オペレーティングシステム

2022年10月11日

海谷 治彦

目次

- 人の情報処理モデル
- タスク分析
- 画面遷移図 状態遷移図
- ユーザーインタフェース設計
- プロトタイプ
- ヘルプと文書化
- UI標準とガイドライン
- 評価法

- 演習1

人の情報処理モデル

- UI設計に際して，人間がコンピュータとどのように相対しているかの模型(モデル)を作るのは有効.
- 以下の3つのモデルが代表的，特に最初のやつ.

- 認知情報処理モデル 1983
  - (model) human processor と呼ばれる，Card氏のモデル.
- ユーザー行為の7段階モデル 1990
- 行為の三階層モデル 1983
  - SRKモデル Skill, Rule, Knowledge

認知情報処理モデル

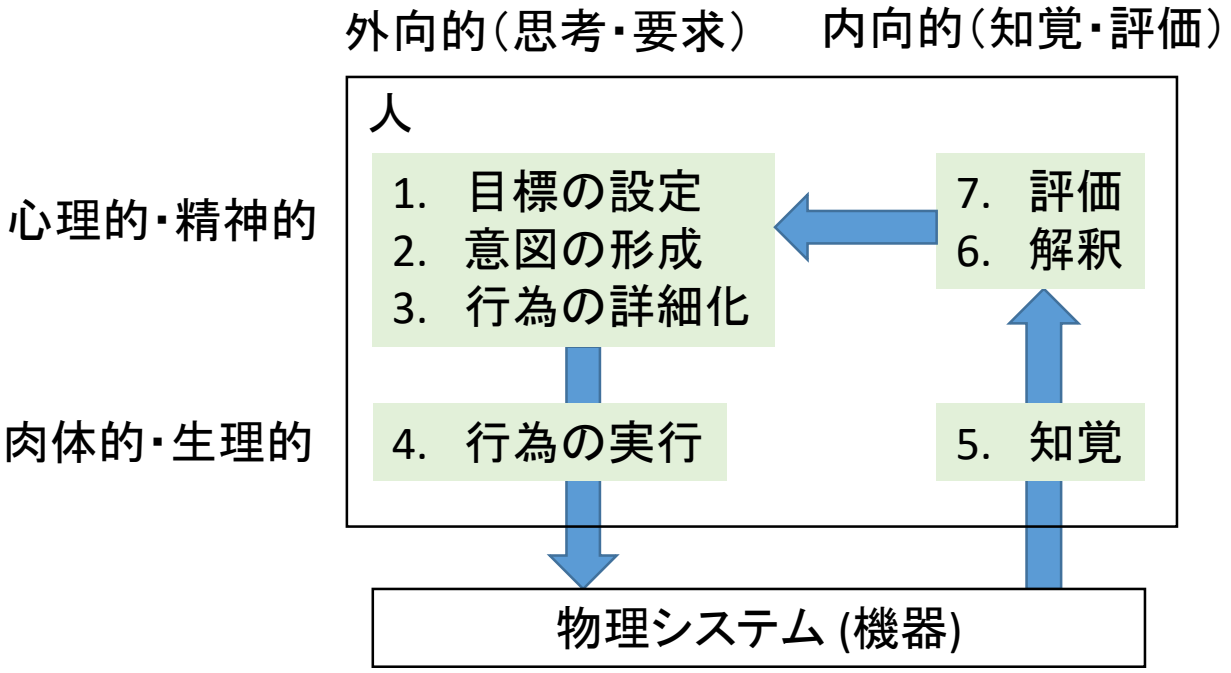
- 人が外界からの入力に反応する際，人の内部で，以下の3種類の処理を行っていると考えるモデル.

1. 知覚 音や映像等のデータを読み込む.
2. 認知 読んだデータの意味を解釈し応答法を決定.
3. 運動 解釈に基づき手や口等で応答を実施する.

- 実験データより，これらは合計数100ms程度の処理時間を使うらしい.
- よって，コンピュータの人への応答は，この時間程度が望ましい.
  - これより遅いと「のろい」と感じる.
- 尚，ビデオの 1 frame が1/30秒(33ms程度)だが，これが人には連続して見えることにも符合する.

