ソフトウェア場概念によるオブジェクト指向システムのクラス抽出過程定式化とその検証

平成 16 年 9 月

大 木 幹 雄

目　　次

1章　序論

1.1　研究の背景

コンピュータを用いた情報システム開発において,「システム分析」の目的は，ユーザの要求から，情報システムの構成要素であるデータや機能を抽出し，分析に引き続き行われる「システム設計」に引き渡す「構成要素の構造や関連」を決定することにある．近年のオブジェクト指向を基盤にした情報システム開発では，システム分析において，クラス（＝変数集合と操作集合をひとまとめにしたオブジェクトのひな型.詳しくは付録 A,D を参照）の集合体やそれらの関連（＝継承や構成，関連の結合度等）を表現した「クラス構造」が決定すると，それに引く続くシステム設計では，決定したクラス構造を中心に，クラスが持つ機能の詳細化が行われる．それゆえ，オブジェクト指向情報システム開発では，システム分析で決定するクラス構造の良否が，それに引き続くシステム設計やプログラム分析設計，プログラミング･テスト等の工程に大きく影響を与え，最終的に開発システム全体の品質（＝システムの要求充足性，拡張性，変更容易性，効率性等）を左右することになる．しかしながら現状におけるシステム分析では，クラス構造の良否は，分析者の経験と能力に強く依存する状況にあり,いまだに一定品質の分析成果を導き出す方法論は存在しない．すなわち，分析者が分析の拠り所とする「分析対象を抽出する視点（たとえば，構造的な側面に着目するか，時間的な変化の側面に着目するか，あるいは動作の相互関連の側面に着目するか等の分析の視点）」や「抽出した分析対象を構造化し，構成するときの判断基準」を形式的，かつ具体的に定めた方法論は存在しない．オブジェクト指向分析で記述すべきドキュメント類とその記述規則を定めた仕様記述図式言語 UML(Unified Modeling Language)[1][2]においても,分析で用いるモデル図式の種類やモデルの表記法を定めているに過ぎず,「分析対象を抽出する視点や分析対象の構成に関する判断基準」については何も言及せず，深入りは避けている．

これらの現状を反映して，工科系大学において筆者が担当するソフトウェア工学講座，データベース基礎・実際講座，および情報工学実験（実験の主な内容は，Client Server システムのシステム分析設計とその実装）においても，受講者に，オブジェクト指向システム分析の要点である「着目すべき視点や分析対象の構成に関する判断基準」を修得させるには，多くの分析事例を積み重ねるさせるしかなかった．たとえば，システム分析の出発点であるクラス構造を「いかなる視点で，いかに正しく構成するか」の技能は，数多くの事例を用いた分析経験によって体得させるしかなかった[3].

このような状況を改善し，大学の受講者はもとより，情報産業に従事している比較的システム分析の経験が少ないシステムエンジニアに対して，正しいオブジェクト指向システ

ム分析技能を身に付けさせるには，まずは正しいクラス構造分析とそのモデル記述を可能にするときの要点，すなわち「クラスの抽出に関する視点とクラス構造を構成する判断基準」を定式化する必要があった．

これらの必要性に応える実践的な研究は，現在までは主に情報システム開発のコンサルティング企業（たとえば，UML 統一開発プロセスを提唱する Rational Rose 社[4],データ中心の開発プロセスを提唱する PFU 社[5]等）が行ってきた．しかしながら，これらのコンサルティング企業でも，厳密な意味での定式化された判断基準は存在せず，旧来と同様，「コンサルタントの支援のもとに，システム分析者が数多くの典型的な事例を通して経験的に修得する」ことに力点が置かれている．

本研究は，これらの現状を背景にして，現在のオブジェクト指向分析作業に必要な「クラスの抽出，およびクラス構造を構成する判断基準の定式化」を目指したものである．具体的には，従来から経験的に体得すべきとされている「分析の視点，および判断基準」をオブジェクト指向が持つ数々の特徴にさかのぼって考察し，数学的な形式に定式化することを目的とする．これらの定式化よって，機械的に判断可能な部分を単純作業化（あるいはツール化）し，最終的に，システム分析の焦点を分析本来の作業である「ユーザ要求の抽出とそれに関連するデータ，機能の洗い出し」に移行させることを目標としている．

1.2　オブジェクト指向分析の特徴と問題点

オブジェクト指向に基づくシステム分析（以後，オブジェクト指向分析と呼ぶ）で必要となる「分析対象を抽出する視点や構成に関する判断基準」の定式化を試みるとき，その理論的な基盤は，当然ながらオブジェクト指向概念が持つ基本的な特徴の上に構成されなければならない．そこで以下に，オブジェクト指向が持つ基本的な特徴について簡単に概観する（詳細は付録 A に記す）．

1.2.1　オブジェクト指向概念の特徴

（1）基本的な特徴

オブジェクト指向概念の代表的な特徴として，次の 5 点があげられる．

① カプセル化（Emcapslation）

プログラムで用いるデータが,プログラム中の多くの箇所から無制限にアクセスすることを防止するために考案された概念である．データを無制限にアクセスすることを防止するために，データを操作する手続きを一箇所にまとめ上げ，それらの手続きを通してのみデータにアクセスできるようにした仕組みである．

② クラスーインスタンス（Class-Instance）

ユーザが任意に扱うべきデータ型を定義可能とするため，データ型名や型が持つ演算，あるいは操作をユーザ自身が記述できるようにした仕組みである．クラスは，従来のプロ

グラミング言語が持つ型（たとえば，整数型，実数型，文字列型等）と同様，変数（＝インスタンス）が属する型の概念を拡張したものに相当する．

③ 情報隠蔽 (Inforamtion Hidding)

カプセル化と類似する概念で，クラスが持つメソッドの利用方法（＝メソッド・インタフェース）の定義とメソッドの実装を分離して行えるようにしたものである．メソッドの実装に関する手続き記述は，メソッドを利用する側から隠蔽される．

④ 継承 (Inheritance)

人工知能における知識表現と同様，既存のクラスが持つ変数やメソッドを，そのクラスから派生させた派生クラスで再利用できるようにした仕組みである．本研究では，継承が持つ基本的な制約，すなわち「継承の親-派生の関係にあるクラス間では，派生クラスは，親クラスが持つ変数名と同一名称の変数は定義できない」との制約を，クラス構造の構成時における基本的な制約条件として積極的に活用している．

⑤ 多相性，あるいは多態性 (Pormorphism)

変数に関しては，継承階層の親-派性関係にあるクラス間で同一名称の変数を定義できないが,メソッドに関しては,同一名称メソッドを再定義することができる仕組みである．継承と同様に，本研究では多相性をクラス構造の構成する上での基本的な制約条件として活用している．

（2）構成上の特徴

オブジェクト指向に基づくシステム（たとえば，C++や Java により記述され有機的に結合したプログラム集合体）の分析では，クラスの集合を抽出すると共に，クラスが持つ役割を決定する作業が重要となる．抽出されたクラス集合には，一般に図 1−1 で示すオブジェクト指向の標準的な機能構成である MVC(Model-View-Controller)モデル[6]に沿った役割が割り振られる．

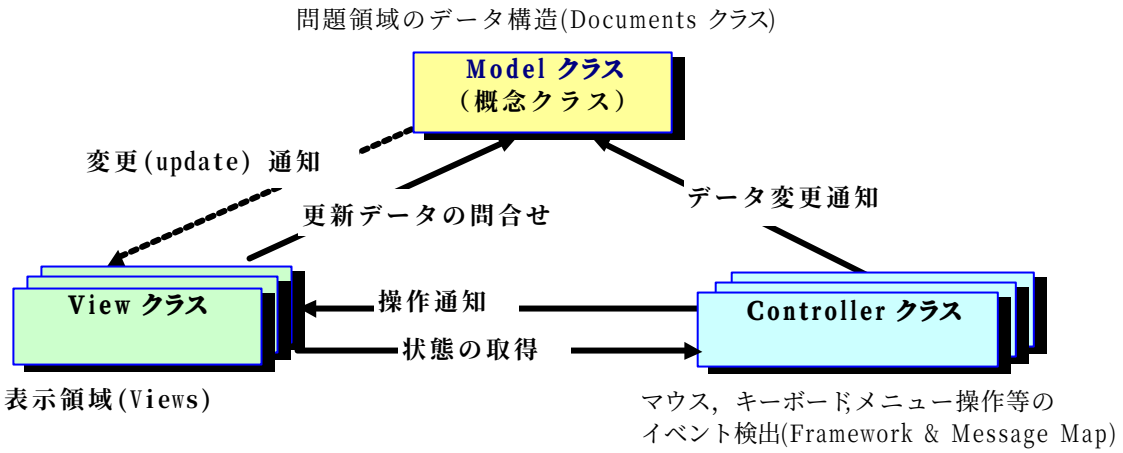


図 1-1 クラス集合の標準的な役割分担を示す MVC モデル

図 1-1 において， Model クラスはデータを格納する役割を持つたクラス集合（C++では Document クラスと呼ばれる）を意味し，情報システムが扱うアプリケーション分野の概念用語に対応することから「概念クラス」とも呼ばれる．View クラスは，入出力・表示を担当するクラス集合，Controller クラスは，マウスの操作や画面内のボタン，メニューの操作に伴って発生するイベントを認識し，識別する役割を持つたクラス集合（ただし，C++では，両クラス群は View クラスに統合化されている）を意味する．

MVC モデルを用いてクラス構造を分析して行く標準的な手順は，まず最も変更の発生する頻度が少ない Model クラスの分析から開始し,次いで入出力を担当する View クラスの分析へ進み，最後に頻繁に変更が生じる恐れのある Controller クラスの分析へと進む．

1.2.2 現状における代表的なクラス分析と問題点

MVC モデルを前提にして，クラス集合を抽出し，その構造を決定する現状の代表的なクラス分析手法，およびそれらが抱える問題点を以下に概観する（分析方法の概要は付録 B に記す）．

(1) 一般的なクラスの抽出法と問題点

現状におけるクラス分析の代表的な手法として次がある．これらの手法は MVC モデルのいずれの役割を持つクラスに対しても適用可能である．

(a) ユースケース分析法

ユースケース分析法は，ユーザと分析者が協力して，開発システムに関する要求機能を具体的な利用シナリオ（use case scenario）として網羅的に列挙し，それらをもとにクラス構造を決定する手法である[7]．ユースケース分析の目的は，外部からシステムに働き掛ける「アクタ」と呼ばれる「システム外部の動作主体」を洗い出すと共に，アクタがシステムと対話するシナリオを自然言語によってすべて記述し，それらを整理することでシステムにもたせるべき機能の仕様を抽出する．クラス構造の決定は，記述された多数のシナリオ中に出現する「名詞」を，システムを構成する基本概念として捉え，クラス候補として抽出する．同様に「動詞」はメソッドの候補，形容詞は属性の候補として抽出する．なお，ユースケース分析法の概要は付録 B に記すとおりである．

《 問題点 》

名詞の集合から,システムを構成する基本概念を抽出し，クラスとして構成する判断は分析者に任されており，現状においては，名詞の集合から発見的にクラスを抽出しなければならない．さらに名詞をクラスとして捉えるか，あるいは属性として捉えるかの判断も分析者の経験に任されており，客観的な判断基準は提供されていない．ただし，UML では，抽出した名詞が値を持つときは名詞を属性とし，値をもたないときはクラスの候補とする簡単な判断基準は存在する [1]．しかし初歩的な判断基準でしかなく，実用上の価値は少ない．たとえば，「売上伝票の記入」シナリオに記載された「担当者」は「売上」に付随する属性であるか，あるいは独立した「営業担当者」との関連として判断すべきか等につ

いては無力である.

(b) 責任駆動型分析法

CRC（Class Responsibility Collaborations）カードと呼ばれるカードを用いて，分析対象クラスが提供する機能を「サービス責任」としてカード上に列挙する．同時にサービス責任に関連する他のクラス名をカードに記述し，分類することで，クラスが持つべきサービス責任を定めて行く方法である[8][9][10]．カードを用いた発想法（たとえば，KJ法－川喜多二郎が考案した発想法－[11]）と同様に，分析者が別々に擬人化されたクラスの立場に立ってサービス責任を考察し，その内容をカードに記述する．それらを分析者全員で検討し，分類することによって，サービス責任の主体であるクラスを抽出する．責任駆動分析法の概要は付録 C に記すとおりである.

《 問題点 》

責任駆動型分析法は，クラスが持つサービス責任（＝クラスが持つメソッドのインタフェース)，すなわちクラスが他のクラスに提供すべき機能を抽出し，クラスとして分類する考え方，および手順を示したに過ぎない．サービス責任の抽出や分類の方法は，経験に全面的に依存しており，定式化された判断基準は存在しない.

以上に述べた代表的な２つの分析手法は，シナリオに出現する名詞，動詞やサービス責任等の具体的なデータをもとにしたものである．しかし具体的なデータから抽象的な概念であるクラスを「発見」し，その上でクラスが持つべき，具体的な属性やメソッドを再検討する意味で，トップダウン的なアプローチに分類できる.

(2) Model クラスに特化したモデリング手法

データの格納を中心とした Model クラスの表記法としては，1976 年に P.P.Chen によって提唱された代表的な概念データモデリング手法である ER モデリング（Entity-Relationship Modeling）[12]がある．ER モデリングは，データベースに格納すべきデータ集合をエンティティ型として記述すると共に，エンティティ型間にある概念的な関連を分析時点で矛盾なく記述し，データベースの無謬性を高めることを目的としたものである． ER モデリングについての概要は付録 D に記すとおりである.

《 問題点 》

ER モデリング手法においても，モデリング表記規則を示したに過ぎず，エンティティ型やリレーション型を抽出し，構成する判断規則は明らかに提供されていない.

ER モデリングは，MVC モデルに基づいたクラス分析の出発点である Model クラスの分析（＝概念クラスモデリング）と基本的に同じ発想と手順を持つ．それゆえ， MVC モデルにしたがったクラス分析を行うとき，出発点となる Model クラスの抽出と構成に関する判断基準と共通する問題を抱える [13]．逆に言うと，ER モデリングにおける判断基準が策定されると，直ちにその判断基準は Model クラス分析の判断基準に適用できることになる．そこで次節では，ER モデリングにおける問題点を詳細に分析した結果について述べる.

1.3　ER モデリングにおける問題点の分析

Model クラス分析と ER モデリングは,分析対象となるデータを捉える視点,分析の手順,およびモデルとして記述する内容の点で基本的に同じ考え方を持つ．特にデータベースを中心にした情報システムの構築では，最初に正しい ER モデリングを行うことが必要不可欠とされ，その必要性は情報システムがオブジェクト指向であるか否かを問わない．

ER モデリングに関する方法論は，すでに幾つか提唱されており，代表的なものとしては DATARUN[5]や T 字型 ER モデリング[14]等がある．これらの ER モデリングに関する方法論は，いずれも実際のシステム開発に数多く適用された実績を持つが，とりわけ DATARUN[5]はデータベースを中心にした企業情報システム開発の分野で，モデル指向の方法論（すなわち，データ,機能，構造，状態遷移等の視点毎にモデルの表記法とドキュメントを使い分けることを指向した方法論）として幅広い実績を持つ．

DATARUN は，帳票や伝票から収集した具体的なデータ項目から，計算や変換等によって導出できない基本データ項目を選択し，それらをエンティティ型が持つ属性の候補として束ね，エンティティ型を抽出するものである．したがって基本的な考え方は「データ中心」である．DATARUN は，具体的な基本データ項目を基にエンティティ型が持つ属性集合を定め，その上で属性集合にふさわしいエンティティ型を命名することから，具体的なデータを基に，抽象的な概念に対応するエンティティ型を抽出するボトムアップアプローチ（所謂，具体的な実体から抽象な概念を構成するアプローチ）に属する．本論は，DATARUN と同様に，ボトムアップアプローチにしたがって，ER モデルを正しく導き出すことを前提にして，判断基準の考察を行う．

以下では， ER モデリングで生じる具体的な問題点を抽出し，かつ正しい ER モデリングに必要な判断基準を明らかにする第一段階として,ER モデルを作成する過程で発生する誤りについて，調査分析した結果[15]を示す．

(ｉ) 誤り分析調査の実施方法

筆者が担当する「データベースの実際」講座では，受講者に対して，具体的なアプリケーション業務の分析から開始して，ER 図を用いた概念データモデルの記述，データベース・スキーマ設計，Microsoft ACCESS による実装までを演習課題として課している．「ER モデリングの誤り分析調査」は，この講座の受講者を対象にして，次の要領で実施したものである．

① 受講者は 3 名以内でチームを構成し，1ヶ月を費やして，チーム毎にデータベースアプリケーション（以後,DB アプリケーションと呼ぶ）開発の中核となる ER モデルを記述する．

ER モデルの記述規則， および DATARUN 方法論に基づいた支援ツール SILVERRUN の利用方法については，3 コマ(270 分)の演習時間を割り当てる．

(ii) 調査対象の母集団

誤り分析の対象としたDBアプリケーションは，1999年度の授業においてチーム毎に作成した表1-1aで示す9つのデータベースアプリケーションを対象とした．記述したERモデル内のエンティティ数，関連数，属性総数の平均は,表1-1bのとおりである．

表 1-1a 調査対象の DB アプリケーション名称

No.	サンプル DB アプリケーション名称
1	野球選手成績管理
2	レンタカー
3	図書館
4	パソコン部品
5	通信プロバイダ検索
6	アパート検索
7	競馬レース成績・血統管理
8	カクテル・レシピ
9	新作ゲーム

表 1-1b 調査対象の DB アプリケーションの各種統計特性

ER 図に含まれる構成要素名	構成要素の平均値
① エンティティ型	**4.2**
② 関連型	**9.1**
③ 属性	**14.9**

(iii) 正解の判定方法

ERモデルの正しさの判定は，筆者がモデルの完成度にしたがって厳密に採点した．採点時における配点比率は，エンティティ型と属性等の正しさに対して 70%，関連の正しさに対して30%とした．採点した内容および配点は，以下のとおりである．

①エンティティ型の種類と属性の正しさ(配点 50 点)

②関連付けすべきエンティティ型の正しさ（配点 20 点）

③関連多重度の設定の正しさ（配点 30 点）

このほかに，主キー属性の正しさも正解の判定には含ませたが，誤りとしては二次的なものであることから，誤り分析の評価には含ませていない．

(iv) ERモデリングにおける誤りの種類と比率

ER モデリングにおいて，誤りのあったサンプルに占める誤りの種類と発生比率は以下のとおりであった.すべての誤りに占めるそれぞれの誤りの発生割合を鍵括弧の中に示す．なお，誤り事例として用いたERモデル図中で下線を付した属性は，主キー属性である．

① 異なる種類の属性の混在 [誤り率＝88%]

属性とはエンティティ型が持つ固有の性質を列挙したものであるとの認識はできているが，しかしエンティティ型とは，管理すべき対象が時間的に永続した存在であるとの認識ができていない誤りである．本来であれば，独立したエンティティ型間の関連として認識すべきものを，エンティティ型が持つ属性とした誤りである．ER モデリングの習熟度に関係なく 88%の学生が誤りを犯した.以下に典型的な誤り事例を示す．

《 事例１ 》エンティティ型固有の属性の混同例

売上に関連する固有のデータとして，日付や売上管理番号，売上担当者が考えられる

が，これらを売上に関するデータを図 1-2a のとおりにまとめ上げ，エンティティ「売上」の属性とする誤りである．売上担当者は，売上とは独立した存在であるエンティティ「営業担当」が持つべき属性であるから，図 1-2a は図 1-2b のとおり，記述されなければならない．

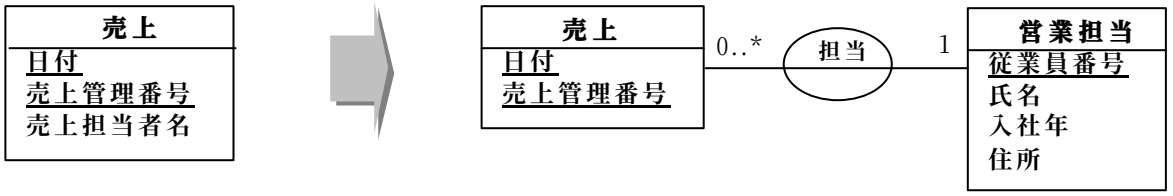


図 1-2a　異なる固有の属性の混入したエンティティ型

図 1-2b　2つのエンティティ型に分離した ER 図

《 事例 2 》エンティティ型と属性の混同例

例 1 と同様に商品に関する固有のデータとして，商品番号や商品名のほかに，値引率もエンティティ「商品」の固有の属性とする誤りである．この場合，商品毎に値引率が異なるか，あるいは商品区分毎に一律の値引き率が定まっているかの分析が必要になる．一般的に値引率は商品の分類単位によって決まり，商品毎に値引き率が異なることは少ない．さらには販売期間毎（たとえば，大安売り日，年末年始特別割引，優勝記念割引等）に異なる値引率が設定されることが多い．したがって値引率は，商品とは独立して存在するエンティティ型になる．それゆえ，図 1-3a のエンティティ型は，図 1-3b で示すとおり，２つのエンティティ型に分離され，関連付けされた ER モデルとして記述しなければならない．

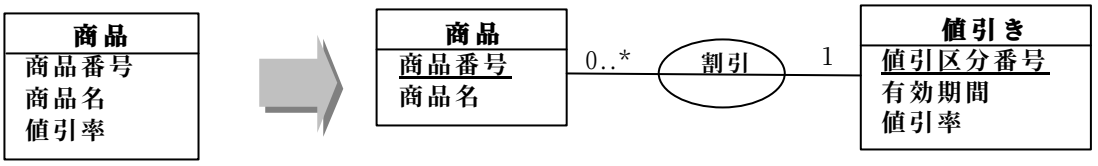


図 1-3a　異なる固有の属性の混入したエンティティ型

図 1-3b　2つのエンティティ型に分離した ER 図

② 属性と実現値の混同 [誤り率＝22%]

エンティティ型中に，属性と属性の実現値を混同して列挙する誤りで，属性と属性の値の区別が理解できていない誤りである． ER モデリングの初心者に多い誤りである．図 1-4a の ER 図はもちろん誤りで，図 1 - 4b のとおりに記述されなければならない．

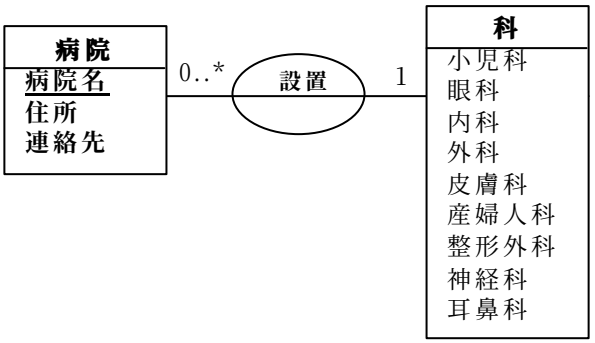


図 1-4a　属性と実現値の混同した ER 図

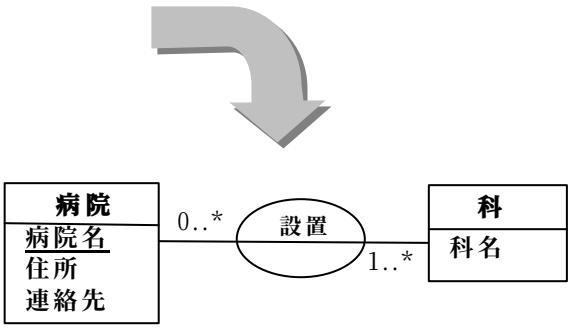


図 1-4b　エンティティ型の属性名として整理した ER 図

③ エンティティ型属性と関連属性の混同 [誤り率＝13%]

関連型が持つ属性とすべきところを，エンティティ型が持つ属性とする誤りである．ER モデリングの考え方をよく理解した学生でも，このような誤りを起こす．たとえば，競馬のレース結果の管理に関する ER モデルとして，図 1-5a で示す ER 図を記述することが多い．しかしながら，エンティティ型「レース結果」に含まれる属性の「馬名」は，独立した存在であるエンティティ型「競走馬」が持つ属性であり，同様に「騎手名」は，独立した存在のエンティティ型「騎手」が持つ属性のはずである．さらに，属性「着順」や「記録」,「配当」は，競走馬と騎手がレースで出走したことにより，関連付け（正確には 3 項関連になる）が行われたとき，はじめて決まる属性であり，関連型「出走」が持つべき属性である．したがって，図 1-5b が正しい ER 図になる．

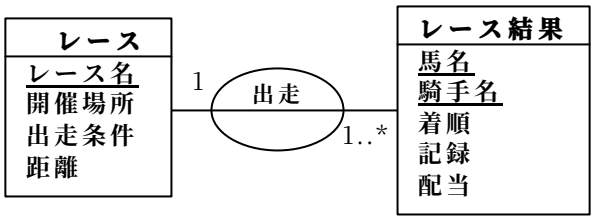


図 1-5a　エンティティ型属性
と関連属性の混同した ER 図

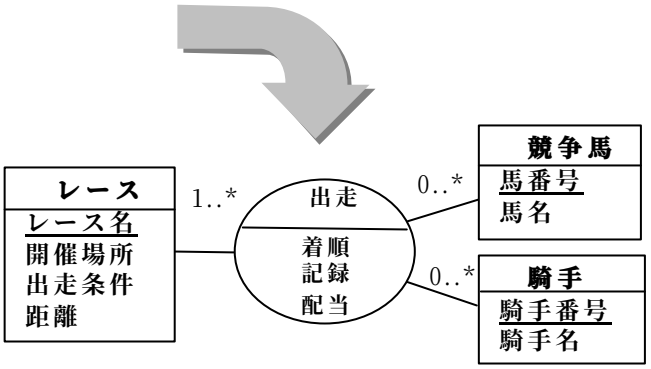


図 1-5b　関連の属性として整理した ER 図

④ 関連付けすべきエンティティ型の誤り [誤り率＝13%]

関連付けるべきエンティティ型を取り違える誤りで ER モデリングに習熟した者でも起こす誤りである．ER モデルとしてモデル化するアプリケーションドメインのビジネスルールに依存することから，必ずしも誤りでないときがある．たとえば，客室の予約を管理する場合，図 1-6a で示す客室予約の ER 図よりも，図 1-6b で示すエンティティ型の関連

として管理した方が正しい．なぜならば予約に対して，個々の客室を確保することより，客室のタイプを指定して確保する方が,客室の変更に対する柔軟性を確保できることから，一般的な客室予約としては，図 1-6b で示す ER 図の方が妥当となる．

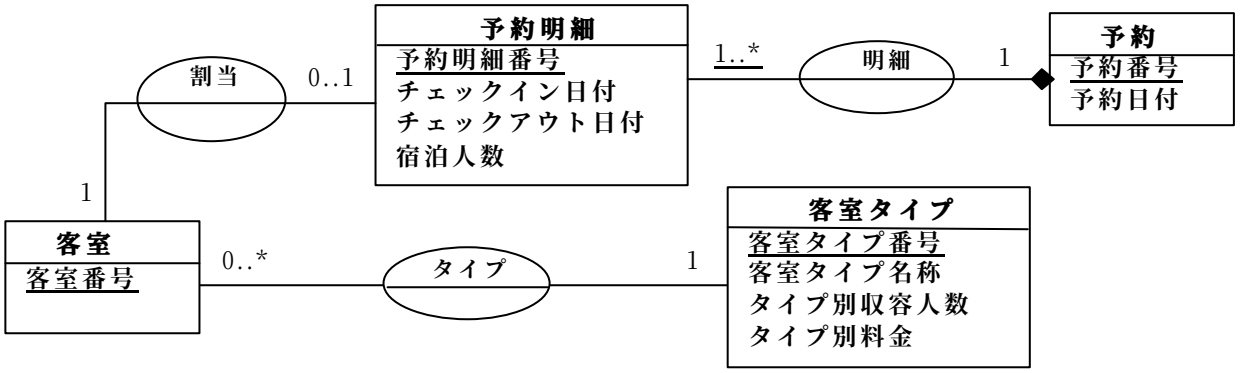


図 1-6a　関連付けすべきエンティティ型が妥当でない ER 図

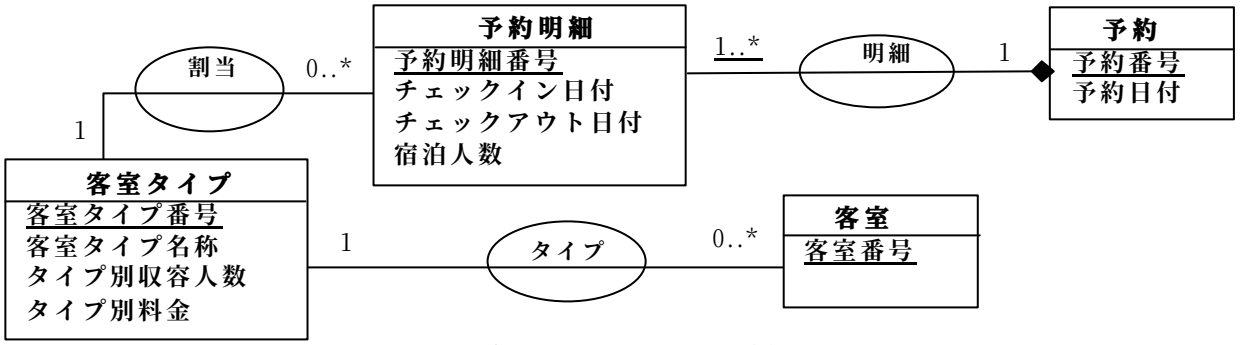


図 1-6b　関連付けエンティティ型が妥当である ER 図

⑤ 関連の多重度決定の誤り [誤り率＝67%]

「異なる種類の属性の混在」に次いで比率の高い誤りである．この誤りは，どのようなビジネスルールにしたがって情報を管理するかに依存しており，一概には誤りとは言えない．たとえば，典型的な例として，図 1-7a の「テレビショッピングの商品」と「クレジットの支払い」に関する ER 図における関連の多重度がある．テレビショッピングで購入した商品の支払いが，必ずクレジットで決済するのであれば図 1-7a の ER 図で正しいが，しかし一般的にはクレジットによる決済は，決済手段の一部に過ぎない．したがって，図 1-7b で示す関連の多重度は「0..1」が正しい．事例として用いた図 1-7a は必ずしも誤りでないが，要求仕様に明記していない限りは，クレジット決済のみ注文は少ないため，誤りとしてカウントしている．

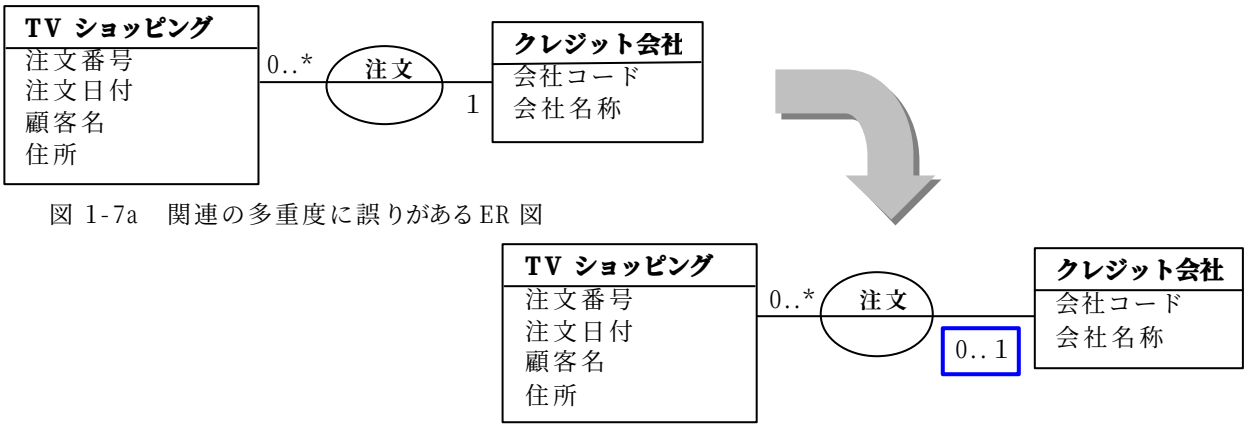


図 1-7a　関連の多重度に誤りがある ER 図

図 1-7b 妥当な関連の多重度に訂正した ER 図

前述の誤り分析の結果において，分析の初心者にとって持つとも厄介な問題は，①の「異なる種類の属性の混在」による誤りと，③の「エンティティ型属性と関連属性の混同」による誤りである．特に「異なる種類の属性の混在」による誤り比率は 88%と高く，この誤りの発生を防止する分析上の判断基準が必要になる．しかし残念ながら，これらの判断基準として明確に定式化した手法は存在しない．

1.4　本研究の位置付け

前節で述べたとおり，オブジェクト指向ソフトウェア開発において，クラス分析はソフトウェアの枠組みを決定するものであり，その完成度がソフトウェアの品質，すなわち要求の充足性，効率，拡張性，保守性，移植性等を左右する．しかし分析結果を図式表記する方法や検証方法に関する研究は，UML を始めとして行われているものの，最も重要な分析における視点や判断基準の定式化に関しての研究は目下のところ見当たらない．現実には，分析経験を積んだ有能なシステムアナリストに依頼して良しとするか，OJT（On the Job Traning）によって，事例から判断基準を「体得」するしかない．このような現状は決して望ましいものでないことは明らかであり，ソフトウェア技術の中心課題が「実装」を中心にした下流工程から，「分析」を中心にした上流工程へ移行している中，その定式化を急ぐと共に，技術移転可能な知識の対象として行かなければならない．

本研究は，これらのモデリング上の課題やオブジェクト指向分析を取り巻く環境を踏まえて，オブジェクト指向システム開発のクラス分析過程で用いられる視点や手順を，次の基本的なアイデアに沿って定式化するものである．

《 基本的アイデア１ 》

分析によって，不確定であいまいなシステム化要件からシステムの構成要素（以後，分析要素と呼ぶ）を洗い出し，クラス構造としてモデル化して行く過程は，あたかも量子論

の世界において，種々の電子の状態が重ね合さった不確定な電子の空間分布状態から，固有のエネルギー状態や存在位置を確定し，分離して行く過程に類比できる．

《 基本的アイデア 2 》

同様に，量子論の世界で，電子が「パウリの排他律」にしたがって異なるエネルギーレベルの状態に配置ことが原子の化学的な特性を決定するように，オブジェクト指向が持つ属性やメソッドに対する制約（たとえば，継承関係にあるクラスでは，同一名称の変数は一つしか存在できない．しかしメソッドは，継承レベルが異なれば重複して存在できる等の規則）を利用することで，クラス構造を構成する規則を定式化できよう．

《 基本的アイデア 3 》

分析によって明らかにするシステムの構成要素は，システム内でその存在が意味を持つ時間，すなわち「存在時間」を必ず持つ．一方，ある特定のクラスから生成されたインスタンスが保有する変数やメソッドは，同じ存在時間をもたなければいけない．なぜなら，インスタンスを生成したとき，すべての変数やメソッドは意味を持ちはじめ，インスタンスが消去したとき，すべての変数やメソッドはその存在の意味を失うからである．この点に着目すると，存在時間を一つの指標として分析要素を束ねてクラスを抽出し，構成する判断基準の定式化が見込める．

本論では，以上の基本的なアイデアをもとにして，以下の構成に沿って議論を展開する．まず 2 章の前半では，本研究の出発点となった ER（Entity Relationship）分析の判断基準の定式化について，データベース設計の概念モデリング作業を具体例として取り上げ述べる．後半では，定式化した判断基準の有効性を検証するために，実際にデータベース分析設計授業で 2 ヵ年をかけ比較実験し測定した「判断基準の効果」について述べる． 3 章では，判断基準をさらにクラス分析にも適応可能とするため，前述の電子の重ね合わせ状態との類比を参考にして拡張し，構成演算として定式化を試みる．さらに定式化した判断基準を用いて，分析過程で出現する典型的なクラス構造としてのデザインパターンが構成演算の適用によって導出できうることを机上実験によって示す． 4 章では，クラス階層の形成過程を時間的な視点に着目してモデル化し，実際のクラス構造に合致するかを Java クラスライブラリが持つ統計的な特性分析と比較して検証する．5 章では本研究に関連する類似研究との比較を行い， 6 章では研究の結論と今後の展望について述べる．

2章　Model クラス分析における判断基準の定式化

2.1 分析における基本的な概念操作

オブジェクト指向にもとづく分析の目的は，開発すべき情報システムに対する要求を，オブジェクト指向が持つ基本概念とその構成に写像することにある．すなわち，情報システムに対する要求を満たす「クラス構造」を決定すると共に，要求に含まれる操作的な側面を，クラスが生成するオブジェクト（＝インスタンス）の相互作用，およびオブジェクト内部の振る舞いに写像することにある．クラス構造は，要求を満たすために個々のクラスが持つべきデータ（＝属性）やデータ操作（＝メソッド）を記述すると共に，他のクラスを内部データ構造として内包する関連や，メッセージを介して他のクラスのインスタンスに作用し影響を与える関係を記述している．

ここで留意すべきは，分析が静的な関連に対する視点から，時間的な要素を加味した振る舞い的な視点へと移行して行く点である．現在の分析方法では，時間的な様相に関する分析は行われずに，静的な視点でオブジェクトとその関連を分析し，クラスが持つべき属性名やメソッド名，あるいはクラス間の関連を分析しているに過ぎない．言い換えると，システムで実装するオブジェクトは，その存在と共に存続すべき時間を必ず持つにもかかわらず，分析の対象としていない．実際，分析者はオブジェクトが存在するとき，その時間的な側面（特にオブジェクトの存在が意味を持つ時間間隔）に関する認識を暗黙的に行われている．しかし，それらを分析ドキュメントとして明示的に記述することはない．

そこで，オブジェクトを認識した後にクラスとその構造決定するボトムアップアプローチを前提にして，分析者が頭脳の中で行っているクラスの抽出と関連付け等の概念操作を時間的な様相に焦点を当てて整理してみると以下のとおりになる（類似研究として[16]）．

(1) オブジェクトの認識

分析の第 1 ステップで行われる情報システムに要求されるオブジェクトの認識の中には，暗黙裡にそのオブジェクトが必要になる時点，および不要になる時点の認識も含んでいる．すなわち，分析者がオブジェクトを認識するとき，そのオブジェクトが存在を開始するタイミング（オブジェクト指向的な観点から言い換えると，オブジェクトを生成する「きっかけ」となるイベントの発生時点）とオブジェクトが存在すべき時間間隔（以後，存在時間と呼ぶ）も同時に認識される．

(2) オブジェクト間の関連の認識

次いで，オブジェクトが認識されるとオブジェクト間にどのような関連があるかを認識

する．しかし，逆に関連が認識され，その後，関連にふさわしい「役割」を持つたオブジェクトの存在を探る場合もあるため，(1)と(2)の順序関係は確定したものではない．

関連の認識を存在時間の視点から捉えると，ある任意のオブジェクトの存在時間に重複があるときのみ，その間に関連が生じることから，関連の存在時間は，関連付けされるオブジェクトの存在時間の共通部分に限定される．この点を加味すると，関連は複数のオブジェクトの共通存在時間でのみ存在する「オブジェクトを結び付ける一時的なオブジェクト」として定義することができる．したがって，関連もまたオブジェクトとして，属性を持ちうる．

関連を中心にしたオブジェクトに関する概念的な操作には次がある．

① 結合(Combine)

関連するオブジェクトを結合し,一つのオブジェクトにする操作である.この操作では,結合されるオブジェクトの存在時間が一致したオブジェクトのみが対象になる．

② 分離(Separate)

１つのオブジェクトが内部構造として持つ属性や操作を複数のオブジェクトに分離し，配分する操作である．当然，分離された複数のオブジェクトは，同じ存在時間を持ち，オブジェクト間には，分離前に同じオブジェクトであったことを示す関連が形成される．

(3) 動作の認識

多数のオブジェクトが機能するとき，オブジェクトは相互に影響を与えあう．その影響を受けて，オブジェクト内で起動すべき動作を認識するもので，次のような動作の認識に分類することができる．

① 初期値の設定(Decide)

オブジェクトが生成されるとき，初期値としてどのような実現値を持つかの認識である．ここで，オブジェクトが生成されるとき，すべての初期値は「同時に」設定されることに留意する必要がある．

② 状態(値)の保持(Preserve)

オブジェクトが持つ値を保持すると共に，それらを読み出す操作の認識である．オブジェクトの存在時間内でしかこの操作が有効ではならないことから，逆にこの操作に先立って，オブジェクトの存在と存在時間の認識がなされていなければならないとの順序的な制約がある．

③ 更新(Modify)

オブジェクトが持つ値を更新する操作の認識である.②と同様,この操作に先立って,オブジェクトの存在と存在時間の認識がなされていなければならないとの順序的な制約がある．

④ 廃棄(Destroy)

オブジェクトを廃棄する操作の認識である．この認識が行われた以後は，①～③の操

作は意味を失うものとして扱われる．

オブジェクトの動作の認識は，時間的な前後関係を配慮しなければならないことから，「オブジェクトの認識」や「オブジェクト間の関連の認識」に比較して，複雑にならざるをえない．

2.2 概念クラス抽出の判断基準

本節では分析者が行う前述の概念操作を加味して，概念クラス抽出の判断基準について考察し，定式化を行う．

2.2.1 存在時間とイベント

オブジェクト指向の大きな特徴の一つとしてイベント駆動がある．特に何らかのイベントの発生に伴って，具体的なオブジェクト（以後,単にインスタンスと呼ぶ）の生成が行われると，そのすべての属性の初期値が設定され，起動可能なメソッドが決まる．これを逆に捉えると,「インスタンスを生成するきっかけとなるイベント（以後，生成イベントと呼ぶ)」が同一の属性集合やメソッド集合は，同じオブジェクトが持つ属性やメソッドであり，したがって同一のクラスに属することになる．より具体的に言い換えると,『同一の生成イベントを持ち，同一のタイミングで初期値が設定される属性名の集合や，同一のタイミングで具体的なメソッド・インタフェース名（他者がメソッドを利用するときのインタフェース定めたパラメータ部を含むメソッド名．以後単にメソッド名と呼ぶ）が決定されるメソッド名の集合は，同じクラスが持つ属性集合，あるいはメソッド集合として分類できることになる』(ER モデリングに限定して，同様な発想でデータを生成するイベント PDG：Primaly Data Generator に着目した類似研究として[17]がある)．この関係を利用すると，あらかじめ洗い出したクラスが持つべき属性の候補（以後，属性候補と呼ぶ）の集合やメソッドの候補（以後，メソッド候補と呼ぶ）の集合の中で「同一の生成イベントを持つか否か」を判断基準として，それらを分類し集約することで，クラスやクラスが持つ属性・メソッドの集合を抽出することが可能になる．

上述の判断基準を生かすには，生成イベントの種類と生成イベントが発生する状況を分析段階から意識的に行い，ドキュメントに明示的に記述する必要がある．しかしながら，現在のオブジェクト指向方法論では，イベントの洗い出しとその時間的な順序関係の認識は，分析の後工程であるオブジェクトの動作分析工程でしか行われない．現状における代表的な分析手法であるユースケース分析や CRC カードによる責任駆動型分析等においても，オブジェクトが生成される生成イベントを明示的に記述し，分類・整理することは行われていない．その原因として，分析の出発点では，データや機能を静的な視点でしか認識せず，当然行われるべき時間的な側面に関する分析の有用性が認識されていないためと考えられる．

そこで筆者は，オブジェクトの生成イベントの同一性を，概念クラス（＝ER モデリングにおけるエンティティ型，あるいは MVC モデルにおける Model クラス）モデリングにおける重要な判断基準として位置付け，定式化を行った．

2.2.2　判断基準の定式化に関する概念操作

前項で述べた認識にしたがって，概念クラスの抽出と構成の判断基準を定式化するために，以下の点に着目する[16]．

(1) クラスからのインスタンス生成

分析に先立って，情報システム化すべき業務で用いられる帳票や伝票類から，データ項目を収集し，他のデータから計算によって値が導出できない基本なデータ（以後，基本データと呼ぶ）の集合を明らかにする．概念クラスの持つ属性候補には，基本データのみがなりうる．帳票類から基本データ集合が明らかになると，次いでそれらを生成するきっかけとなる生成イベントを洗い出す．

特定の生成イベントによってインスタンスが生成されると，インスタンスが持つすべての属性の初期値が「同時」に実現値として決定されることを利用すると,「基本データ集合は明らかにされてはいるものの，いかなる概念クラスの存在し，それらの概念クラスはいかなる属性を持つかが明らかになっていない状況」では，前項でも述べたとおり,「生成イベントの同一性」が，概念クラスを抽出する判断基準となりうる．すなわち「基本データ集合の中で，それらの初期値を決定する生成イベント名が同じであれば，同じ概念クラスが持つ属性候補」になることを利用して，基本データ集合の中で，初期値を決定する生成イベント名が同じ基本データは，同じ概念クラスの属性候補になる．具体的には,「初期値が決定される生成イベント名（以後，単に決定時点 t と呼ぶ）が同じ基本データは，同じ概念クラスに属すべき属性候補」となる．ただし，決定時点 t は実時間を表すものでなく，情報システムが動作する「きっかけ」を，その発生時点の順序関係を加味して，離散的で抽象的な時間概念として表現したものである．概念クラスの存在が明らかになっていない分析段階では，決定時点 t は厳密なものでなくてよく，システム化対象の業務の中における一時点（たとえば，システム外部によって発生させられた「注文」イベントの発生時点）といった粗い認識で十分である．

(2) インスタンスの識別性

インスタンスは，単一の実現値を持つ属性，すなわち「単値属性」の組で一意的に識別できなくてはならない（※もっともオブジェクト指向では，すべての属性値が同じであるインスタンスの存在を許していることから，オブジェクトを一意的に識別するためにオブジェクト ID を便宜的に導入している．しかし，概念クラスの存在や概念クラスが持つ属性が明らかでない分析段階では，オブジェクト ID をもって，単値属性の組によるインスタンスの一意識別性に代替することはできない）．したがって，概念クラスの属性候補となる基本データは，単値の基本データのみでなければならない．複数の実現値（＝多値）を同時

に持つ基本データ，あるいは選択的に実現値が決まる（＝選択値）基本データは，単値の基本データと混在することはできない．なぜならば，多値や選択値の基本データは，唯一決定される実現値をもたないことから，これらを含む基本データの組でインスタンスを一意的に識別することができなくなるからである．

以上から「基本データ ｉ の決定時点ｔにおける実現値集合ξ_iの要素数を　$\#\xi_i$としたとき，単値の基本データと混在する多値 ($\#\xi_i>1$),あるいは選択の基本データ($\#\xi_i=0|1$)は，概念クラスの属性の候補になりえず，多値や選択値の基本データは他の概念クラスが持つ属性の候補として分離する」との判断基準が導かれることになる．分離された概念クラスは，同じ要素数$\#\xi_i$を持つ基本データのみを持つ．なお，ここで用いた一意識別性は，関係（Relation）の集まりを，一意識別可能な要素から構成される関係の集合に分解する関係データモデルの第１正規化，および関数従属性の概念に対応するものである．しかしながら，基本データが所属すべき概念クラスが明らかになっていない分析段階では，正規化の概念は用いることができない．

(3) クラスの属性継承

継承関係もつ２つの概念クラスが存在したとき，親クラスから派生した派生クラスが生成するインスタンスは，派生クラスで定義した属性のみならず，親クラスで定義された属性をすべて継承し併せ持つ．これは親クラスの属性集合と派生クラスの属性集合が，すべて同一の決定時点ｔをもつことを意味する．しかし，継承関係の定まっていない分析段階では，これは逆に決定時点ｔに基づく判断基準のみでは，親クラスが持つ属性集合と派生クラスが持つ属性集合を識別できないことになる．

一方，親クラスは,一般には複数の派生クラスを派生させることから，親クラスが持つ属性集合は，派生クラスの数に対応して多くの決定時点を持つことになる．仮に親クラスが持つ属性集合の決定時点ｔは一つのみと限定すると矛盾が生じることになる．そこで，概念クラスの継承関係を矛盾なく形成する判断基準を導くために，基本データ毎に固有の決定時点集合を対応させて考えることにする．また，基本データ ｉ の決定時点の集合をτ_iとしたとき，その要素数を $\#\tau_i$で表すものとする．すると決定時点が唯一のときには$\#\tau_i$は１になり，継承関係が存在するときには，親クラスに属すべき属性候補としての基本データ ａの$\#\tau_a$は，派生クラスに属するべき基本データ ｂの$\#\tau_b$に比較して多くなる．この関係を利用するため，ここで，便宜上，決定時点集合τ_iから特定の決定時点を選択するパラメータ「状況 S_i」を導入する．「状況 S_i」が特定の決定時点を選択したとき,「状況 S_i」の値を「状況値 s_i」とし，これらの状況値 s_i を集めた集合の要素数を「状況数 $\#S_i$」とする．クラス間に継承がないときは，基本データ ｉの決定時点は，唯一の状況値 s_i を持つ．

これらを用いて「インスタンスが持つ属性の初期値は同一の決定時点ですべて決定される」との判断基準を，継承を加味した判断基準，すなわち「状況数$\#S_i$が異なる属性集合は同じ概念クラスに混在させることはできない」に拡張する.状況数$\#S_i$が異なる属性間では，

属性の決定時点集合τ_iの要素に必ず異なる決定時点を含んでいる.よって他の概念クラス候補に分離しなければならないことになる.

