第7章　提案システムによる既存モデルの再現

本論文の提案 (モデル・フレームワーク、シミュレーション・プラットフォーム、モデル・パターン) の適用可能性を明らかにするために、ここでは、代表的な既存モデルを再現することにしたい。取り上げるモデルは、成長するネットワークモデル、繰り返しの囚人のジレンマモデル、貨幣の自生と自壊モデル、Sugarscape モデル、人工株式市場モデルの 5 つである。

7.1　成長するネットワークのモデル

近年、ノードの生成やリンクの生成・組み替え・除去など、ネットワークのトポロジーに影響するプロセスを扱うことのできる「成長するネットワーク」の理論研究が進んでいる。Barabási (2002) の指摘するように、現実世界に存在するネットワークは、そのほとんどが「成長する」という特徴をもっている。このような成長するネットワークについての理解を深めるためには、シミュレーションによるアプローチが不可欠であるが、Agent(ノード) や Relation(リンク) が動的に生成・削除できる BEFM / BESP は、このようなシミュレーションに適しているといえる。

ここでは、成長するネットワークの研究における代表的なモデルをいくつか再現することにしたい。まず最初に、ノード数が一定のままリンクが生成される「ランダムリンクモデル」を作成し、次に、新しいノードを生成してランダムにリンクを張っていく「ランダム選択成長モデル」、そして最後に、優先的選択によってリンクを張っていく「優先的選択成長モデル」を作成する。

7.1.1　ランダムリンクモデル

まず最初に作成するランダムリンクモデルでは、最初多くの孤立したノードが存在する。時間の経過とともに、このノード同士をランダムにつないでいく (図 7.1)。

このモデルの全体像は、図 7.2 のようになる。ここでは、2 人の Node エージェントを選んでリンクを張る処理を行う Organizer エージェントを用意する。シミュレーションの流れは、次のようになる (図 7.3)。Organizer エージェントは、TimeEvent を RandomNetworkingBehavior (図 7.4) で受け取り、Node エージェントの中からラン

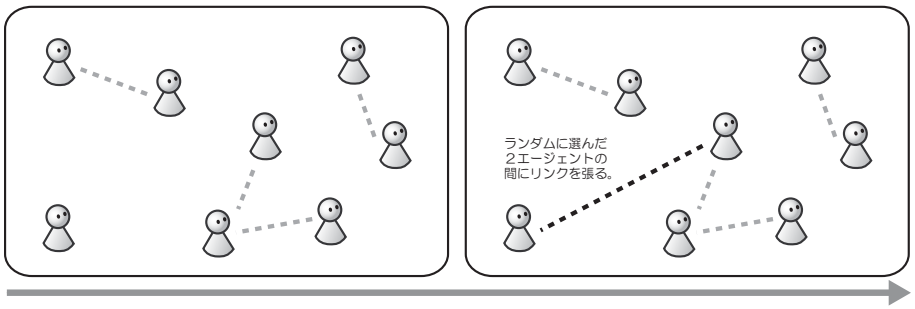


図 7.1: ランダムリンクモデルのイメージ

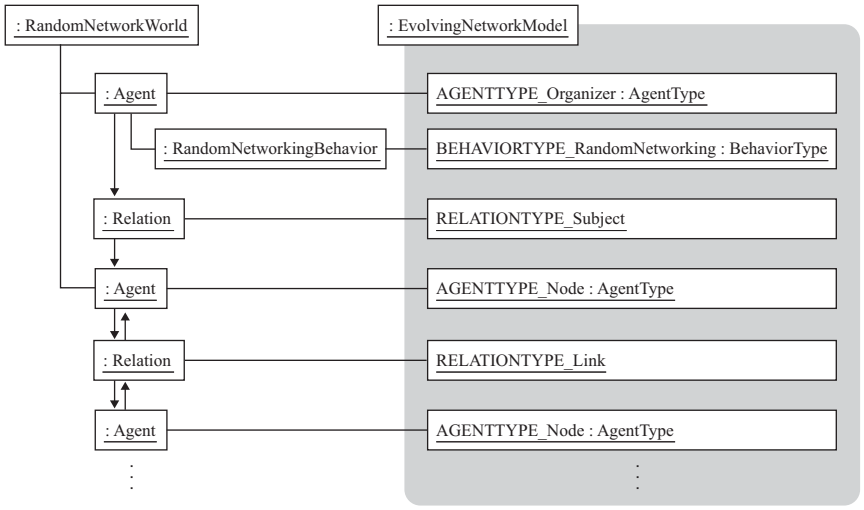


図 7.2: ランダムネットワークモデルの全体像

ダムに 2 人を選び、その 2 人の間に関係を結ぶ。

シミュレーション結果は、図 7.5, 7.6 のようになる。最初は、ノードとノードがつながったペアが生まれるだけだが、しばらくすると、ペアとペアがつながってクラスターが形成される。初期のクラスターは小規模なものだが、ある時、クラスター同士が結びついて巨大クラスターが出現する。その時々の最大クラスターに属するノードの数を時系列で描くと図 7.7 のようになる。

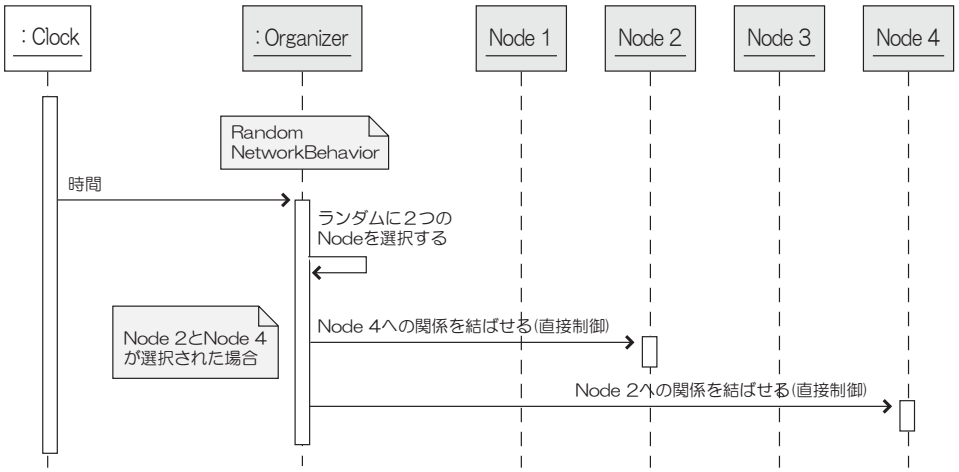
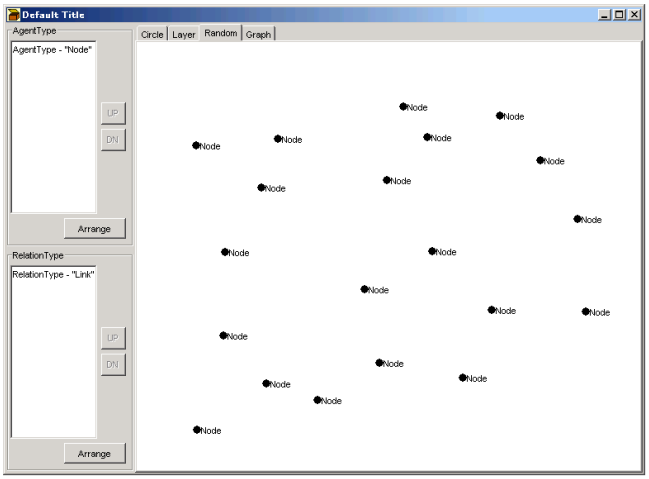


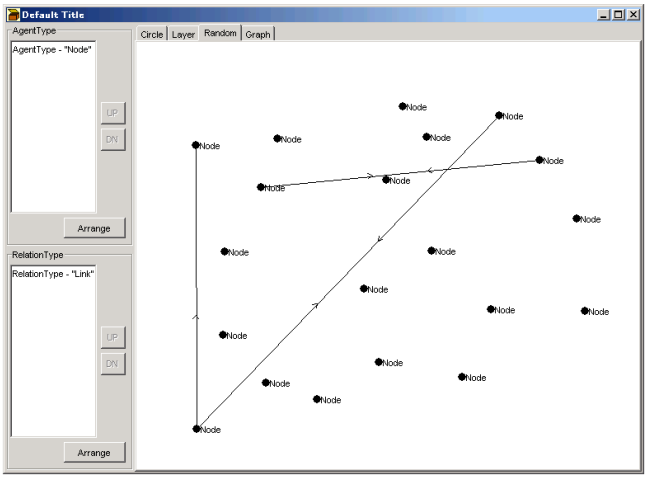
図 7.3: ランダムリンクモデルのシーケンス図

図 7.4: ランダムリンクモデル: RandomNetworkBehavior

step 0



step 3



step 12

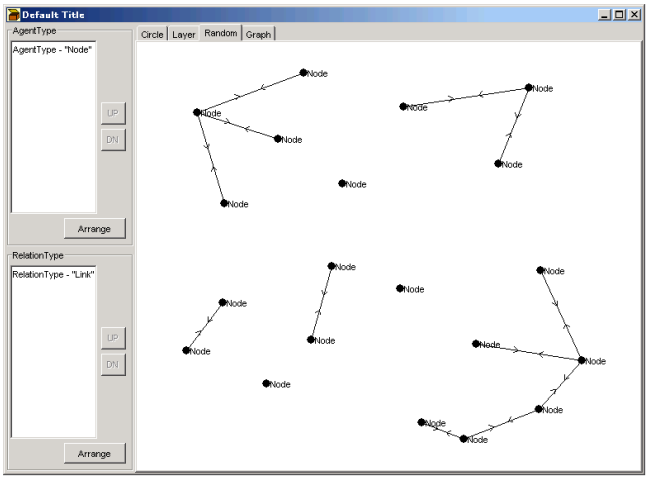
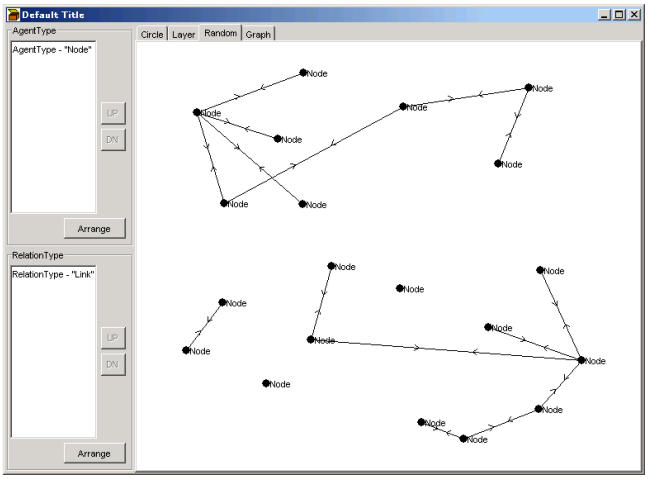
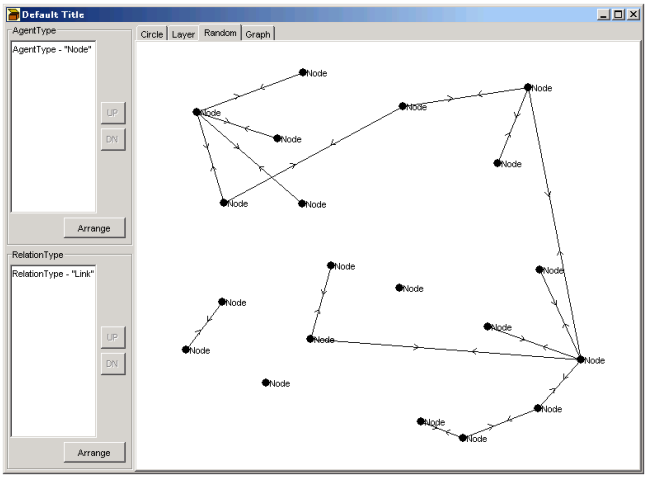


図 7.5: ランダムリンクモデルのシミュレーション結果 (1)



step 15



step 16

図 7.6: ランダムリンクモデルのシミュレーション結果 (2)

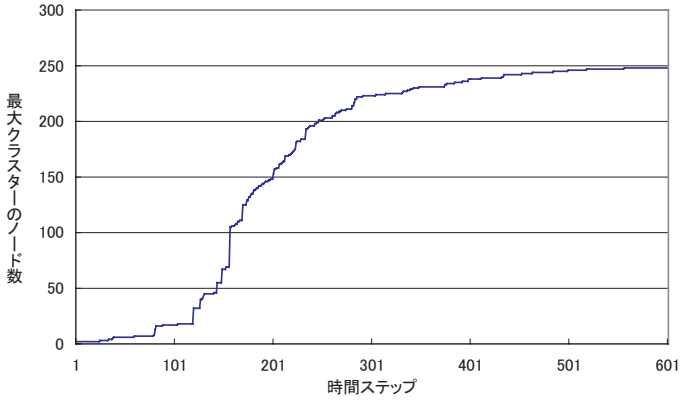


図 7.7: ランダムリンクモデルにおける最大クラスターのノード数の推移

7.1.2　ランダム選択成長モデル

成長するネットワークは、小さなネットワークから始めて、徐々にノードを追加していくというモデルで表現できる。ここではまず最初に、ランダム選択成長モデルを考えることにしよう。ランダム選択成長モデルでは、新しく追加されたノードは、ランダムに選ばれた 2 つのノードとリンクが張られる (図 7.8)。

図 7.9 は、このモデルの全体像である。新しい Node エージェントを追加し、既存の Node エージェントとリンクを張る処理を行う Organizer エージェントを用意するする。このシミュレーションの流れは、次のようになる (図 7.10)。TimeEvent を RandomAttachmentBehavior で受け取った Organizer エージェントは、Node エージェントを 1 人生成し、世界に追加する (図 7.11)。そして、既に存在する Node エージェントの中からランダムに 2 人を選び、さきほど追加した Node エージェントとの間に関係を結ぶ。

シミュレーションの結果は、図 7.12 のようになる[61]。このネットワークの各ノードのリンク数とその順位を両対数グラフにプロットすると、図 7.13 のようになる。このグラフから、度数分布が指数関数的な減少を示していることがわかる。

近年、友人関係や経済ネットワーク、ワールド・ワイド・ウェブ (WWW) などのような「成長するネットワーク」では、従来考えられてきたような均質的なネットワーク構造ではなく、多数のリンクをもつ「ハブ」が少数存在するという構造になっていることがわかっている。そして、そのリンクのつながれ方にはべき乗分布がみられる[62]。べき乗分布に従うネットワークでは、普通は少ない数のリンクをもっているが、ごく一部のノードは非常に多くのリンクをもっている。通常、自然界で起こる現象では、大きな変動の頻度は指数関数的に急激に減少するが、べき乗法則に従う場合には、このような「稀有な事象」も共存することになる。べき乗分布は両対数グラフで表すと、図 7.14 のように直線になる。このシミュレーションからわかることは、ランダム選択成長モデルでは、現実世界におけるネットワークは説明できないということである。

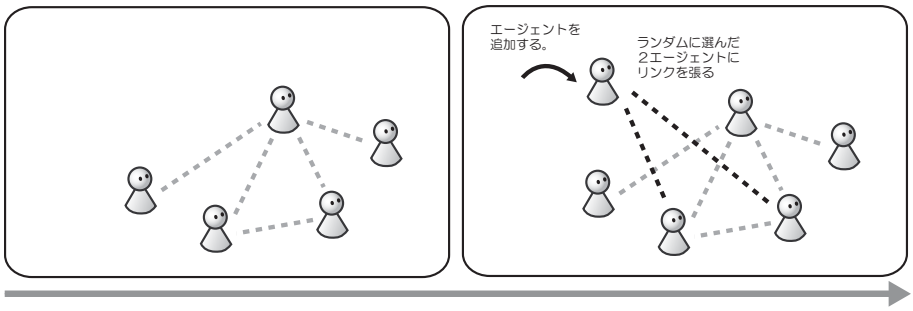


図 7.8: ランダム選択成長モデルのイメージ

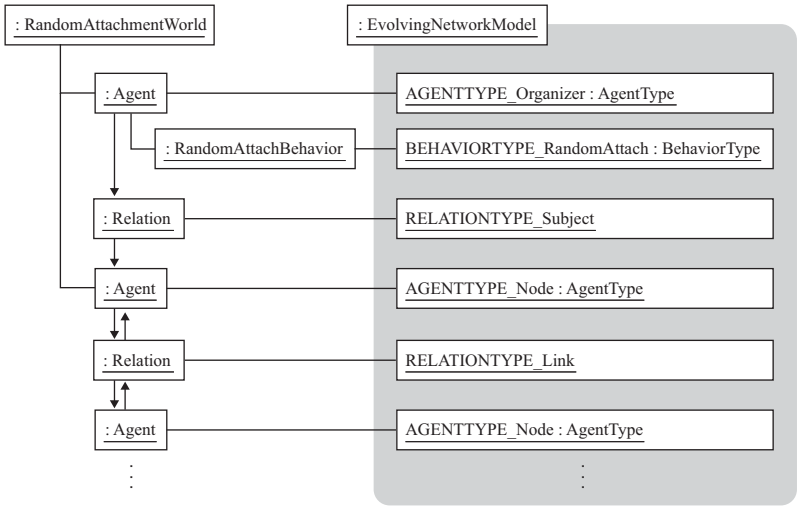


図 7.9: ランダム選択成長モデルの全体像

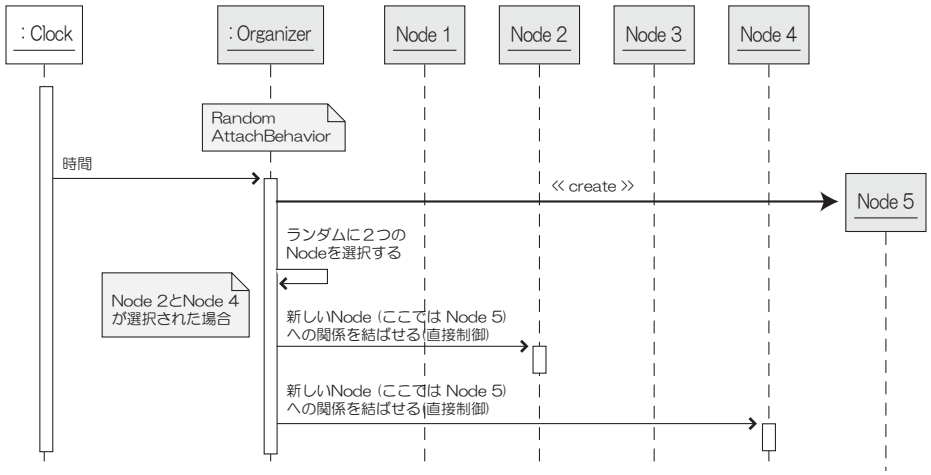


図 7.10: ランダム選択成長モデルのシーケンス図

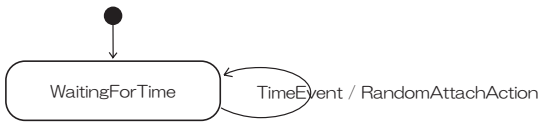


図 7.11: ランダム選択成長モデル: RandomAttachBehavior

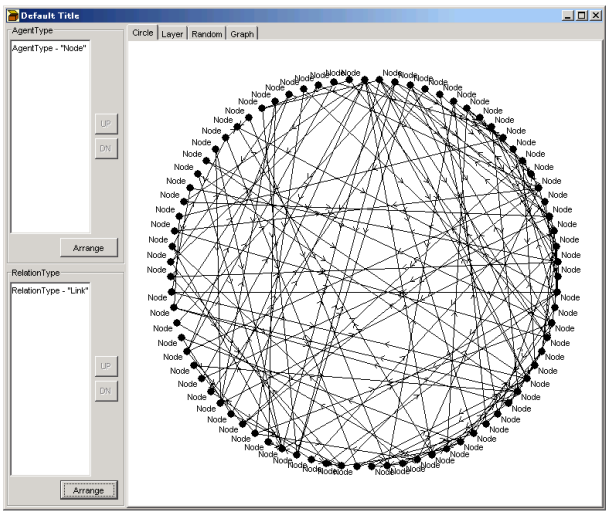


図 7.12: ランダム選択成長モデル: 形成されたネットワーク

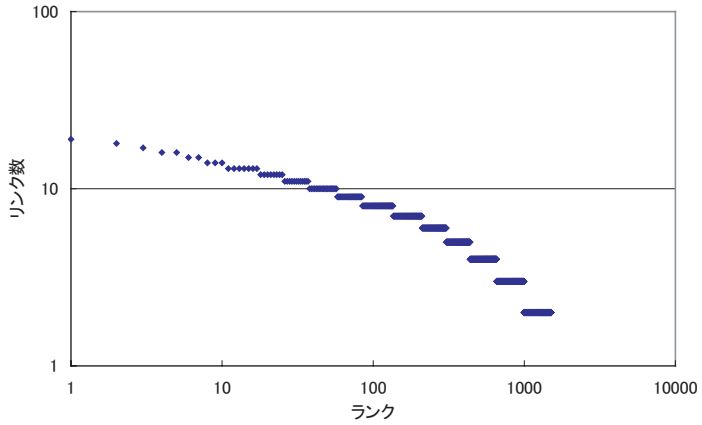


図 7.13: ランダム選択成長モデル: リンク数と順位の関係 (両対数グラフ)

