大事なのでもう一回:

振舞いのコードもモデルを使って作成しよう

久保秋 真

株式会社チェンジビジョン

このセッションの概要

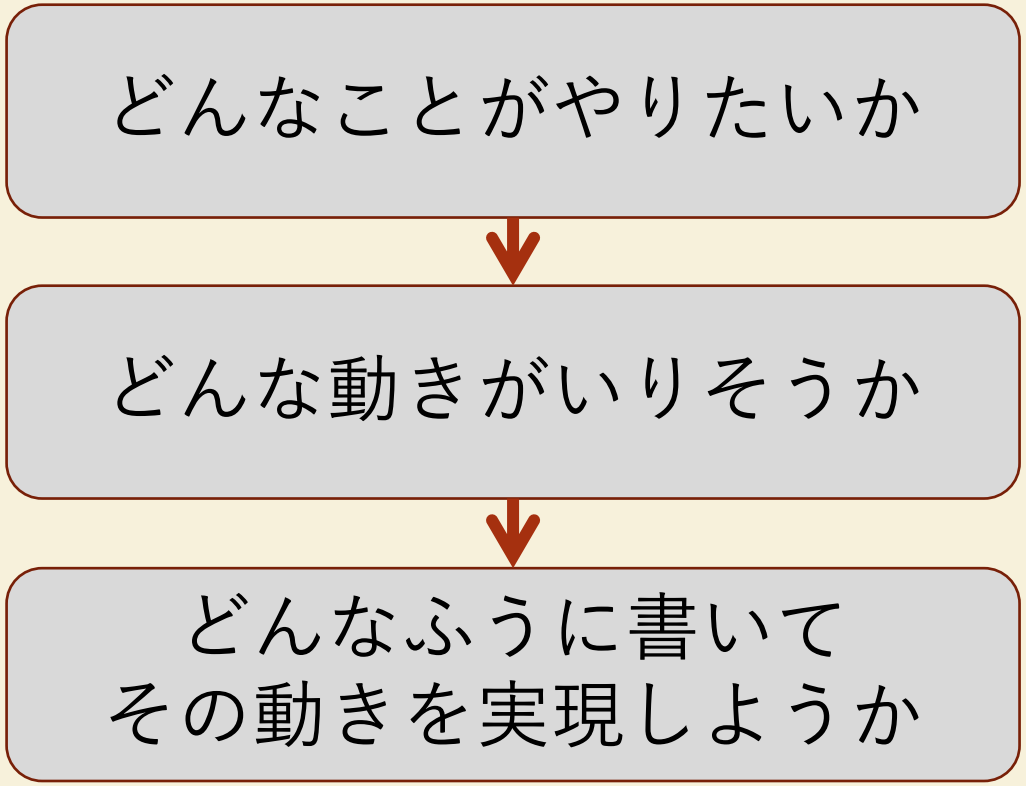
- 設計でモデル図を作成しても、どうして実装で乖離が生じてしまうのか
- モデルと対応したコードを安定して得るには？
  - いつ、なにを決めておけばよいのか
- 実例で確かめてみよう
  - Astah*のプラグインを使うと試せるよ

久保秋　真（くぼあき　しん）

- 株式会社チェンジビジョン勤務
- モデリングツール「Astah*」の開発支援
- UMLやSysMLを使ったモデリングのコンサルティングや技術教育の開発・講師
- ETロボコンの本部モデル審査員
- アジャイル開発とモデル駆動開発（MDD）の双方に関心
- 情報処理学会、日本ソフトウェア科学会各会員
- 早稲田大学理工学術院、日本大学生産工学部、関東学院大学理工学部
  各非常勤講師
- トップエスイー講師、スマートエスイー講師
- 日本雨女雨男協会IT本部長
- 日本あんこ協会あんバサダー

