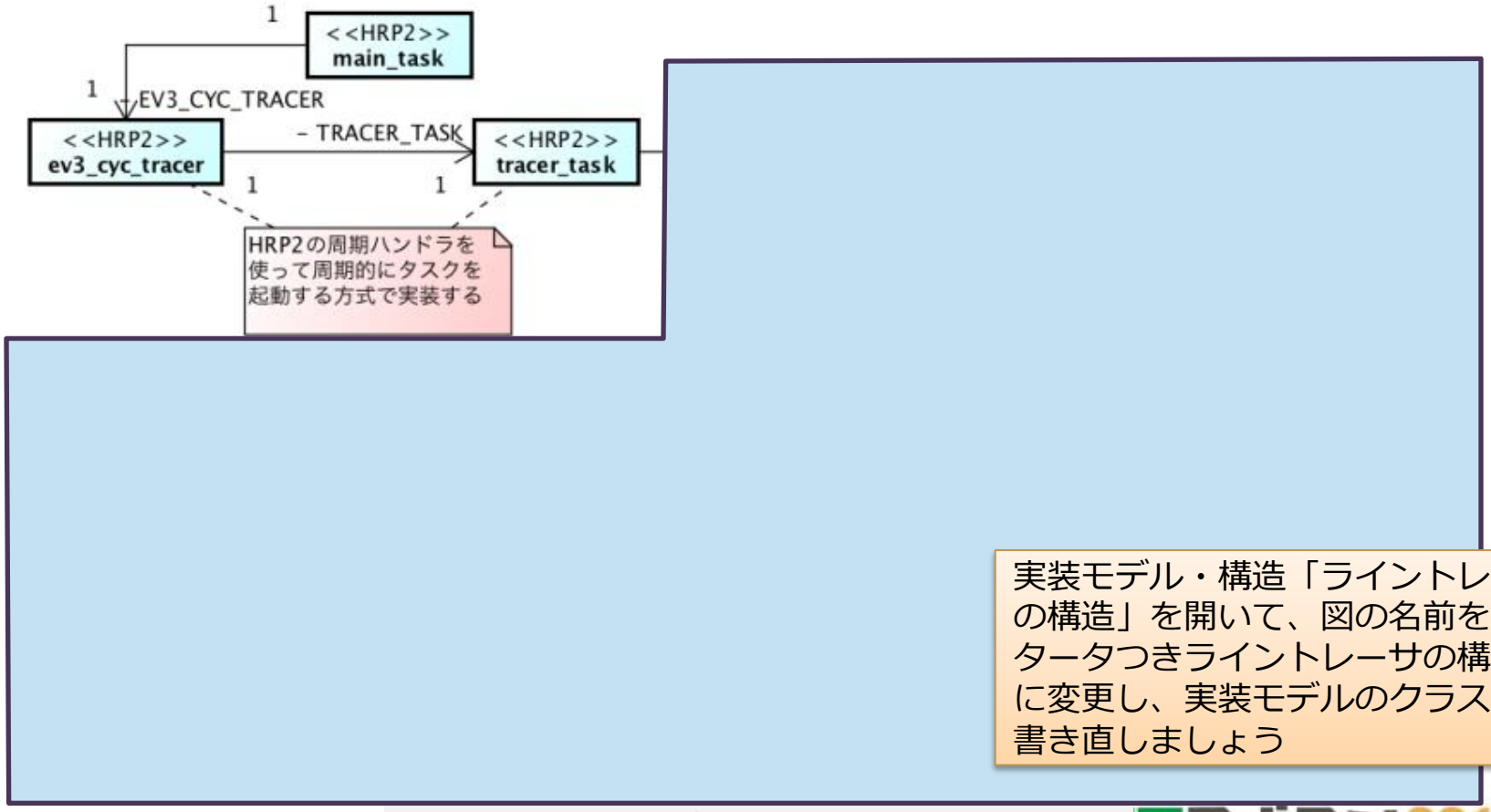
詳細設計

- 構造に関するアーキテクチャを整理します
  - app（ライントレーサ）、unit（ライン監視部や倒立走行部）、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
    - どこでライントレーサの１周期分の処理を呼び出すか考える
- 提供されたライブラリのクラスやAPIに読み替えます
  - EV3: 追加で使用するデバイスの呼び出しにEV3Rt C++ APIを使う
- 自分たちが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
  - 追加したクラスを英語名にする
    - 今回のように設計モデルと実装モデルが1対1にならず、実装モデルの複数のクラスに対応づけられる場合もあります