以上をもとめると，属性の継承を加味した判断基準として「状況数 $\#S_i$ が異なる基本データは，$\#S_i$ の大小関係にしたがって，継承関係の親クラス・派生ラスに分離されるべき概念クラスの属性候補である」を導くことができる．この判断基準を用いるとき,「すべての派生クラスから見て共通する親クラスの属性候補名は一致していなければならない」との制約が付くことに注意する必要がある．なお，基本データの初期値を決定する状況がいくつかに分けられるとき，状況値 s_i は，概念クラスの存在が明らかになっていない分析段階では，状況の名称（たとえば,「在庫不足のとき」,「注文受注のとき」）といった粗い認識で十分である.

2.2.3 導入した判断基準

前項の議論を踏まえて，基本データをもとに概念候補を抽出する判断基準をまとめると次のようになる(詳しくは[15]). なお，基本データの集合はΔで表し，基本データ i ($i\in\Delta$)の実現値は,独立した２つの変数，決定時点 t_i と状況 S_i から関数 $f(t_i,s_i)$によって決定されるものとする.

【 判断基準１ 】初期値の決定時点の同時性

初期値が決定される時点 t_0 が同一の基本データ i は，同じ概念クラスに属する属性候補とする．この判断基準は，式(1-1)を満たす集合への分類として表される．ここで f^{-1} は関数 f の逆関数を意味し，基本データ i と状況 S が与えられたとき，その決定時点を定める関数である．∧記号は論理積を意味する.

$$C_1 \equiv \{\, i \mid (\exists t_0)\ (\exists S)\ i\in\Delta \ \wedge\ t = f^{-1}(i,S) \wedge\ t=t_0 \,\} \qquad (1\text{-}1)$$

【 判断基準２ 】実現値の構造性

多値や選択的に実現値が決まる基本データは，単値基本データと混在する形で概念クラスの属性候補となりえない．別の概念クラスの属性候補として分離しなければならない.この判断基準は，式(1-2)を満たす集合への分類として表される．ここで，$\#\xi(i)$は，基本データ i の実現値集合の要素数を意味する.

$$C_2 \equiv \{\, i \mid (\exists n)\ i\in C_1 \ \wedge\ \#\xi(i)= n \,\} \qquad (1\text{-}2)$$

【 判断基準３ 】初期値の決定状況の単一性

初期値の決定時点が複数の状況に依存する基本データは，状況によって一意的に決まる初期値の決定時点を持つた概念クラスの親概念クラスに属する属性候補である．ただし，親概念クラスに置くとき属性候補名は,一致しなければならない.この判断基準は,式(1-3)を満たす集合への分類として表される.ここで，#S(i)は，基本データ i の状況数を意味する.

$$C_3 \equiv \{\, i \mid (\exists m)\ i\in C_2 \ \wedge\ \#S(i)= m \,\} \qquad (1\text{-}3)$$

判断基準 2 は，構造化分析設計において一部で用いられているが，判断基準 1 と判断基準 3 は，筆者が新たに追加したものである．これらの判断基準は，概念クラスを属性候補が持つ実現値の決定時点，値構造，値の決定状況から分類するものとなっている．

2.2.4　机上実験による判断基準の妥当性検証

本項では，前項で述べた判断基準が実際に有効であるかを，いくつかの事例をもとに机上実験し，妥当性を検証する．ただし，基本データ集合とそれらの実現値の決定時点，および決定される状況は，あらかじめ分析によって明らかにされているものとする．

(1) 基本データの実現値の決定時点が異なるとき

注文伝票をもとに次のような基本データ集合が洗い出されたと仮定して，先に定めた判断基準にしたがって基本データ集合を分類してみよう．

基本データ集合 ψ_1 :　｛ 顧客番号，日付，単品注文番号，商品番号，顧客名，住所，電話番号，商品名，商品単価 ｝

基本データ集合 ψ_1 は，「注文を受付けた」ときにすべて伝票に記入される．しかしながら，詳細に見ると，それぞれの基本データが持つ実現値の決定時点は異なる．たとえば，注文番号や日付は，伝票に記入されたときが実現値の決定時点であるが，顧客番号や顧客名，顧客の電話番号，住所の基本データは，取引を開始するにあたって顧客情報を登録するとき，はじめて実現値が決定されている．注文受付時に注文伝票に記入される値は，それらの値の参照にすぎない．　同様に,商品番号や商品名，商品単価の基本データは，取扱商品として商品登録するときに実現値は決定され，注文受付時にはそれらを参照しているに過ぎない．したがって，判断基準 1 によって基本データ集合 ψ_1 を分類すると，図 2-1 のとおりになる．

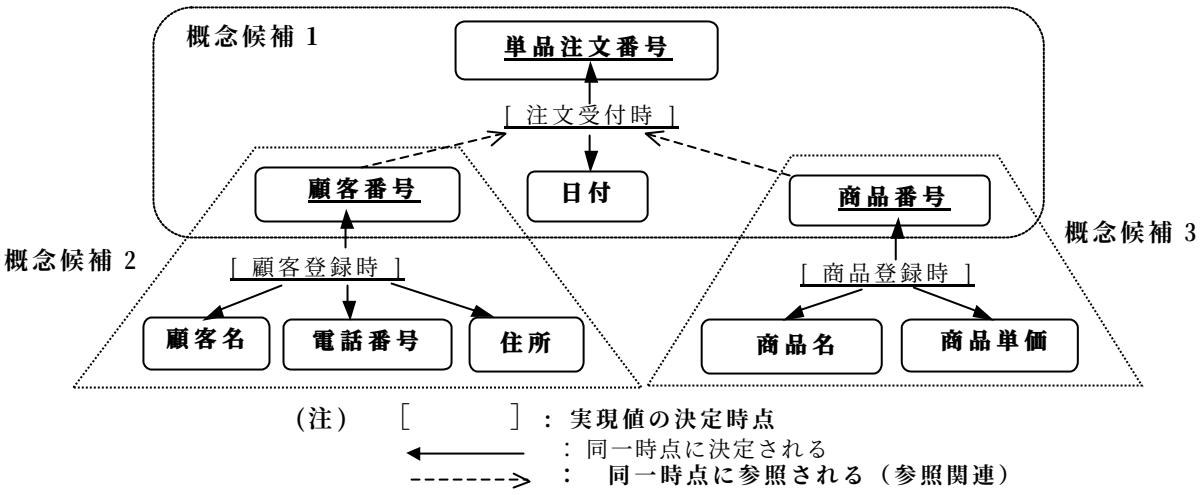


図 2-1　判断基準 1 を用いた基本データ集合 ψ_1 の分類

図 2-1 は基本データの実現値の決定時点によって，3 つのグループ，すなわち 3 つの概念クラスの候補に分類できることを示している． 実現値の参照は，一般に一対多の関係にあることから，図 2-1 は図 2-2 で示すような概念クラスモデルに対応することになる．

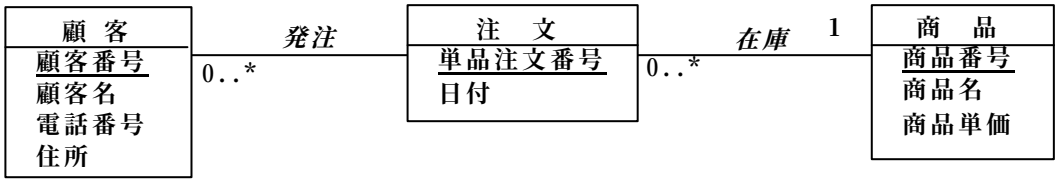


図 2-2　基本データ集合ψ_1の分類に対応する概念クラスモデル

ここで，図 2-2 中の関連の正確な多重度はビジネスルールの分析から，また概念クラス名は，属性候補集合を代表するにふさわしい名称を別途検討して決定するものとする．

(2) 決定時点は同一で，実現値構造が異なるとき

同様に次のような基本データ集合を判断基準にしたがって分類してみよう．

基本データ集合ψ_2：{ 日付，注文番号，商品名，商品単価，
明細番号*，明細注文数*，商品番号* }

ここで，＊印の付いた基本データは，実現値の決定が同一時点で複数回行われるような多値構造を持つことを示している．基本データ集合ψ_1と同様にして，判断基準 1 を用いて基本データ集合ψ_2を分類すると図 2-3 で示すとおりに分類される．

概念クラス候補 1 の属性候補集合を見てみると，実現値の決定時点が同一で多値構造を持つ明細番号，明細注文数を含んでいる．そこで，判断基準 2 にしたがって，それらを別の概念クラス候補に格納すべき属性候補として強制的に分離する．

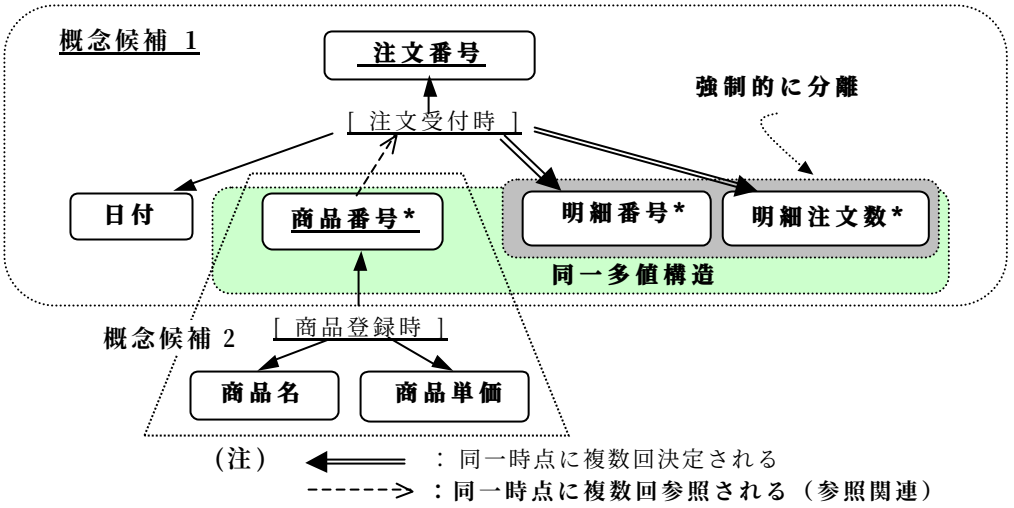


図 2-3　判断基準 1 を用いた基本データ集合ψ_2の分類

強制分離後の分類を概念クラスモデルに対応つけて示すと，図 2-4 のとおりになる．実現値の決定時点が同一ではあるが，実現値構造が異なるために強制的に分離され生成された概念クラス候補同士の関連は，識別子依存関係 (⇐存在依存関係)に対応する．

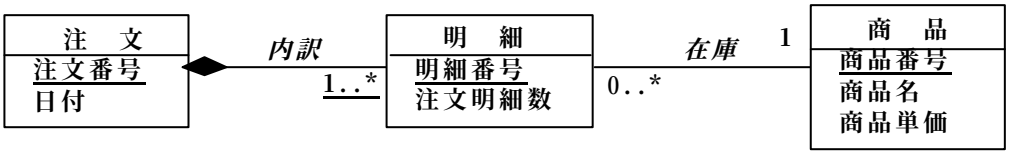


図 2-4　基本データ集合ψ_2の分類に対応する概念クラスモデル

(3) 決定時点が異なり，かつ実現値構造が異なるとき

販売店において，特定の日に販売した商品と顧客の実績データを集計したときのように，それぞれが多値構造を持つ基本データ集合ψ_3を判断基準にしたがい分類してみよう．

基本データ集合ψ_3：　{ 日付，顧客番号*，　顧客名*，商品番号*，価格* }

基本データ集合ψ_3をψ_1，ψ_2と同様にして分類してみる．実現値の決定時点は「販売商品実績の集計時」，「商品の値決め時」，「顧客登録時」の 3 通りあるので，判断基準 1 にしたがえば，図 2-5 のとおり，　3 つの概念クラス候補に分類されるかのように見える．しかし，実現値の決定時点「販売商品実績の集計時」は，構造の異なる 2 つの多値属性を参照する決定時点に過ぎず，その決定時点を持つ独自の属性は存在しない．したがって，図 2-6 で示す概念クラスモデルとなる．

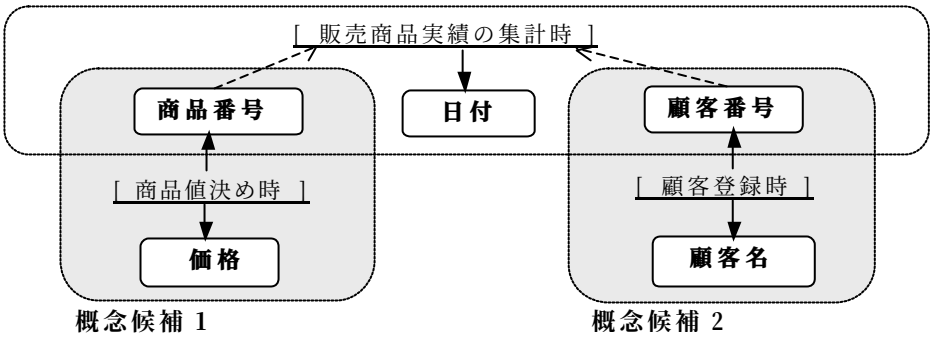


図 2-5　判断基準 1,2 を用いた基本データ集合ψ_3の分類

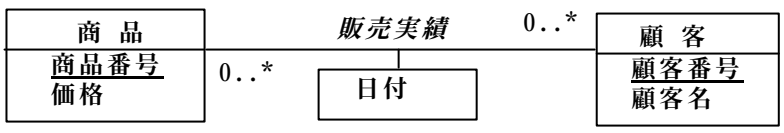


図 2-6　基本データ集合ψ_3の分類に対応する概念クラスモデル

(4) 実現値の決定時点が状況に依存するとき

以上に示してきた事例では，実現値の決定時点が状況に依存しないことを前提にしてきた．しかし状況に依存して，用いられる基本データ集合が異なる場合が考えられる．たとえば，次のような事例が考えられる．

①《 状況=顧客受注による発注 》**基本データ集合ψ_4:**

{ 発注番号，日付，発注業者番号，発注数量，納入予定日，納入顧客番号，
運送業者番号 }

②《 状況=在庫不足による発注 》**基本データ集合ψ_5:**

{ 発注番号，日付，発注業者番号，発注数量，入庫予定日，倉庫番号 }

この基本データ集合ψ_4，ψ_5は，商品を発注するときに，「顧客受注による発注」と「在庫不足による発注」の２つの状況があることを示している．状況に対応して，それぞれ実現値の決定時点を持つ．

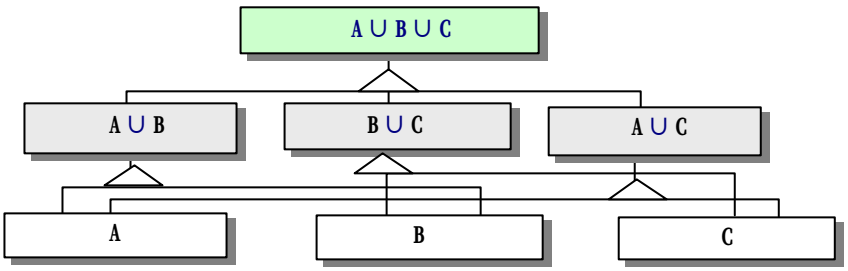


図 2-7　状況数が 3 のとき，基本データ部分集合間の継承構造

このような場合には，判断基準１の適用に先立って判断基準３を適用して，複数の状況を持つ属性候補を継承関係から見て上位の概念クラス候補群に属すべき属性候補として分離する．たとえば，複数の状況を持つ基本データ部分集合 { 発注番号，日付，発注業者番号，発注数量 } と，それ以外の基本データ部分集合 { 納入予定日，納入顧客番号，運送業者番号 }, { 入庫予定日，倉庫番号 } に分離してから，それぞれの部分集合に判断基準１を適用する．

このとき，複数の状況を持つ部分集合とそれ以外の部分集合間には，継承関係で結ばれる．事例では状況は２つであったが，状況がｎ個に分けられるときは，状況数ｎに対して（2^n-1）個の部分集合に分離される．たとえば，3 つの状況に対応する基本データの集合 A,B,C が存在したとき，これらは図 2-7 で示す部分集合に分離され，継承構造が構成される．基本データ集合ψ_4，ψ_5に判断基準 3，判断基準 1 を適用して分類された構造は図 2-8 で示すとおりになる．これを概念クラスモデルに対応つけると図 2-9 で示すとおりになる．

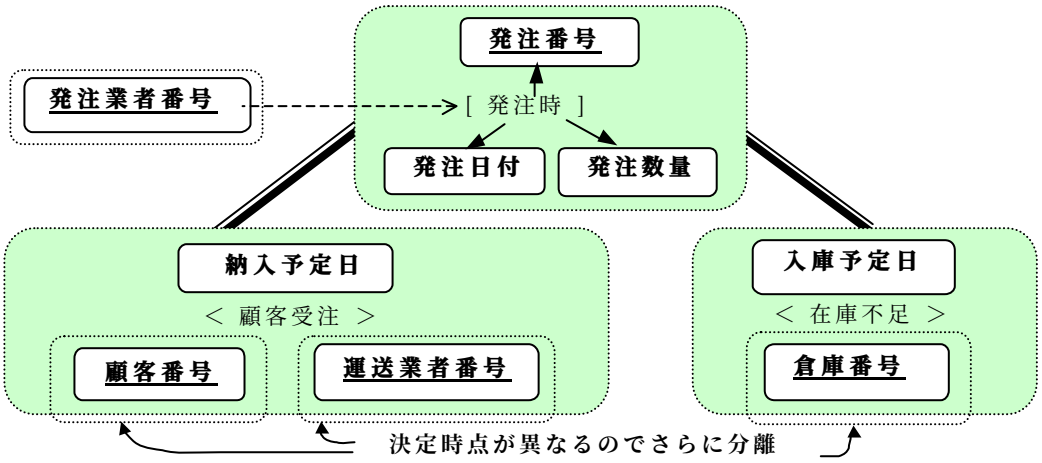


図 2-8　判断基準 3,1 を用いた基本データ集合ψ_4, ψ_5の分類

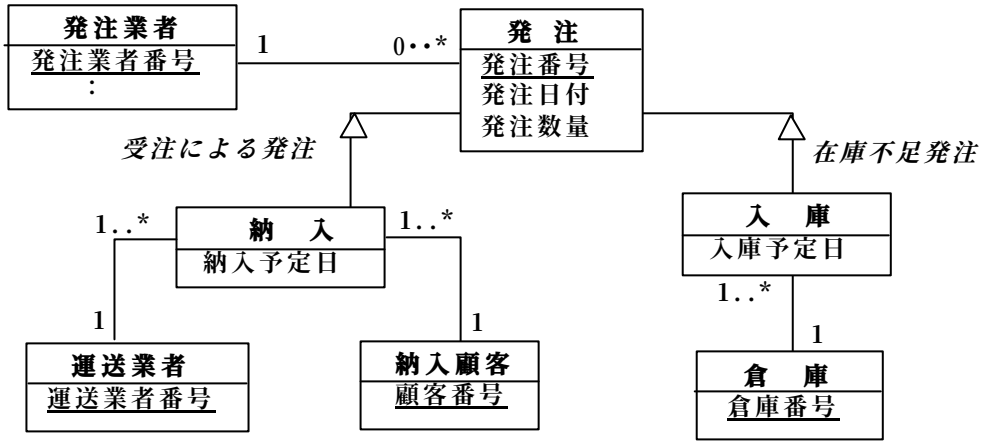


図 2-9 基本データ集合ψ_4, ψ_5の分類に対応する概念クラスモデル

2.3 判断基準の評価実験

2.3.1　データベース授業における ER 分析演習への適用

導入した判断基準が，実際の分析作業にどの程度の効果をあるかを評価するため，ER モデリング分析演習授業に適用し，比較評価実験を行った．評価実験は「概念抽出の判断基準を示したグループ」と「示さないグループ」に分け，モデリングの誤り減少率の観点から評価した[15].

(1) 実験の実施対象者

授業の実施中に「概念抽出の判断基準を示したグループ」と「示さないグループ」とに分割し，比較することは困難である．そこで，年度を分けて行った 2 回の演習授業をもとに比較評価実験を行った．具体的には，判断基準を用いない年度の受講生が作成した ER モデルの正解率と，判断基準を示した年度の受講生の正解率について比較を行った．判断

基準を用いない年度の受講生は,1 章で述べた ER モデリングの誤り分析で示した受講生と同じである．再履修者は，比較の母集団から除外し，両年度の受講者が持つ実体関連分析の意義，表記法の内容に関する熟知度のレベルは同水準と仮定した．また，往々にして受講者が前年度の解答を非公式にコピーし入手することの影響が予想されるため，両年度で重複したアプリケーションはサンプルから除外した．ただし，最終的な ER モデルは類似するものの，分析で用いるデータ項目名が異なるアプリケーション（たとえば，在庫管理や販売管理等）については，分析作業の類似性に関して影響なしと判断し，除外はしていない．

比較実験に用いたサンプル母集団は以下のとおり．

母集団		
	1999 年度	9　サンプル
	2000 年度	11　サンプル

(2) 実験の実施方法

両年度とも，被験者を 3 名以下のチームに分割し，チーム毎に分析するアプリケーション分野の決定を自主的に行わせた．データ項目の収集・整理方法，ER モデルの記述規則に関する解説は，両年度とも同時間を費やし，ER モデルの表記方法は「CASE ツールの利用方法」として 3 コマ（270 分）の解説時間を割り当てた．なお，判断基準の適用を試みた 2000 年度は，演習に先立って，判断基準の考え方と判断規則を解説する時間 60 分を設けた．

(3) サンプル・DB アプリケーション

表 2-1a で示すとおりの 20 アプリケーションをサンプルとした．サンプル全体の特性は表 2-1b のとおりである．

表 2-1a 調査対象とした両年度 DB アプリケーション名称

No.	1999 年度ｻﾝﾌﾟﾙ DB ｱﾌﾟﾘｹｰｼｮﾝ名称
1	野球選手成績管理
2	レンタカー
3	図書館
4	パソコン部品
5	通信プロバイダ検索
6	アパート検索
7	競馬レース成績管理
8	カクテル
9	新作ゲーム

No.	2000 年度ｻﾝﾌﾟﾙ DB ｱﾌﾟﾘｹｰｼｮﾝ名称
1	アルバイトシフト管理
2	図書館
3	カラオケ
4	就職企業検索
5	中古車在庫管理
6	電気店営業管理
7	邦楽アーティスト検索
8	コミック雑誌検索
9	競馬血統管理
10	観光地 DB
11	料理レシピ

表 2-1b 調査対象の両年度 DB アプリケーションの各種統計特性

年度 / サンプルアプリケーションの統計的特性	1999 年度	2000 年度
サンプル数	9	11
ER 図に含まれる構成要素数 平均値		
① エンティティ型	4.2	5.7
② 関連型	9.1	8.3
③ 属性	14.9	18.1

(4) 正解の判定方法

ER モデルの正しさの判定は，1999 年度,2000 年度の両年度とも，筆者が完成度にしたがって採点した．また，採点における配点比率も，両年度とも実体と属性等の正しさに対して 70%，関連の正しさに対して 30%とした．採点時の着眼点および配点は，以下のとおりであった．

① 実体の種類と属性の正しさ(配点 50 点)
② 関連付けすべき実体の正しさ（配点 20 点）
③ 関連多重度の設定の正しさ（配点 30 点）

このほかに，主キー属性の設定の正しさも含まれているが，判断基準の有効性評価とは直接関係が薄いため，両年度とも評価実験の得点からは除外してある．

表 2-2　誤りサンプルに占める誤りの種類の比率

誤りサンプルに占める比率 / 誤りの種類	1999 年度	2000 年度
①異なる種類の属性の混在	88%	40%
②属性と実現値の混同	25%	10%
③エンティティ型属性と関連属性の混同	13%	10%
④関連付けすべきエンティティ型の誤り	13%	20%
⑤関連の多重度決定の誤り	67%	60%

2000 年度の演習では，受講者に対して，判断基準にしたがい，機械的に基本データを分類し実体を抽出するよう指示した．その結果，表 2-2 からも分かるように，実体に異なる種類の属性を混在させる誤り①は半減している．ただし，誤りが 0%にいたらなかった理由としては，初期値の決定時点に対する理解不足や決定時点を必要以上に詳細に分けすぎて基本データの分類時に混乱した点等が上げられる．また，関連の多重度の設定に関して

も大きな改善は見られなかった.これは判断基準が,関連の多重度の正確な決定に関して,ビジネス上のルール（＝ビジネスルール）に依存するとして何も定式化しておらず，そのため，両年度ともビジネスルールの分析を同程度にしか行えなかったことを示している.

2.3.2　判断基準の有効性評価

判断基準の有効性を ER 図の完成度から比較するために,各サンプルの正解率平均値の有意性検定（t 検定）を行い表 2-3 のとおりの結果を得た．表 2-3 に示すとおり，正解率の平均値に差がないとする帰無仮説 H_0 は 5%の危険率で棄却され, ER モデルの作成に関して,判断基準を用いた 2000 年度と判断基準を用いない 1999 年度の正解率平均値に有意な差があると結論づけることができた.

表 2-3　平均値の差の有意性検定

比較年度	母数	平均値	標準偏差値
1999 年度	n_1= 9	μ_1= 43.9	s_1= 19.3
2000 年度	n_2= 11	μ_2= 62.7	s_2= 17.2
帰無仮説 H_0 ： 正解率の平均値は $\mu_1 = \mu_2$ である.			
検定統計量 ｜ t ｜ = 2.16 >=1.96 （H_0 は 5%の危険率で棄却）			

2.4　判断基準の一般化に向けての課題

前項で述べた判断基準は，比較的構造が単純な ER モデルに適用することができる.しかしながら，判断基準の適用実験を重ねるにしたがい，新たに次の課題が生じた.

① 再帰構造を含む場合には適用できない.

たとえば，図 2-10 で示すとおりの ER モデルは，導入した 3 つの判断基準のみでは導び き出せない.

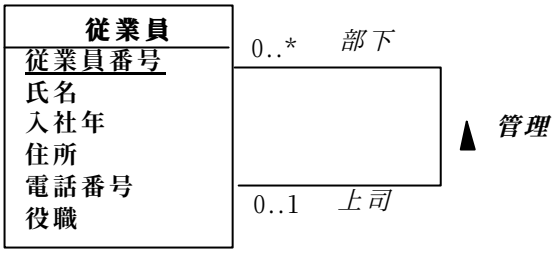


図 2-10　自己再帰構造を持つ ER 図

② メソッドのみを持つクラス構造までは適用できない.

当然ながら，前項で述べた判断基準は Model クラス（＝ER モデル）のモデリングに関す

る判断基準であり，オブジェクト指向ソフトウェア開発で頻繁に出現するメソッドのみを持つたクラス構造に関しては，適用の対象外としている．

これらの問題点を解消するために，3 章では，新たにソフトウェア場の概念と存在寿命の概念を判断基準の基本概念に追加し，「構成演算」として形式的に定義した判断基準について述べる．さらに定義した構成演算を用いると，再帰構造やメソッドを中心にしたクラス構造であっても，構成演算の機械的な適用によって導出できることを，「構造に関するデザインパターン」を事例とした検証実験によって示す．

3章　ソフトウェア場概念によるクラス抽出の判断基準

3.1　クラス分析過程の特徴

ユースケースシナリオ[7]，CRC カード[8]，あるいは帳票分析のいずれの手法を用いるにせよ，要求仕様から属性名やメソッド・インタフェース名の候補を洗い出し，それらをまとめ上げてクラス名を命名し，さらにクラス間の関連を決定する「クラス分析における一連の概念操作」には，「2.1 分析における基本的な概念操作」で述べた特徴に加えて，次のような特徴がある．

《 クラス分析における概念操作の特徴 》

(1) 属性名やメソッド名，クラス名はあくまで候補であって，確定したものでない．それらの存在は，あいまいであり(ソフトウェア開発における不確定性にいて述べたものとして[18]がある)，場合によっては，同一概念が複数の異なる識別名（=ドメイン毎の専門用語や手順名）で認識される．分析されたシステムの構成要素（＝分析要素）は，適宜レビューによって仕様を決定したとき，はじめて確定する．

(2) 分析要素を洗い出す手掛かりは，サービス責任（＝クラスが他のクラスに対して責任持つて提供する機能）を示すメソッド名やそれらを役割としてまとめたクラス名の「意味的な関係」のみに着目して行われる．意味的な関係はあいまいであり，分析者が持つアプリケーション領域の知識（＝ドメイン知識）や分析経験に依存する．

(3) オブジェクトは，存在寿命（Lifetime：生成から消滅までの時間）を必ずもち，分析者はオブジェクトを認識するとき，「2.1 分析における基本的な概念操作」で述べたとおり，暗黙のうちにオブジェクト生成の「きっかけ」となるイベントとその存在寿命を利用している(たとえば，いかなるイベントで生成される永続的なオブジェクトであるか，あるいは特定の時間だけ存在する一時的なオブジェクトであるか等)．

これらの特徴のうち，特に(3)は重要で，2 章で示した「時間的な生成イベントの同一性」がエンティティ型を抽出する良い判断基準になりえたのと同様に，存在寿命の同一性は，洗い出した属性やメソッドから，ボトムアップにクラス構造を抽出する有効な判断基準になりうる．しかしながら，存在寿命の概念を分析において積極的に活用した分析方法論は，目下のところ存在しない．

3.2　ソフトウェア場モデルの基本的考え方

前述の「クラス分析における概念操作の特徴」を自然に説明し，かつクラス抽出の判断基準を形式的に取り扱うために「ソフトウェア場」の概念を導入する[19]～[27]．ソフト

ウェア場の概念とは，分析者が要求仕様から洗い出した属性やメソッドをもとに， Model クラス以外の「一般的なクラス構造」であってもボトムアップに抽出できるよう，分析の判断基準をモデル化するために導入した概念である．分析の初期段階で洗い出した属性やメソッドは，クラスを構成する基本要素であることから，以後，両者を統一的に扱い，「**構成子**」と呼ぶ．

構成子を要求仕様から抽出するときの前提として，それが属性の場合，「他のデータから演算等によって導出できない基本的な属性」を意味し，メソッドの場合は，「他のメソッドと実装内容が一致しない独自のメソッド」を意味するものとする．ソフトウェア場モデルでは，分析者が構成子をまとめ上げ，クラス構造を抽出する作業は，分析者の頭脳の中で構成子同士がその間に働く力によってまとまりを形成し，クラスを構成して行く過程であると捉える．具体的には，構成子をインスタンスとして生成するメタクラス「構成子」を想定し，そこから生成された構成子のインスタンスが，構成子間に働く力にしたがってまとまりを形成し，それぞれクラスとして抽出されるものとして捉える．

ソフトウェア場モデルでは，分析開始前のソフトウェアは，ソフトウェア場Ψ（χ，τ，ν，t）として記述されるものとする．なお，Ψが持つ座標系χ，τ，ν，t の意味については，「3.4 ソフトウェア場が持つ特性」で述べる．

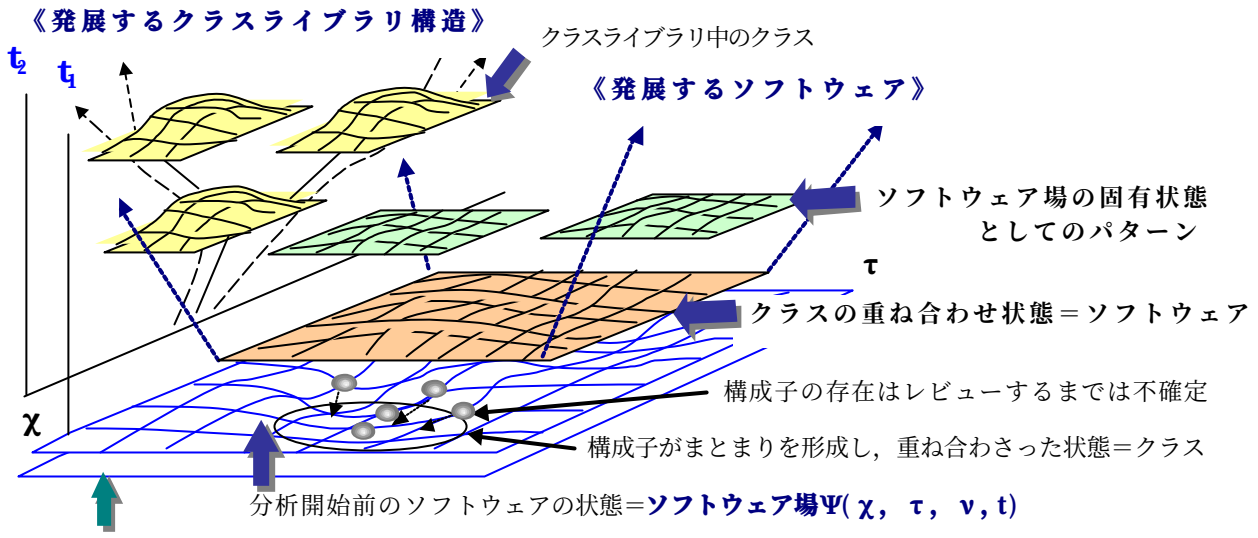


図 3-1　ソフトウェア場の概念

ソフトウェア場内では，構成子の存在は不確定な状態であり，分析によって構成子が確定し，まとまりを形成した状態に相当するクラスは，図 3-1 で示すとおり，それぞれの構成子が重ね合わさった状態として捉える．クラスはさらに寄り集まった重ね合わせ状態を形成し，クラス間の相互作用も含めて定義された状態の総和として「分析後のソフトウェ

ア」状態を形成する．このソフトウェアは常に外部からの要求の変化や，開発工程の進捗等によって，その存在状態を遷移して行く．バージョンアップや保守によるソフトウェアの変更もまた存在状態の遷移を引き起こす原因となるが，しかしソフトウェアの存在状態が時間と共に遷移しても，変化しない部分，すなわち存在状態の変化に対して不変な固有の状態が存在する．このような部分を「状態パターン」と呼び，ソフトウェア分析設計や実装で出現する「パターン」と対応を付ける．パターンはその内部の微細構造の詳細化レベルによって，分析パターンや設計パターン，実装パターンなどが考えられる．

一方,「クラスライブラリ」と呼ぶ状態の総和が別途存在し，それらはクラスの重ね合わせの総和であると同時に，クラス間に「継承」という外部からの要求の変化に応じて状態励起を繰り返して生じた全体構造として捉える.「クラスライブラリ」はソフトウェアから独立しており，ソフトウェアと相互に作用する．

3.3　オブジェクト指向から見たソフトウェア場モデル

オブジェクト指向の視点から見ると，クラス構造を形成する上で用いられる判断基準は，構成子の集合を内部の構成要素として含むメタオブジェクト（＝構成要素を内部に持ったオブジェクトをインスタンスとして生成するオブジェクト）である「ソフトウェア場」が持つメタメソッド（＝メタオブジェクトが持つメソッドで構成要素を構成させるメタルール）に相当する．

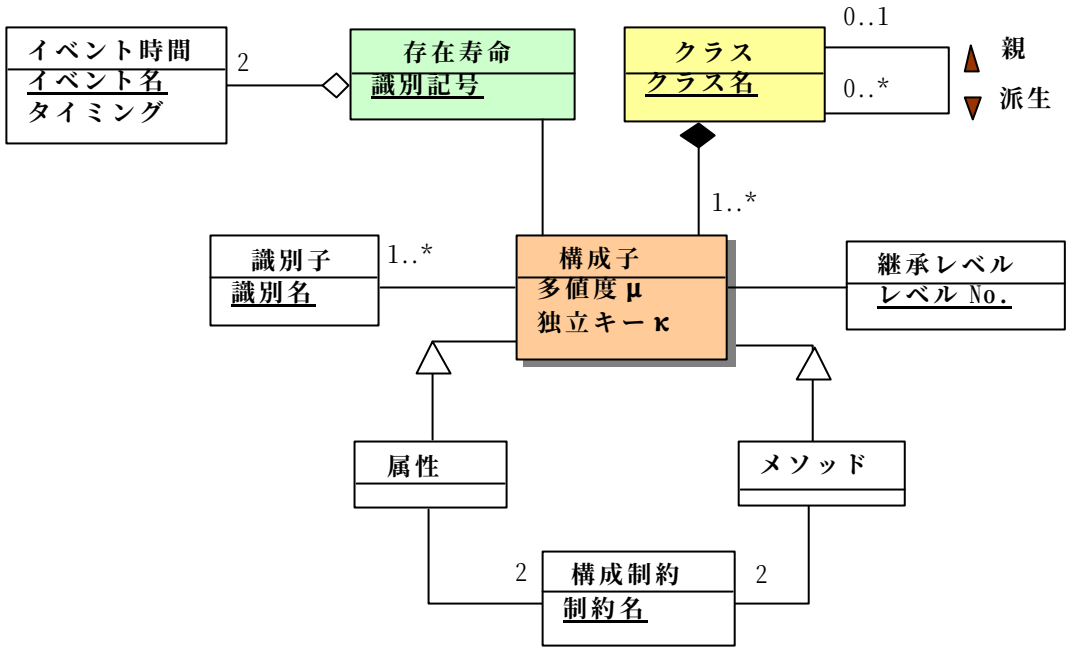


図 3-2　ソフトウェア場を構成する要素のメタモデル

本論では，それらのメタルールを演算形式で定義することを試みる．メタオブジェクト「ソフトウェア場」が内部の構成要素として保有する分析作業中の分析要素は，図 3-2 のメタモデル図として表される．図 3-2 から，「構成子」は「クラス」と存在依存の集約関係（すなわち，クラスと構成子の存続時間は同じ）にあり，唯一の「存在寿命」および「継承レベル」と関連を持つ．「存続寿命」は，２つの「イベント時間」から構成される．さらに，「構成子」は派生クラスとして「属性」，「メソッド」をもち，それぞれの「属性」，「メソッド」は２つの「構成制約」と関連を持つ．「クラス」は階層構造を形成し，最大１つ（＝0..1）の「親クラス」と複数（0..*）の「派生クラス」を持つことを示している．

3.4　ソフトウェア場が持つ特性

ソフトウェア場は，構成子をもとにクラス構造が形成される振る舞いを記述するために導入した仮想的なメタオブジェクトである．しかし，イメージ的には構成子が配置される空間として捉えると理解が容易になる．そこでソフトウェア場を１つの空間を表現したオブジェクトとして捉え，そのメタ属性として次の座標軸を対応させる．

(1) 識別子軸 χ

分析において最も基本的な作業は，要求仕様書で定義された，あるいはドメイン分析で見出された構成子に対する識別名の「命名」である．識別子軸 χ は，ドメイン毎に洗い出された用語を「名義的な指標（すなわち，距離が定義されない）としての座標値」に置き換え，一次元軸上に展開したものである．分析の初期段階で頻繁に出現する１つの概念を複数の同義の識別名で表すような現象（たとえば，給付の代償として受ける金銭を「代金」，「報酬」，「給与」，「俸給」等と複数の用語で言い表す現象）は，識別子軸 χ 上に異なる識別子座標位置に同時に存在する状態として捉える．

(2) イベント時間軸 τ

分析者によって洗い出された属性の初期値を設定する「きっかけ」やメソッドが実際に必要となる「きっかけ」を，時間の概念に類比させ抽象化したものである．「きっかけ」としてのイベントは，たとえば，「注文の発生」や「在庫不足の発生」等の名称（＝イベント名）を持つ．そのイベント名とイベント発生よって構成子が生成・消滅するタイミングを一組にして，タイミングが持つ開始時間の順序関係にしたがって並び換えたものがイベント時間軸 τ である．ここで生成・消滅のタイミングとは，具体的には，構成子が属性の場合は初期値が設定されるタイミングと意味を失うタイミング，メソッドの場合は実装が決定されるタイミングと意味を失うタイミング等を指す（ただし，タイミングは順序関係のみが意味を持つ名義的な指標に過ぎず，距離は定義されない）．τ 軸上にはイベント名が配置される．構成子が生成・消滅するイベント時間軸方向の間隔（＝その間にあるイベント名の数）を以後，「**存在寿命**」と呼ぶ．

(3) 継承レベル軸 ν

クラスの継承階層に対応する座標軸である．分析で洗い出された同じ存在寿命を持つ構成子は，同じ継承レベルに配置する．継承階層の最上位は継承レベル「0」をもち，継承を派生の方向にたどるごとに継承レベルは１ずつ増加して行く．構成子は，次節で述べる排他制約にしたがって順次高い継承レベルに配置され，継承階層を構成して行く．

(4) 実時間軸 t（改訂軸）

分析したソフトウェアが改訂（バージョンアップ）されて行く時間的な経過を示す座標軸である．本論では,議論を分析の初版に限定し，ユーザの要求の変更においてソフトウェアの構成が改訂されて行く過程は考慮しない．したがって，実時間軸は以後の議論では省略して考える．

3.5　構成子の表現と持つべき特性

分析過程において，不確定であいまいな存在として抽出された構成子は，その存在がχ-τ空間に確率的に分布する関数で定義されたものとすると見通しがよくなる．議論を簡単にするため，構成子の存在に関して，次を仮定しても実用性は失わない．

① 識別名は唯一つ決まり，異なる識別名の同義語は存在しない．同義の識別名は，十分検討した結果，統一化された識別名で扱われる（現実には，同義語の分析に多くの時間を費やすが，それらが確定した後を想定しても本研究の意義は失われない）.