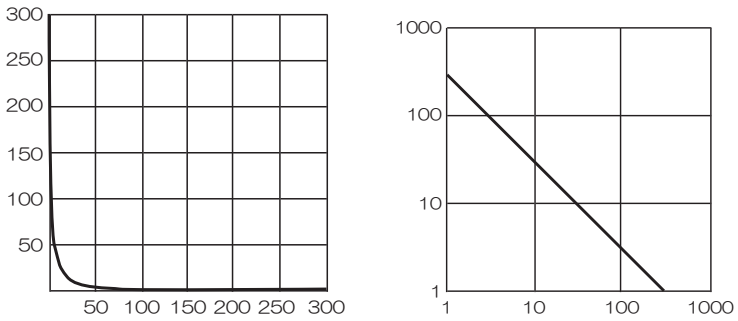


図 7.14: 線形グラフと両対数グラフにおけるべき乗分布

7.1.3　優先的選択成長モデル (スケールフリー・モデル)

現実世界でみられるようなべき乗の結合分布をもつネットワークは、どのようにすれば形成することができるのだろうか。この問題に取り組むには、「現実のネットワークでは、リンクがランダムに張られたりはしない」(Barabási, 2002) という点に注目することが重要となる。現実には、ウェブページでも友人関係でも、リンク数が多いノードほどますます多くのリンクを獲得するということが起こっている。実はこの原理を用いることで、現実に近い成長するネットワークを作成できることがわかっている。

Barabási et al. (1999) は、「成長」と「優先的選択」という 2 つの基本原理によって、結合分布がべき乗になるネットワークを生成できることを解析的に示した。これらの原理は、どちらか片方だけではべき乗の結合分布を生み出さないため、両者とも不可欠であることがわかっている。「成長」とは、新しいノードを加えるということであり、「優先的選択」とは、新しいノードを追加する際に、多くのリンクを持っているノードを優先的に選択することである (図 7.15)。新しいノードが、k 個のリンクをもつノードにリンクを張る確率は、次の式で与えられる。

$$prob = \frac{k}{\sum_i k_i}$$

図 7.16 は、このモデルの全体像である。新しい Node エージェントを生成・追加して既存の Node エージェントとリンクを張る処理を行う Organizer エージェントを用意する。このシミュレーションの流れは、次のようになる (図 7.17)。Organizer エージェントは、TimeEvent を PreferentialAttachBehavior (図 7.18) で受け取り、Node エージェントを 1 人生成し、世界に追加する。そして、既に存在する Node エージェントのもつリンク数を調べ、もっているリンク数に比例した確率で Node エージェントを 2 人選ぶ。そしてこの 2 人の Node エージェントと、新しく追加した Node エージェントの間にリンクを結ぶ。

このシミュレーションの結果は、図 7.19 のようになる。非常に多くのリンクをもつ

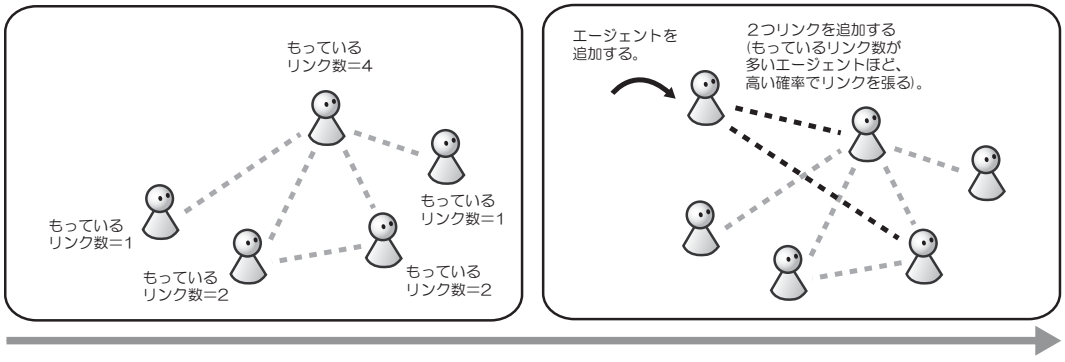


図 7.15: 優先的選択成長モデルのイメージ

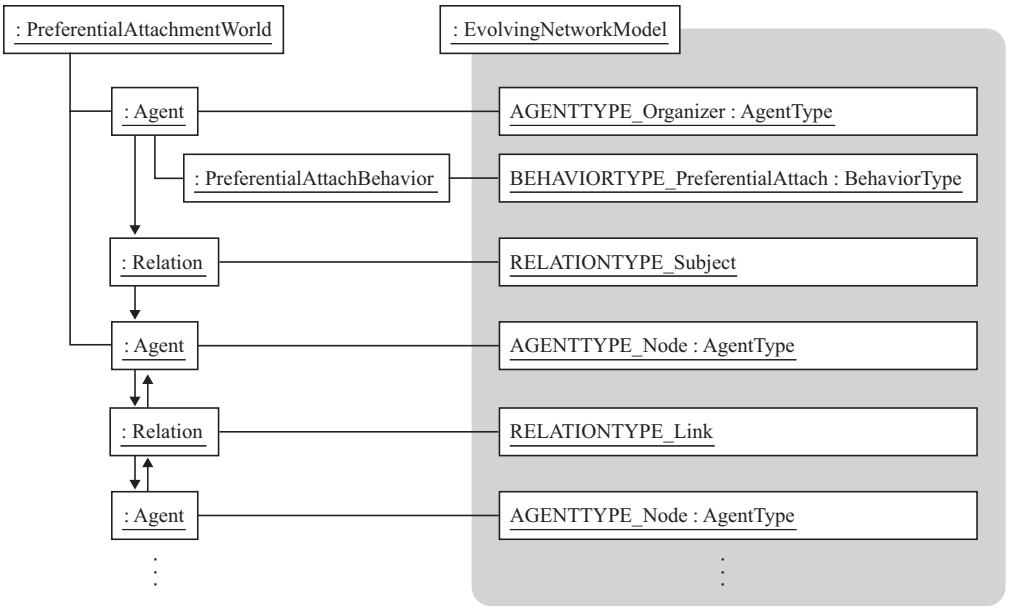


図 7.16: 優先的選択成長モデルの全体像

ハブが出現していることがわかる。この結果を両対数グラフで表すと、図 7.20 のようにベキ乗になっていることが確認できる(63)。このモデルは、最初にスケールフリーのべき法則を説明したため、「スケールフリー・モデル」と呼ばれるようになった。このモデルによって、「まず第一に、ベキ法則によってハブの存在に正当性が与えられた。次に、スケールフリー・モデルによって、現実のネットワークに見られるベキ法則が、数学的基礎をもつ概念上の進歩に格上げされた。さらには、進化するネットワークという洗練された理論に支えられて、スケーリング指数やネットワークのダイナミクスが精密に予測できるようになった。」(Barabási, 2002, 邦訳 p.135) といわれている。

また、社会・経済シミュレーションの研究における意義も大きい。というのは、社会・経済のシミュレーションを行う場合には、何らかの社会ネットワークを想定する必要があるが、その際に現実と同型のスケールフリー・ネットワークを用いることで、より現実に近いモデルになるからである。また、動的に成長する社会ネットワークにおける社会・経済現象を分析することも可能になる。

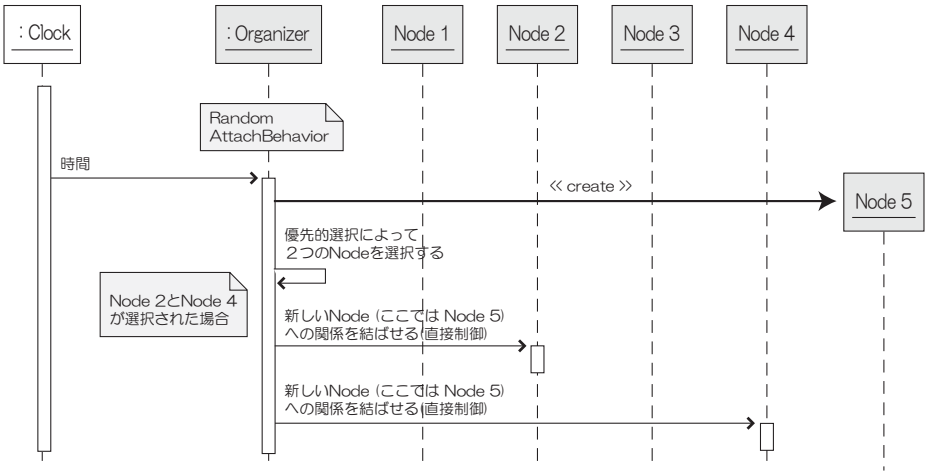


図 7.17: 優先的選択成長モデルのシーケン ス図

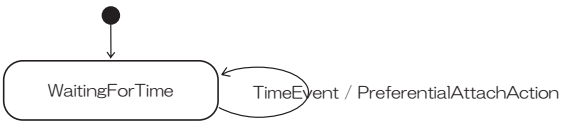


図 7.18: 優先的選択成長モデル: PreferentialAttachBehavior

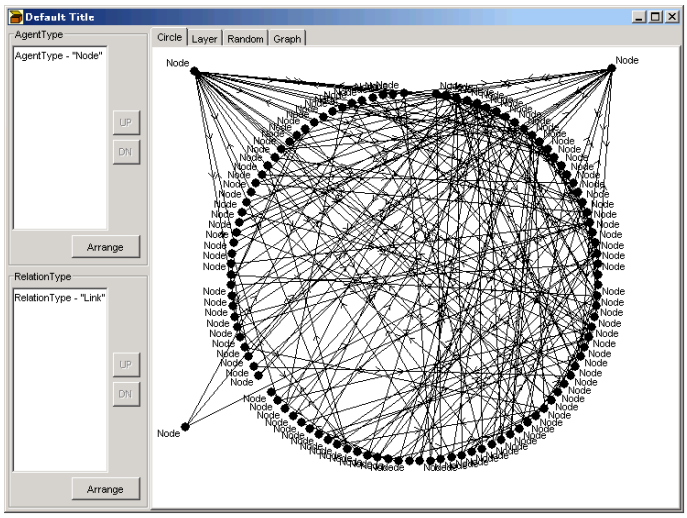


図 7.19: 優先的選択成長モデル: 形成されたネットワーク

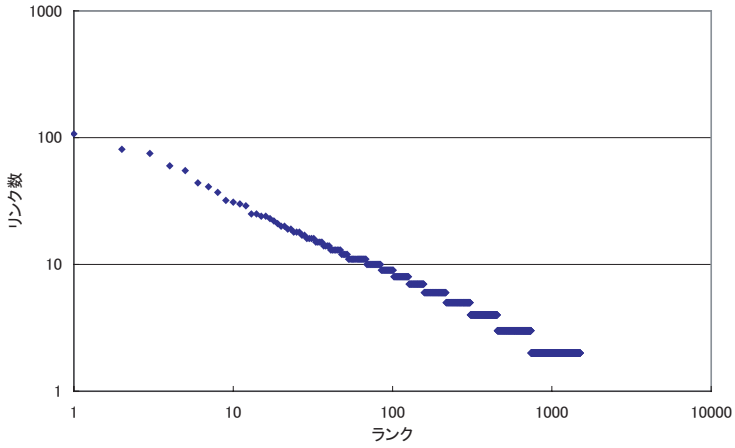


図 7.20: 優先的選択成長モデル: リンク数と順位の関係 (両対数グラフ)

7.2　繰り返し囚人のジレンマモデル

「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の中で、どのように協調が形成されるのかを調べる枠組みとしてしばしば用いられている。このジレンマは、1950 年頃、心理学研究のなかで M.Flood と M.Dresher によって提唱されたものであり、A.W.Tucker が「囚人のジレンマ」というストーリー仕立てで広めてからは、政治学や経済学、社会学において、ジレンマの社会的モデルとして頻繁に用いられてきた。二人のプレイヤーが、それぞれ独立に、協調 (Cooperation) か裏切り (Defection) かのどちらかの行動をとり、自分の選択と相手の選択の組合せによって、得られる利得が異なるようになっている。

囚人のジレンマは一回限りのゲームであるが、これを反復的に行うというゲームの考察も行われており、これを「繰り返し囚人のジレンマ」という(64)。この繰り返しの囚人のジレンマは、あらかじめわかっている有限回の対戦であれば、裏切る方がより高い利得を得られることがわかっている。しかし、いつまで続くかわからない場合には、必ずしもそのような結果にはならず、万能の戦略がないといわれている。

ここでいう「戦略」とは、各回の選択のことではなく、過去の手を踏まえて自分の手を決めるための行動決定規則である。同じ戦略でもこれまでの経緯によって (選択の履歴によって)、協調することもあれば裏切ることもある。自分の状態によって反応が異なるという意味で、このモデルは、本論文でいうところの広義の複雑系のモデルになっている。また、戦略を変更するということが起こるならば、それは狭義の複雑系と捉えることができる。

7.2.1　コンテスト・シミュレーション

ここでは、まず最初に、広義の複雑系のモデルとして、コンテスト・シミュレーションを行う。戦略の状態の変化は、BEFM の Behavior の状態変化でモデル化する。

モデルの基本的な枠組みを説明すると、シミュレーションには、1 人の Referee エージェントと、複数の Player エージェントが登場する (図 7.21)。この Referee エージェントは、コンテストを仕切る役割を担っている。Referee エージェントは、Player エージェント同士が総当りになるように、順番に 2 人ずつ Player エージェントを呼び出し、試合を行わせる。このとき、試合では 200 回の対戦が行われる。各対戦における得点は、両者が協調すれば 3 点ずつ、裏切りあえば 1 点ずつ、片方だけ協調し他方が裏切れば、それぞれ 0 点と 5 点となる。試合が終了すると、最終的な得点が記録される。こうして試合が次つぎに行われ、総当りが実現したら、そのコンテストが終了し、Referee エージェントは結果を公表する。コンテストと試合と対戦の関係は、図 7.22 のようになる。

Referee エージェントは、コンテスト全体を管理する ManageContestBehavior と、

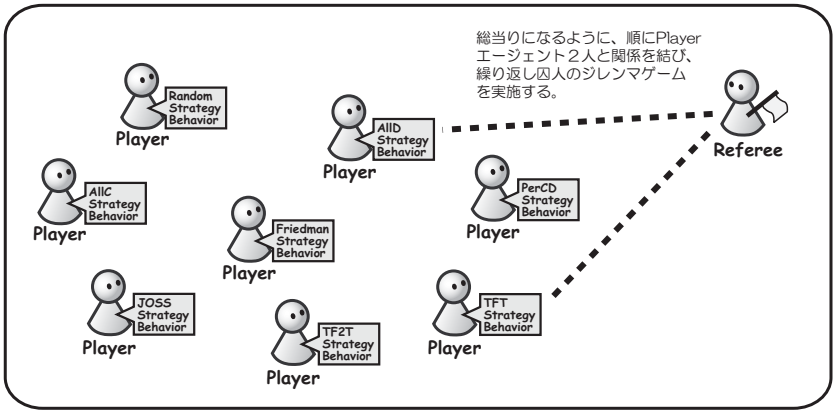


図 7.21: コンテスト・シミュレーションのイメージ

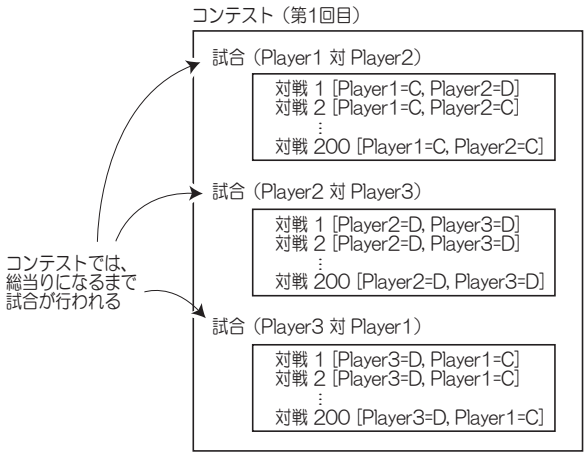


図 7.22: コンテストと試合と対戦の関係

各試合を取り仕切る ConductMatchBehavior をもっている。Player エージェントは、Referee エージェントとコミュニケーションをとるための PlayBehavior と、戦略行動をもっている (図 7.23)。対戦では、各 Player エージェントは、前回の相手の手のみが知らされ(65)、自分の次の手を決めていくことになる。戦略行動は、戦略の状態変化を Behavior の状態変化で表現したものであり、次のような種類がある。

ALL-C 相手の手に関係なく、必ず協調する (ALLCStrategyBehavior; 図 7.25)。

ALL-D 相手の手に関係なく、必ず裏切る (ALLDStrategyBehavior; 図 7.26)。

RANDOM 相手の手に関係なく、協調と裏切りをランダムに選択する (RandomStrategyBehavior; 図 7.27)。

TFT 最初は協調し、次からは相手が前回とった行動を真似する (TFTStrategyBehavior; 図 7.28)。

図 7.23: 戦略を行動として表現する

TF2T 最初は協調し、2 回連続して相手が裏切ったときに、裏切る (TF2TStrategyBehavior; 図 7.29)。

FRIEDMAN 最初は協調し、相手が裏切らないかぎり協調を続ける。相手が一度でも裏切ると、それ以降はずっと裏切り続ける (FRIEDMANStrategyBehavior; 図 7.30)。

