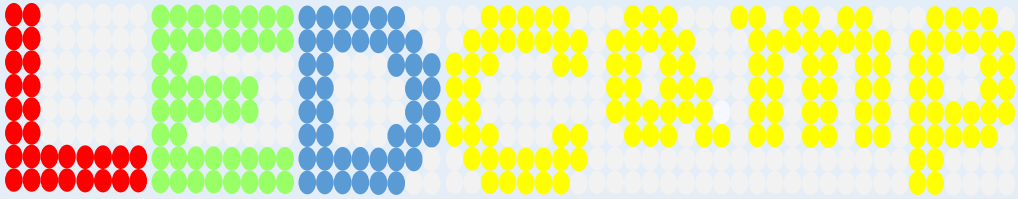


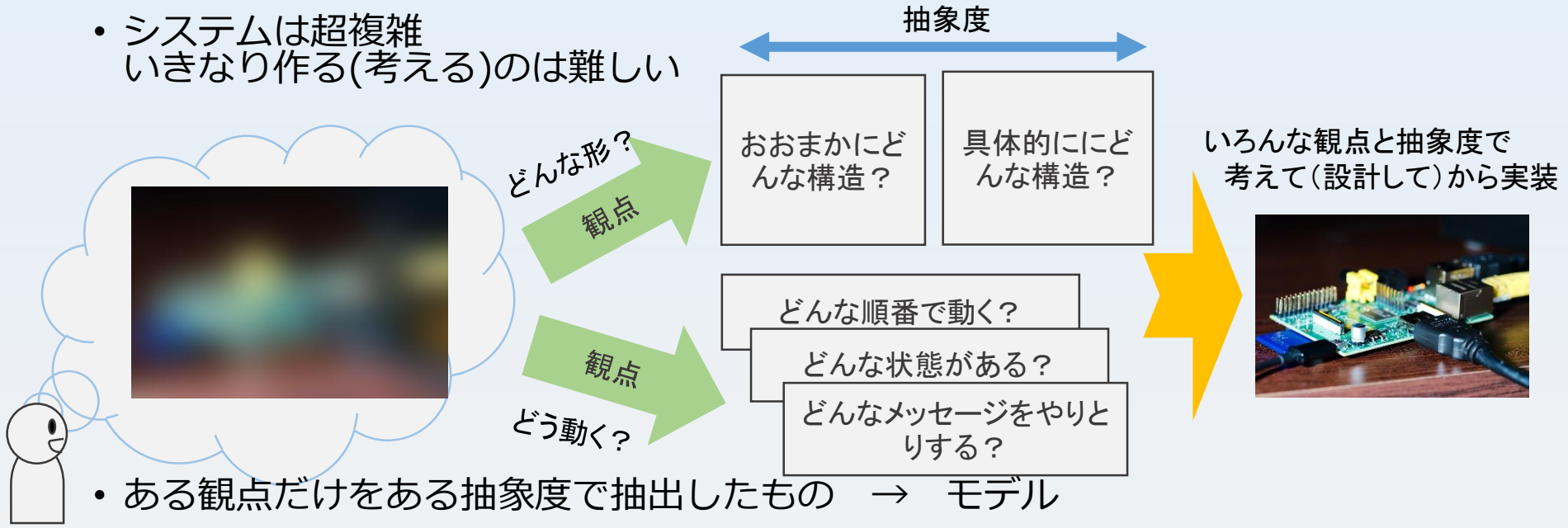
UMLの使い方

Unified Modeling Language

細合　晋太郎

そもそもモデリングとは？

- システムは超複雑
  いきなり作る(考える)のは難しい



- ある観点だけをある抽象度で抽出したもの　→　モデル
- 世にあるほとんどの～図はモデルと言ってもいい
  - Ex) 上から見た構造 → 上面図
  - Ex) 粘土で形だけ → クレイモデル
  - Ex) 音階とタイミングで音楽を分解 → 楽譜

