エージェントベース社会シミュレーション のためのフレームワークデザイン

井庭崇 †1†2†3, 岩村拓哉 †1, 広兼賢治 †5, 竹中平蔵 †3†4, 武藤佳恭 †5

†1 慶應義塾大学政策・メディア研究科
†2 日本学術振興会特別研究員
†3 フジタ未来経営研究所
†4 慶應義塾大学総合政策学部
†5 慶應義塾大学環境情報学部

1 はじめに

近年国内外を問わず，マルチエージェントモデルによる社会シミュレーションが脚光を浴び始めている [1, 2, 3]．エージェントベース社会シミュレーションは特に経済学や市場研究の分野では，現在主流の新古典派経済学や効率的市場仮説への批判を受けるかたちで，新しい理論の模索のための実験的研究が意欲的に行われている．例えば，単純な経済シミュレーションである Sugerscape モデル [4]，Agent-Based Keynesian Economics モデル [5]，経済全体を扱う ASPEN モデル [6, 7, 8, 9]，株式市場をモデル化した stockmarket モデル [10]，外国為替市場を扱う AGEDASI TOF モデル [11, 12] などがある．そのような状況の中で，我々もこれまでに Winner-Take-All 現象やバブル現象などに関するシミュレーションを作成してきた [13, 14, 15, 16].

エージェントベース社会シミュレーションに関する研究の現状には，技術的な観点から見ると改善すべきいくつかの問題がある．まず第一に，ほとんどのシミュレーションプログラムがその度ごとにゼロから作られているという点である．その結果個々のシミュレーションプログラムを開発するのに莫大なコストと時間がかかることになる．

第二に，ほとんどのシミュレーションプログラムはソースコードが公開されておらず，追試や拡張ができない．シミュレーションプログラムには，紙面の限られた論文には記述しきれない様々なコード化のテクニックも含まれているため，他の研究者が行ったシミュレーションを再構成することは非常に困難である [17]．またソースコードが公開されていないため，シミュレーションが誤りなくコード化されているかを調べることができない．このことは，科学的方法としてシミュレーションを用いる際の大きな問題となる．

これまでの社会シミュレーション研究は対象モデルが小規模であったり実験的意味合いが強かったために，再利用性や拡張性のある設計とその開発プロセスについての議論はほとんどなされてこなかった．上で述べた問題を解決するためには，社会シミュレーションプログラムを単なる実験の道具としてとらえるのではなく，ひとつの開発製品として重視し，その開発方法に関する議論を深めていく必要がある．

本論文では，フレームワークデザインを考慮したエージェントベース社会シミュレーションの開発について論じ，ソフトウェア工学の知見を社会シミュレーションに適用した「社会シミュレーション工学」(Social Simulation Engineering) を提唱する．まず最初に，ソフトウェアの再利用とフレームワークの概念を整理し，それらの社会シミュレーション開発への応用を考察する．そしてモデルケースとして現在開発中のエージェントベース経済シミュレーションのためのフレームワークを取り上げる．最後に今後の課題を整理し，まとめとする．

2　社会シミュレーションのソフトウェア的側面

社会シミュレーションとは，コンピュータ上に社会モデルを作成して社会をシミュレートするものであるが，これには従来あまり言及されることのなかった重要な側面がある．それは社会シミュレーションのソフトウェア的側面，つまりコンピュータプログラムとしての側面である．社会シミュレーション分野の進展に伴い，科学的信頼性の高い大規模なシミュレーションを作成することが求められつつあるため，社会シミュレーションのソフトウェアとしての側面を重視し，その品質や開発プロセスに注意を向ける必要がある．ここでは，ソフトウェアとしての社会シミュレーションの品質，およびその開発プロセスについて考察する．

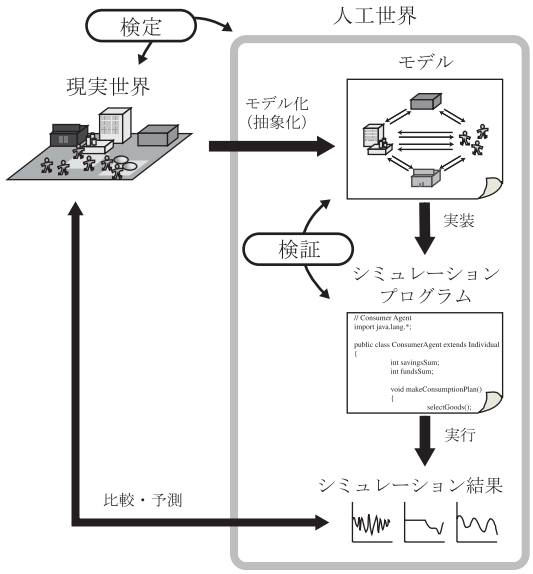

図 1: シミュレーションの検証と検定

2.1　社会シミュレーションの品質

社会シミュレーションのソフトウェアとしての品質を，項目別に整理すると表 1 のようになる[1]．この表の中で，科学的な観点からみて特に重要なのものは「正しさ」に関する品質である．一般に，ソフトウェアには潜在欠陥はつきものであり，しかもソフトウェアの規模が大きくなるほど潜在欠陥も増大するため，開発過程や保守過程における徹底的な品質管理が重要である [22]．言うまでもなく科学研究や政策分析で用いられる社会シミュレーションの場合,「正しさ」に関する品質はゼロ欠陥であることが求められる．

シミュレーション研究の観点からみると，通常，「検証」(verification) と「検定」(validation) という二つの妥当性チェックが行われている（図 1）．前者はシミュレーションが正しくコード化されて動作しているかどうかの判定であり，後者は現実と比べてモデルが妥当であるかの判定である [24, 25, 26]．シミュレーションの場合，何らかの振舞いがモデルから演繹的に生じるが，モデルが複雑化するほど結果の事前予測が難しくなり，プログラムの検証が困難になる [17]．そのため，シミュレーションの開発プロセスの中にソフトウェア品質を確保するための仕組みを導入する必要がある．

[1] ソフトウェアの品質については過去に多くの議論があるが，一般に定義が難しいといわれている [18, 19, 20, 21, 22]．ここでは文献 [23] で用いられた 6 項目で分類・整理した．

2.2　インクリメンタル型開発プロセス

社会シミュレーションのモデルを開発する際には，コンピュータ上に対象を構成していく過程でその対象を理解していくという「構成的手法」のプロセスがとられる [27, 28]．多くの場合，社会シミュレーションを構成する部分モデルは実証されていない仮設的なものであるため，代替的な部分モデルと交換・比較しながらの模索的な作業となる．この構成的手法の理解の仕方は，ソフトウェアを構築するプロセスの中でシステム要件が理解されていくという，ソフトウェア開発における経験と類似している [29, 30]．

