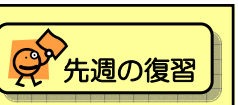
Keio University SFC

『企業と市場のシミュレーション』

第6回：オブジェクト指向の導入

いば　たかし
井庭 崇
慶應義塾大学総合政策学部
iba@sfc.keio.ac.jp
http://www.sfc.keio.ac.jp/~iba/lecture/

新しい知識の源としてのシミュレーション



Herbert A. Simon

- 「いったいシミュレーションは、いかにしてわれわれに未知の事柄を教えることができるのだろうか」

① すでにわかっている前提から、結論を導き出す。

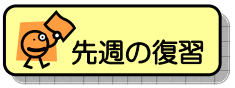
② 内部の仕組みについて理解を深める。

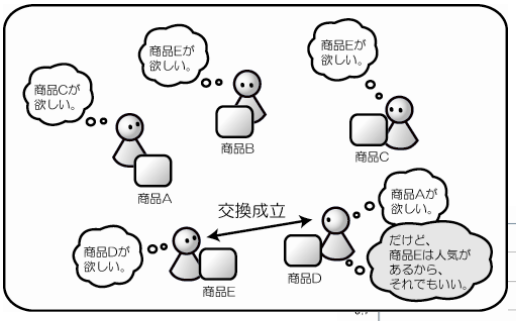
ハーバート・A・サイモン,『システムの科学』, 第3版, パーソナルメディア, 1999

フィードバックコメント

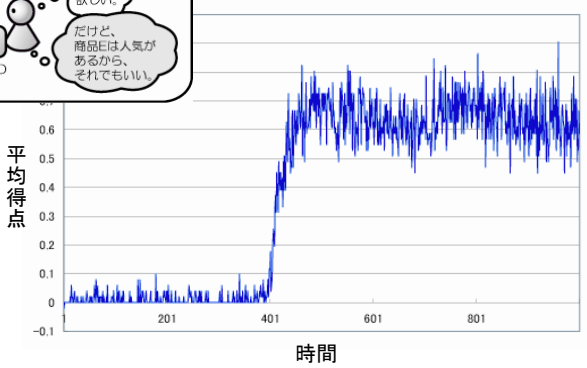
「シミュレーションについてですが、「①すでにわかっている前提から結論を導き出す」という意味合いに多少疑問を感じました。自分の受け取り方がひねくれてるのかもしれませんが、この表現の仕方だと何か確実な前提から結論を導き出すという印象しか受けなく、色々な方法を試す意味でのシミュレーションとはまた違った印象を受けました。私の中のシミュレーションは色々な前提を元に、様々な結論をだし、それをあくまで考察の手助けとするというものなので、今回の授業を聞きながら、疑問を感じるばかりでした。」

貨幣の自成と自壊シミュレーション





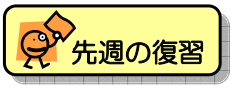
他の人が欲しがっていた財も受け取るようにすると、あるとき突然、交換のために共有される財（＝貨幣）が創発する

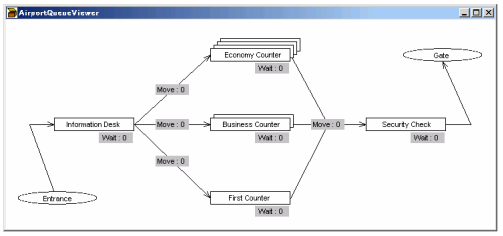


フィードバックコメント

「しかしこのシミュレーションを楽しみながらも疑問に思ったことは、シミュレーションとは「すでにわかっている前提から結論を導きだす」ことに意味があるとおっしゃっていたが、貨幣商品説は貨幣生成に関する一つの仮説であるということだ。つまり貨幣の生成仮説には他にケインズの貨幣法制説や岩井克人さんが自著で提示している「予想の無限の連鎖」などもある。つまりこの仮説が必ずしも正しいかどうかわからないという前提からシミュレーションして導き出した結果ではないのかと感じた。」

空港の待ち行列シミュレーション





- 1. Source（指数分布にもとづく確率で、モデル内に乗客を生成）
- Server（サービスを行う。手が空いていたらQueueManagerに対して列の先頭の乗客を要求してサービスをはじめる。サービスが終わったら次の行き先（QueueManager）を乗客に教える。
- 2. InformationDesk
- 3. CheckInCounter
- 4. SecurityCheck
- 5. QueueManager（行列を管理。乗客が到着したら、列の最後に加える）
- 6. Sink（モデル内から乗客を削除）
- 7. Passenger（乗客。列に並んだり、サービスを受けたり、移動したりする）

フィードバックコメント

「様々な値をいれて実行してみたが、どれがいいかという客観的数値みたいなものがないので、いざこの方法を取り入れてみようという決定の段階においては何を基準にするのか、疑問に思った。」

フィードバックコメント

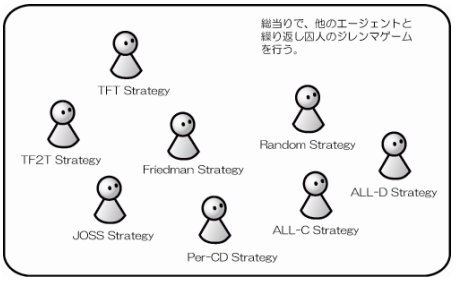
「シミュレーションをやりながら常に感じることは、ミクロレベルのシミュレーションしか実現できないのかということである。それによって今後シミュレートするモデルを考える際に、ある事象をミクロレベルまで分解する必要があるのかなと悩んでしまう。」

今日の話に入る前に、
もう1モデル紹介。

繰り返し囚人のジレンマの戦略模倣シミュレーション

囚人のジレンマ

- 「囚人のジレンマ」は、利己的な主体間で利害が対立する状況において、どのように協調が形成されるのか、を調べるための枠組みとして用いられている。



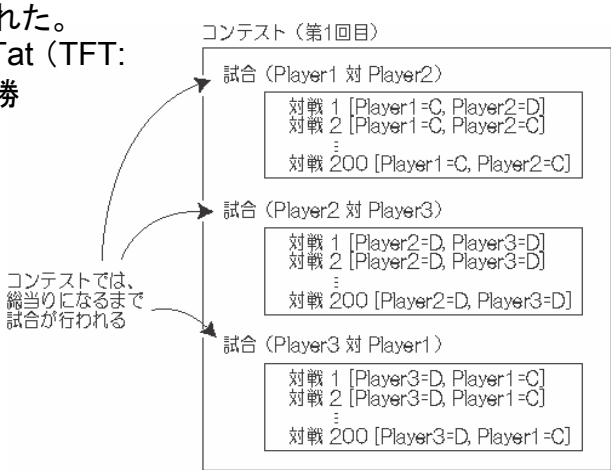
【得点のつけ方】
●両者が協調(C)ならば3点ずつ
●両者が裏切(D)ならば1点ずつ
●一方が協調(C)で他方が裏切(D)ならば、協調の人は0点、裏切の人は5点。

繰り返し囚人のジレンマの戦略模倣シミュレーション

繰り返し囚人のジレンマ

1970年代に、R. Axelrodによって、「繰り返し囚人のジレンマゲーム大会」が行われた。

結果は、Tit For Tat（TFT: しっぺ返し）が優勝



繰り返し囚人のジレンマの戦略模倣シミュレーション

繰り返し囚人のジレンマ・コンテストシミュレーション

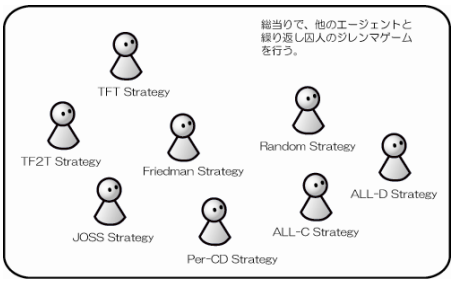
[井庭 2003]

戦略	総得点
FRIEDMAN	8872
FRIEDMAN	8824
TFT	8128
TFT	8041
PERCD	8011
PERCD	8001
TF2T	7846
TF2T	7799
ALLD	7716
ALLD	7684
RANDOM	7481
RANDOM	7468
PERCCD	7384
PERCCD	7348
ALLC	7296
ALLC	7275
JOSS	7197
JOSS	7281

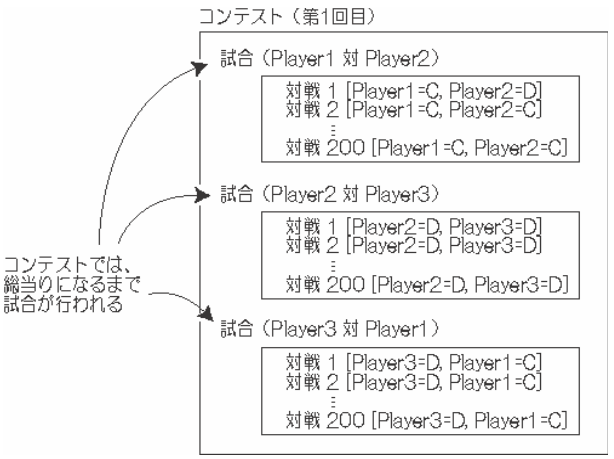
繰り返し囚人のジレンマの戦略模倣シミュレーション

[井庭 2003]