また，設計者・実装者間で意思統一を図るためにもモデリングは重要

構造モデルと振舞いモデル

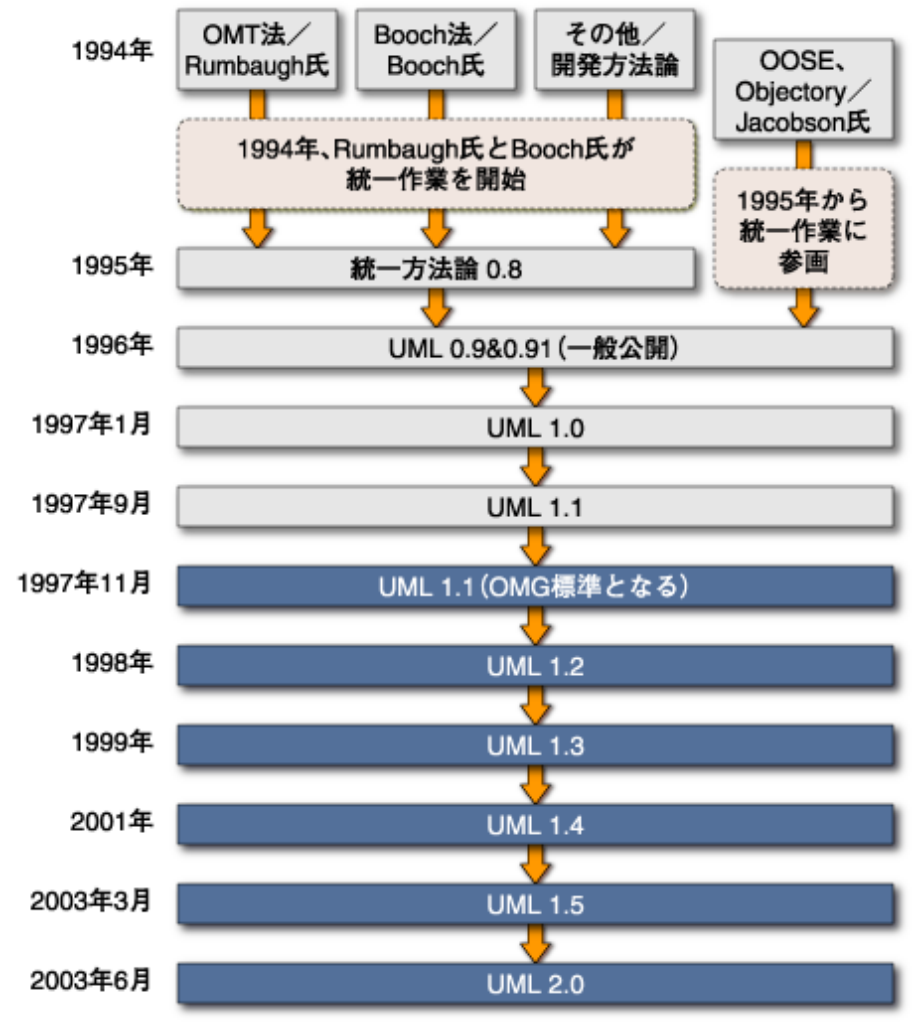
- 観点は大きく分けると構造と振舞いに大別できる。
- どんな形をしているのか？どんな動きをするのか？

- UMLでは、構造と振舞いのモデルが様々な抽象度と観点で定義されている。

UMLモデルは厳密に定義されている。

- UMLの仕様書（Super Structure）
- http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