またすべてのモデルがそうであるように，社会モデルもひとつの抽象であり，絶対的で完全なものにはなり得ない．対象である社会は自由度が高く，しかも不断に変化するものであるため，モデルもそれに合わせた変更を余儀なくされる．

以上のような理由から社会シミュレーションでは，分析，設計，実装，テストというサイクルを繰り返すインクリメンタル型プロセスによって開

表 1: 社会シミュレーションにおけるソフトウェア品質

項目	品質の意味	社会シミュレーションにおいて重要な理由
正しさ	シミュレーションモデルが要求仕様通りに実装されており，障害がなくシミュレーションが実行できること．	プログラムの故障や誤ったコード化によってシミュレーションの結果が左右されてしまうため．
使いやすさ	シミュレーション分析を行いたい利用者が，学びやすく使いやすいこと．	社会科学者など，コンピュータの専門家以外が利用するため．
再利用性	ソフトウェアシステム，または一部のモデルコンポーネントが，他のシミュレーション等で再利用できること．	類似した社会シミュレーションがそれぞれゼロから開発されるということは，個別としても全体としても効率が悪いため．
読みやすさ	プログラムの構造やソースコードが読みやすく理解しやすいこと．	間違いなく実装されているかの検証や，モデルを拡張する際に，本人や他人がソースコードを読む必要があるため．
効率	ハードウェアやソフトウェアの資源を最適に利用していること．	シミュレーションの実行速度が遅くなるとシミュレーション分析に支障をきたすため．
可搬性	特定のハードウェアやソフトウェアプラットフォームに依存しない実装がされていることや移植が容易であること．	利用者のもつ様々な環境で利用できるようにするため．

発することが適している（図 2）．このインクリメンタル型のプロセスを支援するために，以下では再利用とフレームワークの概念を導入する．

3　ソフトウェア開発における再利用

開発対象の複雑化によって引き起こされる開発費用や難易度の増大という「ソフトウェア危機」[31] の問題に対し，再利用技術の重要性が認識されてから多くの研究が行われてきた [32, 33, 34]．特に 1990 年代になってからオブジェクト指向におけるフレームワークやデザインパターンの概念の発展と共に，多くの研究成果が発表されている [35, 36]．

ソフトウェアにおける再利用は，分析，設計，実装およびテストのすべての段階で実現可能であり，社会シミュレーションにおいても同様にすべての段階で再利用が可能である．Jones[37] は再利用可能なものとして，見積，計画，ユーザ用文書，要求定義，ヒューマンインターフェース，画面，テスト計画，テストケース，アーキテクチャ，データ，設計，コードの 12 種類をあげている．

これらの再利用を行うことによって開発コストが大幅に減少することが知られており，Jacobson ら [30] は様々な分野において再利用率が 50%から 90%を実現したプロジェクトの事例を紹介している．またその実質的な利益として，開発期間の短縮 (2 分の 1 から 5 分の 1)，欠陥発生率の減少 (5 分の 1 から 10 分の 1)，保守コストの減少 (5 分の 1 から 10 分の 1)，一般的なソフトウェア開発コストの減少 (長期プロジェクトで 15%から 75%の減少）があることを述べている．

ここで注意すべき点は，再利用を意識した開発は全体としてはコストが減少するものの，その初期開発は通常よりもコストがかかるという点である．そのため，必要がある場合にのみ再利用を意識した開発を行うべきである．Jacobson ら [30] は経済的観点から，再利用可能なコンポーネントは通常より 1.5 倍から 3 倍のコストがかかり，そのコストを回収するために 3 回から 5 回使用される必要があるとしている．

社会シミュレーションの場合，科学的研究過程で用いられるため，何度も同じ分析・モデル・データ等が利用される可能性が高い．また，他の研究者が行ったシミュレーションの設定を変えて再実験することがしばしばあるため，再利用を意識した設計を行うメリットが十分にあると判断できる．

以下では，特に設計とコードの再利用に焦点を

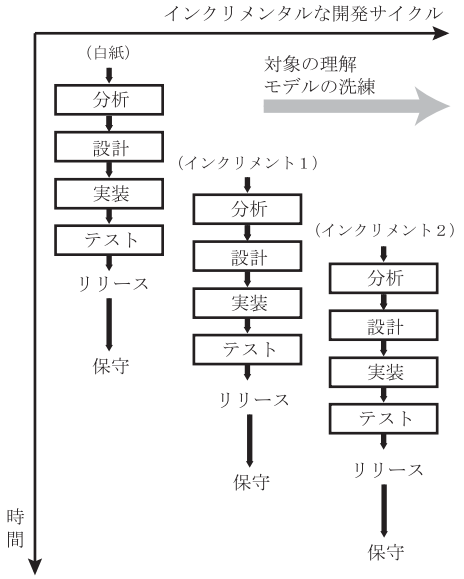

図 2: 構成的手法とインクリメンタル型開発の融合

絞って考察を行う．

4 社会シミュレーションにおける設計とコードの再利用

4.1 オブジェクト指向モデリング

エージェントベース社会シミュレーションの作成には，差分実装などによる高い拡張性，生産性，信頼性などの特徴があるオブジェクト指向モデリングおよびオブジェクト指向プログラミングが有効である．これらの特徴は，開発効率の向上という工学的利点だけでなく，シミュレーション結果の信頼性の向上や，モデルの交換・拡張による理論発展の可能性という科学的利点がある．

また，オブジェクト指向の起源がシミュレーション用言語 (Simula [38]) であることからもわかるように，オブジェクト指向モデリングはシミュレーションを作成する際に，より自然なモデル化を可能とする．エージェントベース社会シミュレーションでは，現実と同様の要素構成および関係性で人工社会を構築するので，機能を抽象化してモデル化した場合に比べ，現実の変化に伴うモデルの変更・拡張が容易になるという点でも，社会モデルを扱う技術として適切である．

しかし，単にオブジェクト指向方法論を用いるだけではこれらの特徴を活かすことができないということが経験的に明らかになっている[2] [23, 30, 39]．そこで近年，設計の単位として複数のオブジェクトの関係性をまとめたメカニズムやフレームワークの概念が注目されており，それらを有効に活用するための開発プロセスや組織形態が議論の対象になっている [36, 40]．