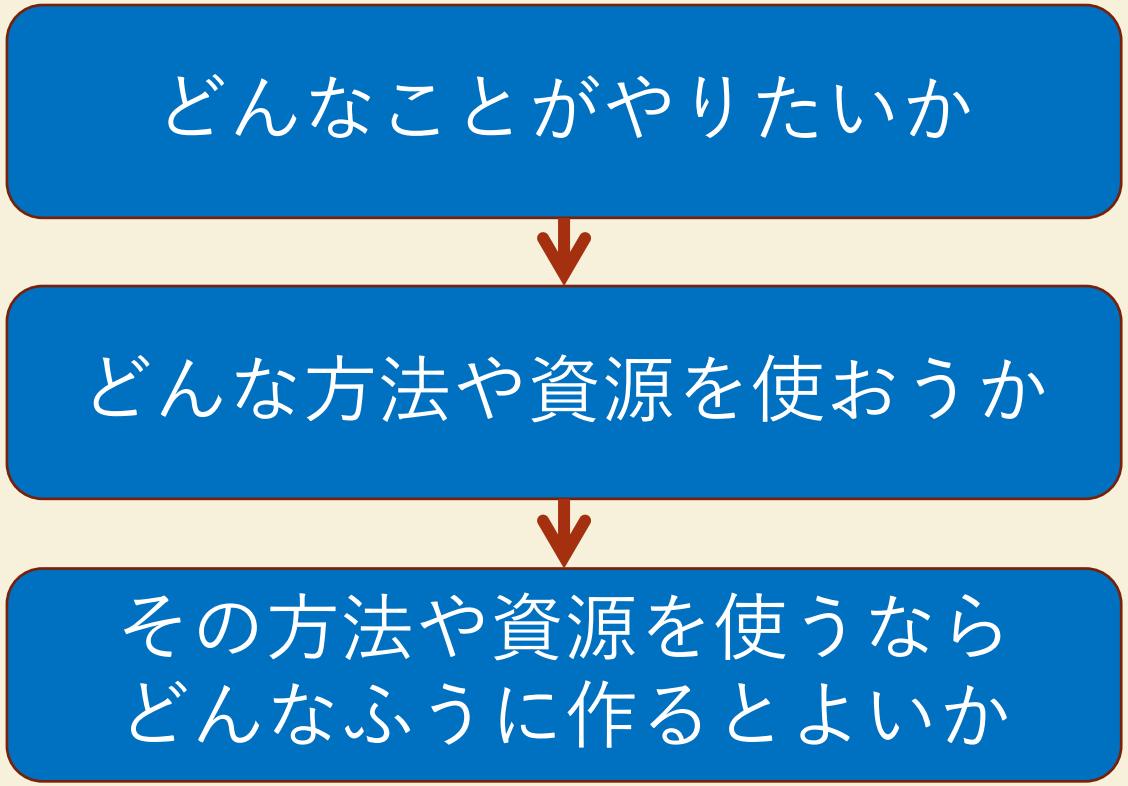
設計でモデル図を作成しても
どうして実装で乖離が生じてしまうのか

開発しているとき



- この順序だと、実装するときに
  「どのようにコードを書けば動くか」を、
  考えてしまうことになる

そうではなくて…



- を考えるようにする方が、設計と実装の乖離は少なくなる

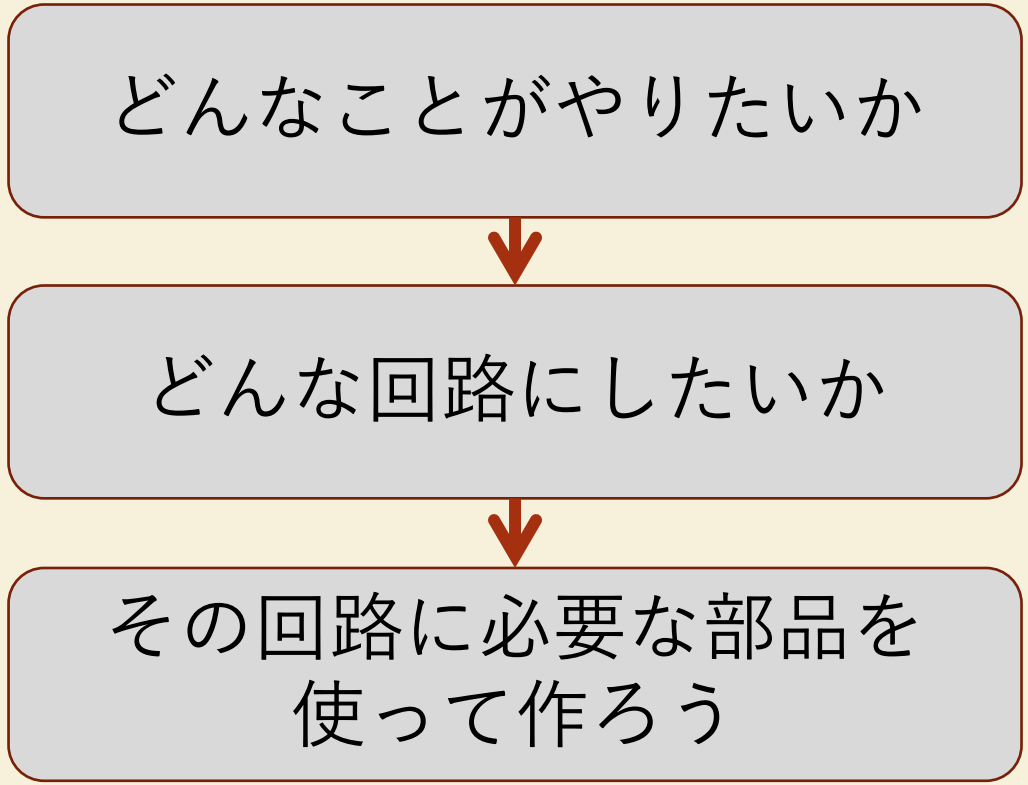
実際、できるプログラマは、

- どんな組み方があるかのストックを持っていて
- それを無意識に使っている

- つまり、できるプログラマは、
  前の手順でやってるようで、じつは後の手順を
  踏んでいる

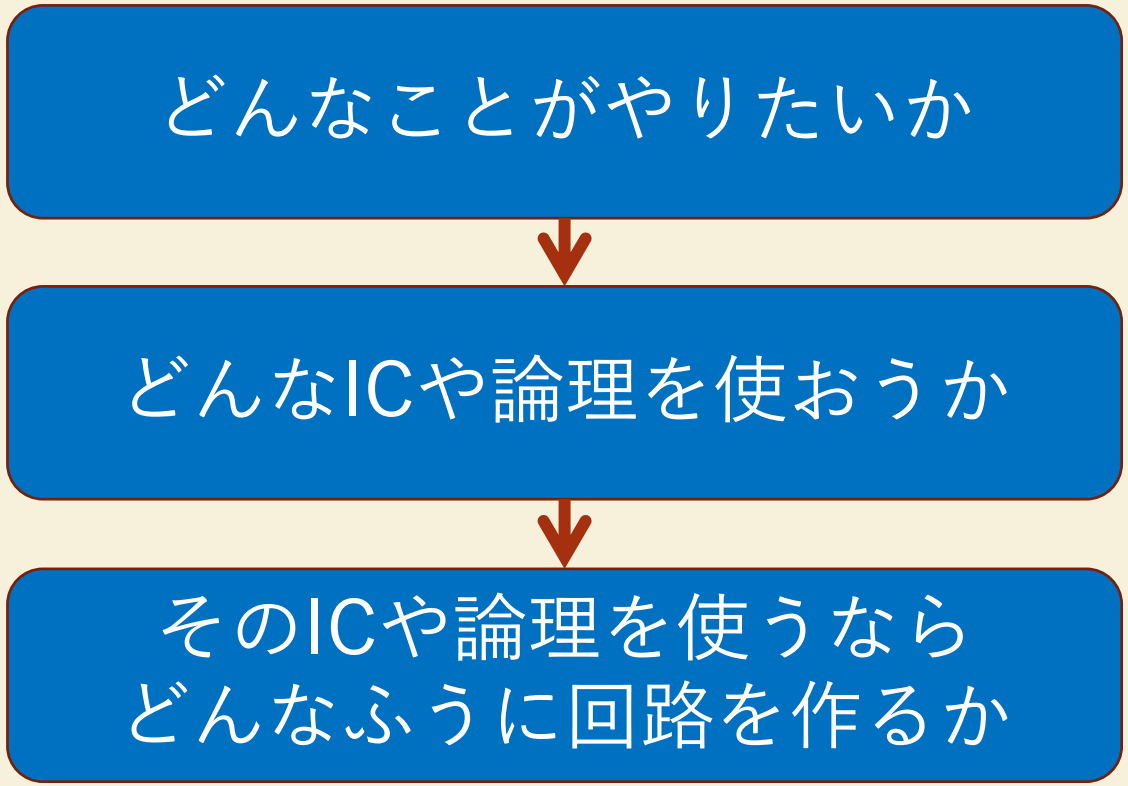
それは特別なことか？

- たとえば、電子回路の場合
- バラバラの
  部品を使うなら



それは特別なことか？

- ICを使うなら



- ANDとORで組まないで、NANDを使うことで使うICの種類を減らすとか

ソフトウェアの設計においても

- いまは実装する人が思案していることを設計の段階でやっておくべきではないか
- だとすると、設計するときに考えることも変わってくる

振る舞いのモデルと実装ではどうか

- 同じステートマシン図なら同じコードになる？
  - その設計者が期待した動作ができるか
  - その設計者が期待したコードの書き方ができるか
  - 他の人に任せたとき、自分と同じコードを書いてもらえるか
- イベントの扱いは決まっている？
  - イベントフラグ使う？それともイベントキュー使う？
- コードから元のステートマシン図が書けるか？
  - 状態、遷移、アクションなどがコードから読み取れるか

振る舞いの実装方式が決まっていない

- 実装者によってコードの書き方が変わっている
  - コードができてこないと、どんな実装になったかかわからない
- ステートマシン図と同じように振る舞うが、図と対応していないコードを書いている
  - 図の要素と対応していない実装だと、障害が見つかっても、そのコードを直して対処するので、図に戻ってこれない
- 設計と実装が乖離してしまうのは、振る舞いのモデルに対応した実装方式を決めていないのが原因

