ライフタイム分析にもと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

大木　幹雄†

あらまし

要求仕様からクラス構造を抽出する方法として，属性やメソッド等の基本的な分析要素がもつ特性から，ボトムアップにクラスを抽出するアプローチは，アプリケーションドメインに関する知識がない分析者でも，正しいクラス構造に比較的容易に到達できる長所をもつ．本論では，要求仕様から属性やメソッドが識別名と存在寿命を含めて洗い出されることを前提にして，存在寿命を活用し，クラスの抽出とクラス構造の構成過程を形式化する「構成演算」について提案する．次いでこれら構成演算を用いると，再帰構造を含むクラス構造であっても抽出しうることを示し，その構成演算の有効性を検証するため，構成演算を用いて代表的な構造に関するデザインパターンを抽出した机上実験の結果を述べる．最後に Pree のメタパターンアプローチとの相違について考察した後，オブジェクト指向システム開発における存在寿命分析の重要性と要求仕様書のあり方について論じる．

Formalization of Class Structure Extraction through Lifetime Analysis and Extraction of Structure Design Pattern

Mikio OHKI †

Abstract

For an analyst who tries to extract class structures from given requirements specifications for an application area with which he/she is not familiar, it is usually easier first to extract analysis elements, such as attributes, methods, and relationships, then to compose classes from those elements, than to extract entire classes at the same time. This paper demonstrates how to define the set of operations that can be used to derive lifetime-based class structures, provided that methods, including their identification names and lifetimes, can be extracted from given requirements specifications. The latter part of this paper describes an experiment that validates the defined operations by deriving typical design patterns, and also describes the differences between my approach and Pree's meta-pattern approach. Finally, it discusses the important role of lifetime analysis and an effective style of requirements specifications for object-oriented system development.

1. まえがき

オブジェクト指向ソフトウェア開発で最も重要かつ経験を要する作業として,クラス構造の抽出がある．クラス構造の抽出には，豊富な業務知識や深い洞察力が必要とされることから，抽出の補助手段として，責任駆動アプローチ[1)]やユースケース分析[2)]など，責任や利用シナリオの視点からクラス抽出を行う手法が考案されている．しかしこれらの手法を用いたとしても，最終的には分析者の「ひらめき」によってクラスを発見しなければならない[3)]．そのため，アプリケーションドメインに関する知識がない分析者が，正しいクラスを抽出することは至難の技になる[4)]．理想的には，要求仕様から一定の判断基準にしたがった手順を踏むことで，正しいクラス構造が抽出できることが望ましい．

一方，企業情報システム開発の分野では，データベースの出来いかんがシステムの品質を左右することから，正しい ER（Entity Relationship）モデルを抽出する判断基準が従来から模索されてきた．たとえば，データを中心にしたモデル指向の方法論である DATARUN[5)]では，帳票や伝票から収集・分析したデータ項目をもとに，ボトムアップに実体型(Entity Type)を抽出するため，PDG（Primary Data Generator）と呼ばれる「データ項目の実現値が決定されるきっかけ」を一つの判断基準として用いている．この判断基準では，他のデータ項目から演算等で導出できない「基本データ項目」の中から，同じ PDG をもつ基本データを同一の実体型に属する属性として分類し，分類後の属性名集合を手掛りにして，実体型とその名称を決定する．その根拠は,「実体型からインスタンスを生成したとき，インスタンスがもつ属性の初期値は同時に決定されることから，逆に，特定の PDG によって初期値が同時に定まる基本データ集合は，同じ実体型に属する属性集合となる」点にある．

筆者は，PDG の概念を一般化すると共に，実現値の多値度（=同時に決定される実現値の数）や状況数（実現値が決定される状況の数）等を加えた ER モデリングの判断基準を考案し，概念データモデリング演習授業の学生を対象にして，その有効性を検

†日本工業大学工学部　情報工学科
Nippon Institute of Technology, Department of Computer and Information

証するための比較実験を行った．その結果，判断基準を用いたグループが正しい ER モデルに到達する割合は，用いないグループに比較して，統計的に有意な水準で向上することが実証された[6]．

この比較実験の意義は，時間的な視点である「きっかけ」の発生時点を一つの判断規則として用いると，ドメイン知識の少ない分析者でも，正しい ER モデルに到達できることを実証した点にある．しかしながら，それらの判断基準のみでは，再帰構造を含む ER モデルやメソッドを中心としたクラス構造までは抽出できない問題点が残されていた．本論では，これらの問題点を解消するために，新たにソフトウェア場，および存在寿命の概念を判断基準の概念基盤として追加し，演算形式を用いた判断基準の定式化を図る．同時にそれらの演算を用いて，典型的なクラス構造が正しく抽出できるかを「構造に関するデザインパターン」を用いて検証する．