- 試合得点やコンテスト総得点の結果をもとに、自分より強い人の戦略を真似る、というモデル。



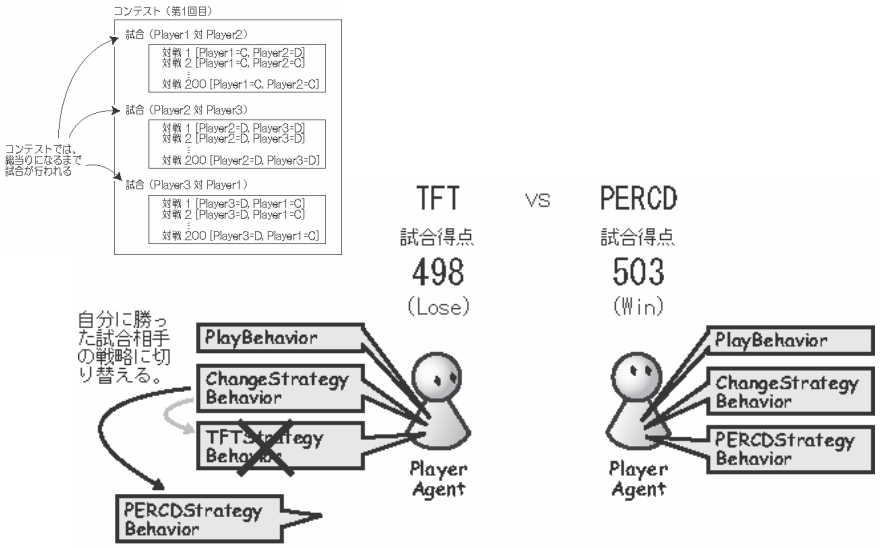
繰り返し囚人のジレンマの戦略模倣シミュレーション



ここでは、次の2つのケースを実験
①試合得点にもとづく戦略変更
②コンテスト総得点にもとづく戦略変更

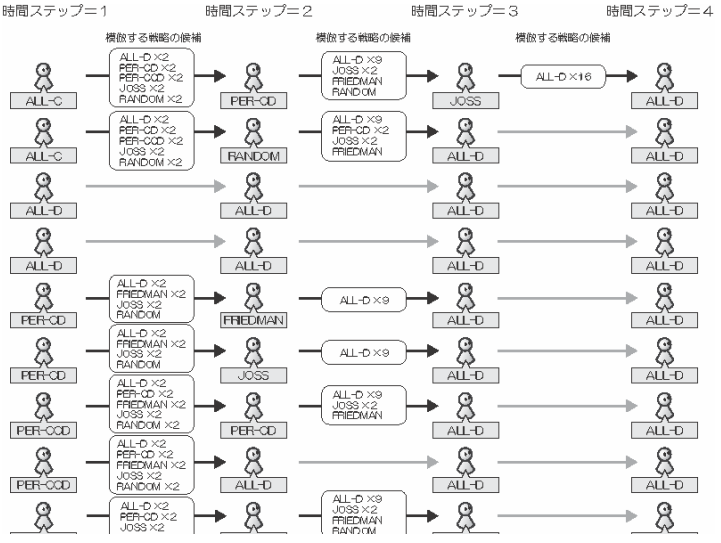
繰り返し囚人のジレンマの戦略模倣シミュレーション

実験①:試合得点にもとづく戦略模倣

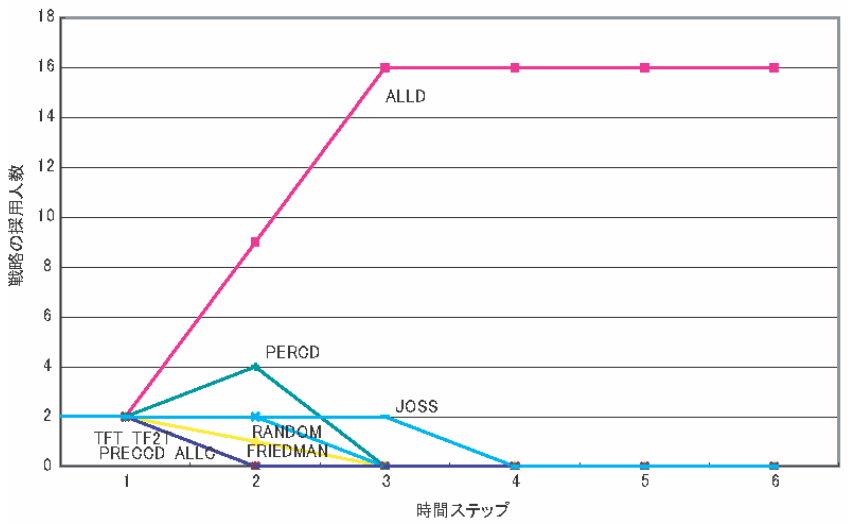


繰り返し囚人のジレンマの戦略模倣シミュレーション

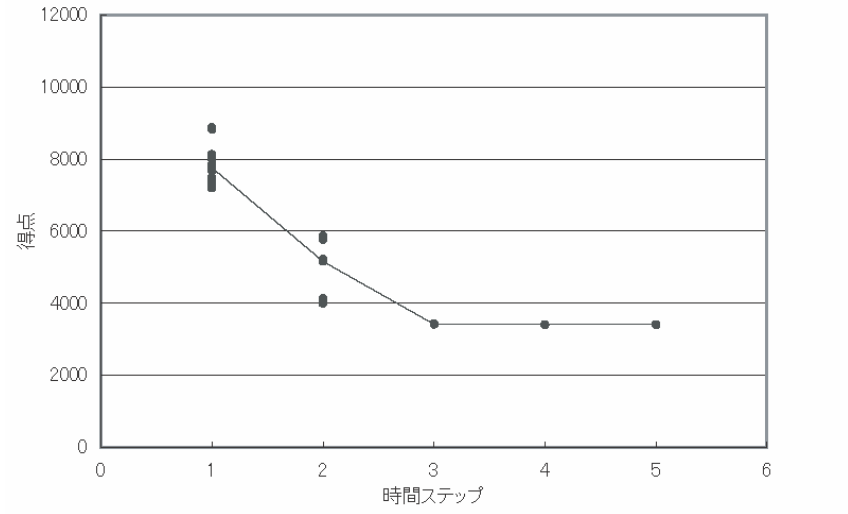
実験①:試合得点にもとづく戦略模倣



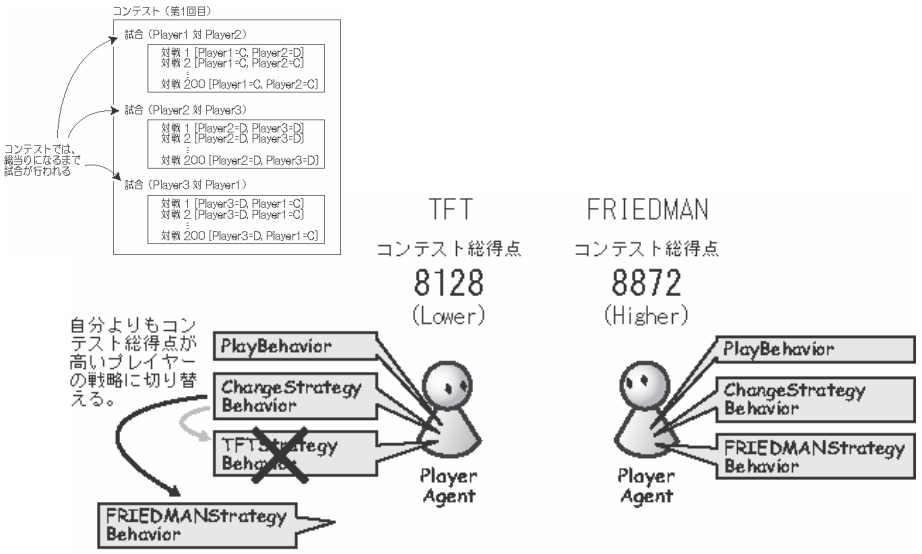
繰り返し囚人のジレンマの戦略模倣シミュレーション



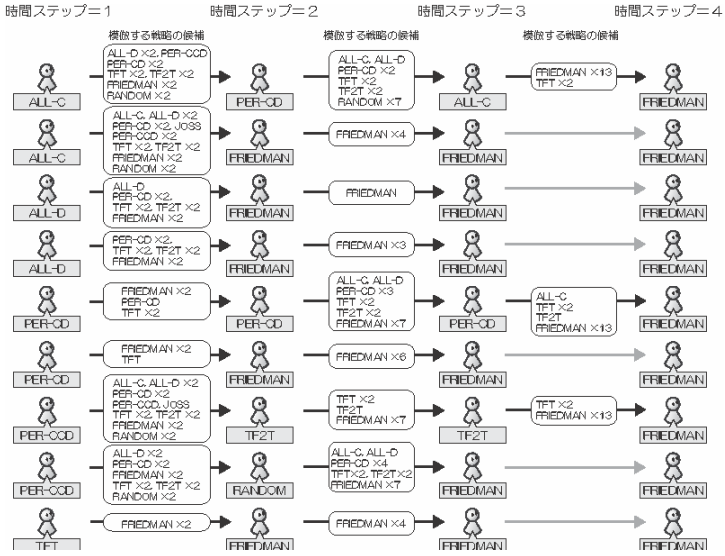
繰り返し囚人のジレンマの戦略模倣シミュレーション



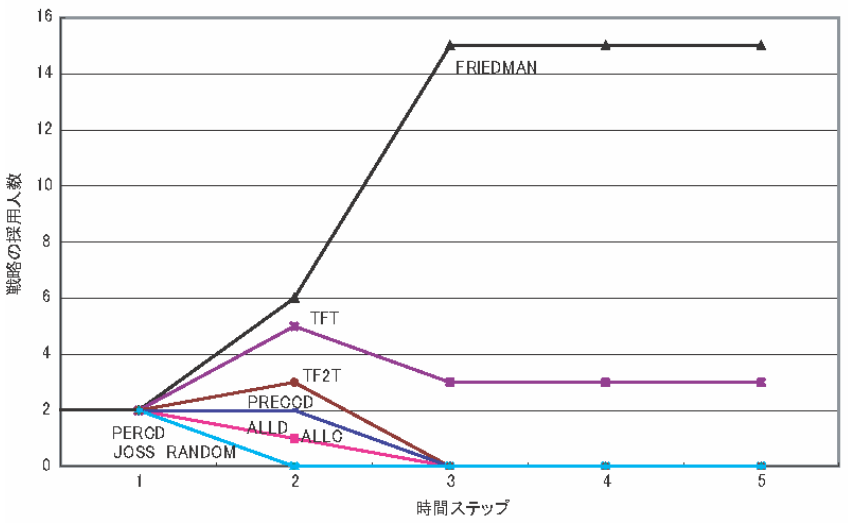
繰り返し囚人のジレンマの戦略模倣シミュレーション



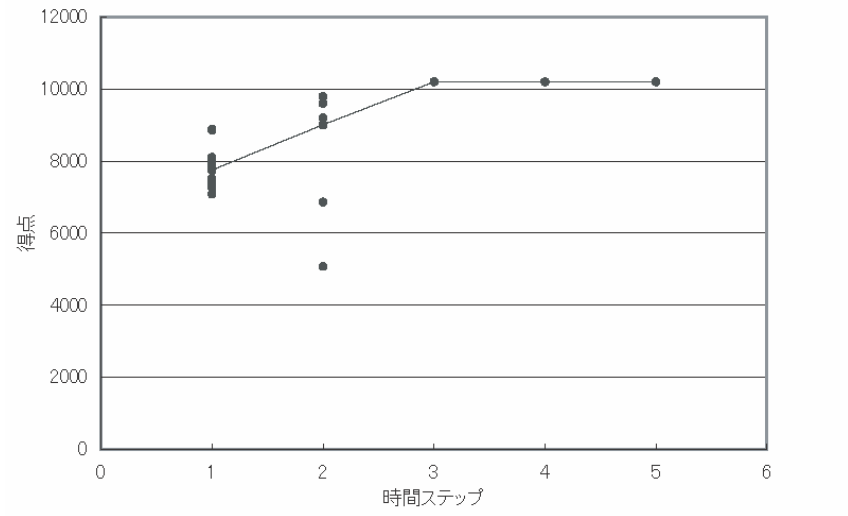
繰り返し囚人のジレンマの戦略模倣シミュレーション



繰り返し囚人のジレンマの戦略模倣シミュレーション



繰り返し囚人のジレンマの戦略模倣シミュレーション



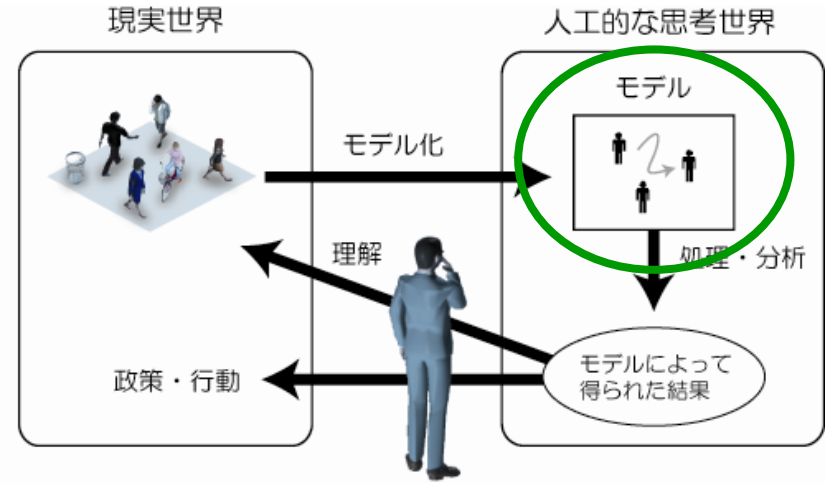
Keio University SFC

『企業と市場のシミュレーション』

第6回：オブジェクト指向の導入

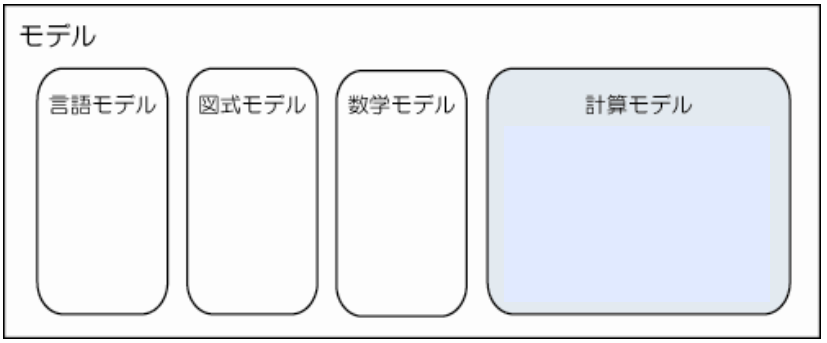
いば　たかし
井庭 崇
慶應義塾大学総合政策学部
iba@sfc.keio.ac.jp
http://www.sfc.keio.ac.jp/~iba/lecture/

モデルをどのように記述するか？



モデルの種類

- 社会科学に、「計算モデル」(computational model)の表現形式を導入することを試みたい。