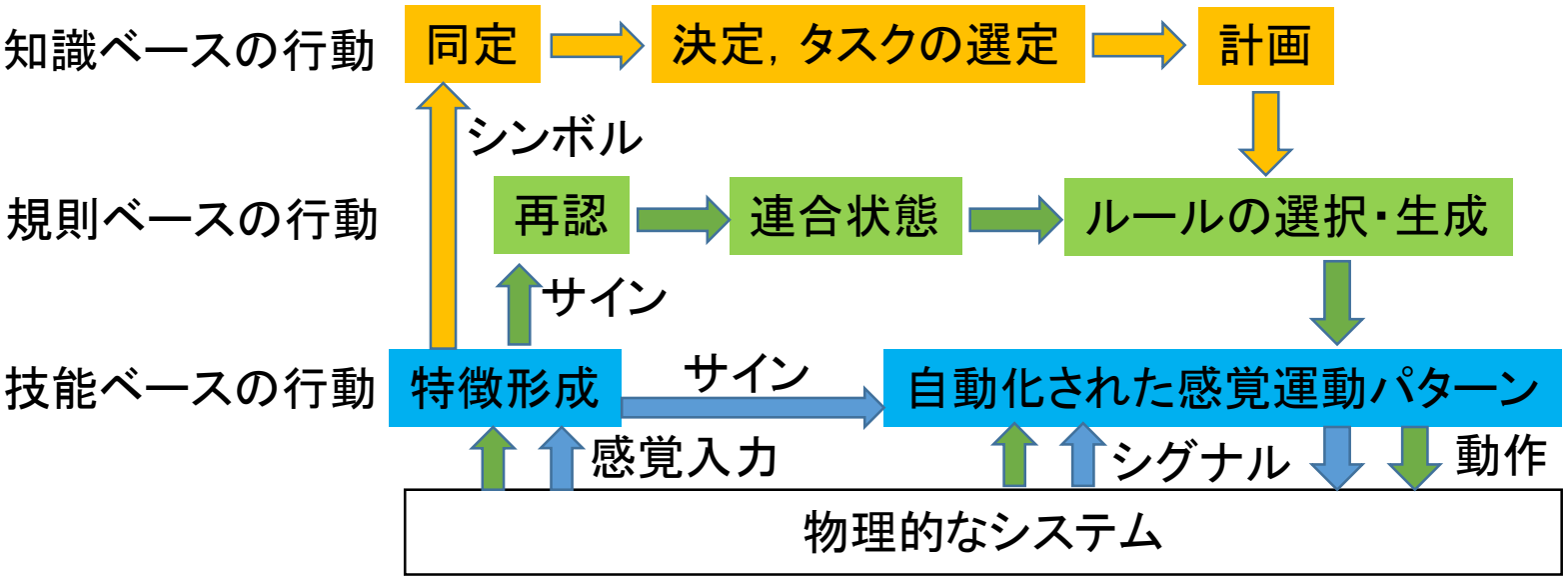
ユーザー行為の7段階モデル

- 人は外界の機械との相互作用を以下の7段階を繰り返して行っているとするモデル.
- いわゆる PDCA (Plan, Do, Check, Action)の拡張版ともいえる.



行為の三階層モデル

- 慣れてることは自然に行えて，初めてのことは難しいことを説明するモデル．
- 技能ベース 慣れてることは，ほぼ反射的に処理できる階層．「勝手に手が動く」かんじ．
- 規則ベース 内容は吟味せず，記憶中の規則に基づき対処する階層．「手が覚えてる」かんじ．
- 知識ベース 内容を吟味して対応する階層．



プレ・インタラクション設計

- インタフェース以前の問題として，あるアプリが，どんな機能を提供し，どんなユーザーとインタラクションするかを決めないといけない．
- HCIやUIとは関係なく，一般的なソフトウェア開発のモデルを使う．
  - ソフトウェア工学の授業とかぶる．
  - どんな機能が必要かについての探求は情報システム論で述べる．
- ユースケースモデルとユースケース記述
  - アプリやシステムの機能と，個々の機能に，どのようなユーザー(アクター)が関与するかまとめた図
  - 機能毎に，関与するアクターとシステムが実施することを順に列挙する．
- 状態遷移図 (ステートマシン図)
  - 画面遷移図にほぼ対応する．

ユースケース

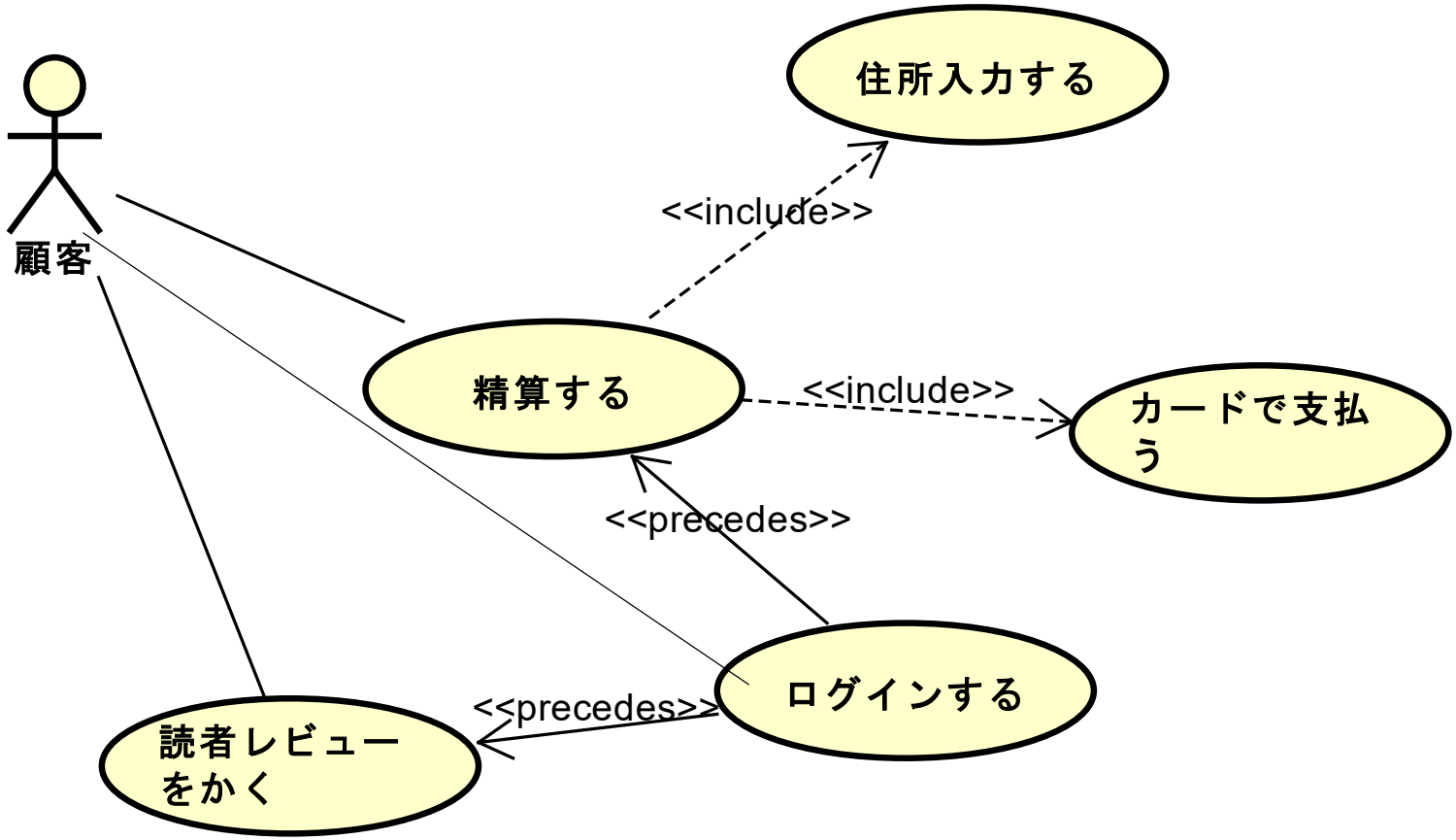
- 端的にいって，システムの機能を示す．
- その機能を遂行するために，システムとシステム外部の人や別システム(アクターと呼ぶ)が相互作用する様として仕様化する．