5-13. 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のクラス図を作成します
  - 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



実装モデル・構造「ライントレーサの構造」を開いて、図の名前を「スタータつきライントレーサの構造」に変更し、実装モデルのクラス図を書き直しましょう

（クラス図）

5-13. 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

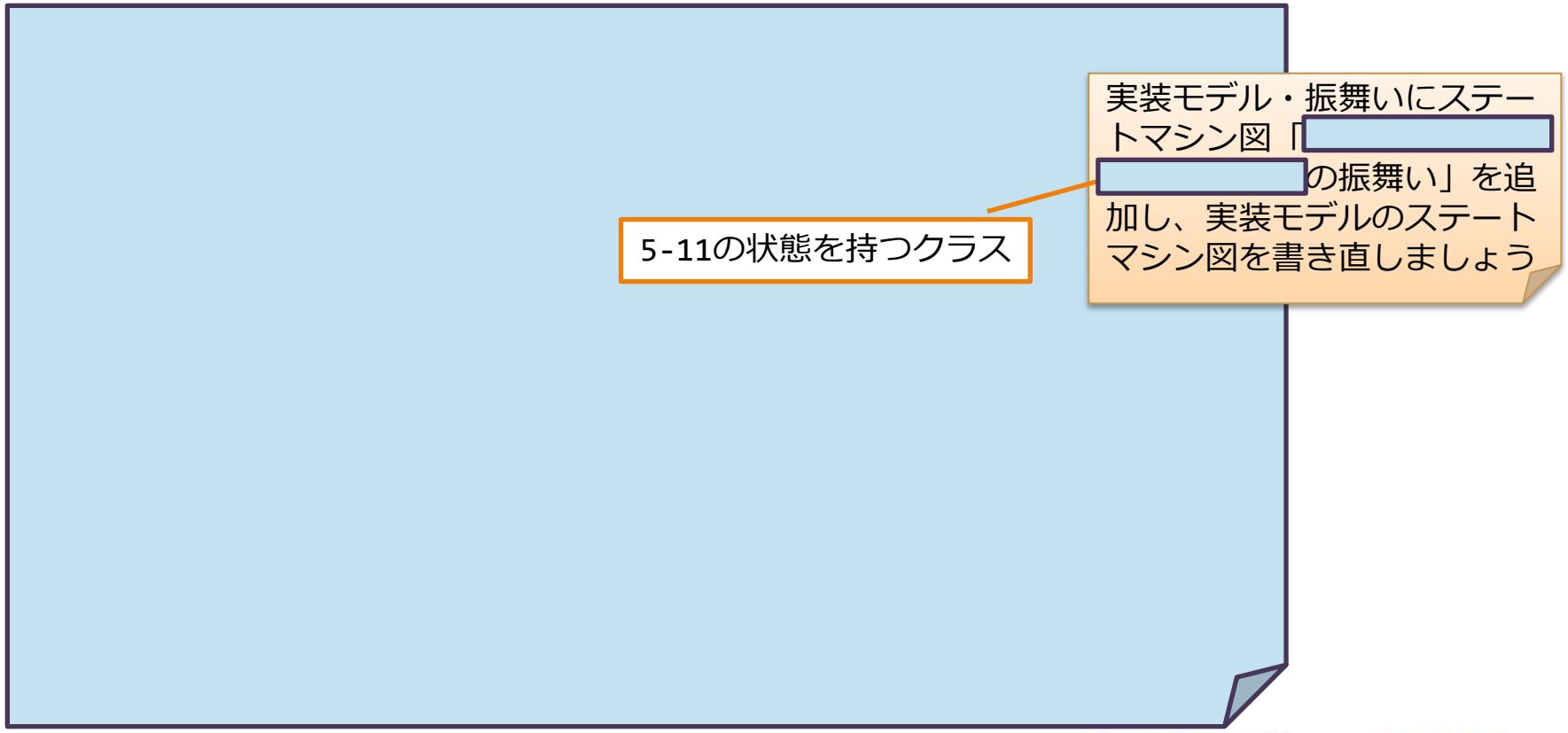
実装モデル・構造「ライントレーサのオブジェクト」を開いて、図の名前を「スタータつきライントレーサのオブジェクト」に変更し、実装モデルのオブジェクト図を書き直しましょう