社会科学の理論の形式化において、数学的モデルより計算的モデルが適している理由

N.Gilbert and K.G.Troitzsch (1998)

- 現実との対応関係の把握が容易
- 並列的なプロセスや、順序の決まっていないプロセスの扱いが容易
- モジュール性をもたせることが容易
- 異質で多様な主体を組み込んだモデルの構築が容易

N・ギルバート, K・G・トロイチュ, 『社会シミュレーションの技法』, 日本評論社, 2003

D.E. Knuth (1985)

- 数学には2つの思考型が欠如
  ①「複雑度」=「操作の節約」の概念
  ②過程の状態に関する動的な概念
- コンピュータサイエンスでは
  ①同時に実行される諸過程間の相互作用を研究するときにも、状態の概念が重要
  ②異なる多様な場合を扱おうという傾向があり、本質的に均質でない諸概念に柔軟に対処できる

Donald E. Knuth, Algorithmic Thinking and Mathematical Thinking, American Mathematical Monthly, March 1985, pp.170–181 有澤誠(編), 『クヌース先生のプログラム論』, 共立出版, 1991 所収「算法的思考と数学的思考」

計算的モデル（Computational Model）

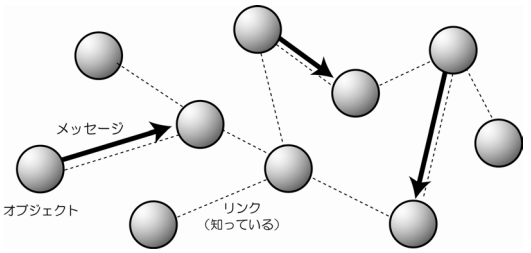
- 近年の計算的モデルの発展は、「命令から宣言へ、手続きからオブジェクトへ、逐次集中から並列分散へ」という方向性にある。この流れにあるパラダイムの一つが「オブジェクト指向」。
  - **「命令から宣言へ」**
    - 「これはそれとこのような関係にある」というような、計算の意味を宣言的に記述するスタイルへ。
  - **「手続きからオブジェクトへ」**
    - 計算手順とデータをひとまとまりとして扱うようなスタイルへ。
  - **「逐次集中から並列分散へ」**
    - 分散して存在する複数の実行部が、協調して計算を行うというスタイルへ。

青木淳, オブジェクト指向システム分析設計入門, ソフト・リサーチ・センター, 1993
青木淳,『例題による！！オブジェクト指向分析設計テクニック』, ソフト・リサーチ・センター, 1994

