第4章　モデル・フレームワークの提案

4.1　モデル・フレームワークとは

本章では、複雑系として社会・経済をモデル化する際によく現れる要素や構造を抽出し、「モデル・フレームワーク」として定義する。ここでは、抽象度の違いにより、「概念モデル・フレームワーク」と「シミュレーションモデル・フレームワーク」という2種類のフレームワークを考える。これらのフレームワークは一貫性を有しており、概念モデルの作成からシミュレーションモデルの実装までをシームレスに行うための支援をする (図 4.1)。

4.1.1　概念モデル・フレームワーク

概念モデル・フレームワークは、対象についての概念モデルを作成する際に、共通して登場する要素と構造を定義したものである。モデル作成者は、モデル化しようとしている対象が「どのようなものであるか」(What) を洗い出し、記述する際に、この概念モデル・フレームワークを用いることができる。加えて、概念モデル・フレームワークがシミュレーションモデル・フレームワークと一貫性を有することから、シミュレーションモデルへの移行をシームレスに行うことができる。概念モデル・フレームワークは、(1) 現実世界の認識のための準拠枠の明示化、(2) 概念モデルを記述するための語彙の提供、(3) 概念モデルの共有化とコミュニケーションの支援、という3つの役割を果たす (図 4.2)。

(1) 現実世界の認識のための準拠枠の明示化

概念モデル・フレームワークは、対象となる現実世界を認識する際の準拠枠となる。概念モデル・フレームワークを、「世界の中に含まれるものを拾い集めようとして用いる、一種のふるい」(William, 1925) として用いることにより、動的で捉えどころのない世界を把握することが容易になる。

私たち人間は、現実世界からありのままの「事実」というものを受動的に受けとっているのではなく、認知枠のフィルターを通じて能動的に選びとっている。生体的・認知的制約から現実世界のすべてを認識することはできないため、重要と思われる部分に焦点を当てて把握しているのである。このように、「知覚する」という行為は「枠

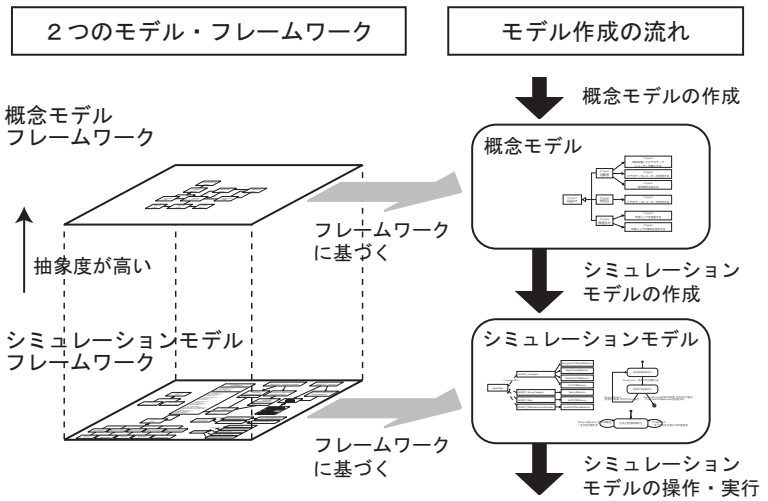

図 4.1: ２つのモデル・フレームワークとモデル作成の流れ

組みのなかへの定位」(村上, 1971) を意味している。前提をおくことによってそれが認識の枠組みとなり、動的で捉えどころのない世界をある側面から把握できるようになるのである(37)。

(2) 概念モデルを記述するための語彙の提供

概念モデル・フレームワークにおけるモデル要素は、モデルを記述するための語彙として用いることができる。認識や分析で得られた概念は、それを記述するためのなんらかの表現手段が必要となるが、概念モデル・フレームワークは、そのドメインに特化した語句と、それらの可能な組み合わせの規則を提供する。

認識や分析で得られた概念は、そのままでは形がないため、それを記述するためにはなんらかの具体的な表現手段が必要である。適度の長さと複雑さをもつ表現を用いて思考し表現するのであるが、それを効率よく運用するために、表現の意味を記号に圧縮する (藤村, 1999)。このような言語——ここでの対象としてはモデル化のための言語——は、表現単位となる語句(38)と、その組み合わせの規則(39)とによって成り立っている。このような対象世界を表現するための体系によって、私たちは安定的に、あらかじめ定められた有限個の記号を用いて、並べ方の形式に従って組み合わせて表現することができるようになる。