4.2 ３つのレベルのパターン

ソフトウェアプログラムには，いくつかのレベルの抽象において再利用可能なパターンが存在する．それは，実装言語レベルの「イディオム」，特徴的なオブジェクト間コラボレーションを取り出した「メカニズム」，アーキテクチャを記述する「フレームワーク」である（図 3）．これらのパターンを適切に活用することにより，生産性および品質の向上が期待される．また，技術者の経験の差から派生する設計・コードの差異を小さくし，読みやすく拡張しやすいプログラムを構築できるため，再利用や検証を必要とする社会シミュレーションにおいてこれらのパターンは重要な概念となる．ここでは，抽象度の異なる３つのパターンを社会シミュレーション分野へ適用することを考える．

- **イディオム**

  イディオムとは，実装言語の慣習的な表現に関する共有された約束事項である．例外の扱い方のパターンやネーミング規則などがこれにあたり，コーディングパターンとも呼ばれている．このような一貫した記述の規則は，共同開発や共有化を行う際に，開発者同士で混乱が生じないための前提条件となる．イディオムはプロジェクトの初期段階でスタイルガイドとしてまとめられ，これに基づいてコード化が進められることになる．

[2] Jones[22] は，オブジェクト指向分析・設計に関する学習曲線が非常に急であることを指摘し，開発プロジェクトが新規にオブジェクト指向方法論を採用することは，短期的には不利でさえあると述べている．

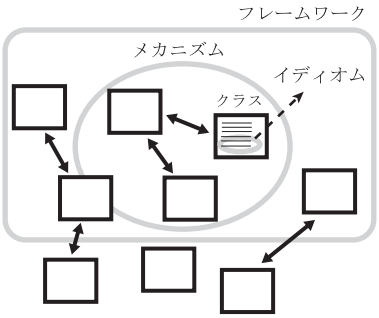

図 3: 再利用可能な 3 つのパターン：イディオム，メカニズム，フレームワーク

再利用や拡張可能な社会シミュレーションの開発においても，事前にスタイルガイドを共有しておく必要がある．ただし，社会シミュレーションにおける独自のイディオムを考案するというよりは，一般的なソフトウェア開発におけるイディオムを活用することになると思われる[3]．

- **メカニズム**
  メカニズムは，ある特徴的なオブジェクト間コラボレーションの仕組みに名前をつけたものである．イディオムがクラスなどの設計に関して言及しないのに対し，メカニズムは，ある振る舞いを達成するためにオブジェクトがどうのように協調するのかという小規模の設計を示す．

  モデル化の際に，過去に提案されているメカニズムを採用することによって開発効率が向上する．また，相互作用をまとまりとして捉えることができるため，プログラムの可読性も向上する．柔軟なメカニズムをいかにして設計するのかのノウハウは，デザインパターンとしてまとめている [35].

  社会シミュレーション開発においてはイディオムと同様，一般的なソフトウェア開発で用いられるメカニズムを適用することになると思われる．

- **フレームワーク**
  フレームワークは，ドメインに特化したソフトウェアのアーキテクチャである．フレームワークには，その分野に必要だと思われるコンポーネントの集合と，それらをまとめるアーキテクチャ部分が含まれている．フレームワークを基盤としてアプリケーションを作成する場合には，適切なコンポーネントが存在すればそれを利用し，存在しない場合には継承によって差分を実装して新しい振る舞いを追加する．フレームワークに焦点をあて，まとまりとして扱うことにより，プログラムのコードだけでなく設計も再利用できるということになる[4]．

  社会シミュレーションを作成する際にも，既に開発されているフレームワークを利用することができれば，開発コストが減り，開発速度が向上する．社会シミュレーションモデルのフレームワークとしては，一般経済モデルやや人工市場モデルのフレームワークなどが考えられる．またコンポーネントとしては，エージェントやそれらの相互作用を定義したものなどが考えられ，さらにエージェント内部の各アルゴリズムもコンポーネントとして扱うことができる．この社会モデルのアーキテクチャと，個々のコンポーネントの両方を再利用することが可能である．

  一段高いレベルの抽象度では，モデルとユーザーインターフェースのアーキテクチャがフレームワークとして定義できる．ユーザインタフェースを分離することにより，モデルの対象となるドメインの知識をもたずに，ユーザーインタフェースを独立して開発できることになる．これは，Bruderer と Maiers [51] が指摘した社会シミュレーションの課題「ユーザーインタフェースの貧しさ」に対するひと

[3] 例えば，Java 言語におけるイディオムは文献 [41, 42] などにまとめられている．

[4] オブジェクト指向フレームワークの最初の成功例は，グラフィカルユーザーインターフェースの分野であり，Smalltalk の MVC(Model-View-Controller)[43] や MacApp [44, 45, 46]，ET++ [47, 48] などがある．ビジネスアプリケーションの分野では IBM SanFrancisco フレームワークがあり，典型的な業務アプリケーションの約 40%をカバーしているといわれている [49]．シミュレーション分野では，Campos と Hill[50] によってフレームワークの初期の研究が生態系のシミュレーションに適用されている．

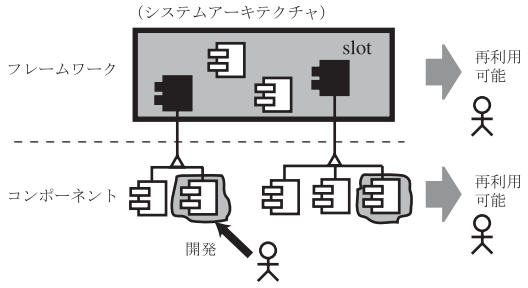

図 4: フレームワークにおける再利用

つの解決策となると期待できる．

5 フレームワークの開発

ここでは，現在議論されているフレームワーク開発に関する事項を整理し，社会シミュレーションへの適用を考えるための準備とする．

5.1 フレームワークの構造

フレームワークは，インターフェースが定義されている複数の抽象クラスと具象クラスとで構成されており，それらの組み合わせと相互作用が定義されているアーキテクチャのテンプレートである (図 4)[52]．フレームワークは，それをベースにして開発される個別アプリケーションに対し，一意に決められない部分をプラグポイントとして用意し，コンポーネントを後から実装してプラグインできるようになっている．

ここで，フレームワークは単なるクラスライブラリとは異なるという点を強調しておきたい．クラスライブラリではシステムの一部をなす部品が共有されており，開発者はそれらを呼び出すアーキテクチャを書くことになる．これに対してフレームワークでは，部品だけでなくアーキテクチャも共有することが可能である．開発者はフレームワークで定義されていない部分を，共有されている部品の中から選択するか，新たに作成することになる．またフレームワークで定義されている部分についても，オーバーライドすることによって変更ができる．