② 存在寿命の任意の時点で，構成子は存在するか否かの２値をとり，中間の状態（＝存在が保留された状態）はない.

これらを仮定すると，構成子が存在寿命の間，存在する状態は，式(3-1)で示す２つのステップ関数?(χ，τ)の差で定義される分布関数として表現できる．ここで，ステップ関数?(χ，τ）は，任意の識別子軸の値χとイベント時間軸の値τに対して，τが０未満のときは0，τが０以上のときは１の値をとる存在分布を意味する．τ_1，τ_2は構成子が生成，消滅するイベント時間値を意味し，ρは分析によって見出される構成子の存在確率を示すものとする．すると，構成子の存在状態を示すφは，τ_1～τ_2のイベント時間間隔内で１（＝必ず存在）,それ以外は0(＝存在しない)の２値関数となり，上記の①②の仮定を形式的に表現したものとなる.

$$\phi(\chi,\tau;\tau_1,\tau_2) = ?(\chi,\tau-\tau_1) - ?(\chi,\tau-\tau_2) \qquad (3\text{-}1)$$

要求仕様から多数の構成子が洗い出されたときのソフトウェア場の状態は，図3-3で示すとおり，構成子がχ-τ-ν軸空間に多数存在している状態として表現される．なお，図3-3では，理解を容易にするため，構成子の存在空間χ-τ-ν軸とその存在確率空間χ-τ

- ρ 軸を同時に示している.

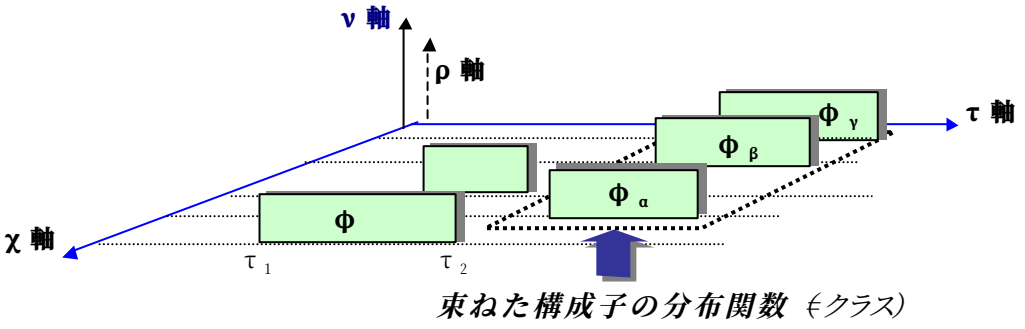


図 3-3 構成子の分布関数

構成子をτ軸方向の分布関数として表現することは，要求仕様から構成子を洗い出すとき，単にその識別名だけでなく，存在寿命も含めて洗い出すべきことを前提とした本研究の基本思想を反映したものになっている．分布関数が持つ存在寿命は，クラス構造を構成する演算を定義する上で重要な役割を持つ．

一方，個々の構成子は，構成子が属性のときには多くの実現値を持ち，構成子がメソッドのときには多くの想定される実装方法を持つ．これはクラスとしての構成子から，何らかの具体的な内容を実現したインスタンス（＝実現値インスタンス）が複数個生成されることに対応している.構成子から生成された実現値インスタンスは,図 3-4 で示すとおり,構成子の分布関数とχ軸,τ軸は共有するが，χ - τ空間とは異なる実現値インスタンスの個数を示す構成子の内部次元μ軸方向に重ね合わさった状態として存在する．重ね合わさった実現値インスタンスの個数は，生成された実現値インスタンスの個数を示している．

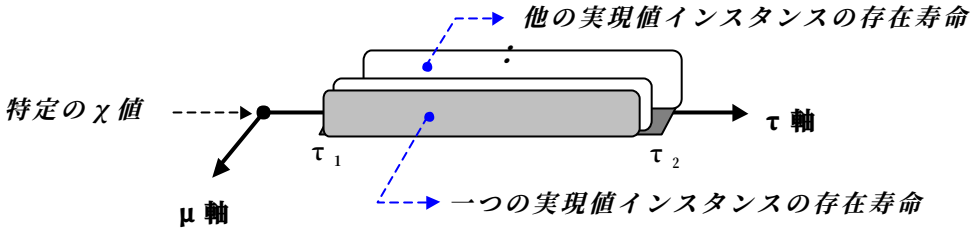


図 3-4 実現値インスタンス空間における構成子

実現値インスタンス空間で，個々の構成子が持つ実現値インスタンスの個数をもとに,構成子に対して次のメタ特性を定義する.

(a) 多値度 μ

インスタンス空間で，ある生成イベント時間に構成子のインスタンス（＝実現値）が生

成されたとき,「同時に」生成されるインスタンスの個数を示す．すなわち，構成子が属性のときは，多値度は「**属性値の個数**」を意味し，構成子がメソッドのときは，「**同じインタフェース名で実装される（あるいは必要になる）メソッドの個数**」を意味する．たとえば，ある生成イベント時間に，同じメソッド･インタフェース名で複数の異なる実装を決定したとき，あるいは実装が必要になったとき，多値度は２以上になる．

(b) 独立キー κ

構成子が持つ存在寿命の決定順序に関する概念であり，構成子 b の存在寿命が，他の構成子集合 a の存在寿命に従属して決まるとき,構成子 b は構成子集合 a に従属すると言う．どの構成子にも従属しない構成子集合は，メタ属性値「独立キー」を持つと言う．構成子が属性のとき，独立キーは，データモデルの主キー概念を言い換えたものに過ぎない．構成子がメソッドのときには，仕様が変更すると，それに伴って変更あるいは再定義されるメソッド（＝フックメソッド）は，変更を生じず単にそれらのメソッドを呼び出すだけのメソッド（＝テンプレートメソッド）に従属する．メソッド間にこのような関係が存在するとき，呼び出し側のメソッドはメタ属性値「独立キー」を持つ．

3.6　クラス構造の構成時に構成子に働く制約

オブジェクト指向では，属性やメソッドの定義に際していくつかの基本的な制約が課せられる．これに対応して，構成子には次の制約を持たせる．

(1) 排他制約 π

構成子集合をクラスにまとめるとき，同じ継承レベルで，同じ識別子座標値に配置できる構成子の数を規定する制約である．構成子が属性かメソッドかによって排他制約の内容は異なる．具体的には，構成子が属性のとき，同じ識別名をもつ構成子は， ソフトウェア場で0, 1以外の排他度を持つことはできない．すなわち, Private属性が持つ制約にもとづいて，同じ識別名を持つ属性は，継承階層上に1つ以上存在しえない（※Public属性であれば，複数存在できるが，情報隠蔽の原則に反するため，本論では除外して考える).

一方，構成子がメソッドのときは，同じ識別名（=メソッド・インタフェース名）が複数存在しても，継承レベルが異なれば，同じ識別子座標値に配置できる．これはメソッドの再定義に相当する．

(2)多値度制約 υ

構成子の実現値が多値のとき（すなわち，一つの属性が同時に複数の値を持つときや，一つのメソッドが同時に複数の実装を持つとき)，単一の実現値しかもたない単値の構成子と混在してクラスを構成できないとする制約である．これは，クラスからインスタンスを生成したとき, すべての構成子の実現値は, 一意に定められるべきことからくる制約で,「2.2.2　判断基準の定式化に関する概念操作」で述べた判断基準と同じである．たとえ

ば，クラス内に多値と単値の構成子が混在したときには，多値の構成子集合を強制的に分離し，単値の構成子集合をもち，複数のインスタンスを同時に生成するクラスに置換する．このとき，多値の構成子集合を分離して形成したクラスは，分離によって残った単値の構成子集合のみを持つクラスに対して「存在依存」する．なぜなら，両者は，同じ存在寿命を持つ構成子集合を強制分離したものであることから，単値のクラスが存在を失えば，多値のクラスもその存在を失うことになるからである．

3.7 存在寿命に基づくクラス構造の構成演算

前章まで述べた構成子のメタ特性，制約を用いて，本章では要求仕様から洗い出した構成子集合をもとに，クラスの抽出とクラス構造の構成演算を定義する．

3.7.1 クラスの抽出演算

構成子の存在が確率的な分布関数で表されることを反映して，次の演算群を定義する．

(1) 分布関数の代数積演算

ソフトウェア場に置かれた２つの構成子の分布関数ϕ_α，ϕ_β間に働く力の強さ$F_{\alpha\beta}$を(式 3-2)で定義する．$F_{\alpha\beta}$は図 3-5 で示すとおり，２つの分布関数ϕ_α，ϕ_βが重なり合う部分の比率（0～1 の範囲の値）を示している．ここで，ϕ_αは生成イベント時間値τ_1と消滅イベント時間値τ_2のイベント時間間隔で値 1 を持つ関数$\phi(\chi,\tau;\tau_1,\tau_2)$の省略形を意味し，$\alpha$は生成・消滅のイベント時間間隔$[\tau_1,\tau_2]$が存在寿命の識別記号$\tau_\alpha$として表すことから，添え字付けしたものである．$\phi_\beta$も同様である．式(3-2)中の存在寿命$\tau_\alpha$，$\tau_\beta$の絶対値は，存在寿命間のイベント数を意味する．以後，τに対するギリシャ文字の添え字は，存在寿命の識別記号を意味し，数字の添え字は，イベント時間軸上の特定の値を意味するものとする．

$$f_{\alpha\beta}(\tau_\alpha;\tau_\beta) \equiv 2*\phi_\alpha(\tau_\alpha)*\phi_\beta(\tau_\beta) \;/\; \{|\tau_\alpha|+|\tau_\beta|\} \qquad (3\text{-}2)$$

ただし　$\alpha \neq \beta$

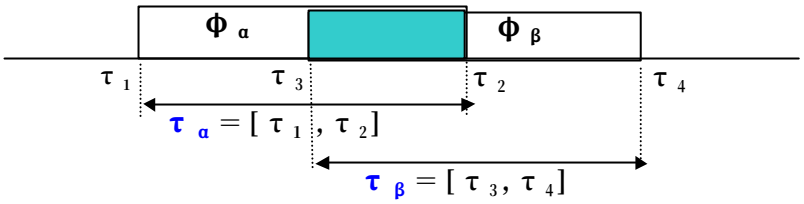


図 3-5　構成子の重なりの割合を示す代数積演算の定義

(2) 分布関数の論理和演算

式(3-2)で示した代数積演算を用いて，２つの構成子分布関数の論理和を算出する演算を

式(3-3)で定義する．図 3-5 を例にとると，$\Phi_\alpha(\tau_\alpha)$，$\Phi_\beta(\tau_\beta)$から $[\min(\tau_1,\tau_3), \max(\tau_2, \tau_4)]$の存在寿命を持つ $S_{\alpha\beta}$を作成する．結果的に，$S_{\alpha\beta}$は$\Phi_\alpha(\tau_\alpha)$と$\Phi_\beta(\tau_\beta)$の重複する部分を削除して，$\Phi_\alpha(\tau_\alpha)$と$\Phi_\beta(\tau_\beta)$の論理和を算出する演算に対応している．

$$S_{\alpha\beta} \equiv \Phi_\alpha(\tau_\alpha)+\Phi_\beta(\tau_\beta)- f_{\alpha\beta}(\tau_\alpha;\tau_\beta) \qquad (3\text{-}3)$$

(3) 実体/クラスの抽出演算

代数積演算をもとにして，構成子を束ねクラスを構成する演算を式(3-4)で定義する．ここで，Πは識別子軸χ上にある任意の２つの識別子α，βを持つた構成子の組み合せを意味する． Select は，それらの組み合せの中から，独立キー特性を持つ構成子を固定して，代数積の値が１である識構成子を選択・抽出して束ねる操作を意味する．

生成・消滅イベント時間の一致を含めた構成子の存在寿命が完全に一致すると，式(3-4)の値は１を取ることから，式（3-4）は，同一の存在寿命を持つ構成子を束ね，クラスを抽出する演算を定義することになる．独立キー特性を持つ構成子が存在しないとき，式(3-4)は定義することができないことから，クラスは必ず独立キー特性を持つた属性（＝主キー属性）かメソッドをもたなければならない．

$$\text{Entity/Class } \Phi \equiv \text{Select } \Pi\ f_{\alpha\beta}(\tau_\alpha;\tau_\beta) \qquad (3\text{-}4)$$

3.7.2 クラスの構成演算

抽出演算で抽出したクラスをもとに，クラス構造を構成する演算を次に定義する．規則中に現れる記号は，それぞれ以下の意味を持つものとする．

《 構成子集合に関して 》

① $\{\tau_\alpha\alpha, \tau_\beta\beta\}$：存在寿命が$\tau_\alpha$，$\tau_\beta$である構成子α，βを要素とする集合．存在寿命の識別記号はτ_αで表す．

② $\tau_\alpha\{\alpha,\beta\}$ ： 同一の存在寿命τ_αを持つ構成子α，βを要素とする集合．$\{\tau_\alpha\alpha, \tau_\alpha\beta\}$ と同じ意味を表す．

③ $\{\ \alpha^*\ \}$ ：多値度が 2 以上の構成子α．

④ $\tau_\beta\beta \in \tau_\alpha\alpha$ ：構成子αは構成子βを構成要素として含む．

⑤ $\{\alpha \mid \beta\}$ ：構成子集合を属性αとメソッドβに分離したときの表現．

《 存在寿命に関して 》

⑥ ⇒ ：構成子集合からクラス構造への変換．

⑦ $(\tau_\alpha \cup \tau_\beta)$ ：２つの存在寿命τ_α，τ_βの論理和．

⑧ $\tau_\alpha[\alpha]$ ：存在寿命の識別子記号がτ_αである構成子αを束ねて構成したクラス．

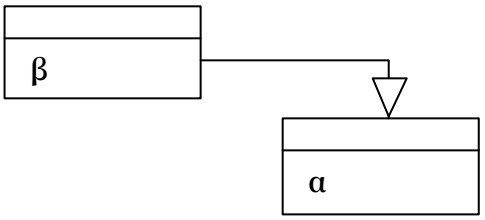
存在寿命は構成子のそれと等しい.

⑨{τ$_{\alpha}$α，τ$_{\beta}$[α]}: 存在寿命がτ$_{\alpha}$の構成子αと存在寿命τ$_{\beta}$の構成子αを持つクラスが混在した中間の状態を示す.

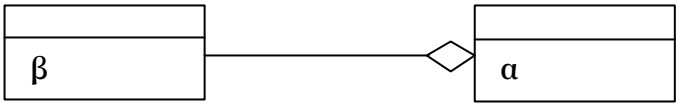
《 クラス構造に関して 》

⑩[α]＋[β]　：構成子α，βを構成要素として持つクラス[α]，[β]が独立した存在のとき，＋記号を独立したクラスの分離記号として用い示すもので，下図のクラス図に対応する．なお，クラス名はこの段階では未定のため，ブランクでしている（以下,同様).

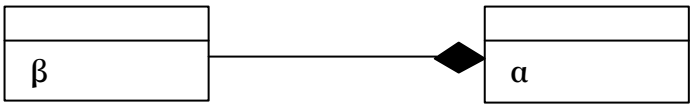
⑪ [β]◿[α]　：構成子αを持つクラス[α]は構成子βを持つクラス[β]の親クラスであることを示すもので，下図のクラス図に対応する.



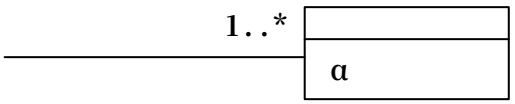
⑫[β]◇[α]　：構成子αを持つクラス[α]と構成子βを持つクラス[β]は集約関係にあることを示すもので,下図のクラス図に対応する.以後,[α]を集約の基底項と呼び，[β]を集約の被基底項と呼ぶ.



⑬ [β]◆[α]　：構成子αを持つクラス[α]と構成子βを持つクラス[β]は存在依存の集約関係（すなわち，[β]が存在するには，[α]が存在しなければならない）にあることを示すもので，下図のクラス図に対応する．そのほかは，⑫と同じ.



⑭ $[\alpha]^*$　　:クラス$[\alpha]$は，複数のインスタンスを同時に生成するクラスを示すもので，下図のクラス図に対応する．



(1)　基本的な構成規則

構成子のメタ特性，および構成子間の制約をもとに，クラス構造を変換する規則を以下に定義する．これらは，クラス構造を構成するときに用いるさまざまな判断を演算形式で表現したものに相当する．

(a) 構成子集合のクラス化に関する規則

i) 多値度が 2 以上の構成子のクラス化

多値度が 2 以上の構成子集合からクラスを構成するとき，「構成子間に働く制約」で述べた多値度制約にしたがって，多値度が 1 である構成子のみをもち，かつ複数のインスタンスを同時に生成するクラス([]記号の右上に*記号をつけて識別する)に置換する．

$$\{\ \tau_\alpha \alpha^*\ \} \Rightarrow \tau_\alpha [\alpha]^* \qquad (3\text{-}5)$$

ii) 集合要素の性質に基づくクラス化

識別名と存在寿命が全く同じ構成子が複数存在しても，クラスとして構成されるものは 1 つとする．

$$\{\tau_\alpha \alpha, \tau_\alpha \alpha\ \} = \tau_\alpha \alpha \Rightarrow (\tau_\alpha [\alpha] + \tau_\alpha [\alpha]) = \tau_\alpha [\alpha] \qquad (3\text{-}6)$$

(b) 排他制約 π に基づく汎化演算

クラスに束ねられるべき構成子集合内に同じ識別名が存在したとき，排他制約 π によって同じ階層レベルに存在できない．そのため，同じ識別名を持つ構成子の分布関数の論理和をとった新たな構成子の分布関数を作成し，その構成子を新たなクラス（＝親クラス）の要素とし，親クラスとの間に継承関係を構成する．この操作を「汎化」と呼ぶ．汎化によって構成された親クラスの存在寿命が，その派生クラスの論理和となる根拠は，ボトムアップに継承を構成したとき，「汎化の対象となったクラスのいずれかが存在すれば，汎化によって作成された親クラスも必ず存在する」ことから，親クラスが持つ構成子の存在寿命は派生クラスの論理和（＝分布関数の論理和）となるためである．したがって，汎化で構成されたクラスの存在寿命は，汎化前の個々のクラスが持つ存在寿命より拡大する．

汎化演算は，オブジェクト指向の基本的な特徴に対応して，(3-7)，(3-8) の 2 通りの定義がある．式中の下線は，汎化演算の対象として着目している構成子を意味する．

i) 継承を利用した一般的な汎化

$$\{\ \tau_\alpha \{\underline{\alpha}, \beta\},\ \tau_\gamma \{\underline{\alpha}, \gamma\}\ \} = \{\ \tau_\alpha \underline{\alpha},\ \tau_\alpha \beta,\ \tau_\gamma \underline{\alpha},\ \tau_\gamma \gamma\ \}$$
$$\Rightarrow (\tau_\alpha [\beta] + \tau_\gamma [\gamma]) \triangle (\tau_\alpha \cup \tau_\gamma)[\underline{\alpha}] \qquad (3\text{-}7)$$

ii）継承を利用して抽象メソッドを再定義する汎化

$$\{\ \tau_{\alpha}\{\underline{\alpha},\beta\},\ \tau_{\gamma}\{\underline{\alpha},\gamma\}\ \} = \{\tau_{\alpha}\underline{\alpha},\tau_{\alpha}\underline{\alpha},\tau_{\alpha}\beta,\tau_{\gamma}\underline{\alpha},\tau_{\gamma}\underline{\alpha},\tau_{\gamma}\gamma\}$$

$$\Rightarrow(\tau_{\alpha}[\underline{\alpha},\beta]+\tau_{\gamma}[\underline{\alpha},\gamma])\triangleleft(\tau_{\alpha}\cup\tau_{\gamma})[\underline{\alpha}] \qquad (3\text{-}8)$$

(3-8)では，$\tau_{\alpha}\alpha$，$\tau_{\gamma}\alpha$を重複して展開したが，これは集合の要素を一時的に重複展開しても，(3-6)で示したとおり，結果的には同等になることによる．

理解を容易にするため，(3-7)，(3-8)で示した変換規則の意味を図 3-6（a）,図 3-6(b)で示す．汎化演算の特徴として，親クラスを作成すると，その影響は次々と継承階層の上位クラスに波及してゆき，最終的に，最上位クラス（=クラス構造全体を代表するクラス）の存在寿命を拡大させる．同様に派生クラスを追加し，クラス構造を拡張（=extend）しても，最上位クラスの存在寿命は拡大する．

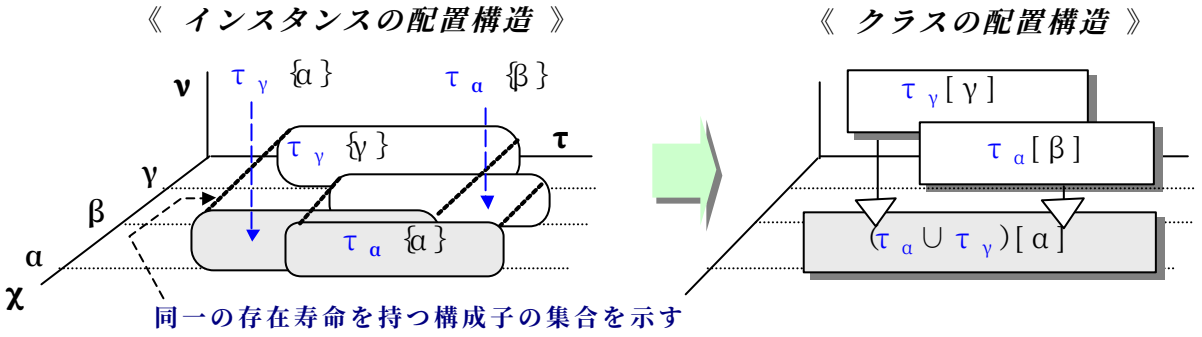


図 3-6(a) 存在寿命から見た一般的な汎化演算

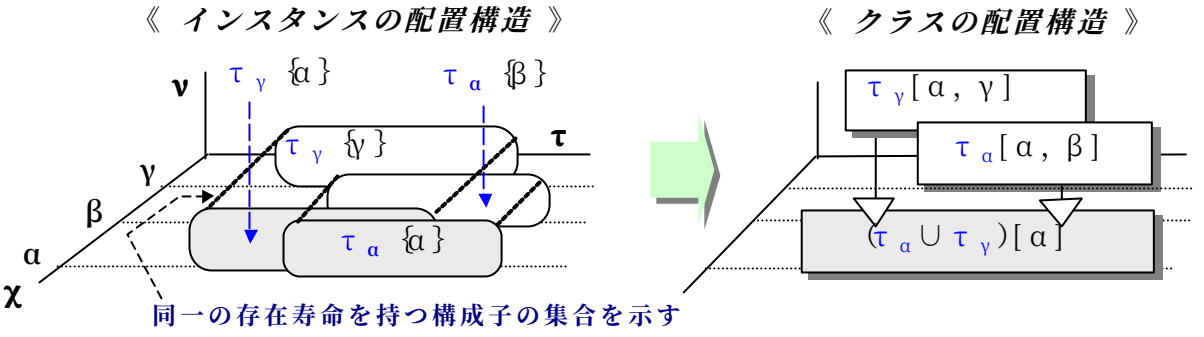


図 3-6(b) 存在寿命から見た抽象メソッドを再定義する汎化演算

（3-7）（3-8）は，構成子から新規に継承構造をボトムアップに構成するときの汎化演算である．しかし，あらかじめ着目した親クラスから派生クラスを派生させることで継承構造を拡張させる場合には，親クラスが元来持つ独自の存在寿命に加えて，派生させるすべての派生クラスが持つ存在寿命のすべてについて論理和をとったものになる．既存のクラス構造を基に派生クラスを拡張するときの親クラスと派生クラスの存在寿命の関係については，4 章で述べる．

(c) 多値度制約 υ に基づく集約演算

同じ存在寿命を持つが多値度は異なる構成子集合が存在したとき，「構成子間に働く制約」で述べた多値度制約 υ にしたがって，集約関係を持つクラス構造を構成する演算である．(3-9) で定義する．

$$\upsilon:\ \tau_{\alpha}\{\alpha, \beta^{*}\} = \{\tau_{\alpha}\{\alpha\}, \tau_{\alpha}\{\beta^{*}\}\}$$
$$\Rightarrow\ \tau_{\alpha}[\beta]^{*} \blacklozenge \tau_{\alpha}[\alpha] \qquad (3\text{-}9)$$

存在寿命から集約演算を見たとき，集約演算は，多値度制約 υ によってクラスを強制的に分離することから，単にクラスが持つ「存在寿命の合計値」を増大する演算であり，クラス構造全体に相当する最上位クラスの存在寿命の増加には寄与しない．

(2) その他の構成規則

その他の構成規則として以下がある．

(a) 存在寿命に包含関係があるときは，他を包含する存在寿命に簡略化する．

$\tau_{\beta} \subset \tau_{\alpha}$ のとき

$$\tau_{\gamma}[\gamma] \triangleleft (\tau_{\alpha} \cup \tau_{\beta})[\alpha] \rightarrow \tau_{\gamma}[\gamma] \triangleleft \tau_{\alpha}[\alpha] \qquad (3\text{-}10)$$

$\tau_{\gamma}[\gamma]$ は，任意の存在寿命を持つクラスを意味する．

(b) 排他制約は多値度制約に優先して適用する(優先度付の理由は 3.7.4 で述べる)．

$$\pi: \{\tau_{\beta}\{\alpha^{*}, \beta\},\ \tau_{\gamma}\{\alpha^{*}, \gamma\}\}$$
$$\Rightarrow (\tau_{\beta}[\alpha,\ \beta] + \tau_{\gamma}[\alpha, \gamma]) \triangleleft (\tau_{\beta} \cup \tau_{\gamma})[\alpha]^{*} \qquad (3\text{-}11)$$

(c) 存在時間がすべて異なる構成子集合の要素が集約関係を含むとき，集約関係の基底項を優先して演算の対象とする．集約関係では，被基底項は基底項に依存していることから，独立した存在の基底項を汎化演算の対象にする．

$$\{\underline{\tau_{\alpha}[\alpha]},\ \tau_{\gamma}[\gamma] \diamond \underline{\tau_{\beta}[\alpha]}\} \quad \text{// 中間状態}$$
$$= (\underline{\tau_{\alpha}[\alpha]} + \tau_{\gamma}[\gamma] \diamond \underline{\tau_{\beta}[\alpha]}) \triangleleft (\tau_{\alpha} \cup \tau_{\beta})[\alpha] \qquad (3\text{-}12)$$

(d) 存在時間がすべて異なる構成子集合の要素が継承を含むとき，継承レベルから見て上位レベル(すなわち，存在寿命が長いレベル)の構成子を汎化演算の対象とする．汎化は，存在寿命の拡大を図ることから導かれる．

$$\{\underline{\tau_{\alpha}[\alpha]},\ \tau_{\gamma}[\gamma] \triangleleft \underline{\tau_{\beta}[\alpha]}\} \quad \text{// 中間状態}$$
$$= (\underline{\tau_{\alpha}[\alpha]} + \tau_{\gamma}[\gamma] \triangleleft \underline{\tau_{\beta}[\alpha]}) \triangleleft (\tau_{\alpha} \cup \tau_{\beta})[\alpha] \qquad (3\text{-}13)$$

(e) 集約関係の基底となる構成子が同一の存在時間を持つとき，それらをまとめる（基底項を優先する理由は(c)と同じである)．

$$\{\tau_{\beta}\beta \in \underline{\tau_{\alpha}\{\alpha,\ \mu_1\}},\ \tau_{\gamma}\gamma \in \underline{\tau_{\alpha}\{\alpha,\ \mu_2\}}\}$$
$$\Rightarrow \{\tau_{\beta}\beta \in \{\underline{\tau_{\alpha}\{\alpha,\ \mu_1\}}, \underline{\tau_{\alpha}\{\alpha,\ \mu_2\}}\},\ \tau_{\gamma}\gamma \in \{\underline{\tau_{\alpha}\{\alpha,\ \mu_1\}}, \underline{\tau_{\alpha}\{\alpha,\ \mu_2\}}\}\} \qquad (3\text{-}14)$$

(3) クラス構造の変換規則

クラスが特定の構造を持つとき，次の変換規則でクラス構造を簡略化する．

(a) 自明なクラス構造の簡略化

$$\tau_{\alpha}[\alpha] \triangle \tau_{\alpha}[\alpha] = \tau_{\alpha}[\alpha] \qquad (3\text{-}15)$$

(b) 派生クラスの置換規則

他のクラスを集約の要素として持つクラスが継承の派生クラスのとき，式（3-16）で示すとおり，集約関係を継承の親クラスに置換する．継承関係をもたない独立したクラス間の集約関係は，互いに存在寿命の合計値を増大させる方向に作用することから導かれる．式(3-16)の置換規則の持つ意味は，図 3-7 で示すとおりである．

$$(\tau_{\beta}[\beta] \diamond \tau_{\gamma}[\alpha]) + \tau_{\gamma}[\alpha] \triangle \tau_{\alpha}[\alpha]$$
$$= (\tau_{\beta}[\beta] \diamond \tau_{\alpha}[\alpha]) + \tau_{\gamma}[\alpha] \triangle \tau_{\alpha}[\alpha] \qquad (3\text{-}16)$$

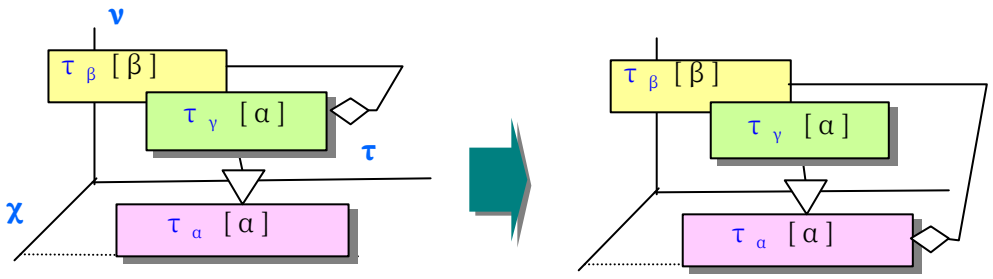


図 3-7　派生クラスの集約関連を置換する規則

3.7.3　構成演算のまとめ

(3-5)から(3-16)に示した構成演算を判断基準としてまとめて自然言語で表現すると以下のとおりになる．

《 **構成演算 0** 》存在寿命が同じ構成子を束ね，クラスとする．

$$\{\tau_{\alpha}\alpha, \tau_{\alpha}\beta, \tau_{\beta}\gamma\} \Rightarrow \tau_{\alpha}[\alpha, \beta] + \tau_{\beta}[\gamma]$$

《 **構成演算 1** 》多値度が 2 以上の構成子集合を持つクラスは，多値度が 1 である構成子のみを持つクラスに置換する．

$$\{\tau_{\alpha}\alpha^{*}\} \Rightarrow \tau_{\alpha}[\alpha]^{*}$$

《 **構成演算 2** 》識別名と存在寿命が全く同じ構成子が複数存在しても，クラスとして構成されるものは 1 つである．

$$\{\tau_{\alpha}\alpha, \tau_{\alpha}\alpha\} = \tau_{\alpha}\alpha \Rightarrow (\tau_{\alpha}[\alpha] + \tau_{\alpha}[\alpha]) = \tau_{\alpha}[\alpha]$$

《 **構成演算 3** 》存在寿命は異なるが同一の識別名を持つ構成子を抜き出し，存在寿命の論理和をとり，汎化した親クラスに配置する．

$$\{\tau_{\alpha}\{\underline{\alpha}, \beta\}, \tau_{\gamma}\{\underline{\alpha}, \gamma\}\} = \{\tau_{\alpha}\underline{\alpha}, \tau_{\alpha}\beta, \tau_{\gamma}\underline{\alpha}, \tau_{\gamma}\gamma\}$$
$$\Rightarrow (\tau_{\alpha}[\beta] + \tau_{\gamma}[\gamma]) \triangle (\tau_{\alpha} \cup \tau_{\gamma})[\underline{\alpha}]$$

《 **構成演算 4** 》存在寿命は異なるが同一の識別名を持つ構成子を複写して，存在寿命の論理和をとり，汎化した親クラスに配置する．

$\{\ \tau_\alpha\{\alpha,\beta\},\ \tau_\gamma\{\alpha,\gamma\}\ \} = \{\tau_\alpha\alpha,\tau_\alpha\alpha,\tau_\alpha\beta,\tau_\gamma\alpha,\tau_\gamma\alpha,\tau_\gamma\gamma\}$

$\Rightarrow(\tau_\alpha[\alpha,\beta]+\tau_\gamma[\alpha,\gamma])\triangleleft(\tau_\alpha\cup\tau_\gamma)[\alpha]$

《 **構成演算 5** 》同じ存在寿命を持つが多値度は異なる構成子集合が存在したとき，集約関係を持つクラス構造を構成する．

$\upsilon:\ \tau_\alpha\{\ \alpha,\beta^*\ \} = \{\tau_\alpha\{\alpha\},\tau_\alpha\{\beta^*\}\ \} \Rightarrow \tau_\alpha[\beta]^* \blacklozenge \tau_\alpha[\alpha]$

《 **構成演算 6** 》存在寿命に包含関係があるときは，他を包含する存在寿命に簡略化する．

$\tau_\gamma \subset \tau_\alpha$ **のとき**

$\triangleleft(\tau_\alpha\cup\tau_\gamma)[\alpha] \rightarrow \triangleleft\tau_\alpha[\alpha]$

《 **構成演算 7** 》汎化では排他制約は多値度制約に優先して適用する．

$\pi:\{\tau_\beta\{\alpha^*,\beta\},\ \tau_\gamma\{\alpha^*,\gamma\}\}$

$\Rightarrow(\tau_\beta[\alpha,\ \beta]+\tau_\gamma[\alpha,\gamma]\ )\triangleleft(\tau_\beta\cup\tau_\gamma)[\alpha]^*$

《 **構成演算 8** 》存在時間がすべて異なる構成子集合の要素が集約関係を含むとき，集約関係の基底項を演算の対象とする．

$\{\tau_\alpha[\alpha]\ ,\ \tau_\gamma[\gamma]\diamond\tau_\beta[\alpha]\ \}$ // 中間状態

$= \tau_\alpha[\alpha] + \tau_\gamma[\gamma]\diamond\tau_\beta[\alpha]\ )\ \triangleleft(\tau_\alpha\cup\tau_\beta)\ [\alpha]$

《 **構成演算 9** 》存在時間がすべて異なる構成子集合の要素が継承を含むとき，継承レベルから見て上位レベル(すなわち存在寿命が長いレベル)の構成子を汎化演算の対象とする．

$\{\tau_\alpha[\alpha],\ \tau_\gamma[\gamma]\triangleleft\tau_\beta[\alpha]\ \}$ // 中間状態

$= \tau_\alpha[\alpha]+\ \tau_\gamma[\gamma]\triangleleft\tau_\beta[\alpha]\ )\triangleleft(\tau_\alpha\cup\tau_\beta)\ [\alpha]$

《 **構成演算 10** 》集約関係の基底となる構成子が同一の存在時間を持つとき，それらをまとめる．

$\{\tau_\beta\beta\in\tau_\alpha\{\alpha,\ \mu_1\},\ \tau_\gamma\gamma\in\tau_\alpha\{\alpha,\ \mu_2\}\ \}$

$\Rightarrow\{\tau_\beta\beta\in\{\tau_\alpha\{\alpha,\ \mu_1\},\tau_\alpha\{\alpha,\ \mu_2\}\},\ \tau_\gamma\gamma\in\{\tau_\alpha\{\alpha,\ \mu_1\},\tau_\alpha\{\alpha,\ \mu_2\}\}\}$

《 **構成演算 11** 》**自明なクラス構造の簡略化**

$\tau_\alpha[\alpha]\triangleleft\tau_\alpha[\alpha]\ = \tau_\alpha[\alpha]$

《 **構成演算 12** 》他のクラスを集約の要素として持つクラスが継承の派生クラスのとき，集約関係を継承の親クラスに置換する．

$(\tau_\beta[\beta]\diamond\tau_\gamma[\alpha])+\tau_\gamma[\alpha]\triangleleft\tau_\alpha[\alpha]=(\tau_\beta[\beta]\diamond\tau_\alpha[\alpha])\ +\tau_\gamma[\alpha]\triangleleft\tau_\alpha[\alpha]$

3.7.4 関連の構成演算

(1) 関連の基数

２つの構成子を束ねたクラスが存在したとき，その間に「関連」が生じるのは，２つのクラスが存在するクラスの存在寿命に重複があるときである．また，任意のクラスから参照関連を持つクラスは，それ以前か，または同時点から存在するクラスである．この事実

を利用すると，独立キー特性を持つ構成子のインスタンスが生成・消滅を繰り返すイベント時間の共有度(=同じイベント時間を共有する比率)を用いて,関連の基数(Cardinality)を定義することができる．たとえば，存在寿命に重複部分を持つクラス間で，主キー特性を持つクラス A の構成子αとクラス B の構成子βが，互いのイベント時間集合ε_1，ε_2の一部分を共有するとき，クラス A，クラス B は 1 対多，もしくは多対多の関連を持つ．すべてのイベント時間集合を共有するときは，１対１の関連を持つ．したがって，要求仕様から，属性の実現値を更新するイベントの発生頻度やメソッドをアクセスするイベントの頻度が判明すると，下記に示す関連の基数を判断するガイドラインが導ける．ただし，関連の基数の妥当性は，ビジネスルール等を加味して慎重に判断する必要があるため，基数決定のガイドラインに留める.なお，以下においてφは空集合，∧は論理積を意味する．

《 １対多関連 》次の条件を満たすイベント集合の要素ε_1，ε_2が存在するとき

/* $\tau_\gamma[\alpha]$と$\tau_\gamma[\beta]$の一部のイベントを共有する */

$$\{\varepsilon_1, \varepsilon_2 \mid \tau_\gamma = (\tau_\alpha \cap \tau_\beta) \neq \phi \ \wedge \ (\varepsilon_1 = \varepsilon_2) \ \wedge \ \varepsilon_1 \in \tau_\gamma[\alpha] \ \wedge \varepsilon_2 \in \tau_\gamma[\beta]\}$$

《 １対１関連 》次の条件を満たすイベント集合の要素ε_1，ε_2が存在するとき

/* $\tau_\gamma[\alpha]$と$\tau_\gamma[\beta]$のすべてのイベントを共有する */

$$\{\varepsilon_1, \varepsilon_2 \mid \tau_\gamma = (\tau_\alpha \cap \tau_\beta) \neq \phi \ \wedge \ \forall \varepsilon_1, \varepsilon_2 \ (\varepsilon_1 = \varepsilon_2) \ \wedge \ \varepsilon_1 \in \tau_\gamma[\alpha] \ \wedge \ \varepsilon_2 \in \tau_\gamma[\beta] \ \}$$

※ α,βは独立キー特性を持つ構成子とする．

3.7.5 構成演算の解の一意性と制約の優先度付け

要求仕様から洗い出した構成子の集合に，前節で述べた構成演算を適用すると，その適用順序によって，最終的に異なるクラス構造が構成される可能性が生じる．存在寿命の視点から汎化演算と集約演算を見ると，汎化演算は，汎化前のクラスより存在寿命が拡大した親クラスを構成する．その影響はクラス構造の全体を意味する最上位クラスまで及び，「クラス構造階層の根元に当たるクラス（＝ルートクラス）の存在寿命」を拡大させる．たとえば，存在寿命τ_1,τ_2を持つクラス 1,クラス 2 を派生クラスとして追加したとき,ルートクラスの存在寿命は，ルートクラスの存在寿命τ_0に（τ_1＋τ_2）を加算した存在寿命になる．一方，集約演算は,「集約関係にあるクラスがそれぞれ持つ存在寿命の共通部分を合算した存在寿命」のみを増大させる．たとえば，存在寿命τ_1,τ_2を持つクラス 1,クラス 2 で集約関係を構成するとき，存在寿命が重複した部分でのみ集約関係が意味をもつことから，構成後のクラス構造の存在寿命は，τ_1,τ_2の共通部分（$\tau_1 \cap \tau_2$）×（集約関係にあるクラス数(＝２)）になる．

存在寿命の視点から見たクラス構造の構成は，クラス構造全体の存在寿命が拡大したとき，妥当な構成になる．実際，クラス構造を拡張する行為は，外部要求の変化に対応して，クラス構造全体を適応させ，存在寿命を拡大させる（=生き延びさせる）行為にほかならない．したがって，汎化演算と集約演算を比較したとき，汎化演算によるクラス構造の存在

寿命の増加は，集約演算による存在寿命の増加より大きくなることから(たとえば，クラス1,クラス2を派生させた場合と集約関係を構成した場合，$(\tau_1 + \tau_2) > 2 \times (\tau_1 \cap \tau_2)$)，汎化演算は集約演算に優先させる．

「構成子間に働く制約」で排他制約を多値度制約に優先させる適用規則は，以上に述べた考え方を反映したものである．

3.8 構成演算によるクラス分析手順

クラスの構成演算は存在寿命を重要な手掛りとして用いる．また，存在寿命は，要求仕様から生成・消滅イベント時間を洗い出すことによって明らかにすることを前提としている．以下では，存在寿命を構成するイベント時間の具体的な決定手順について示す．

3.8.1 きっかけとなるイベント時間の抽出方法

イベント時間は，想定するシステムの状態に変化を起こす「きっかけ」であるイベントを洗い出し，名義的な指標として並べたものであることは既に述べた．これらのイベントの洗い出しは，従来から行われているビジネスプロセス分析やシナリオベースの分析手法に，次の拡張を加えることで行える．

(1) ビジネスプロセス図を拡張した方法

ビジネスプロセス図（＝業務フロー図，またはデータフロー図）を記述し，それぞれのプロセスを起動する「きっかけ」としてのイベントを明らかにして，イベント名とイベント発生時間の順序を定める方法である．次の２段階の手順を踏む．

(a) プロセス毎のイベント名の問合せ

次の問合せを分析者自身が行うことによって，プロセスを起動させるイベント名を明らかにする．

①プロセスで使用されるデータが生成され初期値が決定される「きっかけ」やデータの存在が意味(=管理する価値)を失う「きっかけ」としてのイベント名は何か?