オブジェクト指向
Object-Oriented

オブジェクト指向計算モデルとは

- オブジェクト指向では、世界の構成要素を「オブジェクト」という基本単位で捉え、その状態変化や関係変化によって現象を表現する。



振舞い(機能)と内部状態を保持している「オブジェクト」がたくさん存在し、それらが相互作用しているという点が、オブジェクト指向のポイント。

オブジェクト指向とシミュレーション

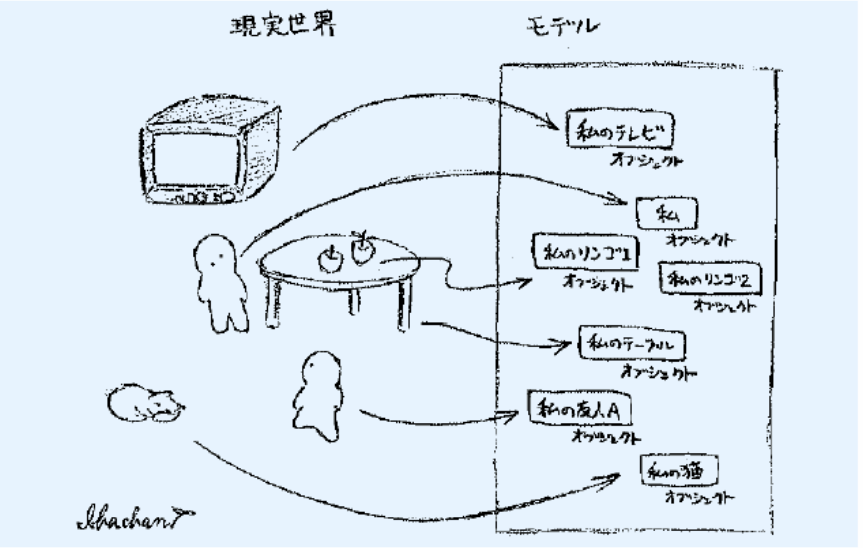
先週の復習

- オブジェクト指向の考え方の起源
  - シミュレーション用プログラミング言語 SIMULA
  - ノルウェーのO.J.ダールとK.ニガードが開発
  - 何千もの構成要素からなるような複雑なシステムのモデルを作成してコンピュータ上で動かすことを目的に設計された。
  - そのため、動的で複雑な現実世界をそのまま取り込むための工夫がなされた。

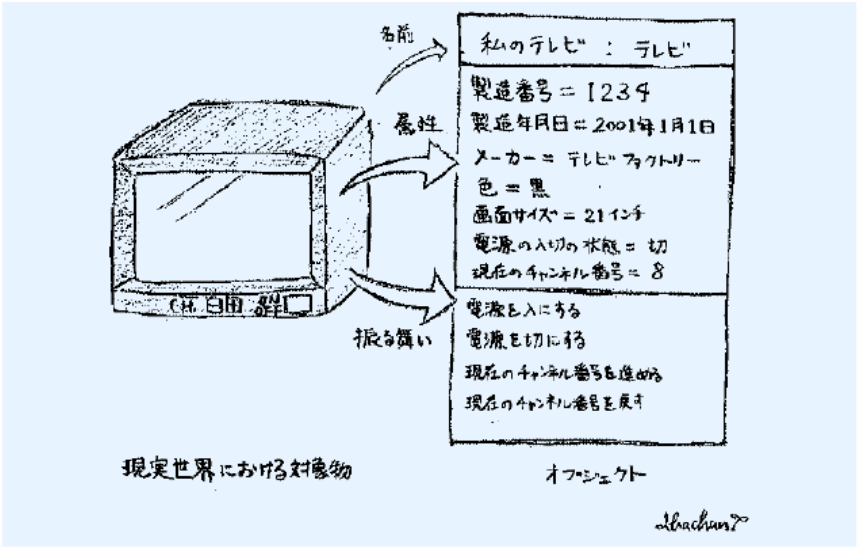
「オブジェクト指向」の広がり

- 実装のための考え方から、設計の考え方へ。
- そして現実の分析のための考え方へ。
  - ビジネスモデルの記述への適用なども模索されている。
- オブジェクト指向の記法は、近年、UML（Unified Modeling Language：統一モデリング言語）として標準化されている。
- プログラミング言語に置き換えて、コンピュータ・シミュレーションを行うことができる。