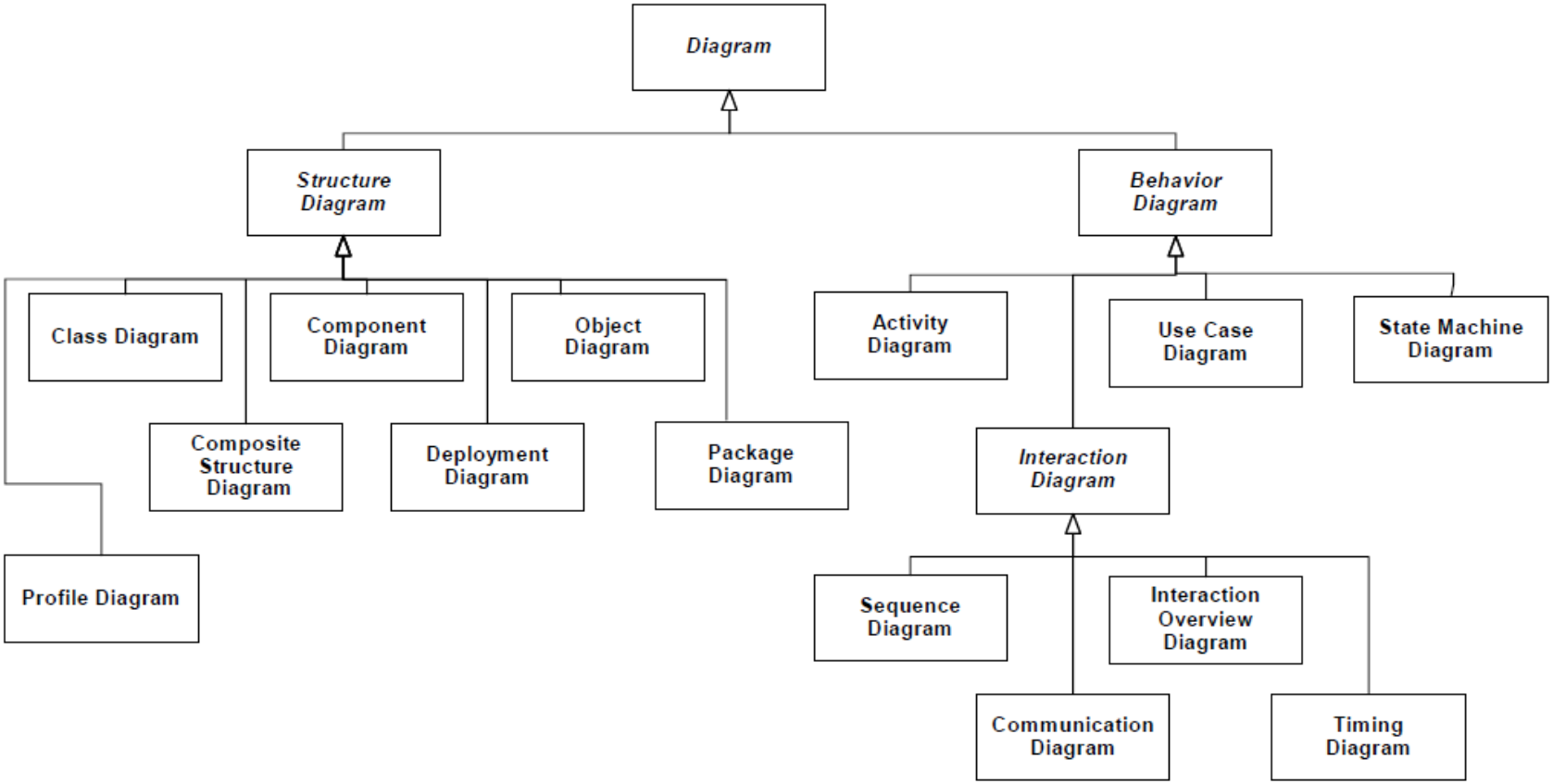
モデリング言語の統一



http://thinkit.co.jp/free/compare/12/1/より転載

2014/08/03 現在
http://www.omg.org/spec/UML/

UML Diagrams (version 2.4.1)



http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/ pp710

構造モデル

- Deployment Diagram：配置図
- Component Diagram：コンポーネント図
- Composite Structure Diagram：複合構造図
- Package Diagram：パッケージ図
- Class Diagram：クラス図
- Object Diagram：オブジェクト図