- ユースケースモデル
  - システム全体がどんな機能群を持ち，それぞれの機能遂行にどんなアクターがかかわるかを記述したもの．
- ユースケース記述
  - 個々のユースケースを遂行するにあたり，システムやアクターが順番に何をするかを記述したもの．

ユースケースモデル

- システムの機能を楕円でかき，
  - この機能ひとつひとつを「ユースケース」と呼ぶ
- その機能遂行に関連するアクターと線でつなぐ簡易な図.
- ユースケース間の関係はわりと沢山あるけど，本講義では，以下の二種類の利用のみ推奨する
  - includes ユースケースAがBを呼び出すこと.
    - サブルーチンや関数コールとほぼ同じ.
  - precedes AがBより先に起こること.
- ユースケースモデルではシステム内部のデータに相当するものは書いてはいけない.

ユースケースモデルの例

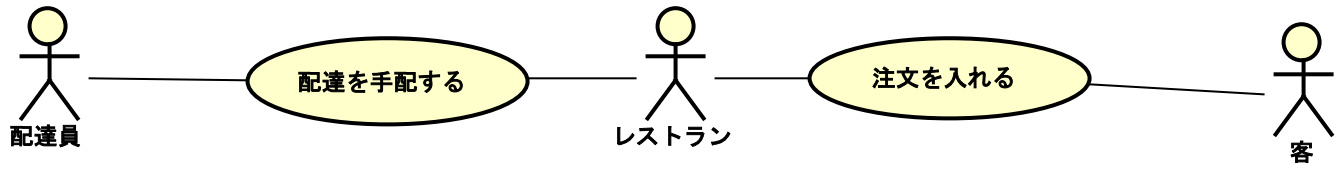


ポイント

- 楕円の集合がシステムなので，システムを表す人型（ステックマン）は存在しない．
- アクターは人ばかりとは限らない．
  - データベース，認証システム，クレジットカード決済システム等，外部システムもアクターになりうる．
- 複雑なシステムは1つの機能を実施するのに，複数のアクターが関与する．

複数関与する機能をもつシステム例

- ウーバーイーツのシステムの一部.
- 金銭授受関係は省略してある.



基本系列	1. レストランは配達可能になったことを入力する. 2. システムは配達先に適した配達員に配達依頼をする. 3. 配達員は配達依頼を受諾する. 4. システムはレストランに配達者がとりにくる旨を通知する. 5. システムはレストランに送金する. 6. システムは配達員に送金する.
例外系列	適した配達員がみつからない場合: システムは配達できる人がいない旨をレストランに通知する.

基本系列	1. 客はレストランを選び注文を要求する. 2. システムはレストランに注文を送る. 3. レストランは注文を受諾する. 4. システムは客に注文が受け付けられた旨を連絡する. 5. 客は料金をシステムに送金する.
例外系列	レストランが注文を拒否した場合, システムは注文が拒否された旨を客に連絡する

ユースケース記述について

- 各ユースケース(楕円で示した機能)が，システム外のアクターと，どんなステップをふんで機能を遂行するかの手順を書く.
- システム内の手順は書かない，書くのはシステム外部とのやり取りのみ.
- 基本，システムもしくはアクターが主語となる文の列となる.
- 通常の手順に加え，代替の手順，例外の手順も書く.

ユースケース記述のフォーム構成

- 概要
  - 当該機能の概要を一～二行くらいで要約して書く.
- 事前条件
  - この機能を実行する際の前提条件.
- 事後条件
  - この機能が実行された後に成り立つ条件.
  - 要は機能の宣言的な意味.
- 基本系列
  - 機能を遂行する正攻法の手順.
  - 要は機能の手続き的な意味.
- 代替系列 (略可能)
  - 別ルートの手順. もしあれば.
- 例外系列
  - 機能遂行が失敗に終わる場合の手順.
  - システムやアクターが回復不能な状態に陥らないような手順が望ましい.

例

項目	内容
ユースケース	精算する
概要	購入代金の精算を行う.
アクター	顧客
事前条件	顧客はログイン状態にある
事後条件	支払いが完了している
基本系列	顧客はシステムに住所を入力する. システムは入力された住所を表示し確認ボタンとやり直しボタンを提示する. 顧客は表示内容が正しい場合，確認ボタンを押す. システムは支払い方法の選択画面を表示する. 顧客は支払い法を選択する. システムはカード決済が選択された場合，カード番号を要求する. 顧客はカード番号を入力する. システムは決済確認のためのボタンを表示する. 顧客はボタンを押し決済を承認する. システムは承認を受け付けたことを表示する.
代替系列	
例外系列	システムは入力されたカード番号がvalidなものでなければ，その旨を表示し精算を中断する.

記述のポイント

- 文の列として書く.
- 能動態の文で書く，受動態や支持的な文はダメ.
  - 「<アクター>が<何か>をする」もしくは,
  - 「システムが<何か>をする」の形式がよい.
- 各文の主語はアクターもしくはシステムでなければならない.
- 主語は省略してはならない.