(3) 概念モデルの共有化とコミュニケーションの支援

概念モデル・フレームワークは、複数のモデル作成者に共通の視点を与えるため、モデル要素の粒度や捉え方がモデルごとに異なってしまうという問題を回避できる。

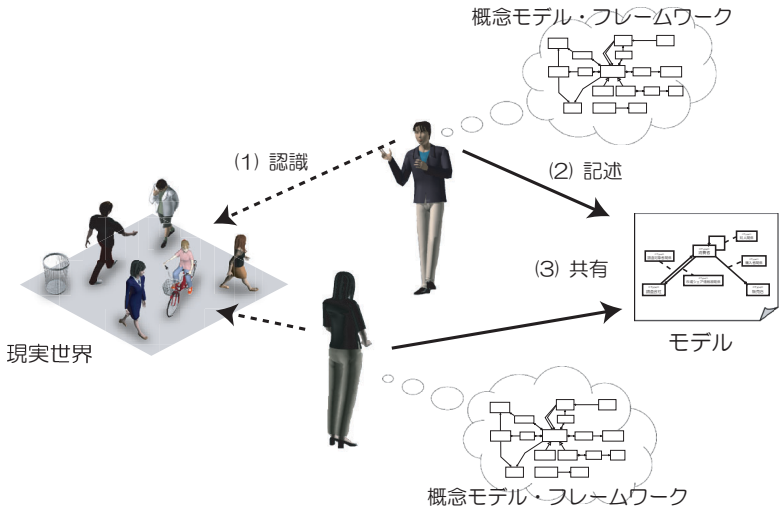

図 4.2: モデル作成における概念モデル・フレームワークの役割

また、共有された語彙を用いることで、詳細な情報を伝達することなく重要な点を強調することができるため、円滑で効率的なコミュニケーションが可能となる。

一般にコミュニケーションは、発信者が決まりに従って表現し、受信者はその決まりによって読みとって理解する。この決まりを「コード」と呼ぶのであるが、「理想的なコミュニケーションでは、話し手と聞き手が「メッセージ」の作成と解読に際して利用する「コード」は同一のもの」(池上ほか, 1994) である場合で、特に科学的コミュニケーションにおいては重要となる[40]。共通の枠組みを用いることで、認識・分析して得られたものをそのまま記述し、他者へ伝達し、知的蓄積へ積み上げていくことができるのである。

4.1.2 シミュレーションモデル・フレームワーク

シミュレーションモデル・フレームワークは、得られた概念モデルを、シミュレーションモデルとして「どのように実現するか」(How) を規定するものである。シミュレーションモデル・フレームワークは、ソフトウェア・フレームワークとして実行環境の一部となることで、シミュレーションモデルを実行することができる。シミュレーションモデル・フレームワークは、(1) シミュレーションモデルへの変換方法の明示化、(2) シミュレーションの実装の支援、(3) シミュレーションモデルの共有化と再利用の支援、という 3 つの役割を果たす。主に次の 3 つの役割を果たす (図 4.3)。

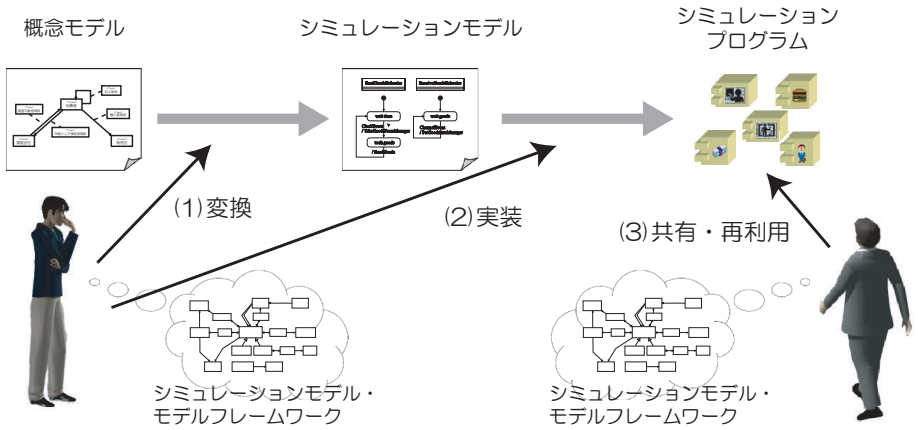

図 4.3: シミュレーション作成におけるシミュレーションモデル・フレームワークの役割

(1) シミュレーションモデルへの変換方法の明示化

シミュレーションモデル・フレームワークは、シミュレーションモデルとして何をどのように記述すべきかを明らかにする。概念モデル・フレームワークと一貫性があるため、概念モデルをシミュレーションモデルに変換する手続きを事前に規定することができる。