JOSS TFT (しっぺ返し) と同様に、最初は協調し、相手に裏切られると裏切り返す。相手が協調した場合には、9 割協調して、1 割裏切る (JOSSStrategyBehavior; 図 7.31)。

PER-CD 協調、裏切り、協調、裏切り …· を繰り返す (PERCDStrategyBehavior; 図 7.32)。

PER-CCD 協調、協調、裏切り、協調、協調、裏切り …· を繰り返す (PERCCDStrategyBehavior; 図 7.33)。

モデルの全体像は、図 7.24 のようになり、シミュレーションの流れは、次のようになる (図 7.34)。最初に TimeEvent を受け取るのは、Priority が高く設定されている Referee エージェントである。Referee エージェントは、ManageContestBehavior(図 7.35) で TimeEvent を受け取り、総当たりの対戦表を作成する。その後、ConductMatchBehavior(図 7.36) に対戦組合せの情報を 1 組分ずつ渡し、その試合を開始する。

Referee エージェントは、今回対戦する 2 人の Player エージェントに、まず初回の手を尋ねる。Player エージェントは PlayBehavior(図 7.37) で連絡を受け取り、それぞれのもつ StrategyBehavior に初回の手を出すよう連絡する。StrategyBehavior は初回の手を返答し、PlayBehavior を通じて、Referee エージェントに伝える。Referee エージェントは、ConductMatchBehavior で返答を受け取り、2 人の Player エージェントの手を照合して、今回の得点を計算する。

2 回目以降の対戦では、Referee エージェントは、各 Player エージェントに対戦相手の前回の手を知らせて、次の手を求める。Player エージェントは対戦相手の前回の手を PlayBehavior で受け取り、その手を StrategyBehavior に伝える。そして、StrategyBehavior から次の手を受け取り、それを Referee エージェントに返答する。この返答を Referee エージェントは ConductMatchBehavior で受け取り、各 Player エージェントの手を照合して、今回の得点を計算する。そして、それぞれの Player エージェントがこれまでに獲得した得点に今回の得点を加算する。200 回の対戦が終了するまで、以上の処理を繰り返す。

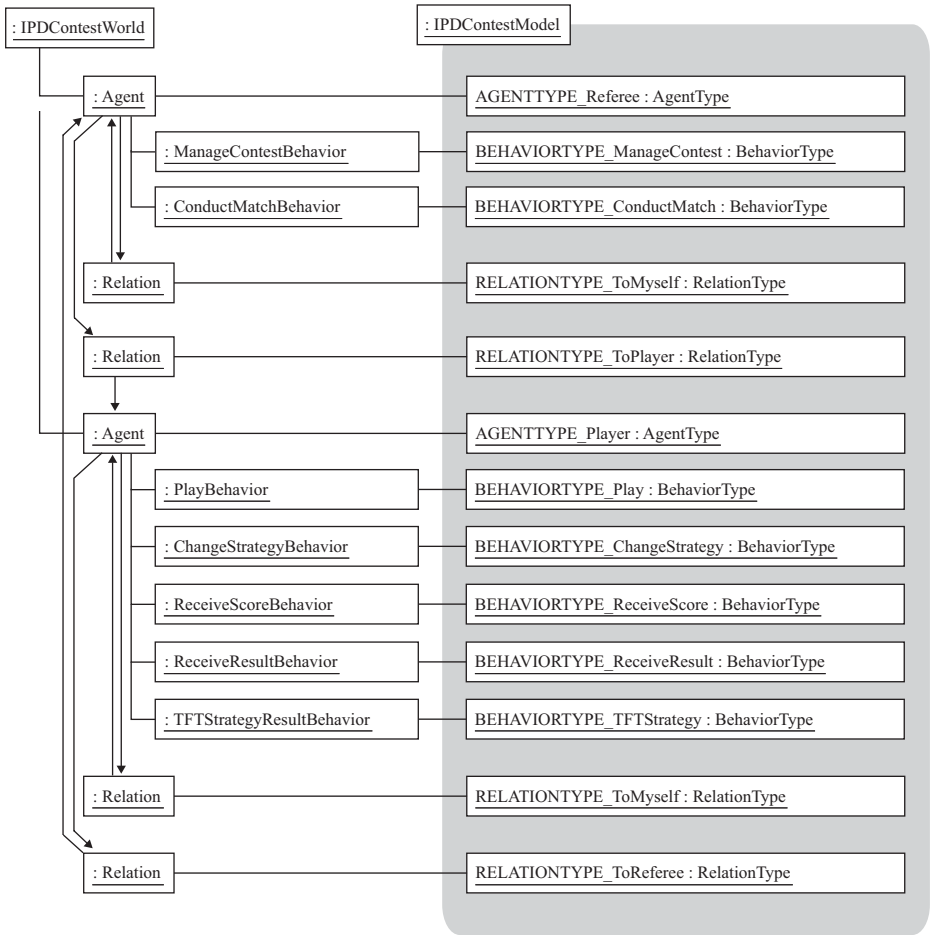


図 7.24: コンテスト・シミュレーションの全体像

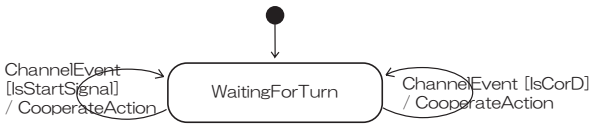


図 7.25: 戦略行動: ALLCStrategyBehavior

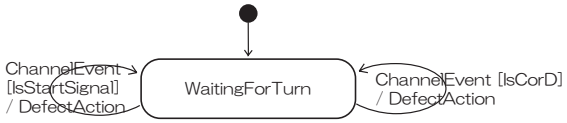


図 7.26: 戦略行動: ALLDStrategyBehavior

200 回の対戦終了後、Referee エージェントは、対戦した Player エージェントに、今回の試合におけるお互いの得点を知らせる。Player エージェントは、ReceiveScoreBehavior でその情報を受け取り、記憶する。次に Referee エージェントは、今回の試合の得点を、自らの ManageContestBehavior に知らせる。知らせを受けた ManageContestBehavior は、今回の試合の得点を、これまでの試合で各 Player エージェントが獲得した総得点のリストに加算する。こうして 1 組の試合が終了する。以上の処理を、すべての Player エージェントが総当りで試合を終えるまで繰り返す。

総当たりを終了した後、Referee エージェントは、今回のコンテストの総得点のリストを受け取り、すべての Player エージェントに知らせる。Player エージェントは、ReceiveResultBehavior で総得点のリストを受け取り、記憶する。

このシミュレーションの結果は、表 7.1 のようになった。同じ戦略でも総得点が異なるのは、乱数を用いる戦略 (RANDOM と JOSS) があるからである。

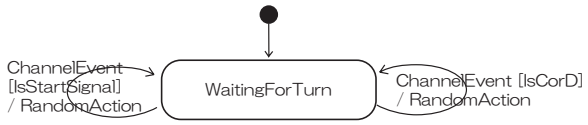


図 7.27: 戦略行動: RandomStrategyBehavior

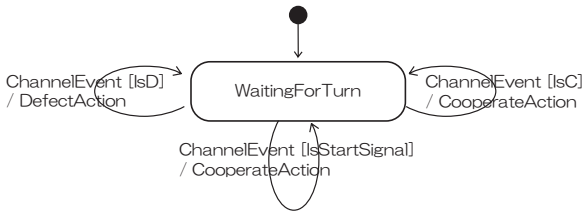


図 7.28: 戦略行動: TFTStrategyBehavior

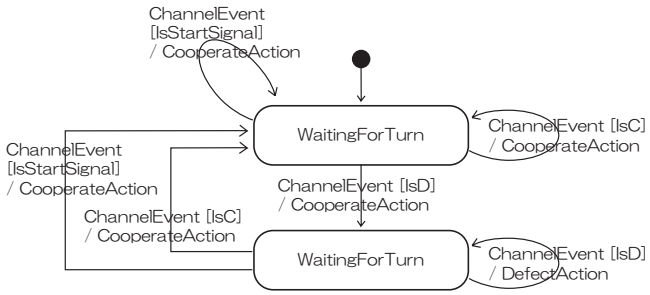


図 7.29: 戦略行動: TF2TStrategyBehavior

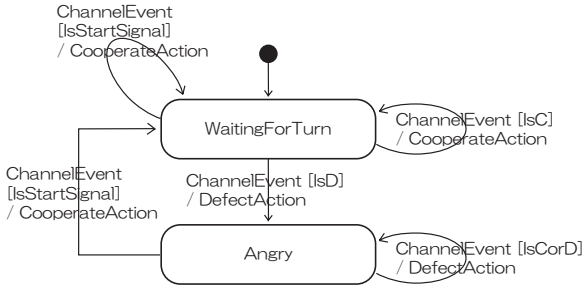


図 7.30: 戦略行動: FRIEDMANStrategyBehavior

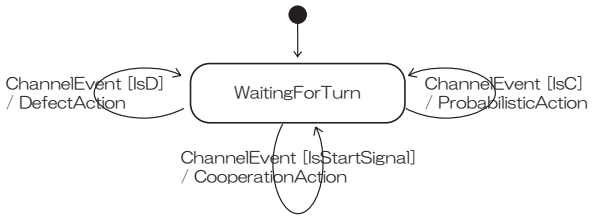


図 7.31: 戦略行動: JOSSStrategyBehavior

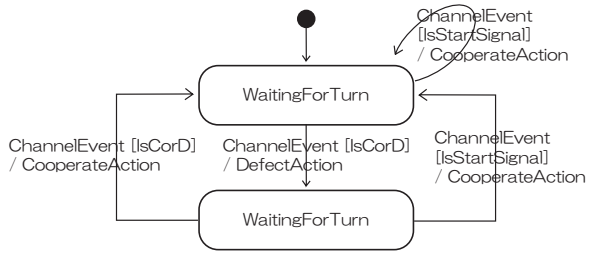


図 7.32: 戦略行動: PER-CDStrategyBehavior

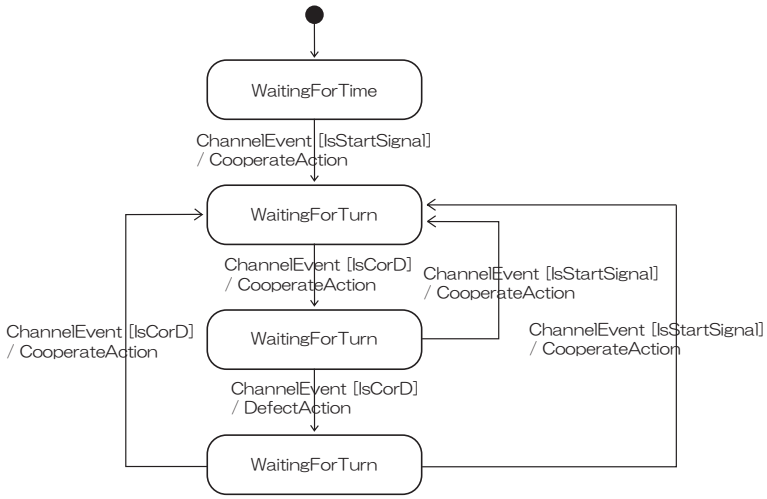


図 7.33: 戦略行動: PER-CCDStrategyBehavior

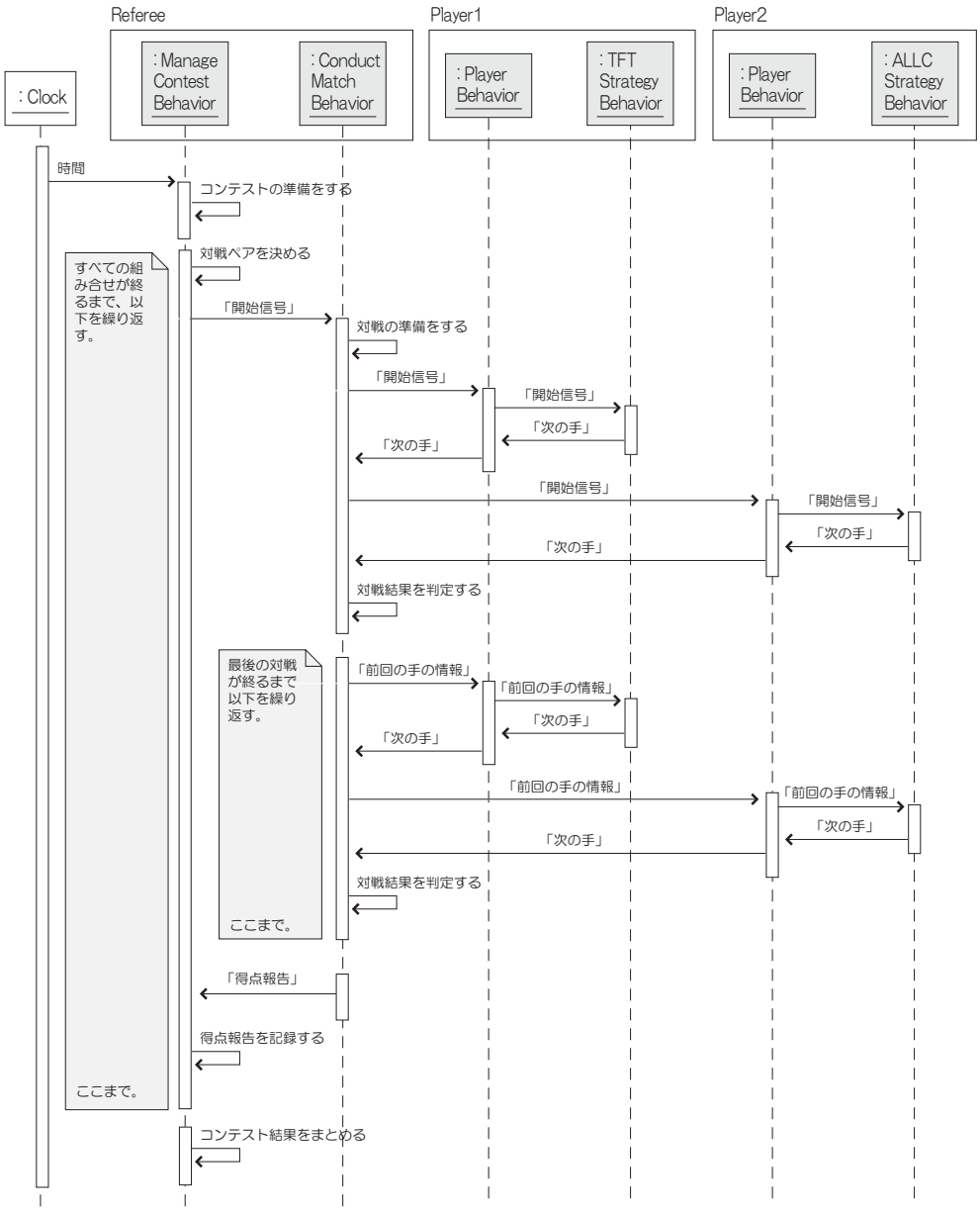


図 7.34: コンテスト・シミュレーションのシーケンス図

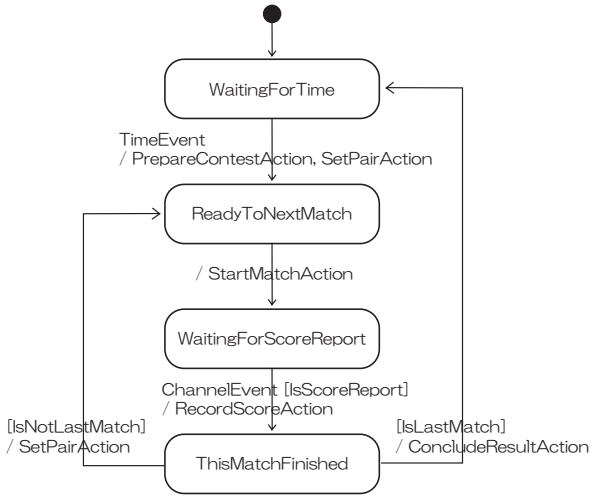


図 7.35: コンテスト・シミュレーション: ManageContestBehavior

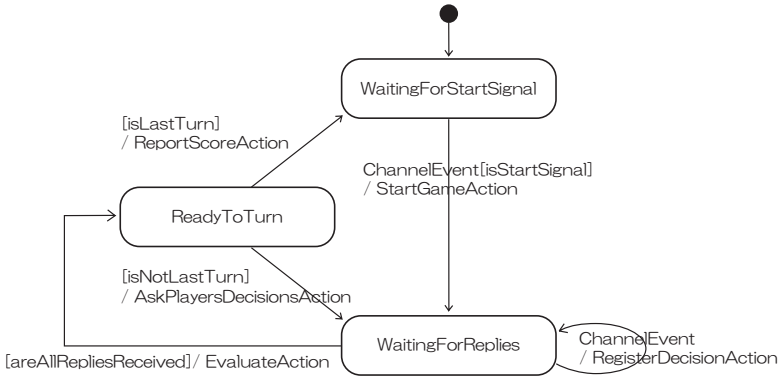


図 7.36: コンテスト・シミュレーション: ConductMatchBehavior

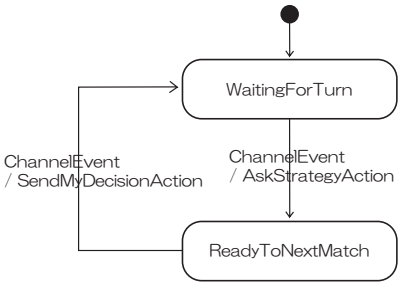


図 7.37: コンテスト・シミュレーション: PlayBehavior

表 7.1: コンテスト・シミュレーションの結果

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

7.2.2　戦略模倣シミュレーション

ここでは、繰り返し囚人のジレンマのコンテスト・シミュレーションに、独自の拡張を行ってみることにしたい。その拡張とは、各コンテスト終了後に、各 Player エージェントが自分より強い相手の戦略を模倣するというものである(66)。模倣相手の候補の選択は、次の 2 つの方法のいずれかで行い、候補が複数の場合には、その中からランダムに選択することにする(67)。

1. 個別対戦において、自分に勝ったプレイヤーの戦略を採用する (試合結果による選択)。
2. コンテストにおける総得点が、自分よりも高いプレイヤーの戦略を採用する (コンテスト結果による選択)。

この拡張モデルでは、それぞれの Player エージェントが戦略の変更を行うが、これは行動のルールが変化するという意味で、本論文でいうところの狭義の複雑系のモデルになっている。

戦略模倣シミュレーションでは、コンテスト・シミュレーションの最後の部分に、次のような流れを追加する (図 7.38)。TimeEvent によって発火する Referee エージェントの一連の行動が終わった後、Player エージェントが TimeEvent を受け取る。Player エージェントは、TimeEvent を ChangeStrategyBehavior で受け取り、他者を模倣して戦略変更する (図 7.39)。

戦略ごとに 2 人ずつ Player エージェントを用意したシミュレーションを行ったところ、結果は次のようになった。まず、候補選択方式 1(試合結果による選択) の場合には、数ステップで「ALL-D」戦略のみになり (図 7.40)、平均得点は初期状態よりも低い水準になる (図 7.41)。最終的に「ALL-D」戦略のみになった状況では全員が裏切りあうため、社会的にみて得点が低い水準になるのである。この結果を Player エージェントの個別状況で見てみると、次のことがわかる (図 7.42)。協調を基本とする戦略は、裏切りを交える戦略よりも得点が低いことが多く、また、対戦相手が協調を基本とする場合には同点になるため、戦略模倣の候補にはなりにくい。これに対し、裏切りを交える戦略は、協調を基本とする戦略よりも高い得点を獲得するため、戦略模倣の候補となる。裏切りを交える戦略のなかでも、裏切りが多いほど得点が高くなるため、「ALL-D」戦略が広まることになる。この結果は、乱数シードを変えても同様の結果になる。