基本，代替，例外と分ける意味

- プログラムのような形式記述なら，これらを分けて書く必要は無いかもしれない．
- しかし，原則，文のリストとしてかくため，制御構造が複雑にならないためにも，分けて書くほうがよいとされている．
- 加えて，本来，意図する動作である「基本」，バックアップとしての「代替」，障害処理的な「例外」は，それぞれ要求分析的に意味合いが違うため，文法的に合成して書けたとしても，合成すべきではない．

ステートマシン図

- 本来は，システム内のオブジェクトの状態をモデル化する図.
- 前述のユースケース記述を画面遷移として記述するための記法と考えてよい.

■角の丸い四角 = 画面

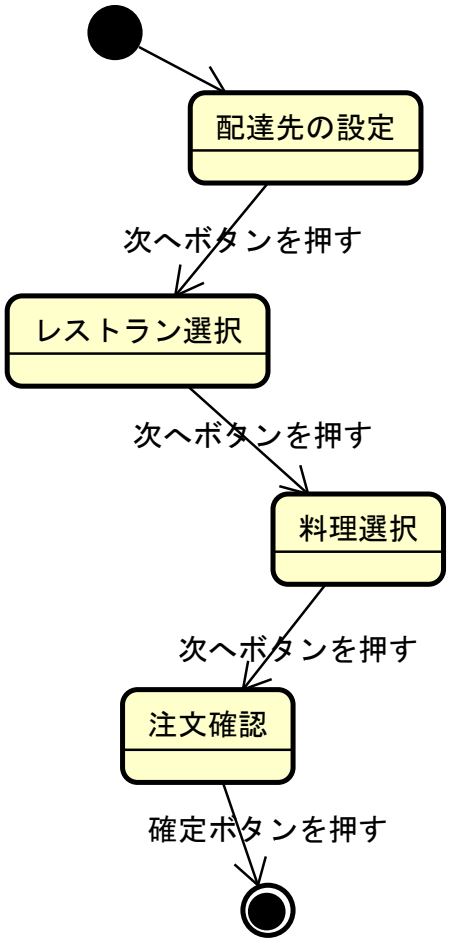
■矢印 = 画面遷移

と考えてよい.（若干，おおざっぱ）

- 丸が処理であるフローチャート等とは考え方が逆.
  - 状態にはある程度の時間幅があるものと考える.

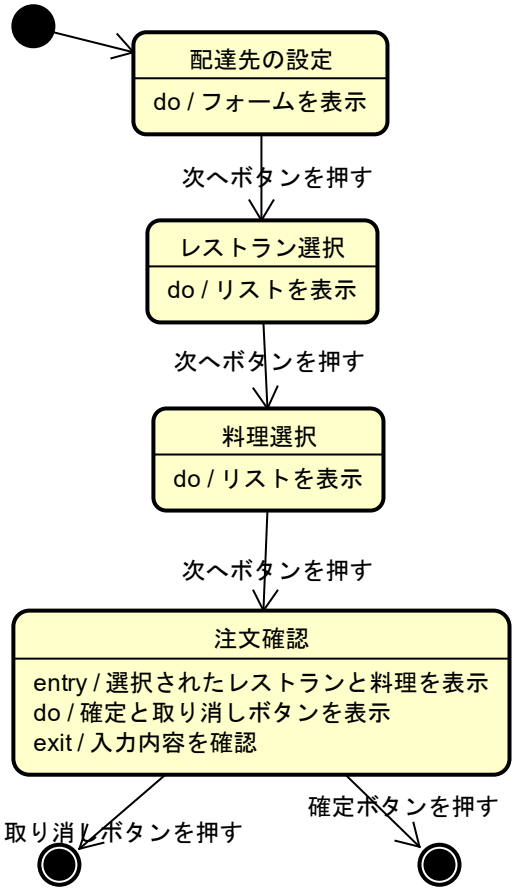
簡単な例 出前の注文

- 状態が画面で，遷移がボタンを押す等のアクションである典型例.
- レストラン，料理の選択は逆の仕様もありうる.
- 配達先は最初に指定しないと，レストランは絞り込めない.
- 状態の名前に「～画面」とつけると意味がしっくりくる.



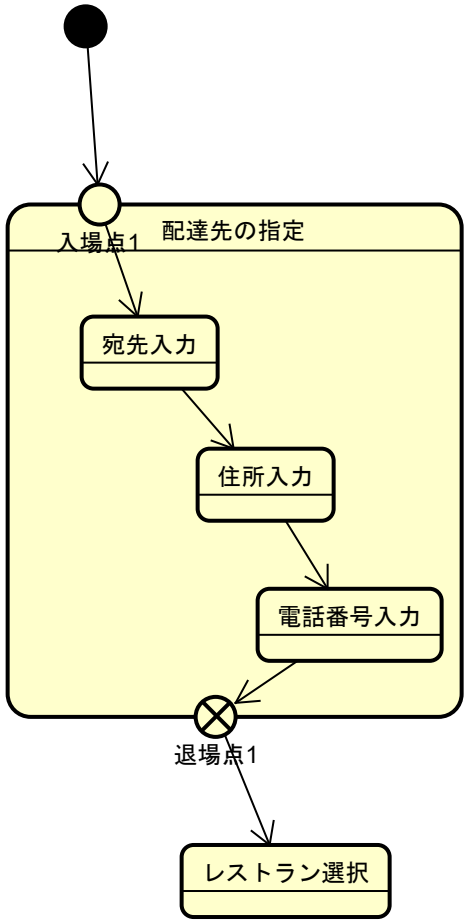
状態中の動作を指定できる

- 状態にかかわる動作を指定できる.
- entry
  - 状態に遷移した際に行う動作
- do
  - 状態中に行う動作
- exit
  - 他の状態等に遷移してしまう時点で行う動作