5.2 フレームワークの開発プロセス

フレームワークは設計とコードの再利用のために有効であることがわかっているが，一般にその構築は困難であるといわれている [53]．これは，複数のアプリケーションに適用するための汎用性を用意することが難しいということに起因する．また，従来のオブジェクト指向方法論では，フレームワークの構築や再利用の概念が扱われておらず，開発サイクルに組み込まれていなかったことも大きな原因となっている [23, 30, 54]．そこで現在，フレームワークを考慮した開発方法に関する議論が活発に行われている．これらの議論を整理すると大きく二つに分類される．

一つは当初から指摘されてきたもので，類似した複数のアプリケーションを開発した後に，汎用的な部分を抽出してフレームワークを構築するという方法である [52, 55]．フレームワークの成熟には３つ以上の具体的アプリケーションを作成する必要があるといわれている [56, 53]．

これに対しもう一方では，最初からフレームワークの構築を目指すための開発方法 (a priori framework development) が提案されている [57, 58, 59, 59]．これは，フレームワークにおいて頻繁に変化するホットスポットと固定的なフローズンスポットとをあらかじめ識別し，ホットスポットに柔軟性をもたせるように設計を行うというものである [60, 23, 61]．このとき追加や拡張される可能性がある部分や，実現方法が複数ある部分を識別し，それらがどのくらいの柔軟性を備えるべきかをあらかじめ判断するため，ドメインに特化した知識が求められる．これにはドメイン分析やユースケースによる分析手法などが用いられている．

いずれのアプローチを採用する場合でも，開発はインクリメンタル型で進められ，フレームワークは徐々に洗練され発展していくことになる．ここで，繰り返しプロセスが重要なのは確かであるが，それにもまして初期のデザインは今後のアーキテクチャの発展の基礎となるために極めて重要である [30]．また，フレームワークの開発を成功させる鍵は，段階的なフレームワーク構築の計画を，プロジェクトの初期段階から意識しておくことだといわれている [30]．社会シミュレーション

の開発においても，初期段階からフレームワークの開発を視野にいれて計画することが重要である．

5.3 フレームワークのドキュメンテーション

フレームワークを用いたシミュレーション開発を促すためには，フレームワークの構造とプラグポイントの場所などを，シミュレーション開発者に伝達しなければならない．フレームワークのアーキテクチャや適用方法を教えるためには，クックブックの形でまとめるのが一般的である [45, 43, 62, 63]．クックブックには複数のレシピが集められており，レシピにはフレームワーク使用方法が非形式的な形でまとめられる．

社会シミュレーションの場合，ドキュメントには個々のコンポーネントのインターフェースや機能だけでなく，コンポーネント同士の理論的な相性も記述する必要がある．異なる分野の理論を組み合わせて社会モデルを作成する際には，そのモデルの前提条件が異なると全体としての理論的一貫性が壊れてしまう可能性があるからである．そのため，設計・実装レベルのドキュメントのほかに，実現されているモデルの論文などへの参照情報なども記述する必要がある．

6 Boxed Economy フレームワークの例

本論文でこれまで述べてきたように，拡張性のある社会シミュレーションを開発するためには，初期段階から再利用を意識した設計を行うことが重要である．ここでは一つのモデルケースとして，現在私たちが取り組んでいるエージェントベース経済シミュレーションのフレームワークである「Boxed Economy フレームワーク」[5] を取り上げ，フレームワークデザインの具体例を示したい [64, 65].

Boxed Economy フレームワークでは，経済主体とその相互作用からなる社会のミクロ構造を土台として定義し，一般経済のシミュレーションの基盤を提供する．柔軟で拡張性の高い設計を目指し，またコンポーネントを作成・蓄積するための環境も同時に提供することにより，経済シミュレーションのオープンソース開発を促進させることを意図している．

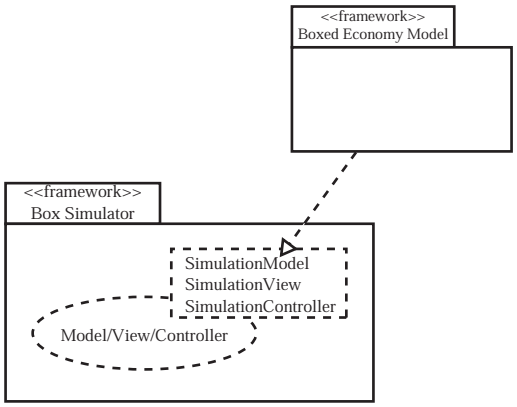

図 5: 二層のフレームワークアーキテクチャ

マルチエージェントモデルのフレームワークのひとつに “Swarm” [66, 67] があるが，これは対象を特定していない一般汎用的なシステムである．これに対し，ここで取り上げるフレームワークは，経済シミュレーションに特化して設計されている．特定のドメインに焦点を絞ることにより，一般汎用的なものに比べより可読性・再利用性・効率性の高いシミュレーション開発を支援することができる．

また実装に Java 言語を用いることにより，プラットフォームに依存しない可搬性のあるシミュレーションが構築できる．開発に際して特殊なシミュレーション開発言語を学習する必要がなくなるため，実装段階ではこのドメイン以外のアプリケーション開発者と協働して開発することが可能となる．

Boxed Economy フレームワークは抽象度の異なる二層のアーキテクチャにより構成されている (図 5)．ひとつはシミュレーションを実行するのに必要なサービスを提供するためのフレームワーク (Box Simulator Framework) であり，もうひとつはその上で実行される具体的な経済モデルを定義するフレームワーク (Boxed Economy Model Framework) である [64].

[5] Boxed Economy フレームワークに関する詳細は，文献 [64] および http:// www. boxed-economy. org/ 参照.

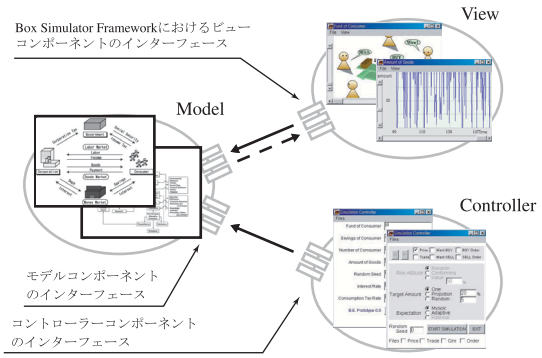

図 6: Box Simulator Framework の構造

6.1 Box Simulator Framework

フレームワーク階層の第一層は，シミュレーション環境のフレームワークである．基本構造は図 6 に示すような Model-View-Controller[43] の構造になっている．すなわち実行するシミュレーションのモデル本体である「モデル」と，結果の表示部分である「ビュー」，そしてシミュレーションを制御する「コントローラー」の 3 つの部分に分割する (図 6) [68].