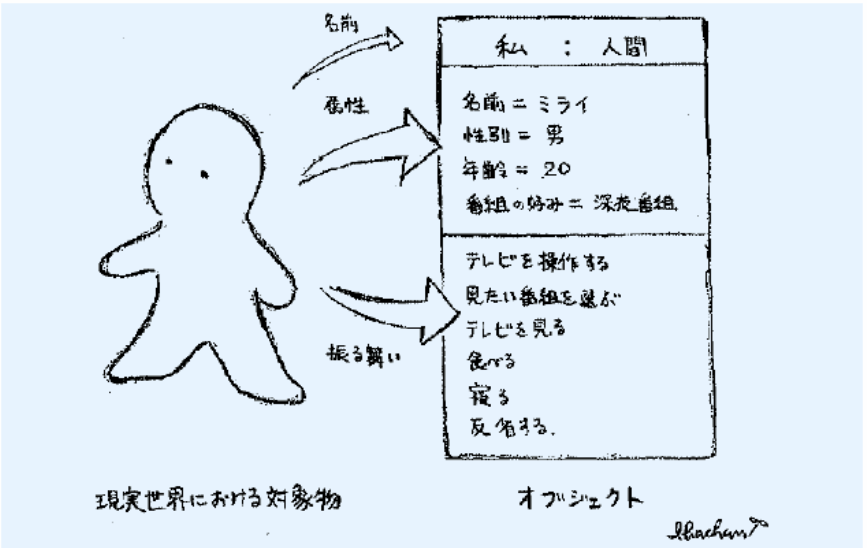
現実世界の構成要素をオブジェクトとして写し取る



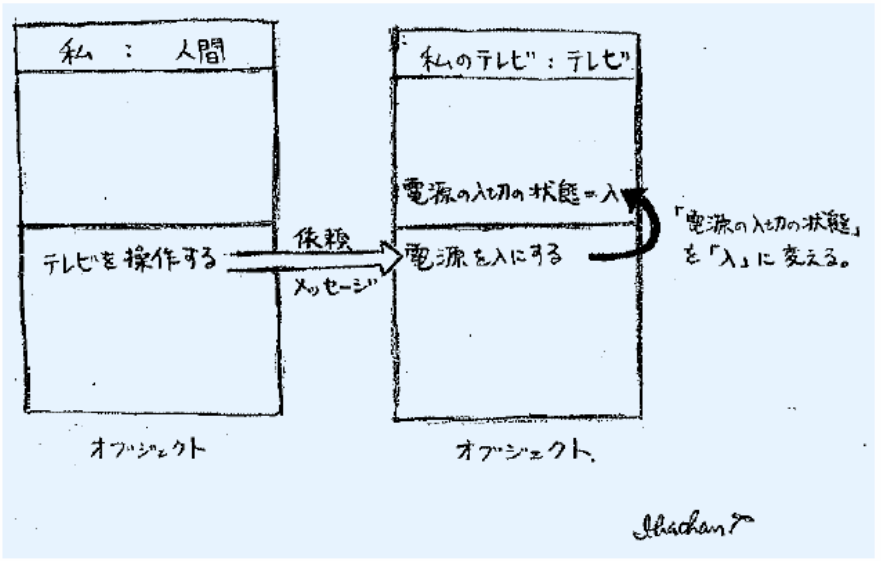
「私のテレビ」をオブジェクトとして表現すると・・・



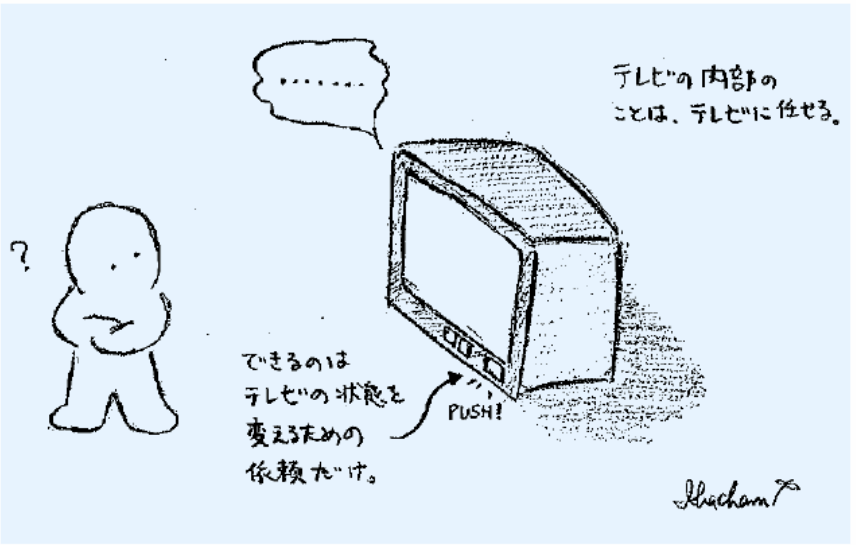
「私」をオブジェクトとして表現すると・・・



オブジェクトからオブジェクトへのメッセージ

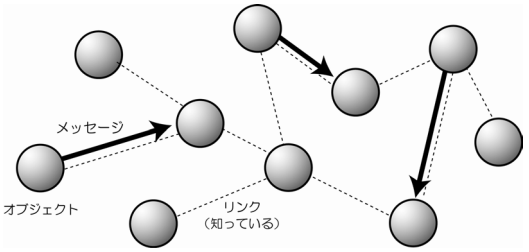


オブジェクトは情報隠蔽して自己管理する



オブジェクト指向計算モデルとは

- オブジェクト指向では、世界の構成要素を「オブジェクト」という基本単位で捉え、その状態変化や関係変化によって現象を表現する。



振舞い(機能)と内部状態を保持している「オブジェクト」がたくさん存在し、それらが相互作用しているという点が、オブジェクト指向のポイント。

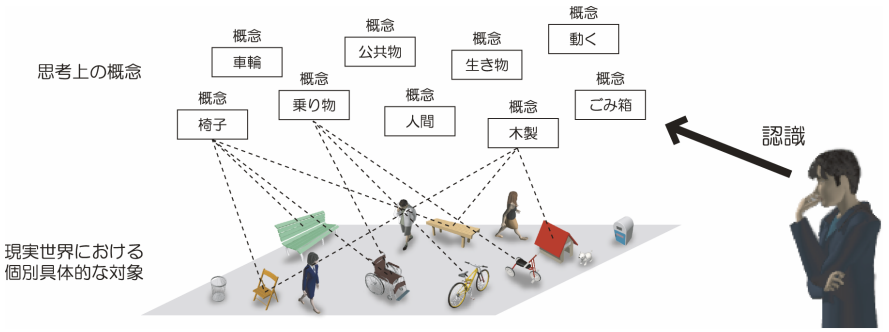
オブジェクト指向における
「クラス」

オブジェクト指向：クラス

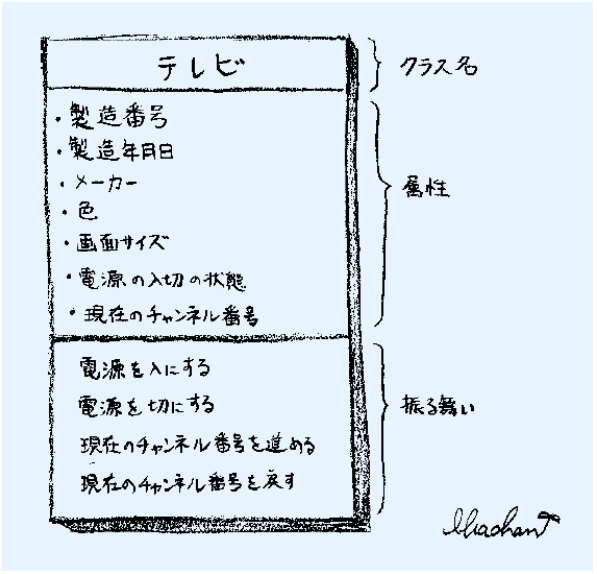
- 「クラス」とは、共通の性質（属性の種類と振舞い）をもつオブジェクトを分類したものである。

オブジェクト指向：クラスの利点

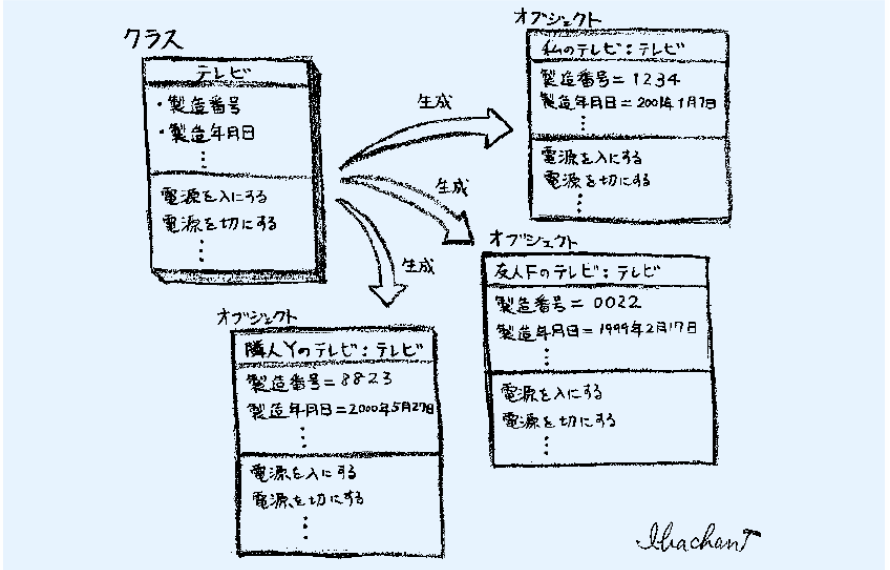
- オブジェクトをクラスで分類するということは、世界の複雑さに対処するためのひとつの方法。
- 人間の認知プロセスにおける「概念化」と同じメカニズム。