乖離を生じないためには
どんな工夫をすればよいのか

振る舞いの実装方式を決めよう（設計の仕事）

- ステートマシン図の要素を登場させる
  - 状態、イベント、状態遷移、アクション、ガード条件
  - どのクラスのどのステートマシン図でも同じ書き方にする
- ステートマシン図の働き方を決める
  - イベントの作り方、溜め方、マシンの繰り返し処理の実現方法
- どのステートマシン図も同じ規則でコードを書く
  - コードを書くときになって書き手が決めない
  - 書く人や、作るクラスによって、勝手に書き方を変えない

設計者は規則を意識して振る舞いを設計する

- 振る舞いのモデルを書く前に規則を決める
  - ステートマシン図とコードの対応規則を決めるのは設計の仕事
- 振る舞いのモデルは好き勝手にかけない
  - 規則に従ってコードを書いたら、そのコードが期待する動作をするように図を書く必要がある

（備考）構造にも似た面はある

- インスタンスが1つか複数か
  - インスタンスが1個なら、変数1つで済む
  - 複数なら、それらを格納する方法（配列orリスト？）が必要
- 集約かコンポジットかで実装構造が変わる
  - リンク（内部要素のデータへのポインタ）を持たせるか
  - 内部要素の正味をメンバ変数として保持するか
- 多重度に0を含むかどうか
  - 0になったとき、それが正常な場合か、異常な場合かが変わる

ここまでのまとめ

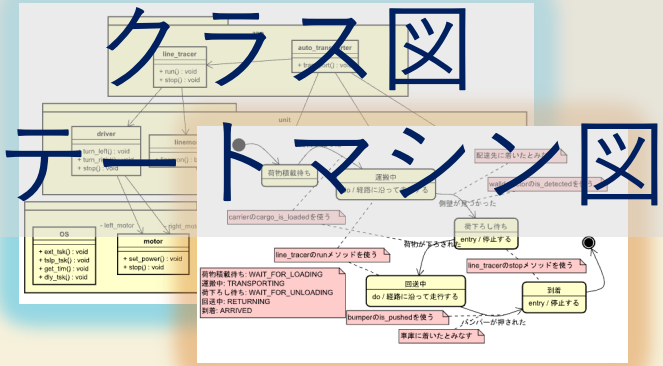
- 設計と実装の乖離の原因は実装時の自由さ
  - 振る舞いのモデルを実装するときに、実装方式をプログラマに任せていたから
- 乖離を減らすにはモデルを実装する規則を使う
  - 構造のモデルと振る舞いのモデルに加えて、振る舞いのモデルとコードを対応づける規則を用意する
  - 規則に従って実装することで、モデル図とコードの要素の対応関係が維持される

変換による実装

- モデルとコードの対応づけを設計の責務にする
- 設計側はこの方式を使うことを前提に設計する

クラス図

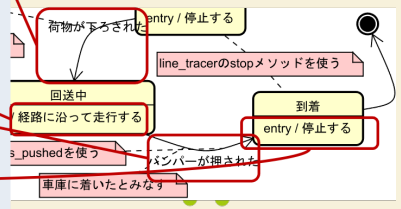
ステートマシン図



変換による実装

図からコードへの変換規則

```
void auto_transporter_transport(void) {
  switch(p_state) {
  // 略
 case WAIT_FOR_UNLOADING:
    line_tracer_stop();
    if (! carrier_cargo_is_loaded()) {
      p_state = RETURNING;
    }
    break;
  case RETURNING:
    line_tracer_run();
    if (bumper_is_pushed()) {
      p_state = ARRIVED;
    }
    break;
  case ARRIVED:
    line_tracer_stop();
    break;
  }
```




```
void auto_transporter_transport(void) {
  switch(p_state) {
  // 略
 case TRANSPORTING:
    line_tracer_run();
    if (walldetector_is_detected()) {
      p_state = WAIT_FOR_UNLOADING;
    }
    break;
  case WAIT_FOR_UNLOADING:
    line_tracer_stop();
    break;
  }
}
```