Box Simulator Framework では，ビューおよびコントローラーを，モデルに合わせて自動的に生成する機構を組み込んでいる．モデル内で変数を定義する際に，その変数を表示するビューを生成するか否かの属性 viewable と，コントローラーで初期設定できるか否かの属性 controllable を設定すると，他のプログラムを変更することなしに，必要な数のビューやコントローラーが自動生成される．これを実現するために，モデル内の各変数を Parameter オブジェクトとして管理し，受け渡すという方法をとる．Parameter オブジェクトは，変数の値や過去の値，単位，データ型などを格納するものである．

モデルとユーザーインターフェースを分離し，フレームワークを階層化することによって，Box Simulator Framework とそれに付随するコンポーネントは，ここでの対象である一般経済モデルだけではなく，市場モデルや生態系モデルなどの様々なシミュレーションに適用することが可能となる．

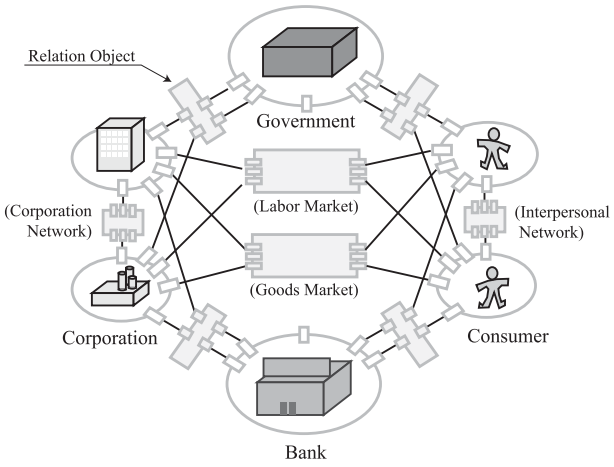

図 7: Boxed Economy Model Framework

6.2 Boxed Economy Model Framework

フレームワーク階層の第二層は，シミュレートする経済モデルを定義するフレームワークである．エージェントベース経済シミュレーションにおけるモデル化の特徴は，経済をその構成主体である消費者・企業・政府などの相互作用として捉える点にある．フレームワークではこれらエージェントの内部構造と関係性をプラグポイントとして公開する（図 7）.

6.3 エージェントの内部構造

エージェントの内部構造では，個々の行動のアルゴリズムは柔軟に変更できるようになっている．社会科学では同一の行動について学問分野や学派によって異なるモデルが提案されていることが多々あり，それらのモデルを交換してシミュレーション結果の相違を比較することが想定される [69]．例えば，消費者の購買については，経済学，マーケティングサイエンス，消費者心理学において異なるモデル化がなされており，それらを目的に応じて入れ換えることができる設計にする必要がある．そこで個々のアルゴリズムを「アルゴリズムオブジェクト」としてエージェントから独立させ，Strategy パターン [35] を用いてアルゴ

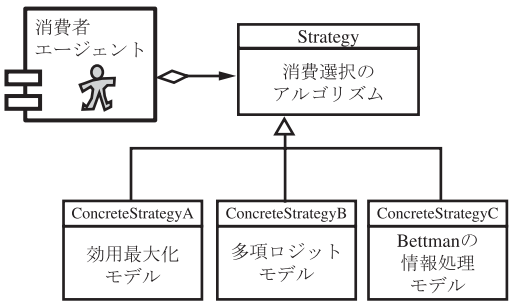

図 8: Strategy パターンによるアルゴリズムが交換可能な設計

リズムの交換を実現する（図 8）．図 8 に示すように，アルゴリズム群をオブジェクトとして定義することにより，そのアルゴリズムの内容 (実装) と，それを実行するオブジェクトとの依存関係を弱め，独立に変更ができるようになる [68].

6.4 エージェント間のデータ交換

経済モデルを拡張する際に，エージェントがやりとりするデータの種類を新たに追加したいということがしばしばある．しかし，エージェントが発信／受信するデータの種類をインタフェースに直接明記すると，データの数が増えるごとにエージェントクラスを継承してオーバーライドすることが必要になる．

そこで，データとエージェントの強い依存関係を弱めるために，複数のデータを「データパックオブジェクト」としてパッケージ化し，それを引数として引き渡すようにインタフェースを定義する（図 9）．データパックオブジェクトにデータの種類を隠蔽してやりとりすることによって，エージェントのインターフェースを変更することなしにデータの追加が実現できる．

エージェントとデータパックが正しく組み合わされていない場合，エージェントとデータの依存関係が不明確になり，モデルの共有化の際に問題が生じる可能性がある．そこで「Abstract Factory パターン」[35] を用いることによって，依存関係にあるエージェントとデータパックオブジェクトの関係ごと変更できるようになる．

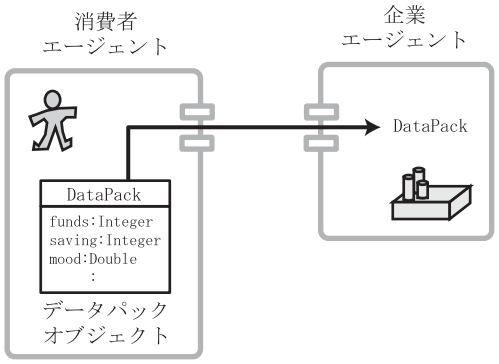

図 9: データの種類の変更に対して柔軟性のある設計

6.5 エージェント間の関係性

エージェント同士の関係性は，固定的な場合もあれば動的に変化する場合もある．また，その関係に間接的に関わるエージェントを追加するということも想定される．このようなエージェント間の関係性に対する柔軟性を確保するために，その関係性を「関係オブジェクト」としてカプセル化する [68]．例えば図 7 に示されている消費者と企業を結ぶ労働市場オブジェクトや商品市場オブジェクトなどが関係オブジェクトである．同様に企業と政府を結ぶ関係なども関係オブジェクトとして扱うことができる．