テレビクラス

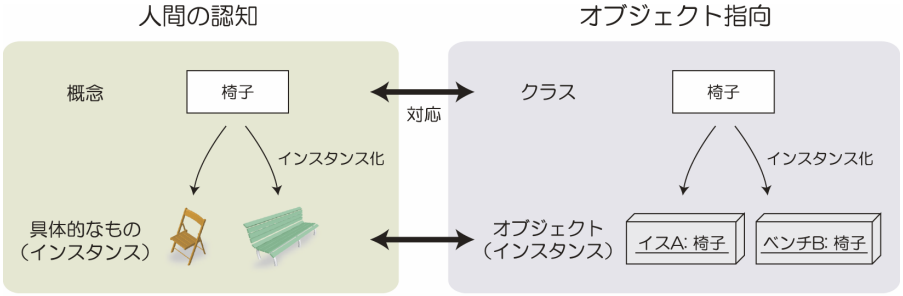


クラスからオブジェクトを生成

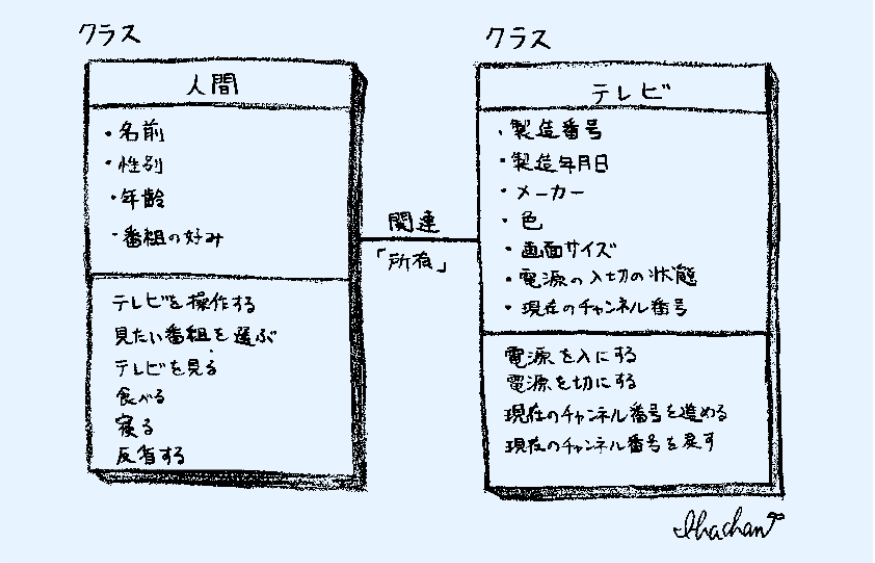


オブジェクト指向：クラスの利点

- クラスを用いることによって、共通項を一括して表現できるようになるため、オブジェクトの体系的な整理が可能となる上、効率的な記述が可能となる。



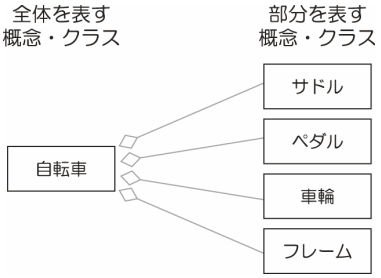
クラス間の関連



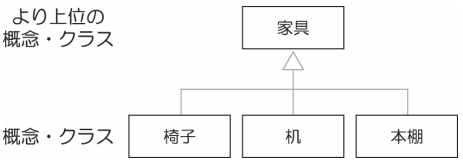
オブジェクト指向(4):クラス間関係

- 人間の認知と同様に、オブジェクト指向でも複数のクラスを関係づけるメカニズムを用いて複雑性に対処する。

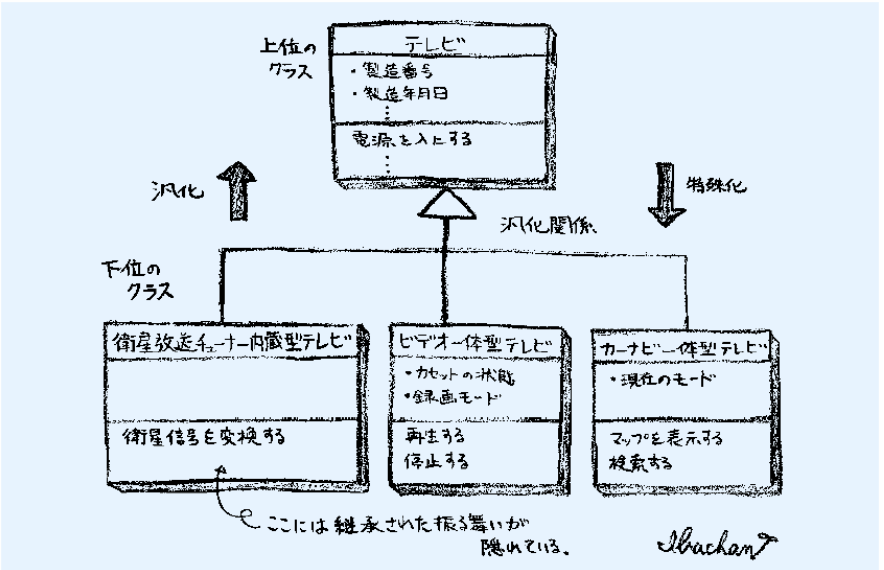
集約／複合化
aggregation / composition



汎化
generalization

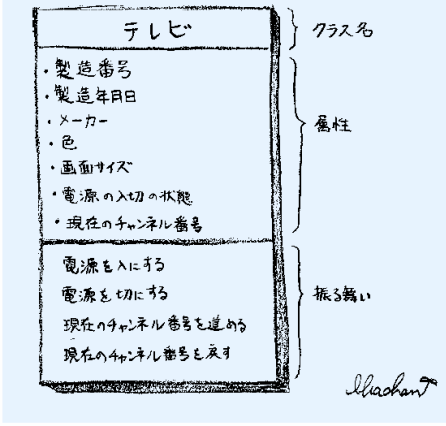


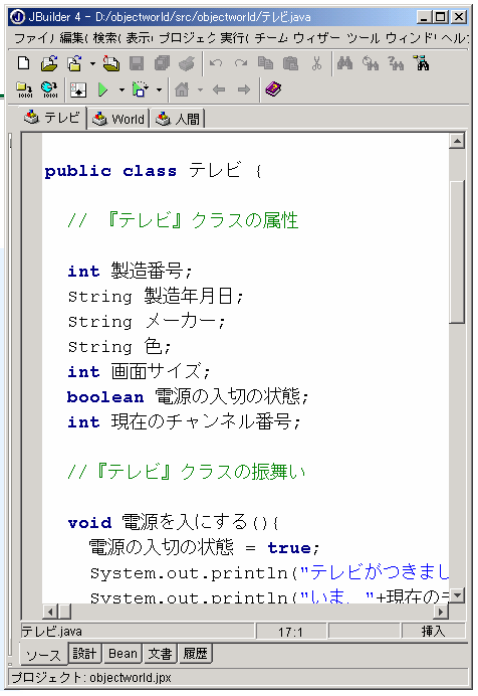
クラスの汎化と特化



オブジェクト指向プログラミング

- Java言語で表現すると・・・





オブジェクト指向のモデルを記述するための

UML：統一モデリング言語

UML (Unified Modeling Language)

- 50以上のオブジェクト指向方法論による方法論戦争の末、3人の代表的なメソドロジストが、Rational Software社に集まり、統一モデリング言語としてまとめる流れをつくった。

Booch法	OMT法 (Object Modeling Technique)	OOSE法 (Object-Oriented Software Engineering)
Grady Booch	James Rumbaugh	Ivar Jacobson

「スリーアミーゴ（Three Amigos)」

UMLの目標

- 「UMLの開発の背後には、さまざまな目標がありました、第1の最も重要な目標は、UMLがすべてのモデル作成者が利用することのできる汎用のモデリング言語となることです。UMLは所有権の設定されたものではないと同時に、コンピュータ業界の大多数による共通の合意の基づいたものです。」
- 「UMLの最終目標は、できるだけシンプルでありながら、それでいて構築しなければならない広範な実用システムをモデリングできるようにすることでした。」

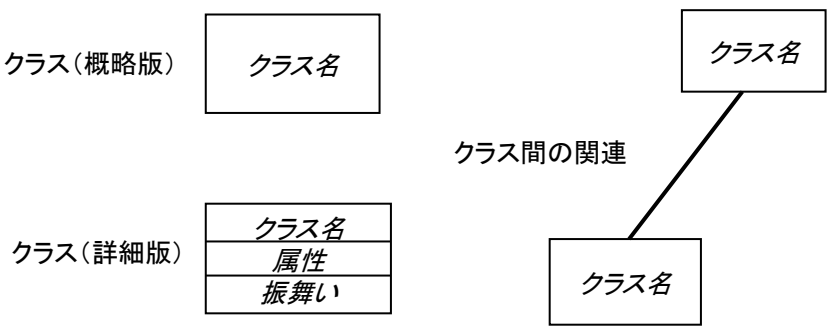
ジェームズ・ランボー, イヴァー・ヤコブソン, グラディ・ブーチ,『UMLリファレンスマニュアル』, 2002

UMLにおけるいくつかのビュー

- 静的モデリング
  - クラス図
  - ユースケース図
  - コンポーネント図
  - 配置図
- 動的モデリング
  - アクティビティ図
  - シーケンス図
  - ステートチャート図
  - コラボレーション図
- モデル管理
  - クラス図

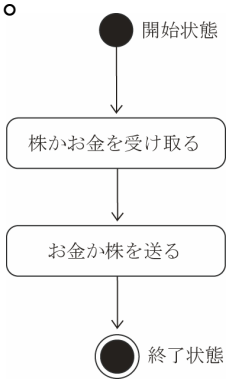
クラス図

- クラス図は、モデルの静的・構造的な側面を表現するための図。



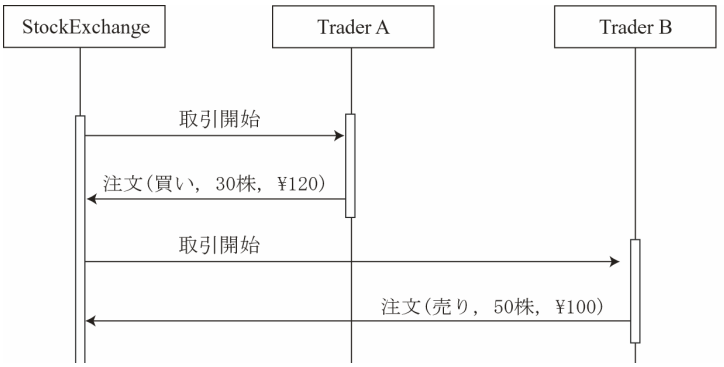
アクティビティ図

- アクティビティ図は、システムやオブジェクトの振舞いを記述するための図。
- フローチャートだと思ってよい。



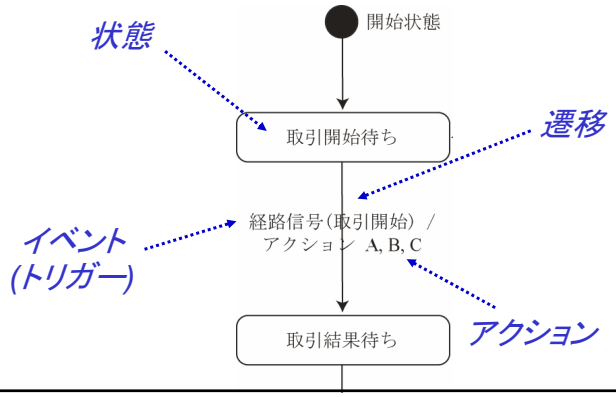
シーケンス図

- オブジェクト間の相互作用を、時系列で記述したもの。



ステートチャート図

- ステートチャート図は、システムやオブジェクトの状態の変化（状態遷移）を記述するための図。
  - 外界のイベント（オブジェクトに影響を及ぼすさまざまな出来事）が発生すると、オブジェクトの状態が変わる。



UMLにおけるいくつかのビュー

- 静的モデリング
  - クラス図
  - ユースケース図
  - コンポーネント図
  - 配置図
- 動的モデリング
  - アクティビティ図
  - シーケンス図
  - ステートチャート図
  - コラボレーション図
- モデル管理
  - クラス図

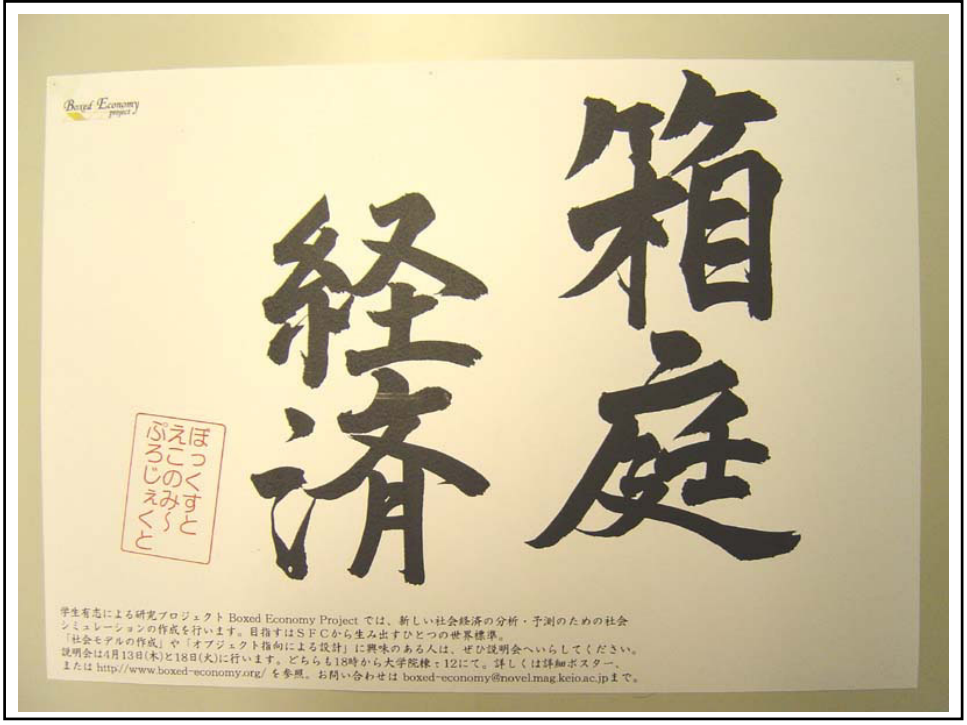
UMLによるビジネスモデリングについての文献

ハンス=エリク・エリクソン, マグヌス・ペンカー, 『UMLによるビジネスモデリング』, ソフトバンクパブリッシング, 2002

クリス・マーシャル, 『企業情報システムの一般モデル』, ピアソン・エデュケーション, 2001

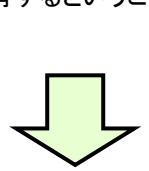
モデル・フレームワーク
～ Boxed Economy Approach ～


Boxed Economy project
学生有志による研究プロジェクト
A NEW RESEARCH PROJECT FOR AGENT-BASED ECONOMIC SIMULATIONS
ここから世界へ
Boxed Economy Project
まずは説明会。
4／13(木)
18:30〜説明会 20:30〜交流会
4／18(火)
18:30〜説明会 20:30〜交流会
大学院棟〒12にて
「箱庭経済」モデル
社会をまるごとシミュレート！
参加者大募集！
BOXED ECONOMY
http://www.boxed-economy.org/
★新しいコンセプトのもと、社会経済の分析や予測ができる
★政策提言や新しい社会の仕組みを提案できる
★オブジェクトモデリングから実装までの専門的な経験が積める
★最先端の科学的研究に貢献できる
何より、自分の得意分野・やりたいことをやることで、貢献してください！
Core Members
ADVISERS
ご質問などがございましたら boxed-economy@novel.mag.keio.ac.jp までご気軽に！