(2) シミュレーションの実装の支援

シミュレーションモデル・フレームワークは、シミュレーションモデルの実装の方法を明らかにする。そして、シミュレーションモデル・フレームワークは、シミュレーションの基本的な仕様の定義を行うため、シミュレーションの実行に関するプログラムや環境をあらかじめ用意しておくことが可能となる。また、一部のプログラムを自動生成する作成支援ツールを開発することができるようになる。

(3) シミュレーションモデルの共有化と再利用の支援

シミュレーションモデル・フレームワークは、モデルコンポーネント間の仕様やそれらの接続方法を規定するため、モデルコンポーネントを共有したり、再利用したりするための仕組みを提供する。

4.2 提案モデル・フレームワーク: Boxed Economy Foundation Model (BEFM)

複雑系として社会・経済のモデルを作成するためのモデル・フレームワークとして、「Boxed Economy Foundation Model」(以下、BEFM) を提案したい[41]。以下では、概念モデルの作成を支援する「BEFM 概念モデル・フレームワーク」と、シミュレーションモデルの作成を支援する「BEFM シミュレーションモデル・フレームワーク」について、順に説明する。

4.2.1 BEFM 概念モデル・フレームワーク

BEFM の主な特徴としては、エージェント間の相互作用を、財 (情報が付随することがある) のやりとりとして明示化するという点と、エージェントの行動を、エージェントとは別のモデル要素として定義するという点である。BEFM 概念モデル・フレームワークは、World、Space、Clock、Entity、Agent、Goods、Information、Behavior、Relation、Channel などのクラス (型) から構成されている (図 4.4)。各クラスについてまとめると以下のようになる。

World, Space, Clock

対象世界を表現する土台が「World」である。World は、その世界に固有の空間と時間によって規定されている。空間は「Space」によって、その世界の地理的な構造が表される。また、不可逆的な時間の流れを表すために「Clock」があり、この時間が経過することで現象が進行する。World には、後述の Entity、すなわち Agent と Goods が配置される。

Entity, Agent, Goods

世界に存在する実体が「Entity」である。Entity には、Agent と Goods の 2 種類があり、どちらにも後述する Information を付随させることができる。

社会・経済において、さまざまな活動を行う個人や社会集団 (企業・政府・家族・学校・地域社会・国) が「Agent」である。また、動的に変化する環境や「モデルの外部」なども Agent として表現することがある。Agent は Goods を所有し、その行動 (振る舞い) は、後述するように Behavior として定義する。

Agent に所有し交換される有形/無形のものが「Goods」である。BEFM における「Goods」とは、人間の欲求を充足する性質をもつという経済学における狭義の意味ではなく、世界におけるさまざまなものを示す広義の概念として用いている。例えば、

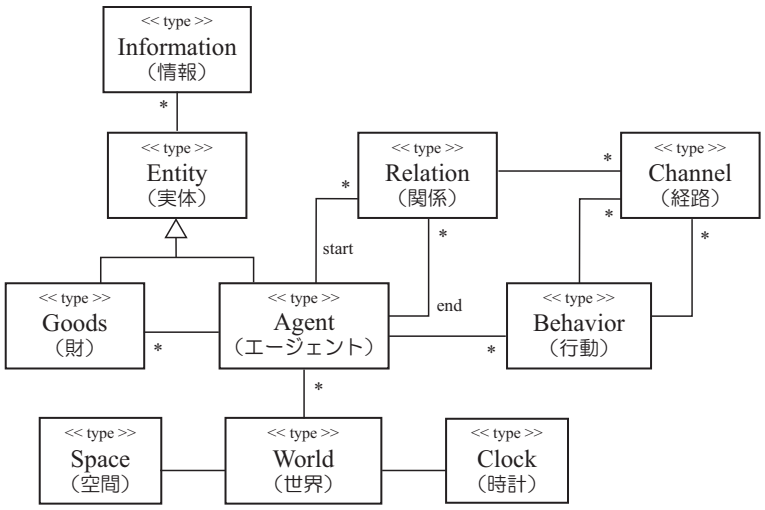

図 4.4: BEFM 概念モデル・フレームワークのクラス図

自動車、石油、トウモロコシ、株、土地の権利、書籍、広告、日記、回覧板、水、声、騒音、ごみ、貨幣などは、どれも Goods である。

Information

Entity が保持する情報は「Information」として表される。例えば、Agent が記憶した情報や、Goods に付随して取引される情報などが、Information である[(42)]。Information は単独では存在せず、必ず Entity、つまり Agent や Goods によって保持されている。Agent が保持する Information は、主体の内部に貯蔵されている「記憶」や「遺伝子」をはじめとして、Agent のさまざまな属性を表現する。Goods に Information が付随するというのは、例えば、新聞は「紙」(Goods) に「記事内容」(Information) が付随したものであり、会話は無形で瞬間的な「声」(Goods) に「会話内容」(Information) が付随したものと捉えるということである。Goods に付随している Information は、その Goods が他の Agent に渡されると、Goods とともに送られる。また、Agent によって保持されている Information は、すでに持っている Goods か、そのために作成した Goods に付随させて、他の Agent に送ることができる。