これに対し、候補選択方式 2(コンテスト結果による選択) の場合には、「FRIEDMAN」戦略が広まり (図 7.43)、平均得点は初期状態よりも高い水準になる (図 7.44)。平均得点が候補選択方式 1(試合結果による選択) の場合に比べて高いのは、「FRIEDMAN」戦略における協調の効果である。最終的に「FRIEDMAN」戦略のみになったときには、すべての対戦で協調するため、社会的にみて得点が高い水準になるのである。こ

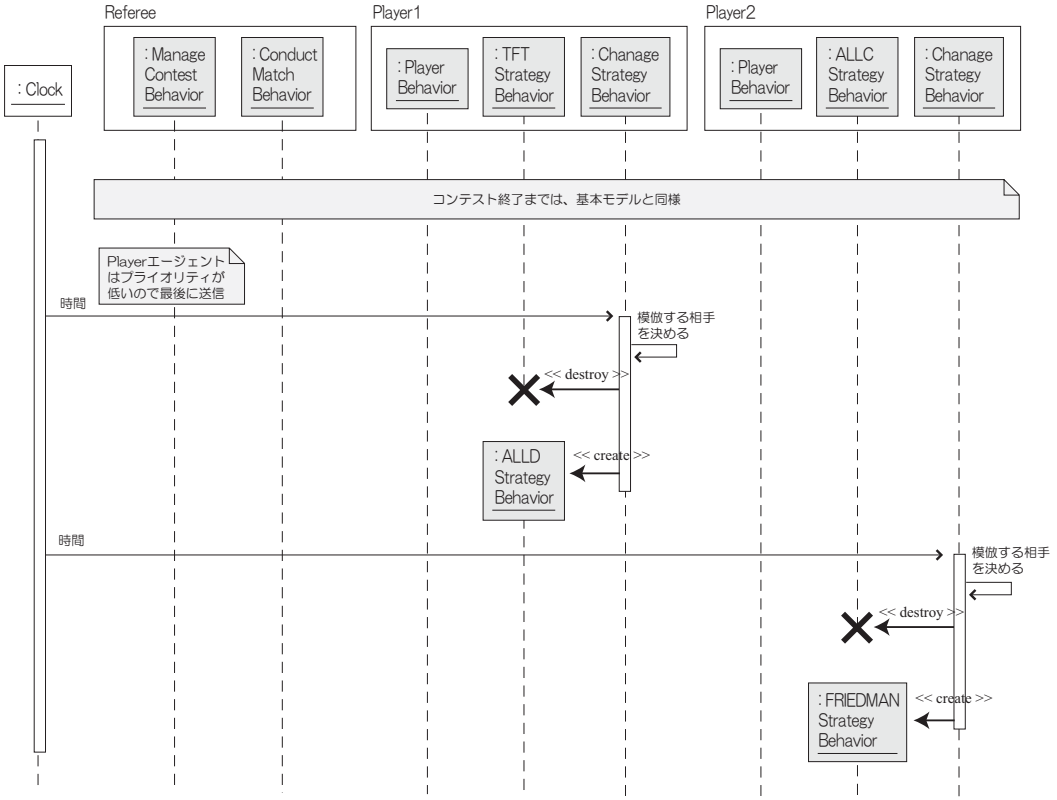


図 7.38: 戦略模倣シミュレーションのシーケンス図

の結果を、Player エージェントの個別状況で見てみると、協調を基本とするいくつかの戦略が残った後に、「FRIEDMAN」戦略が広まっていることがわかる (図 7.45)。協調を基本とする戦略の中でも特に「FRIEDMAN」戦略が残るのは、「ALL-D」戦略などの裏切りの多い戦略に対して効果的に反撃することができるためだと考えられる。この候補選択方式 2(コンテスト結果による選択) では、乱数シードを変更して実行すると、「FRIEDMAN」戦略に混じって「TFT」戦略が残ることがある (図 7.46)。この場合にも、両戦略とも協調するので得点は高い水準になる (図 7.47)。

以上の結果をまとめると、次のようになる。まず、試合結果というミクロ的で個別的な勝敗を判断材料にして戦略を変化させると、裏切りが強調されて加速度的に広まっていく。これに対し、コンテスト結果というマクロ的で総合的な勝敗を判断材料にして戦略を変化させると、協調による得点の上昇が効果を発揮し、協調が加速度的に広まっていく。裏切りがもっている得点のインパクトに比べ、協調の効果は静かなものであるが、総合的に見た場合には、この効果が大きくなる。以上のシミュレーション結果だけで何かを主張することはできないが、社会全体への公表や表彰の存在が、社会的効用を高めるのに効果がある可能性が示唆されたといえるだろう。

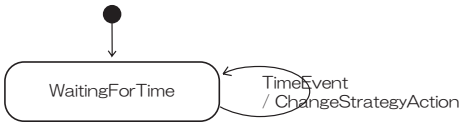


図 7.39: 戦略模倣シミュレーション: ChangeStrategyBehavior

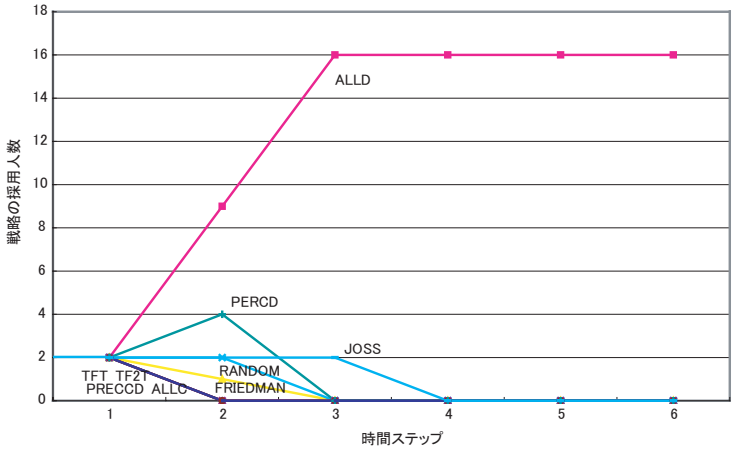


図 7.40: 戦略模倣シミュレーション: 各戦略を採用しているプレイヤー数の推移 (試合結果による戦略変更)

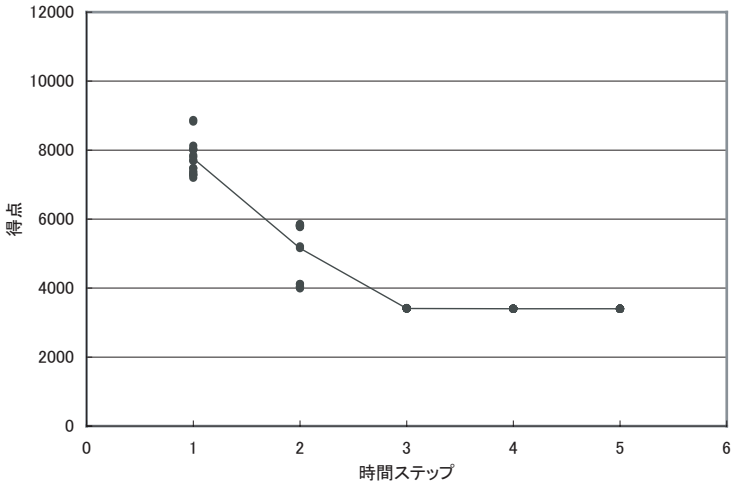


図 7.41: 戦略模倣シミュレーション: 各プレイヤーの得点と平均得点の推移 (試合結果による戦略変更)

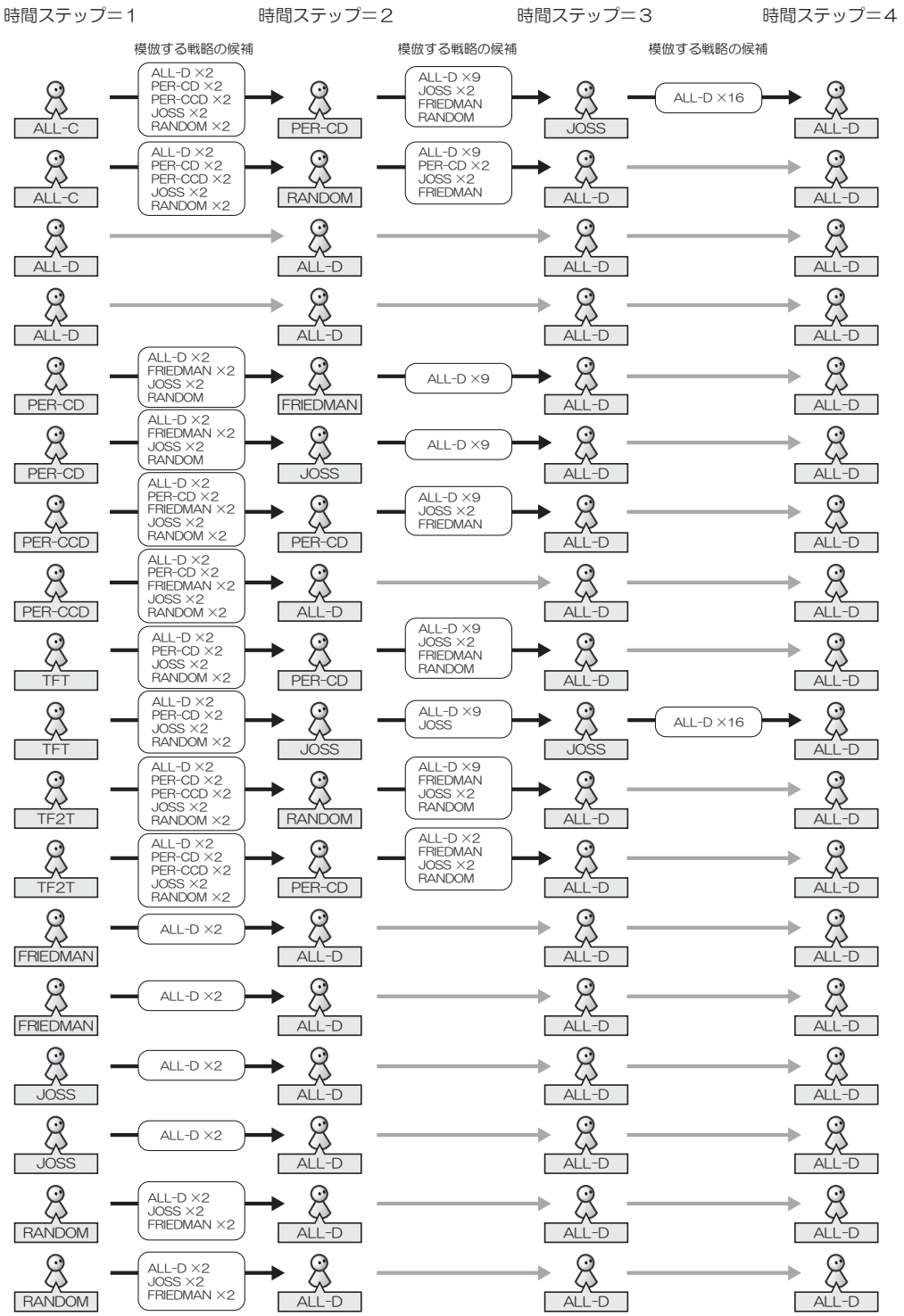


図 7.42: 戦略模倣シミュレーション: プレイヤーの戦略の変化 (試合結果による戦略変更)

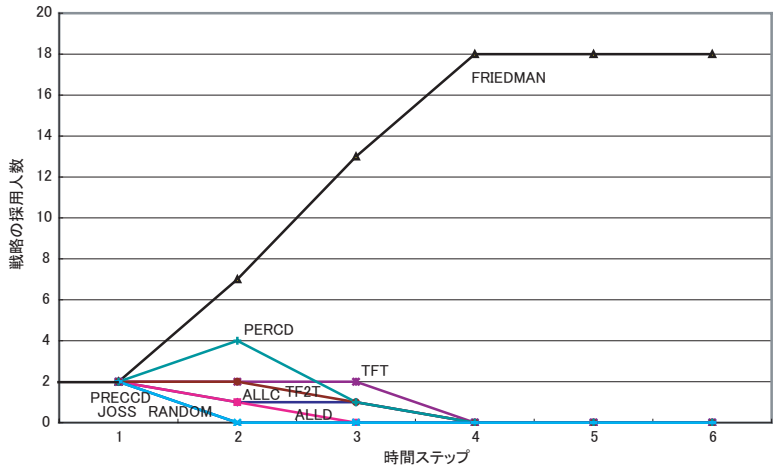


図 7.43: 戦略模倣シミュレーション: 各戦略を採用しているプレイヤー数の推移 (コンテスト結果による戦略変更　)

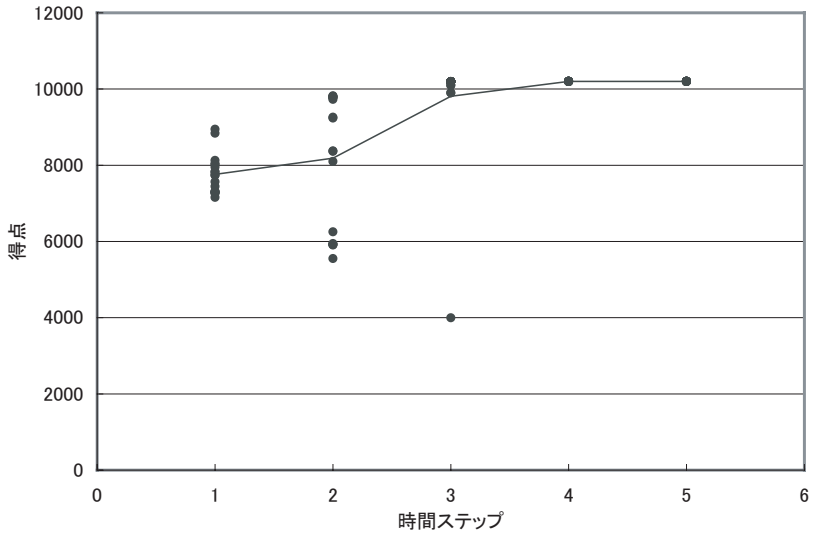


図 7.44: 戦略模倣シミュレーション: 各プレイヤーの得点と平均得点の推移 (コンテスト結果による戦略変更　)

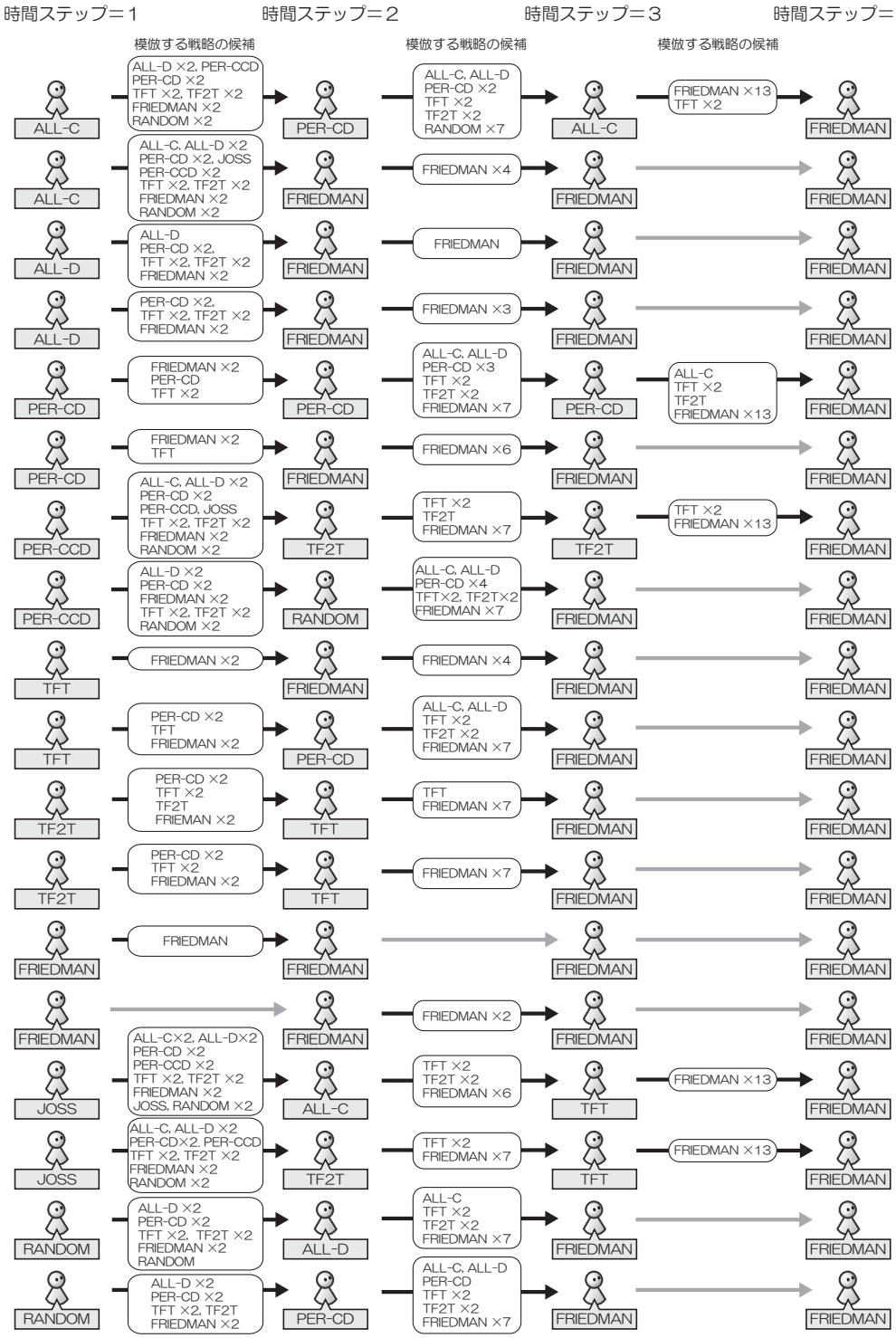


図 7.45: 戦略模倣シミュレーション: プレイヤーの戦略の変化 (コンテスト結果による戦略変更　)

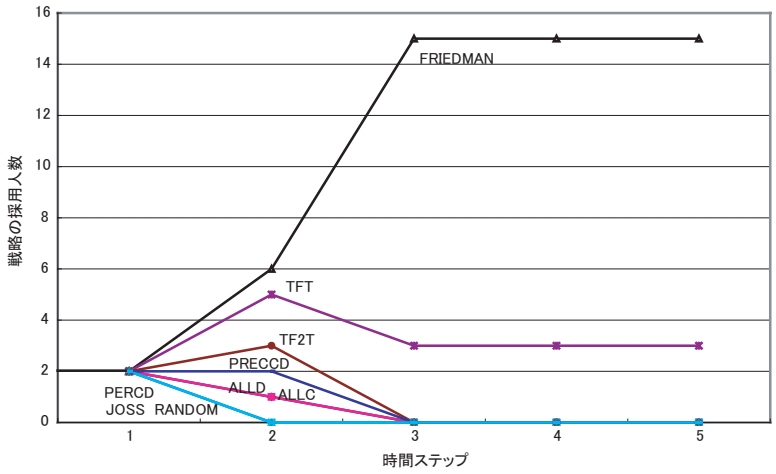


図 7.46: 戦略模倣シミュレーション: 各戦略を採用しているプレイヤー数の推移 (コンテスト結果による戦略変更　)