本論の構成は次のとおりである．2 章ではクラス分析の基本的な特徴にもとづき導入した判断基準の基盤となるソフトウェア場概念について述べる．3 章では，クラス分析で用いる判断基準に数学的な根拠を与えるために考案した構成演算の概念と演算規則について述べ，4 章でクラス分析手順との関係について示す． 5 章では，これらの構成演算の妥当性を実証するため，要求仕様から洗い出した分析要素に構成演算を作用させ，GoF の代表的な「構造に関するデザインパターン」[7),8)]が導き出せることを示す．6 章では Pree のメタパターン[9),10),11)]アプローチ等との比較を行い，最後に本論が提案するアプローチのまとめと今後の展望について述べる．

2. クラス分析過程のモデル化

2.1 分析過程の特徴

ボトムアップアプローチによるクラス分析では，分析者は，帳票に記載されたデータや CRC（Class Responsibility Collaborations）カード，ユースケースシナリオを用いて，要求仕様から基本データ名（=属性名）や機能名（=パラメータ部を含むメソッド･インタフェース名. 以後, 単にメソッド名と呼ぶ）の候補を洗い出し，次いでそれらをまとめ上げ，まとまりにふさわしいクラス名を命名する作業を行う．これらの洗い出しやクラスへのまとめ上げ作業には，次の特徴がある．

(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[12)]．それらの存在は，あいまいであり，場合によっては，概念的に同一のもの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，責任名やメソッド名, クラス名がもつ意味的な関係のみである．

(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，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）．

以上に述べた特徴のうち，特に(3)は重要で，冒頭で述べた ER モデルの比較実験において，時間的な「きっかけ」の同一性が，エンティティ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，存在寿命を積極的に活用した分析手法は，現在，存在しない．

2.2 ソフトウェア場モデル

上述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[13),14),15),16),17)]．ソフトウェア場とは，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，ボトムアップにクラス構造を抽出する過程をモデル化するために導入した概念である．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**」と呼ぶことにする．

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・インタフェース」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，構成子同士がその間に働く力（3.1 節で述べる）によってまとまりを形成し，クラスを構成してゆく過程であると捉える．具体的には，メタクラス「構成子」から生成されたインスタンスとして構成子が，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．

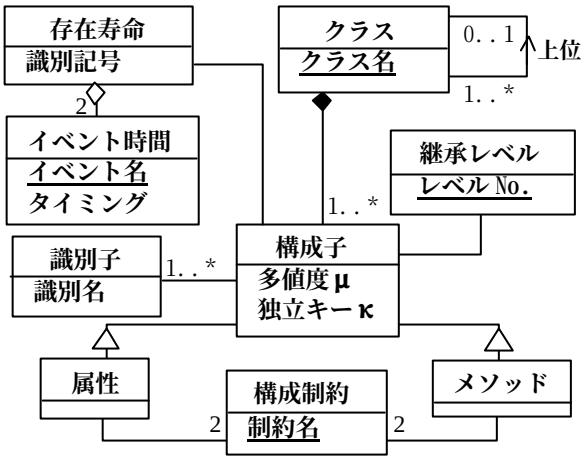

図 1 ソフトウェア場を構成する要素のメタモデル
Fig.1 The software field meta-model

クラス構造を形成する上で用いられる判断基準は，構成子集合を内部要素としてもつメタオブジェクト「ソフトウェア場」がもつメタルールに相当し，本論では，それらを演算として定義する．「ソフトウェア場」がもつ内部要素の関連は，図１のメタモデル図で示すとおりである．その意味を次節から述べる．

2.3　ソフトウェア場がもつ座標系

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が，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つの空間として捉え，メタ属性として次の座標系軸をもたせる．

(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」作業である．識別子軸χは，ドメイン毎に洗い出された「用語」を「名義的な尺度としての座標値」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同義の識別子は，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値の設定タイミングやメソッドが必要とされる「きっかけ」時間の概念を一般化したものである．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」等のイベント名をもつ．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（具体的には，属性の初期値設定タイミングやメソッドの実装決定タイミング，および属性，メソッドが意味を失うタイミングを指す．ただし，タイミングは，前後関係のみが意味をもった名義的な指標に過ぎない）をひと組にして，タイミングがもつトポロジカルな時間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．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，すなわち，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もつ構成子は，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，継承の下位になるにしたがい，継承レベルは１づつ増加してゆく．後述する排他制約によって，構成子は順次高い継承レベルに配置され，継承階層を構成してゆく．

2.4　ソフトウェア場内の構成子の表現

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，統一化された識別名で扱われる)．

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 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によって表現できる．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，前述の①②の仮定を満たす表現形となる．

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 2 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．図 2 では，理解を容易にするため，χ-τ-ν軸空間と存在空間χ-τ-ρ軸を重ね合わせて示している．ただし，ρは構成子の存在確率を意味する．

$$\phi(\chi,\tau;\tau_1,\tau_2) = ?(\chi,\tau-\tau_1) - ?(\chi,\tau-\tau_2) \quad (式1)$$