モデル拡張に際してデータの流れを変更したいことがしばしばあるが，そのとき重要となるのがモデルの柔軟性を高めている関係オブジェクトの存在である．個々のエージェントは関係オブジェクトを介して他のエージェントと繋がっており，関係オブジェクトはデータの流れをエージェントから独立させている．つまり各エージェントは具体的にどのエージェントとデータのやり取りをしているのかを知る必要はない．

例えば図 10 左上のように，労働市場において消費者と企業が労働市場オブジェクト（関係オブジェクト）を介して結び付いており，市場を通じて完全なマッチングが行われるというモデルがあるとしよう．しかしここで新たに，企業や消費者がお互いに不確実な情報しか得ることができず，その不確実性を減らすための就職情報会社が仲介をするというモデルに拡張したいとする．このよ

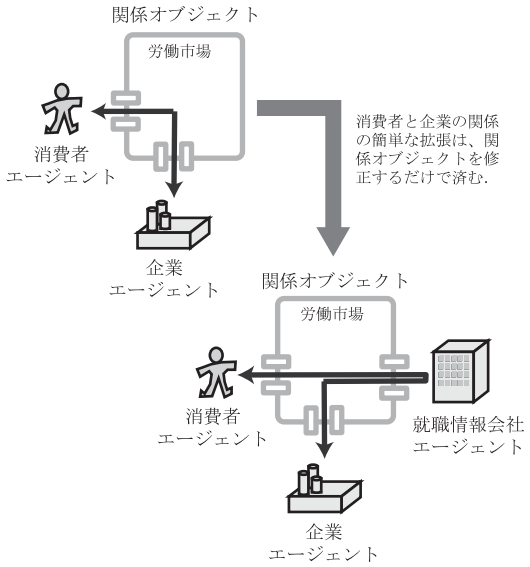

図 10: エージェント間の関係を柔軟にする設計

うな場合，消費者と企業は関係オブジェクトを介して結ばれているため，関係オブジェクトの内部でデータの流れを変更するだけで拡張が可能となる[6] (図 10).

7 社会シミュレーション開発に関する今後の課題

本論文でみてきたような，フレームワークやコンポーネントを用いた社会シミュレーションの開発と共有化のためには，今後議論するべきいくつかの課題がある．以下では，課題とその解決のための関連事項を示す.

7.1 オープンソースに伴うライセンス形式

社会シミュレーション研究の科学的発展のためには，モデルの内部構造や振る舞いの記述のほかにソースコードを公開し，透明性を高めることが求められる．設計情報がモデルの検定のための情報であるのに対し，実装情報はプログラムの検証のためのものであり，シミュレーションの信頼性を示すためには両方の情報公開が不可欠である.

この社会シミュレーションのオープンソース化に伴い，今後議論されるべき問題は，そのプログラムコードの扱いに関するライセンスや契約についてである．この点に関しては，オープンソース・ソフトウェアの先行事例である GNU ソフトウェアや Linux における “GNU General Public Licence (GNU GPL)” [70] や “The FreeBSD Copyright” [71] などが参考になるだろう.

7.2 コンポーネントの信頼性に関する情報の共有化

作成された社会シミュレーションのフレームワークやコンポーネントは，インターネット経由でアクセスできるデータベースに蓄積・共有されると考えられる．その際，設計情報やプログラムソースだけでなく，ピア・レビューによる信頼性の情報なども付加できるようにするとよい.

再利用と品質管理は表裏一体をなしており，再利用するコンポーネントは基本的にゼロ欠陥レベルが要求されるが，実際にはプログラムの検証を完全に保証することは不可能である [72]．そのため開発組織内部でのテストを通過して確証 (certification) されたものを公開し，複数の目による検証を続けるという方法が現実的である．ピア・レビューによる検証結果がデータベースに登録されていることによって，シミュレーション開発者は，コンポーネントの信頼性を知るための二次情報としてその情報を利用することができる．ユーザー側の複数の目による検証については，オープンソースソフトウェアにおけるバザール方式の開発事例が示唆的である [73, 74, 75, 76].

7.3 心理的障壁と教育に関する議論

社会シミュレーションの開発における分析，設計，実装，テストの各段階で再利用が可能であるが，心理的な要因で再利用が効果的に行われないことがある．障害となるのは，Not-Invented-Here (NIH) シンドロームと呼ばれる「ここで開発されたものではない」という意識である [23][72]．多くの開発者がシステムを一から作らなければなら

[6] 関係オブジェクトの内部の変更だけで済まない拡張の場合には，新たにそのためのインタフェースとメソッドを消費者と企業に加える必要がある.

ないと思い込んでいるため，再利用の開発スタイルに慣れる必要がある [77]．そのためには，再利用を効果的に行うための仕組み作りと教育の両方が必要である．

7.4 組織形態に関する議論

再利用やフレームワークの開発を視野にいれた開発のために，どのような組織形態が望ましいかについて研究が必要である．要求分析やモデル作成・利用するのが社会科学者などであり，技術者ではないため，開発における役割分担と全工程の明確化が重要となる．

一般に，再利用を意識した開発への移行には組織に体得された古い方法論がネックになるのだが，幸いなことに社会シミュレーションの分野では，開発スタイルが確立されていない．この初期段階から組織形態やプロセスについて議論を深めていくことが重要である．これまで他のドメインで培われてきた開発スタイルの経験を再利用していくことができるだろう．

7.5 社会シミュレーション開発におけるデザインパターンの探求

社会シミュレーションの開発における様々なノウハウやテクニックは，デザインパターンとして記述していくことができると考えられる．デザインパターンとは設計に関するノウハウを形式的にまとめたものであり，建築や街の設計に関する記述方式として提案された [78, 79]．その後，オブジェクト指向におけるメカニズムの設計に適用されたが [80, 35]，現在では特定ドメインの知識に特化したパターンも発掘されている [81, 82, 83]．

さらに進んで，社会モデリングや社会シミュレーション開発に関するノウハウやテクニックを，デザインパターンとして記述し共有することができると思われる．これにより，経済・社会・市場・生態系など異なる対象を扱うシミュレーション開発プロジェクトの間で技術移転を行うことが可能となるだろう．また，失敗例とその解決策のパターンである「アンチパターン」[77, 84] の共有も社会シミュレーション分野の発展のために有効であろう．

8 さいごに

社会シミュレーションをソフトウェアとして捉え，その設計や開発プロセスを探求することの重要性は未だに低い認識にとどまっている．論文のモデル・数式レベルでの共有化を超え，シミュレーションの設計やプログラムレベルでの共有化を促進することが重要である．