（オブジェクト図）

5-13. 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
- 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

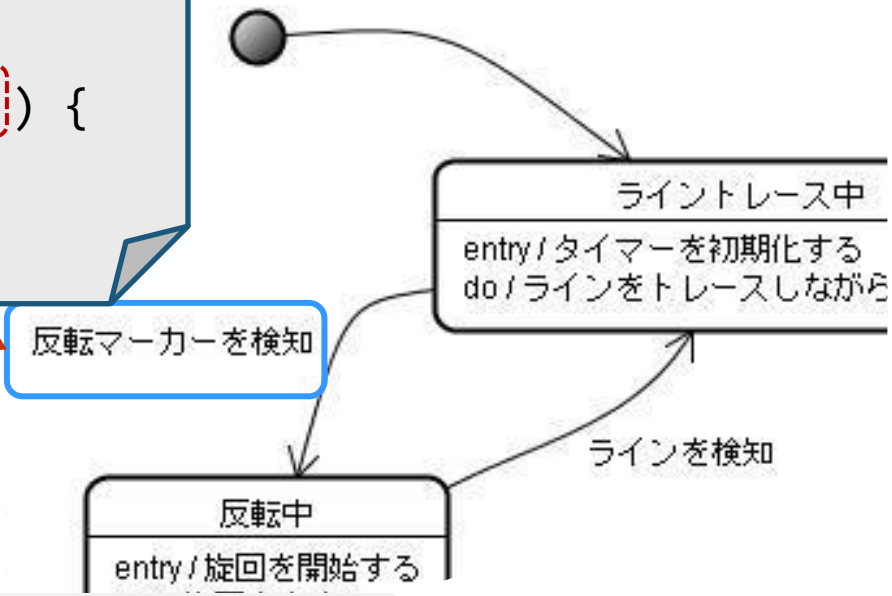


5-14. ステートマシンの実装を検討する（１）

- この演習では、イベント発生をポーリングで実装しましょう
  - 各状態の中で、監視対象を繰り返し監視します
  - もし監視対象に変化が起きたら、イベントの発生とみなします
  - イベントクラスやイベントキューを作らずに実装できます

「ライントレース中」は反転マーカーを監視し、
マーカーを見つけたら検知したとみなし、
現在の状態を「反転中」にする

```
if( marker_detector.isDetected() ) {
  current_state = 反転中;
}
```



イベント発生チェックのソースコードとモデル図の対応づけの例

5-14. ステートマシンの実装を検討する（2）

- ステートマシン図とソースコードの関係を決めましょう
  - switch-caseで実装する方法を使います
- ステートマシン図が所属するクラスに情報を追加します
  - 実装する時の技術的な情報なのでクラス図には明記しないルールにします
  - ステートマシンを実行するメソッドを決めます（なければ追加します）
  - 現在と次の状態を保持する属性を追加します
- ステートマシン図に作成した状態を追加します
  - ヘッダファイルに状態のenumを作成し、UNDEFINEDも追加します
  - コンストラクタで現在と次の状態を保持する属性を初期化します
- ステートマシン図を実現するメソッドを実装します
  - ステートマシン図の状態、イベント、アクションに対応づけながら実装します
- 入場動作、実行活動、退場動作、イベントのチェックと次の状態を求める操作を実装します
  - 自己遷移の場合には、次の状態にSELFを設定します
- 状態を遷移する処理と、ステートマシンとしての全体の処理を実装します

5-15. 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
  - 「app」と「unit」に配置するクラスを考える
- 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

5-15. ソースコードを書く（2）

EV3

実装

- ヘッダファイルを作成しましょう

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

変換ルールに従って
5-13で追加したクラスのヘッダファイルを作る

5-15. ソースコードを書く（3）

EV3

実装

- CPPファイルを作成しましょう

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

変換ルールに従って
5-13で追加したクラスのcppファイルを作る

5-16. オブジェクト図を実装する

EV3

実装

- 追加したクラスのオブジェクト（インスタンス）を作成します
  - app.cppに、新しいアプリケーションのクラスのヘッダファイルをインクルードします
    - 前のアプリケーション（ライントレーサ）のヘッダは、インクルードをやめます（なぜ？）
  - オブジェクト図で使用しているオブジェクトを作成します
  - オブジェクト間のリンクが作成できていることを確認します