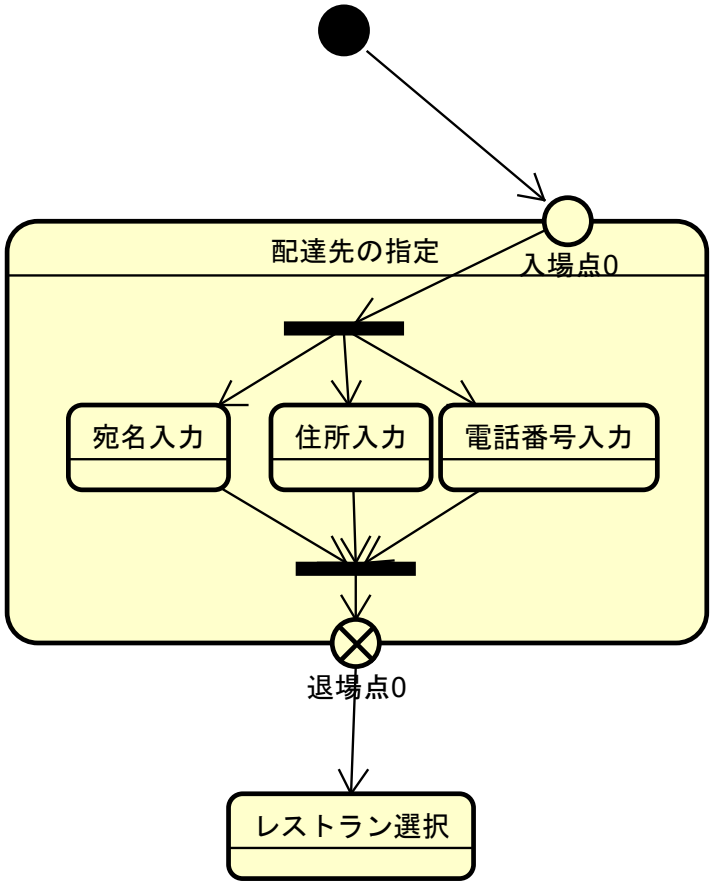
部分状態

- 配達先の指定といっても，名前入力状態，住所入力状態，電話番号入力状態と，より細かい状態があると考えてもよい．
- そのような状態を，状態の中に書いてもよい．
- まぁ，どの状態をグループ化するかは意味次第．



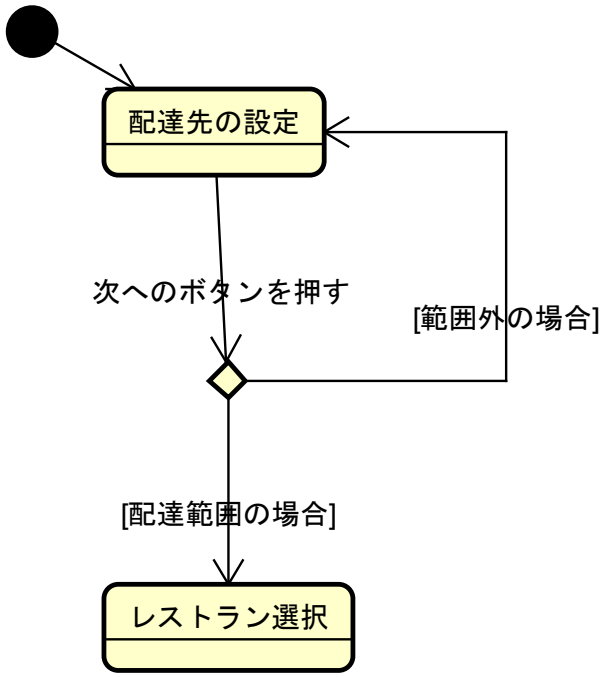
同時並行の状態

- フォーム等に，氏名，住所，電話番号等を任意の順番に埋める場合を考える.
- それぞれに，入力済/未という状態を持つととらえることができる.
- また，入力の順番も特に問わないので，並行して確定してよいということになる.
- そのような場合は右のように書いてもよい.
- まぁ，ここまで詳しく書く必要は無いだろう.



状態の値で条件分岐

- ボタンを押す等の動作ではなく，状態が保持する値によって，遷移先を変えることも可能.
- プログラムの if then とほぼ同じ.



開発プロセス

- UIに限らず，ソフトを開発する際には，以下の手順を踏む．

1. 要求分析 利用者が何を達成したいかの明確化
   - スポンサーが利用者を利用したいなら，スポンサーの達成目標も明確化する．
2. 仕様化 達成を助ける機械（ソフト）の振る舞いを決める，加えて利用者の新しい作業の進め方も決める．
   - 機械のみが要求を満たすのではなく，機械を導入した新しいやり方が要求を満たすのだという点に注意．
3. 設計 その機械の機能，構造，UI等を決める．
4. 実装 機械を作る．コーディング．
5. テスト 機械が達成したいことの助けになるかを確認．

- 基本，上記の順番でやるが，繰り返し小分けに行ったり，適宜，関係者と協議して行ったりする場合もある．

UIはごく一部の要因

- UIはソフトや機械の一部である.
- よって，機能や構造と並行して決めるべきであろう.
- UX的な観点は，十分な要求分析によって仕様に落とし込める.
  - 例えば，ホントは商品取得をしたいんじゃなくて，ただ，見て回りたいだけであるという利用者の願望を見抜く.
  - ホントは高機能な製品の利用を迫られているわけではなく，綺麗な箱からオシャレな製品を取り出したい自分に酔いたいだけなのを見抜く.
- UIはごく一部でしかなく，むしろ，業務分析，ワークフロー分析等が重要.
  - 一体，人々は何のために，何をやっているかの理解.

利用状況の把握法

- 利用者の目標と，その現達成法を知るのは，UI設計にも非常に役立つ.
- 目標達成のためのユーザーの活動をタスクと呼ぶ.
- 大きく分けて二種類の方法がある.

1. 観察 タスクを外から観察.
   - 民俗学や文化人類学で行われる手法の流用.
   - エスノメソドロジーと呼ばれる.
   - IoT等を用いたログ収集と分析もこちらに入る.
2. 面接 利用者に話を聞く.

- どちらも，かなりコスト（労力）がかかるが，予算があるならやっておいたほうがよい.