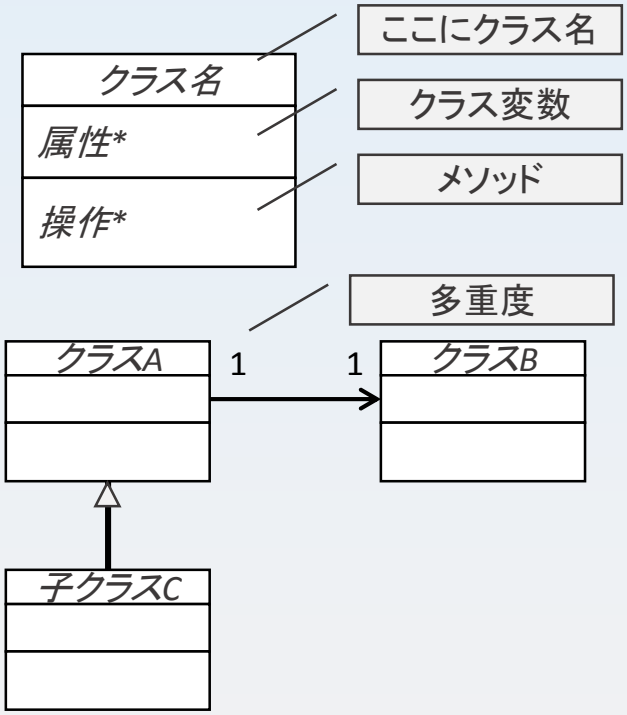
- Profile Diagram：プロファイル図　これは別観点

振舞いモデル

- Use Case Diagram：ユースケース図　外界との関係、要求
- Activity Diagram：アクティビティ図　システムフロー
- State Machine Diagram：ステートマシン図　状態
- Interaction Diagram：相互作用図
  - Sequence Diagram：シーケンス図　オブジェクト間フロー
  - Communication Diagram：コミュニケーション図　オブジェクト相互関係
  - Interaction Overview Diagram：相互作用概念図　図間のオーバービュー
  - Timing Diagram：タイミング図　動作タイミング