Behavior

エージェントの行動は、「Behavior」として表される。例えば、企業における生産行動や販売行動、個人における購買行動や労働行動などは、どれも Behavior である。後述するように、オブジェクト指向によってエージェントをモデル化する場合には、エージェントの行動を Agent クラスの操作として記述するのが一般的であるが、BEFM で

は行動をモデル要素の 1 つとして分離する。これは、状況によって振る舞いが動的に変化することを表現できるようにするためである。

BEFM では、Agent は複数の Behavior を並列的に実行することができる。内部状態を各 Behavior にもたせることで、複数の行動の多様な組み合わせを実現することができる。

Relation, Channel

ある Agent から他の Agent への関連性は、「Relation」によって表される。これにより、友人関係や家族関係、雇用関係などの関係を表現することができる。実際のコミュニケーションの際には、この Relation に基づいて開設されるコミュニケーション・パスである「Channel」を通じて、商品や会話、貨幣などの Goods をやりとりする。

Relation と Channel は、密接に関わっている概念であるが、これらは別のものである。Relation は「参加」という「構造的関係」(西部, 1997) を表しており、Channel は「活動」という「過程の関係」(西部, 1997) である。Relation は Agent 間の関連性を表すが、それに基づいて開設される Channel は、Behavior 同士を接続する。同一の Agent 内における Behavior 間のやりとり[43]であっても、Channel 上の Goods のやりとりで表現されるので、Behavior どうしのやりとりはすべて、Channel を通じた Goods の取引として抽象化されることになる。

4.2.2 BEFM シミュレーションモデル・フレームワーク

BEFM シミュレーションモデル・フレームワークは、BEFM 概念モデル・フレームワークと一貫性をもつように設計されている。また、このフレームワークに基づいて作成されたモデルは、Boxed Economy Simulation Platform 上でシミュレートすることができる。

BEFM シミュレーションモデル・フレームワークでは、概念モデルのモデル要素のそれぞれに対し、種類の違いをどのように表現するのかの変換方法を規定している[44]。種類の表現方法としては、「継承」による方法と「パワータイプ」による方法の 2 種類がある。

継承による方法は、クラスを特化したサブクラスを定義することで、種類の違いをクラスレベルで実現するものである。この方法を用いるものには、Behavior と Information がある。この 2 つのモデル要素は、種類の違いが、単なる属性値の差異ではなく、振る舞いや内部構造の差異であるためである。

パワータイプによる種類の表現は、種類クラスを作って、その種類クラスのインスタンスレベルで多様性を実現するというものである (Martin and J.Odell, 1995; Fowler, 1996)。この方法によって種類を表現するものは、Agent と Goods、そして Relation

である (表 4.1)。この場合、新たにクラスを作成することなく、インスタンスレベルで差異を表現することができる(45)。BEFM シミュレーションモデル・フレームワークでは、後述する「Type」クラスをパワータイプとして用いることができる。

以下では、BEFM シミュレーション・プラットフォームの概要を説明する (詳細については、付録 B を参照してほしい)。

Agent

BEFM シミュレーションモデル・フレームワークでは、エージェントの行動を、Behavior オブジェクトとして外部化し、「オブジェクトコンポジション」によって Agent オブジェクトに付加するという方法を採用している。オブジェクトコンポジションとは、役割を外部化するためのオブジェクトを用意して振る舞いを委譲し、そのオブジェクトを実行時に関連づけるという設計のことである (Gamma et al., 1995; Coad and Mayfield, 1999)。同様に、エージェントの記憶 (情報) や、他のエージェントへの関係も、Information や Relation のオブジェクトコンポジションとして保持する設計になっている。