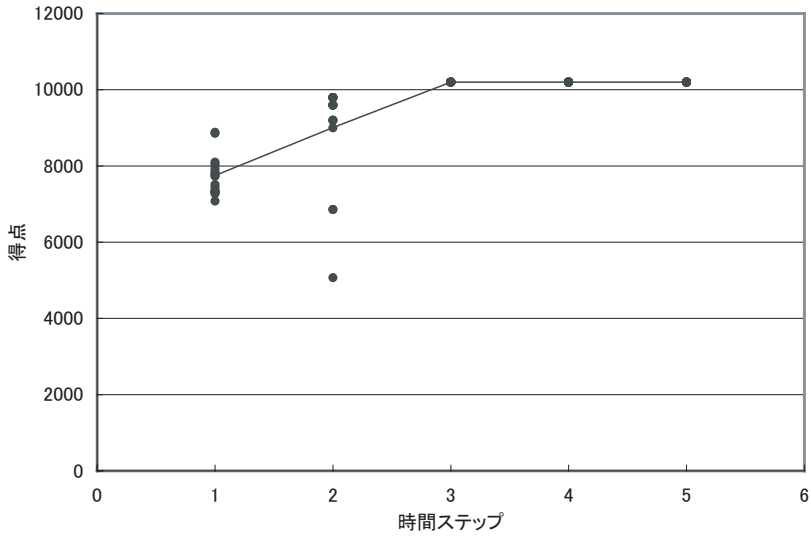


図 7.47: 戦略模倣シミュレーション: 各プレイヤーの得点と平均得点の推移 (コンテスト結果による戦略変更　)

7.3 貨幣の自生と自壊モデル

次に取り上げるモデルは、安冨 (2000) の貨幣の自生と自壊モデルである。このモデルは、主体全員が生産者かつ消費者である社会において、物々交換している商品のひとつが、ある時から貨幣としての役割を担うようになるという興味深いモデルである。また、同じモデルにおいて、貨幣の自生という事実への各主体の適応行動が、今度は逆にその貨幣が崩壊させてしまうということも観察されている。

安冨 (2000) によると、貨幣経済は、人間が単独で自給自足することができず、生存・欲求充足のために他人とのコミュニケーションが必要となる分業社会において発生しうる。原始的な分業社会における人びとは、自分の生産する財を、他人が生産しかつ自分が欲する財と物々交換することで、自分の生存・欲求を充足させている。このような物々交換取引の際に、人びとが「自分の欲求する財しか受け取らない」のであれば、欲望の二重の一致の困難が生じるため、取引がきわめて成立しにくい。しかし、実際は人間はこのように近視眼的ではなく、将来予測を行うことができる。それでは、人びとが将来の交換の有便性を考慮し、「自分は欲求しないが、多くの人が需要する財も受け取る」という行動をとる場合には、どのようなことが起こるのだろうか。安冨 (2000) は、近視眼的な社会と将来予測を行う社会の 2 つのケースについて、シミュレーションによる分析を行っている。その結果、将来予測を行う社会において、ある商品の交換可能性が極端に高くなり、「貨幣的商品」が発生することが観察されている。

ここでは、安冨 (2000) における「物々交換モデル」、「貨幣的交換モデル」、「進化的モデル」の 3 つのシミュレーションを再現することにしたい。

7.3.1 物々交換モデル

物々交換モデルに登場するエージェントは、生産者であり消費者である主体 1 種類だけであり、これを Agent エージェントとする (図 7.48)。このシミュレーションの流れは次のようになる (図 7.49, 7.50)。

まず、Agent エージェントが TimeEvent を受信する。どの順番で Agent エージェントが TimeEvent を受信するのかは、ランダムになっている。TimeEvent が奇数回目のときには、Agent エージェントは、SearchBehavior(図 7.51; 厳密には SearchBehavior を特化した MaximumSearchBehavior) で、他のすべての Agent エージェントに、どのような財を所有しているのかを質問する。他の Agent エージェントは、それぞれ RespondToSearchBehavior (図 7.52) で質問を受けると、現在所有している財のリストを返答する。Agent エージェントは、MaximumSearchBehavior で返答を集めていき、返答をすべて受け取った後に、自分の欲求する財を最もたくさんもっている Agent エージェントを選び、その Agent エージェントと物々交換を行うために自分の

DecideTradeBehavior (図 7.53) に連絡する。

Agent エージェントは、DecideTradeBehavior で取引相手が誰なのかという情報を受け取ると、お互いが実際に交換できる財を特定するために、取引相手に自分のもっている財のリストを知らせる。取引相手となった Agent エージェントは、RespondToDecideTradeBehavior (図 7.54) で財のリストを受け取ると、自分の欲求する財から、相手の所有財リストから自分の需要する財のリストを作成して、自分の所有財リストと一緒に返答する。相手の需要する財のリストと、所有財リストを受け取った Agent エージェントは、同様に自分の需要する財のリストを作成して、相手の需要する財のリストと照合する。この時、相手が何も需要していなければ、実際には何も交換しない。

相手が需要している場合には、お互いに渡す財の量が同じになるように、実際に交換する財のリストを作成し、お互いの ExchangeBehavior (図 7.55) に知らせる。Agent エージェントは、ExchangeBehavior で実際に交換する財のリストを受け取ると、お互いの RespondToExchangeBehavior (図 7.56) に財を渡す。この時、渡した個数に応じた運送コストを効用から減少させる。RespondToExchangeBehavior で財を受け取った Agent エージェントは、その財を自分の所有財とする。

一連の物々交換の過程が終了した後に、Agent エージェントは DecideTradeBehavior からお互いの ConsumeAndProduceBehavior (図 7.57) に対して、財を消費して効用を高め、欲求を変更するよう連絡する。ConsumeAndProduceBehavior で連絡を受け取ったお互いの Agent エージェントは、もし欲求する財をもっていればそれをすべて消費して、個数分だけ効用を高め、必ず欲求する財を変更する。もし欲求する財をもっていなければ、何も消費せず、効用も高めないが、一定の低い確率で欲求する財を変更する。

Agent エージェントは、偶数回目の TimeEvent を受け取ったときに、ResetUtilityBehavior (図 7.58) によって自分の効用得点を 0 にリセットする。

シミュレーションの結果は、図 7.59 のようになった。この図からわかるように、自分の欲求する財しか受け取らない主体によって構成される社会では、欲望の二重の一致の困難によって、自分の欲求する財を入手・消費するのは容易ではない (図 7.60)。

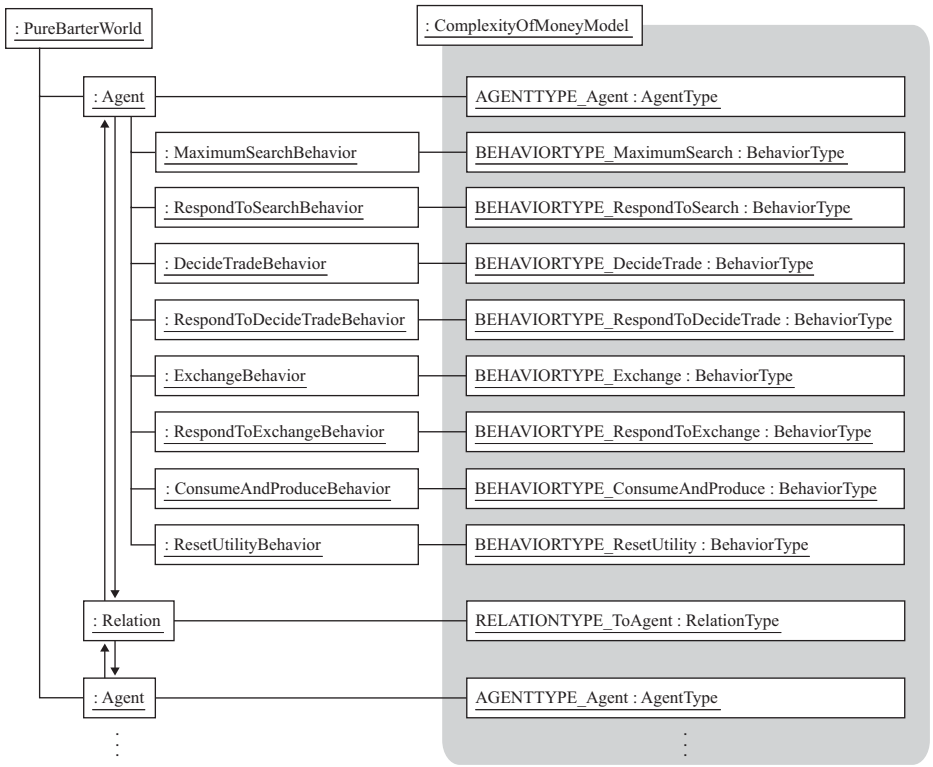


図 7.48: 物々交換モデルの全体像

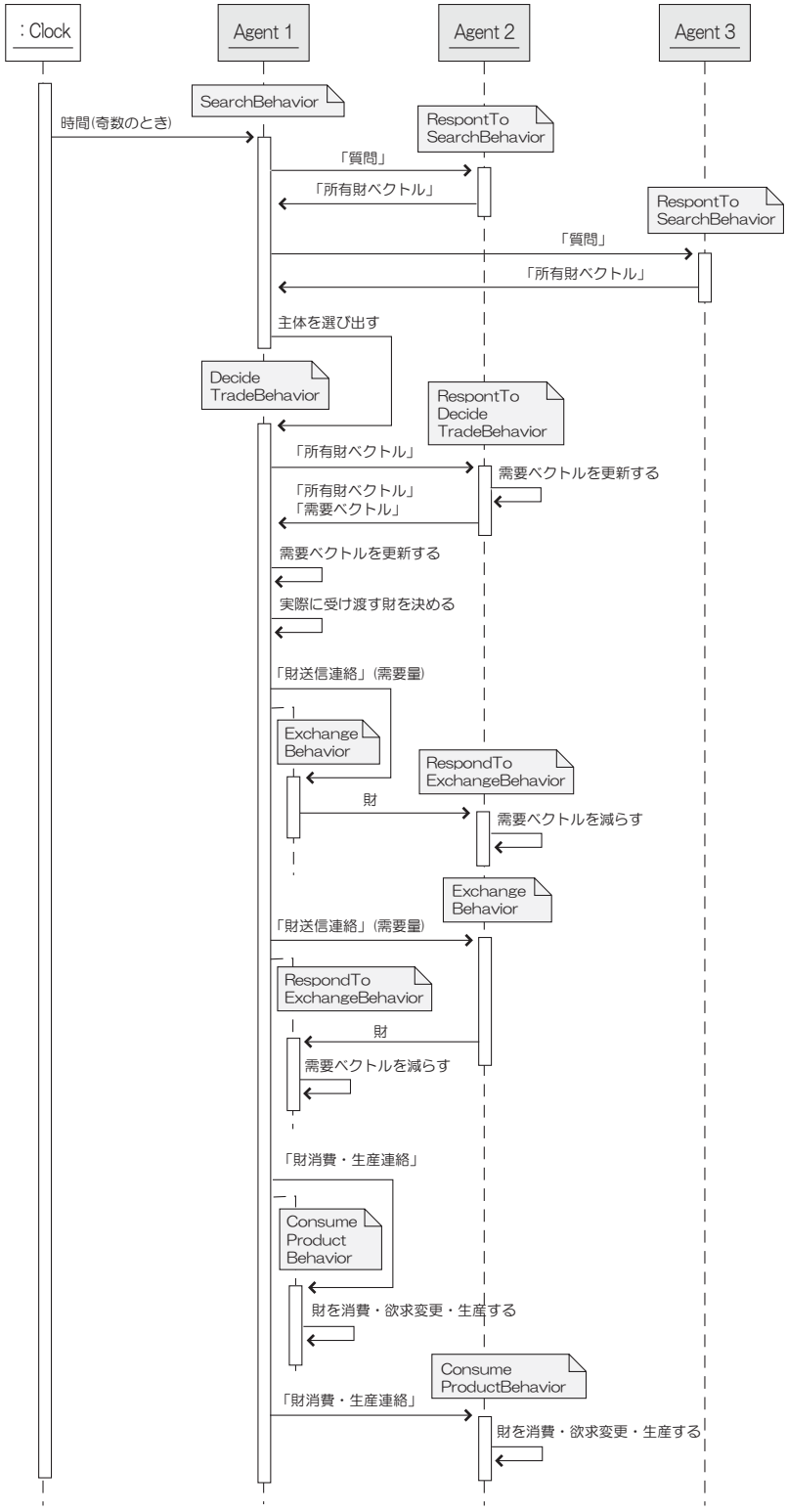


図 7.49: 物々交換モデル: TimeEvent(奇数) のときのシーケンス図

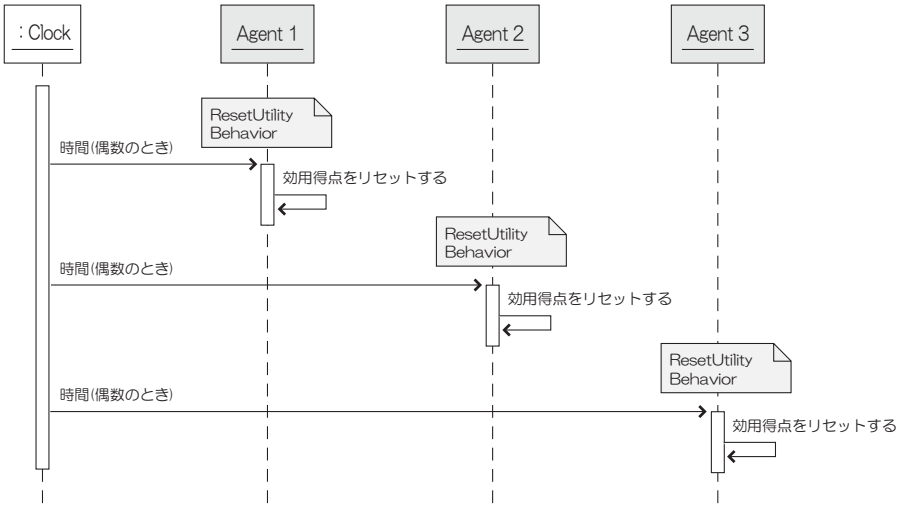


図 7.50: 物々交換モデル: TimeEvent(偶数) のときのシーケンス図

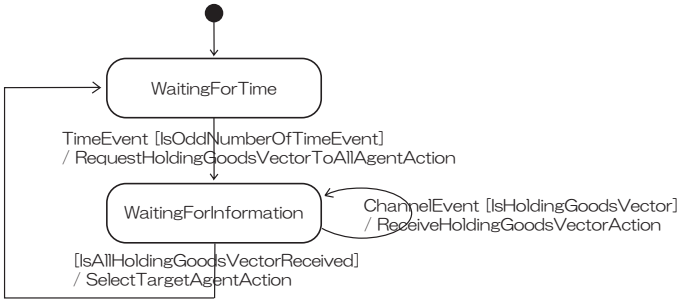


図 7.51: 物々交換モデル: SearchBehavior

図 7.52: 物々交換モデル: RespondToSearchBehavior

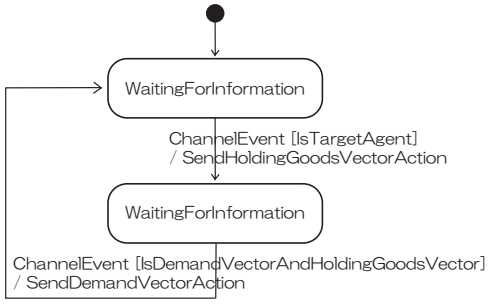


図 7.53: 物々交換モデル: DecideTradeBehavior

図 7.54: 物々交換モデル: RespondToDecideBehavior

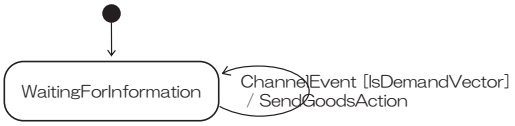


図 7.55: 物々交換モデル: ExchangeBehavior

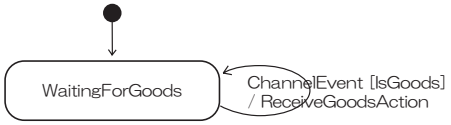


図 7.56: 物々交換モデル: RespondToExchangeBehavior

図 7.57: 物々交換モデル: ConsumeProductBehavior

図 7.58: 物々交換モデル: ResetUtilityBehavior

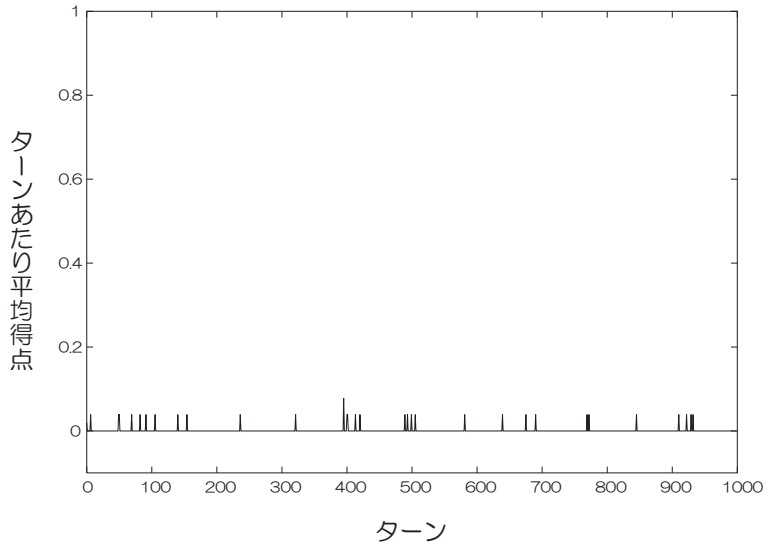


図 7.59: 物々交換モデル: 各ターンごとの得点の推移 (N=50, Threshold=0.078)

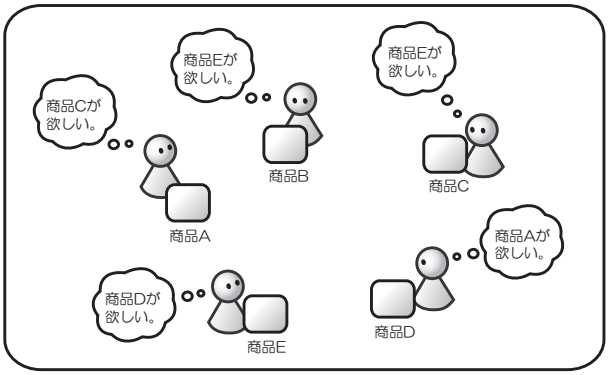


図 7.60: 物々交換モデルで起こっていることのイメージ (欲望の二重の一致の困難)