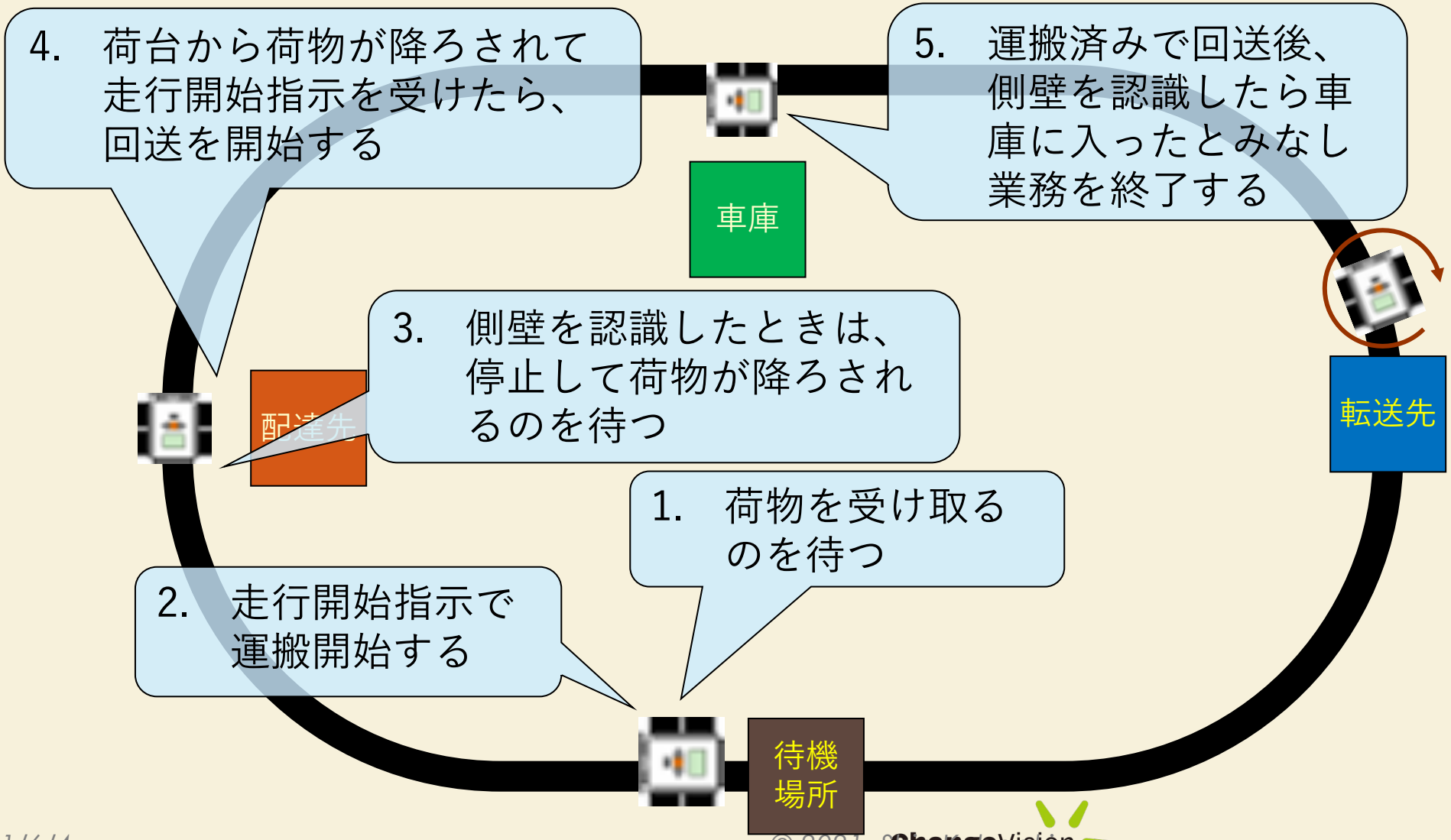
実装コード

具体的なモデルとコードで
変換による実装の実際をみてみよう

対象のシステム

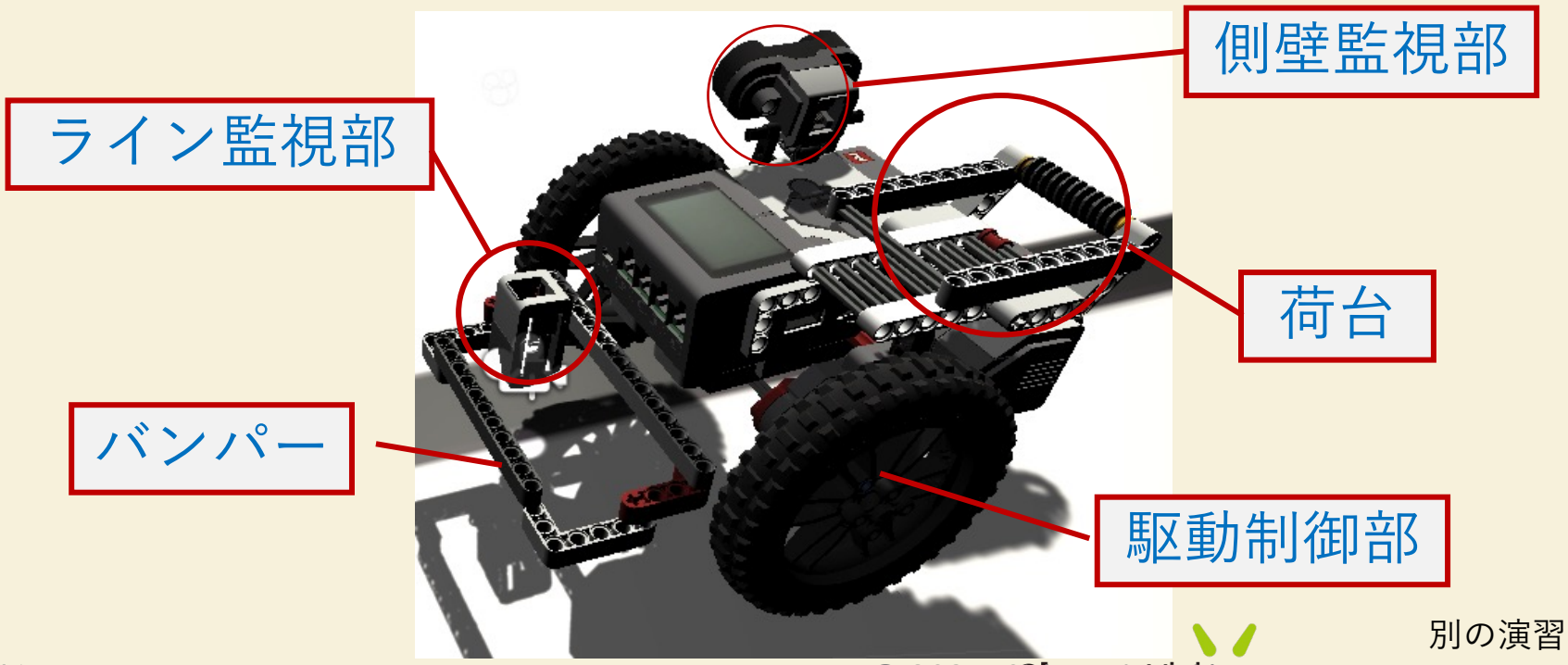
- 自動搬送システム
  - 状況に応じて「配達」「転送」「回送」の各業務を処理する
- 配達業務
  - 待機場所で荷物を受けるのを待つ
  - 走行開始指示を受け取ったら、配達先へ荷物を運搬する
  - 配達先に到着したら停止する
  - 荷物が下ろされるのを待つ
  - 荷物が下ろされたら、車庫へ回送して停止する