このような設計により、シミュレーションモデルにおけるコーディング作業は、Behavior や Information の記述が中心となり、それらの組み合わせ方によって、エージェントが設定できるようになる。この場合、Agent 自体はそれらの行動を束ねる役割を果たしているにすぎない。例えば、Agent をインスタンス化すると、単なる Agent オブジェクトが得られるが、そこに PurchaseBehavior を加えると、購買行動を行う「消費者エージェント」となる。このようなエージェントの設計は、新しい行動の追加や削除、そして行動の組み換えなどを簡単に行えるという柔軟性がある。また、既に作成されている Behavior を利用することも可能になるため、再利用性を考慮した設計といえる。

Agent のもつ主な機能は、TimeEvent の受信、OpenChannelEvent の受信、Behavior の追加・削除・取得、Goods の追加・削除・調査、Relation の追加・削除・取得である。

表 4.1: 各モデル要素の作成方法

基礎モデル要素	作成方法
Agent	AgentType を追加
Goods	GoodsType を追加
Relation	RelationType を追加
Information	Information インターフェースを実装
Behavior	Behavior クラスを継承（コンポーネントビルダーによって自動化）
World	World クラスを継承
Clock	AbstractClock を継承（StepClock、RealClock を利用可能）
Space	Space インターフェースを実装（CellSpace を利用可能）

Agent は、時刻の経過を示す TimeEvent を受信し、もっている Behavior に配信する。また、経路の開設を求める OpenChannelEvent を受信した場合には、経路開設を試みる。成功すれば、適切な Behavior に対して開設された Channel を返す。Agent は、BehaviorType を指定することで Behavior の追加や取得、削除をすることができる。追加された Behavior は自動的に開始状態となる。また、Goods の追加や、GoodsType をもとにした Goods の取り出しや Goods の量を調査が可能である。そして、Relation の追加や Type による検索・削除を行うことができる。

Behavior

BEFM シミュレーションモデル・フレームワークでは、エージェントの持つそれぞれの Behavior を「状態機械」として定義する。状態機械とは、何らかのトリガーとなるイベント (影響を及ぼすさまざまな出来事) を受け取ると、現在の状態に応じた「アクション」(動作) を行い、新しい状態へ遷移するシステムである。ある時点をみてみると、Behavior オブジェクトは、必ずどれか 1 つの状態にとどまっている。Behavior の状態遷移を引き起こすイベントには、時間が経過したことを表す「TimeEvent」と、Goods が送られてきたことを表す「ChannelEvent」がある。

このような設計により、外界のイベントの種類と、現在の自分の状態によって、振る舞いが異なるというモデルを実現できる。本来、システムの内部状態とは、システムのすべてのパラメータの値の集合のことであるが、状態機械では、それらの状態のうちの一部を意識的に切り出して注目することになる。このことは、本論文で目指している複雑系 (状態の変化によって振る舞いのルールが変化するシステム) のモデルを記述する際に、モデルの状態の複雑さを隠蔽し、注目すべき状態を強調することができるのである。

エージェントが複数の行動を並列的に動作させる場合には、その内部状態は複雑にならざるを得ないが、BEFM に基づくモデル化では、Behavior という分かりやすい単位ごとに状態を把握することができる。また、Behavior の多くの詳細な部分はユーザーには見えない形で隠蔽されているので、状態機械としての Behavior の動作部分の実装については、ユーザーはほとんど意識する必要はない[46]。

Event

Behavior が受け取る Event には、代表的なものとして「TimeEvent」と「ChannelEvent」がある。TimeEvent は、時間の経過を知らせる Event である。モデル外部から時刻が経過したことを知らせるために Agent に送られて、Agent が Behavior に転送する。ChannelEvent は、ほかの Behavior から Goods が送られたことを知らせる Event である。

このほか、通常用いないが用意されている Event もある。「OpenChannelEvent」は、Relation から Agent を通して受け取る Event である。明示的に書かなくても受け取って Channel を開設するが、あえて明示的にこの Event を受け取るように記述することもできる。「(AutoTransition)」は、状態機械において遷移は外部からの刺激のみによるという定義があるため推奨はしないが、Event を受け取らなくても自動的に状態が遷移するように記述することも可能である。これは条件によって分岐させる場合などに用いることができる。

Type

BEFM シミュレーションモデル・フレームワークでは、モデルに存在するオブジェクトの要素を分類するための識別子として、「Type」クラスが用意されている。Type には、AgentType、GoodsType、InformationType、RelationType、BehaviorType の 5 種類がある (図 4.5)。Type の導入により、柔軟な検索・取得・識別が可能となる。例えば、同じ振る舞いをする Agent であっても、異なる Type が付加してあれば、別々に識別可能となる。