具体的な手法

- 観察
  - フィールドワーク 対象業務に入り込み観察・記録する.
  - ダイアリー法 観察対象者に日記や記録をつけてもらい，後から話を聞く方式.
  - 今日では，監視カメラ，ログ，通信記録，センサー(IoTデバイス)等でも情報を収集できる.
- 面接
  - 質問用紙（いわゆるアンケート）
  - インタビュー
  - どちらも自由に聞く場合と，予め構造を決めておく方法がある.

仕様の決定に向けて

- 対象（利用者等）が何をやっているかを把握した後，
  - その目標を明確にし，
  - 目標を達成し，高める機械が何かを考えて，

機械の仕様を決めなければならない．

- 発想法をはじめとする各種手法は上記に役立つものもある．

具体的な手法

- ブレインストーミング
  - アイディアを多数収取するための議論法．反論や否定を禁止する等が特徴．
- KJ法
  - カードにアイディアを書いてグループ化し，考えをまとめる方法．日本人が発案した．
- グランデッドセオリー
  - KJ法同様，データ整理の手法．
- ペルソナ
  - 具体的なユーザー（近所の田中さん等）のプロファイルを想定する手法．
- 品質機能展開
  - 使いやすさや信頼性等，直接実現できない品質特性を，具体的な機能や構造に対応付けする手法．
- ゴールモデル
  - 達成したいことを手段に分解してゆき，最終的に誰がどのゴールを達成するかをきめる手法．機械とそれ以外の役割分担が明確になる．

UIプロトタイプ

- 機械製品では試作品を指すが，UIの場合，できを確認するための模型のことを指す．
  - 試作品までいってないものもUIではプロトタイプと呼ぶ．
  - コンセプトカー的なものもプロトタイプと呼んでしまう．
    - 機能(自動車なら走る)が伴わなくても見た目があればOK
- ローファイとハイファイ
  - Low-Fidelity vs High-Fidelity
  - 大雑把なものと，完成品に近いもの
  - ローファイの場合，手書きのメモ程度でOK

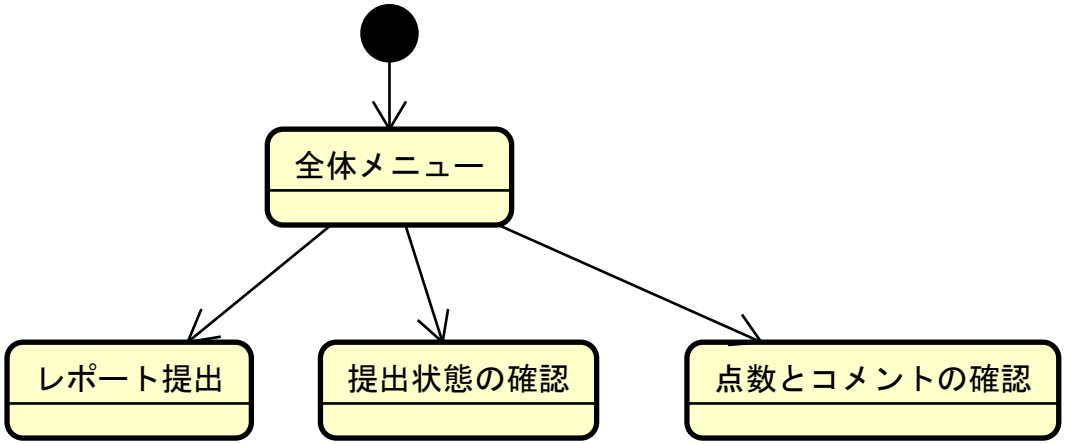
垂直・水平プトロタイプ

- 複数の画面遷移によって構成されるアプリは多い.
- この画面遷移全てのプロトタイプを作るのはシンドい.
- そこで，以下の二種類の略し方がある.

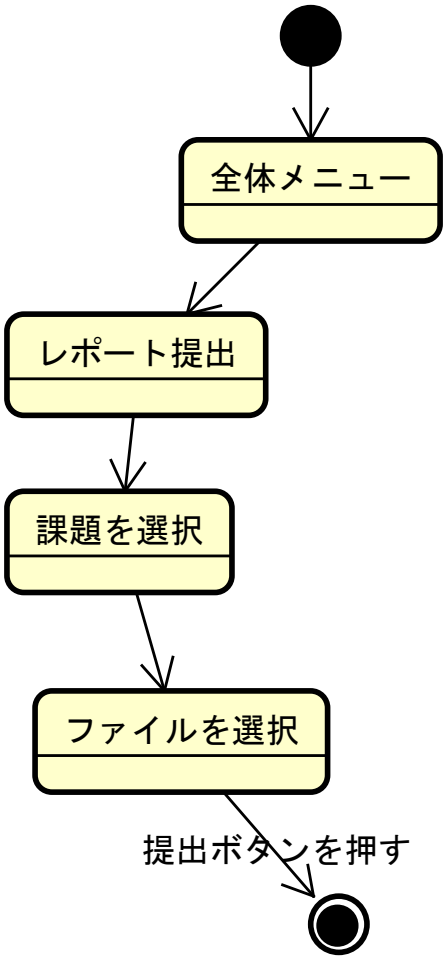
1. 水平プロトタイプ
   - 最初の遷移画面群のみのプロトタイプを作る.
2. 垂直プロトタイプ
   - ある一連のページ遷移列のプロトタイプのみつくる.

- Tプロトタイプ
   - 上記の組み合わせ.

水平で扱う画面群の例



垂直で扱う画面群の例



ペーパープロトタイプ

- パワポをはじめツールを使うのも良いが，紙と鉛筆等で画を描くのが，一番安上がりで，しかも効果的な方法.
- 基本，前回紹介したGUI部品，レイアウト等を参考にして画をかいてゆく.