②プロセスが持つ具体的な手順が要求される「きっかけ」や手順が不要になる「きっかけ」としてのイベント名は何か?

たとえば，図書館システムのとき，データ「貸出番号」の値が初めて決定されるきっかけは，「貸出の要求」であり，データの存在が意味を失うきっかけは「貸出図書の返却」である（ただし，貸出履歴を記録しないとしたとき）．また，貸出しに関する諸々の手順は，「貸出の要求」によって必要とされ，「貸出図書の返却」によって不要になる．

イベント名が明らかになると，図3-8で示すとおり，イベント名（EN: Event Name）をビジネスプロセス図にそれぞれ書き込む．図3-8中でイベント名の前に付した＋記号は，データの生成や手順を必要とする「きっかけ」を示し，－記号はデータが存在意味を失うか，あるいは手順が不要になる「きっかけ」を示している．

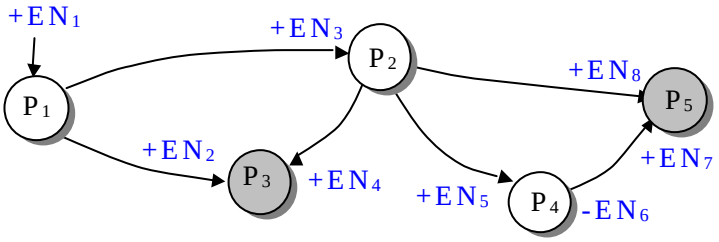


図 3-8　ビジネスプロセス図によるイベント名の抽出

(b) イベント順序の決定

ビジネスプロセス図上で，終端プロセスに至るすべてのパスをたどり，パス毎にイベント名をその順序関係にのみに着目したトポロジカルソート・アルゴリズムによって並び替え，順序木を作成する．次いで順序木をルートから幅優先探索でたどり，順次，イベントの順序を決定する．最後に，それらをイベント軸値として割り当てる．たとえば，図 3-8 で示した例では図 3-9 のとおりになる．

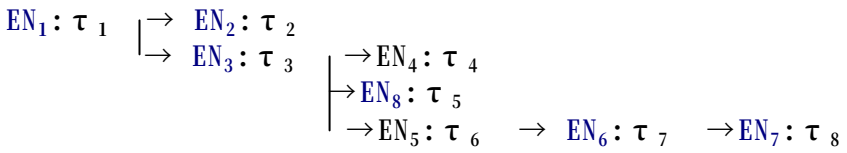
① パス 1：　$P_1 \rightarrow P_5$ 上のイベント順序

$EN_1 \rightarrow EN_3 \rightarrow EN_8$　|　$EN_1 \rightarrow EN_3 \rightarrow EN_5 \rightarrow EN_6 \rightarrow EN_7$

② パス 2：　$P_1 \rightarrow P_3$ 上のイベント順序：

$EN_1 \rightarrow EN_3 \rightarrow EN_4$　|　$EN_1 \rightarrow EN_2$

《 順序木 》



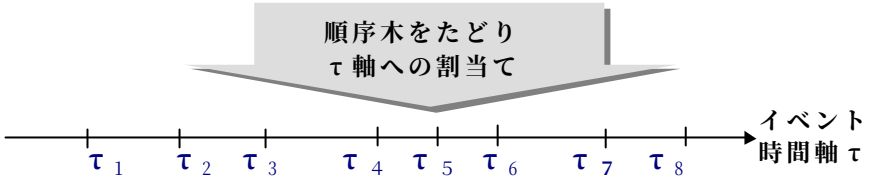


図 3-9　順序木をたどったイベント時間の割当て

(2) ユースケースシナリオを拡張した方法

ユースケース法では，シナリオが起動する事前条件は記述するものの，シナリオを起動する直接的な「きっかけ」を明示的に記述する方法をとっていない．そこで，シナリオを記述するとき，シナリオが起動する「きっかけ」を付加して記述し，イベント名を洗い出す方法である．ユースケースシナリオにおけるケース分けは，発生イベントの種類に強く依存していることから，「きっかけ」の分析は比較的容易に行える．「図書館システム」のユースケース分析で，きっかけを付加して記述したシナリオ・ステップの簡単な記述例を

表 3-1 に示す.

表 3-1 きっかけを付加した 図書館貸出管理」のシナリオ例

きっかけ(ｲﾍﾞﾝﾄ名)	シナリオ・ステップ
[貸出の要求]	学籍番号を入力する
[貸出の要求]	貸出資格をチェックする
[制限の超過]	返却要求を提示する
[貸出の要求]	貸出日付を記録する
[貸出の要求]	返却予定日を書き込む

イベントの順序関係は，シナリオ起動時の「事前条件」，起動完了時の「事後条件」の連鎖関係によって発生順序を決定する．ビジネスプロセス図を拡張した方法と同様にして，イベント時間軸に割り当てる．ユースケースシナリオを用いた場合には，シナリオが意味を失うきっかけ(たとえば異常事態の発生)も含めて，明示的に記述しなければならない．

いずれの方法を用いても，消滅イベントの洗い出しは，発生イベントに比較して困難であることが多い．不明の場合は，システムの存在限界（すなわち，システムが存在するであろう限界時間．通常は無限大）を用いる．

3.8.2 構成演算を基盤にした分析手順図

以上に述べた構成演算の適用を前提とした分析作業の手順は,図 3-10 で示すとおりにまとめられる．図 3-10 の分析手順から，従来の経験的な能力に依存した分析作業は,「構成子の識別名と存在寿命の洗い出し」と「構成演算の適用」作業に還元できることになる．

現時点では，両作業とも人手に頼らざるを得ないが,「構成演算の適用」作業に関しては機械化が可能であり,「構成子の識別名と存在寿命の洗い出し」が完了すれば，自動的にクラス構造図が導出できる可能性は十分ある．

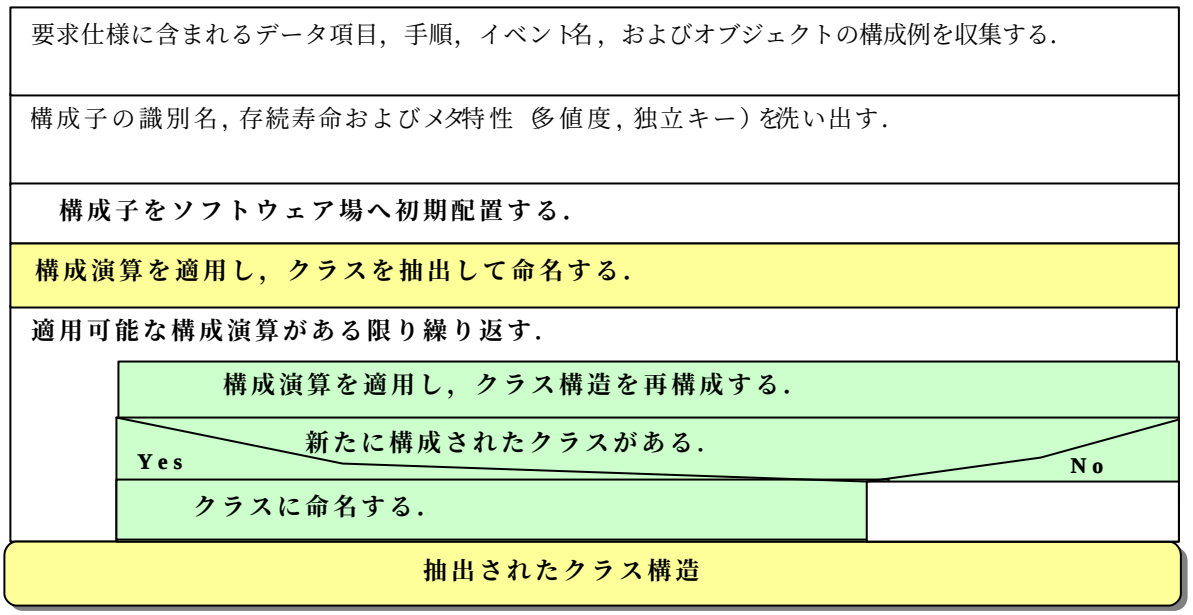


図 3-10 構成演算を基盤にした分析手順（NS 図）

3.9 構成演算の妥当性検証実験

本節では，図 3-10 の手順にしたがって，クラス構造が正しく抽出されるかの妥当性の検証実験を行う．ここでは典型的なクラス構造である「GoF(Gang of Four；E.Gamma,R.Helem,R.Johnson,J.Vlissides ら４人の愛称)の構造に関するデザインパターン[28] [29]」を検証実験用の例題として用いる．GoF の構造に関するデザインパターンの目的は，「経験豊富な優秀な分析設計者が開発アプリケーションの要件，制約等をもとに経験的に用いているクラス構造分析に関する典型的なパターンを整理・体系化し，経験の少ない分析設計者であっても，それらを再利用することで，分析設計の効率化，高品質化を達成する」ことにある． GoF のデザインパターンの概要については，付録 E に記す．

(2) 実験目的

実験の目的は，要求仕様からいろいろな構成子の識別名と存在寿命が洗い出せたと仮定して，それらに対する構成演算の繰り返し適用（＝適用列）によって，典型的なクラス構造である「構造に関するデザインパターン」が抽出できることを示すことにある．

GoF の構造に関するデザインパターンを例題に取り上げた理由は次による．

① 代表的なクラス構造であり，クラス構造を構成する目的や動機,構成要素が明らかにされている．

② デザインパターンを適用し効果を上げたドメインに対して，同様な効果が期待できる．

当然ながらデザインパターンには，構成演算で重要な意味を持つクラスや構成子の存在寿命が明示されてない．そのため，抽出すべきデザインパターンが持つ目的や動機に遡って，構成子の存在寿命にいくつかの仮定を加えなければならない．すると，それらの仮定を調整することによって，恣意的にデザインパターンに合致するクラス構造が抽出できる可能性が生じる．以下では，これらの恣意性を排除するため，洗い出したと仮定する初期構成子集合の妥当性を検証できるよう，初期構成子集合内の各構成子に想定する存在寿命や識別名，ならびに構成子集合に対する要求も明示する．

3.9.1 Adapter パターンの構成

(1) 想定する初期構成子集合

洗い出された初期構成子集合として，識別名と存在寿命がそれぞれ異なるメソッド集合を想定する．たとえば，識別名が Request()で存在寿命が τ_α であるメソッドと，識別名が SpecifiedRequest()で存在寿命が τ_β であるメソッド集合を想定する．ただし，それらの存在寿命には重複があるものとする．

(2) 構成子集合に対する要求

識別名は既存のメソッド集合に一致するが，存在寿命が異なるメソッドを既存のメソッド集合を利用する形で新たに追加したい．具体的には，分析によって新たに存在寿命 τ_γ

を持つメソッド集合{ Request(),SpecifiedRequest() }が明らかになったため，構成子集合に追加し，新たに構成されるクラス構造を求める．ここでτ_γは,既存の存在寿命τ_αを持つ Request()と存在寿命τ_βを持つ SpecifiedRequest()の両者を利用することから，新たに追加する構成子の初期集合が持つ存在寿命は，存在寿命τ_αと存在寿命τ_βの重複部分（$\tau_\alpha \cap \tau_\beta$）内に含まれていなければならない．

(3) 構成演算の適用列

便宜上，Request()をαで，SpecifiedRequest()をβで表す．要求にしたがった構成子集合をΨ_0 とし，式(3-17a)で表す．Ψ_0にクラス構成演算を適用すると，式(3-17b)のとおりに変形できる．

$$\Psi_0 = \{ \tau_\alpha \alpha,\ \tau_\beta \beta,\ \tau_\gamma \{\alpha, \beta\} \} \qquad (3\text{-}17a)$$

ただし，構成子集合に対する要求から $(\tau_\gamma \subset (\tau_\alpha \cap \tau_\beta)) \wedge ((\tau_\alpha \cap \tau_\beta) \neq \phi)$

$$\Rightarrow \ (\tau_\alpha[\alpha] + \tau_\gamma[\alpha,\beta]) \triangleleft (\tau_\alpha \cup \tau_\gamma)[\alpha] + (\tau_\beta[\beta] + \tau_\gamma[\alpha,\beta]) \triangleleft (\tau_\beta \cup \tau_\gamma)[\beta]$$

/--式（3-10）から,*

$\triangleleft(\tau_\alpha \cup \tau_\gamma)[\alpha] \rightarrow \triangleleft\tau_\alpha[\alpha]$, $\triangleleft(\tau_\beta \cup \tau_\gamma)[\beta] \rightarrow \triangleleft\tau_\beta[\beta]$に置換する --/*

$$= \tau_\alpha[\alpha] \triangleleft \tau_\alpha[\alpha] + \tau_\beta[\beta] \triangleleft \tau_\beta[\beta] + \tau_\gamma[\alpha,\beta] \triangleleft (\tau_\alpha[\alpha] + \tau_\beta[\beta])$$

$$= \tau_\alpha[\alpha] + \tau_\beta[\beta] + \tau_\gamma[\alpha,\beta] \triangleleft (\tau_\alpha[\alpha] + \tau_\beta[\beta]) \qquad (3\text{-}17b)$$

(4) 抽出されたクラス構造

求められた式(3-17b)は,図 3-19 で示す Adapter パターンのクラス構造に対応している．

クラス名はボトムアップアプローチの特徴から，クラスが持つ構成子の識別名をもとに命名する（以後のデザインパターンでも同様とする）．

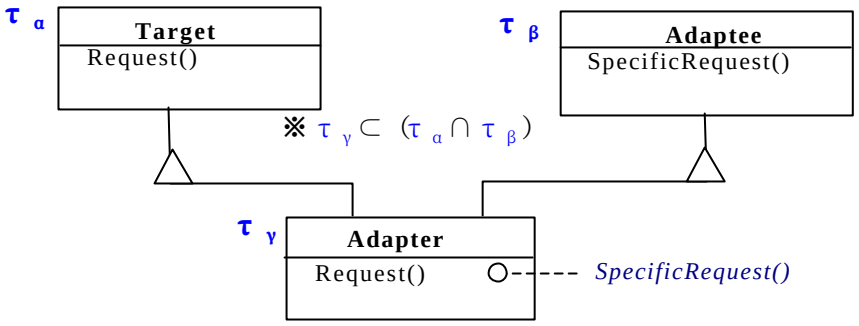


図 3-19 構成演算後の配置構造～Adapter パターン

3.9.2 Bridge パターンの構成

(1) 想定する初期構成子集合

初期構成子集合として,識別名 Operation()で存在寿命τ_αを持つメソッド集合を想定する．

(2) 構成子集合に対する要求

分析から，同じ識別名 Operation()で異なる存在寿命 τ_β，τ_γで複数実装される可能性があるメソッド集合が明らかになったとし，構成子集合に追加して新たに構成されるクラス構造を求める．ここで，τ_β，τ_γの論理和（$\tau_\beta \cup \tau_\gamma$）は，初期構成子集合の存在寿命$\tau_\alpha$に等しいものとする．

(3) 構成演算の適用列

Operation()をαで表すものとする．初期のメソッド集合に異なる存在寿命τ_β，τ_γで複数実装されるメソッドを新たに構成子集合に追加したものを式(3-18a)で表す．これにクラス構成演算を適用すると式(3-18b)に変形できる．

$$\Psi_0 = \{\ \tau_\alpha \alpha,\ \tau_\beta \alpha^*,\ \tau_\gamma \alpha^*\ \} \qquad (3\text{-}18a)$$

ただし，構成子に対する要求から　$\tau_\alpha = (\tau_\beta \cup \tau_\gamma)$

$$\Rightarrow \{\ \underline{\tau_\alpha}[\alpha]\ ,\ (\tau_\beta[\alpha] + \tau_\gamma[\alpha])\ \triangleleft (\tau_\beta \cup \tau_\gamma)[\alpha]^*\ \}$$

$$= \{\ \underline{\tau_\alpha}[\alpha]\ ,\ (\tau_\beta[\alpha] + \tau_\gamma[\alpha])\ \triangleleft \tau_\alpha[\alpha]^*\ \}$$

/ $\tau_\alpha = (\tau_\beta \cup \tau_\gamma)$*であるので式（3-9）を適用して共通する* $\tau_\alpha[\alpha]$*で集約化* */*

$$= (\tau_\beta[\alpha] + \tau_\gamma[\alpha])\ \triangleleft \tau_\alpha[\alpha]^* \diamond \tau_\alpha[\alpha] \qquad (3\text{-}18b)$$

(4) 抽出されたクラス構造

式(3-18b)は，図 3-20 で示す Bridge パターンが持つクラス構造に対応している．

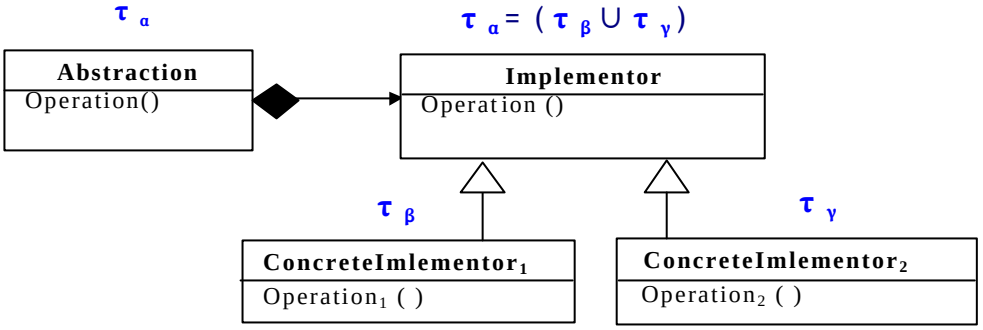


図 3-20　構成演算後の配置構造～Bridge パターン

3.9.3 Composite パターンの構成

(1) 想定する初期構成子集合

Composite パターンの初期構成子集合は多少複雑であることから，インスタンス構造(＝事例)を手掛りにする．Composite パターンのインスタンス構造は，図 3-21 で示すとおり，一つのインスタンスが再帰的に他のインスタンスを含む構造である．インスタンス構造をもとに洗い出された識別名は，各インスタンス上の四角枠内に，また，存在寿命はインスタンスの左上に示す．メソッド Draw()をαとし，Add()，Remove()，GetChild()をμ_1，μ_2，μ_3で表すものとする．インスタンス aline，aRectangle，aPicture の存在寿命をそれ

ぞれτ_α，τ_β，τ_γとしたとき，初期構成子集合は，式(3-19a)で表せる．

(2) 構成子集合に対する要求

インスタンス構造が持つ条件を満たしたクラス構造を求める．

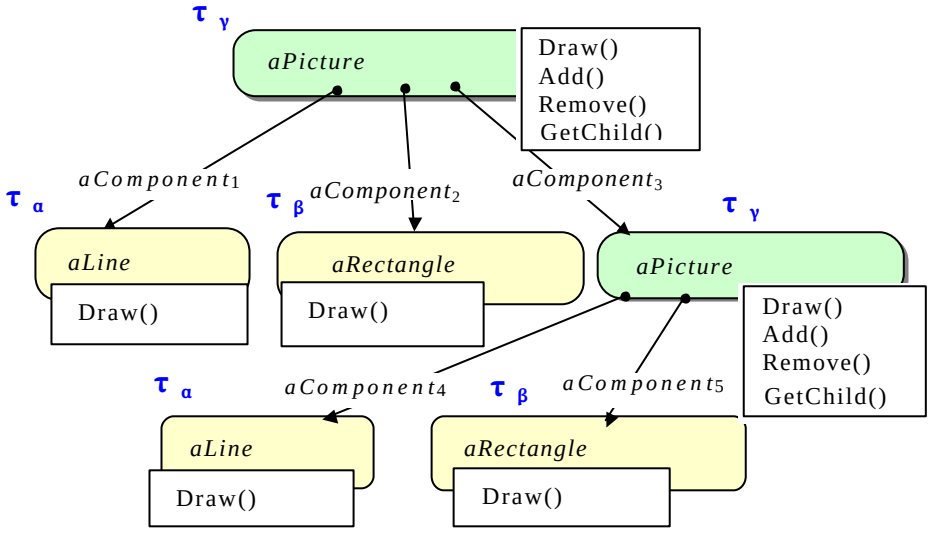


図 3-21 Composite の出発点となるインスタンス構造

(3) 構成演算の適用列

(式 3-19a)にクラス構成演算を適用すると，最終的に式(3-19b)に変形できる．

$$\Psi_0 = \{ \tau_\alpha \alpha , \tau_\beta \alpha , \underline{\Psi_0} \in \tau_\gamma \{ \alpha , \mu_1 , \mu_2 , \mu_3 \} \} \qquad (3\text{-}19a)$$

/ 共通する識別子αの項を括り出し継承構造化 */*

$$\Rightarrow (\tau_\alpha [\alpha], \tau_\beta [\alpha] , \underline{\Psi_0} \diamond \tau_\gamma [\alpha , \mu_1 , \mu_2 , \mu_3]) \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma) [\alpha]$$

*/*親クラスを展開する．Ψ_0は$\triangle(\tau_\alpha \cup \tau_\beta \cup \tau_\gamma) [\alpha]$を最上位クラスとする構造自身 */*

$$= \tau_\alpha [\alpha] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma) [\alpha] + \tau_\beta [\alpha] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma) [\alpha]$$
$$+ [\underline{\Psi_0}] \diamond \tau_\gamma [\alpha , \mu_1 , \mu_2 , \mu_3] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma) [\alpha] \qquad (3\text{-}19b)$$

(4) 抽出されたクラス構造

(3-19b)は，図 3-22 で示す Composite パターンが持つクラス構造に対応している．

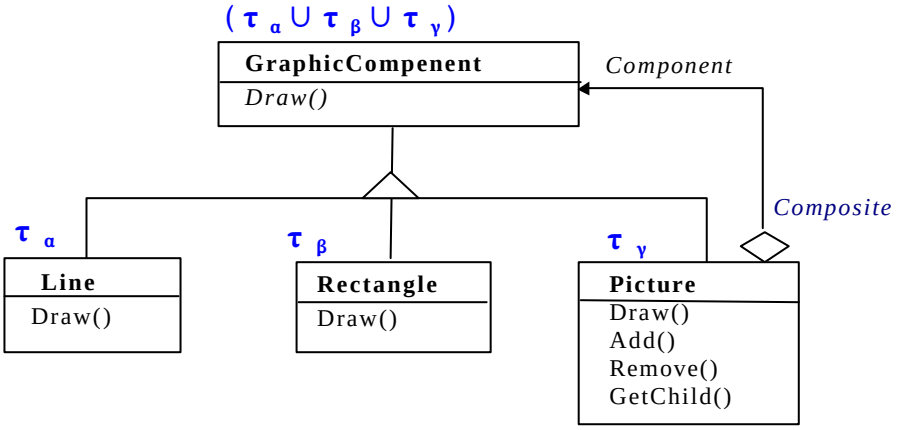


図 3-22 構成演算後の配置構造～Composite パターン

3.9.4 Decorator パターンの構成

(1) 想定する初期構成子集合

Decorator パターンの初期構成子集合も Composite パターン同様，図 3-23 で示すインスタンス構造(=事例)を手掛りにする．メソッド Draw()を α，DrawScrollTo()を μ_1，DrawBorder()を μ_2 で表す．

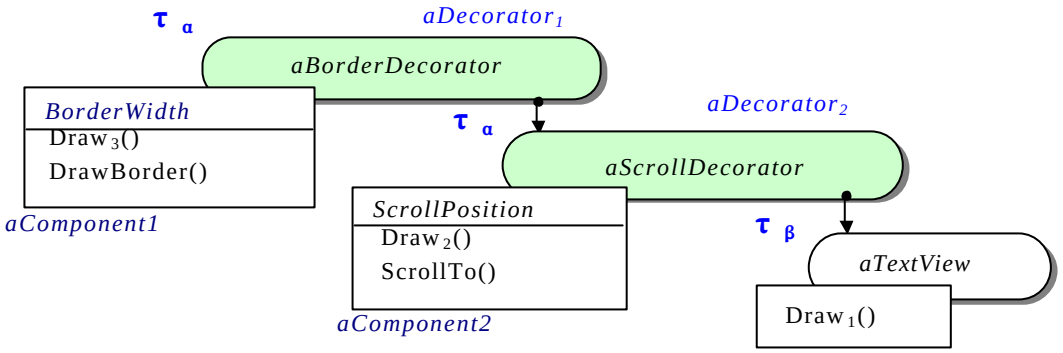


図 3-23 Decorator の出発点となるインスタンス構造

インスタンス aTextView の持つ Draw()の存在寿命は，明らかに aScrollDecorator や aBorderDecorator が持つ存在寿命 τ_α とは異なる存在寿命 τ_β を持つことから，初期構成子集合は式(3-20a)で表せる．

(2) 構成子集合に対する要求

インスタンス構造が持つ条件を満たすクラス構造を求める．

(3) 構成演算の適用列

式(3-20a)に対して，構成子集合の簡略化，およびクラス構成演算を適用すると，最終的に式(3-20b)に変形できる．

$$\Psi_0 = \{\ \tau_\beta\alpha,\ \tau_\alpha\{\alpha,\ \mu_1\} \in \underline{\tau_\alpha\{\alpha,\ \mu_2\}},\ \tau_\beta\alpha \in \underline{\tau_\alpha\{\alpha,\ \mu_1\}}\ \} \quad (3\text{-}20a)$$

/ 共通する $\tau=\tau_\alpha$, $\chi=\alpha$ を持つ項を括り出す */*

$$= \{\ \tau_\beta\alpha,\ \tau_\alpha\{\alpha,\ \mu_1\} \in \{\ \underline{\tau_\alpha\{\alpha,\ \mu_1\}},\ \underline{\tau_\alpha\{\alpha,\ \mu_2\}}\ \},$$
$$\tau_\beta\alpha \in \{\ \underline{\tau_\alpha\{\alpha,\ \mu_1\}},\ \underline{\tau_\alpha\{\alpha,\ \mu_2\}}\}\ \}$$

/ 共通項 $\{\tau_\alpha\{\alpha,\ \mu_1\},\ \tau_\alpha\{\alpha,\ \mu_2\}\}$ を汎化し，式(3-16)から作成した親クラス $\tau_\alpha[\alpha]$ で集約の基底項を置換する */*

$$\Rightarrow \{\ \tau_\beta[\alpha],\ (\underline{\tau_\alpha[\alpha,\ \mu_1]} + \underline{\tau_\alpha[\alpha,\ \mu_2]})\triangleleft\underline{\tau_\alpha[\alpha]},$$
$$\tau_\alpha[\alpha,\ \mu_1]\diamond\tau_\alpha[\alpha],\ \tau_\beta[\alpha]\diamond\tau_\alpha[\alpha]\ \}$$
$$= \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] +$$
$$((\tau_\alpha[\alpha,\ \mu_1]+\tau_\alpha[\alpha,\ \mu_2])\triangleleft\tau_\alpha[\alpha])\ \triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$$
$$+ (\underline{\tau_\alpha[\alpha,\ \mu_1]}+\tau_\beta[\alpha]\_)\diamond\tau_\alpha[\alpha]$$

/ 被集約項 $\tau_\alpha[\alpha,\ \mu_1]$ を汎化した親クラス $\tau_\alpha[\alpha]$ が存在するので置換する */*

$$= \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$$
$$+ ((\tau_\alpha[\alpha,\ \mu_1]+\tau_\alpha[\alpha,\ \mu_2])\triangleleft\tau_\alpha[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$$
$$+ (\tau_\alpha[\alpha]+\tau_\beta[\alpha])\diamond\tau_\alpha[\alpha]$$
$$= \tau_\beta[\alpha]\ \triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$$
$$+ ((\tau_\alpha[\alpha,\ \mu_1]+\tau_\alpha[\alpha,\ \mu_2])\triangleleft\tau_\alpha[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$$
$$+(\tau_\alpha[\alpha]+\tau_\beta[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]\diamond\tau_\alpha[\alpha] \quad (3\text{-}20b)$$

(4) 抽出されたクラス構造

式(3-20b)は，図 3-24 で示す Decorator パターンが持つクラス構造に対応している．

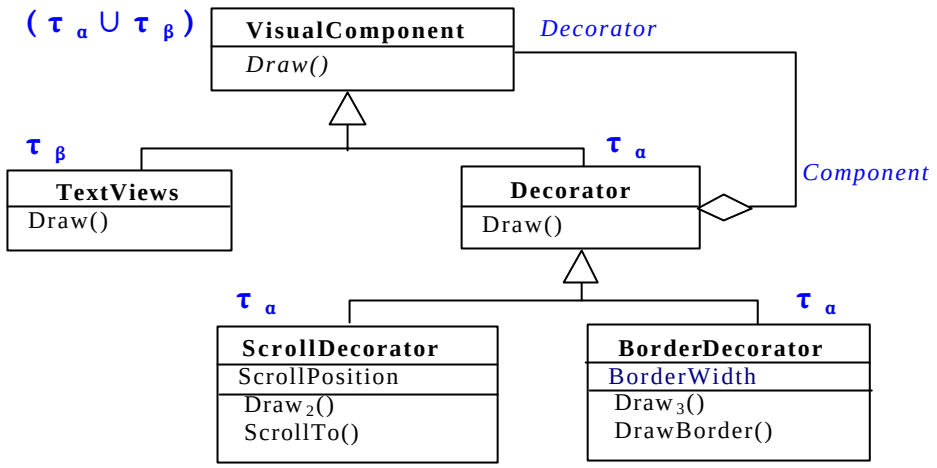


図 3-24 構成演算後の配置構造～Decorator パターン

3.9.5 Proxy パターンの構成

(1) 想定する初期構成子集合

初期構成子集合として，同じ識別名であるが，存在寿命τ_α，τ_βをもち，かつ他の一方に含まれる（すなわち，$\tau_\alpha \supset \tau_\beta$）メソッド集合 {Draw(), Store(), Load()} と存在寿命が　τ_α，τ_βである属性集合 {image}, {fileName} を想定する．属性 image, fileName をα，β，メソッド Draw(), Store(), Load()をμ_1，μ_2，μ_3とすると，初期構成子集合は式（3-21a）で表せる．

(2) 構成子集合に対する要求

存在寿命τ_α，τ_β毎にクラスを形成するとき，長い存在寿命τ_αを持つ構成子は，短い存在寿命τ_βを持つ構成子の代理（あるいはバックアップ）として機能する．ここで，短い存在寿命τ_βを持つクラスは，長い存在寿命τ_αを持つクラスの属性 image をそれと同じ存在寿命を持つメソッド Load()を介してアクセスするものとする．

(3) 構成演算の適用列

式（3-21a）にクラス構成演算を適用すると式（3-21b）に変形できる．

$$\Psi_0 = \{\ \tau_\alpha\{\alpha \mid \mu_1, \mu_2, \mu_3\},\ \tau_\beta\{\beta \mid \mu_1, \mu_2, \mu_3\}\ \} \qquad (3\text{-}21a)$$

$$\Rightarrow (\tau_\alpha[\alpha \mid \mu_1, \mu_2, \mu_3] + \tau_\beta[\beta \mid \mu_1, \mu_2, \mu_3]) \triangleleft (\tau_\alpha \supset \tau_\beta)\ [\mu_1, \mu_2, \mu_3] \qquad (3\text{-}21b)$$

(4) 抽出されたクラス構造

式（3-21b）は，図 3-25 で示す Proxy パターンが持つクラス構造に対応する．

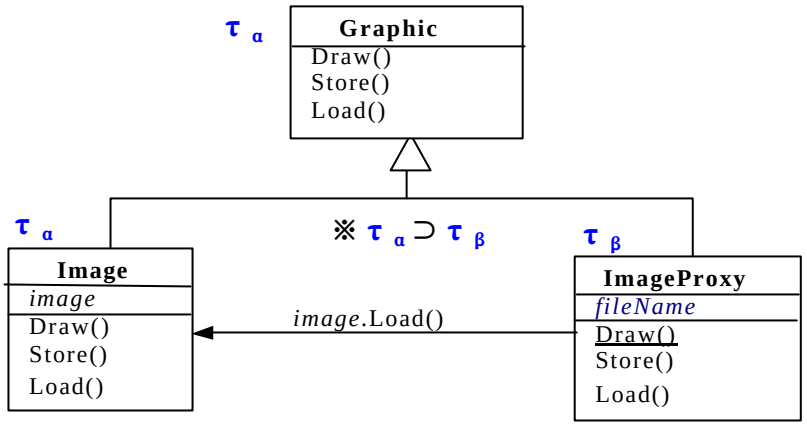


図 3-25 構成演算後の配置構造～Proxy パターン

オブジェクト間のメッセージ交換は，クラス間でのメソッドの配置に重要な働きを持つが，構成演算による方法では，メッセージ交換はメソッドの持つイベント時間上での前後関係と存在寿命の重なり度合いに規定される（実際，存在寿命に重なりがなければ，直接，メッセージ交換できない）．したがって，構成演算を適用する段階では，メッセージ交換は考慮に入れず，クラス構造が抽出された後に検討する．メッセージ image.Load()による メソッド呼び出しは，クラス構造の構成後に構成子集合に対する要求から追加する．

3.9.6　Facede パターンの構成

(1) 想定する初期構成子集合

洗い出された構成子集合が存在寿命も異なるいくつかの構成子集合(＝クラス)にまとめられている.

(2) 構成子集合に対する要求

存在寿命の異なる構成子集合に対するアクセスは，統一的なインタフェースを持つた新たな構成子集合$\underline{\delta}$を介して行う.

(3) 構成演算の適用列

Facede パターンは，オブジェクト指向の基本的な特徴に基づく構成演算から導くことはできない．逆に言うと Facede パターンはオブジェクト指向特有のパターンではない．強いて構成子集合$\{\tau_\alpha\{\alpha\},\tau_\beta\{\beta\},\tau_\gamma\{\gamma\}\}$からクラス構造を構成させる過程を示すと式(3-22)のとおりとなる.汎化演算が継承レベル軸方向に上位クラスを作成するのと同様に，識別子軸方向に新たな識別子集合を作成して束ね，それぞれのクラスが持つ分布関数の論理和である存在寿命を持つたクラス$(\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\underline{\delta}]$を生成することに相当する．オブジェクト指向特有のパターンでないため，本論では Facede パターンについては，これ以上立ち入らない.

$$\Psi_0=\{\tau_\alpha\{\alpha\},\tau_\beta\{\beta\},\tau_\gamma\{\gamma\}\}\Rightarrow(\tau_\alpha[\alpha]+\tau_\beta[\beta]+\tau_\gamma[\gamma])\diamondsuit(\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\underline{\delta}]\qquad(3\text{-}22)$$

3.9.7　導出実験の考察

以上,3.9.1 から 3.9.5 までに示した検証実験から，各デザインパターンが想定する構成子集合の存在寿命に想定した諸関係が成立していれば，構成演算の機械的な適用が意味を持ち,分析者によるデザインパターンの適用と同じ分析結果をもたらすことが検証できた.

3.10　Model クラスモデリングへの回帰的検証

前項では，構成演算の妥当性を検証するために，要求仕様から抽出した識別子，存在寿命を持つ構成子集合に構成演算列を作用させることで，代表的なクラス構造である「構造に関する代表的なデザインパターン」に等しいクラス構造が導き出せることを検証した．以下では，これらの構成演算を用いることで，第 2 章で用いた ER モデルの事例が正しく導き出せるか検証，すなわち，概念クラスモデル（Model クラスモデル）のエンティティ型が正しく導き出せるかの検証を行う．なお，適用する構成演算名は「3.7.3　構成演算のまとめ」で列挙した名称を用いるものとする.