Type は、複数の Type 間に親子関係を定義することができるので、上位概念・下位概念を表現することができる。これによって、異なる Type をもつものであっても、同じ親 Type をもつものどうしであれば、一括して扱うことができるのである。例えば、図 4.6 左は、「ビデオ」には「VHS 規格のビデオ」と「Beta 規格のビデオ」があるということを表している。このような概念間の関係性を定義することによって、エージェントにその Goods の種類についての複雑な知識をもたせることができるようになる。例えば、市場におけるビデオの普及率を知りたい場合には、GoodsType「ビデオ」を親タイプとしてもつ Goods の数を調べればよいし、各規格の市場シェアを知りたい場合には、GoodsType「VHS 規格のビデオ」と GoodsType「Beta 規格のビデオ」の Goods の数をそれぞれ調べればよい。また、図 4.6 右は、「ビデオ一体型テレビ」が「テレビ」でもあり「ビデオ」でもある、ということを表している。あるエージェントが GoodsType「ビデオ一体型テレビ」の Goods をもっている場合、そのエージェントは、「ビデオ一体型テレビを所有しているか」という質問にはもちろんのこと、「テレビを所有しているか」や「ビデオを所有しているか」という質問にも、正しく答えることができるようになるのである。

World

World は、まず、モデルにおける時間・空間を定義するためにそれぞれただ 1 つの Clock、Space を持つ。World は Agent を配置してそれらを管理し、それらの追加、削除、生成、取得を行うことができる。また、RandomNumberGenerator を同様に

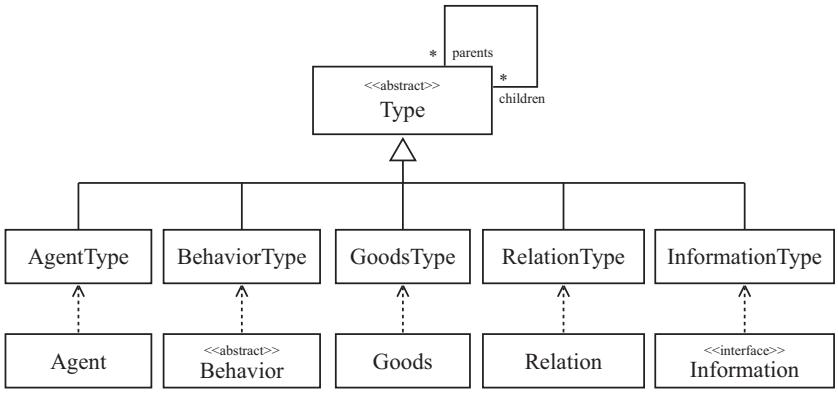

図 4.5: Type とその関連クラス

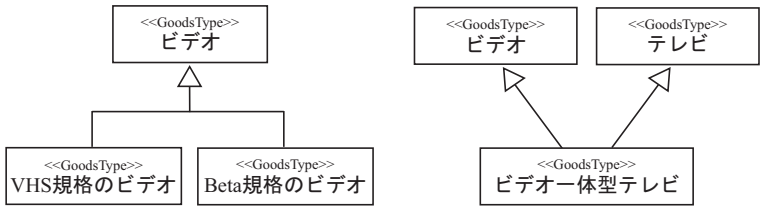

図 4.6: Type の継承の例

複数もつことができる。RandomNumberGenerator は、モデルで乱数を利用するときに用いるクラスである。コンピュータは、その構造上完全にランダムな数字を生成することはできないが、複雑なアルゴリズムによって事実上乱数と捉えて差し支えない数字 (擬似乱数) を生成することができる。乱数生成にはいろいろな方法があるため、モデルやモデル作成者によって利用したい方法も異なると考えられる。このことを考慮し、RandomNumberGenerator は、乱数として数字を生成するアルゴリズムを内包して、さらに値を取得するためのメソッドを持つクラスと定義されている。RandomNumberGenerator は、World 上に同時に複数存在できる。その場合、RandomNumberGenerator は名前によって識別され、Behavior などから名前で取得されて、利用されることになる。

モデル作成の際には、World クラスを継承したクラスで、initializeWorld() および initializeAgents() をオーバーライドし、Agent の配置や Relation の構築、Behavior の追加を行う。

Goods

Goods は、種類 (GoodsType)、量 (GoodsQuantity)、付随情報 (Information) からなるオブジェクトとして定義する。Goods の Quantity を直接変更することはできず、必ず Agent のメソッドを用いて分割や結合を行う。

Entity

Entity は Agent、Goods の親クラスである。これらの 2 つのクラスの共通の性質である Information を管理するインターフェースを持っている。