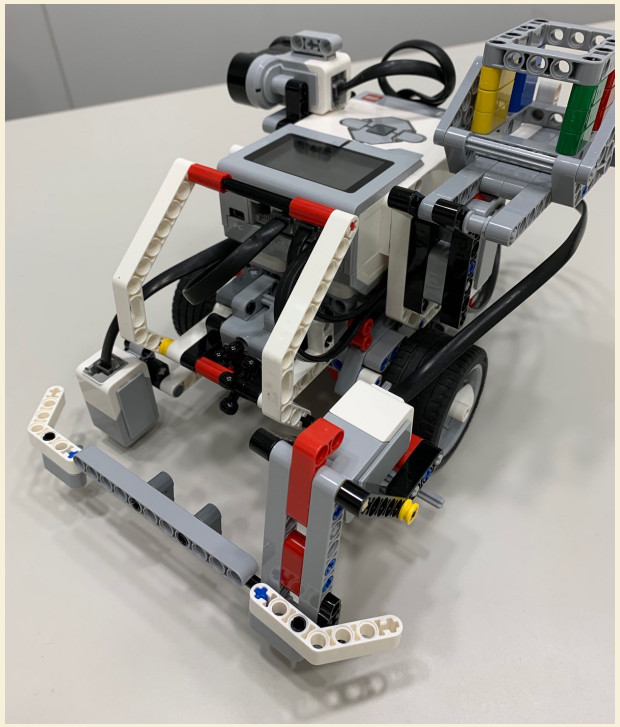
業務の流れ



使用する機器

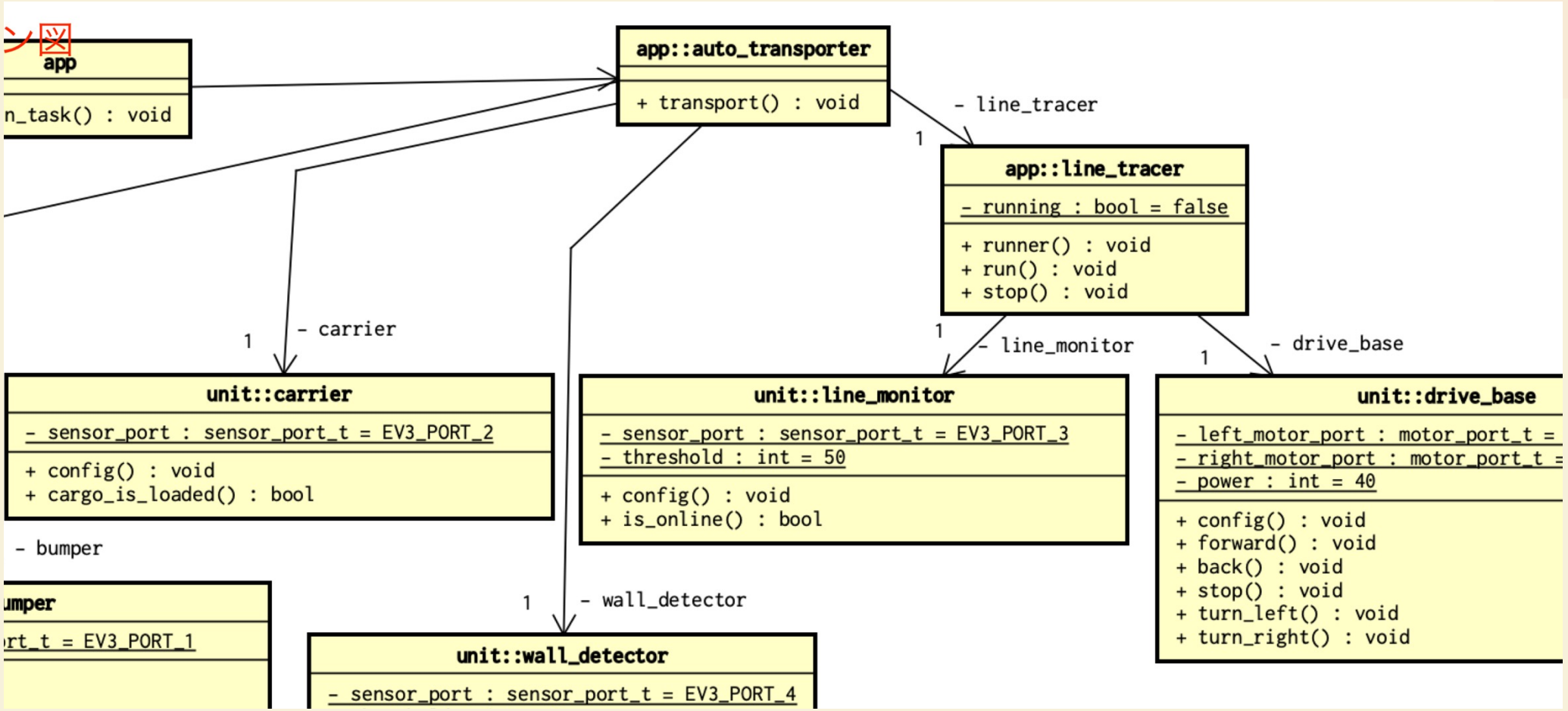
- 荷物を運搬する自動搬送ロボット
  - 駆動制御部による走行
  - 荷台、ライン監視部、側壁監視部、バンパーを装備