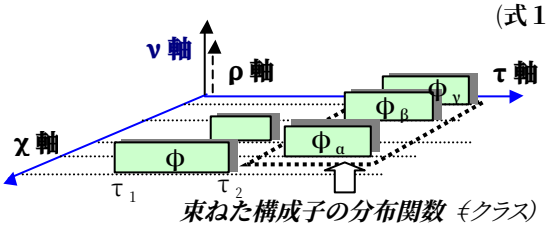

図 2　構成子の分布関数

Fig.2　An analysis element as a distributed function

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」ことを前提とした本研究の基本思想を反映したものである．分布関数がもつ存在寿命は，後に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もつ．

2.5　構成子のメタ特性

（式 1）で示した分布関数は，構成子の存在状態を示すもので，構成子のインスタンス（=属性の実現値や実装されたメソッドの実行プロセス）の存在状態を表しているわけでない．

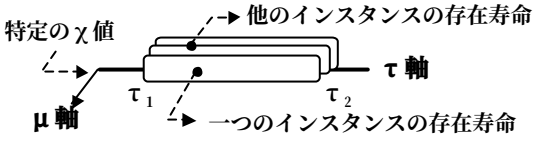

図 3　インスタンス空間における構成子

Fig.3 The existing style of an analysis element in the instance space.

構成子から生成されたインスタンスは，図 3 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空間とは異なる次元のμ軸方向に重ね合わせ状態として置かれる．存在寿命もメタオブジェクトとしての構成子のそれとは異なる．インスタンス空間μにおける構成子のインスタンスの分布状態をもとに，各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実現値としてのインスタンスが生成されたとき，「同時に」生成されるインスタンスの個数を示したものである．すなわち，構成子が属性のときは，多値度は「**属性の実現値の個数**」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には，多値度は 2 以上になる．

(b) 独立キーκ

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，メタ特性値「独立キー」をもつ．構成子が属性のとき，メタ特性値「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は，「他のメソッドを呼び出す側のメソッド」に相当し，必要に応じて呼び出されるメソッドは，この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，呼び出し側メソッドは，メタ特性値「独立キー」をもつ．

2.6 構成子間に働く制約

オブジェクト指向のもつ基本的な特徴から，構成子に次の制約をもたせる．

(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以外の排他度をもつことはできない．すなわち，同じ識別名をもつprivate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．すなわち，サブクラス化によるメソッドの再定義が行える．

(2)多値度制約υ

構成子の実現値が多値のとき，言い換えると，一つの属性が同時に複数の実現値をもつときや，あるいは一つのメソッドが同時に複数の実装をもつときには，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，インスタンスがもつすべての構成子の実現値は，一意に定められていなければならないことから来る制約で，ER モデルにおける「正規化」に相当する[6]．

この制約から，多値と単値の構成子が混在するクラスは，多値の構成子集合が強制的に分離され，単値の構成子集合のみをもつインスタンス（すなわち実現値が一意的に定まる構成子集合からなるインスタンス）を複数個，同時に生成するクラスに置換される．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，分離後に残った単値の構成子のみをもつクラスに対して「存在依存」する．なぜなら，両者は同じ存在寿命をもつ構成子集合を強制的に分離したものであり，単値の構成子のみのクラスが存在を失えば，そこから分離された多値のクラスもその存在を失うからである．

3. クラスの抽出 ･構成演算の定義

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を構成する演算を定義する．

3.1 クラスの抽出演算

構成子が存在寿命をもった分布関数であることを反映して，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 2)で定義する．$F_{\alpha\beta}$は図 4 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をもつ関数$\phi(\chi,\tau;\tau_1,\tau_2)$の省略形を意味する．生成・消滅のイベント時間間隔$[\tau_1,\tau_2]$を存在寿命の識別記号$\tau_\alpha$で表すことから，$\Phi$の存在寿命が$\tau_\alpha$であることを明示するために，αを添え字付けする．$\Phi_\beta$も同様とする．(式 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α，β，．．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．

$$F_{\alpha\beta} ::= f_{\alpha\beta}(\tau_\alpha;\tau_\beta) \equiv 2*\Phi_\alpha(\tau_\alpha)*\Phi_\beta(\tau_\beta) / \{\,|\tau_\alpha| + |\tau_\beta|\,\}$$

ただし　$\alpha \neq \beta$　　(式 2)

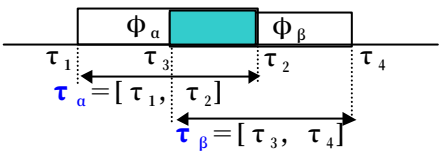

図 4 構成子の重なりの割合を示す代数積演算の定義
Fig.4 Definition of the algebraic product operation

(2) 分布関数の代数和演算

(式 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代数和を算出する演算を(式 3)で定義する. 図 4 を例にとると, $\Phi_\alpha(\tau_\alpha)$, $\Phi_\beta(\tau_\beta)$ から, $[\min(\tau_1, \tau_3), \max(\tau_2, \tau_4)]$ の存在寿命をもつ $S_{\alpha\beta}$ を作成する.