(1) 基本データ集合ψ_1

帳票類に記載されている基本データを収集し，整理した結果，次で定義される存在寿命集合τ_α，τ_β，τ_γと構成子集合ψ_1が洗い出されたものとする．ここで下線を付した構

成子の識別名は，収集時に同一の実現値が存在することがないキー属性を持つことを意味する．存在寿命τ_α，τ_β，τ_γは，それぞれ鍵括弧内の第１項，第２項として，存在の始端イベント時間と終端イベント時間のペアによって表すものとする（以後も同様の表記法を採る）．なお，「UnKnown」は，イベント時間が不明であることを意味する．

τ_α：[顧客登録，顧客抹消]

τ_β：[商品登録，商品抹消]

τ_γ：[注文発生，UnKnown]　　ただし，$(\tau_\gamma \subset \tau_\alpha) \wedge (\tau_\gamma \subset \tau_\beta)$

$\psi_1 = \{\tau_\alpha$ 顧客番号，τ_γ 注文日付，τ_γ 注文番号，τ_β 商品番号，τ_α 顧客名，
τ_α 住所，τ_α 電話番号，τ_β 商品名，τ_β 商品単価 $\}$

構成子集合ψ_1に重複する識別子が存在しないことから，最も基本的な《 構成演算 0 》を適用し，式(3-23)で示すとおり，ψ_1に含まれる同じ存在寿命を持つ構成子を束ねて，エンティティ型を構成する．

$\psi_1 \Rightarrow$　τ_α[顧客番号，顧客名，住所，電話番号] ＋
τ_β[商品番号，商品名，商品単価] ＋ τ_γ[注文番号，注文日付]　　(3-23)

エンティティ型名は，束ねた構成子を属性として持つような概念名を命名する（実際，システムの実現段階では概念名は記号にすぎない）．この結果は，分析者による分析結果である図 3-26 と対応し，誤りなく抽出できたことになる．

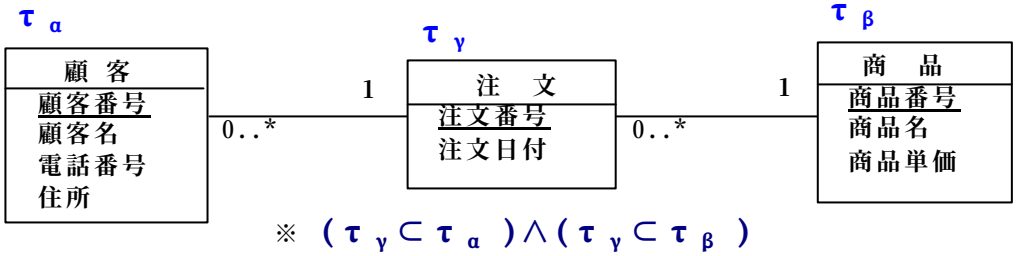


図 3-26　基本データ集合ψ_1の分類に対応する ER モデル

図 3-26 では，エンティティ型間の関連は，存在寿命の包含関係の有無を手掛かりにして記述しているが，厳密な意味での関連付けは，ビジネスルールにしたがって行わなければならない．以後の ER モデリングにおいても同様とする．

(2) 基本データ集合ψ_2

同様にして帳票類に記載されている基本データを整理した結果，次で定義された存在寿命集合τ_α，τ_βと構成子集合ψ_2が洗い出されたものとする．ψ_2中の明細番号，明細注文数は，注文発生イベントを始端とする存在寿命τ_βにおいて，同時に複数個発生するので，＊記号を付記する．明細番号も主キー属性を持つことを破線で示しているが，これは

注文番号と合成されたとき，主キーの一部を構成することを示すものとする．

τ_α：[商品登録，商品抹消]

τ_β：[注文発生，UnKnown]　　ただし，（ $\tau_\beta \subset \tau_\alpha$ ）

ψ_2 = { τ_β 注文日付，τ_β 注文番号，τ_α 商品番号，　τ_α商品名，τ_α商品単価，
τ_β 明細番号*，τ_β明細注文数* }

構成子集合ψ_2には重複する識別子は存在しない．そこで《 構成演算 0 》を適用して同じ存在寿命の構成子を分離し，次いで《 構成演算 5 》を適用すると式(3-24)で示す結果が導かれる．この結果は，分析者による分析結果である図 3-27 と対応し，誤りなく抽出できたことになる．

$\psi_2 \Rightarrow$ τ_α[商品番号，商品名，商品単価] +
τ_β{ 注文番号，注文日付，明細番号*，明細注文数* }
= τ_α[商品番号，商品名，商品単価] +
τ_β[明細番号，明細注文数]* ◆ τ_β[注文番号，注文日付]　　(3-24)

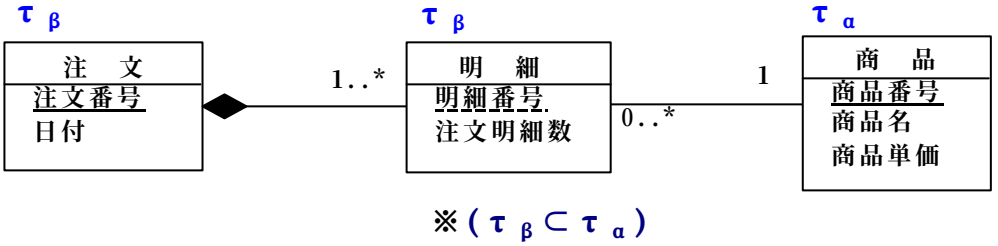


図 3-27　基本データ集合ψ_2の分類に対応する ER モデル

(3) 基本データ集合ψ_3

同様にして，以下の存在寿命を持つ構成子集合ψ_3が洗い出されたものとする．

τ_α：[顧客登録，顧客抹消]

τ_β：[商品登録，商品抹消]

τ_γ：[注文発生，UnKnown]　ただし，（ $\tau_\gamma \subset \tau_\alpha$ ）　∧　（ $\tau_\gamma \subset \tau_\beta$ ）

ψ_3= { τ_γ　注文日付，τ_α　顧客番号，τ_α　顧客名，τ_β　商品番号，τ_β　価格}

基本的な構成子集合の構成は，ψ_1 と同じである．したがって，式(3-25)に示すとおり，3 つのエンティティ型に分離できる．

$\psi_3 \Rightarrow \tau_\alpha$[顧客番号，顧客名] + τ_β[商品番号，価格] + τ_γ[注文日付]　　(3-25)

しかしながら，τ_γ[注文日付]の注文日付は，重複して実現値が発生する可能性があるため，一意的な識別の役割を持つ主キー属性を持ちえない．さらに注文日付は，顧客が

商品の存在寿命が重複するときのみ意味を持つので，関連の持つ属性とする．すると分析者による分析結果である図 3-28 に対応することになる．

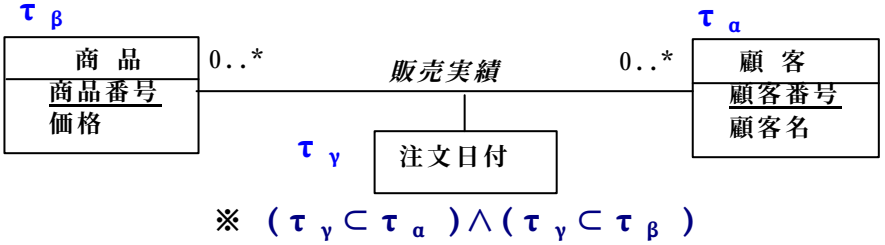


図 3-28　基本データ集合ψ3の分類に対応する ER モデル

(4) 基本データ集合ψ4

同様に，以下の存在寿命を持つ構成子集合ψ4が洗い出されたものとする．

τ α：[顧客登録，顧客抹消]
τ β：[発注業者登録，発注業者抹消]
τ δ：[顧客からの受注，UnKnown]
τ κ：[在庫不足発生，UnKnown]
τ λ：[倉庫番号割当，倉庫番号削除]
τ μ：[運送業者登録，運送業者抹消]
ただし，(τ δ⊂τ α)∧(τ δ⊂τ β)∧(τ δ⊂τ μ)，(τ κ⊂τ λ)∧(τ κ⊂τ β)

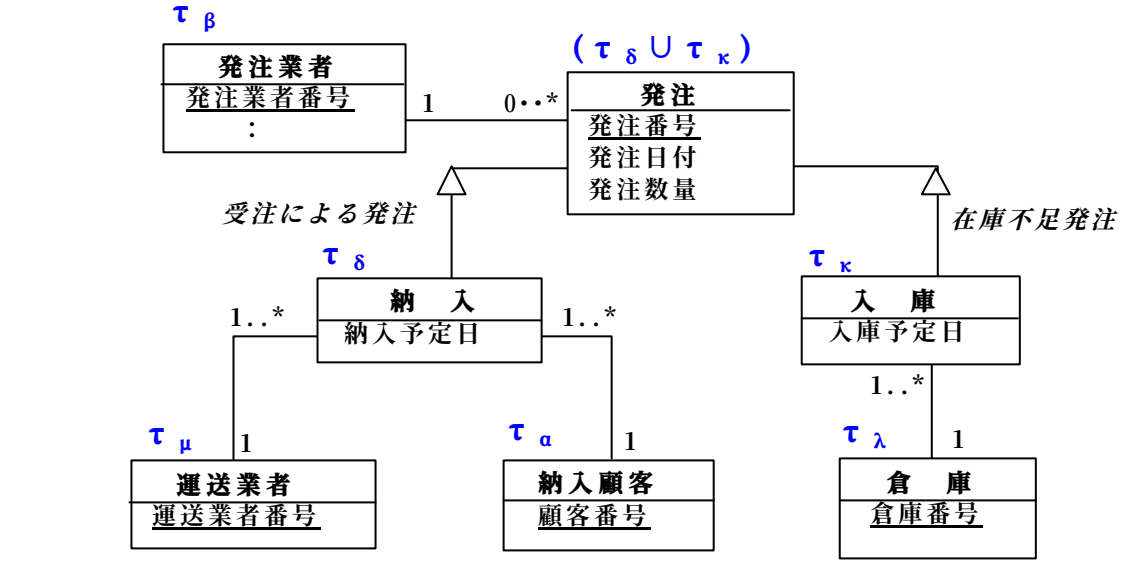
ψ4 ＝{τ δ 発注番号，τ δ 発注日付，τ β発注業者番号，τ δ 発注数量，
τ δ 納入予定日，τ α 納入顧客番号，τ μ 運送業者番号，
τ κ 発注番号，τ κ 発注日付，τ γ 発注業者番号，　τ κ 発注数量，
τ κ 入庫予定日，τ λ 倉庫番号 }

ψ4にまず《 構成演算 0 》を適用し，同じ存在寿命を持つ構成子を束ねて，エンティティ型を構成する．すると，τ δ，τ κの存在寿命を持つエンティティ型の中に共通する識別子を持つものが存在するので，「《 構成演算 3 》汎化」を適用して，共通する構成子を抜き出し，継承の親エンティティ型が持つ構成子とすると，式(3-26)で示すとおりのエンティティ型が抽出される．この結果は，図 3-29 で示した分析者による分析結果と対応している．

ψ4 ⇒ τ δ [発注番号，発注日付，発注数量,納入予定日] +
τ κ [発注番号，発注日付，発注数量，入庫予定日] +
τ β[発注業者番号] ＋ τ α [納入顧客番号]+

$$\tau_{\mu}[\text{運送業者番号}] + \tau_{\lambda}[\text{倉庫番号}]$$
$$=(\ \tau_{\delta}[\text{納入予定日}] + \tau_{\kappa}[\text{入庫予定日}]\ )$$
$$\triangleleft(\tau_{\delta} \cup \tau_{\kappa})[\text{発注番号，発注日付，発注数量}]$$
$$+\tau_{\beta}[\text{発注業者番号}] + \tau_{\alpha}[\text{納入顧客番号}] + \tau_{\mu}[\text{運送業者番号}] + \tau_{\lambda}[\text{倉庫番号}] \quad (3\text{-}26)$$

図 3-29 に含まれる継承構造は明示的に(式 3-26)内で記述されている．しかしながら関連付けに関しては，基本データ集合ψ_1でも述べたとおり，存在寿命の包含関係をもとに関連付け可能なエンティティ型の候補を決めているに過ぎない．



※1 $(\tau_{\delta} \subset \tau_{\alpha}) \wedge (\tau_{\delta} \subset \tau_{\beta}) \wedge (\tau_{\delta} \subset \tau_{\mu})$

※2 $(\tau_{\kappa} \subset \tau_{\lambda}) \wedge (\tau_{\kappa} \subset \tau_{\beta})$

図 3-29 基本データ集合ψ_4, ψ_5の分類に対応する ER モデル

抽出されたエンティティ型間の関連を厳密に決定するには，エンティティ型が持つ属性が他のエンティティ型と相互に関連して利用されるかの利用局面を規定するビジネスルールに基づき，決定される．すなわち，構成演算の適用によって，一意的に ER モデルが抽出されるわけでなく，関連に関しては，「関連の候補」を導出する範囲にあることを留意する必要がある．

4章　クラスライブラリによる検証

3 章での議論は，継承はあらかじめ抽出されたクラスが持つ構成上の制約から，ボトムアップにアプリケーションのクラス構造を構成することを前提としていた．すなわち，分析において，すべてのクラスを存在寿命と共に抽出した後，それらのクラスが持つ構成子の構成上の制約から，階層的な継承構造を順次構成して行く「スクラッチ(Scrach)状態からクラス構造を構成する」ことを前提に，構成演算とその有効性の議論を行ってきた．

本章では，存在寿命の観点から，クラスライブラリが持つクラス階層の発展形式を定性的に説明し，クラスライブラリに派生クラスを追加するとき，そのメソッド数の合計値を定量的に予測できうることを示す．これによって，メソッド数の派生クラスの追加に要する作業工数を事前に見積ることが可能になるが,「工数見積り」に関する議論は本論では割愛する．

4.1　存在寿命とクラスの発展形式の関係

3 章 では，派生する派生クラスの存在寿命の論理和をとったものを親クラスの存在寿命とし，派生クラスから親クラスに向けて，ボトムアップにクラス階層を構成するアプローチにもとづく継承関係の構成方法について述べた．ボトムアップな継承関係の構成では，図 4-1（図 3-6 の再掲）で示すとおり，親クラスの存在寿命は，存在寿命 τ_α を持つ派生クラスと存在寿命 τ_γ を持つ派生クラスの存在寿命の論理和であり，したがって τ_α と τ_γ の合計は，親クラスが持つ存在寿命より大きくなる．すなわち，$(\tau_\alpha \cap \tau_\gamma) \neq \phi$ であり，必ず両クラスに**共通する**イベント時間間隔を持つ．

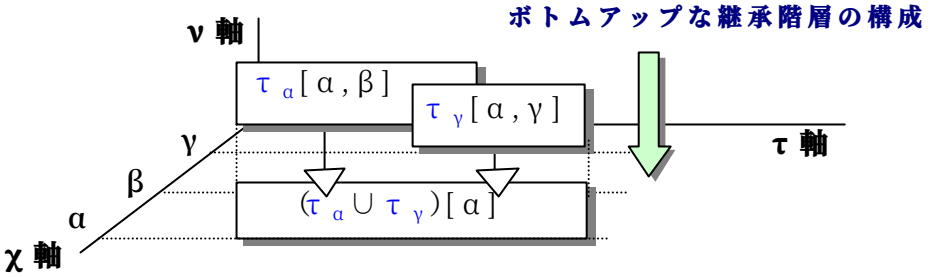


図 4-1　存在寿命から見た抽象メソッドを用いた汎化演算（図 3-6 再掲）

一方，既存クラスを親クラスとして，トップダウンに派生クラスを派生させ，クラス階層を拡張したすると，図 4-2 で示すとおり，ボトムアップな継承関係の構成と同様に親クラスの存在寿命は拡大する．しかしながら，親クラスは派生クラスを派生させなくても存在することから，親クラスの存在寿命 $\tau_0^{(0)}$（ただし，上付き添字は階層レベルを示す），

存在寿命$\tau_{\alpha}^{(1)}$，$\tau_{\beta}^{(1)}$の２つの派生クラスの派生によって（$\tau_{0}^{(0)} \cup \tau_{\alpha}^{(1)} \cup \tau_{\beta}^{(1)}$）に拡大する．クラス階層全体が持つ存在寿命についても，$\tau_{0}^{(0)}$ から，（$\tau_{0}^{(0)} \cup \tau_{\alpha}^{(1)} \cup \tau_{\beta}^{(1)}$）に拡大する．これは，オブジェクト指向機構が持つ基本的な特徴から，派生クラスのインスタンスを生成すると，継承関係にある親クラスの属性やメソッドも同じ存在寿命を持つと共に，親クラス自体も抽象クラスでない限り，独自にインスタンスを生成でき，独自の存在寿命を持つことに対応している．

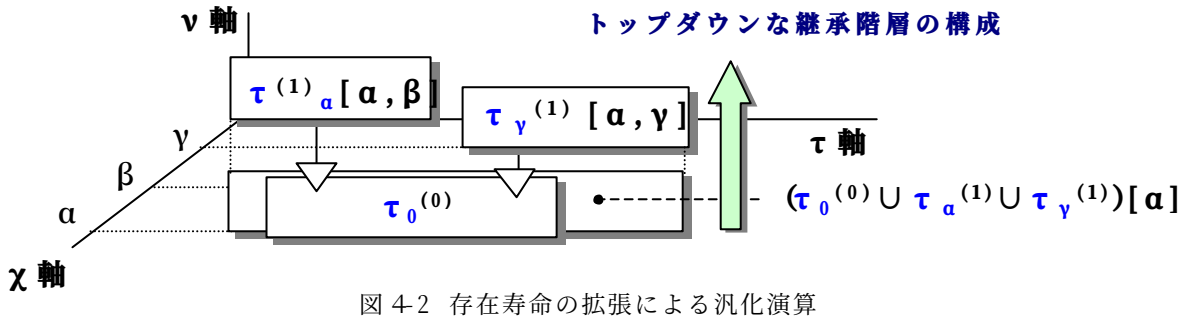


図 4-2 存在寿命の拡張による汎化演算

したがって，存在寿命$\tau_{\alpha}^{(1)}$の派生クラスから，さらに存在寿命が$\tau_{\gamma}^{(2)}$，$\tau_{\delta}^{(2)}$であるクラスを派生させると，存在寿命$\tau_{\alpha}^{(2)}$は，さらに（$\tau_{\alpha}^{(1)} \cup \tau_{\gamma}^{(2)} \cup \tau_{\delta}^{(2)}$）に拡大する．継承階層レベルが増加しても同様である．このような継承レベルの増加によりクラス階層全体の存在寿命も単調増加する過程を図示すると，図 4-3 のとおりになる．

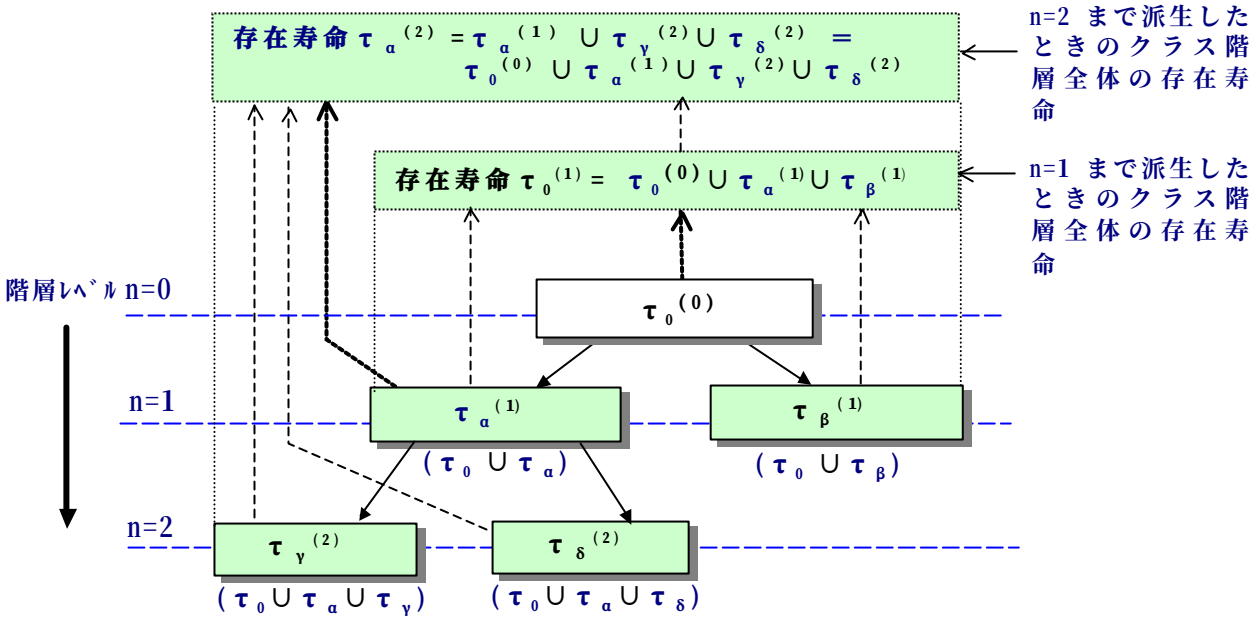


図 4-3 継承の発展に伴う存在寿命の拡大

4.2　存在寿命に基づく派生クラスメソッド総数の予測

現実に提供されているクラスライブラリでは，クラス階層上のいずれかのクラスから派生クラスを派生させることで，トップダウンにクラス階層を発展させることを前提としている．ユーザの立場から見たクラスライブラリの継承階層は，それが形成される過程は不問としてよく（たとえば，派生クラスを追加するとき，ボトムアップに継承関係が形成されるときもあるが，ユーザからはその形成過程は見えない），ユーザがアプリケーションに応じて再利用すべきクラス（＝ターゲットクラス）を見出すとき，継承階層がトップダウンに構成されたもの見なして，役割毎に分類されたクラス系統木を上位階層から下位階層に向けてたどる．したがって，ユーザの視点から見て，クラスライブラリが提供するクラス階層では，前節で述べたトップダウンに継承階層が構成されたときに成り立つ存在寿命の関係が満たされていると写る．ユーザは，この前提に立って，ターゲットクラスから派生クラスを派生させる．しかし現状では，クラス系統木が持つ構成上の性質（たとえば，Model クラスは，継承階層が深いが，View クラスは継承階層が浅い等）を守りつつ，ターゲットクラスからどの程度のメソッド数を持つた派生クラスを派生させるべきかの判断基準は存在しない．

そこでクラスが持つ存在寿命に着目し，かつ次の仮説を仮定することで，派生クラスからクラス継承階層を拡張させるときの派生クラスが持つメソッド総数（＝派生クラスが持つメソッド数の合計）を予測することが可能になる[31].

《仮説》　継承上の任意のクラスαが継承を含めた利用可能なメソッドの数がξ_αであるとき，メソッド数ξ_αは，クラスが持つ存在寿命τ_αに比例する．すなわち，$\xi_\alpha \propto \tau_\alpha$である．

派生クラスが持つメソッド数の予測モデルは,次のとおりに求めることができる.

任意のターゲットクラスから派生クラスを派生させることによるクラス階層全体（＝クラス階層のルートクラス）の存在寿命の増加は式（4-1）で表される．すると仮説から，メソッド数の増加を示す式（4-2）が導かれる．ここで$\Sigma\tau_i^{(1)}$は，階層レベル 1 で派生させた派生クラスの存在寿命の論理和（$\tau_\alpha^{(1)} \cup \tau_\beta^{(1)}$）を示し，$\xi_0^{(0)}$ は階層レベル 0 にある親クラスが持つメソッド数を示す．また，$\xi_0^{(1)}$ は派生クラスをすべて派生させ，クラス階層を拡張したときの親クラスのメソッド数を示し，$\Sigma\xi_i^{(1)}$は，存在寿命に比例したメソッド数の総和を意味する．すると，メソッド数ξ_0を持つ親クラスから派生クラスを追加しクラス階層を発展したとき，クラス階層全体でのメソッド数の増加$\Delta\xi_0^{(1)}$，すなわち（$\xi_0^{(1)} - \xi_0^{(0)}$）は，式（4-3）で表せる．

$$\tau_0^{(1)} = \tau_0^{(0)} \cup (\tau_\alpha^{(1)} \cup \tau_\beta^{(1)}) = \tau_0^{(0)} \cup (\Sigma \tau_i^{(1)}) \quad (4\text{-}1)$$

ただし，i は派生させたクラスに付けた連番

$$\xi_0^{(1)} = \xi_0^{(0)} + \Sigma \xi_i^{(1)} \quad (4\text{-}2)$$

$$\Delta \xi_0^{(1)} = \xi_0^{(1)} - \xi_0^{(0)} = \Sigma \xi_i^{(1)} \quad (4\text{-}3)$$

同様にして，任意の階層レベル(n-1)にある，メソッド数ξ_kを持つ親クラス k から階層レベルnの派生クラスをすべて派生させたとき，存在寿命の増加に対応したメソッド数の増分$\Delta \xi_k^{(n)}$は，親クラス k から派生した派生クラス i が持つメソッド数ξ_{ki}を派生クラス全体で足しあげた数 $\Sigma \xi_{ki}^{(n)}$ として，式（4-4）で表せる．

$$\Delta \xi_k^{(n)} = \xi_k^{(n)} ? \xi_k^{(n-1)} = \Sigma \xi_{ki}^{(n)} \quad (4\text{-}4)$$

さて，親クラスから派生クラスを派生させたときのメソッドは，継承を用いて追加する場合と継承を無効にする（＝メソッドの Final 宣言）場合が考えられる．したがって，派生クラスが持つメソッド数の合計は，親クラスkが持つメソッド数$\xi^{(n-1)}$に比例する成分と親クラスのメソッドを無効にするメソッド数$(1-\xi^{(n-1)})$の相互作用として表されるはずである．すなわち，**メソッド数の増分$\Delta \xi_k^{(n)}$は，階層レベル（n-1）の親クラスが持つメソッド数$\xi_k^{(n-1)}$を用いた式(4-5)のロジスティック関数で表せるはずである．**ここで，式（4-5）中のα,βは，メソッド数$\xi^{(n-1)}$の効果を増大,あるいは縮小する比例定数を意味するものとする．

$$\Delta \xi_k^{(n)} = \alpha \xi_k^{(n-1)} (1 - \beta \xi_k^{(n-1)}) \quad (4\text{-}5)$$

式（4-5）は，式（4-4）を用いると，派生クラスのメソッド数の合計と親クラスのメソッド数の関係式として， 式（4-6）で表せることになる．

$$\Sigma \xi_i(P)^{(n)} = \alpha \xi(P)^{(n-1)} (1 - \beta \xi(P)^{(n-1)}) \quad (4\text{-}6)$$

式（4-6）の意味する内容は，図 4-4 で示すとおりである．ここで，P はターゲットクラス（＝着目する派生クラスの親クラス）にいたるまで，最上位クラスから継承階層を経由してきたクラスのパス P を意味する．

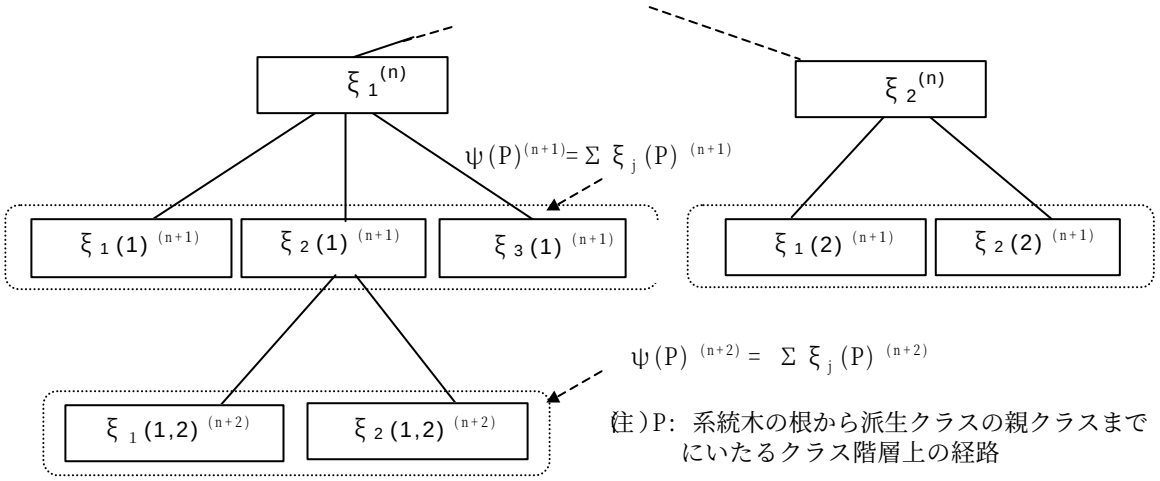


図 4-4 式 (4-6)が持つ意味

以上の議論を踏まえて，クラスライブラリが，式(4-6)の関係を満たしていれば，クラスが持つメソッド数と存在寿命が比例関係にあることが間接的に証明されることになる．さらに式(4-6)の関係から，任意のターゲットクラスから派生クラスを派生させるとき，派生クラス全体が持つであろうメソッド総数，すなわち派生クラスが持つメソッド数の合計を予測することが可能になる．

4.3 Java クラスライブラリに基づく実験的な検証

4.3.1 予測モデル式の層別化

前節で述べた式(4-6)の予測モデル式をクラスライブラリに適用するに当たって，クラスライブラリには，クラスの基本的な役割によって異なるクラス系統木があることに留意しなければならない． Java クラスライブラリを対象にしたとき，標本データとなるクラスの数が大きな系統木は，Component クラス系統木，および ComponentUI クラス系統木しか存在しない．そこで，親クラスが持つメソッド数とその直下の派生クラスが持つメソッド総数の関係を調査するに先立って，ComponentUI クラス系統木を対象にして，親クラスが持つメソッド数とその派生クラスが持つメソッド総数を計測し，散布図として図示すると図 4-5 のとおりになる．

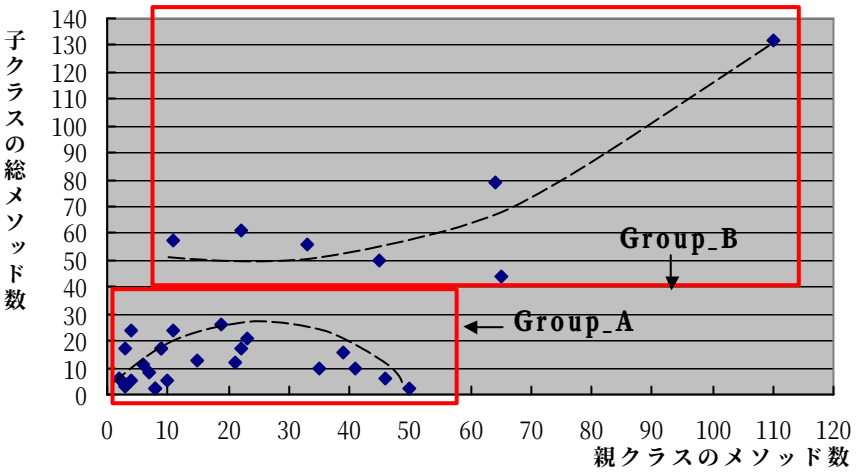


図 4-5　ComponentUI クラス系統木での親クラスのメソッド数と子クラスの総メソッド数のプロット図

図 4-5 から，式（4-6）で予測される関係式が，複数の層で独立して成立しているかのように観測される．発見的なアプローチではあるが，ここで派生クラスが持つメソッド数を合計した総メソッド数が「40」を境界にして，２つのグループ A とグループ B にクラス群を層別化し（層別化の理論的な根拠は，「4.3.4　仮説の考察」で述べる），それぞれのクラスが持つ階層上の平均レベルに統計的な差異があるか否かを t 検定によって検証した．その結果，表 4-1 で示すとおり，2 グループ間で平均的な階層レベル数に有意な差があることが判明した（「2 グループ間の平均的な階層レベル数は等しい」との帰無仮説を 5%の危険率で棄却）．表 4-1 から，グループ A は，クラス階層から見て末端（＝葉）に近いクラスのグループに対応し，グループ B は，クラス階層の最上位，すなわち根（＝ルート）に近いクラスのグループに対応している．

表 4-1　終端クラスからの乖離レベル数に関する平均値の差の t 検定

	グループ A	グループ B
対象クラス数	25	7
終端クラスからの平均乖離レベル数	1.292	2.000
乖離レベル数の分散	0.373	0.285
乖離レベル数の標準偏差	0.624	0.577
t値の絶対値	2.61	(5%有意)

4.3.2 予測モデル式の当てはめとパタメータの決定

計測結果からクラス系統木中での階層レベルを指標にして層別化を行い，まず Java クラスライブラリの複数のクラス系統木に対して，メソッド総数の予測式，および予測式のパ

ラメータを求めた.

(1) Java クラスライブラリ

手順としては，標本データ数が十分な系統木 Component および ComponentUI に対して，式(4-6)の予測モデル式のみならず，複数の予測モデル式（たとえば，指数曲線，高次曲線，対数曲線，ゴンペルツ曲線等）を最小自乗法によって当てはめ，最も適合度の高い予測式（すなわち，ピアソンの相関係数の高い予測式）と予測式のパラメータ値を求めた.その結果，表 4-2 で示すとおりの結果が得られた． グループ B に関しては，式(4-6)で示した予測モデル式より，線形回帰式 $\Sigma\ \xi_{i+1} = a\xi_i + b$ の方が適合度が高かったため，線形回帰式を予測モデル式とした．なお，表 4-2 中の NOC は，標本クラス数（Number of Classes）を示している．また有意水準は，ピアソン相関係数の帰無仮説の棄却水準を示している．

表 4-2 Java クラスライブラリを対象にした予測モデル式のパラメータ

クラス系統木 group 名	NOC	予測 モデル式	パラメータ	ピアソン 相関係数	有意 水準
ComponentUI					
Group A	24	Quadratic	α = 1.534 β = 0.020	0.535	5%
Group B	7	Linear	a = 1.059 b = 8.367	0.758	5%
Component					
Group A	14	Quadratic	α = 1.874 β = 0.028	0.473	5%
Group B	10	Linear	a = 1.305 b = 15.683	0.758	5%

表 4-2 で示した予測モデル式を用いるとき，表 4-1 で示した各グループの末端クラスからに平均乖離率レベル数をもとにして，継承させるサブクラスの階層レベルが末端から 2 階層以内にあるときには Group A の予測式を用い，それ以外のときには Group B の予測式を用いる.

図 4-6 と図 4-7 は， ComponentUI と Component 系統木において実測したメソッド数 ξ を持つ親クラスから派生した派生クラスの総メソッド数(＝実測値)と,表 4-2 で示した Group A のパラメータを予測式(4-6)に代入し,実測したメソッド数 ξ を持つ親クラスから理論的に予測できる派生クラスの総メソッド数（＝予測値）をプロットしたものである.

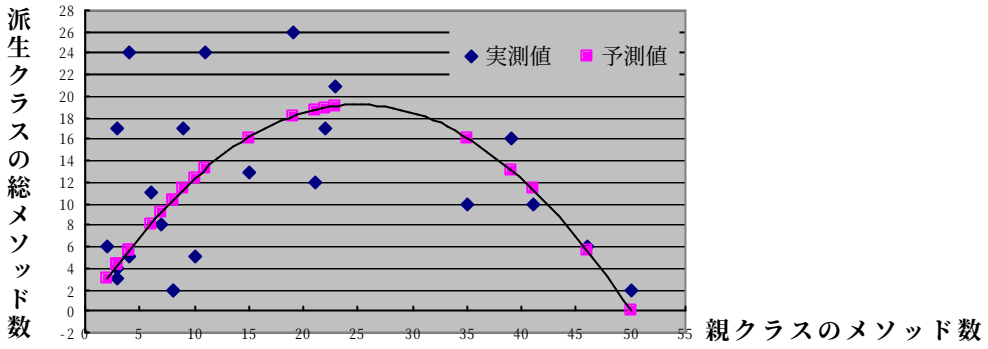


図 4-6 ComponentUI クラス系統木グループ A における派生クラス総メソッド数の実測値と予測値の比較

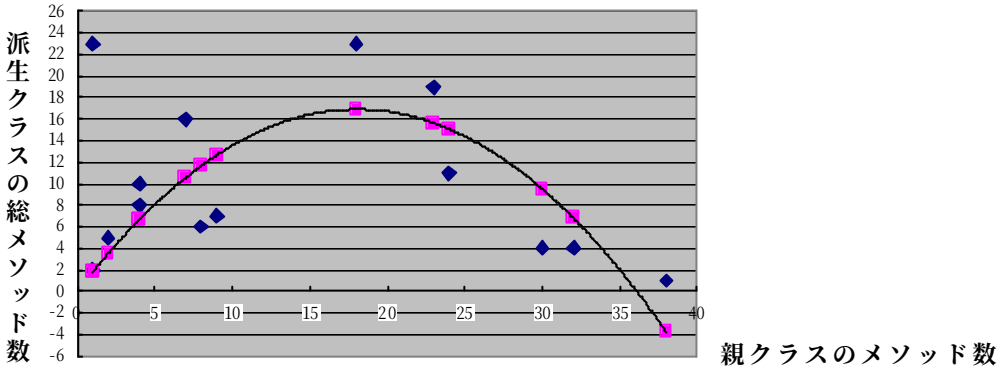


図 4-7 Component クラス系統木グループ A における派生クラス総メソッド数の実測値と予測値の比較

(2) Delphi クラスライブラリ

同様な計測と予測式への当てはめをオブジェクト指向プログラミング言語 Delphi のクラスライブラリ VCL(Visual Component Library) に対しても行った．その結果は，表 4-3 で示すとおりである．Delphi のクラスライブラリにおいても，TPersistant クラス系統木を除いて，Java クラスライブラリと同様に式(4-6)の予測モデル式がきわめてよく適合している．ただし，Java クラスライブラリと異なり層別化後の B グループに関しては，適合度が有意であるような予測式は見出せなかったため，表 4-3 では省略した．

派生クラスの総メソッド総数を層別化する基準に関して，両クラスライブラリとも Component 系統木は 30-40 前後に設定すると，予測モデル式の適合性が高くなる． また Delphi では Java クラスライブラリとは異なり，層別化するメソッド数が 40 と 150 の 2 つの系統木グループに分類された． 層別化の基準となる総メソッド数に関しては，現段階では，計測データから発見的に見出す段階に留まっている．

表 4-3 VCL を対象にした予測モデル式のパラメータ

クラス系統木 group 名	(層別ﾚﾍﾞﾙ) NOC	予測 モデル式	パラメータ	ピアソン 相関係数	有意 水準
Tobject Group A	(40) 12	Quadratic	α= 2.313 β= 0.017	0.858	1%
TPersistant Group A	(150) 6	Quadratic	α= 6.152 β= 0.019	0.741	10%
TComponent Group A	(40) 5	Quadratic	α= 4.215 β= 0.065	0.929	1%
TWinControl Group A	(150) 11	Quadratic	α= 2.921 β= 0.009	0.625	5%

4.3.3 計測結果の考察

以上に述べた計測結果から,クラスライブラリの発展形態に関する次の結論が導かれる.

(1) 継承階層の末端に近い部分は，ロジスティック写像にしたがって，派生クラスのメソッド総数は発展する．しかしながら，クラス階層のルートに近い階層レベルでは，派生クラスのメソッド総数は直線的に発展する.このような差異がメソッド総数の発展形態に生じるのは，階層のルートに近いクラスにおいては，継承を利用してクラス階層を発展させる基本的なメソッドが多くを占めることから，無効にされるメソッドはほとんどなく，それゆえ,親クラスのメソッド数のみに比例して発展する成分のみが派生するメソッドに反映されるためと考えられる.

一方，クラス階層の末端では，系統木ごとにクラスが持つ機能が成熟することから，選択的に無効にされるメソッドが多くなり,親クラスのメソッド数に比例して発展する成分と無効にされる成分が相互に関連した結果,ロジスティック写像で表される派生メソッド総数の発展形態を示すものと考えられる.

図 4-6,図 4-7 から，親クラスのメソッド数が少ない（＝十分な機能を持たない）と派生させるべきメソッド総数も少なく，親クラスのメソッド数が多くても（＝ほぼ機能が網羅されている)，派生させるべきメソッド数は少ないことが読み取れる．これは派生クラスを作成するときの経験にも合致する.