7.3.2　貨幣的交換モデル

貨幣的交換モデルでは、「将来の交換の簡便を考えて、たとえ自分が欲求してなくても、他人が需要するものを需要するようにする」という戦略を各主体にもたせる (図 7.61)。モデルでは、実際に交換する財を特定する前に知識交換を行うようにするために、DecideTradeBehavior を ExtendedDecideTradeBehavior に拡張し、ChangeKnowledgeBehavior と RespondToChangeKnowledgeBehavior を、Agent エージェントにもたせる (図 7.62)。また、各 Agent エージェントに「商品に対する見解」と「閾値」(0 から 1 までの実数) の情報をもたせ、商品に対する評価が「閾値」以上であれば、自分が欲求していなくてもその商品を受け取るようにさせる。この貨幣的交換モデルでは、すべての Agent エージェントは同じ閾値をもっており、閾値の値も初期設定のまま変化しないとする[68]。

シミュレーションは、次のような流れになる (図 7.63)。Agent エージェントは、ExtendedDecideTradeBehavior で取引相手が誰なのかを知り、知識交換を行うように自分の ChangeKnowledgeBehavior (図 7.64) に連絡する。連絡をうけた ChangeKnowledgeBehavior は、商品に対する見解を取引相手の Agent エージェントに伝える。取引相手の Agent エージェントは、RespondToChangeKnowledgeBehavior (図 7.65) で見解を受け取り、自分の見解に加算・規格化し、新しい見解を返答する。返答を受け取った Agent エージェントは、新しい見解の数値を記憶する。

物々交換モデルと異なり、貨幣的交換モデルでは、相手の所有する財のリストから、自分の欲求する財のみならず、自分が欲しくなくても他人が需要しているものがあれば需要する。ある商品に対して需要を表明するか否かは、見解の数値が Agent エージェントのもつ閾値を超えているか否かで判断する。こうして得られたお互いの需要する財のリストを照合し、実際に交換できる財のリストを作成する処理以降の流れは、物々交換モデルと同じである。

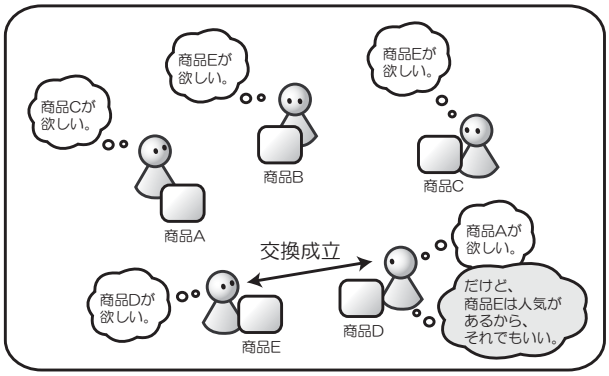


図 7.61: 貨幣的交換モデルのイメージ (人気のある商品の需要)

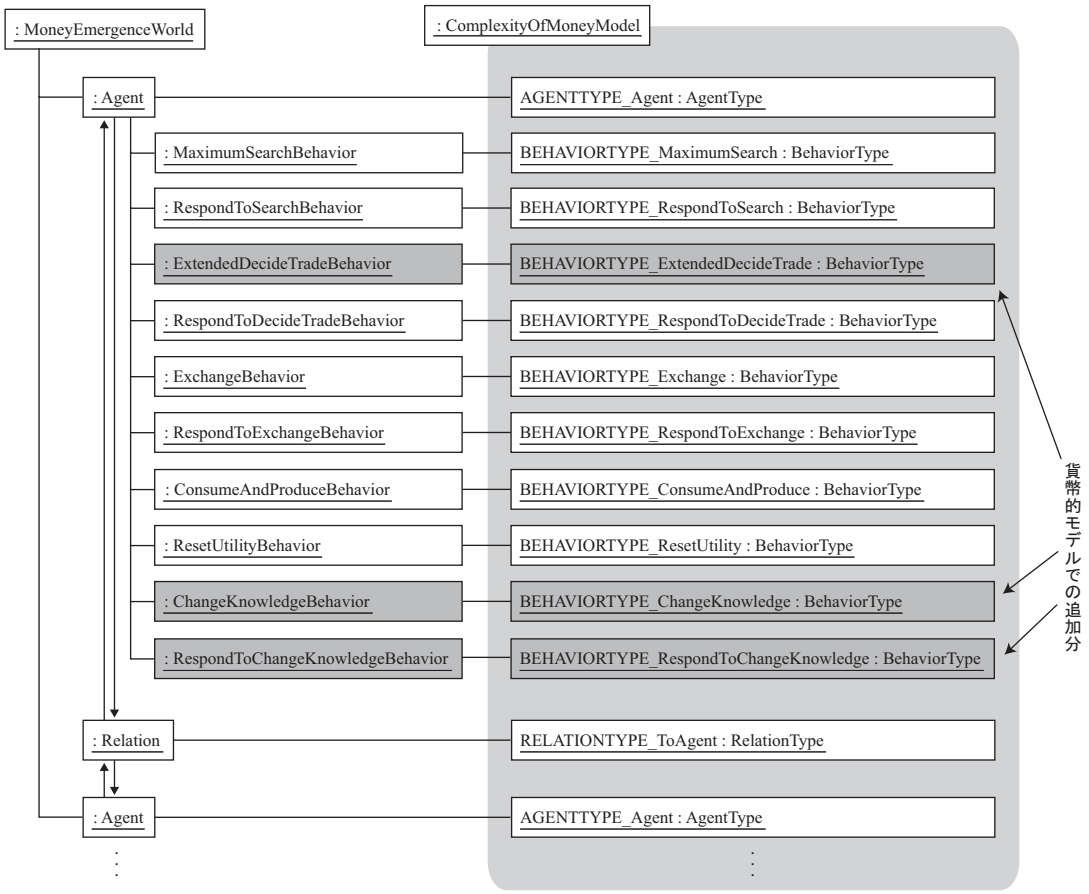


図 7.62: 貨幣的交換モデルの全体像

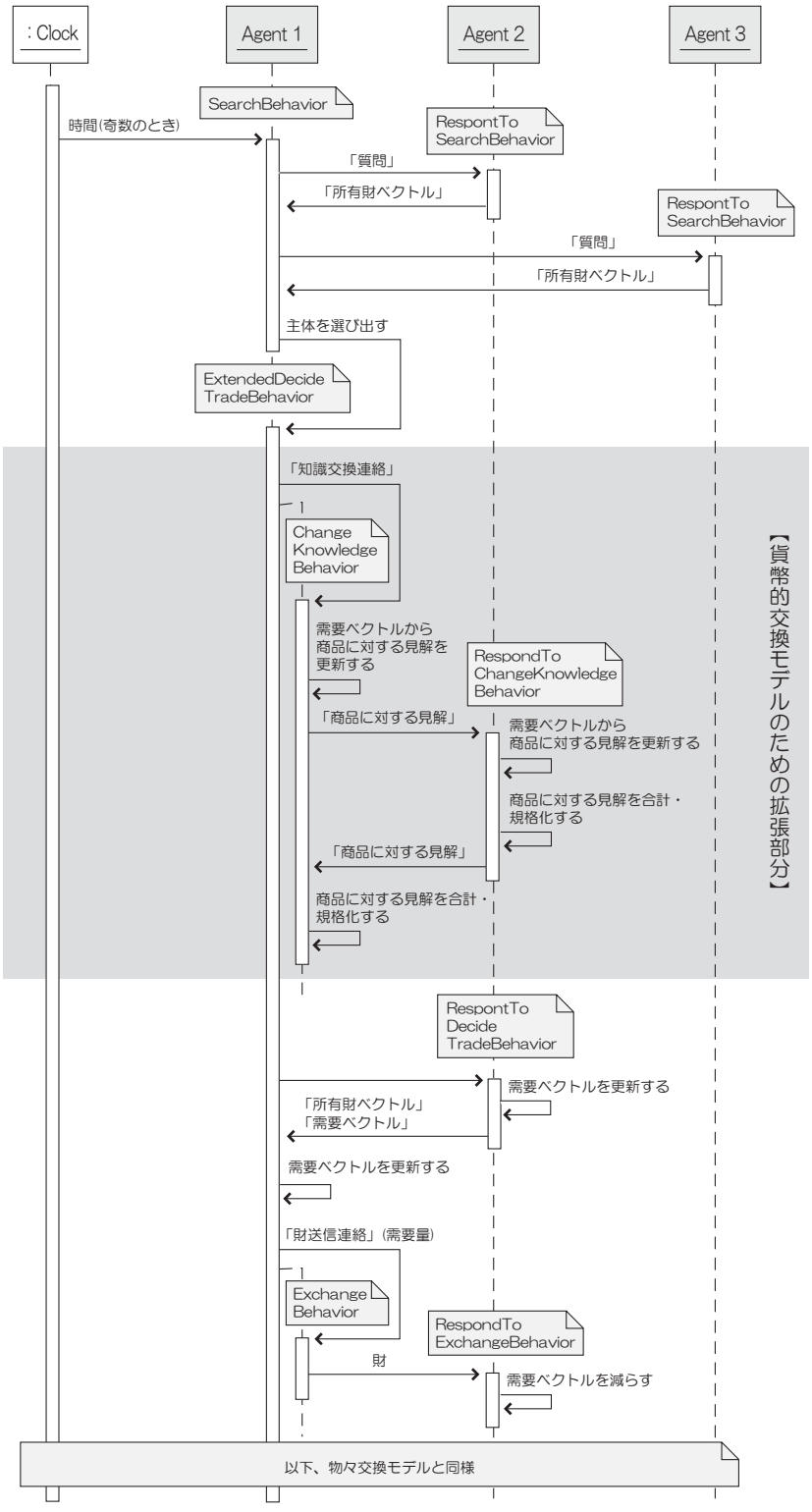


図 7.63: 貨幣的交換モデル: TimeEvent(奇数) のときのシーケンス図の一部

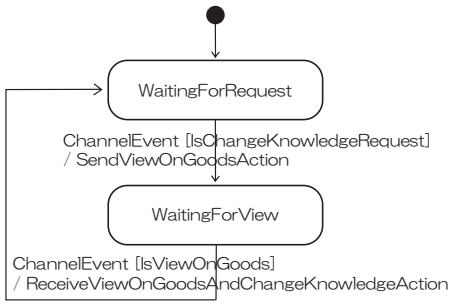


図 7.64: 貨幣的交換モデル: ChangeKnowledgeBehavior

図 7.65: 貨幣的交換モデル: RespondToChangeKnowledgeBehavior

シミュレーション結果は、図 7.66, 7.67, 7.68 のようになる。図 7.67 と図 7.66 からわかるように、最も市場性の高い商品の市場性が急激に高くなったと同時に、交換のために保有されている最も市場性の高い商品の単位数も増加し、それ以外の商品は交換の媒介として所有されなくなっている。このことから、「自分が欲してなくても他人の受け取るものであれば自分も受け取る」という戦略をもつ主体によって構成される社会では、ある時点において、最も市場性の高い商品が交換の媒介として認識されるようになり、それ以外のものを交換の媒介と見なすことがなくなるということがわかる。また、図 7.68 のように、貨幣的商品が生まれたことによって、主体の効用得点の平均値は明らかに高くなっている。貨幣的商品を媒介とすることによって、主体は本来自分の欲する財を、より効率的に入手・消費できるようになっているからである。

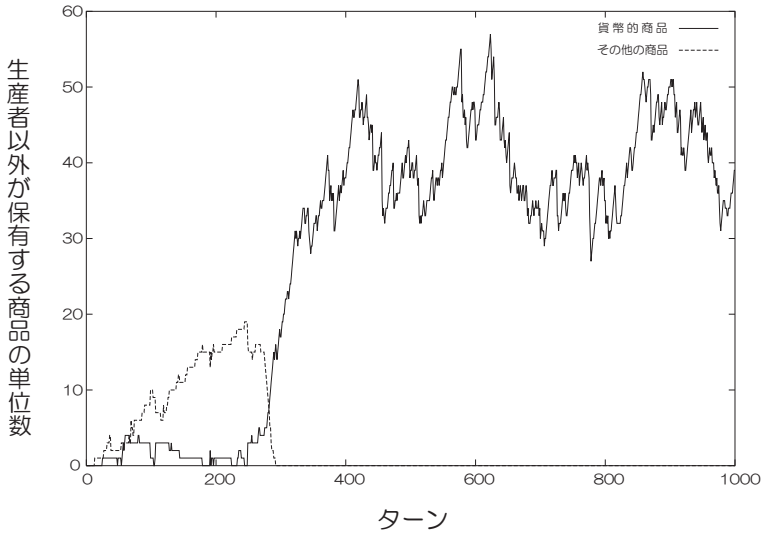


図 7.66: 貨幣的交換モデル: 交換のために保有されている商品の単位数の推移 (N=50, Threshold=0.078)

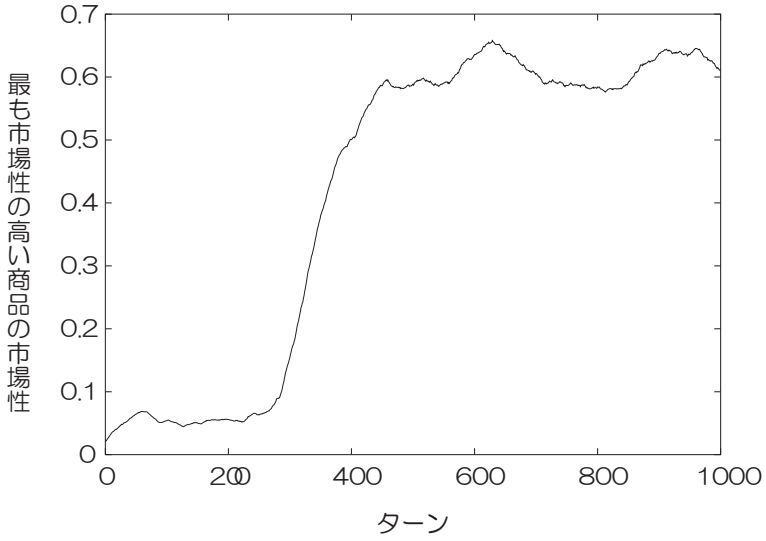


図 7.67: 貨幣的交換モデル: 最も市場性の高い商品の市場性の推移 (N=50, Threshold=0.078)

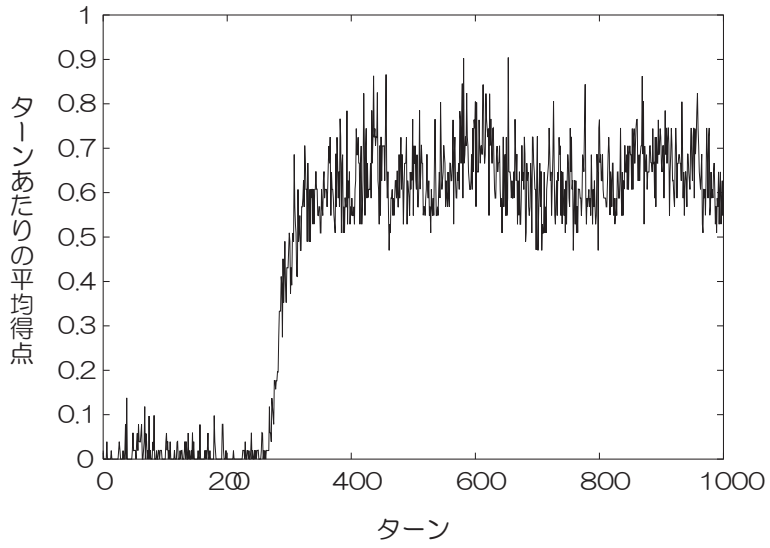


図 7.68: 貨幣的交換モデル: 各ターンごとの得点の推移 (N=50, Threshold=0.078)

7.3.3　進化的モデル

進化的モデルでは、「欲求する財を入手しやすい閾値をもつ Agent エージェントの閾値を、他の Agent エージェントが模倣する」という行動をモデルに導入する。具体的には、一日の最後に、得点の低い三人が得点の高い三人の閾値を模倣し、全員の閾値にノイズをかけるということを行う。

このことを実現するために、閾値の模倣を仲介する EvolutionFunction エージェントをモデルに追加する (図 7.69)。この EvolutionFunction エージェントが、2024 ステップ(69)に 1 回、効用得点が高い Agent エージェント 3 人の閾値を、低い Agent エージェント 3 人に模倣させる。

このシミュレーションの流れは、次のようになる (図 7.70)。EvolutionFunction エージェントは、ChangeThresholdBehavior (図 7.71) で TimeEvent を受け取ると、2048 回に 1 回、すべての Agent エージェントに対して、効用得点と閾値を尋ねる。RespondToChangeThresholdBehavior で質問を受け取った Agent エージェントは、過去 2048 ステップの効用得点の合計値と現在の閾値を答える (図 7.72)。すべての Agent エージェントから返答をうけとった EvolutionFunction エージェントは、得点の高い Agent エージェント 3 人と得点の低い Agent エージェント 3 人を調べ、得点の高い方の Agent エージェントの閾値を低い方の Agent エージェントに教える。得点の低い Agent エージェント 3 人は、RespondToChangeThresholdBehavior で閾値を受け取ると、それぞれの閾値を更新する。

以上の処理の後、EvolutionFunction エージェントは、すべての Agent エージェントに、閾値にノイズをかけるよう連絡する。Agent エージェントは、RespondToThresholdBehavior で連絡を受け取ると、自分の閾値に平均 0、分散 0.00005 のノイズをかける。この時、ノイズによって閾値が 1 を超える、あるいは 0 未満になる場合は、差分だけ跳ね返す。例えば、1. 02 になってしまう場合は、$1 - (1.02 - 1) = 0.98$ とする。

なお、この進化的モデルでは、過去 2048 ステップの効用得点の合計値を手に入れるため、以前のモデルでは偶数回目の TimeEvent で呼び出されていた ResetUtilityBehavior が、2048 回に 1 回呼び出されるように変更する。

このシミュレーション結果は、図 7.73, 7.74 のようになる。およそ 58 日目に商品 0 (GOODSTYPE_Goods0) が貨幣として選ばれ、310 日目まで用いられるが、その後突然崩壊してしまう。次に商品 39 (GOODSTYPE_Goods39) が貨幣となるが、この商品は 20 日間貨幣であり続けた後に崩壊し、その座を商品 15 (GOODSTYPE_Goods15) に受け渡している。図 7.74 を見ると、最初急激に閾値が落ちている。これは、貨幣的商品がまだ生まれていない社会では、閾値の低い主体のほうがより効率よく自分の欲する商品を入手・消費できることを表している。