$$S_{\alpha\beta} ::= \Phi_\alpha(\tau_\alpha) + \Phi_\beta(\tau_\beta) ? f_{\alpha\beta}(\tau_\alpha; \tau_\beta) \quad (式 3)$$

(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 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 をもっ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, メタ特性値が「独立キー」をもつ構成子を固定して, それとの代数積の値が 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 4)の値は 1 を取ることから, (式 4) は, 同一の存在寿命をもつ構成子を束ね, クラスを抽出する演算を定義することになる. 独立キーをもつ構成子が存在しないときには, (式 4) が定義されないため, クラスは必ず独立キーをもった属性か, またはメソッドが存在しなければならない.

$$\text{Entity/Class } \Phi ::= \text{Select } \Pi f_{\alpha\beta}(\tau_\alpha; \tau_\beta) \quad (式 4)$$

3.2 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, それぞれ以下を意味するものとする.

《 構成子集合に関して 》

① $\{\tau_\alpha \alpha, \tau_\beta \beta\}$: 存在寿命が τ_α, τ_β 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 をもつ構成子 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 の集合.

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 に分離した表現.

《 クラス構造に関して 》

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 の代数和.

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. 存在寿命は構成子と等しい.

⑨ $\{\tau_\alpha \alpha, \tau_\beta[\alpha]\}$: 存在寿命が τ_α の構成子 α と存在寿命 τ_β の構成子 α をもつクラスが混在した状態 (中間状態).

⑩ [β]⊿[α] : 構成子 α をもつクラス[α]は構成子 β をもつクラス[β]の上位クラス. [α]を継承の基底項と呼び, [β]を継承の被基底項と呼ぶ.

⑪ [β]◇[α]: 構成子 α をもつクラス[α]と構成子 β をもつクラス[β]は集約関係にある. [α]を集約の基底項と呼び, [β]を集約の被基底項と呼ぶ.

⑫ [β]◆[α]: 構成子 α をもつクラス[α]と構成子 β をもつクラス[β]は存在依存の集約関係にある. そのほかは⑪と同じ.

⑬ [β]* :クラス[β]は, 複数のインスタンスを同時に生成するクラス.

⑭ [α]+[β] : 構成子 α, β を構成要素としてもつクラス[α], [β]は独立した存在.

3.2.1 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なっている.

(1) 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

「2.6 構成子間に働く制約」で述べた多値度制約 υ から, 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, 多値度が 1 である構成子のみをもち, 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.

$$\{\tau_\alpha \alpha^*\} \Rightarrow \tau_\alpha[\alpha]^* \quad (式 5)$$

ii) 集合要素の性質にもとづくクラス化

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 つである.

$$\{\tau_\alpha \alpha, \tau_\alpha \alpha\} = \{\tau_\alpha \alpha\}$$
$$\Rightarrow (\tau_\alpha[\alpha] + \tau_\alpha[\alpha]) = \tau_\alpha[\alpha] \quad (式 6)$$

(2) 排他制約 π にもとづく汎化演算

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, 同じ識別名をもつ構成子の分布関数の代数和をとった新たな構成子の分布関数を作成し, その構成子を上位クラスの構成要素として配置する. 汎化演算の適用によって構成された上位クラスの存在寿命が, 下位クラスの代数和となる根拠は,「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上位クラスも存在する」とのオブジェクト指向の基本機構から, 上位クラスがもつ構成子の存在寿命は, 下位クラスの分布関数の代数和となるためである. したがって, 汎化で作成されたクラスの存在寿命は, 汎化前の個々のクラスがもつ存在寿命より拡大する.

汎化演算には, オブジェクト指向の基本的な特徴に対応して, (式 7), (式 8) の 2 通りの定義がある.

i) 継承を利用した一般的な汎化

$$\{\tau_\alpha\{\underline{\alpha}, \beta\}, \tau_\gamma\{\underline{\alpha}, \gamma\}\}$$
$$=\{\tau_\alpha \underline{\alpha}, \tau_\alpha \beta, \tau_\gamma \underline{\alpha}, \tau_\gamma \gamma\}$$
$$\Rightarrow (\tau_\alpha[\beta] + \tau_\gamma[\gamma]) \triangle (\tau_\alpha \cup \tau_\gamma)[\underline{\alpha}] \quad (式 7)$$

ii)　**継承を利用して抽象メソッドを再定義する汎化**

$\{\ \tau_\alpha\{\underline{\alpha},\beta\},\ \tau_\gamma\{\underline{\alpha},\gamma\}\ \}$
$=\{\tau_\alpha\underline{\alpha},\tau_\alpha\underline{\alpha},\tau_\alpha\beta,\tau_\gamma\underline{\alpha},\tau_\gamma\underline{\alpha},\tau_\gamma\gamma\}$
$\Rightarrow(\tau_\alpha[\underline{\alpha},\beta]+\tau_\gamma[\underline{\alpha},\gamma])\triangleleft(\tau_\alpha\cup\tau_\gamma)[\underline{\alpha}]$
(式 8)