カードソート

- 画面中の要素のレイアウトを決めるための手法.
- 配置すべき要素を紙のカードに書き，ユーザー等が分類・配置するという超ローテクな方法.
  - カードは付箋紙（ポストイット）でもOK
- クローズド・カードソート
  - 予め考えた分類が正しいかをチェックするのにつかわれる.
  - 分類したい内容（例えば画面に配置する商品名等）が，それぞれどの分類に入るかをカードで視覚化する.
- オープン・カードソート
  - 分類したい内容を，グループ化して，その後に，できたグループに分類を与える.
- ちょっと，KJ法に似てる.

UIの表示内容の分類

- UIの表示内容は大雑把にいって，以下の二種類になる.

1. 手続き型
   - 時間を設定する，ファイルを選ぶ等，何をやるかを選ぶもの．Taskを記述.
2. 目的型
   - 弁当を温めたい，写真を消したい等，やりたい目標を選ぶもの．Goalを記述.

- 無論，上記を組み合わせて一連の作業を利用者が行うものが多い.

取り扱いの説明

- 現代のUIには説明書の機能も付加されているものが多い.
- 説明書は以下の3種類に分類できる.

1. ガイダンス
   - 用語の説明等を含めた, コレは何なのかの説明.
2. ヘルプ
   - Q/Aのような, やりたい事に対する答え.
   - 操作の文脈を理解し, Q/Aを取捨選択するものもある.
3. レクチャ
   - このように使ってください等のいわゆるチュートリアル.
   - 典型的な利用例のシナリオという感じなので, アプリに組み込まれる場合は, 模擬実行的になる.

文書化について

- OSやアプリを利用するに際して，予め説明書を念入りに読む人は今時はほどんどいない．
- それでも，API(予め提供される関数やクラス)の文書は，プログラマは念入りに読まざるを得ない．
- APIのマニュアルの様式は言語等によって決まっている．
  - C言語のmanコマンドによるマニュアル．
  - JavaのJavadoc API
- 一般ユーザー向けには前述のようなアプリに組み込まれた取り扱い説明，開発者向けには，詳細はマニュアルという住み分けが行われている．

```
[kaiya@flute03 ~]$ man 2 read
```

kaiya@flute03:~

```
READ(2)                    Linux Programmer's Manual                    READ(2)

名前
       read - ファイル・ディスクリプターから読み込む

書式
       #include <unistd.h>

       ssize_t read(int fd, void *buf, size_t count);

説明
       read() はファイル・ディスクリプター (file descriptor) fd から最大 count
       バイトを buf で始まるバッファーへ読み込もうとする。

       count が 0 ならば、 read() は 0 を返し、他に何も起きない 。 count が
       SSIZE_MAX より大きければ、結果は特定できない。

返り値
       成功した場合、読み込んだバイト数を返す (0 はファイルの終りを意味する)。
       ファイル位置はこの数だけ進められる。この数が要求した数より小さかった とし
       てもエラーではない; 例えば今すぐには実際にそれだけの数しかない場合 (
       ファイルの最後に近いのかもしれないし、パイプ (pipe) や端 末 (terminal)
       か ら読み込んでいるかもしれない) や read() がシグナル (signal) によって
       割り込まれた場合にこれは起こりえる。エラーの場合は、-1 が返され、 errno
       が 適切に設定される。この場合はファイル位置が変更されるかどうかは不定で
       ある。

エラー
       EAGAIN ファイル・ディスクリプター fd がソケット以外のファイルを参照して
              いて、非停止 (non-blocking) モード (O_NONBLOCK) に設定されており
              、読み込みを行うと停止する状況にある。

       EAGAIN または EWOULDBLOCK
              ファイル・ディスクリプター fd がソケットを参照していて、非 停 止
:
```

マニュアルの内容

- 書式
  - インクルードすべきヘッダー
  - 関数名
  - 返り値の型
  - 引数の数と型
- 説明
  - 関数の意味が文章で説明してある.
- 返り値
  - 返り値の意味が文章で説明してある.
- エラー
  - エラー値の種類と意味が説明されている.

https://docs.oracle.com/javase/jp/8/docs/api/index.html?java/lang/String.html

概要 パッケージ クラス 使用 階層ツリー 非推奨 索引 ヘルプ

Java(tm) Platform Standard Edition 8

前のクラス 次のクラス フレーム フレームなし

サマリー: ネスト | フィールド | コンストラクタ | メソッド 詳細: フィールド | コンストラクタ | メソッド

compact1、compact2、compact3

java.lang

クラスString

java.lang.Object

java.lang.String

すべての実装されたインタフェース:

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

Stringクラスは文字列を表します。Javaプログラム内の"abc"などのリテラル文字列はすべて、このクラスのインスタンスとして実行されます。

文字列は定数です。この値を作成したあとに変更はできません。文字列バッファは可変文字列をサポートします。文字列オブジェクトは不変であるため、共用することができます。たとえば、

```
String str = "abc";
```

は、次と同じです。

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

https://docs.oracle.com/javase/jp/8/docs/api/index.html?java/lang/String.html

matches

public boolean matches(String regex)

この文字列が、指定された正規表現と一致するかどうかを判定します。

このフォームのメソッド呼び出し*str*.matches(*regex*)では、次の式と正確に同じ結果が得られます。

```
Pattern.matches(regex, str)
```

パラメータ:

regex - この文字列との一致を判定する正規表現

戻り値:

この文字列が指定された正規表現と一致する場合にだけ、trueが返される

例外:

PatternSyntaxException - 正規表現の構文が無効な場合

導入されたバージョン:

1.4

関連項目:

Pattern

contains

public boolean contains(CharSequence s)

この文字列が指定されたchar値のシーケンスを含む場合に限りtrueを返します。

UI標準

- ISO9241-11
  - 1998 ユーザビリティの国際規格
- ISO9126
  - 2001 ユーザビリティを含む品質特性全般の規格
- ISO25010 SQuaRE
  - 2011 同上

ISO9241-11

- ある製品が，指定された利用者によって，指定された利用の状況下で，指定された目的を達成するために用いられる際の，
- 有効さ
  - 目的達成のための正確さと完全性.
  - 例 近所の飲み屋を検索した場合，適切な結果が出た.
- 効率
  - 正確さと完全性に費やした資源，主に時間.
  - 例 検索は瞬時に出て，バッテリー消費も少ない.
- 満足度
  - 不快感の無さ，肯定的な態度.
  - 例 ランチを探すときにも使いたいな，と思った.