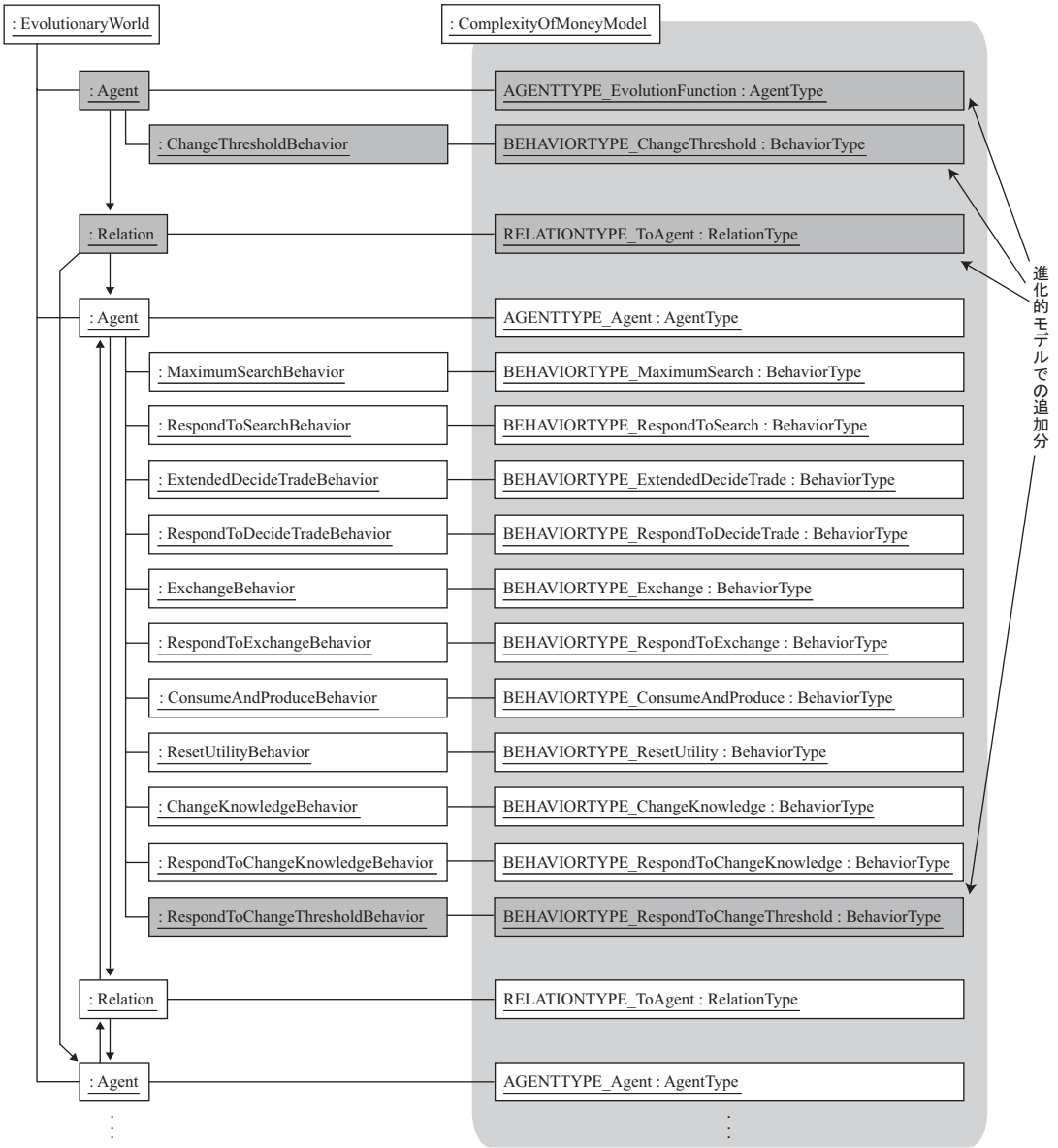


図 7.69: 進化的モデルの全体像

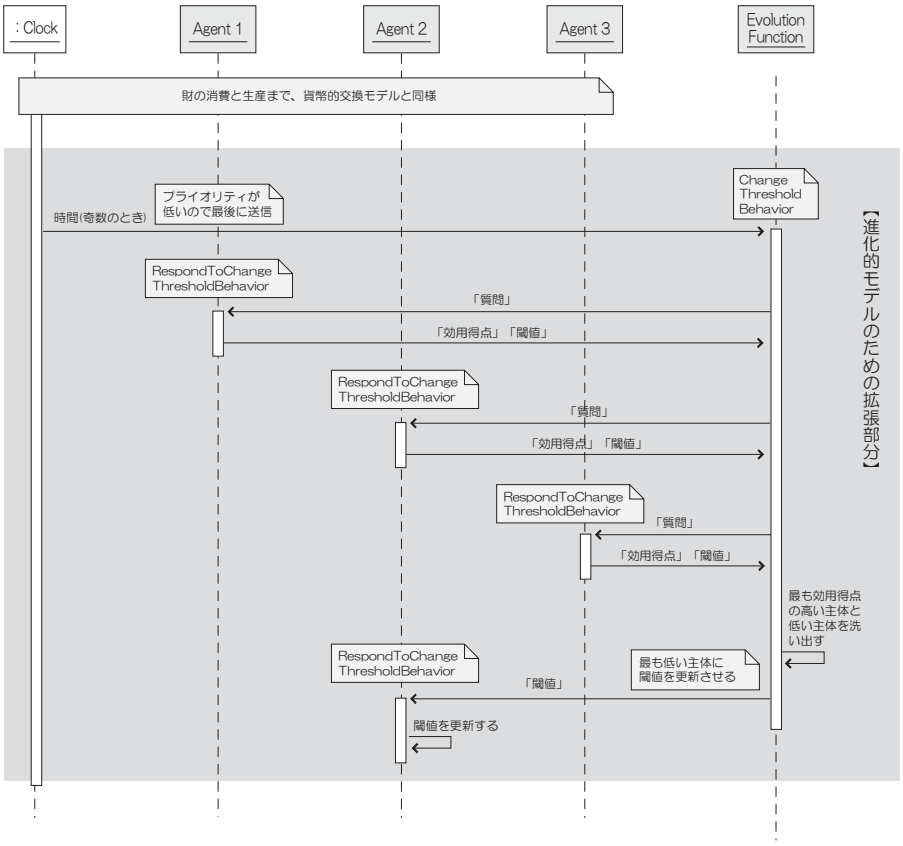


図 7.70: 進化的モデル: TimeEvent(奇数) のときのシーケンス図の一部

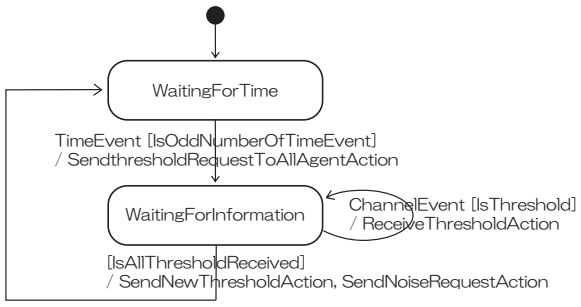


図 7.71: 進化的モデル: ChangeThresholdBehavior

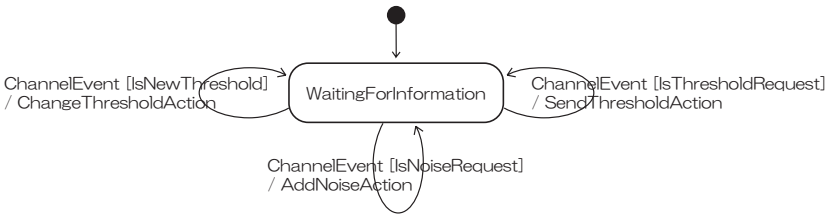


図 7.72: 進化的モデル: RespondToChangeThresholdBehavior

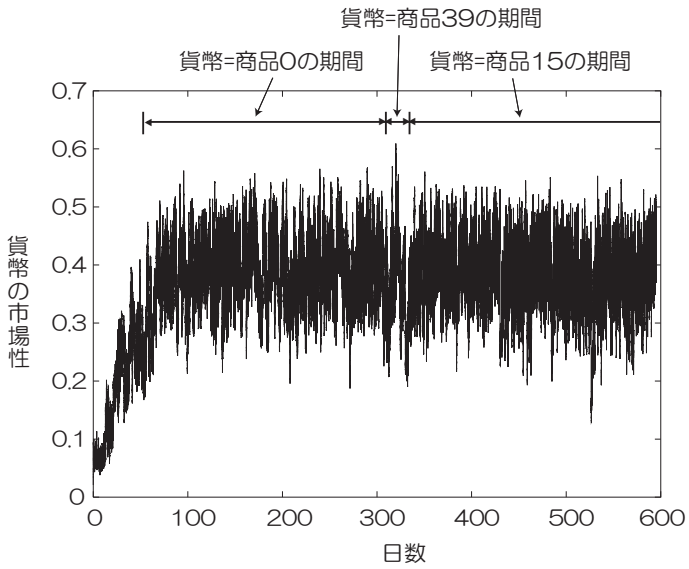


図 7.73: 進化的モデル: 貨幣の市場性の推移

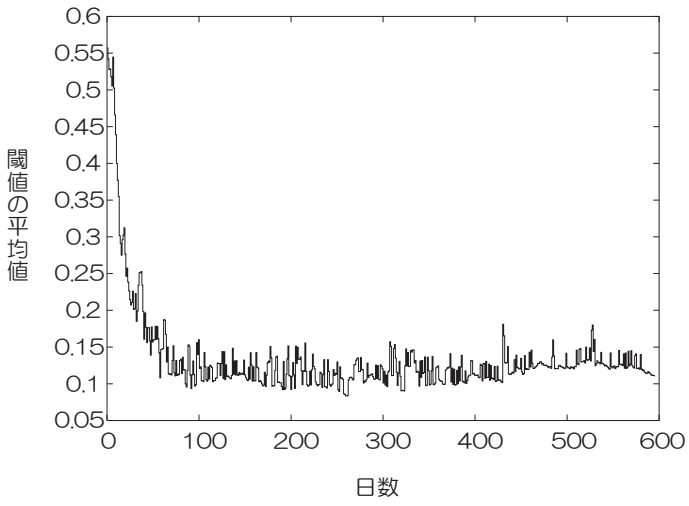


図 7.74: 進化的モデル:閾値の平均値の推移

7.4 SugarScape モデル

Sugarscape は、Epstein and Axtell (1996) によって提案・分析が行われた人工社会モデルである。2 次元セル空間の一部に、時間とともに再生する砂糖が配置されており、その砂糖をエージェントが取得する。この Sugarscape を直接的に表現するならば、エージェントは、環境 (空間に配置された砂糖) と相互作用することになるが、BEFM では、このようなモデルをそのまま記述することはできない。なぜなら、空間に (どのエージェントにも所有されていない) Goods を配置することや、その Goods が自動的に変化することを表現できないからである。この制約が生じるのは、BEFM が「時間経過とともに変化するものは、エージェントとしてモデル化する」という方針を採っているためである。

空間に配置された砂糖が時間とともに再生することを表現するためには、次の二つの実現方法が考えられる。第一の方法は、2 次元セル空間の全セルに、砂糖を生成・保持するエージェントを配置するというものである。第二の方法は、2 次元セル空間上の砂糖を管理する環境エージェントを作成するという方法である。ここでは、実装の容易さと実行負荷を考慮して、後者の方法によってモデル化することにする。

ここでは、Epstein and Axtell (1996) の中でも最も基本となる無限再生モデルを再現することにしたい。

7.4.1 Sugarscape モデル

Sugarscape モデルの全体像は図 7.75 のようになる。Epstein and Axtell (1996) における「エージェント」を Agent エージェントとし、砂糖の山などの環境を制御するエージェントを Environment エージェントとする。Agent エージェントは、2 次元セル空間上の 1 セルに存在し、時間とともに移動する。Environment エージェントは、セル上のどこにも存在しないが、Agent エージェントと関係を持ち、砂糖の取引を行うことができる。Agent エージェントが配置される 2 次元空間は、Space クラスを継承した CellSpace クラスを作成して表現する。モデルの初期設定で、エージェントが配置される格子状で端がトーラスとなっている Cell の空間を定義し、各 Cell に Sugar という財を配置する。CellSpace クラスを用いて、Cell クラスと Agent クラスを関連付けることによりエージェントのセルへの配置を表現する。また、この関連付けを変えることで、エージェントの移動を表現する。各セルの砂糖の量は、Environment エージェントのもつ FieldInformation によって保持される (図 7.76)。

このシミュレーションの流れは、次のようになる (図 7.77)。まず最初に TimeEvent を受け取るのは、Priority が高く設定されている Environment エージェントである。AddSugarBehavior (図 7.78) で TimeEvent を受け取った Environment エージェントは、あらかじめ設定されているターンあたりの Sugar の回復量に応じて、セル上に配

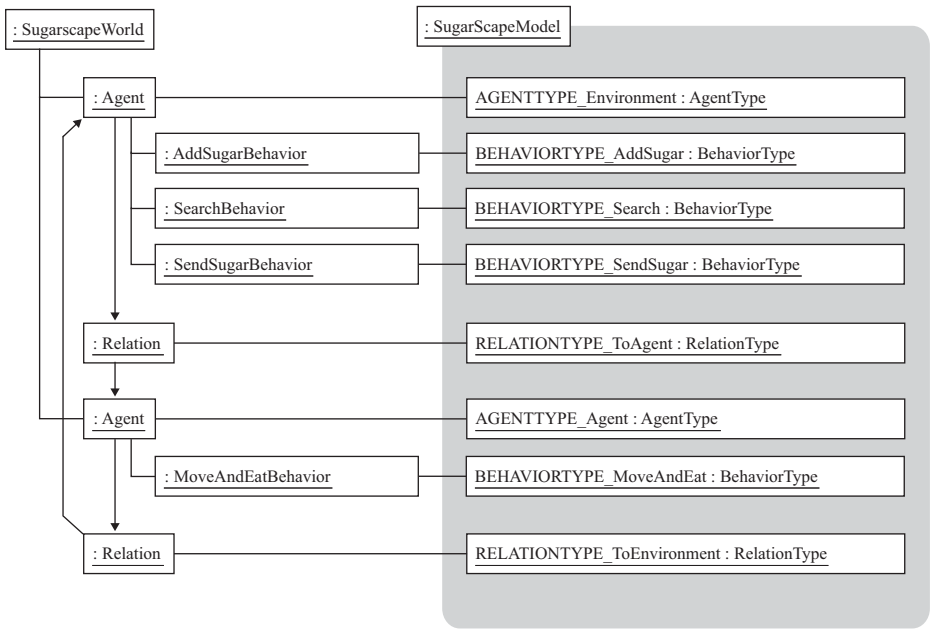


図 7.75: Sugarscape モデルの全体像

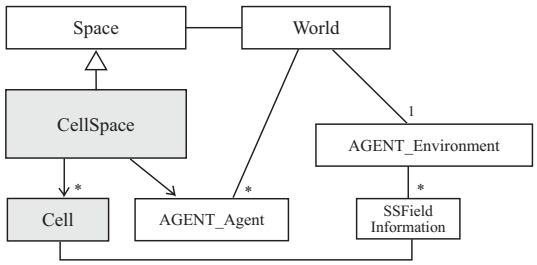


図 7.76: Sugarscape モデルのための CellSpace クラス

置された Sugar の量を回復させる。

次に TimeEvent を受け取るのは、Agent エージェントである。MoveAndEatBehavior (図 7.79) で TimeEvent を受け取った Agent エージェントは、自分の周囲の Cell に Sugar がどれだけあるのかを Environment エージェントに質問する。

Environment エージェントは、SearchBehavior (図 7.80) で質問を受け取り、Agent エージェントの視界に応じて、周囲の Cell の Sugar の配置状況を答える。Agent エージェントは、MoveAndEatBehavior で返答を受け取り、Sugar が最も多い Cell の方向へ移動し、移動先の Cell に配置された Sugar を Environment エージェントに要求する。Environment エージェントは要求を SendSugarBehavior (図 7.81) で受け取ると、Agent エージェントのいる Cell に配置されている Sugar をすべて渡す。Agent エージェントは、MoveAndEatBehavior で Sugar を受け取り、所有財としてもっておく。

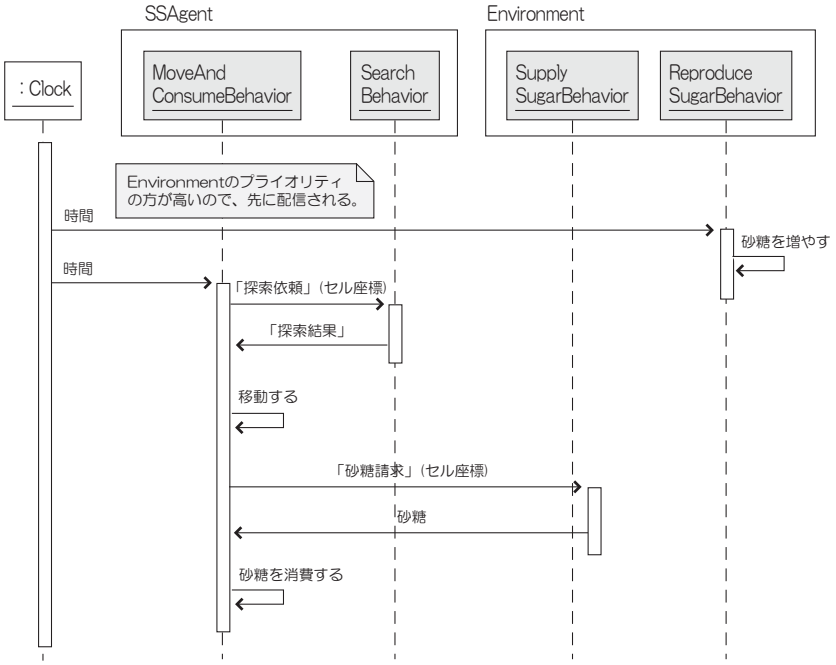


図 7.77: Sugarscape モデルのシーケンス図

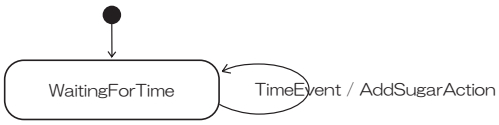


図 7.78: Sugarscape モデル: AddSugarBehavior

砂糖獲得後、Agent エージェントは、自分の代謝量に応じた Sugar を消費する。もし Sugar が足りなければ、Agent エージェントは死亡し、モデル上から削除される。

このシミュレーションの結果は、図 7.82 のようになる。最初、全体に散在していた Agent エージェントは、時間が経つにつれて砂糖の山に集まってくる。また、山から遠くにいた Agent エージェントは死亡してしまう。