ここで，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である．理解を容易にするため，(式 8)で示した変換規則の意味を図 5 で示す．

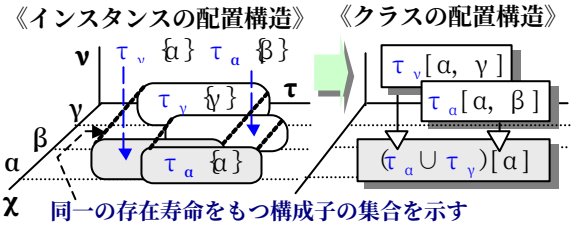

図 5　存在寿命から見た抽象メソッドを用いた汎化演算
Fig.5 A supper class abstraction operation depending on the lifetime

汎化演算の特徴として，上位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てクラス構造を拡張（=extend）しても，最上位クラスの存在寿命は拡大することに留意する必要がある．

(3) 多値度制約 υ にもとづく集約演算

「2.6　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，同じ存在寿命をもつが多値度は異なる構成子集合が存在したとき，それらを分離し，集約関係をもつクラス構造を構成する演算である．存在依存の集約関係として，(式 9) で定義する．

$\upsilon:\ \tau_\alpha\{\alpha,\beta^*\}=\{\tau_\alpha\{\alpha\},\tau_\alpha\{\beta^*\}\}$
$\Rightarrow\{\tau_\alpha[\alpha],\tau_\alpha[\beta]^*\}=\tau_\alpha[\beta]^*\blacklozenge\tau_\alpha[\alpha]$
(式 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，クラスの「存在寿命の合計値」を増大する演算と見ることができる．

3.2.2　その他の構成規則

その他の構成規則として以下がある．

(1) 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
$\triangleleft(\tau_\alpha\cup\tau_\gamma)[\alpha]\rightarrow\triangleleft\tau_\alpha[\alpha]$　(式 10)

(2) 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 3.3 節で述べる)．

$\pi:\{\tau_\beta\{\alpha^*,\beta\},\ \tau_\gamma\{\alpha^*,\gamma\}\}$
$\Rightarrow(\tau_\beta[\beta]+\tau_\gamma[\gamma])\triangleleft(\tau_\beta\cup\tau_\gamma)[\alpha]^*$
または
$\Rightarrow(\tau_\beta[\alpha,\beta]+\tau_\gamma[\alpha,\gamma])\triangleleft(\tau_\beta\cup\tau_\gamma)[\alpha]^*$
(式 11)

(3) 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汎化演算の対象とする．その根拠は，集約関係において，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のみが対象になるためである．

$\{\tau_\alpha[\alpha],\ \tau_\gamma[\gamma]\diamond\tau_\beta[\alpha]\}$　// 中間状態
$=(\tau_\alpha[\alpha]+\tau_\gamma[\gamma]\diamond\tau_\beta[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$
(式 12)

(4) 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，存在寿命が長い継承の基底項)の構成子を汎化演算の対象とする．その根拠は，汎化演算がクラス構造の存在寿命の拡大を図ることにあることから導かれる．

$\{\tau_\alpha[\alpha],\ \tau_\gamma[\gamma]\triangleleft\tau_\beta[\alpha]\}$　// 中間状態
$=(\tau_\alpha[\alpha]+\tau_\gamma[\gamma]\triangleleft\tau_\beta[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]$
(式 13)

(5) 集約関係の基底となる構成子が同一の存在寿命をもつとき，それらをまとめる（基底項を優先する理由は(3)と同じである)．

$\{\tau_\beta\beta\in\tau_\alpha\{\alpha,\mu_1\},\ \tau_\gamma\gamma\in\tau_\alpha\{\alpha,\mu_2\}\}$
$\Rightarrow\{\tau_\beta\beta\in\{\tau_\alpha\{\alpha,\mu_1\},\tau_\alpha\{\alpha,\mu_2\}\},$
$\tau_\gamma\gamma\in\{\tau_\alpha\{\alpha,\mu_1\},\tau_\alpha\{\alpha,\mu_2\}\}\}$
(式 14)

3.2.3　クラス構造の変換規則

クラスが特定の構造をもつとき，次の変換規則でクラス構造を簡略化する．

(1) 自明なクラス構造の簡略化

$\tau_\alpha[\alpha]\triangleleft\tau_\alpha[\alpha]=\tau_\alpha[\alpha]$　(式 15)