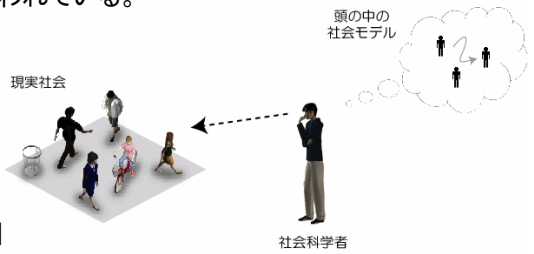

Boxed Economy project
箱庭経済
ぼっくすとえこのみ〜くと
ふろじぇくと
学生有志による研究プロジェクト Boxed Economy Project では、新しい社会経済の分析・予測のための社会シミュレーションの作成を行います。目指すはＳＦＣから生み出すひとつの世界標準。
「社会モデルの作成」や「オブジェクト指向による設計」に興味のある人は、ぜひ説明会へいらしてください。
説明会は4月13日(木)と18日(火)に行います。どちらも18時から大学院棟〒12にて。詳しくは詳細ポスター、
または http://www.boxed-economy.org/ を参照。お問い合わせは boxed-economy@novel.mag.keio.ac.jpまで。

モデルフレームワークの戦略的導入

- 現実世界を分析・体系化する際に、毎回白紙の状態から行うのは大変な作業となる。
- このような問題への戦略的なアプローチとしては、
  - 科学的研究では、概念や用語、理論などを定義し、共有する。
  - ソフトウェア工学では、ドメインに特化したフレームワークを定義し、共有するということが行われている。

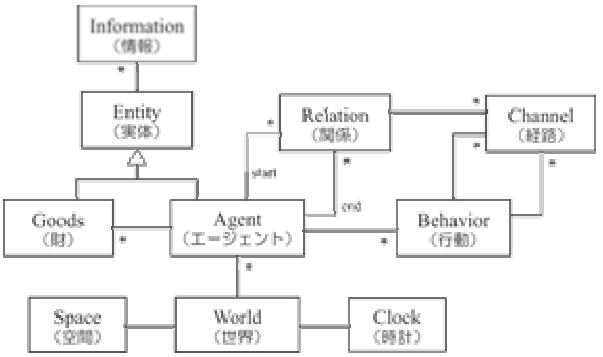


「モデル・フレームワーク」



モデルフレームワーク：Boxed Economy基礎モデル

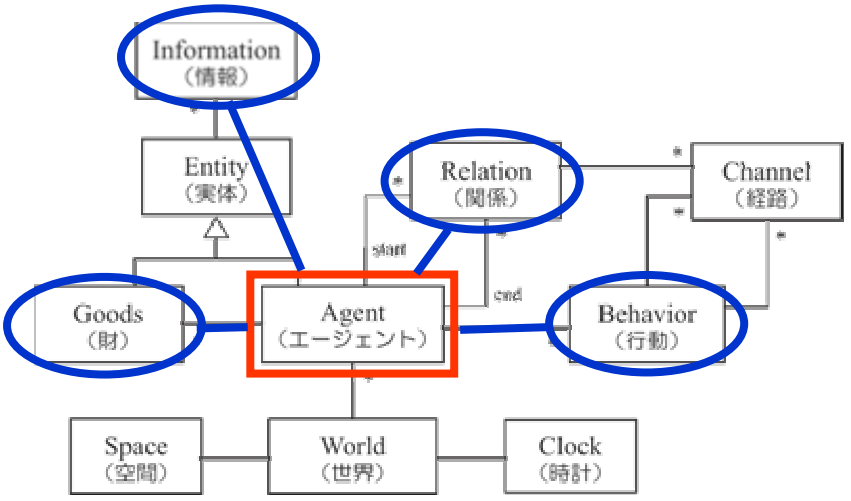
- 現実の経済社会のもつ構造をオブジェクト指向分析によって抽象化し作成したモデル・フレームワーク
- エージェントベースによる社会・経済のモデルのための基本デザインを提供する