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

変換ルールに従って
5-13で追加したクラスのインスタンスを作成する

5-17. タスクの処理を修正する

EV3 ET ROBOT CONTEST

実装

- trace_taskの処理を修正します
  - アプリケーションのタスクは、例題1のときはライントレーサを動かしていましたが、今度は ライントレーサを動かすクラス を動かします
  - タスクの処理に書いてあったLineTracer::run()を呼び出すコードを、
    “ライントレーサの１周期分の処理を呼び出すメソッド”を呼び出すコードに修正する

app.cpp

```
// ...

void tracer_task(intptr_t exinf) {
    if (ev3_button_is_pressed(BACK_BUTTON))
        wup_tsk(MAIN_TASK);
    } else {
        “ライントレーサの１周期分の処理を呼び出すメソッド”をここで呼ぶ
    }
    ext_tsk();
}
```

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

スライド中のコードは端折っていますので、詳細は、実際のソースコードで確認しましょう

5-18. ビルドできるか確認する

テスト

- 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
  - まだ、メソッドの中身を実装していないので、動作テストはメソッドの中身を実装してからやりましょう

```
Makefile

 (途中省略)
APPL_CXXOBJS += ¥
        LineMonitor.o ¥
        BalancingWalker.o ¥
        BalancerCpp.o ¥
        LineTracer.o ¥
        
 (以下省略)
```

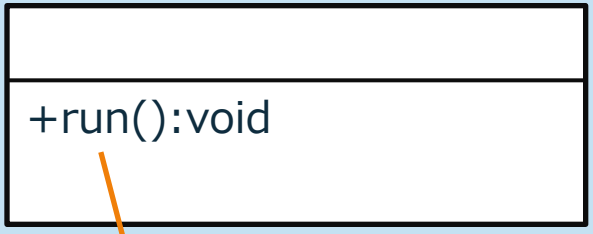
ここに、追加したクラスの実装ファイル（*.o）を追加してください

5-19. クラスの振舞いを実装する（1）

EV3

実装

- runメソッドをステートマシンを動かすメソッドとして実装しましょう
  - runメソッドは、周期ハンドラから繰り返し呼び出されることを思い出しましょう
  - UNDEFINEDはいずれの状態でもないことを表します



5-14で決めたステートマシンを動かすメソッド

変換ルールに従って
ステートマシンの状態遷移を
ソースコードに追加・修正する

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

5-19. クラスの振舞いを実装する（2）

EV3

実装

- 状態を持つクラス のヘッダファイルを編集しましょう
  - 状態を表すenumを定義します
  - 現在の状態を保持する属性を追加します
  - 状態ごとの処理用のメソッドを追加します

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

5-19. クラスの振舞いを実装する（3）

EV3

実装

- 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

5-19. クラスの振舞いを実装する（４）

EV3

実装

- クラス図を更新しましょう
  - ステートマシンから実装した操作や列挙型を追加します

5-13で作成したクラス図に
ステートマシンから実装した操作や列挙型を追加する

（振舞い追加後のクラス図）

5-20. ビルドして走行体を動かす

EV3

テスト

- 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(app)をMicroSDカードにコピーします
  - 転送手順については、4章を参考にしてください
  - 転送できないとき
    - MicroSDカードの空きが不足しているかもしれません
    - これまでの演習で使ったファイルを削除してから転送してみましょう

- 動作を確認しましょう
  - 作成したステートマシン図の通りに動作しているか確認します
  - 最初に作成したアクティビティ図の通りに動作しているか確認します

理解度チェック（５）

- システムに機能を追加する

1. 簡単な機能追加要求（改造）を実現するために、どの開発工程、どのモデルで何を行ったでしょうか？
   a. まずは
   b. 次に
   c. そして
   d. 　　：
2. ステートマシンで表現されたものを、どのようにソースコードへ変換していましたか？
3. 4章の例題に、5章の演習1と似たような設計ができそうなところはないでしょうか？
   a. 状態を使って表せそうなところを、ステートマシン図を使って設計してみましょう