(2) 上述の結論は，クラス階層は存在寿命を拡大させる方向に発展する点と存在寿命とメソッド数が比例するとの仮説に立ったもので，かつクラスライブラリ中の ComponentUI と Component 系統木に限定したものである．議論をこれらのクラス系統木に限定すると，親クラスのメソッド数と派生クラスが持つメソッド総数の関係がよく適合するとの調査結果は,「存在寿命がメソッド数に比例する」との仮説と「クラス系統木が持つクラス階層は,

ルートクラス（＝Object クラス:最上位クラス）の存在寿命を拡大する方向に発展する」とのクラスライブラリの発展形態として当初に想定した関係を間接的に証明したものである．しかし,「存在寿命とソッド数の比例関係」に関する仮説そのものの根拠が不十分であることから，次項ではこの仮説に焦点を当てて考察する．

4.3.4　仮説の考察

クラスが持つメソッドは，クラスが提供する機能の羅列であり，生物の遺伝子配列が持つ遺伝子に類比させて考えることができる．そこで，遺伝子配列がその適応度によって，どのように進化するかをコンピュータ・シミュレーションによって明らかにしたカウフマンの中立進化モデル[37]～[40]との類比から,「存在寿命とクラスが持つメソッド数の比例関係」に関する仮説の根拠を理論的に導き出すことを試みる．さらに，クラス階層のルートでは，派生クラスが持つメソッド総数が線形的に発展し，クラス階層の末端では，ロジスティック写像の関係で発展する根拠を理論的に導き出すことを試みる．

（1）カウフマンの中立進化モデルの概要

カウフマンは，中立進化モデルを提唱した中で，個体発生の源である遺伝子配列の長さと遺伝子交差の結果,保持される正しい遺伝子配列の適応度を適当に設定することによって，遺伝子の進化が自然選択によらずに中立的に起こる可能性を示した．その概要は次のとおりである．

個体からＴ個の遺伝子配列を取り出したとき，その遺伝子配列の結合が正しい機能を維持している個体ｉは，式(4-7)で定義する相対適応度ϕ^iにしたがって生き残るものと仮定する．すると，個体が世代を重ねた後に，正しい機能の遺伝子配列が維持された個体が生き残る割合は，取り出した遺伝子要素の数（結合数と呼ぶ）Ｔとパラメータγ，εの値の変化によって全く異なることをカウフマンは実験的に示した．具体的には，個体が子孫を残せる比率は，式(4-7)で示した相対適応度ϕ^iに比例するものとし，相対適応度ϕ^iは個体全体の平均適応度W_0に対するそれぞれ個体Ｉ固有の適応度W^iの比で表されるものとする．この相対適応度はさらに，Ｔ個の遺伝子配列の中で，世代が代わるごとに生き残る正しい遺伝子配列の数r_iとの比r_i/T（＝残存率）のべき乗で表されるものとする．ただし，残存率のべき乗の寄与率は，適用寄与率εによって制限されるものとする．

$$\phi^i = (W^i / W_0) = (1-\varepsilon)(r_i / T)^{\gamma} + \varepsilon \qquad (4\text{-}7)$$

式(4-7)で定義した個体の相対適応度において，たとえば，パラメータγ=1，ε=0とすると，Ｔ個の遺伝子配列が世代を重ねて正しい遺伝子配列に安定するときの残存率r_i/Tは，Ｔの増加に伴って線形関数的に減少することをカウフマンはコンピュータ・シミュレーション実験によって示した．これは相対適応度に対する適用寄与度εがなく，相対適応度が初期の遺伝子配列を保持した遺伝子配列の残存率r_i / Tに比例して決まることから，

あたかも自然淘汰が働いてるかのように，相対適応度が線形関数的に減少して行くことを示している．

一方，γ=10，ε=0.5としたとき，すなわち，相対適応度φ^i に残存率が適度に寄与し，かつ相対適応度が，初期の遺伝子配列を保持した遺伝子配列のべき乗になるような非線形と仮定したとき,遺伝子配列から取り出した遺伝子要素の数Tを25程度に増加させただけで，初期の遺伝子配列とは全く異なる遺伝子配列に収束することをカウフマンは示した．これは進化がランダムに起こりうること，すなわち，自然淘汰が機能せず，中立的な進化が起こりうることに対応している．

(2) 中立進化モデルとクラスライブラリ発展モデルの対応付け

同様な考え方をクラス系統木毎にクラスライブラリが継承階層を拡張しながら発展する過程に適用する．中立進化モデルとクラスライブラリが持つクラス階層の概念的な対応関係は図4-8で示すとおりである．

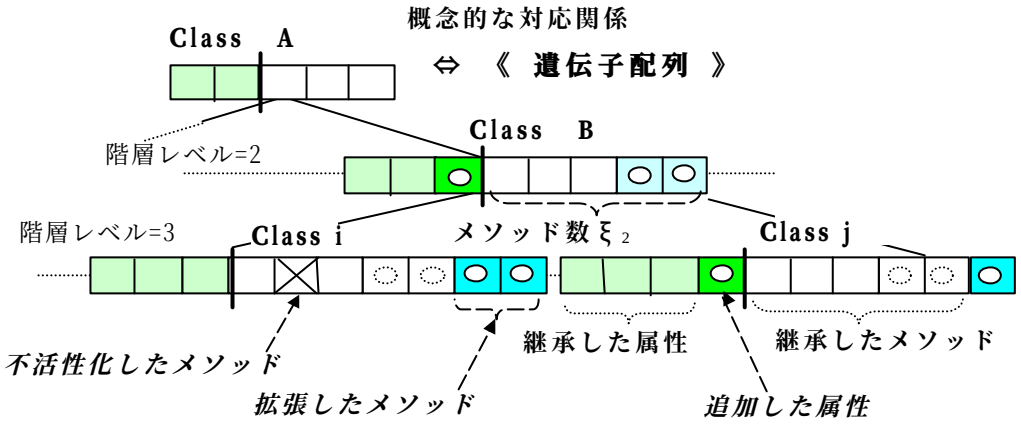


図 4-8 クラス構造と遺伝子進化との対応付け概念

図4-8で示したカウフマンの中立進化モデルで用いられた用語とクラスの発展過程との対応関係をさらに具体的に示すと次のとおりになる．ここで「＝」記号は対応関係を示す．

①個体＝クラスから生成されたインスタンス．

②種＝クラス．

③遺伝子配列＝メソッドの集合．

④適応度にしたがった遺伝子配列の変化過程＝クラス継承構造の発展過程．

⑤遺伝子配列から取り出した遺伝子数 T＝クラス系統木毎に定まっている本質的な機能の複雑さを示す指標（Primary Function Score：以後，基本機能数と呼ぶ）．

⑥世代を重ねて正しい遺伝子配列が保持される残存率 r_i / T＝クラス系統木内のクラス i で変化しないで残った基本機能数の残存率．

⑦相対適応率＝クラス系統木全体の平均適応率を基準にした特定のクラスの相対的な

適応率．値が高いほど安定的なクラスを意味し，発展が少ない．

(3) クラス発展モデルの進化論的解釈

カウフマンの中立進化モデルと類比させ，親クラスと派生クラスのメソッド数の関係を導出するために，存在寿命が直接,メソッド数に比例すると考えるのではなく，その中間的な指標として,クラスが持つ機能的な複雑度を表した抽象的な指標「機能数」を導入する．その上で,「**機能数は存在寿命に比例する**」ものと仮定する．言い換えると，クラスライブラリがクラス階層を発展させ，全体が持つ存在寿命を拡大して行くことは，クラスライブラリが持つ機能数を増大して行くことであると仮定する．

生物の進化論の視点から考えると,「生物体系が持つ機能の複雑度は，生物発生から見た時間的な経過としての存在寿命に比例する（すなわち，機能が複雑な高等生物ほど，生物発生からの時間が経過しており，種としての存在寿命は長い）」とする仮定に相当し，妥当なものであろう．なお，機能数は，ソフトウェア開発コスト見積手法であるファンクションポイント法[41]で導入された概念で，ソフトウェアをソースコードの規模を基準にして計測するのでなく，ソフトウェアが持つ機能の数（＝機能点：Function Point）を基準にして計測するとの考え方にもとづいており，したがって機能数は加算可能である．

以上を仮定すると，あるクラス系統木μに属する階層レベルｎのクラスｉが持つ基本的な機能数は式(4-8)のとおりに定義できる．ここで，Pf^{μ}はクラス系統木μに属するクラスが共通して持つ基本機能数を意味し，δ^{i}_{n}は階層レベルｎにあるクラス ｉで拡張・変更された機能数を意味する．δ^{i}_{n}は，クラス ｉで拡張されたメソッドの機能数η^{i}_{n}と不活性化されたメソッドが持つ機能数κ^{i}_{n}の差を表し，式(4-9)で定義される．$\Sigma\,\delta^{i}_{n}$は，クラス系統木の出発点から階層レベルｎまでの拡張機能数の蓄積を意味する．

$$f^{i}_{n} = Pf^{\mu} + \Sigma\,\delta^{i}_{n} \quad （ただし，\Sigma\,\delta^{i}_{n} はｎに関する合計） \tag{4-8}$$

$$\delta^{i}_{n} = \eta^{i}_{n} - \kappa^{i}_{n} \tag{4-9}$$

一方，クラスが持つ機能数は，クラスの複雑度がメソッド毎の複雑度の合計に比例するとした CK メトリックス（＝クラスの複雑度を定量的に表現するための尺度[43]）の一つ WMC(Weighted Methods per Class)の考え方[42]～[44]をもとにメソッド数と対応付け，式(4-10)で表すことができる．ここで，ξ^{i}_{n}はクラス ｉのメソッド数を意味し，ｈは比例定数を意味する．また継承を重ねても，変更されないクラスの基本機能数の残存率g^{i}_{n}は，カウフマンの相対適応率との類比から式(4-11)で定義する．

$$f^{i}_{n} = h\ \xi^{i}_{n} \tag{4-10}$$

$$g^{i}_{n} = (Pf^{\mu} - \Sigma\,\delta^{i}_{n}) / Pf^{\mu} \tag{4-11}$$

式(4-11)に式（4-8）,(4-9)の関係を用いると，さらに式(4-12)のとおりに変形できる．クラスの相対適応率は，カウフマンの相対適応率との類比から式(4-13)で定義する．

$$g^{i}_{n} = 1 - \Sigma \delta^{i}_{n} / Pf^{\mu} = 2\{1 - (h / 2Pf^{\mu})\xi^{i}_{n}\} \quad (4\text{-}12)$$

$$w^{i}_{n} = w_{0}(1-\varepsilon)(g^{i}_{n})^{\gamma} + \varepsilon w_{0} \quad (4\text{-}13)$$

特定の親クラスから継承を利用して派生クラスを形成させる過程は，親クラスが持つ機能をもとにして,多様な拡張機能を場合分けして追加する過程に相当する.言い換えると，継承関係から見た派生クラスの集合は,親クラスを特化した機能を持つクラスを OR 関係でまとめあげた「特化の OR 関係」にある.それゆえ，親クラスが持つ機能と派生クラスの機能の関係は，一つの親子関係で捉えるべきものでなく，親クラスと派生クラス全体で捉えるべきことになる．その上で，親クラスが持つ機能数 f^{i}_{n} と派生クラス全体がもつ機能数には，式（4-14）の関係が成り立つものとする．すなわち，親クラスの機能数 f^{i}_{n} のうち，適応率 w^{i}_{n} を乗じて残った機能数が，派生クラス全体が持つメソッド総数に等しいものと考える．式（4-14）は，式（4-10）を代入すると,式（4-15）に変形できる．

$$\Sigma \xi^{i}_{n+1} = w^{i}_{n} f^{i}_{n} \quad (4\text{-}14)$$

$$\Sigma \xi^{i}_{n+1} = w^{i}_{n} \nu \xi^{i}_{n} \quad (4\text{-}15)$$

ここで，クラスライブラリが系統木を形成してクラス階層を発展させる形態として，系統木のルートの部分と末端に近い部分との２つの場合を考える．

系統木のルート部分のクラス階層は，系統木の進化が多岐にわたる可能性を秘めてことから，カウフマンの中立進化モデルと類比させ，相対適応率として，残存率の効果が非線形的（＝中立的）で，かつ残存率の効果は 50%程度しか寄与しない場合を考える．すなわち式(4-13)において，γ=10，ε=0.5 とした相対適用率を考える．式（4-13）の g^{i}_{n} はクラス i で不変な機能数の比率であり，クラス i で機能に少しでも変更が加えられると急減に値が減少することから，$w^{i}_{n} = \varepsilon w_{0}$ の定数項の効果のみが残る．その結果，系統木ルート部分の親クラス・メソッド数と派生クラスのメソッド総数の間には，式(4-16)で示す関係が成り立つ．ここで，$a = \varepsilon w_{0} h, b = \varepsilon w_{0} h\omega$ を意味し，ωは非常に小さな値を意味する．

$$\Sigma \xi^{i}_{n+1} = a\xi^{i}_{n} + b \quad (4\text{-}16)$$

他方，系統木の末端部分のクラスでは，クラス系統木を通して継承されてきた機能に依

存してメソッドの拡張・削除が働くはずである．これはカウフマンの中立進化モデルにおいて，選択的な自然淘汰が働く場合に相当し，クラスの相対適応率と基本機能数の残存率との間に線形関係が成り立つ場合に相当する．そこで，式(4-15)に式(4-13)でγ=1，ε=0としたクラスの相対適応率を代入すると，最終的に式(4-17)が導出される．ここでα,βはそれぞれ$\alpha = 2 h w_0$，$\beta = (h / 2pf^{\mu})$を表す定数を意味する．したがって，系統木の末端クラス付近での，親クラスと派生クラスのメソッド総数に式（4-17）で示す関係が成り立つ．

$$\Sigma \xi^{i}_{n+1} = \alpha \xi^{i}_{n} (1 - \beta \xi^{i}_{n}) \qquad (4\text{-}17)$$

式（4-16）,式（4-17）の両式は，前項で述べたクラスライブラリ中のComponentUI，Componet系統木で見出された実験式と合致する．したがって，存在寿命とメソッド数の間に機能数を媒介変数として持ち込み,存在寿命と機能数が比例すると仮定することにより，クラス階層のルートおよび末端部分で親クラスのメソッド数と派生クラスの総メソッド数の関係を表す関係式が導出できることになる．

式（4-16）,式（4-17）の関係式は，実際のクラスライブラリのクラス系統木で計測したデータとよく合致することから，存在寿命と比例関係をもつ機能数の概念を持ち込むことにより，クラスライブラリにおけるメソッド数の発展形態を自然に説明できることが示せた．この結果は，さらに「クラスライブラリにおけるクラス継承階層の発展が，クラス全体の存在寿命を拡大する方向に発展する」との本章で目的とした基本的な仮説を間接的に立証したことになる．

5章　類似研究

5.1　メタパターン研究との比較

本論で述べたクラスの構成要素が持つ「存在寿命」に着目して，クラス構造を抽出する判断基準を取り扱った研究は，目下のところ存在しない．しかし，3 章において構成演算の妥当性を立証するために用いたデザインパターンの研究に絞ると，類似研究として，Pree が提唱しているメタパターンがある[45]～[47]．メタパターンの考え方は，図 5-1 で示すとおり，利用に際して，変更しない Frozen Spot（固定 spot)を含む Template クラスと，頻繁に変更される Hot Spot(変更 Spot)を含む Hook クラスに対し，表 5－1 に示す，構成属性と呼ばれるメタ属性を考えることで，Template クラスと Hook クラスの構成関係を決定しようとする研究である．

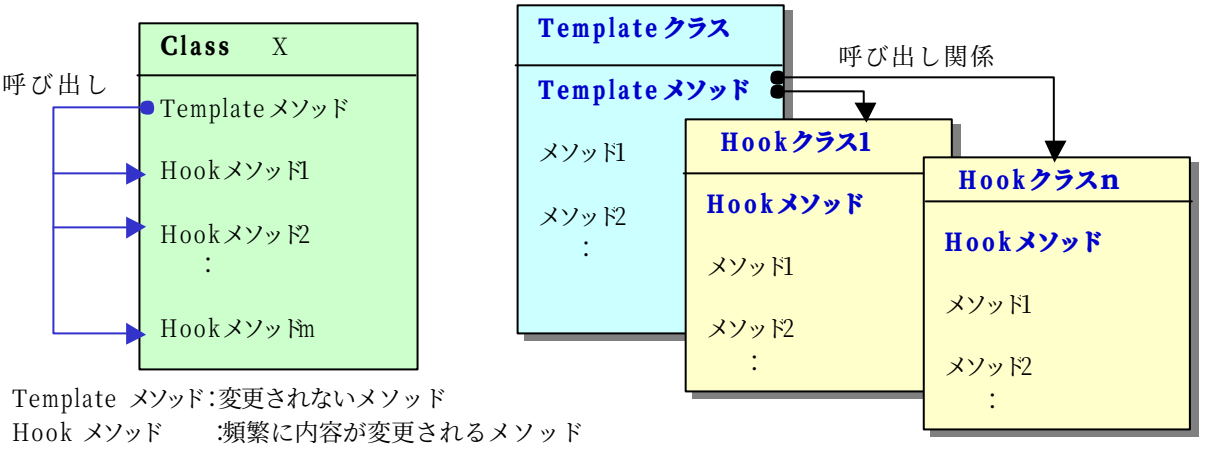


図 5-1　クラス内とクラス間での Template クラスと Hook クラスの関係

表 5-1　メタパターンの構成属性

構成属性 1	Template クラスのオブジェクトは，Hook クラスの 1 つのオブジェクトをどの程度，参照するか?
構成属性 2	Template クラスと Hook クラスの間には継承関係があるか？

メタパターンの考え方では，Template クラスを固定し，Hook メソッドをオーバーライトすることを前提にして，「適応性のあるクラス構造を構成するパターン」を生成する各種のメタパターン（＝パターンをインスタンスとして生成するパターン）を，図 5-2 で示すとおり「経験的」に定めている．クラス構造のパターンは，構成属性から，最も妥当なメタ

パターンを選択し，組み合わせることで構成して行く．

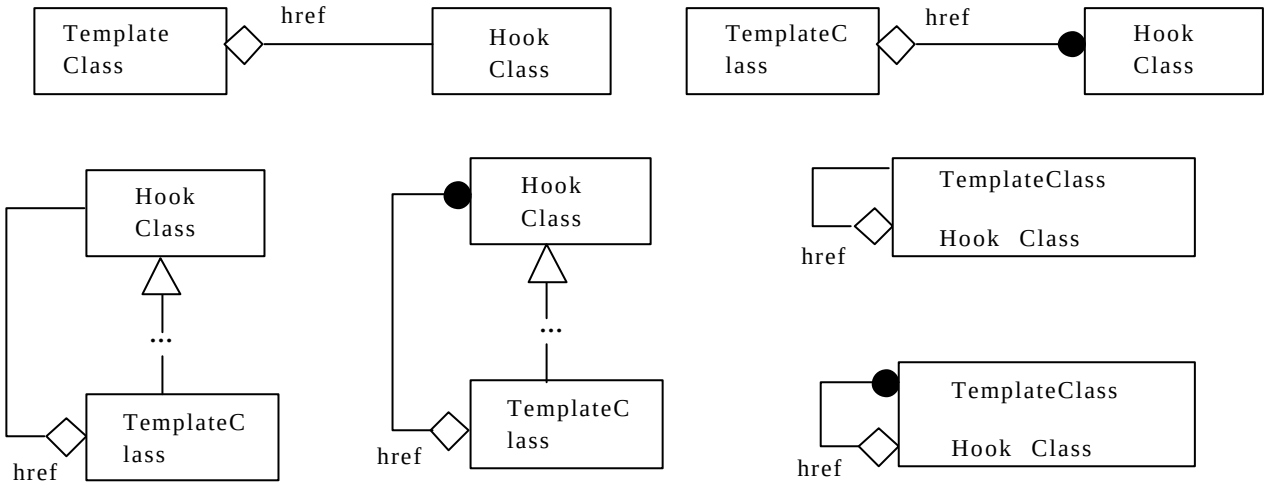


図 5-2 Pree が定めたメタパターンの代表的な種類

これらの構成属性に基づくメタパターンと本研究のアプローチには，それぞれ次の相違点と類似点がある．

《 相違点 》

ⅰ）メタパターンの発想はあくまでもデザインパターンの発想の延長線上にあり，オブジェクト指向が持つ基本的な特徴（たとえば，存在寿命はその一つである）にさかのぼってまで，クラス構造の構成過程を掘り下げることはせず，現象的にパターンを整理し，その基本的な構成部分をパターン化（＝メタパターン化）することに力点を置いている．逆に本研究は，オブジェクト指向の基本的な特徴からクラス構造の構成過程を理論的に導き出すことに力点を置いている．

ⅱ)メタパターンは,あらかじめ Template クラスや Hook クラスが抽出されていることを前提にしている．困難を伴うクラスそのものの抽出方法については触れていない．

ⅲ)メタパターンでは,クラスの持つ意味的な関係のみからクラス構造の構成を試みている．たとえば,「Template メソッドは Hook メソッドを利用するから，Hook メソッドより具象的であると判断する」といった，意味的にあいまいな判断基準をもとに，継承の上位-下位関係の妥当性を判断する．

ⅳ)メタパターンは,具体的な事例としてのインスタンス構造を分析の手掛かりにする発想をもたない．

《 類似点 》

ⅰ）メタパターンも本研究も，構成上の属性をメタ属性として与えて，クラスの構成過程を理論化しようとしている．ただし，メタ属性を与える対象が，Pree の研究ではクラスであるのに対して，本研究は属性やメソッドである．

ii)メタパターンは，Hot Spot や Frozen Spot があらかじめ分析されていることを前提にしている．本研究も同様に，想定されるメソッドの実装数があらかじめ分析されるものとしている．

iii)Pree の研究における Template メソッドと Hook メソッドとの依存関係と本研究における構成子の「独立キー」メタ特性の依存関係は，概念的に等価な関係にある．

5.2 デザインパターン定式化に関する研究との比較

デザインパターンの定式化に限ってみると，Temporal Logic of Actions[48]を基盤に，ザインパターンが持つ振る舞いの定式化を試みた Mikkonen の研究[49]がある．定式化の方法は，一定の役割を持つたオブジェクトを層別化し，その間のコミュニケーションによって引き起こされる振る舞いを論理形式の「アクション」として記述することを基盤にしている．デザインパターン内の各クラスが持つ関連(＝抽象化されたコミュニケーション)から，それらのコミュニケーションによって起こる状態変化を，デザインパターンが持つクラス全体に拡大して定義し，デザインパターンの振る舞いを定式化する試みである．したがって，考察するパターンも，Observer や Mediator パターンといった「振る舞いに関するデザインパターン」が主になっている．

一方，本研究は，存在寿命といった静的な特性からクラス構造の構成過程を定式化するものであり，イベントの発生によって起こる状態の変化は，考察の対象に含めていない．そのため，本研究では，妥当性検証実験で用いたパターンは「構造に関するデザインパターン」の範囲に限定している． しかしながら，Mikkonen の研究も本研究も，インスタンスからボトムアップに個々のクラスの関連を定式化し，デザインパターン全体がもつ関連の究明へと視点を移して行く点で共通点を持つ．

5.3 その他の研究との関連

このほか，振る舞いと実装を委譲（＝従来のサブルーチン呼び出しに類似した機構）によって分離して，デザインパターン間の関連を定める Zimmer の研究[50]もあるが，分析の判断基準に関しては，明確に定式化されたものはない． GoF のデザインパターンが持つ相互関連の分類（たとえば，X は Y を解として用いる，X は Y に類似する，X は Y と結合される）や層別化の方法を用いて，意味的な関連を明らかすることを試みており，デザインパターン全体の構成過程を解明する研究のロードマップ的な役割を果たしている．

6章　結論と今後の展望

6.1　結論

本論では，経験を要するとされる「オブジェクト指向分析におけるクラス構造の抽出」問題を，ソフトウェア場概念とそれに基づく構成演算による判断基準の定式化を中心にして考察してきた．その結果，次の結論を得ることができた．

分析過程のあいまい性をモデル化したソフトウェア場概念の妥当性について

従来，ソフトウェア開発では，システム分析段階で生じる使用のあいまい性を形式的に表現する手段をもたなかった．本論では，ソフトウェアを構成する要素が不確定な状態をソフトウェア場としてモデル化し，分析における判断基準を導出する背景を理論的に定義した．その上で，分析要素（＝構成子）の抽出を場の演算として定義できうることを示した．

存在寿命に着目した構成演算の妥当性検証について

ソフトウェア場上で定義した構成子に対して，存在寿命やオブジェクト指向の基本的な制約等をメタ特性として与え，その演繹としてクラス構造を構成する各種の構成演算を定義した．これらの構成演算を用いることで，従来,「経験」や「ひらめき」に依存していたクラス構造の抽出が,構成子集合に対する構成演算の適用列に置換できることを実証した．さらに具体的な事例として，GoFの「構造に関するデザインパターン」が持つ構成子集合を想定し，それらに構成演算列を機械的に適用することによって，ベテラン分析者が分析した「構造に関するデザインパターン」と同等のクラス構造が導出できうることを実証した．

（3）分析の主作業が，構成子の抽出作業へ移行について

経験やひらめきを要するクラス構造に代わって，構成子は，帳票や伝票上のデータ分析やユースケースシナリオから，分析の初心者でも比較的容易に抽出することができる．そこで，構成演算の適用を前提すると，分析の主眼は，その出発点となる構成子集合をユーザといかに綿密にコミュニケーションしながら，正しく抽出するかへと移行する．すなわち，正しいクラス構造を抽出する問題から「正しい識別名」と「存在寿命」の抽出問題に力点が移行することになり，分析の過度の経験依存から脱却が図れる．

（4）派生クラスがもつべきメソッド総数の予測式の導出について

「存在寿命」とクラスがもつ機能数とが比例するとの仮説のもとに，今まで明らかにされていなかった「クラスライブラリを活用するとき，派生クラスがもつべきメソッド総数の予測式」が導けることを理論的に立証できた．

6.2　今後の展望

ソフトウェア場概念と構成子の存在寿命に着目した今後の本研究の展望として，以下がある．

(1) 構成演算の妥当性検証の継続

デザインパターンの抽出実験に用いた構成子集合は，実際のシステム分析の現場で直面する構成子集合と規模的にも複雑性の点でも異なるが，しかし構造上の特徴は共通している（それゆえ，開発分野特有のアナリシスパターン[51]が意味を持ってくる）．構成演算の理論的な根拠は，オブジェクト指向概念の基本的な特徴に基づくものであることから，今後，さらに大規模なオブジェクト指向分析に適用実験を繰り返すことで，構成演算の実用性の実証事例を拡大して行く予定である．

(2) 要求仕様定義研究との接点

構成演算の妥当性検証の次ステップとして，要求仕様から，構成子の「χ，τ空間上での存在範囲を特定する判断基準」や「構成子のメタ特性を特定する判断基準」が必要になる．「役割」はその一つの基準になりうる（たとえば，役割を手掛りとしたパターンの構成に関する研究[52]がある）．役割はクラスが持つメタ特性の一つとして定義でき，クラス構造全体の変更・拡張に伴って，役割が動的に変化したり，移動，あるいは再編成されたりする．これは「役割」が「構成子集合を配置するときに機能する制約メタオブジェクト（＝独自の存在寿命を持つた分布関数で，クラスに配置される構成子を規定する機能をもったメタオブジェクト)」になりうることを示唆している．

本論で展開した構成子演算によるアプローチと同様にして，仮に要求仕様に含まれる役割分析から，クラス構造を構成する上で制約となる「役割構造（＝役割の重ね合わせ状態，および相互の関連)」が抽出できれば，この役割構造の制約のもとに，構成子集合はχ－τ空間に配置され，クラス構造を構成するとの分析過程モデルを描くことができる．役割を加味したクラス構造の構成モデルは，要求定義手法と密接に関連するものであり，構成演算を前提とした要求定義手法の研究との接点を模索する必要がある．

(3) ソフトウェアの複雑性尺度研究への適用

ソフトウェア場の概念は，視点を変えると，ソフトウェア開発の過程で起こっている数々の現象を「場のダイナミクス」の観点からモデルできる可能性を秘めている．ソフトウェア場概念では，構成子を結び付け，クラスを構成する力（=クラス構成演算）を定義し，構成子自身が持つメタ属性を定義した．これらの概念をさらに発展させると，ソフトウェア開発過程を理論的に解明できる可能性がある．たとえばクラスは，一般に実時間の経過とともに当初のクラスで想定された要求と異なる要求に直面し，改訂の必要性が生じる．それに伴い，①クラスの内部構造を変更，②クラスを分割し新たな派生クラスの追加，③関連する他のクラスのメソッドを修正する等のいずれかの方法で対処する必要性が生じる．

これらの構造の変化は，ソフトウェア場概念では,「クラス内部で定義された結合力によって結合された属性とメソッドの構成子集合に対して，要求の変化の衝撃が加えられると，その大きさに応じて，①クラス内部で属性・メソッドの構成子集合に追加・更新の操作が加えられる，②クラスを分裂して新たな構成子集合から成る派生クラスを形成する，のいずれかがが生じる」ことに対応する．それに伴い，いずれの構造変化を引き起こすべきかの判断基準を理論的に定式化する必要性が生じる．この定式化はクラスが持つ複雑性に強く依存することから,複雑性を定量的に定義した尺度であるCKメトリックス[42]の研究と通じるものであり，定式化が完成すると，現状では単なる定量化の尺度でしかないCKメトリックスに対して，理論的な根拠を与える可能性がある．

このような背景から，本研究を発展させる活動として，以下の【補足】で示すとおり，クラス内部，クラス間で働く各種の結合力を定式化し，考察を加えている．将来的には，要求の変化がもたらす存在寿命の変化によって，いずれのクラス構造の変化を選択すべきかの定量的な関係式を導く研究に繋げて行きたい．

※【補足】

（1）クラス内結合力を用いたCKメトリックスCohesion係数の表現

あるクラスが持つ属性集合を列ベクトルとして表現したものを｜α＞（要素数m),同じクラス内のメソッド集合を列ベクトルとして表現したものを｜μ＞（要素数 n）とする．また属性集合のｉ番目の要素がメソッドｊで使用されているの関係を示す行列$\phi^{\mu}{}_{\alpha}$を，式（6-1）として定義する．ここで，ｆはその関係が存在すれば1，存在しなければ0に変換する関数とする．すると集合論の包含関係を基盤にした Chidamber,Kemerer の定義したCohesion係数ε（クラス内の属性集合を共有してアクセスするメソッド集合の各メソッド間の関連度）は，式（6-2）で定義できる．

$$\text{行列}\quad \phi^{\mu}{}_{\alpha} \equiv f \cdot |\mu\rangle\langle\alpha| \qquad (6\text{-}1)$$

$$\varepsilon \equiv \mathrm{tr} \parallel \phi^{\alpha}{}_{\mu} \parallel / 2 = \mathrm{tr}\,(\phi^{\alpha}{}_{\mu}{}^{T} \times \phi^{\alpha}{}_{\mu}) / 2 \qquad (6\text{-}2)$$

$\phi^{\alpha}{}_{\mu} \parallel$ はｎ×ｎの行列

（2）メソッドのクラス間結合力σ

クラスＸのメソッド集合（要素数ｎ）を表す列ベクトルを｜$\psi^{X}{}_{\mu}$＞，クラスＹのメソッド集合（要素数 n）を表す列ベクトル｜$\psi^{Y}{}_{\mu}$＞，ｆを前述の同様としたとき，２つのクラスのメソッド集合間に働く結合力は，式（6-3)，(6-4）で定義できる．

$$\omega^{XY}{}_{\mu} = f \cdot |\psi^{X}{}_{\mu}\rangle\langle\psi^{Y}{}_{\mu}| \qquad (6\text{-}3)$$

$$\sigma^{XY} = \mathrm{tr} \| \omega^{XY}{}_{\mu} \| / 2 \qquad (6\text{-}4)$$

（3）属性のクラス間関連力λ

２つのクラスの属性集合間に働く力は，式（6-5）として定義できる．ここで，n1,n2 は Class$_1$ ，Class$_2$ の最大生成インスタンス数を示している．

$$\lambda = \text{Class}_1\text{の属性要素数} \times \text{Class}_2\text{の属性要素数} / (n1 \times n2) \qquad (6\text{-}5)$$

謝 辞

本論文をまとめるにあたり，終始懇切なご指導とご鞭撻を賜りました日本大学大学院理工学研究科量子理工学専攻主任　相澤正満教授に心からの感謝を申し上げます．また本論文の機会を与えていただきました，日本大学大学院理工学研究科物理学専攻主任　仲滋文教授に厚く御礼申し上げます．

参考文献

1章

[1] Craig Larman, ”Applying UML and Patterns: an introduction to object-oriented analysis and design,” 邦訳 ”実践 UML” ピアソン・エデュケーション,(1999)

[2] Rational Software Corporation, “UML Notation Guide,” http://www.rational.com

[3] B. Adelson , E. Soloway, ”The Role of Domain Experience in Software Design,” IEEE Trans. on Software Engineering, Vol.11 No. 11, pp. 1351-1360 (1985)

[4] I.Jacobson,G.Booch,J.Rumbaugh, ”The Unified Software Development Process,” Addison-Wesley (1999)

[5] D. Pascot, ”DATARUN CONCEPT” CSA Research Pte.,(1996)

[6] http://ootips.org/mvc-pattern.html

[7] Carroll, J. M. ed., Scenario-Based Design, John Wiley & Sons,1995.

[8]R.Wirfs-Brock, B.Wilkerson, “Object-Oriented Design: A Responsibility-Driven Approach," Proc of OOPSLA ’89, ACM, pp. 71-75(1989).

[9] R.Wirfs-Brock, “Designing Objects and Their Interactions: A Brief Look at Responsibility-Driven Design,”

[10]B.Beck, W.Cunningham, “ A Laboratory for Teaching Object-Oriented Thinking,” Proc. of OOPSLA'89, ACM, pp.1-6 ,1989.

[11]川喜多二郎, ”発想法 ”中公新書

[12] http://db.cs.berkley.edu/topics/cs186/public_html/lecs/chapter14 Conceptual Design Using the Entity-Relationship(ER) Model

[13]S.Shlaer,S.Mellor, “Object-Oriented Systems Analysis- Modeling the World in Data-, ”Prentice-Hall, 1988.

[14] 佐藤正美, “T 字型 ER 手法, ”http://www.asahi-is.co.jp/

[15] 大木幹雄, 秋山構平, “概念モデリングにおける判断基準の提案とその有効性評価, ”電子情報通信学会 論文誌 VOL.J84-D-I No.6 pp.723-735(2001)

2章

[16] 河原哲雄, “概念の構造と処理, ”人工知能学会誌 Vol.16 No.3 pp.435-440 (2001)

[17] 加藤貞行, “ オブジェクト識別についての一考察とその効果, ” 情報処理学会 研究報告, SE Vol. 98, No. 100, 1998.

[18] Ziv's Uncertainty Principle in Software Engineering：Uncertainty is inherent and inevitable in software development processes and products Ziv,H. and D.Richardson,Proc.of 19th ICSE (1996)

3 章

[19]大木幹雄, 稲田晃,“クラスモデリングにおける判断基準,”電子情報通信学会 信学技報 Vol.99,No.312, pp. 9-16,1999.

[20]大木幹雄,秋山構平,“多重継承を含むクラス自動抽出の検証,”電子情報通信学会 信学技報 Vol.100 No.90 pp.25-30,2000.

[21] 大木幹雄,“クラスライブラリ発展モデルへのソフトウェア場概念の応用と評価,”情報処理学会 情処研報 Vol.2002 No.64 SE-138 pp.97-104 (2002)

[22] 大木幹雄,“場の量子化によるクラス構造パターン生成の定式化,”電子情報通信学会 信学技報 KBSE2001-50 Vol.101 No.601 pp.9-16(2002)

[23] 大木幹雄,“クラス構成演算によるデザインパターン導出実験,”情報処理学会 オブジェクト指向シンポジウム'2003 論文集 オブジェクト指向最前線 2003 PP.145-148 近代科学社(2003)

[24] 大木幹雄,“ライフタイム分析に基づくクラス構造抽出の定式化と構造に関するデザインパターンの抽出実験 -Formalization of Class Structure Extraction through Lifetime Analysis and Extraction of Structure Design Pattern-,” 情報処理学会論文誌 45 巻 6 号 (2004)

[25] M.Ohki ,Y.Kambayashi,“A Formalization of the Design Pattern Derivation by Applying Quantum Field Concepts,” Proc.of the Fifth Joint Conference on Knowledge-Based Software Engineering, IOS Press , pp.66-71 (2002)

[26] Kambayashi Yasushi ,Ohki Mikio ,“Extracting the software elements and design patterns from the software field,” Proc. of 5th International Conference on Enterprise Information Systems, pp.603-608 (2003)

[27] Ohki Mikio ,“FORMALIZATION OF CLASS STRUCTURE EXTRACTION THROUGH LIFETIME ANALYSIS,” SWIM/EIC and ACM SIGMIS , 6th International Conference on Enterprise Information Systems,Proc.of the ICEIS2004 PP.635-642 (2004)

[28] P. Coad, “Object-Oriented Patterns,” CACM, Vol. 35 No. 9, pp. 152-159 (1992)

[29] Gamma,Helm,Johnson&Vissides,“Design Patterns: Elements of Reusable Object-Oriented Software ,” Addison-Wesley (1995)

4 章

[30] Kemerer C.F.,Slaughter S., "An Empirical Approach to Studying Software Evolution," IEEE Trans. SE Vol.25, No.4 pp.493-509 (1999)

[31] 大木幹雄,秋山正二郎, "クラス構造の進化モデルと統計的進化パラメータの分析, "情報処理学会 情処研報 Vol.2001 No.92 SE-133-3 pp.15-22(2001)

[32] Yacoub S., Ammar H. , Robimson T., "Dynamic Metrics for Object Oriented Designs, " Proc. Of 6th International Sympo. Software Metrics, PP.50-61(1999)

[33] 中谷多哉子, 玉井哲雄, "継承木進化における統計的特性, " オブジェクト指向'99 シンポジウム論文集, 情報処理学会 pp.137～144 (1999)

[34] Briand L.C.,Wust J., "Modeling Development Effort in Object-Oriented System Using Design Properties, "IEEE Trans. SE Vol.27, No.11 pp.963-986 (2001)

[35] Kitchenham B.A., Hughes R.T, Linkman S.G., "Modeling Software Measurement Data, "IEEE Trans. SE Vol.27, No.9 pp.788-804 (2001)

[36] Briand L.C.,Daly J.W.,Wust J.K., "A Unified Framework for Coupling Measurement in Object-Oriented Systems, "IEEE Trans. SE Vol.25, No.1 pp.91-121 (1999)

[37] 木村資生, "分子進化の中立説, "紀伊国屋書店(1986)

[38] Kauffman S.A. , "Neutral Model in Biology, "pp.56-89vOxford UP (1987)

[39] Kauffman S.A., "At Home in the Universe: The Search for laws of Self-organization and Complexity, " Oxford Unv.Press(1995) (邦訳) "自己組織化と進化の論理 " 日本経済新聞社

[40] 大野乾, "大いなる仮説, "羊土社 (1991)

[41] IPFUG: International Function Point Users Group, http://www.ifpug.org/

[42] Chidamber S.R.,Kamemer C.F., "A Metrics Suite for Object Oriented Design, "IEEE Trans. SE Vol.20, No.6 pp.476-493 (1994)

[43] Chidamber S.R.,Darcy D.P.,Kamemer C.F., "Managerial Use of Metrics for Object Oriented Software: An Explpratory Analysis, "IEEE Trans. SE Vol.20, No.6 pp.476-493 (1994)

[44] Chidamber S.R.,Darcy D.P., Kemerer C.F., "Managerial Use of Metrics for Object-Oriented Software:An Exploratory Analysis, "IEEE Trans. SE Vol.24, No.8 pp.629-639 (1998)

5 章

[45] W.Pree, "MetaPatterns—A Means For Captureing the Essentials of Reusable Object-Oriented Desin, " Proc. Of ECOP pp.150-162 (1994)

[46] W.Pree, "Design Patterns for Object-Oriented Software Development, "

Addison-Wesley (1996)
[47] D.Kling, "Metapatterns-An overview" www.idt.mdh.se/kurser/cd5130/msg/2002lp3/download/CD5130%20VT02%20Metapatterns.pdf
[48] L.Lamport, "The temporal logic of actions," ACM Trans. PL and Systems Vol.16. No.3 pp,872-923 (1994)
[49]T.Mikkonen, "Formalizing Design Patterns," Proc. of 20th ICSE pp.115-124(1998)
[50] W.Zimmer, "Relationship Between Design Patterns," Cop95 pp.345-364 (1995)
[51] M.Fowler, "Analysis Pattern: Reusable Object Models," Addison-Wesley (1995)
[52] D.Riehle, "Describing and Composing Patterns Using Role Diagrams," http://www.ubilab.org/publications/print_versions/pdf/ubilab-woon-96.pdf

付 録

[53]大木幹雄,"ソフトウェア設計の基礎,"日本理工出版会,1998.
[54]大木幹雄,"データベース設計の基礎,"日本理工出版会,1998.

付録 A　オブジェクト指向概念の基本的な特徴

（1）カプセル化（Emcapslation）

カプセル化の概念は，プログラム中で用いられるデータが無制限に操作されると，プログラム中の多くの箇所から，故意，あるいは過失によって予想外の操作が行われ，その結果，原因を把握することが困難な予想外のトラブルが発生することを防止するために考案された概念である．実際，プログラムのデバックにおいて，最も発見が困難なバグは，予想外の箇所でのデータ値の更新である．

「カプセル化」の概念は，図 A-1 で示すとおり，具体的には，特定のデータ集合(以後，変数または属性と呼ぶ)とそれらを操作する手続を関数として（以後，メソッド－Method－と呼ぶ）一箇所に集約して，機構的にメソッドを通してしか変数集合を操作することはできないものとしたものである．集約された変数（以後，内部変数と呼ぶ）集合と内部変数集合を操作するメソッドは，「クラス」と呼ぶ入れ物に格納され，概念的な単位を構成する．

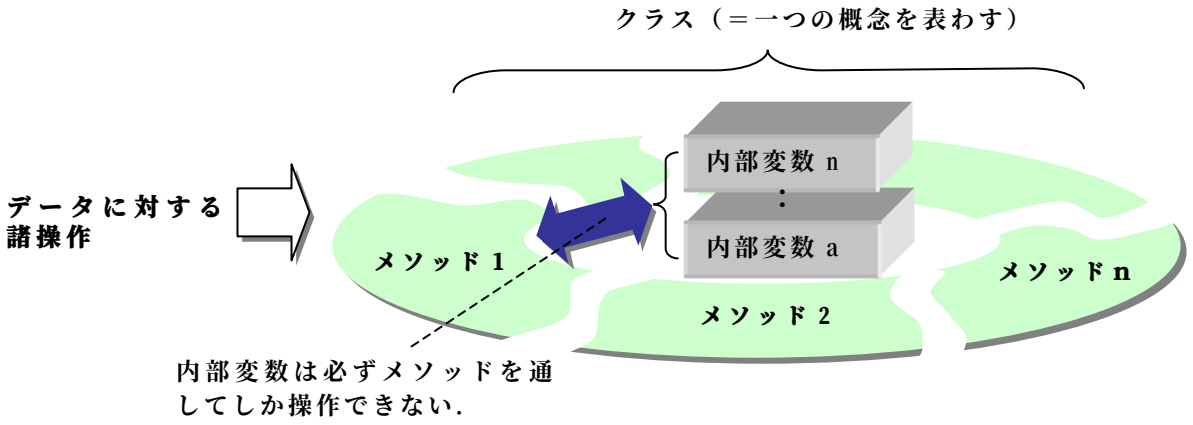


図 A-1　カプセル化の概念

（2）クラス-インスタンス（Class-Instance）

クラス概念の他の原点として，ユーザがコンピュータハードウェアに用意されたデータの格納形式（たとえば，整数，実数，文字列等）や中央演算装置が持つ演算操作（四則演算，文字列の結合，差分等）に制約されることなく，データの格納形式や演算操作を拡張して定義することを可能にした抽象データ型の概念がある．

プログラミング言語の中で変数定義に用いられる整数(Integer)型や実数(Float)型，文字列(String)型等の文字列が，それぞれ異なったデータの格納形式と異なった演算の意味を持つ（たとえば，整数型の＋演算は加算であるが，文字列型では文字列の結合を意味する）のと同様に，ユーザがデータ型の格納形式や演算操作を任意に行えるようにしたもの

である．その結果，それまでのプログラミング言語ごとに異なっていた演算の意味をユーザが明確に定義できことから，演算の意味の取り違いによるプログラミング・ミス（たとえば，FORTRAN では，整数の除算は切り捨てであるが，ALGOL では四捨五入である）を減少させた．

抽象データ型では，抽象データ型名は，整数型,実数型，文字列型と同様，具体的な識別子が与えられた変数に対して如何なる格納形式と操作が定義されているかを示す識別名であり，抽象データ型を明示した変数定義によって，具体的な格納領域を確保する（たとえば,抽象データ型 Complex があったとき,“ Complex a,b,c; ”によって,a,b,c が Complex 型に属することを定義し，同時に a,b,c の格納領域を確保する）．同様にクラスにおいても，抽象データ型に属する個々の抽象的な変数を定義することが可能であり，それはクラスをひな型にして生成された「具体的なもの＝インスタンス」として扱われる．これらの関係を示したものが，図 A-2 である．

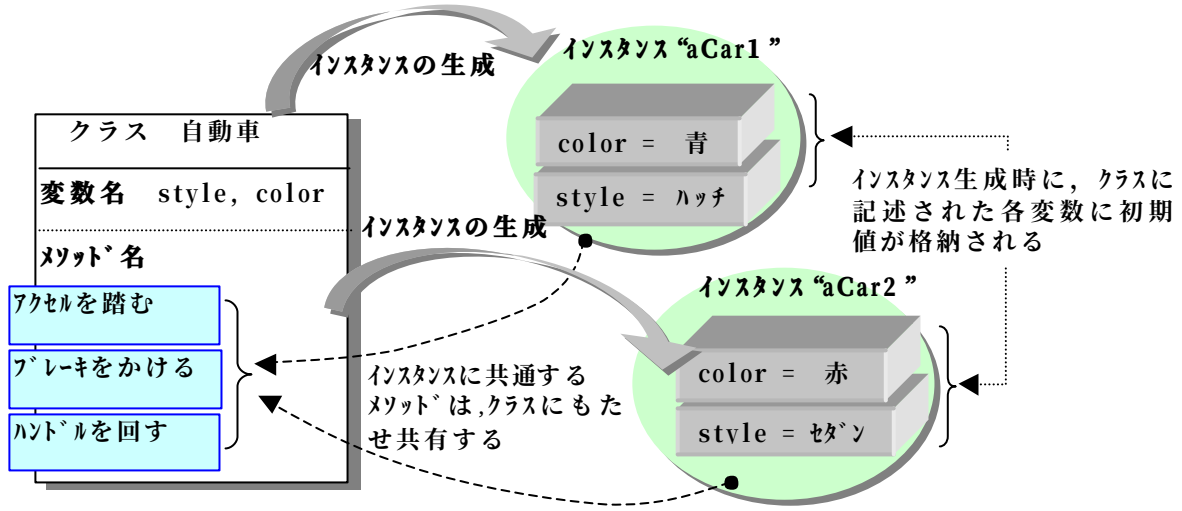


図 A-2 クラスとインスタンスの関係

インスタンスは,クラスをひな型にして生成され,変数に関する具体的な値（=実現値）を持つ．インスタンスが持つメソッドは，すべてのインスタンスに関して共通であることから，クラスに保持し，共有して使用する．すなわち，クラスが持つメソッドは，クラスから生成されたインスタンスが持つ操作手続きを示している．

この考え方にしたがうと，クラスからインスタンスを生成したり，インスタンス生成時に初期値を設定するメソッドは,クラスが持つメソッドとして記述できない.なぜならば,クラスが持つメソッドは,インスタンスが持つ操作手続きであり,インスタンスが生成しない時点で行われる操作手続きは,クラスに記述しえないからである．そこで，これらの操作手続きを同じ発想で体系的に取り扱うために導入された概念がメタクラスである．

メタクラスは，図 A-3 で示すとおり，クラスを唯一のインスタンスとして生成するクラスであり，インスタンスとしてのクラスからさらにインスタンスを生成したり，生成したインスタンスに初期値を設定する操作手続きをメソッド(＝クラスメソッド)として持つ．この考え方を延長すると，さらにメタクラスからクラスを生成する操作手続きを記述するメタメタクラスが概念的に必要になる．ただし，本論で扱う概念はメタクラスまでであることから，メタメタクラスについては割愛する．