(2) 下位クラスの置換規則

他のクラスを集約の要素としてもつクラスが継承の下位クラスのとき，(式 16) および図 6 で示すとおり，集約関係を継承の上位クラスの集約関係に置換する．これは継承と集約関係を比較したとき，独立したクラス間の集約関係は，それらのクラス間の「存在寿命の合計値」を増大させるに過ぎないが，継承はクラス構造全体の存在寿命を拡大する．したがって，全体の存在寿命の拡大を図るために，継承を集約関係に優先して適用するためである（詳しくは次節で述べる)．

$\tau_\beta[\beta]\diamond\tau_\gamma[\alpha]+\tau_\gamma[\alpha]\triangleleft\tau_\alpha[\alpha]$
$=(\tau_\beta[\beta]\diamond\tau_\alpha[\alpha])+\tau_\gamma[\alpha]\triangleleft\tau_\alpha[\alpha]$
(式 16)

以上から，図 6 左図で示したクラス構造は，全体の存在寿命を増大させるために，継承をたどり，図 6 右図のとおりに置換する．

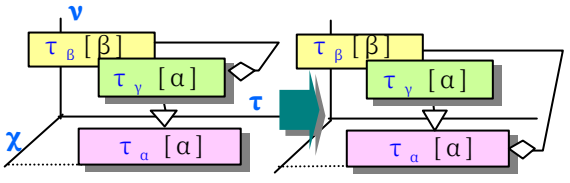

図 6　下位クラスの集約関連を置換する規則
Fig.6 A substitution rule of the subclass aggregation

3.3 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上位クラスを構成する．その影響はクラス構造の全体を指し示す最上位クラスまで及びその存在寿命を拡大させる．一方，集約演算は，クラス構造の存在寿命の合計値を増大させるだけである．

存在寿命から見て，クラス構造は，全体の存在寿命が最も拡大したときが，最も妥当な構成と考えてよい．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，汎化演算の方が，単に存在寿命の「合計値」を増大させる集約演算よりも重要度は高くなる．その結果としての継承も，集約関係に優先する．「2.6　構成子間に働く制約」で述べた排他制約を多値度制約に優先させる適用規則は，上述の考え方を反映したものである．

構成演算の適用に優先度をもち込むと，集約演算と汎化演算の適用が順序化されることから，クラス構造の抽出にそれらの演算を適用したときの一意性は保たれる．構成演算列の適用に関する厳密な解の一意性証明については，本論の目的から逸脱するため，別稿で議論したい．

3.4　関連の基数の決定ガイドライン

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もつクラスは，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，独立キー特性をも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もつクラス間で，主キー特性をもつクラス 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もつ．すべてのイベント時間集合を共有するときは，1 対 1 の関連をもつ．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．

《 1 対多関連 》

-- $\tau_\gamma[\alpha]$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 》

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

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

※ α, β は独立キー特性をもつ構成子とする．

4．構成演算によるクラス分析手順

前章で述べた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．それらの存在寿命は，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，構成演算の適用において重要な役割を果たすイベント時間の具体的な決定手順について述べる．

4.1 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，名義尺度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分析手法に，次の拡張を加えることで行える．

4.1.1 ビジネスプロセス図を拡張した方法

冒頭で述べた PDG と同じ発想であり，ビジネスプロセス図を記述し，外部エンティティ毎に，それぞれ起動するプロセス連鎖の「きっかけ」となる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．

(1) プロセス毎のイベント名の問合せ

下記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プロセスがも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．

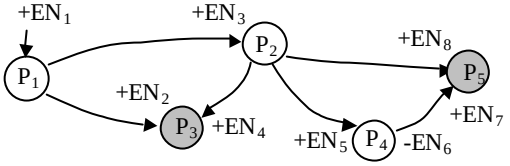

図 7　ビジネスプロセス図によるイベント名の抽出

Fig.7 Extracting event names from a business process diagram

イベント名が明らかになると，図７で示すようにイベント名（EN: Event Name）をビジネスプロセス図にそれぞれ書き込む．図７中の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．

(2) 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トポリジカルソートした順序木を作成する．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７で示した例では図８のとおりになる．

① パス$_1$：P_1→P_5 上のイベント順序
EN_1→EN_3→EN_8 ¦ EN_1 →EN_3→EN_5→EN_6→EN_7

② パス$_2$：P_1→P_3 上のイベント順序：
EN_1→EN_3→EN_4 ¦ EN_1→EN_2

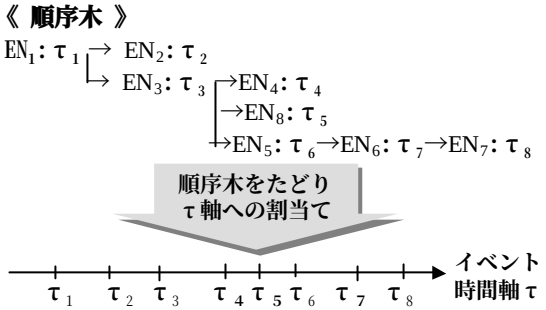

図８ 順序木をたどったイベント時間の割当て
Fig.8 Assignment of event time by traversing the ordered tree