Information の管理はハッシュテーブルによって行われる。ハッシュテーブルにおけるキーも Information である。これにより、Information をキーとして Information を格納し、取り出すことができる。また、取り扱いを簡単にするために、自動的に取り扱いたい Information の InformationType をキーとして操作するメソッドが追加されている。

Information

Information の実装は Information インターフェースを実装する必要がある。Information インターフェースは空インターフェースである。メソッドは何も定義されていないので、それ以外は完全に作成者に依存している。

Relation と Channel

Channel の作成（経路の開設）は Relation の openChannel() メソッドを呼び出すことによって行われる。openChannel() は BehaviorType を引数にとなる。経路開設では、Relation の先の Agent の Behavior の中から BehaviorType に該当する Behavior を探し、その Behavior と経路を開設する。開設した経路 (Channel クラス) は openChannel() の返り値として返される (ただし、この部分の多くは Behavior の持つメソッドで隠蔽されているので Relation の openChannel（）メソッドを呼び出す必要はない)。

また、Channel の持つパラメータとして keep パラメータがある。これは財送信後その Channel が継続して存在するかを設定するものである。false の場合には、財送信・受信処理が完了すると Channel は自動的に close() が呼ばれる。true の場合には、財送信・受信処理後も Channel オブジェクトは存続し、Behavior から取得することができる。

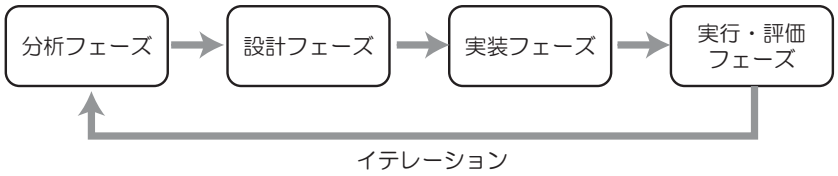

図 4.7: BEFM を用いたモデル作成プロセス

4.3 提案モデル・フレームワークを用いたモデル作成のプロセス

BEFM を用いたモデル作成プロセスは、オブジェクト指向開発プロセスに基づいて、「分析」、「設計」、「実装」「実行・評価」のスパイラルモデルとなる (図 4.7)。このモデル作成プロセスは、次のような流れで進められる (Boxed Economy Project, 2003; 松澤ほか, 2003)。

4.3.1 分析フェーズ

モデル作成プロセスの第一段階である「分析フェーズ」は、どのような対象領域のシミュレーションを行うのかを明らかにし、それを概念モデルとして記述するフェーズである。モデル化しようとしている対象が、「どのようなものであるか」(What) を明確化するために、BEFM 概念モデル・フレームワークを利用して、対象領域に登場する Agent、Information、Goods、Behavior、Relation をすべて洗い出して定義することから始める。まず最初にエージェントとその行動を明らかにし、それらのエージェントの関係について、「概念モデルクラス図」を記述する。そして、エージェントの行動のフローチャートを「行動アクティビティ図」として記述する。この過程で、登場する財や情報も洗い出して概念モデルクラス図に追加・修正していく。また、これらの分析をもとに、各行動の間でどのような相互作用 (財や情報のやりとり) があるかを確認し、その一つのシナリオを時系列に表現する「取引シーケンス図」を記述する。以上のような記述によってモデル要素の洗い出しと定義が終わるまで、このプロセスを繰り返していく。

4.3.2 設計フェーズ

モデル作成プロセスの第二段階である「設計フェーズ」は、 分析フェーズで作成された概念モデルをもとに、シミュレーションモデルの設計を行うフェーズである。概念モデルの各モデル要素について、シミュレーションとして動作させるための詳細を決めていくのである。まず、パワータイプを用いて種類を表現する Agent、Goods、Relation について、それぞれ AgentType、GoodsType、RelationType を定義する。

Behaviorは、Behaviorクラスを継承して詳細を定義するので、行動アクティビティ図と取引シーケンス図をもとに、Behaviorの状態遷移図を記述する。Informationには、Informationクラスを継承して詳細を定義するものと、InformationTypeだけでよいものという2種類がある。まず、Informationクラスを継承して詳細を定義するのは、どのような情報がどのような形式で格納されるかを定義しなければならないInformationの場合である。他の行動に対して依頼や質問をする際に使用するものなど、内容を含まない情報は、InformationTypeを定義するだけでよい(47)。次章でみるように、この設計フェーズは、記述を支援するツールを用いることができる。