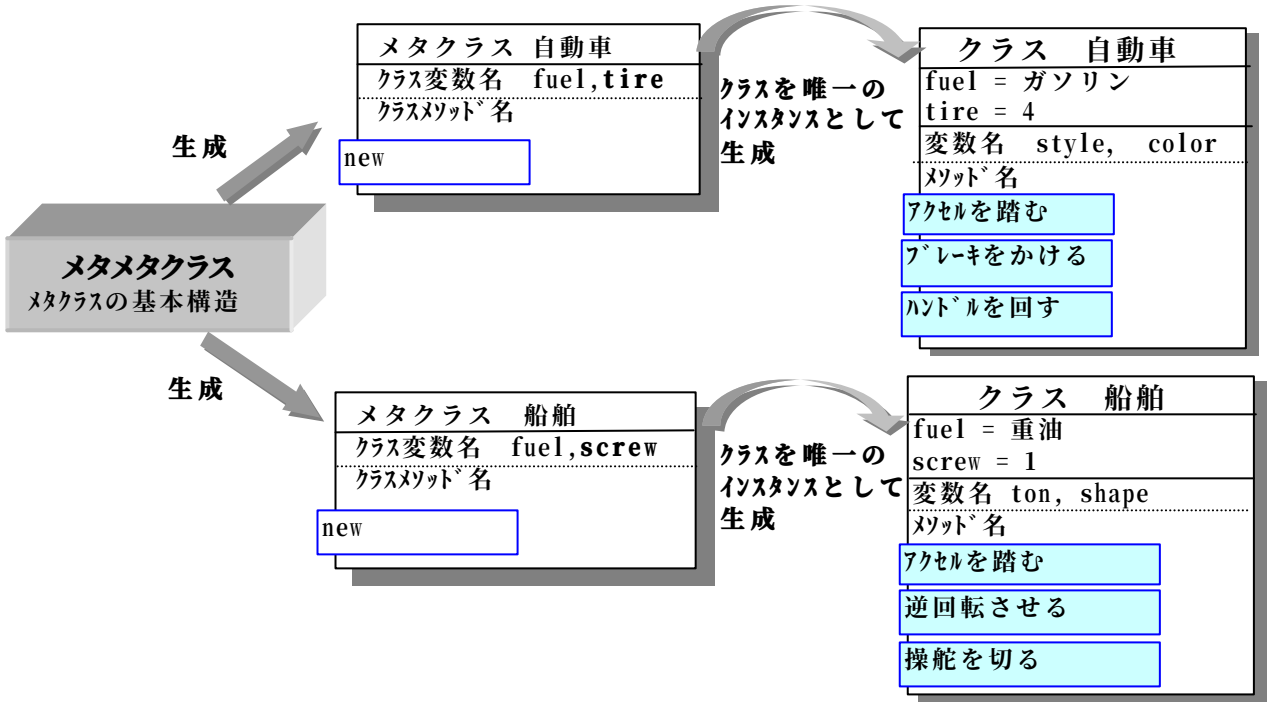


図 A-3 メタクラスとクラスの関係

クラス－インスタンスの関連を実行機構の視点から見ると，クラスは，プログラムコード（命令部が 1 バイトで構成されることから以後，バイトコードと呼ばれる）のプロセスが同時に共有することを可能にしたオペレーティングシステム（OS）のリエントラント（Re-Entrant:再入可能）機構を抽象化したものにほかならない．すなわち，図 A-4 で示すとおり，クラス（たとえば collection クラス）は，変数名リストとメソッド集合のバイトコードを格納した辞書であり，インスタンスは，クラスが持つ変数値と実効制御情報を格納するスタック領域に当たる．クラスからインスタンスを生成するとは，動的にプログラムの実行領域であるスタックを確保することに他ならなず，メソッドは，このスタック領域上で実行される．パーソナルコンピュータでオブジェクト指向プログラミング環境を実現した Smalltalk は，それゆえプログラミング言語としての環境を OS の機能を含む実行環境として提供している．

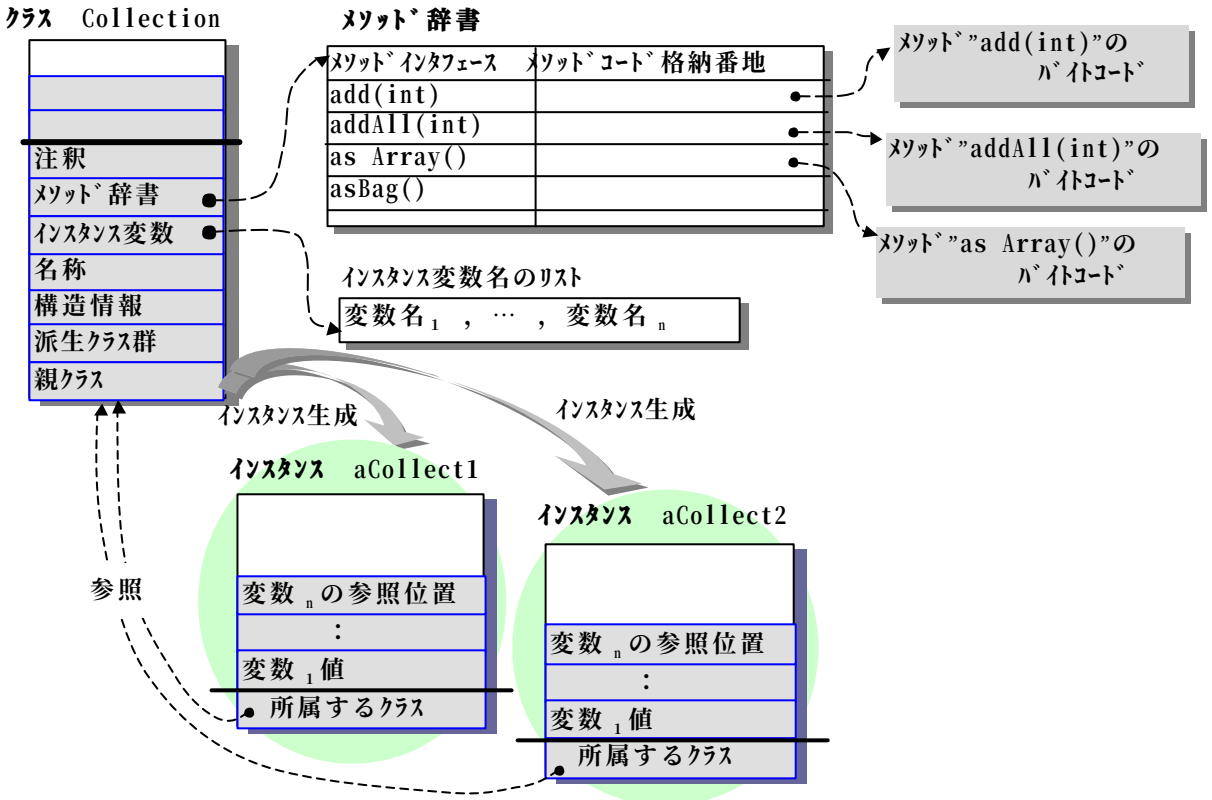


図 A-4 実行機構から見たクラスとインスタンスの関係

※歴史的に見ると，クラスの名称と概念はシミュレーション用言語 SIMULA において初めて用いられたが,SIMULA はシミュレーションの対象とする個々の実体（たとえば，高速道路出入口に設置される料金所の最適数の予測では，料金所および種々のタイプの自動車が実体に相当する）をユーザが定義可能なデータ型に属する変数や配列として捉え，属性や操作の定義をユーザに開放することでシミュレーションモデルの記述を容易にした.

（3）情報隠蔽 (Inforamtion Hidding)

情報隠蔽の概念は，図 A-5 で示すとおり，多くの開発者が並行してクラス開発を行えるよう，クラスが持つ一連のメソッドの定義とメソッドの実現を分離して行えるようにしたものである．すなわち，クラスが持つべきメソッド名とそのパラメータを「メソッド・インタフェース」としてあらかじめ定義しておけば，それらのメソッド・インタフェースを利用した他の開発者の開発作業は，メソッドが持つ機能を実現する開発者と並行して行うことができることになる.

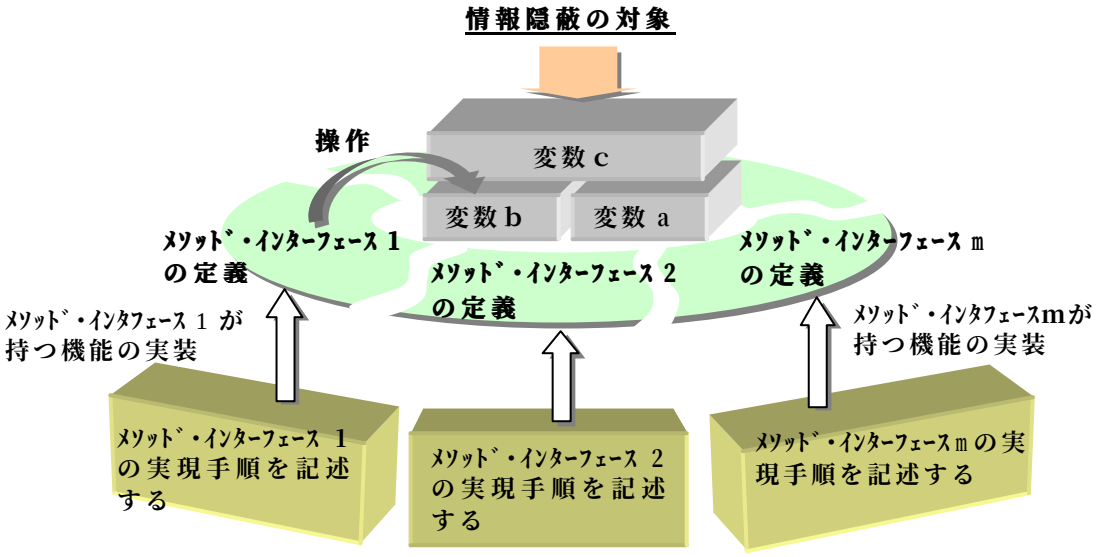


図 A-5 情報隠蔽のイメージ

この情報隠蔽の特徴によって，オブジェクト指向システム開発では，クラスが持つ変数とメソッド・インタフェースの決定作業とそれらの実装作業を明確に分離できることになる．したがって，分析段階でクラスが持つ変数やメソッド・インタフェース，およびクラス構造を正しく決定しておけば，それ以後の設計作業がクラスを中心にして，変数やメソッドを「同一の視点（すなわち，クラス構造は変化させず）」で具体化できることになり，並行作業が行いやすい．現実には，分析設計内容を段階的に具体化するに伴って，クラス構造を修正する必要が生じるため，オブジェクト指向ソフトウェア開発では，繰り返しクラス構造の決定や相互作用，振る舞いの決定が行われることが多い．

分析段階で決定したクラス構造にしたがって，開発作業を具体化できる点は，オブジェクト指向分析設計が，従来の構造化分析設計に比較して有利な点である．実際，構造化分析設計では，分析段階で，システムを構成する処理内容や処理に必要となるデータ集合をデータフロー図によって記述し，設計段階でそれらから機能を抽出し，機能モジュール階層図として再構成する．このとき，「処理の流れ」から「機能モジュール構造」へと視点の変更が生じることから，分析内容と設計内容の間に往々にして不整合が生じるが，オブジェクト指向分析設計では，クラスが持つメソッドとその関連を段階的に具体化するだけであり，不整合が生じることが少ない．

（4）継承 (Inheritance)

継承は，クラスを従来のサブルーチンの集合との差異を際立たせるものであり，図 A-6 で示すとおり，継承の上位-下位関係にある 2 つクラスにおいて，下位にある派生クラスは上位のクラスの属性，メソッドを継承して再利用できる．これは人工知能分野における知識のフレーム表現の考え方を取り入れたものであり，コンピュータ側に人間の持つ概念

体系と同じ知識をあらかじめ保持することが発想の原点になっている．

コンピュータが知識を持つことによって，少ない用語でコンピュータに，人間の思い通りの指示を出すことが可能になる（1980 年，米国 XEROX 社 ALTO プロジェクトの後裔として，子供でも理解できるプログラミング環境の提供を目的とした提供されたオブジェクト指向プログラミング環境 Smalltalk は，この思想に基づいている）．したがって，オブジェクト指向は，あらかじめコンピュータ側が持つ知識であるクラスライブラリを抜きにしては成り立たない．しかしながら，逆に知識体系としてのクラスライブラリの習得に人間側が時間を割かなければならないとの欠点を持つことになる．

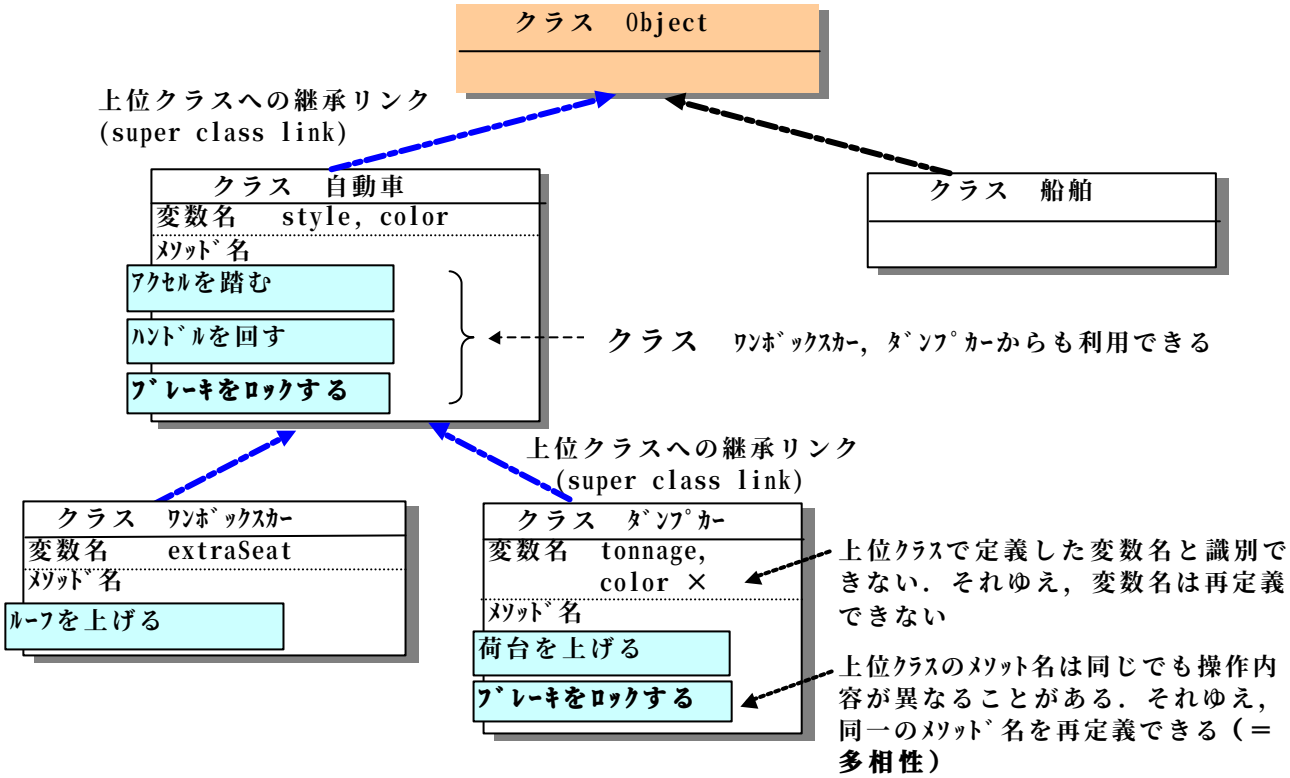


図 A-6　クラスの継承に関する制約

継承の基本的な特徴として，「継承の上位-下位の関係にあるクラス間では，下位にあるクラスは上位にあるクラスが持つ変数名と同一の変数名を持つ変数は定義できない」がある．これはクラスの目的が，変数のカプセル化にあることから，同一名称の変数のカプセル化は，同一のクラスで行うべきとの原則に基づくものである．さらに，あるクラスのインスタンスを生成したとき，継承の上位にあるすべてのクラスの変数をインスタンスは保有することから，同一名の変数が上位クラスに存在すると，インスタンスがそれらを識別できないとの制約に起因している．これらの制約は，たとえば図 A-6 で示すとおりである．図 A-6 では，クラス「自動車」が変数名 style，color を持つとき，自動車の派生概念で

あるクラス「ダンプカー」から生成したインスタンスは，変数 tonnage（積載トン数）に加えて，上位クラス「自動車」から継承した変数 style，color を持つ．同様にメソッドとしては「荷台を上げる」に加えて，上位クラス「自動車」から継承したメソッド「アクセルを踏む」,「ハンドルを回す」,「ブレーキをロックする」を持つ．このとき，クラス「ダンプカー」に変数 color を追加するとなると,ダンプカーのインスタンスが持つ変数 color は,上位クラスで定義された変数 color と識別できない．概念的にも，生成されたインスタンスの独自の特性を示すべき変数 color が上位クラス「自動車」とクラス「ダンプカー」で異なることになる．そのため，上位クラスと同一の変数名は，上位クラスから派生した派生クラスでは定義できない．

（5）多相性 (Pormorphism)

識別子の有効範囲（Scope）概念を持つた伝統的な構造化プログラミング言語（たとえば，ALOGOL,PASCAL 等）では，図 A-7 で示すとおり，階層構造化された有効範囲が異なれば同じ変数名を持つことを許している．

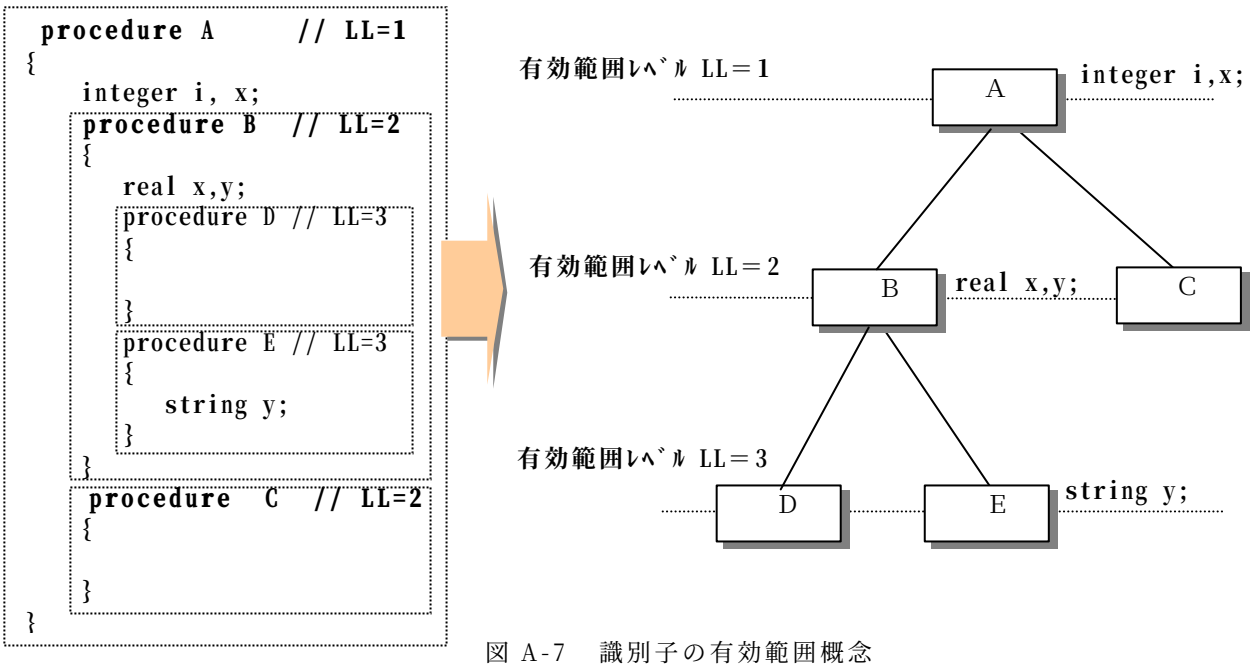


図 A-7　識別子の有効範囲概念

オブジェクト指向では，有効範囲の概念は，識別子の有効範囲の階層構造は継承階層に相当する．しかし「(4) 継承」でも述べたとおり，変数のカプセル化がクラスの基本的な目的であることから,継承階層上に,同一の識別名を持つ変数が存在することを許さない．しかし，メソッドに関しては，このような制約は存在せず，むしろ同一のメソッド名の再定義を許すことで，構造化プログラミング言語が持つ変数名のオーバーローディング機構，すなわち異なる有効範囲で変数名を再定義できる機構と同じ効果を生じさせることがで

きる．そこで，図 A-6 で示したとおり，同一名称のメソッドの継承階層上で再定義することを許すことを多相性(ポルモルフィズム)と呼ぶ．

多相性によって，同一名称のメソッドを用い，異なるクラスで定義された変数にアクセスすることが可能になる．

付録 B　ユースケース（use case）法の概要

ユースケース分析法は，ユーザと分析者が協力して，開発システムに関する要求機能を具体的な利用シナリオ（use case scenario）として網羅的に列挙し，それらをもとにクラス構造を決定するものである．

ユースケース分析の目的は，外部からシステムに働き掛けるアクタと呼ぶシステム外部の動作主体を洗い出すと同時に，アクタがシステムと対話するシナリオを洗い出し整理することで，システムにもたせるべき機能仕様を抽出し，整理分類することにある．このような作業は，図 B-1 のような図を描くことで行う．

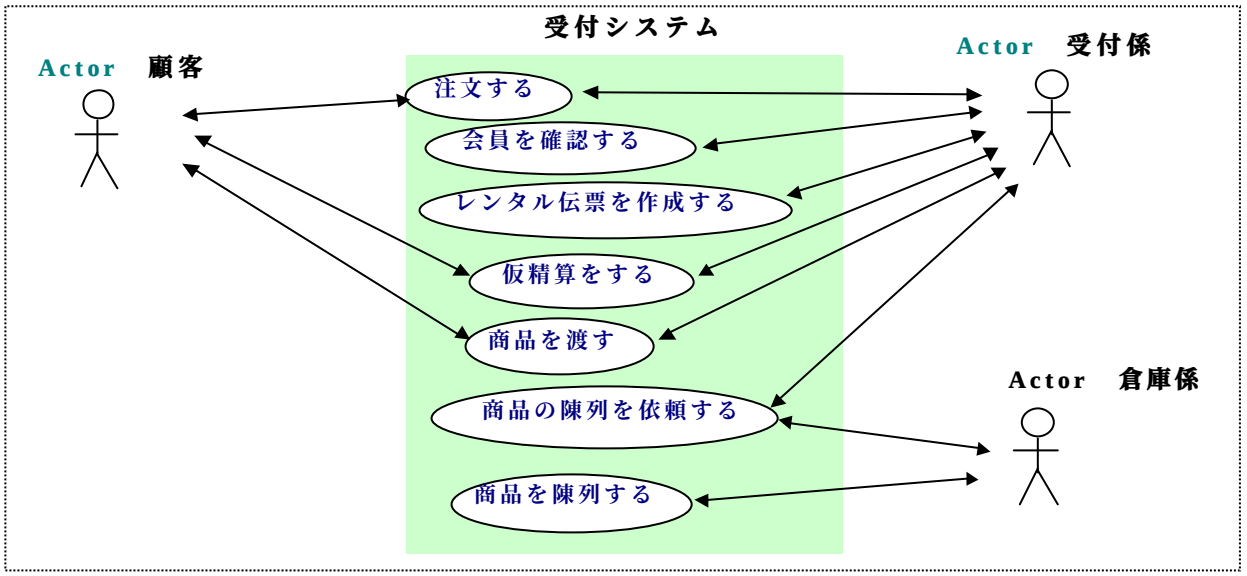


図 B-1　ユースケース図の記述例

アクタは必ずしもユーザを意味するものでなく，システムに働き掛け，一連のシナリオに沿って対話をはじめるようなオブジェクトを意味する．

ユースケースは,外部から見たシステムの振る舞いをシナリオとして抽出することが目的であることから，図 B-1 のような形式を用いて書き下すような方法がとられる．具体的には,図 B-2 で示すような記述形式を用いてシナリオを作成する．

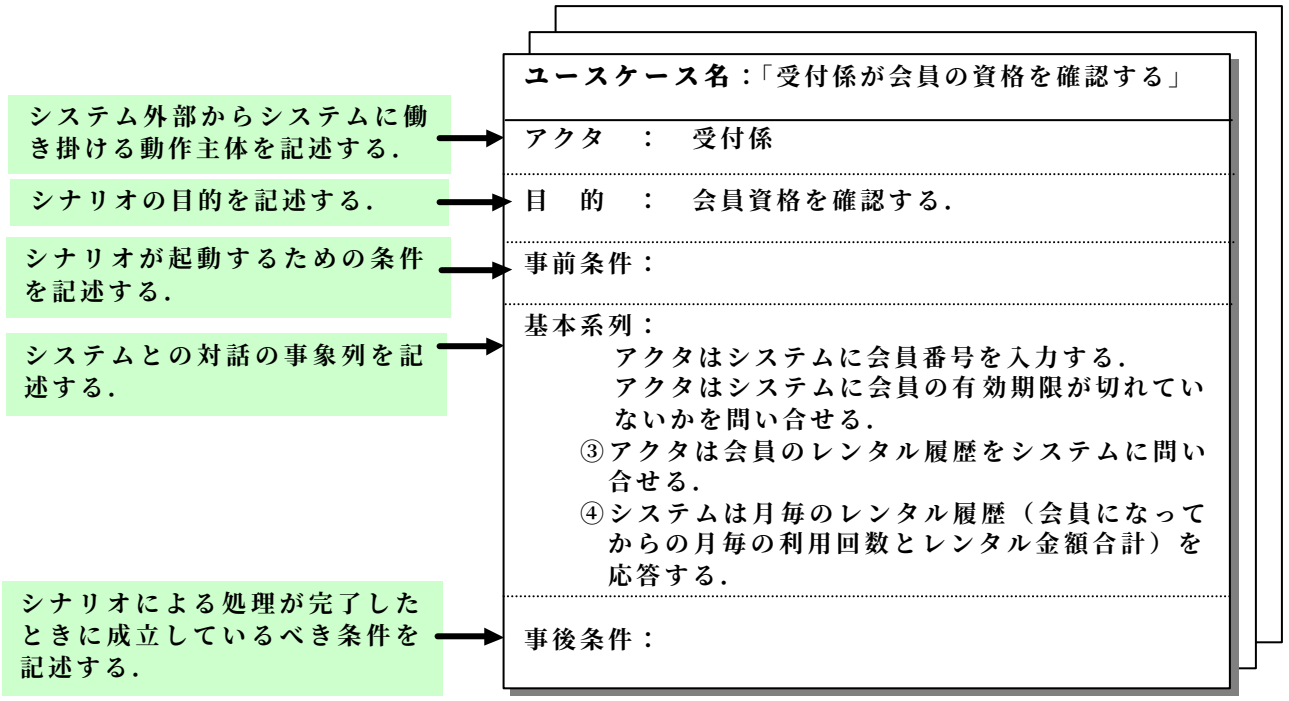


図 B-2　ユースケースのシナリオ記述例

付録 C　責任駆動分析法の概要

責任起動分析手法は，図 C-1 で示すとおり，CRC（Class Responsibility Collaborations）カードと呼ばれるカードを用いて，クラスが提供する機能をサービス責任としてカード上に列挙すると共に，サービス責任に関連するクラスをカードに記述し，それらのカードを分類し，クラスが持つべきサービス責任を定めて行く方法である．カードを用いた発想法（たとえば，川喜多二郎 Kawakita Jiro によって考案された KJ 法）と同様に，分析者が擬人化されたサービス責任者としてのクラスの立場に立って，受け持つべき責任や関連する他のクラスとの関係を発見的に洗い出し，それらを分類して，サービス責任のまとまりとしてのクラスを抽出して行く手法である．たとえば，図 C-2 は，銀行の ATM（自動現金支払い機）業務に必要なクラスを分析した事例を示している．

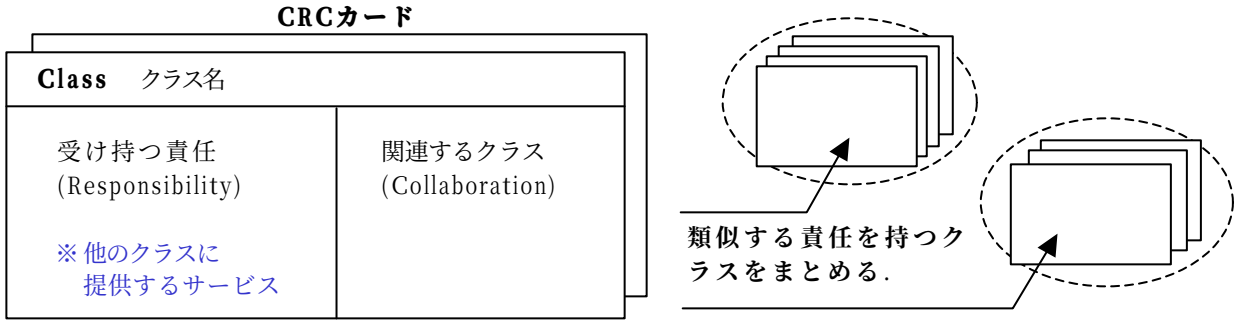


図 C-1 CRC カードの記述規則

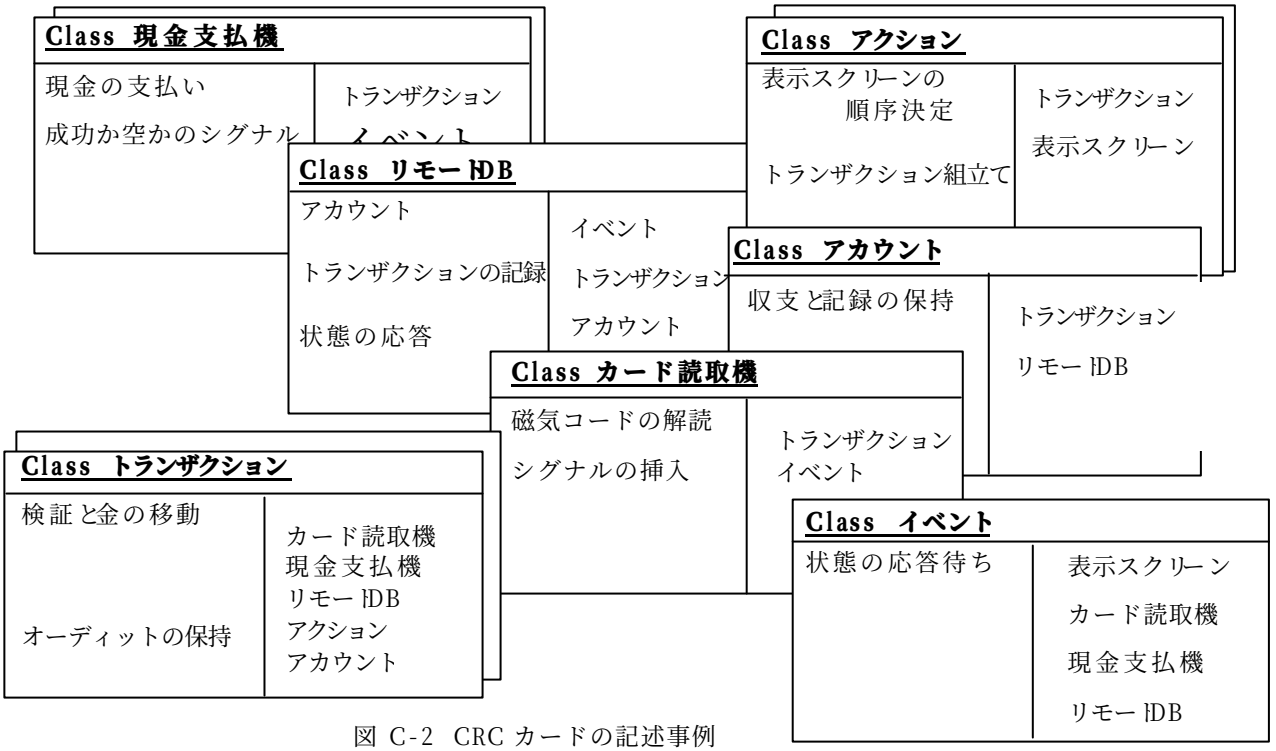


図 C-2　CRC カードの記述事例

付録 D　ER(Entity Relationship)モデリングの記述規則

ER モデリングで記述する基本的な内容,および記述規則の概要は,以下のとおりである.

(1) Entity(実体)

顧客や製品のように何らかの意味的にまとまりのある属性を持つた独立した個別の「もの」を指す．たとえば，図 D-1 では一人の顧客の実体を表している．実体が持つ属性は，実体を特徴付ける「固有の特性」であり，たとえば,他の実体との関連に関するような属性は含めてはならない．また，属性が持つ値は必ず一つ（単値）であり，同時に複数の値（多値）を持つようなことはない.

図 D-1　一つの実体の表現例

(2) Relationship（関連）

実体と実体との結びつきを表す．たとえば，図 D-2 は顧客「山田」が製品「Vioretto」を購入したとき，一つの「購入」と呼ぶ関連で結ばれることを表している.

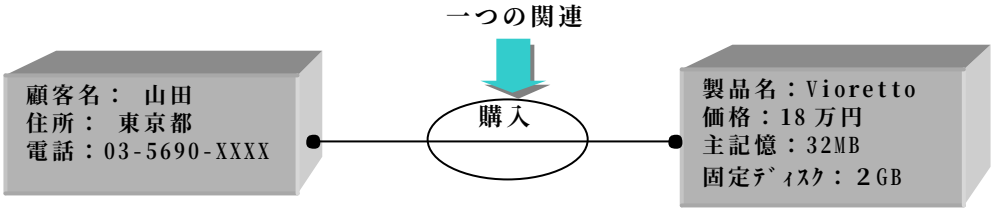


図 D-2　一つの関連の表現例

(3) Entity Type（実体型）

実体の集合体を表現するもので，図 D-3 で示すとおりの長方形で表現する．実体の集合体は,一つの概念を表したもので,長方形の箱の上部にその名称を記述すると同時に,個々の実体が共通して持つ種々の属性 (attribute)を箱の内部に記述する．属性のうち，一意的に個々の実体を識別しうる属性を「キー属性」といい，属性名の下にアンダーラインをつけて明示する.

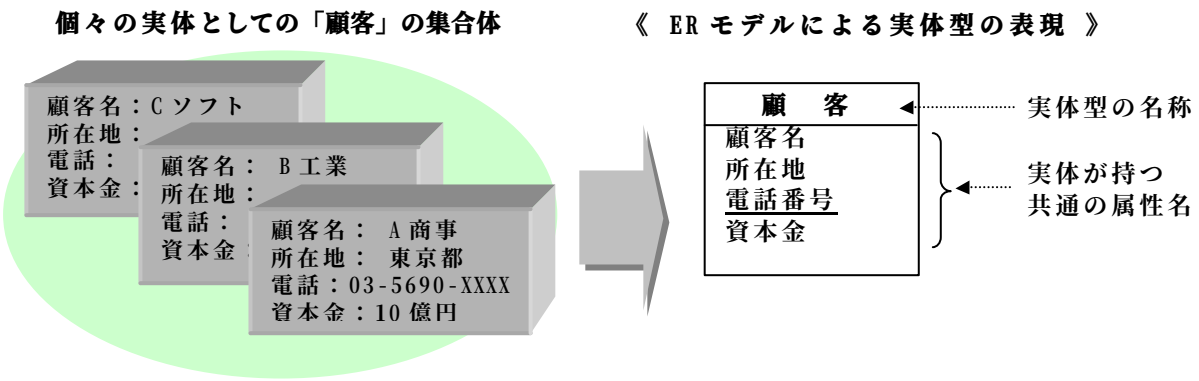


図 D-3 一つの実体型の表現例

(4) Relationship Type (関連型)

結び付けられた実体間の関連の集合体を表したもので,図 D-4 で示すとおり，ひし形，または楕円で表現し，その内部に関連の名称を記述する.

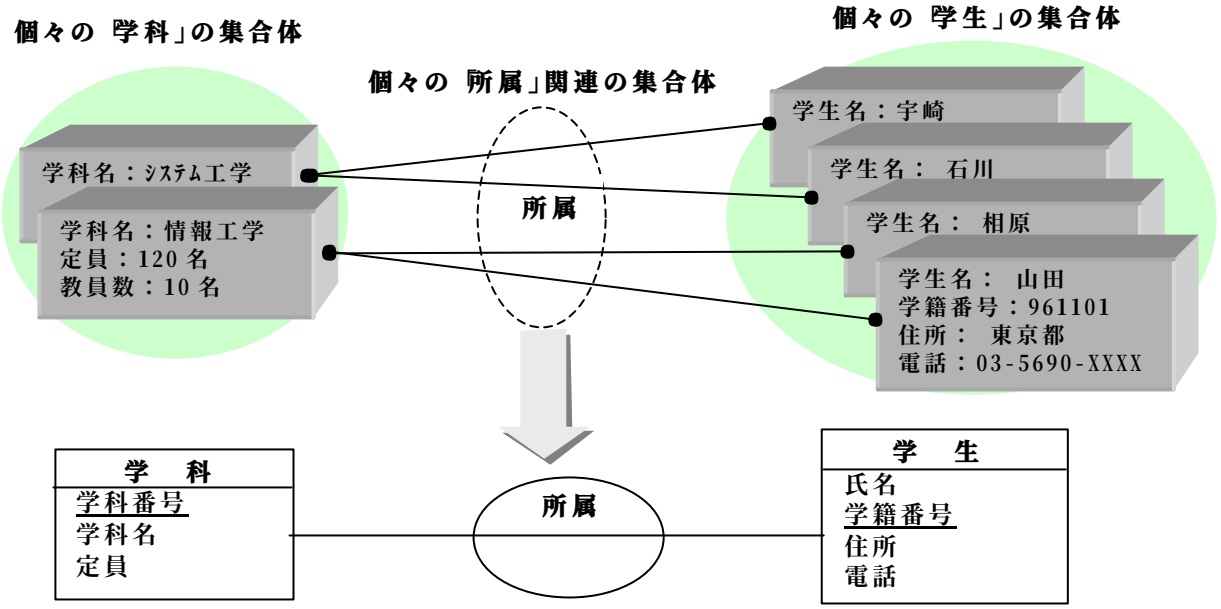


図 D-4 一つの関連型の表現例

(5) 関連の基数，あるいは結合度 (カーディナリティ：Cardinality)

実体間を関連で結んだとき,それぞれ関連の両端に結びつく実体の数がどの程度になるか記述しておくもので,「関連の基数」と呼ぶ. 次の 3 つ区分が存在する.