Boxed Economy 基礎モデルの中心的なクラス図

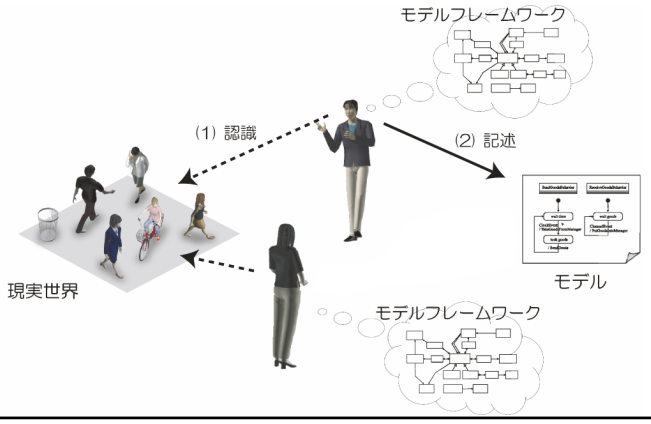
Boxed Economy 基礎モデル

BEFM概念モデル・フレームワーク

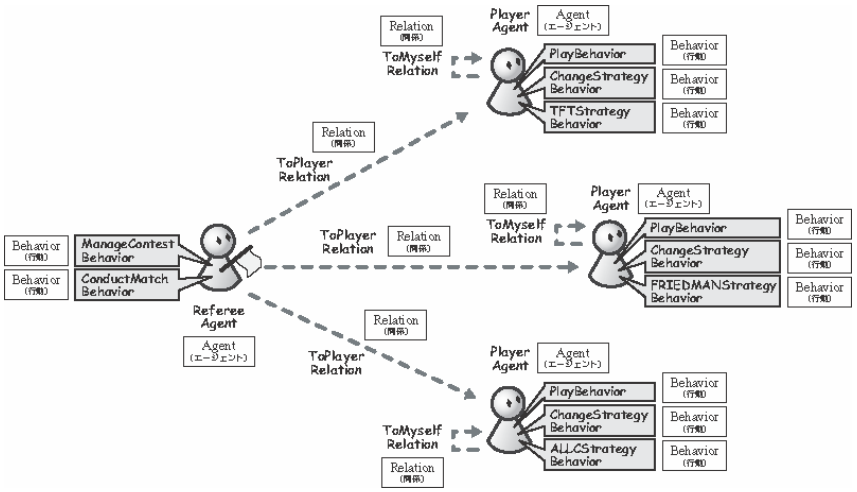


モデルフレームワークの役割

- 現実世界の認識のための準拠枠
- モデルを記述するための語彙
- モデル作成者間のコミュニケーションのためのコード

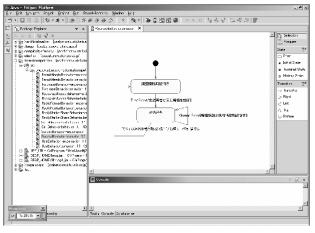


例えば、囚人のジレンマシミュレーションの場合



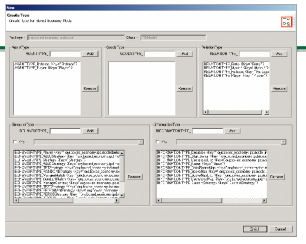
コンポーネントビルダー

- 3つのエディタ



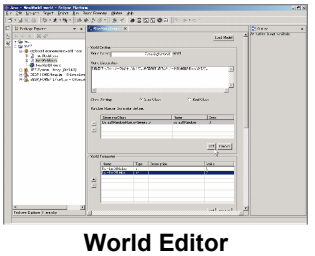
Behavior Editor

（エージェントの行動の状態遷移を記述）



Type Editor

（モデル要素の語彙の宣言）



World Editor

（シミュレーション世界の設定）

eclipse

コンポーネントビルダーによって自動生成されたコード例

AbstractSellVCRBehavior

```
/*
 * AbstractSellVCRBehavior.java
 */
package org.boxed_economy.formatcompetition.model.behavior;

import org.boxed_economy.besp.model.fmfw.bahavior.AbstractBehavior;

import org.boxed_economy.besp.model.fmfw.ChannelEvent;
import org.boxed_economy.besp.model.fmfw.TimeEvent;
import org.boxed_economy.besp.model.fmfw.bahavior.Action;
import org.boxed_economy.besp.model.fmfw.behavior.CompositeState;
import org.boxed_economy.besp.model.fmfw.behavior.Event;
import org.boxed_economy.besp.model.fmfw.behavior.GuardCondition;
import org.boxed_economy.besp.model.fmfw.behavior.State;
import org.boxed_economy.besp.model.fmfw.behavior.StateMachineFactory;
import org.boxed_economy.besp.model.fmfw.behavior.Transition;

/**
* AbstractSellVCRBehavior
*/
public abstract class AbstractSellVCRBehavior extends AbstractBehavior {

    /**
    * This method automatically generated from AbstractBehavior Builder
    * Don't touch by hand
    */
    protected void initializeStateMachine() {
        //factory
        StateMachineFactory factory = this.getStateMachine();

        //states
        State initialState = factory.createInitialState();
        CompositeState 販売スタンバイ = factory.createCompositeState("販売スタンバイ");

        //actions
        Action すべての注文についてVCRを譲渡して販売する = new Action() {
            public void doAction() {
                すべての注文についてVCRを譲渡して販売する();
            }
            public String toString() {
                return "すべての注文についてVCRを譲渡して販売する";
            }
        };

        Action 注文を記録する = new Action() {
            public void doAction() {
                注文を記録する();
            }
            public String toString() {
                return "注文を記録する";
            }
        };

        //guard-conditions
        GuardCondition 注文 = new GuardCondition() {
            public boolean isMatched(Event e) {
                return 注文(e);
            }
        };

        //transitions
        Transition transition販売スタンバイ=販売スタンバイ = factory.createTransition();
        Transition transition販売スタンバイ=販売スタンバイ1 = factory.createTransition();
        Transition transitionInitialState1=販売スタンバイ = factory.createTransition();
```

```
        //states setting

        //structure of states
        this.setInitialState(initialState);
        this.addState(販売スタンバイ);

        //transitions setting
        transition販売スタンバイ=販売スタンバイ.setEventTimeEvent.class);
        transition販売スタンバイ=販売スタンバイ.addAction(すべての注文についてVCRを譲渡して販売する);
        transition販売スタンバイ=販売スタンバイ1.setEventChannelEvent.class);
        transition販売スタンバイ=販売スタンバイ1.setGuard(注文);
        transition販売スタンバイ=販売スタンバイ1.addAction(注文を記録する);

        //connection of transitions
        transition販売スタンバイ=販売スタンバイ.setSourceState(販売スタンバイ);
        transition販売スタンバイ=販売スタンバイ.setTargetState(販売スタンバイ);
        transition販売スタンバイ=販売スタンバイ1.setSourceState(販売スタンバイ);
        transition販売スタンバイ=販売スタンバイ1.setTargetState(販売スタンバイ);
        transitionInitialState1=販売スタンバイ.setSourceState(initialState);
        transitionInitialState1=販売スタンバイ.setTargetState(販売スタンバイ);
    }

    /**
    *すべての注文についてVCRを譲渡して販売する
    */
    protected abstract void すべての注文についてVCRを譲渡して販売する();

    /**
    *注文を記録する
    */
    protected abstract void 注文を記録する();

    /**
     *注文
     */
    protected abstract boolean 注文(Event e);
}
```

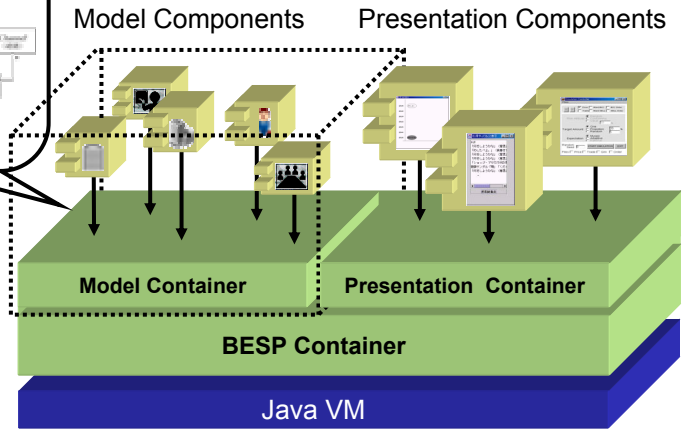
SellVCRBehavior

```
/*
 * SellVCRBehavior.java
 */
package org.boxed_economy.formatcompetition.model.behavior;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

import org.boxed_economy.besp.model.fmfw.Channel;
import org.boxed_economy.besp.model.fmfw.Goods;
import org.boxed_economy.besp.model.fmfw.behavior.Event;
import org.boxed_economy.besp.model.fmfw.informations.IntegerInformation;
import org.boxed_economy.formatcompetition.model.FormatCompetitionModel;
import org.boxed_economy.formatcompetition.model.information.OrderInformation;

public class SellVCRBehavior extends AbstractSellVCRBehavior {

    private Map orders = new HashMap();

    /**
     * @see org.boxed_economy.besp.model.fmfw.behavior.AbstractBehavior#initialize()
     */
    protected void initialize() {
    }

    /**
     * @see org.boxed_economy.besp.model.fmfw.bahavior.AbstractBehavior#terminate()
     */
    protected void terminate() {
    }

    /**
     * @see org.boxed_economy.formatcompetition.model.behavior.AbstractSellVCRBehavior#注文を記録する()
     */
    protected void 注文を記録する() {
        this.orders.put(this.getActiveChannel(), this.getReceivedInformation());
    }

    /**
     * @see org.boxed_economy.formatcompetition.model.behavior.AbstractSellVCRBehavior
     * #すべての注文についてVCRを譲渡して販売する()
     */
    protected void すべての注文についてVCRを譲渡して販売する() {
        Iterator i = this.orders.keySet().iterator();
        while (i.hasNext()) {
            Channel channel = (Channel) i.next();
            OrderInformation order = (OrderInformation) this.orders.get(channel);
            Goods vcr = this.getWorld().create(Goods.order.getFormat(), 1.0);
            channel.sendGoods(vcr, this);
        }
        this.orders = new HashMap();
    }

    /**
     * @see org.boxed_economy.formatcompetition.model.behavior.AbstractSellVCRBehavior#注文(Event)
     */
    protected boolean 注文(Event e) {
        return this.getWorld().getInformation(typeof(this.getReceivedInformation())
            == FormatCompetitionModel.INFORMATION_Order;
    }

}
```

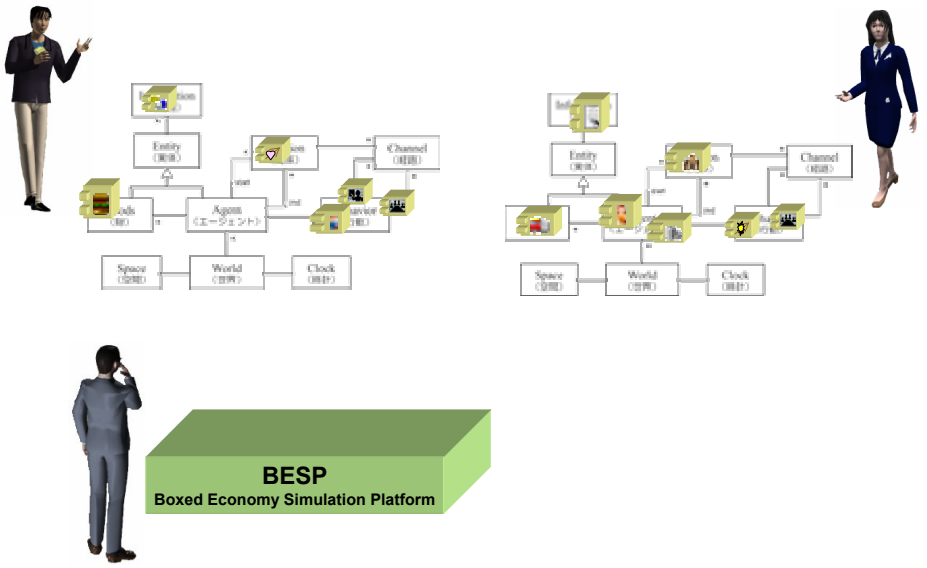
Boxed Economy Simulation Platformの基本構造

Boxed Economy
基礎モデル

BESPでは、コンポーネントとフレームワークの考え方に基づく設計がなされている。シミュレーションのモデルや実行環境をコンポーネントとして分割して定義するため、それらを組み替えることによってユーザー独自のシミュレーションを柔軟かつ容易に構築できる。



コンポーネントの共有・再利用が可能



今後の予定

- 第6回 (11/ 5 水) ←今日
- 第7回 (11/12 水) シミュレーションの作成①

（三田祭休み等）

- 第8回 (12/ 3 水) シミュレーションの作成②
- 第9回 (12/10 水) 対象分析と概念モデルの作成①

第6回フィードバックコメント

●自分がやってみたいシミュレーションの内容・イメージ
●今日出てきた話題についての再考・感想など。

- アドレス（宛先）

ハイフン

simu-staff@sfc.keio.ac.jp

- サブジェクト（題名）と 本文1行目

カンマ（半角）

0123456,山田花子,FC06

今回は第6回の授業なので、06

学籍番号（半角）　姓名（全角漢字）　FC（半角）　何回目の授業か（半角）

- 締切
  - 11月8日（土） 23時30分

※CNSのアドレス以外からの提出でもよい。ただし携帯メールは不可。