4.1.2 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．

表１ きっかけを付加した「図書館貸出管理」のシナリオ例
Table.1 An Example of Library Lent Management scenari considered the trigger event

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

「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１に示す．

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し，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を用いる．

4.2 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９で示すとおりにまとめられる．図９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,「構成子の識別子, 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．

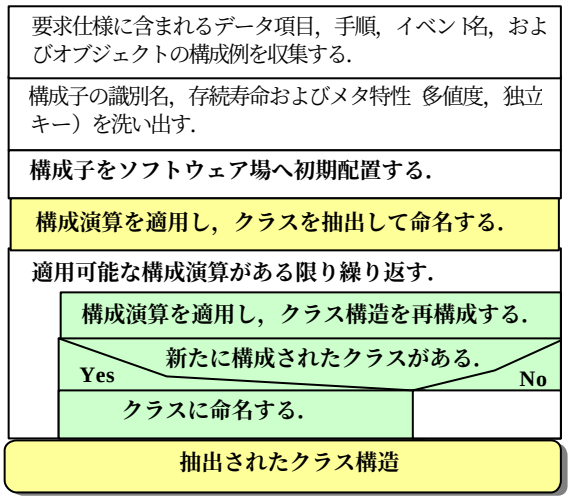

図９ 構成演算を基盤にした分析手順（NS 図）
Fig.9 The system analysis methodology based on the software field

5．構成演算の妥当性検証実験

本章では，図９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実験を行う．ここでは典型的なクラス構造である「GoF の構造に関するデザインパターン」を検証実験用の例題として用いる．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適用列で典型的な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．

「構造に関するデザインパターン」を例題として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も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もつ目的や動機にさかのぼって，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．

5.1 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新たに追加したい．具体的には，分析によって新たに存在時間 τ_γ をもつメソッド集合{ 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たい．ここで，τ_γ は初期構成子集合の存在寿命の重複部分 $(\tau_\alpha \cap \tau_\beta)$ 内にあるものとする．

(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 17a)で表す．Ψ_0 にクラス構成演算を適用すると，(式 17b)のとおりに変形できる．

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

ただし，$(\tau_\gamma \subset (\tau_\alpha \cap \tau_\beta)) \wedge ((\tau_\alpha \cap \tau_\beta) \neq \phi)$

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

/-- 式 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 --/*

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

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

(4) 抽出されたクラス構造

求められた(式 17b)は，図 10 で示す Adaper 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，クラスがもつ構成子の識別名集合と構造上の役割をもとに命名するものとする．以後のデザインパターンでも同様とする．

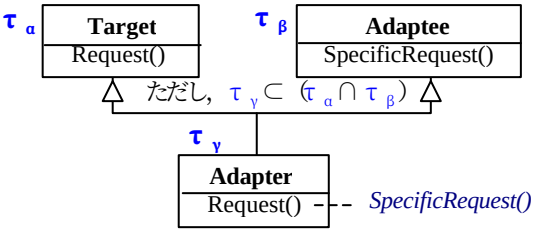

図 10 構成演算後の配置構造～Adapter パターン

Fig.10 The class structure extracted by applying the configuration operation that corresponds to the Adapter pattern

5.2 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 をも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たい．ここで，τ_β，τ_γ の代数和 $(\tau_\beta \cup \tau_\gamma)$ は，初期構成子集合の存在寿命 τ_α 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 18a)で表す．これにクラス構成演算を適用すると(式 18b)に変形できる．

$$\Psi_0 = \{\ \tau_\alpha\alpha,\ \tau_\beta\alpha^*,\ \tau_\gamma\alpha^*\ \}$$

ただし，$\tau_\alpha = (\tau_\beta \cup \tau_\gamma)$ (式 18a)

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

/ $\tau_\alpha = (\tau_\beta \cup \tau_\gamma)$であるから，$\tau_\alpha$に置換する */*

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

/ $\tau_\alpha[\alpha]$ と継承の基底項 $\tau_\alpha[\alpha]^*$ が構成演算の対象 (式 9)が適用できることから $\tau_\alpha[\alpha]$で集約化 */*

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

(4) 抽出されたクラス構造

(式 18b)は，図 11 で示す Bridge パターンがもつクラス構造に対応している．

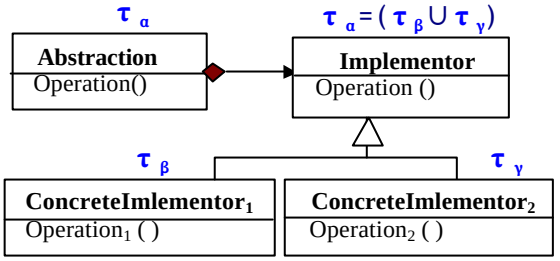

図 11 構成演算後の配置構造～Bridge パターン

Fig.11 The class structure extracted by applying the configuration operation that corresponds to the Bridge pattern