- 1 対 1 （1 : 1） 関連
- 1 対多 （1 : N ） 関連
- 多対多 （N : M ） 関連

ここで 1 ,N,M の数字は，実体間の関連度を示す「基数」と呼ばれるものである. N や M

は不特定多数の数を意味するもので，必要に応じて具体的な値を設定する．具体的な関連の記述例として以下がある．

① 1対1関連

関連の基数は，一方の実体から見て関連付けされる他方の実体の数を，他方側の実体の側に記述するものである． たとえば図D-5で示すとおりの運転手と免許証の関連は，一人の運転手から見て，どの程度の数の免許証が関連付くかを免許証の側に記述する．同様に，一つの免許証は，どの程度の数の運転手と関連付くかを運転手の側に記述する．したがって，関連「所有」によって結び付けられる実体の数は,それぞれ1人の運転手に対して免許証が1であるので1対1となる．これを1対1の関連と呼ぶ．

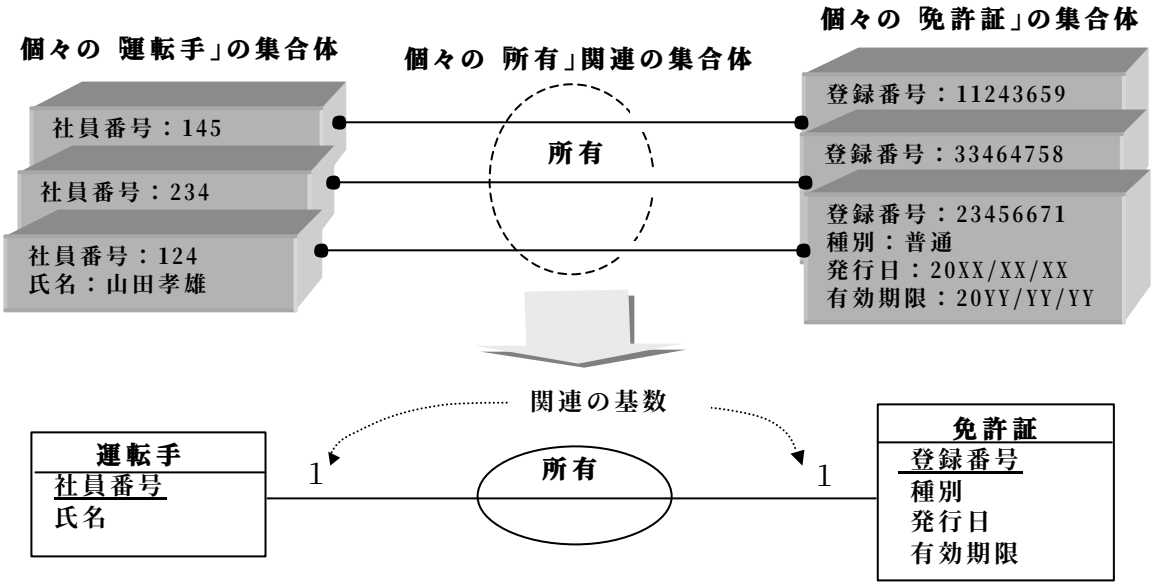


図 D-5 1対1の関連型の表現例

② 1対多関連

1対多の例には，たとえば図D-6に示すような1つの学科から見て,学科に所属する学生の数は不特定多数である．逆に一人の学生から見て，必ず一つの学科に所属し，同時に複数の学科に所属することはない．したがって，このような関連「所属」は1対多（1：N）の関連になる．ここで，実体型「学生」の側に記述した基数「1，N」は，一つの学科から見て，最小1人，最大N人の不特定多数の学生が「所属」関連で結び付けられることを示している．最低，最高の結合数を，それぞれ「最小結合度」,「最大結合度」と呼ぶ．

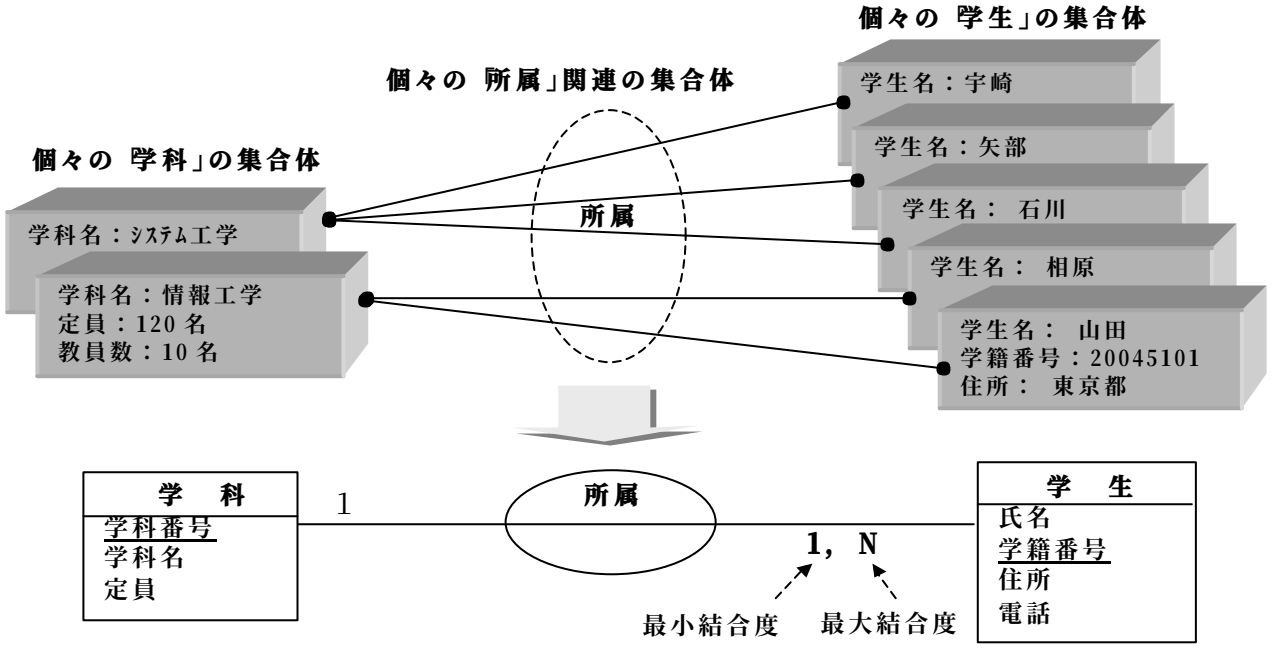


図 D-6　1 対多の関連型の表現例

③ 多対多関連

多対多の例には，たとえば図 D-7 で示すとおり，学生の科目の履修関連がある．個々の実体「学生」は不特定多数の科目を履修する．逆に個々の実体「科目」は，不特定多数の学生によって履修される．したがって「学生」と「科目」を結ぶ「履修」と呼ぶ関連は，不特定多数の学生と不特定多数の科目を結び付ける．関連で結ばれる両端の実体の数はそれぞれ N,M となるので，多対多の関連である．

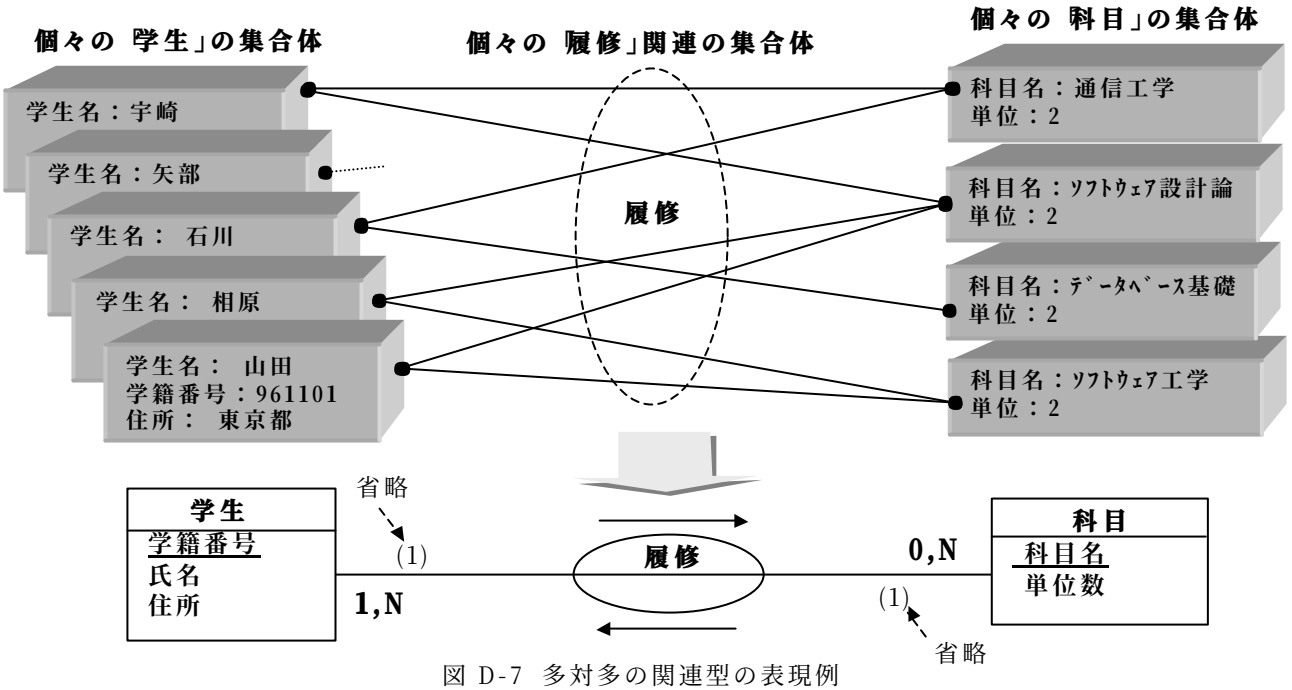


図 D-7　多対多の関連型の表現例

このほかに，ER モデルを作成する上で注意すべき点として以下がある．

※1 関連が持つ属性

図 D-7 で示した多対多関連は，特定の年度内で学生が科目を履修したときの関連を示している．このとき，学生が科目を再履修した年度まで含めて関連を把握するときには，再履修を示すために履修年度を ER 図に追加する必要が生じる．しかし「学生」や「科目」は履修年度に関係なく存在する実体型であり，したがって履修年度の属性はいずれの実体型の属性ではない．履修年度は，関連付けが行われたときに明らかになることから，関連型「履修」そのものにもたせる．図 D-8 で示すとおり，関連型が持つ属性は，関連を示すひし形,あるいは楕円内の下部に記述する．

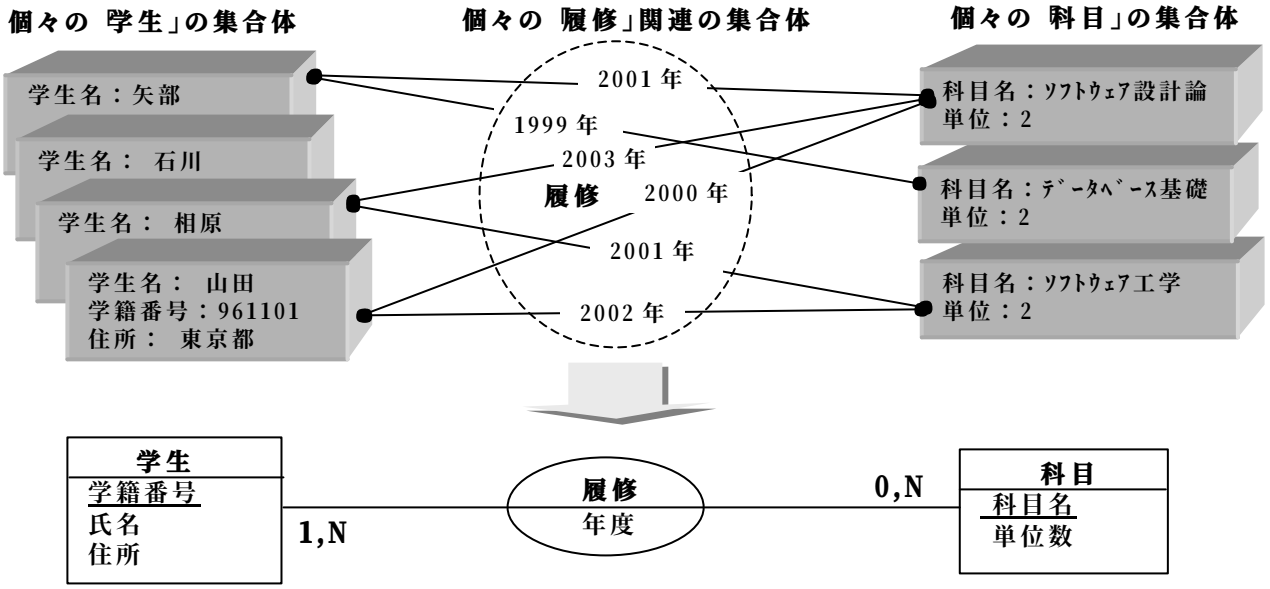


図 D-8　多対多の関連型の表現例

※2 識別子従属性（存在依存性）

1 対多の関連として，部署と所属する複数の担当者の関連，注文と注文明細，顧客と顧客への複数の請求の関連などがある． 1 対多の関連の中には，一方の実体型が存在しないと他方の実体型が意味をもたないような関連型が存在する．たとえば，図 D-9 で示す実体型「請求」と実体型「明細」の関連の基数は 1 対多であるが，実体型「明細」は，単独に存在しえず，実体型「請求」の存在に依存する．なぜなら，明細は請求が存在したとき，はじめて，その内訳として存在するからである．今まで述べてきた実体型は，独立して，かつ永続的に存在するものを表しており,関連型はその間の結び付きを表していた.「明細」のように「請求」が存在しなければ意味をもたない実体型と「請求」との関連は，通常の関連型とは意味が異なる． そこで，図 D-9 で示すとおり， 1 対多の関連であっても，基数 1 に相当する実体型が存在してはじめて基数 N 側の実体型が意味を持つ関連であることを

表現するため，N の下にアンダーラインをつけて記述する．このとき，実体型「明細」は実体型「請求」に識別子従属するという．

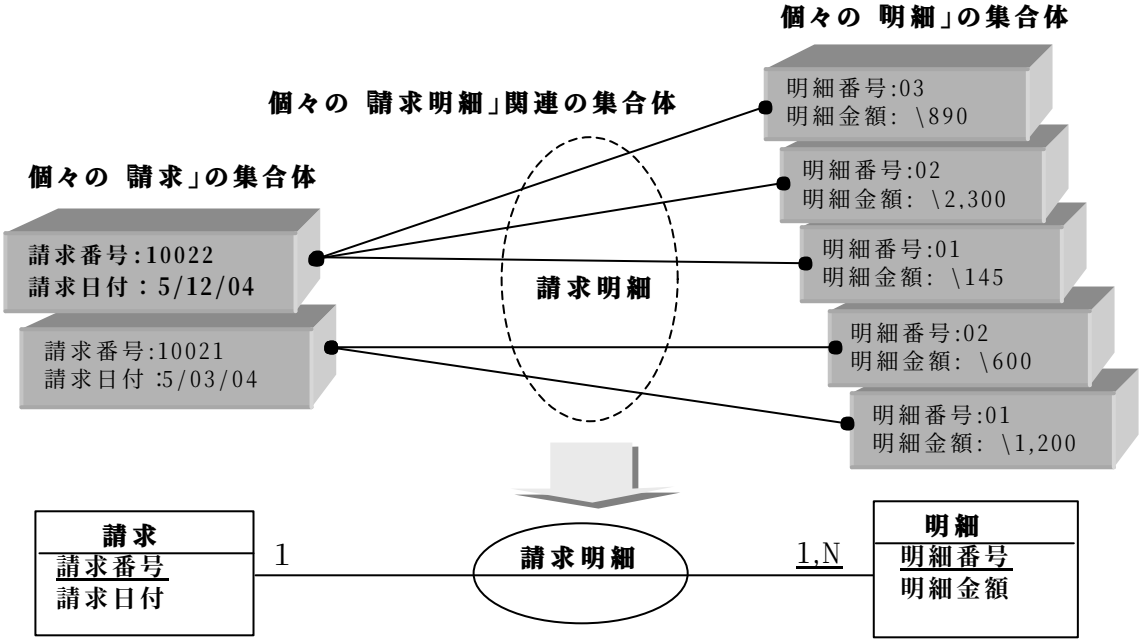
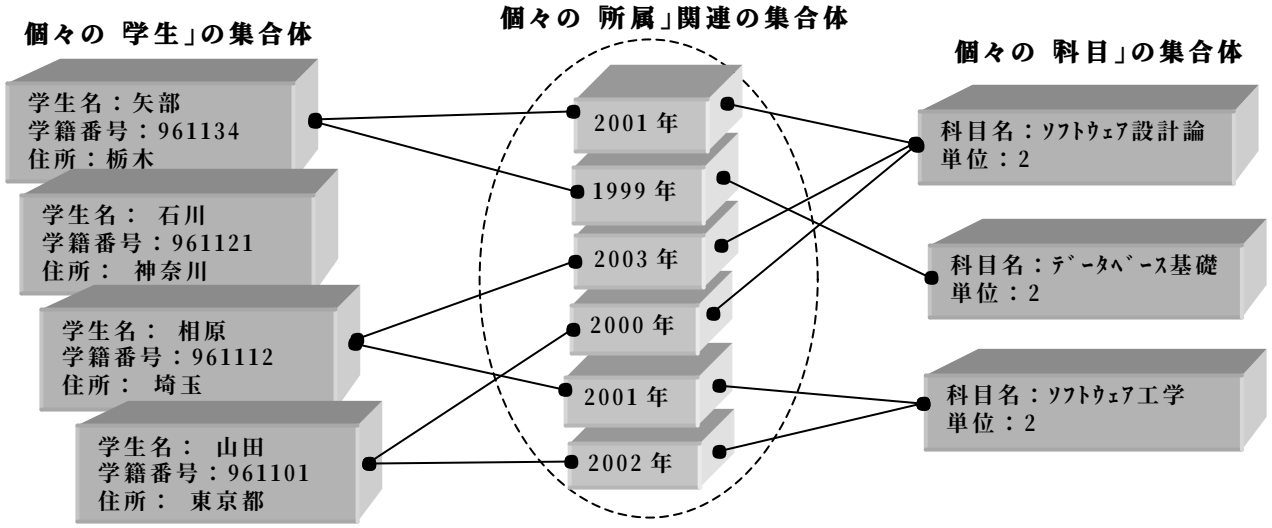


図 D-9　識別子従属を持つ関連の記述例

※3 関連実体

多対多の関連は，実体と実体の対応関連を一つの実体とした“関連実体”を導入することで，二つの 1 対多関連に置き換えることができる．たとえば，図 D-9 の多対多関連は，図 D-10 に示すような二つの 1 対多関連に置き換えることができる．



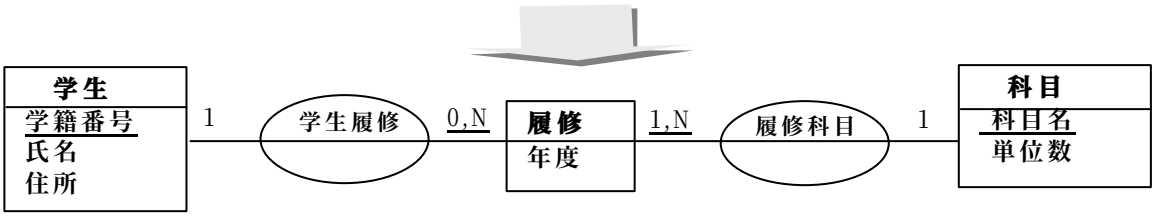


図 D-10　関連実体を導入して多対多関連を置換した記述例

※4　関連と基数の表現形

ERモデリングにおける関連と関連の基数に関する表記方法は,現在まで数多くのものが提唱されている．代表的なものとして，図 D-11 で示す表記法がある．しかし本論では，オブジェクト指向分析設計への連続性と現状における標準化動向から，図 D-12a,b で示すとおり,関連の基数の表現形は UML の標準表記法を用いることとする．

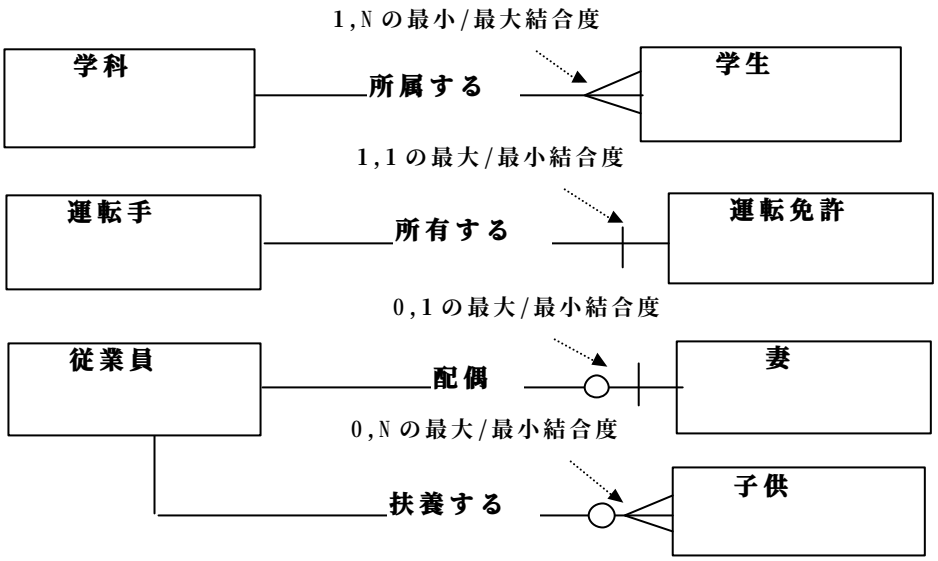


図 D-11　　関連と基数のいろいろな図式的な表現

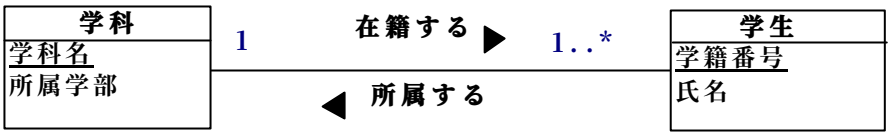


図 D-12a　UML による関連と結合度の表現

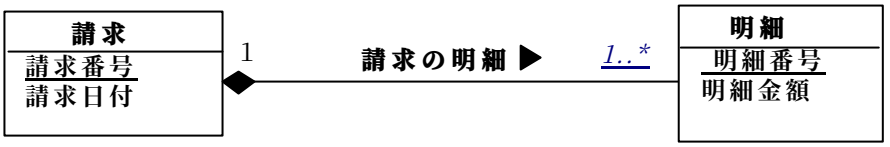


図 D-12b　識別子従属を持つ関連の記述例

なお, UML による表記法では関連の方向を記述する記号“◀ ”,“ ▶ ”を導入することで, 関連の方向性を明確に示せるが,本論では特に必要のない限り,関連の方向性は省略する. 継承関係は, 関連を示す線上に白抜きの小三角形△を記すことで示す.

また, 識別子従属の関連は, 図 D-12b で示すとおり, 黒抜きの小さなひし形を関連の終端に付して示すが,さらに前述の ER 図で示した識別子従属関連を意味する下線付き結合度と併用するものとする. ER 図の場合には, UML の表記法にとらわれず, ひし形, あるいは楕円を関連の記述に用いるものとする.

付録 E　GoF が提唱した標準デザインパターン

GoF(Gang of Four:E.Gamma,R.Helem, R.Johnson,J.Vlissides)によって提唱された，デザインパターンは表 E-1 のとおりである．

表 E-1　E.Gamma,R.Helem, R.Johnson,J.Vlissides らにより提唱されたデザインパターン

利用時点	デザインパターン名	独立に変更できる設計要素
生成方法	Abstract Factory	部品オブジェクトの集合
〃	Builder	複合オブジェクトの生成方法
〃	Factory Method	インスタンス化されるサブクラス
〃	Prototype	インスタンス化されるクラス
〃	Sigleton	クラスの唯一のインスタンス
構造決定	Adapter	オブジェクトへのインタフェース
〃	Bridge	オブジェクトの実装
〃	Composite	オブジェクトの構造と構成
〃	Decorator	サブクラス化を伴わないオブジェクトの責任
〃	Facade	サブシステムのインタフェース
〃	Flyweight	オブジェクトの格納コスト
〃	Proxy	オブジェクトへのアクセス方法
振舞い決定	Chain of Responsibility	要求を満たすことができるオブジェクト
〃	Command	要求を満たすタイミングと方法
〃	Interpreter	集約オブジェクト要素のアクセス方法,走査方法
〃	Iterator	言語の構文規則と解析方法
〃	Mediater	オブジェクト間での相互作用の様子
〃	Memento	オブジェクトの外部に保存されている私的な情報及び保存のタイミング
〃	Observer	特定のオブジェクトに依存するオブジェクト群の変更方法

〃	State	オブジェクトの状態
〃	Starategy	アリゴリズム
〃	Template Method	アルゴリズムのステップ
〃	Visitor	複数のオブジェクトに適用されない操作

出典："Design Patterns Element of Resusable Object-Oriented Software" Addison-Wesley 1995

これらのデザインパターンのうち，生成方法に関する代表的なデザインパターン Abstract Factory と構造決定に関する６つのデザインパターン，および振る舞い決定に関する代表的なデザインパターン Observer の概要を以下に示す．

(1) 生成方法に関する代表的デザインパターン

《 Abstract Factory デザインパターン 》

● 目　的：

相互に関連したり，依存し合うオブジェクト群を，その具体的にクラスを特定せずに生成するためのインタフェースを提供する．

● 適用可能性：

以下の利用局面で適用できる．

・システムを部品の生成・組み合わせ，表現の方法から独立にすべき場合．
・部品の集合が複数存在して，その中の１つを選んでシステムを構築する場合．
・一群の関連する部品を常に使用しなければならないように設計する場合
・部品のクラスライブラリを提供する際に，インタフェースだけを公開して，実装は非公開にしたい場合．

●構　造：

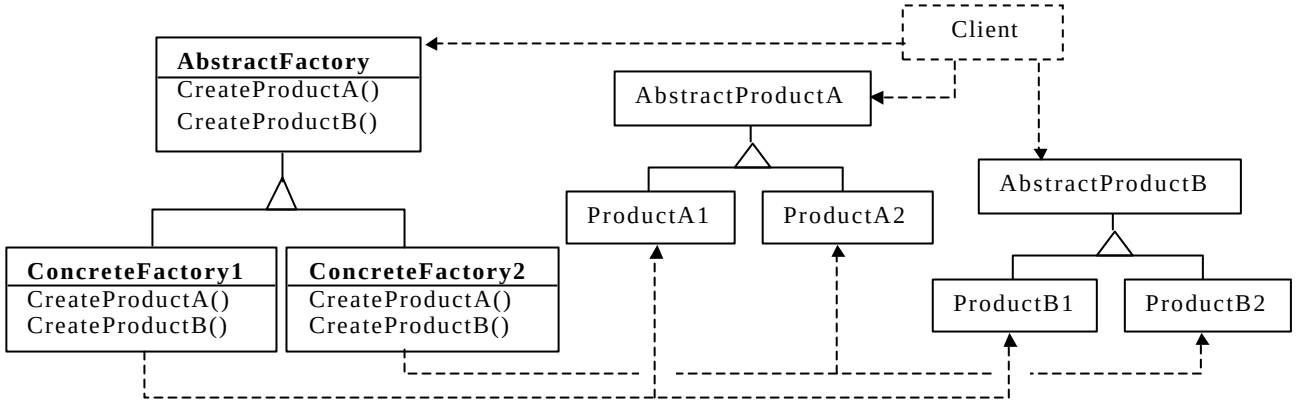


図 E-2　Abstract Factory デザインパターン

ここで，図 E-2 中の各クラスは次を意味する．

* AbstractFactory クラス

AbstractProduct オブジェクトを生成する操作のインタフェースを宣言するクラス.

* **ConcreteFactory クラス**
  ConcreteProduct オブジェクトを生成する操作を実装するクラス.
* **AbstractProduct クラス**
  部品ごとにインタフェースを宣言するクラス.
* **ConcreteProduct クラス**
  対応する ConcreteFactory オブジェクトで生成される部品オブジェクトを定義すると共に，AbstractProduct クラスのインタフェースを実装するクラス.
* **Client クラス**
  AbstractFactory クラスと AbstractProduct クラスで宣言されたインタフェースのみを利用するクラス.

(2) 構造決定に関する代表的デザインパターン

① Adapter デザインパターン

● **目　的：**

あるクラスのインタフェースをクライアントが求める他のインタフェースへ変換し，インタフェースに互換性のないクラス同士を組み合わせることができるようにする.

● **適用可能性：**

以下の利用局面で適用できる.

・既存のクラスを利用したいが，そのインタフェースが必要なインタフェースと一致していない場合.

・まったく無関係で予想持つかないようなクラス（必ずしも互換性のあるインタフェースを持つとは限らない）とも協調し再利用可能なクラスを作成したい場合.

・既存のサブクラスを複数利用したいが，それらすべてのサブクラスをさらにサブクラス化することで，そのインタフェースを適合させることが現実的でない場合.

● **構　造：**

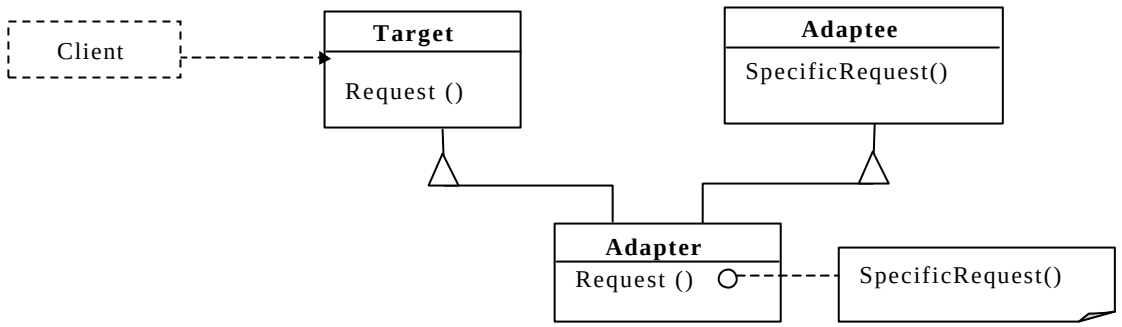


図 E-3　Adapter デザインパターン

ここで図 E-3 中の各クラスは次を意味する.

* **Target クラス**
  Client クラスが利用するドメインに特化したインタフェースを定義するクラス.
* **Client クラス**
  Target クラスのインタフェースに従ってオブジェクトにアクセスするクラス.
* **Adaptee クラス**
  適合させる必要のある既存のインタフェースを持つたクラス.
* **Adapter クラス**
  Adaptee クラスのインタフェースに Target クラスのインタフェースを適合させるためのクラス.

② Bridge デザインパターン

● **目　的：**

抽出されたクラスと実装を分離して，それらを独立に変更できるようにする.

● **適用可能性：**

以下の利用局面で適用できる.

・実装を実行時に選択したり交換する必要が生じることが判明してことから，抽出されたクラスとその実装を永続的に結合することを避けたい場合.

・抽出されたクラスとその実装の両方を派生クラスの追加により拡張可能にすべき場合.

・抽出されたクラスの実装における変更がクライアントに影響を与えたくない場合.

・複数のオブジェクト間で実装を共有し,それをクライアントから隠しておきたい場合.

● **構　造：**

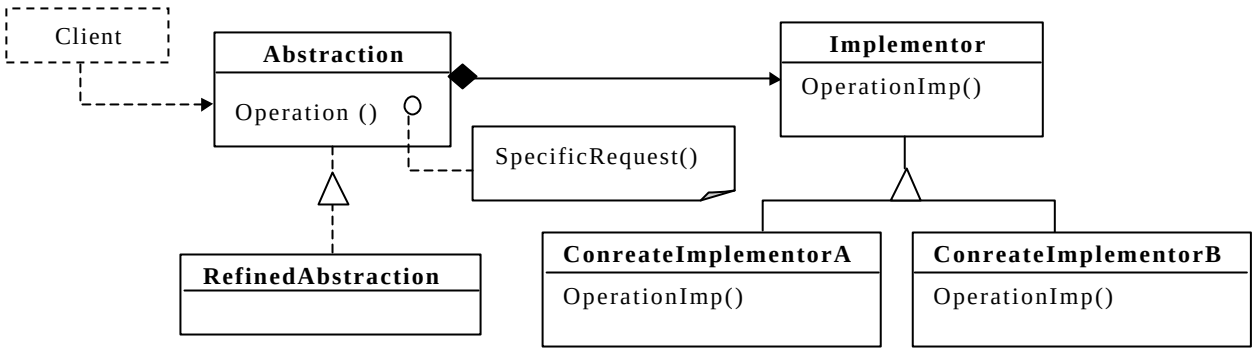


図 E-4　Bridge デザインパターン

ここで図 E-4 中の各クラスは次を意味する.

* **Abstraction クラス**
  抽出されたクラスのインタフェースを定義するクラス.Implementor 型のオブジェクトへの参照を保持する.

* **RefinedAbstraction クラス**
  Abstraction クラスで定義されたインタフェースを拡張する派生クラス.
* **Implementor クラス**
  実装を行うクラスのインタフェースを定義するクラス.このインタフェースは Abstraction クラスのインタフェースに必ずしも一致する必要はない. Implementor クラスのインタフェースはプリミティブな操作のみを提供しており, Abstraction クラスは, これらの操作を拡張してより高機能の操作を定義する.
* **ConcreteImplementor クラス**
  Implementor クラスのインタフェースを具体的に実装するクラス.

③ Composite デザインパターン

● **目　的：**

オブジェクトが持つ部分－全体階層を木構造によって表現する.クライアントは, 個々のオブジェクトとオブジェクトを合成した複合オブジェクトを同じ操作で扱うことができるようになる.

● **適用可能性：**

- オブジェクトの部分? 全体階層を表現したい場合.
- クライアントが,オブジェクトを合成した複合オブジェクトとその構成要素である個々のオブジェクトの違いを無視できるようにしたい場合. クライアントは, composite 構造内のすべてのオブジェクトを一様に扱うことができる.

● **構　造：**

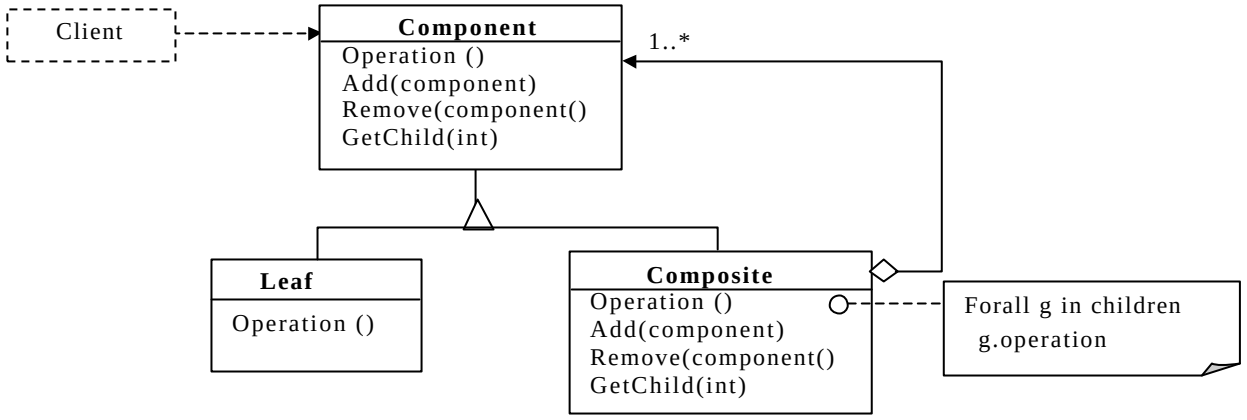


図 E-5　Composite デザインパターン

ここで図 E-5 中の各クラスは次を意味する.

* **Component クラス**
  composite 内のオブジェクト（component と呼ぶ）のインタフェースを宣言するクラス. すべてのクラスに共通なインタフェースのデフォルトの振る舞いを適宜実装す

る．Component オブジェクトにアクセスしたり，それを管理するためのインタフェースを宣言する．再帰構造において,全体－部分の全体にあたる composite にアクセスするためのインタフェースを宣言しておき，適宜実装する．

* **Leaf クラス**
  composite 内の末端のオブジェクト（leaf と呼ぶ）を表し，leaf はさらに部分となるオブジェクトをもたない．また，composite 内のプリミティブなオブジェクトの振る舞いを定義する．

* **Composite クラス**
  部分を構成するオブジェクトを持つ component（すなわち composite）の振る舞いを定義するクラス．部分にあたる component を保持すると同時に，Component クラスのインタフェースで宣言された部分オブジェクトに関する操作を実装する．

* **Client クラス**
  Component クラスのインタフェースを通して,composite 内のオブジェクトを操作するクラス．

④ Decorator デザインパターン

● **目　的：**

オブジェクトに責任を動的に追加し，派生クラスを追加する方法よりも柔軟な機能拡張方法を提供する．

● **適用可能性：**

・個々のオブジェクトに責任を動的，かつ他のオブジェクトには影響を与えないように追加，削除する場合．
・派生クラス化による拡張が非常に多くの独立した組み合わせの拡張を必要な場合．
・莫大な数の派生クラスが必要になるが，クラス定義が隠ぺいされている場合や入手できない場合．

● **構　造：**

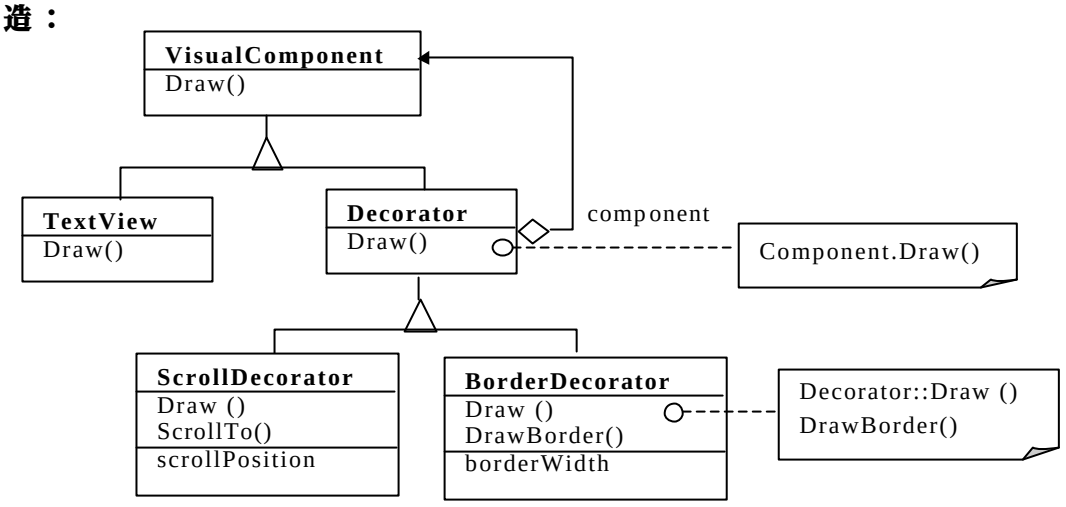


図 E-6　Decorator デザインパターン

ここで図 E-6 中の各クラスは次を意味する．

* **Component クラス**
  責任を動的に追加できるようになっているオブジェクトのためのインタフェースを定義するクラス．
* **ConcreteComponent クラス （TextView クラス）**
  責任を追加できるようになっているオブジェクト（component）を定義するクラス．
* **Decorator クラス**
  component または decorator への参照を保持し，また Component クラスのインタフェースと一致したインタフェースを定義するクラス．
* **ConcreteDecorator クラス（BorderDecorator クラス，ScrollDecorator クラス）**
  component に責任を追加するオブジェクト（decorator）を定義するクラス．

⑤ Proxy デザインパターン

● **目　的：**
あるオブジェクトへのアクセスを制御するために，そのオブジェクトの代理，または入れ物を提供する．

●**適用可能性：**
・remote proxy は，別のアドレス空間にあるオブジェクトのローカルな代理を提供する．
・virtual proxy は，コストの高いオブジェクトを要求があり次第生成する．
・永続オブジェクトが初めて参照されたときに，それをメモリ上にロードする．

●**構　造：**

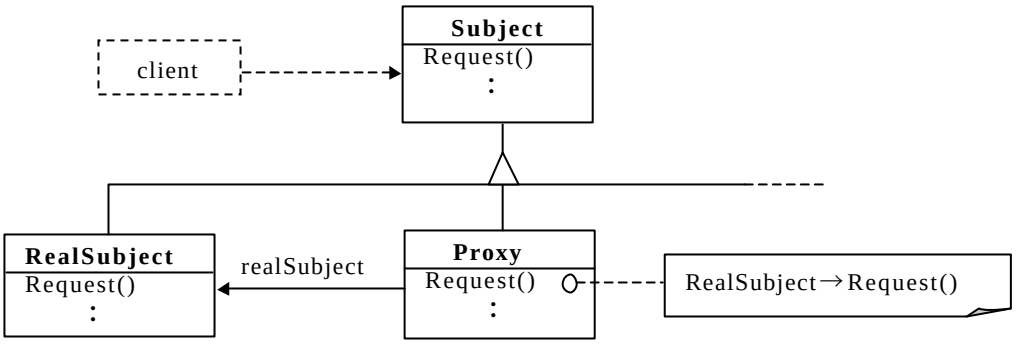


図 E-7　Proxy デザインパターン

ここで図 E-7 中の各クラスは次を意味する．

* **Proxy クラス （ImageProxy クラス）**
  RealSubject オブジェクトにアクセスするための参照を保持する．RealSubject クラスのインタフェースと Subject クラスのインタフェースが等しい場合には，Proxy クラスは Subject のオブジェクトを参照する．

RealSubject オブジェクトへのアクセスを制御すると同時に，RealSubject オブジェクトの生成や消去に責任を持つ.

* **Subject クラス**

RealSubject オブジェクトを利用できるところならばどこでも Proxy オブジェクトを利用できるように,RealSubject クラスと Proxy クラスに共通のインタフェースを定義するクラス.

* **RealSubject クラス （Image クラス）**

Proxy オブジェクトがその代理を務めることになる実オブジェクトを定義したクラス.

⑥ Facade デザインパターン

● 目　的：

サブシステム内に存在する複数のインタフェースに１つの統一インタフェースを与える.サブシステムの利用を容易にするための高レベルインタフェースを定義する.

●適用可能性：

・複雑なサブシステムに統一したインタフェースを提供したい場合.
・ある抽象を実装しているクラスとクライアントの間に多くの依存関係がある場合.
・あるサブシステムをクライアントや他のサブシステムから切り離して，独立性や移植性を高める場合.
・サブシステムを階層化し，各階層の各サブシステムへの入り口を定義する場合.

●構　造：

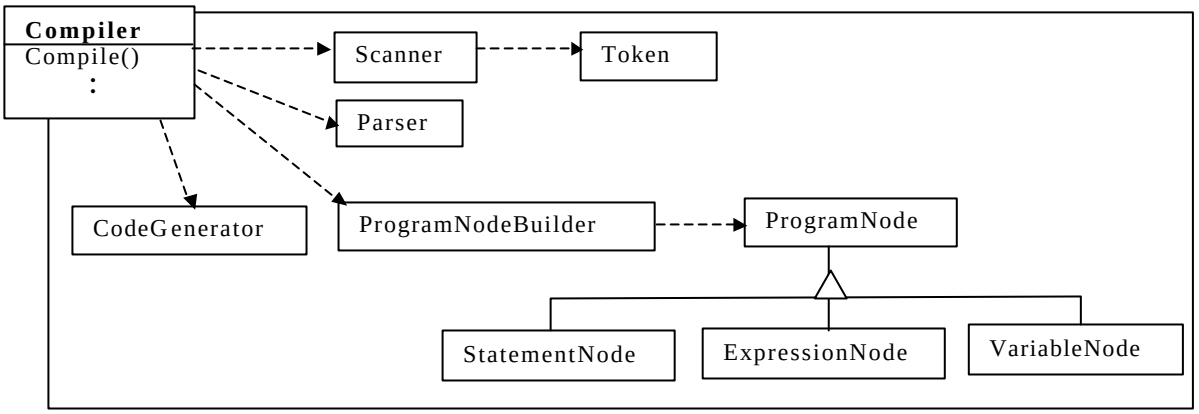


図 E-8　Facade デザインパターン

ここで図 E-8 中の各クラスは,Compile を呼び出しの入り口とする Compiler の構成要素クラスを意味する.

(3) 振る舞い決定に関する代表的デザインパターン

《 Observer デザインパターン 》

● 目　的：

あるオブジェクトが状態を変えたとき，それに依存するすべてのオブジェクトに自動的に変更を通知すると同時に，それらが更新されるようオブジェクト間に一対多の依存関係を定義する．

● **適用可能性：**

・１つのオブジェクトを変化させるとき，それに伴って他のオブジェクトも変化させる必要があり，かつ変化させる必要があるオブジェクトを固定的に決められない場合．

・オブジェクトが他のオブジェクトに対して，密なる結合関係を作りたく場合．

●**構　造：**

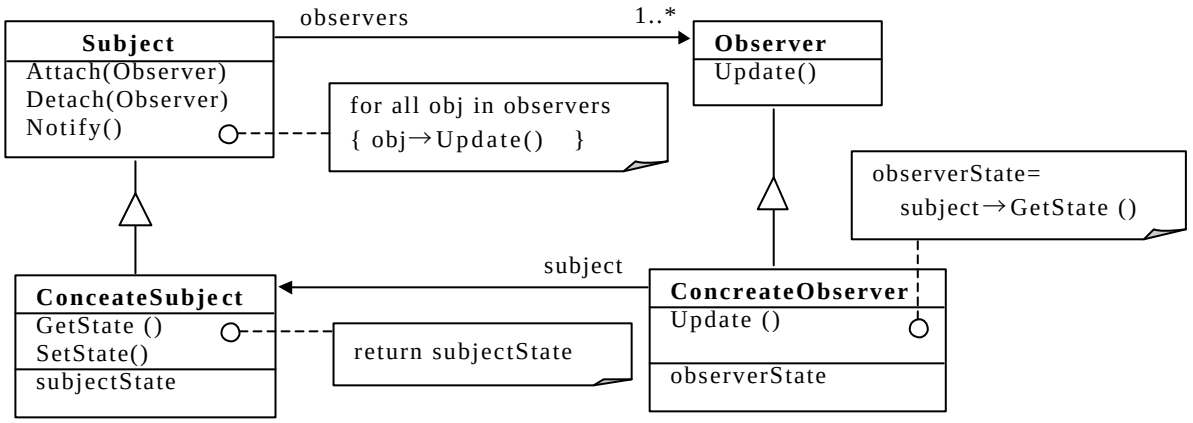


図 E-9　Oberver デザインパターン

ここで図 E-9 中の各クラスは次を意味する．

* **Subject クラス**

任意の数の observers を subject の状態変化に対応して通知する相手として，observers を追加，削除するためのインタフェースを提供するクラス．

* **Observer クラス**

subject からの状態変化を通知されたとき，自分自身の状態を更新するインタフェースを定義したクラス．

* **ConcreteSubject クラス**

ConcreteObserver オブジェクトに影響する状態を保存するクラス．ConcreteSubject の状態が変わったときに ConcreteObserver オブジェクトに通知を送る．

* **ConcreteObserver クラス**

ConcreteSubject オブジェクトへの参照を保持しているクラス．その状態を ConcreteSubject オブジェクトの状態と矛盾しないよう保存する．その状態を ConcreteSubject オブジェクトの状態と矛盾しないようにしておくために，Observer クラスで宣言した更新のインタフェースを実装する．