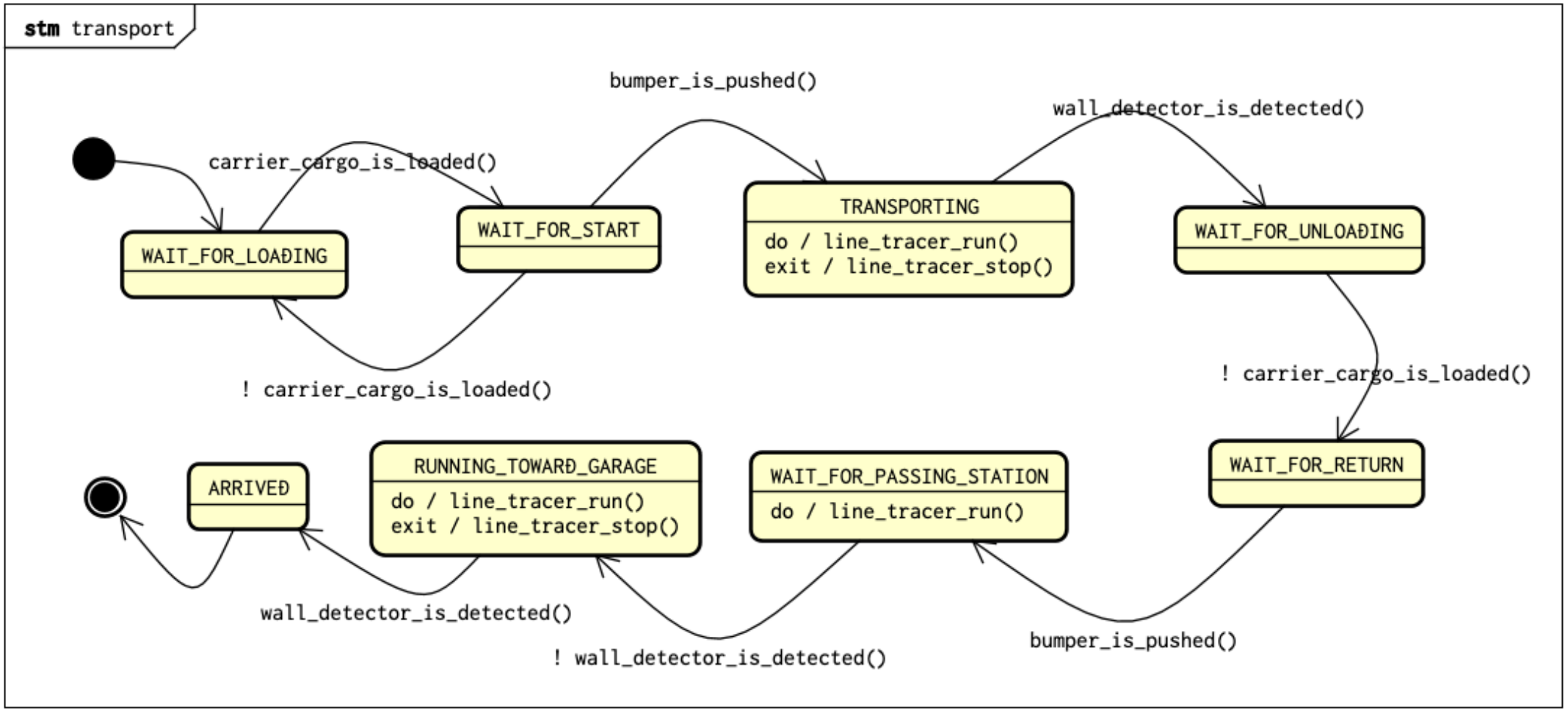


別の演習：https://youtu.be/A24WsSucrKw

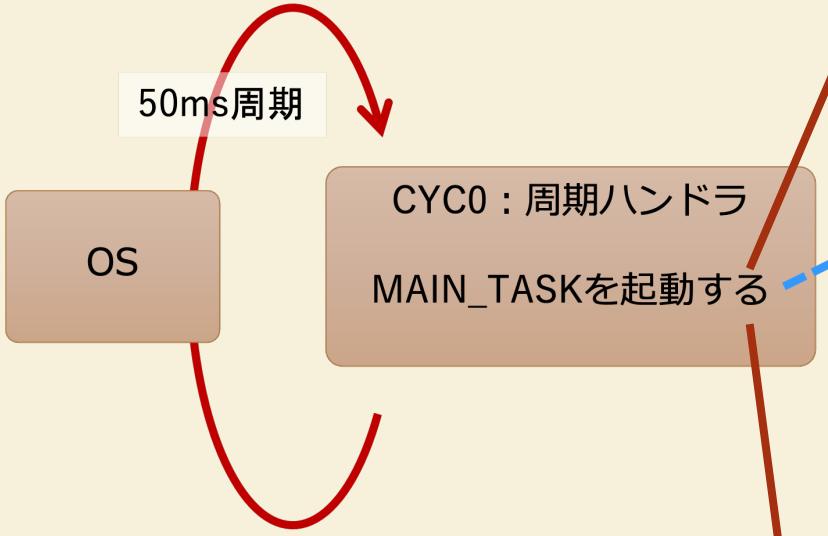
システムのクラス図



運搬業務のステートマシン図

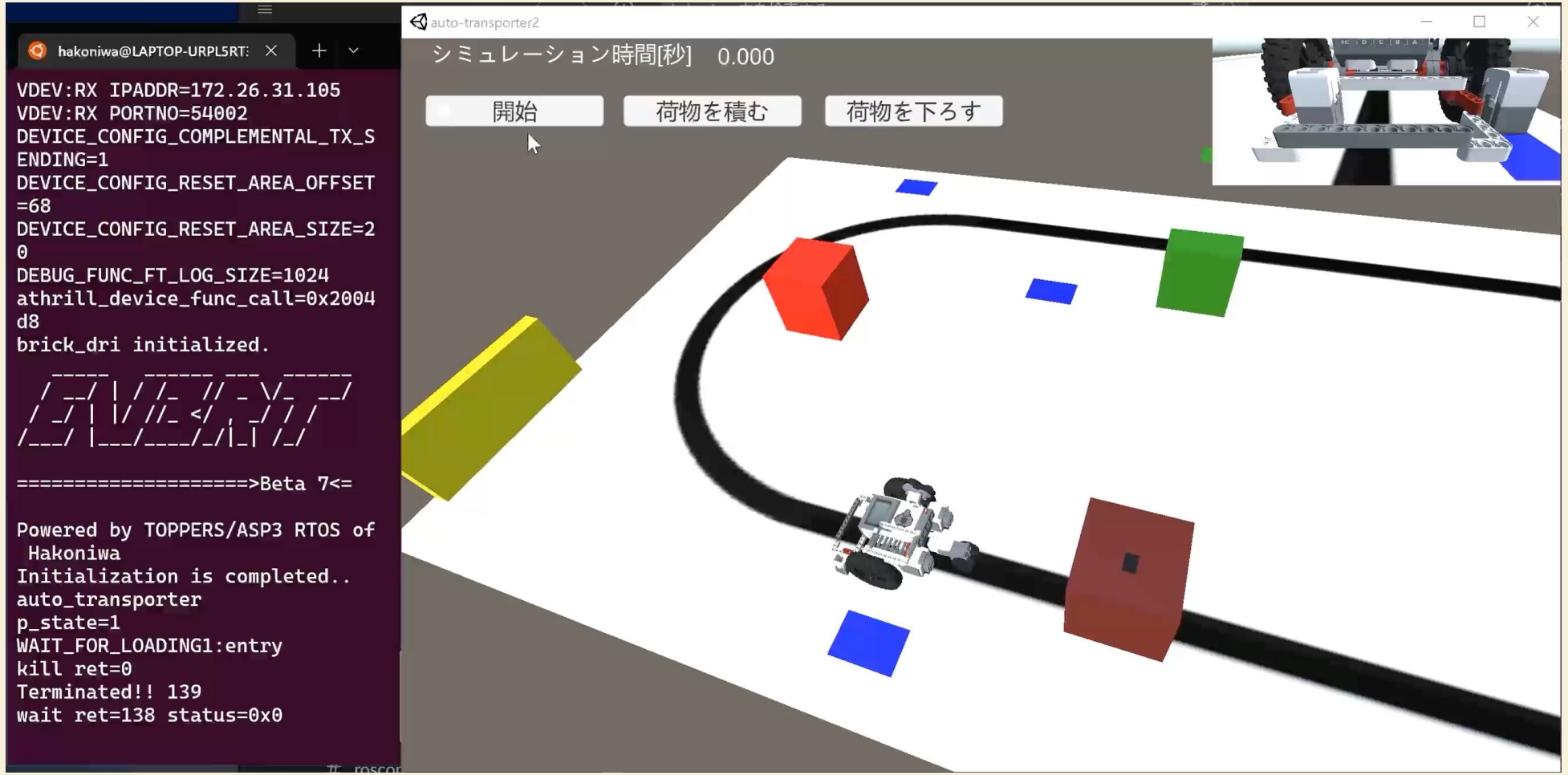


手書きのコード




```
void auto_transporter_transport(void) {
  switch(p_state) {
  case INIT:
    auto_transporter_config();
    p_state = WAIT_FOR_LOAĐING;
    break;
  case WAIT_FOR_LOAĐING:
    if (carrier_cargo_is_loaded()) {
      p_state = TRANSPORTING;
    }
    break;
  case TRANSPORTING:
    line_tracer_run();
    if (walldetector_is_detected()) {
      p_state= WAIT_FOR_UNLOAĐING;
    }
    break;
```

手書きコードのデモ



どんなふうにして
モデルとコードを対応づけるのか

コードの書き方なんて一般化できるの？

- どんな場合にも対応するような一般化は難しい
  - きっと周りくどく、扱いにくいものになってしまうだろう
- しかし、自分達のコードは似ていないか？
  - 同じ機器の別のアプリ
  - 派生品のコード
- 似ているもの同士なら、一定の規則で済ませられそうだ

既存のコードから始める

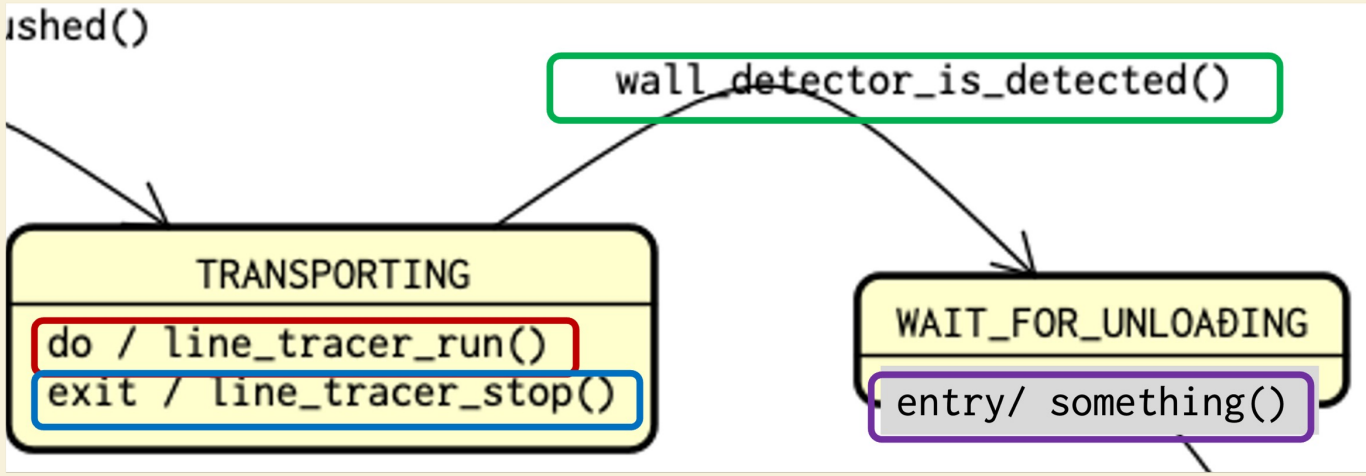
- 機能しているコードをテンプレート化する
  - 可変する部分に、外部のデータを埋め込めるようにしておく
- モデルでは埋め込みに必要なデータを決める
  - モデルはそのテンプレートに必要な情報を書く
- 埋め込むと欲しいコードになることを確認する
  - モデルからデータを取り出してテンプレートに埋め込んだら、それが機能しているコードになればよい