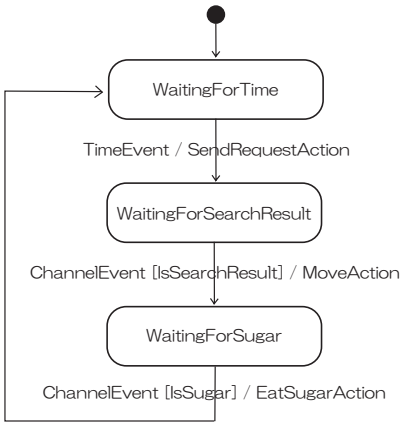


図 7.79: Sugarscape モデル: MoveAndEatBehavior

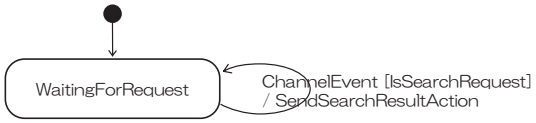


図 7.80: Sugarscape モデル: SearchBehavior

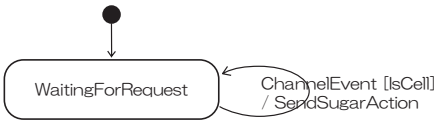


図 7.81: Sugarscape モデル: SendSugarBehavior

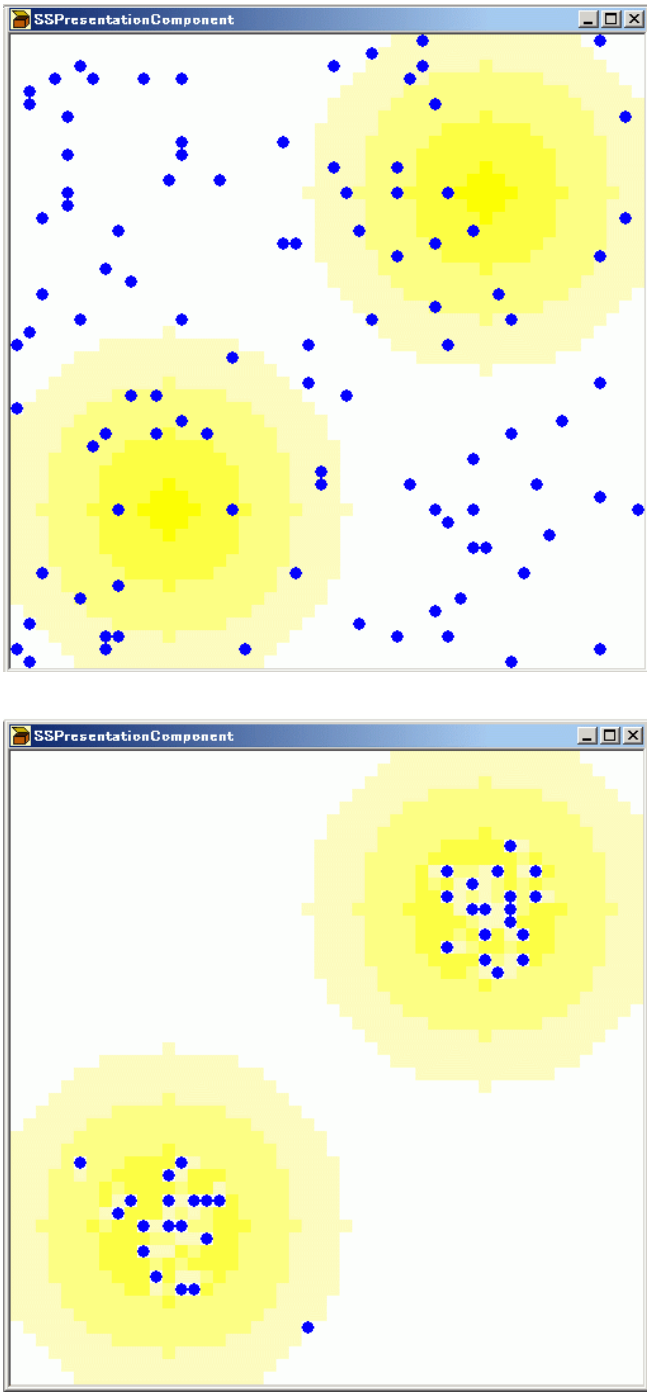


図 7.82: Sugarscape モデル: シミュレーション結果

7.5 人工株式市場モデル

最後に取り上げるモデルは、Arthur et al. (1996); Palmer et al. (1994) の人工株式市場モデルである。この人工株式市場では複数のトレーダーエージェントが、自分の戦略に従って株を売買し、その結果として株の価格などの市場全体の動きが決まる。そして結果に応じてエージェントは自分の戦略を適応的に変更していく。この研究では、「合理的期待論」に代替する新しい「進化経済学」のアプローチを展開しようとしている。このアプローチでは、得られる情報が不完全であり、かつ問題の文脈がわからない状況において、エージェントが学習しながら行動するというモデルを構成する。

ここでは、Information で表した戦略によって振舞いが決まるモデルの例として、Arthur et al. (1996) の人工株式市場を再現することにしたい。

7.5.1 人工株式市場モデル

このモデルの全体像は、図 7.83 のようになる。市場には 1 種類の株式があり、この取引を StockExchange エージェントが管理している。Trader エージェントは、各期において自分の資産を、安全資産と株式に資産配分を行うが、株式保有者には各期ごとに配当が与えられ、安全資産には各期ごとに利子がつく。これらを実現するために、Company エージェントと RiskFreeSecuritySupplier エージェントが存在する。また、市場の状況を調べて伝えるための Press エージェントがいる。

Trader エージェントは、それぞれがもつクラシファイアシステムによって予測・学習を行う。クラシファイアシステム (Holland, 1986; Goldberg, 1989) は、強化学習 (Sutton and Barto, 1998) の一種であるため、行動の有効度を伝える報酬 (強化信号) が必要となるだけで、最適な行動を直接的に指示するような教師信号を必要としない。そのため、エージェントが複雑な環境に自律的に適応するためのメカニズムとして注目されている。クラシファイアーシステムの条件部は市況を識別する。この市況は 12 桁の 2 進数で表される。

- 1-6：$Currentprice \times interestrate/dividend > 0.25, 0.5, 0.75, 0.875, 1.0, 1.125$
- 7-10：$Currentprice > MA(t,5), MA(t,10), MA(t,100), MA(t,500)$
- 11：always on 1
- 12：always on 0

但し、

$$MA(t,m) = \sum_{\tau=0}^{m-1} p(t-\tau)/m$$

である。

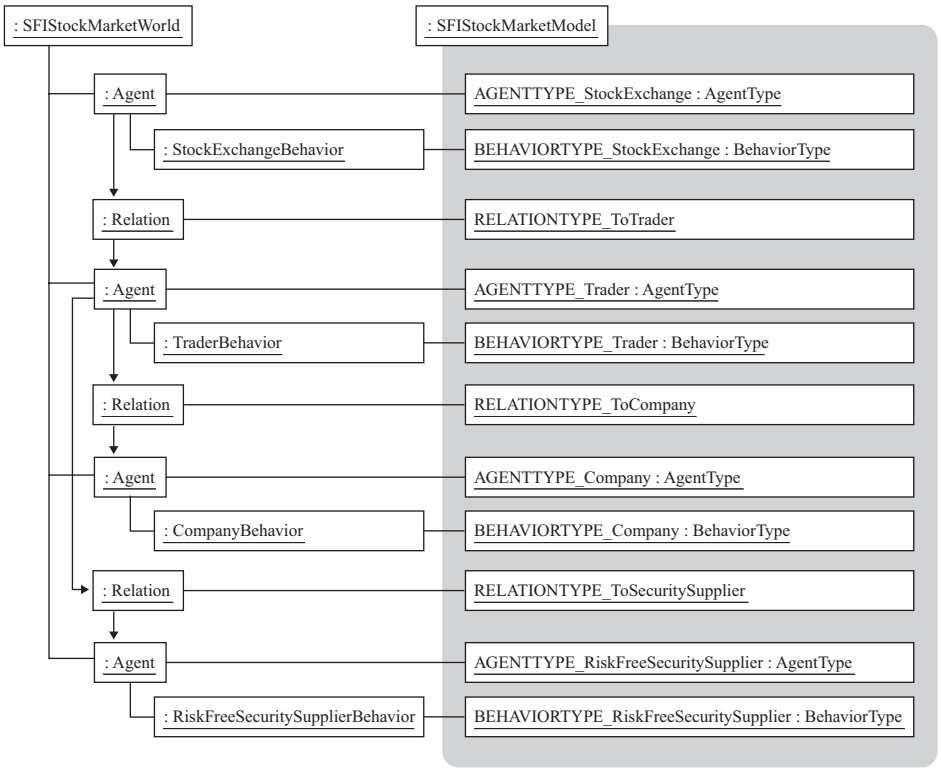


図 7.83: 人工株式市場モデルの全体像

クラシファイアーシステムの行動部は、その市場環境のもとでとるべき行動を示している。行動部には predictors（予測部）と呼ばれる整数の組 (a, b) が出力として出される。出された (a, b) を以下の線形予測式に代入することで、来期の予測を行うことになる。

$$E[p_{t+1} + d_{t+1}] = a(p_t + d_t) + b$$

多数の If Then ルールがあるとき、複数の If Then ルールが同時に起動する可能性が出てくる。その際は、If Then ルール毎に与えられている強度に比例した確率でルールを選択することになる。ルールの条件部と市場状況が合致するたびに、合致したすべてのルールの強度を更新することで、学習が行われる。

このシミュレーションの流れは、次のようになる (図 7.84)。まず最初に TimeEvent を受け取るのは、Priority が一番高く設定されている StockExchange エージェントである。StockExchange エージェントは、StockExchangeBehavior (図 7.85) によって、すべての Trader エージェントに売買注文を提示するように連絡する。Trader エージェントは、連絡を TraderBehavior (図 7.86) で受け取ると、クラシファイアにもとづいて、売買注文を StockExchange エージェントに返答する。StockExhange エージェントは、すべての Trader エージェントから注文を受け取ると、売り注文と買い注文の量から、

今期の株価と各 Trader エージェントが実際に取引できる量を計算し、その結果を各 Trader エージェントに連絡する。Trader エージェントは、その結果を TraderBehavior で受け取り、売り注文の場合は株を、買い注文の場合はお金を、StockExchange エージェントに支払う。StockExhange エージェントは、Trader エージェントから送られてきたすべてのお金と株を集計し、先ほどの結果に準じた対価を各 Trader エージェントに支払う。

取引が終わった後に、各 Trader エージェントは、RiskFreeSecuritySupplier エージェントに利子を要求する。RiskFreeSecuritySupplier エージェントは、RiskFreeSecuritySupplierBehavior(図 7.87) で要求を受け取ると、Trader エージェントの保有する安全資産の量に応じた利子を支払う。Trader エージェントは、TraderBehavior で利子を受け取る。

次に、各 Trader エージェントは、Company エージェントに配当金を要求する。Company エージェントは、CompanyBehavior(図 7.88) で要求を受け取ると、Trader エージェントの持ち株数に応じた配当金を支払う。Trader エージェントは TraderBehavior で配当金を受け取る。配当金を受け取った後、各 Trader エージェントは、複数の条件文が起動した場合の選ばれやすさを決める条件文の強度を更新する。

StockExchange エージェントの次に TimeEvent を受け取るのは、Priority が 2 番目に高く設定されている Company エージェントである。Company エージェントは、各 Trader エージェントに支払う配当金の額を切り替える。

最後に TimeEvent を受け取って行動するのは、Priority の最も低い Press エージェントである。TimeEvent を PressBehavior で受け取った Press エージェントは、StockExchange エージェントに過去の株価の推移について質問する。StockExchange エージェントは、質問を StockExhangeBehavior で受け取ると、過去の株価の推移について答え、Press エージェントはその返答を記憶する。次に、Company エージェントに過去の配当金の推移について質問する。Company エージェントは、質問を CompanyBehavior で受け取ると、過去の配当金の推移について答え、Press エージェントはその返答を記憶する。最後に、RiskFreeSecuritySupplier エージェントに安全資産の利息について質問する。RiskFreeSecuritySupplier エージェントは、質問を RiskFreeSecuritySupplierBehavior で受け取り、安全資産の利息について答え、Press エージェントはその返答を記憶する。

その後、Press エージェントは、現在の世界の状況をの 2 進数の文字列に変換し、それをすべての Trader エージェントに伝える。Trader エージェントは、ReceiveClassifierBehavior で連絡を受け取ると、それを現在の世界の状況として記憶しておく。

シミュレーションの結果は、図 7.89 のようになる。

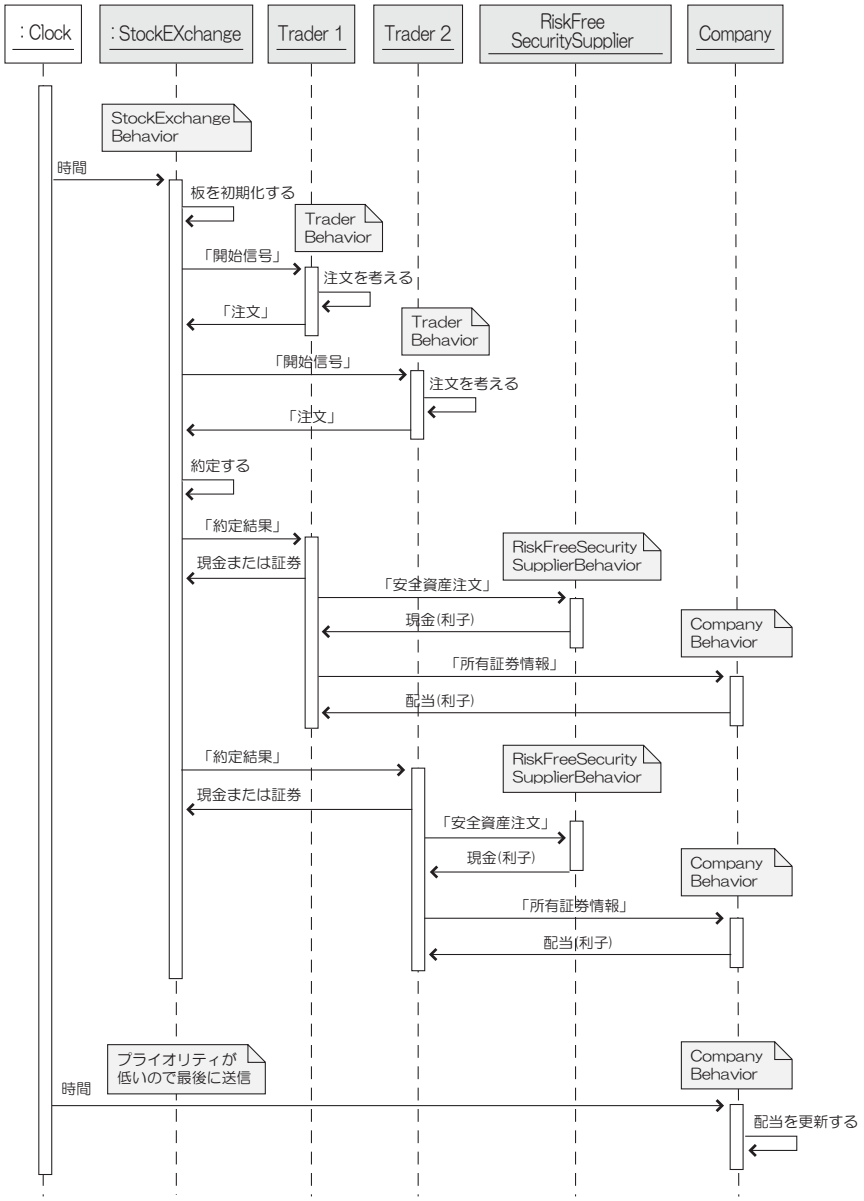


図 7.84: 人工株式市場モデルのシーケンス図

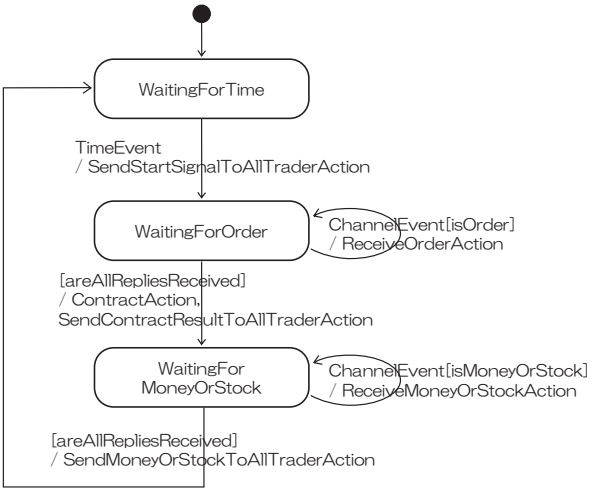


図 7.85: 人工株式市場モデル: StockExchangeBehavior

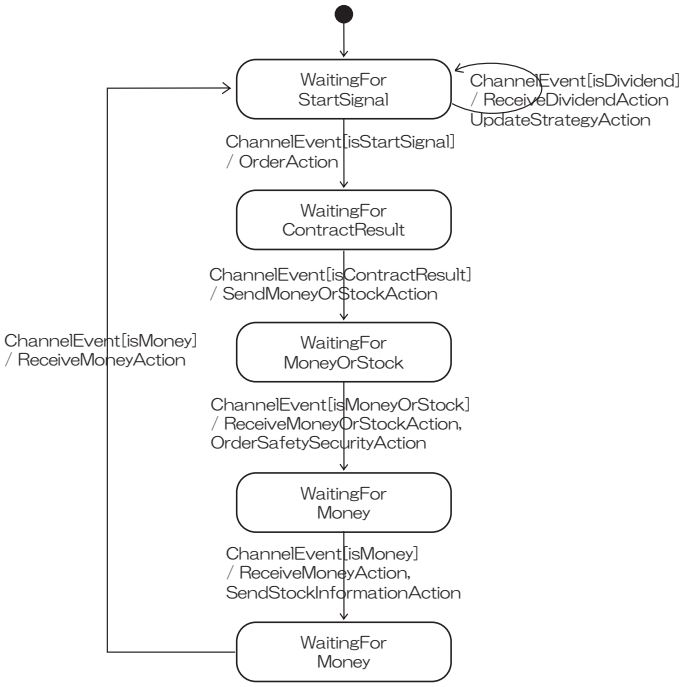


図 7.86: 人工株式市場モデル: TraderBehavior

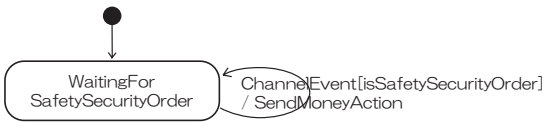


図 7.87: 人工株式市場モデル: RiskFreeSecuritySupplierBehavior

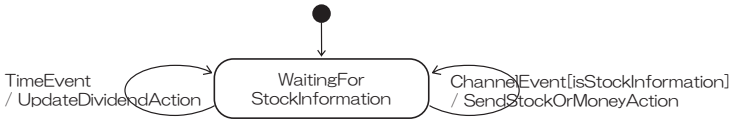


図 7.88: 人工株式市場モデル: CompanyBehavior

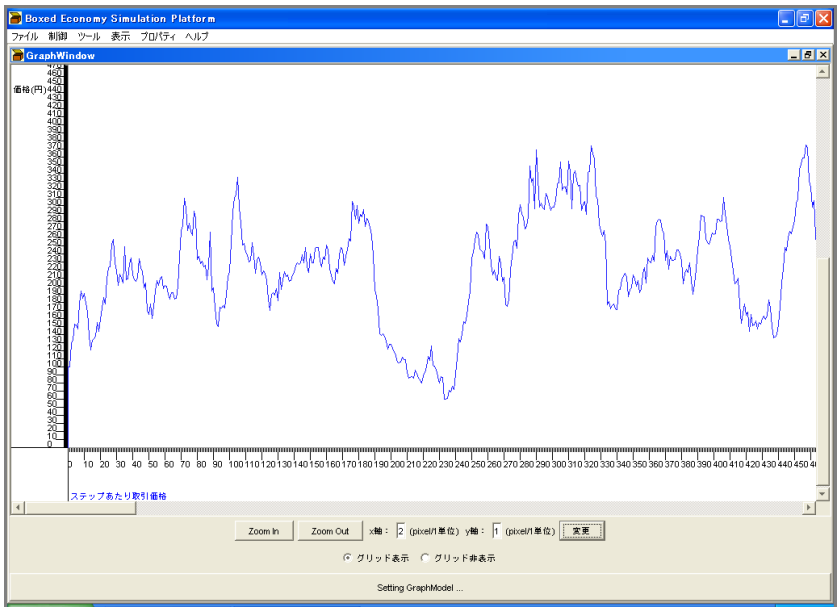


図 7.89: 人工株式市場モデル: シミュレーションの実行画面