振舞いモデルは、見たい観点がそれぞれ異なる。

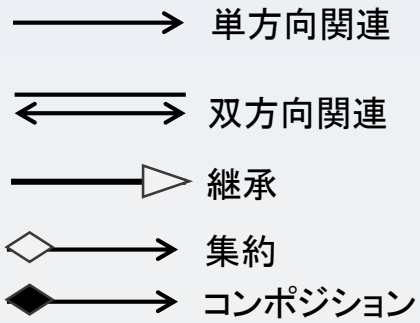
クラス図



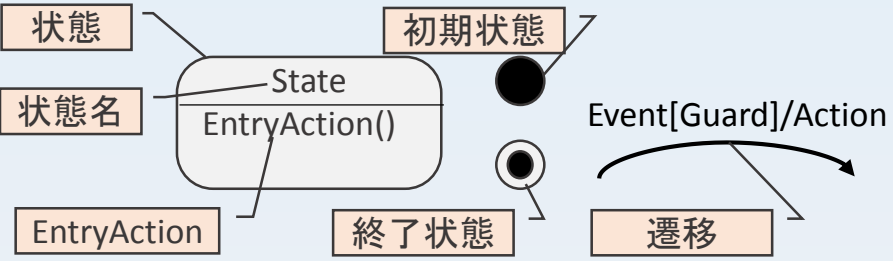
システムのクラス構造を書くためのモデル

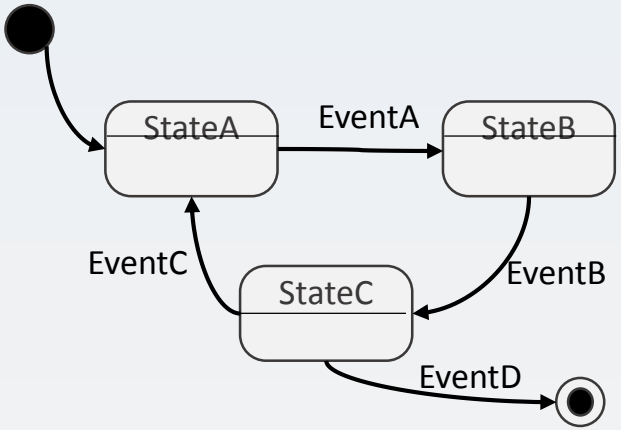
箱でクラスの構造を、線でクラス間の関係を表現する

線種



ステートマシン図





- 状態と遷移を可視化するモデル
- ある状態の時に、その状態からの遷移に指定されているイベントが入ると次状態に遷移する
- Guard
  - イベントが入った際でもGuardに指定された条件が満たされていないと遷移しない
- Action
  - 遷移が発生した際にここで指定したActionが実行される
- EntryEvent
  - その状態に入った際にここで指定したActionが実行される

すべてのUML図を使う必要はない

- UMLの各図は，観点と抽象度のフレームを与えてくれるもの.
  - 必要な図を取捨して選択すればよい

- 図の用途も様々ある
  - 設計分析
  - ステークホルダー間の意思統一
  - エビデンス
  - ．．．

- では，どのようなモデルを選択すればよいか？
  →UMLの開発プロセスを利用してみよう

UMLのプロセス

- 多くのプロセスでは，トップダウンにモデルを記述していく．
- ICONIXプロセスでは．以下の順でモデルを詳細化していく
  - ユースケース図・ユースケース記述
  - ロバストネス図
  - ドメインモデル
  - クラス図
  - シーケンス図
- ただし，何にでも適用できる唯一のプロセスはない
- まずは，既存のプロセスに当てはめてみて，徐々に対象に合わせたモデルを選択し，プロセスを改善していく．

モデリング全体の注意点

- 記法・ルールを守る
  - UMLに厳密に沿う必要はないが，読む可能性のある人の間でルールの統一ができていること.
- 名前付けは的確に
  - ステークホルダーが容易に理解できるものを心がける
- モデル図間，モデル - コード間の整合性
  - 上流から徐々にブレークダウンしていっても，その間の整合性が欠けると破綻してしまう．修正した場合などは，整合性を確認すること
- モデル図のリファクタリング
  - モデル要素やテキストの位置，線の引き方でもモデルの見やすさは変わってくる．綺麗なモデルとなるように，心がける

クラス図のモデリング

- 一つのクラスに一つの責務
  - 複数の責務を持つクラスは分割する.
  - 複数の責務を保つ場合，多くの場合処理が複雑になる，影響範囲がわかりづらくなる等，問題が多くなる.
- 継承関係・依存関係はできるだけシンプルに
  - 循環参照を持たない
  - 多重継承を持たない
  - 所有関係を考慮する（コンポジット？集約？）
- 今回のみの制約
  - 継承不可：自動生成が対応していません．手動実装する場合は可
  - 双方向の関連は持たない
    （今回は自動生成の都合，禁止or手動で実装となります．）

ステートマシン図のモデリング

- 基本的に，イベント，ガード条件，entryアクションで記述していく．（遷移アクション，doアクション，exitアクションはとりあえず使わない．必要な時のみ利用）
- 一つの初期状態を持たせる
- 非決定な遷移を含まない（同じイベント・ガードで複数の遷移を持たない．同じイベントが来た際に，次の状態が非決定とならないように．）
- フローチャートにしない．ステートマシンは状態を捉える図なので，ただ処理手順を書き並べるような図にならないように