4.3.3 実装フェーズ

モデル作成プロセスの第三段階である「実装フェーズ」は、設計フェーズで作成されたシミュレーションモデルを、Java言語を用いて実装するフェーズである。次章でみるように、Typeの実装は、私たちの提供しているタイプエディタを用いて行うことができる。また、Behaviorの実装は、私たちの提供しているコンポーネントビルダーからソースコードの雛型を生成し、その雛型の一部を埋めるようにして実装する。WorldおよびInformationの実装は、ソースコードを記述することで行う。

4.3.4 実行・評価フェーズ

モデル作成プロセスの第四段階である「実行評価フェーズ」は、実装フェーズで作成されたシミュレーションを、実行して評価するフェーズである。シミュレーションの実行は、次章で提案するシミュレーション・プラットフォーム「Boxed Economy Simulation Platform」を用いることができる。この段階で、シミュレーションが意図した通りに動作するかという正当性の検証も行う。そして、シミュレーションの設定をさまざまに変化させながら、シミュレーションの結果と現実の現象を比較し、妥当性の検証を行う。このとき必要があれば、分析フェーズに戻るなどして、次のイテレーションに入ることになる。

4.4 先行研究との比較

オブジェクト指向によってエージェントを表現する場合、現状ではエージェントをひとつのオブジェクトとして設計することが多い。例えば、Bruun (2002) は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として、図4.8のようなエージェントの設計を提案している。またAxtell (2002) などでも、エージェントを表すオブジェクトの操作としてエージェントの行動が定義されている。しかし、実はこのような設計は、エージェントをひとつのオブジェクトにカプセル化してしまうので、行

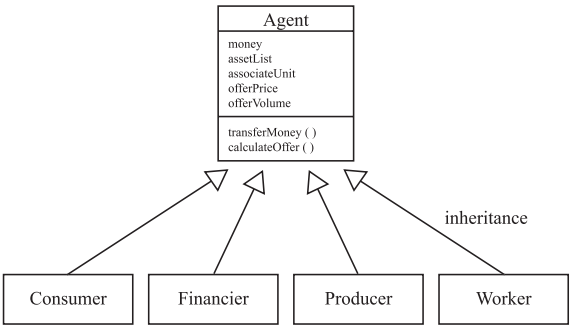

図 4.8: 一般的なエージェントの設計 (Bruun, 2002)

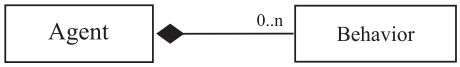

図 4.9: 提案モデル・フレームワークにおけるエージェントの設計

動の種類そのものが動的に変化するエージェントを作成する場合や、エージェントの一部を再利用する場合に限界が生じる。

これに対し、提案モデル・フレームワークでは、「継承」を用いる代わりに「コンポジション」を用いることで、柔軟性と再利用性を考慮した設計になっている。この設計では、エージェントクラスの中に操作として行動を記述するのではなく、エージェントの役割別に行動クラスを作り、それらのオブジェクトを「主体」の核となる部分が保持するという設計となる (図 4.9)。

このような行動のコンポジションによるエージェント表現は、モデルの意味的な側面における利点もある。それは第一に、ひとつのエージェントが複数の社会的役割を担っているということを自然な形で表現できるということである。例えば、「消費者エージェント」や「労働者エージェント」という実在しない主体をモデル化するのではなく、「個人」が時として消費者の役割を担ったり、労働者の役割を担ったりするという表現になるのである。このようなエージェントが複数の役割を担うというモデル化は、特に経済全体をシミュレートするような場合に不可欠となる。経済社会におけるエージェントの異質性というのは、結局のところ行動や役割が異なるということから生じるからである。存在するのはあくまで個人や社会集団であり、それらの行う行動が異なるがゆえに異なる種類のエージェントとして識別されるのである。

行動のコンポジションの第二の利点は、新しい行動の追加や削除、行動の手続きの変更などが可能になるということである。例えば、小売業が銀行機能の一部を担うようになるということを表すためには、小売業のエージェントが、銀行のもつ機能の一部を取り入れる必要がある。従来のようなエージェント単位の設計では、小売業エージェントや銀行エージェントというようにエージェントにその振る舞いをカプセル化

しているため、多重継承で拡張するか、銀行兼務小売業エージェントのようなものを新たに作成しなければならくなる。しかしこのような方法では、継承の階層が深く複雑化するため、長期的にみると限界がある。経済全体が分析対象の場合には、時間的にも長期となり、状況によってエージェントが成長して行動の種類を変更したりすることが考えられる。それゆえ、エージェント単位で定義するのではなく、それらの個別の行動ごとに分割して定義し、コンポジションによってエージェントを特徴づけていくという設計が不可欠となるのである。