科学的に信頼性が高く大規模な社会シミュレーションを構築するために今考えるべきことは，Booch[53] の言葉を借りるならば，アドホックな「犬小屋の開発」スタイルから「高層ビルの開発」スタイルへの移行である．そのために必要な開発プロジェクトの体質としては，オブジェクト指向モデリングを効果的に使っていること，強力なアーキテクチャビジョンがあること，そしてインクリメンタルな開発サイクルを適用していることなどが指摘されている [53]．これらは社会シミュレーション開発を行う上で，事後的ではなく最初から備えているべき特徴である．今後，本論文で扱ったようなテーマを「社会シミュレーション工学」(Social Simulation Engineering) として研究し，実践を通じて深めていくことが大切であろう．

謝辞

本研究の一部は，文部省科学研究費補助金，およびフジタ未来経営研究所のサポートにより遂行されたものである．両サポートに感謝したい．

※本論文は，2000 年 1 月に執筆されたものをワーキングペーパーとして出版したものである．
※本論文で取り上げられているモデル等は，執筆当時のものであり，その詳細などは今後変更されることがある．詳細・最新情報は，http://www.boxed-economy.org/を参照のこと．

参考文献

[1] N. Gilbert, editor. *Artificial societies: The computer simulation of social life*. UCL Press, 1995.

[2] R. Conte, R Hegselmann, and P Terna, editors. *Simulating Social Phenomena*. Springer-Verlag, 1997.

[3] K.G. Troitzsch, U. Mueller, G.N. Gilbert, and J.E. Doran, editors. *Social Science Microsimulation*. SpringerVerlag, 1996.

[4] J.M. Epstein and R. Axtell. *Growing Artificial Societies: Social science from The Bottom Up*. The MIT Press, 1996.

[5] C. Bruun. Agent-based keynesian economics. In R. Conte, R. Hegselmann, and P. Terna, editors, *Simulating social phenomena*, pp. 279 – 285. Springer-Verlag, 1997.

[6] N. Basu, R. J. Pryor, T. Quint, and T. Arnold. Technical report.

[7] R.J. Pryor, N. Basu, and T. Quint. Development of aspen: A microanalytic simulation model of the economy. Sandia report, Sandia National Laboratories, February 1996.

[8] N. Basu and R. J. Pryor. Growing a market economy. Sandia report, Sandia National Laboratories, September 1997.

[9] N. Basu, R. J. Pryor, and T. Quint. Aspen: A microsimulation model of the economy. *Computational Economics*, Vol. 12, pp. 223–241, 1998.

[10] G.R. Palmer, B.W. Arthur, J.H. Holland, B. LeBaron, and P. Tayler. Artificial economic life:a simple model of a stockmarket. *Physica D*, Vol. 75, , 1994.

[11] K. Izumi and T. Okatsu. An artificial market analysis of exchange rate dynamics. In L.J. Fogel, P.J. Angeline, and T. Back, editors, *Evolutionary Programming V*, pp. 27–36. The MIT Press, 1996.

[12] K. Izumi and K. Ueda. Emergent phenomena in a foreign exchange market: Analysis based on an artificial market approarch. In C. Adami, C. Taylor, and R.K. Belew, editors, *Artificial Life VI:Proceedings of the Sixth International Conference on Artificial Life*, pp. 398–402. The MIT Press, 1998.

[13] 井庭崇, 竹中平蔵. マルチエージェントモデルによるバーチャルマーケットと消費者行動. 第二回進化経済学会論集, pp. 155 – 163, 1997.

[14] T. Iba, H. Takenaka, and Y. Takefuji. Agent-based artificial market model: The case study of de-facto standard on vcr market. In *Proceedings, Third International Conference on Multi-Agent Systems*, pp. 437 – 438, 1998.

[15] T. Iba, H. Takenaka, and Y. Takefuji. An agent-based simulation of bubbles and crashes in economy. In *Proceedings of the International ICSC Congress on Computational Intelligence Methods and Applications CIMA'99*, pp. 667 – 673, 1999.

[16] T. Iba, H. Mizoe, H. Takenaka, and Y. Takefuji. Agent-based social simulation as novel method for economics. In *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, Vol. 3, pp. 58 – 65, 1999.

[17] N. Gilbert and K. G. Troitzsch. *Simulation for the Social Scientist*. Open University Press, 1999.

[18] T.J. McCabe. A complexity measurement. 1976. SE-2.

[19] J. A. McCall, P. K. Richards, and G. F. Walters. Factors in software quality. In *Concepts and Definitions of Software Quality*, Vol. I-III. Rome Air Development Center, 1977.

[20] C. V. et al. Ramamoorthy. Techniques in software quality assurance. In *Proceedings of the German Chapter of the ACM*, Vol. 9, 1982.

[21] N. Fenton. *Software Metrics — A Rigorous Approach*. Chapman & Hall, 1991.

[22] C. Jones. *Software Quality : Analysis and Guidelines for Success*. International Thomson Computer Press, 1996.

[23] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.

[24] P. K. Davis. Generalizing concepts of verification, validation and accreditation (vv&a) for military simulation. Technical report r-4249-acq, RAND, 1992.

[25] J. S. Carson. Convincing users of model's validity is challenging aspect of modeler's job. Vol. 18, No. 6, pp. 74–85, 1986.

[26] S. Robinson. Simulation verification, validation and confidence: A tutorial. Vol. 16, No. 2, pp. 63 – 69, 1997.

[27] K. Kaneko and Tsuda I. Constructive complexity and artificial reality: an introduction. Vol. 75, , 1994.

[28] 井庭崇, 福原義久. 複雑系入門：知のフロンティアへの冒険. NTT 出版, 1998.

[29] F. P. Brooks. *The Mythical man-month : essays on software engineering*. Addison-Wesley, anniversary edition edition, 1995.

[30] I. Jacobson, M. Griss, and P. Jonsson. *Software Reuse : Architecture, Process and Organization for Business Success*. ACM Press, 1997.

[31] McIlroy D. Mass produced software components. In *1968 NATO Conf. on Software Engineering*, pp. 138 – 155, 1969.

[32] Success factors of systematic reuse. Vol. 11, No. 5, pp. 15 – 19, 1994.

[33] Schaefer W., R. Prieto-Daioz, and M. Matsumoto, editors. *Software Reusability*. Ellis Horwood, 1994.

[34] E.-A. Karlsson, editor. *Software Reuse: A Holistic Approach*. Wiley, 1995.

[35] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[36] Object-oriented application frameworks. Vol. 40, No. 10, 1997.