の度合い.

ユーザビリティの尺度

- 有効さの尺度
  - 達成された目標の割合，目標を達成できたユーザーの割合，達成できた仕事の割合，結果の正確さ
- 効率の尺度
  - かかった時間，単位時間に処理できた仕事数，金銭やエネルギー（バッテリー）の消費．
- 満足度の尺度
  - 自主的使用の頻度，再利用の頻度．

ソフトウェア品質特性標準 – ISO 9126

- **ソフトウェア品質特性**に関する標準
- http://www.cam.hi-ho.ne.jp/adamosute/software-quality/iso9126.htm より

品質特性(quality charastristics)	品質副特性(quality subcharactristics)	主な内容
機能性(fuctionality)	合目的性(suitablity) 正確性(accuracy) 相互運用性(interoperability) 標準適合性(compliance) セキュリティ(security)	目的から求められる必要な機能の実装の度合い
信頼性(reliability)	成熟性(maturity) 障害許容性(fault tolerance) 回復性(recoverability)	機能が正常動作し続ける度合い
使用性	理解性 習得性 運用性	分かりやすさ、使いやすさの度合い
効率性	時間効率性 資源効率性	目的達成のために使用する資源の度合い
保守性	解析性 変更性 安定性 試験性	保守(改訂)作業に必要な努力の度合い
移植性	環境適用性 設置性 規格適合性 置換性	別環境へ移した際そのまま動作する度合い

ユーザビリティに注目

- 製品自体の品質
  - 理解性，修得性，運用性，魅力性
  - 分かりやすかったり，覚えやすかったりしたからといって，運用効率がよいとは限らないので，上記は独立したもの．
- その製品を使った過程や結果の品質
  - 有効性 正確かつ完全に目標が達成されること．
  - 生産性 効率的に達成できること．
  - 安全性
  - 満足性

ISO 25010 SQuaRE

- Software product Quality Requirements and Evaluation から部分文字をとってる・・・ツライい
- ISO9126を，より整理した感じ，大枠はかわらない.

- ユーザビリティに関する製品の品質
  - 適切度認識性，修得性，運用操作性，ユーザーエラー防止性，ユーザーインタフェース快美性，アクセシビリティ
- 利用時の品質
  - 有効性，効率性，リスク回避性，利用状況網羅性（柔軟性）

GUI設計のガイドライン

- OS各社が，OS下でのアプリの統一性を図るために，ガイドラインを定めている．

- ガイドラインに沿わないアプリは公式配布サイトに掲載させない等の手段に訴える場合もある．
- 最近はプラットフォーマー権威主義がはびこってるので，ガイドラインに沿わないとBANされる・・・
  - プラットフォーマー ≒ OSの開発会社

例: iOSの6原則

1. 外観の整合性
2. 一貫性
3. メタファ
   - 隠喩 暗に例えること.
4. フィードバック
5. 直接操作
6. ユーザーによる制御

iOSのガイドライン

1. 適切なアフォーダンス (affordance, afford)
   - 形態(UI)が使い方を示唆する性質
   - フロッピーディスクを知らない世代には  はアフォードしてないよなぁ・・・
2. 操作の一貫性
3. よい対応付け
4. 適切なフィードバック
5. 簡単なエラー処理
6. ユーザ側に主体的な制御権
7. ユーザの個人差への対応

UI評価

- まずは，前述の標準規格やガイドラインに沿っているかに基づきチェックを開発者が行う.
- また，利用者からアンケートをとったり，実験を個なったりする評価もある.
- 実験に関しては学習効果を考慮するようにする.
- アンケートに関しては誤解やバイアスを避ける必要がある.
  - シュナイダーマンという人が考えたQUISという標準的な設問集がある.

キーストロークレベルモデル

- KLM keystroke-level model
- 操作に必要な時間を予測するための模型
- ある操作を，人の基本的な動作に分解して，その所要時間を積算することで時間を予測.
- 以下が基本的な動作とその標準時間

■K キーボード1個を押して離す 0.28s

■T(n) n文字入力，Kのn倍

■P マウスで画面上の物を指示する 1.1s

■B マウスを押す，および，離す それぞれ 0.1s

■H キーからマウスに手を移動 (および逆) 0.4s

■M 機械的な判断や知覚 1.2s

■W(t) システムの応答時間 t秒 (システム依存)

実際の計算例

- GUIでファイルを削除する操作
- ファイルアイコンの場所までマウスを移動する P
- マウスのボタンを押す B
- ゴミ箱アイコンまでファイルをドラッグ P
- マウスを離す B
- 上記合計は,
  P+B+P+B=P*2+B*2=1.1*2+0.1*2=2.4 秒

- 同じ結果の異なる操作を上記で比較することで,
  どの操作が効率的か知ることができる.

フィッツの法則

- マウス等のポインティングデバイスで，アイコン等の目標を指示するのに要する時間を計算するモデル，というか式.

■ポインタの現在位置から目標までの距離 D

■目標の幅を W

■指示するのにかかる時間 T

$$T = a + b\ log_2(\frac{D}{W} + 1)$$

- なお，a, b はユーザーやデバイスによって決まる定数.
- 距離が遠いほど，目標が小さいほど時間がかかるという直感にあっている.

演習1

- 自分が関心のあるアプリの画面遷移図を，astahを用いて，ステートマシン図で記述せよ.
- ステートマシン図のベースの定義の部分に，どんなアプリかを文章で説明してください. (以下は例)
- 画面のペーパープロトタイプは不要だが，書きたい人は書いてください.
  - astahに図(JPG等)も張り付けられるはずです.
- ✍切 10月中

以上

100文字感想の提出も
忘れずに