Astah m2tプラグイン

- テンプレートにモデルのデータを埋め込んで、ソースコードを生成するプラグイン
- 細合さんがちょっと昔に作った代物
  - LED-Campでも使っていた
- テンプレートによって変換規則を実現したものとみなせる

手書きコードとモデル図を対応づける

```
void auto_transporter_transport(void) {
    switch(at_state){
    // ....
    case ATST_TRANSPORTING:
        line_tracer_run();  // do action
        if(wall_detector_is_detected()){
            line_tracer_stop();    //  exit action;
            at_state = ATST_WAIT_FOR_UNLOADING;
            // 次の entry actionがあれば
        }
        break;
```



対応づけをテンプレートにする

```
<% if(u.statemachine){%>\
void ${u.name}_${u.statemachine.stateMachineDiagram}(void) {
    switch(${stateInstance}){
<% for(state in u.states){%>\
    case ${getStateName(state)}:
<% if(state.getSubvertexes().length!=0 && state.getSubvertexes().find{it instanceof IPseudostate}!=null){%>\
        ${stateInstance} = ${getStateName(state.getSubvertexes().find{it instanceof
        IPseudostate}.outgoings[0].target)};
        break;
<%}else{%>\
<% if(state.doActivity.trim()!=""){%>\
<% for(doAct in state.doActivity.normalize().split("\n")){%>\
        ${replaceAct(doAct)};  // do action
<%}%>\
<%}%>\
<% for(t in state.outgoings){%>\
        if(<%if(t.guard.trim()!="") {out<<"("+t.guard+") && "}%><%if(t.event.trim()=="true"||
        t.event.trim()==""){%>true<% }else if(t.event.trim().toLowerCase()=="timeout"){%>${u.name}
        _Timeout==event<%} else {%>${t.event.trim()}<%}%>){
<% if(state.exit.trim()!="") {%>\
<% for(exitAct in state.exit.normalize().split("\n")){%>\
            ${replaceAct(exitAct)};    //  exit action;
<%}%>\
<%}%>\
<% if(t.action.trim()!="") {%>\
<% for(transAct in t.action.normalize().split("\n")){%>\
            ${replaceAct(transAct)};  // transition action
```