[37] C. Jones. *Patterns of Software Systems Failure and Success*. International Thomson Computer Press, 1995.

[38] O-J. Dahl and K. Nygaard. Simula — an algol-based simulation language. Vol. 9, pp. 671 – 678, 1996.

[39] 中谷多哉子, 青山幹雄, 佐藤啓太（編）. ソフトウェアパターン. 共立出版, 1999.

[40] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, editors. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. Wiley, 1999.

[41] N. Warren and P. Bishop. *Java in Practice : Design Styles and Idioms for Effective Java*. Addison-Wesley, 1999.

[42] J. Langr. *Essential Java Style : Patterns for Implementation*. ————, 1999.

[43] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. Vol. 1, No. 3, pp. 26 – 49, 1988.

[44] K. J. Schmucker. *Object-Oriented Programming for the Macintosh*. Hayden Book Company, 1986.

[45] Apple Computer. *MacApp Programmer's Guide*, 1986.

[46] D. A. Wilson, L.S. Rosenstein, and D. Shafer. *Programming with MacApp*. Addison-Wesley, 1990.

[47] A. Weinand, E. Gamma, and Marty R. Et++ — an object-oriented application framework in c++. Vol. 23, No. 11, 1988.

[48] A. Weinand, E. Gamma, and R. Marty. Design and implementation of et++, a seamless object-oriented application framework. Vol. 10, No. 2.

[49] IBM Corporation. *IBM SanFrancisco : Concepts and Facilities*. International Business Machines Corporation, 1997. Version 1, Release 2.

[50] Campos A. M. C. and D. R. C. Hill. An agent-based framework for visual-interactive ecosystem simulations. Vol. 15, No. 4, pp. 139 – 152, 1998.

[51] E. Bruderer and M. Maiers. From the margin to the mainstream: An agenda for computer simulation in the social sciences. In Hegselmann R. Conte, R. and P. Terna, editors, *Simulating Social Phenomena*, pp. 89–95. Springer-Verlag, 1997.

[52] R. E. Johnson and B. Foote. Designing reusable classes. Vol. 1, No. 5, pp. 22 – 35, 1988.

[53] G. Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1996.

[54] R. E. Johnson, 中村宏明, 中山裕子, 吉田和樹. パターンとフレームワーク. , 共立出版, 1999.

[55] R. Wirfs-Brock and R. Johnson. Surveying current research in object-oriented design. Vol. 33, No. 9, pp. 104 – 124, 1990.

[56] D. Roberts and R. Johnson. Evolving frameworks: A pattern language for developing object-oriented frameworks. In *Proceedings of the Third Conference on Pattern Languages and Programming*, 1996.

[57] G. G. Miller, J. McGregor, and M. L. Major. Capturing framework requirements. Building Application Frameworks: Object-Oriented Foundations of Framework Design, 1999.

[58] D. F. D'Souza and A. C. Wills. *Catalysis : Component and Framework-Based Development*. Addison-Wesley, 1998.

[59] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML*. Addison-Wesley, 1999.

[60] H. A. Schmid. Framework design by systematic generalization. Building Application Frameworks: Object-Oriented Foundations of Framework Design, 1999.

[61] W. Pree. Hot-spot-driven development. Building Application Frameworks: Object-Oriented Foundations of Framework Design, 1999.

[62] R. Johnson. Documenting frameworks using patterns. In *OOPSLA'92 Proceedings*, pp. 63 – 76. ACM Press.

[63] Inc. ParcPlace Systems. *VisualWorks Cookbook*, 1994.

[64] T. Iba, M. Hirokane, Y. Takabe, H. Takenaka, and Y. Takefuji. Boxed economy model: Fundamental concepts and perspectives. In *Proceedings of Computational Intelligence in Economics and Finance*, 2000.

[65] 井庭崇, 廣兼賢治, 吉川知宏, 武藤佳恭, 竹中平蔵. Boxed economy model による政策分析手法の提案. 政策分析ネットワーク第１回年次研究大会 政策メッセ 99 研究発表要旨集, p. 5, 1999.

[66] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system:a toolkit for building multi-agent simulations. http:// www.santafe.edu/ projects/ swarm/ overview/ overview.html, 1996.

[67] C. Langton, R. Burkhart, I. Lee, M. Daniels, and A. Lancaster. The swarm simulation system. http:// www.santafe.edu/ projects/ swarm, 1998.

[68] 岩村拓哉, 廣兼賢治, 井庭崇, 竹中平蔵, 武藤佳恭. エージェントベース経済シミュレーションのためのフレームワークデザイン. 第 8 回マルチエージェントと協調計算ワークショップ, 1999.

[69] T. Iba, M. Hirokane, H. Kawakami, Takefuji Y., and H. Takenaka. Exploratory model building: Toward agent-based economics. 第四回進化経済学会論集, 2000.

[70] Inc. Free Software Foundation. Gnu general public license. http:// www.gnu.org/ copyleft/ gpl.html, 1991. Version 2.

[71] Inc. FreeBSD. Freebsd copyright. http:// www.freebsd.org/ copyright/ freebsd-license.html, 1994.

[72] I. Jacobson. *Object-Oriented Software Engineering : A Use Case Driven Approach*. Addsion-Wesley, 1992.

[73] E. S. Raymond. The cathedral and the bazaar. http:// www.post1.com/ home/ hiyori13/ freeware/ cathedral.html.

[74] E. S. Raymond. Homesteading the noosphere. http:// www.post1.com/ home/ hiyori13/ freeware/ noosphere.html.

[75] E. S. Raymond. The magic cauldron. http:// www.post1.com/ home/ hiyori13/ freeware/ magicpot.html.

[76] E. S. Raymond. *The Cathedral and the Bazaar : Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly, 1999.

[77] W. Brown, R. Malveau, H. McCormick III, and Mowbray T. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, 1998.

[78] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language*. Oxford University Press, 1977.

[79] C Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[80] E. Gamma. *Objektorientierte Software-Entwicklung am Beispiel von ET++: Klassenbibliothek, Werkzeuge, Design*. PhD thesis, Universität Zürich.

[81] J.O. Coplien and D.C. Schmidt, editors. *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[82] J. M. Vlissides, J. O. Coplien, and N. L. Kerth, editors. *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.

[83] R.C. Martin, D. Riehle, and F. Buschmann, editors. *Pattern Languages of Program Design 3*. Addison-Wesley, 1998.

[84] W. Brown, H. McCormick III, and S. W. Tohmas. *AntiPatterns and Patterns: in Software Configuration Management*. John Wiley & Sons, 1999.