switch

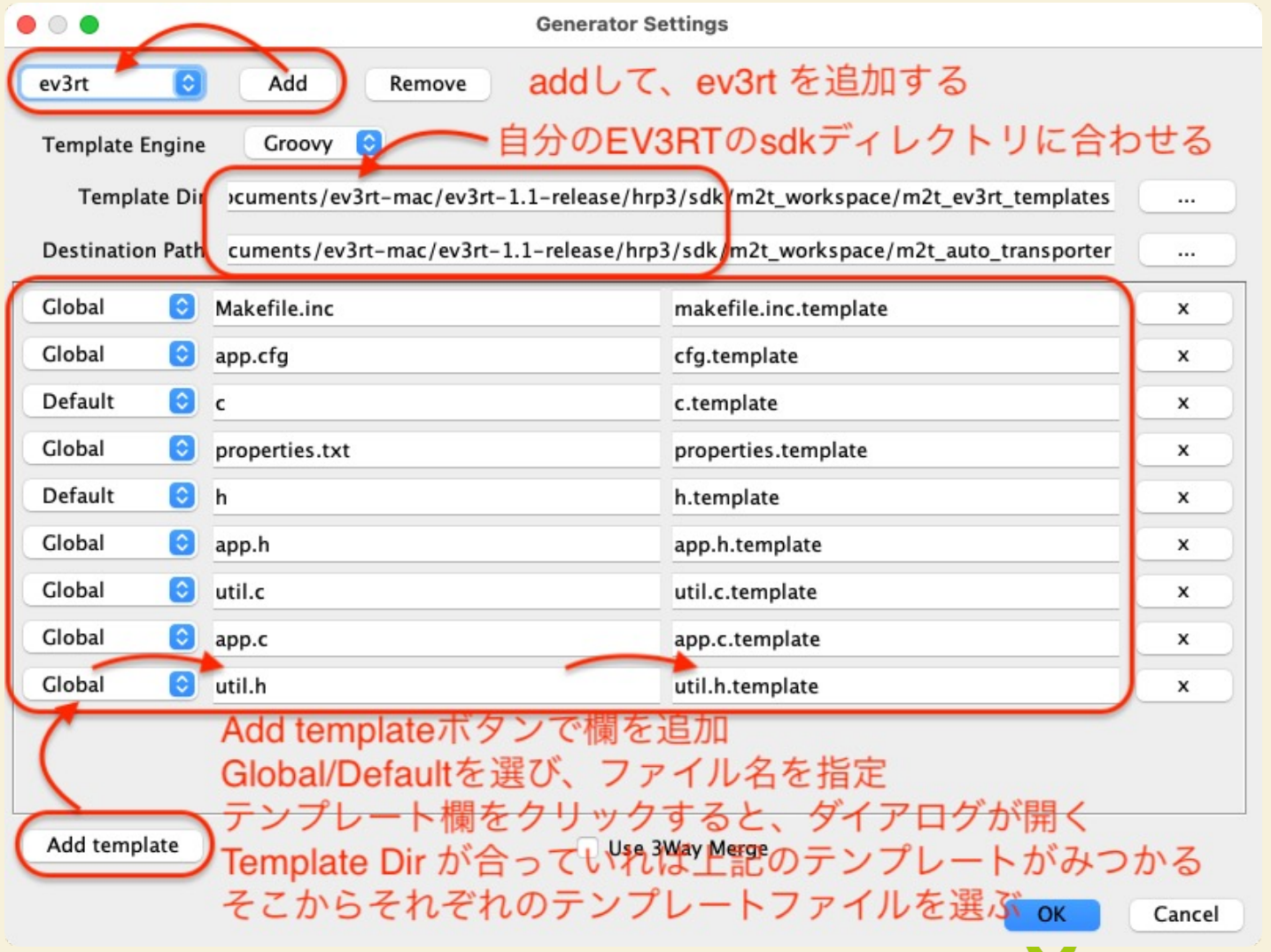
case

do activity

event check

exit action

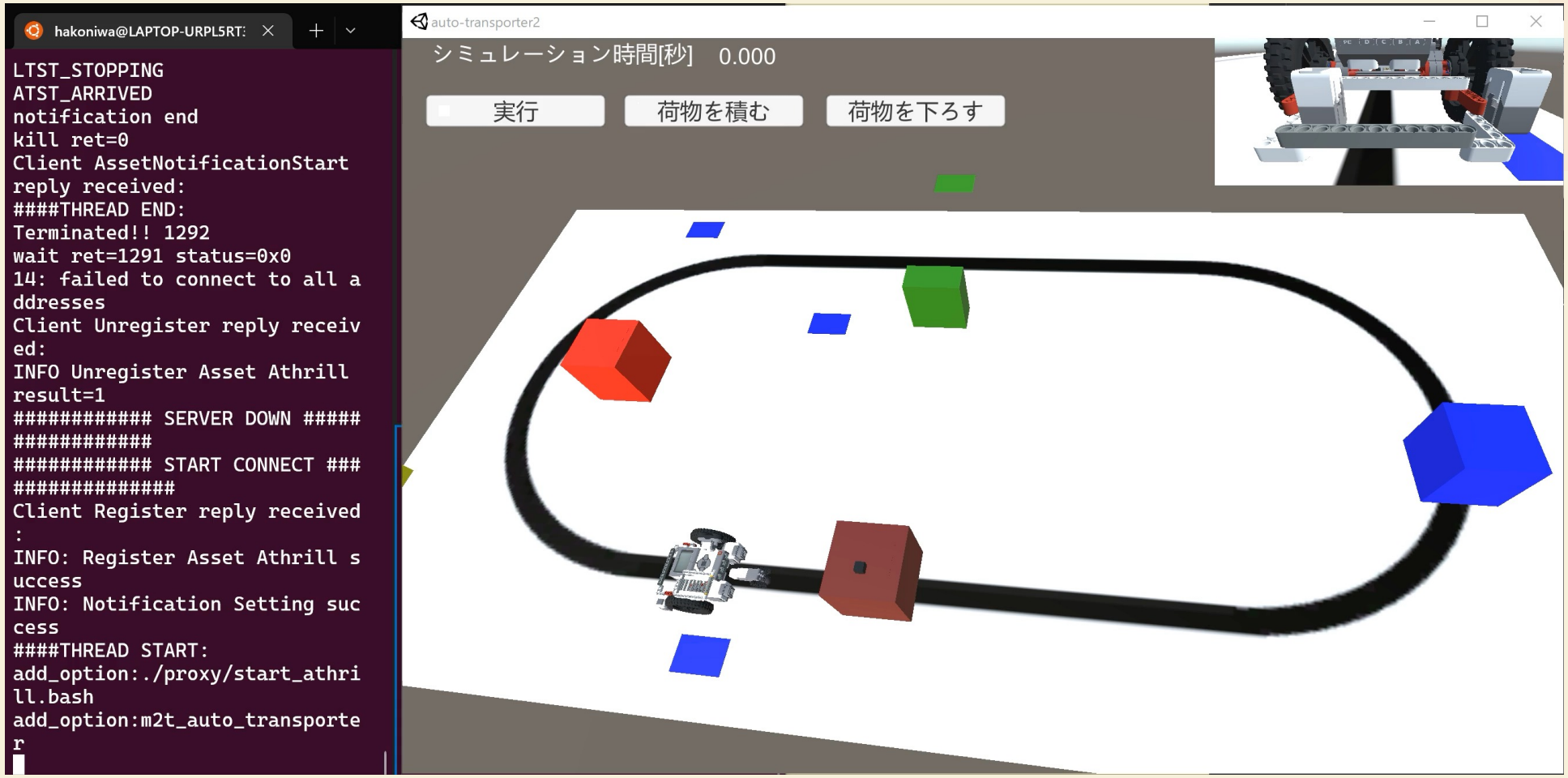
テンプレートをm2tの設定へ反映



生成したコード

```
50  void auto_transporter_transport(void) {
51      switch(at_state){
52      case ATST_WAIT_FOR_LOAĐING:
53          if(carrier_cargo_is_loaded()){
54              at_state = ATST_WAIT_FOR_START;
55              msg_f(auto_transporter_state_names[at_state], 1);
56          }
57          break;
58      case ATST_TRANSPORTING:
59          line_tracer_run();  // do action
60          if(wall_detector_is_detected()){
61              line_tracer_stop();    //  exit action;
62              at_state = ATST_WAIT_FOR_UNLOAĐING;
63              msg_f(auto_transporter_state_names[at_state], 1);
64          }
65          break;
```

生成したコードのデモ


hakoniwa@LAPTOP-URPL5RT:
LTST_STOPPING
ATST_ARRIVED
notification end
kill ret=0
Client AssetNotificationStart
reply received:
####THREAD END:
Terminated!! 1292
wait ret=1291 status=0x0
14: failed to connect to all a
ddresses
Client Unregister reply receiv
ed:
INFO Unregister Asset Athrill
result=1
############ SERVER DOWN #####
############
############ START CONNECT ###
###############
Client Register reply received
:
INFO: Register Asset Athrill s
uccess
INFO: Notification Setting suc
cess
####THREAD START:
add_option:./proxy/start_athri
ll.bash
add_option:m2t_auto_transporte
r
auto-transporter2
シミュレーション時間[秒] 0.000
実行
荷物を積む
荷物を下ろす

作成したテンプレートやコード

- GitHubに登録
  - https://github.com/kuboaki/m2t_workspace
  - READMEにも説明がある

変換による実装

- 作成したテンプレート：変換規則
  - モデルから一定の決まったコードに変換する規則とみなせる
  - 属人化していないコード作成の知識や手法ともいえる
- 変換するように実装できる
  - モデルのデータをテンプレートに埋め込めばよい
  - 入力されるモデルは変わっても、変換する作業は同じ

変換による実装

- チームが保有する資源になる
  - コードの書き方は、個人のセンスや技能ではなくなる
  - できる人の技量や知見は、変換規則に反映する

- 埋め込むデータをすべてモデル図に盛り込めば、そこが設計の終わり

「障害の発生工程」の判定もクリアに

- 変換規則通り実装して障害があるなら
  - 設計の間違いとみなせる
- 変換規則通り実装していないなら
  - 設計に従っていないので、テストが通っていてもダメ
    - 一番最初のテストは、変換規則どおり実装しているかの確認
- 変換規則に間違いや不備がある場合
  - 規則を作る側＝設計側の責任

まとめ

- 振る舞いの実装が、設計と乖離する原因だった
- 先にモデルとコードとの対応づけを決めておく
  - クラス図だけでなくステートマシン図とコードも対応づける
- 設計では対応づけを前提にモデルを作成する
  - つまり、対応づけを決めるのは設計のしごと
- 対応づけがあれば、実装は機械的
  - だったら自動化できるよね？だからプラグインがあるよ

大事なのでもう一回:

振舞いのコードもモデルを使って作成しよう

— おしまい —

久保秋 真

株式会社チェンジビジョン