5.3 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 12 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$\mu_1..\mu_n$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 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もつ条件を満たしたクラス構造を求めたい．

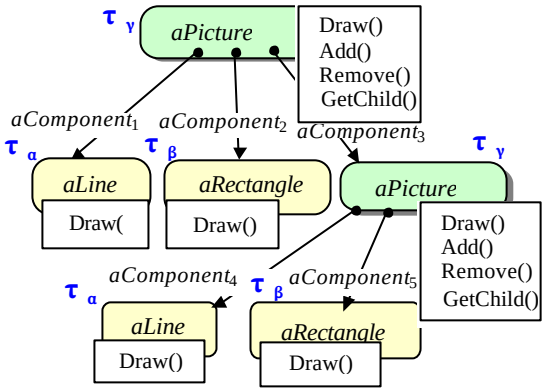

図 12 Composite の出発点となるインスタンス構造

Fig.12 The initial instance structure for the Composite pattern

(3) 構成演算の適用列

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

$\Psi_0 = \{\ \tau_\alpha\alpha,\ \tau_\beta\alpha,\ \underline{\Psi_0} \in \tau_\gamma\{\alpha, \mu_1..\mu_n\}\ \}$ (式 19a)

/* 共通する識別子αの項を括り出し継承構造化 */

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

/* 上位クラスを展開する．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を指す */

$= \tau_\alpha[\alpha] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma)[\alpha]$
$+ \tau_\beta[\alpha] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma)[\alpha]$
$+ [\underline{\Psi_0}] \diamond \tau_\gamma[\alpha, \mu_1..\mu_n] \triangle (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma)[\alpha]$ (式 19b)

(4) 抽出されたクラス構造

(式 19b)は，図 13 で示す Composite パターンがもつクラス構造に対応している．

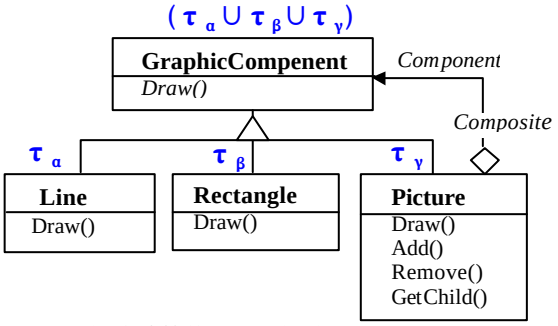

図 13 構成演算後の配置構造～Composite パターン

Fig.13 The class structure extracted by applying the configuration operation that corresponds to the Composite pattern

5.4 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 14 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．

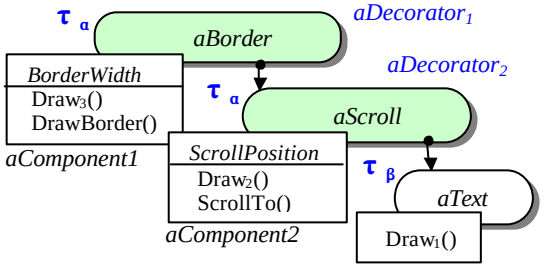

図 14 Decorator の出発点となるインスタンス構造

Fig.14 The initial instance structure for the Decorator pattern

インスタンス aTextView のもつ Draw()の存在寿命は，明らかに aScrollDecorator や aBorderDecorator がもつ存在寿命τ_αとは異なる存在寿命τ_βをもつことから，初期構成子集合は（式 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もつ条件を満たすクラス構造を求めたい．

(3) 構成演算の適用列

(式 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 20b)に変形できる．

$\Psi_0 = \{\ \tau_\beta\alpha,\ \tau_\alpha\{\alpha,\ \mu_1\} \in \tau_\alpha\{\alpha,\ \mu_2\}\ ,$
$\tau_\beta\alpha \in\ \tau_\alpha\{\alpha,\ \mu_1\}\ \}$ (式 20a)

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

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

/* 共通項$\{\tau_\alpha\{\alpha, \mu_1\},\ \tau_\alpha\{\alpha, \mu_2\}\}$を汎化し，

*(式 16)から，作成した上位クラス $\tau_\alpha[\alpha]$ で集約の基底項を置換する */*

$$
\begin{aligned}
\Rightarrow & \{\ \tau_\beta[\alpha], \\
& \quad (\tau_\alpha[\alpha, \mu_1] + \tau_\alpha[\alpha, \mu_2])\triangleleft\tau_\alpha[\alpha], \\
& \quad \tau_\alpha[\alpha, \mu_1]\diamond\tau_\alpha[\alpha],\ \tau_\beta[\alpha]\diamond\tau_\alpha[\alpha]\ \} \\
= & \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] + \\
& (\ (\tau_\alpha[\alpha, \mu_1] + \tau_\alpha[\alpha, \mu_2])\triangleleft\tau_\alpha[\alpha]) \\
& \qquad \triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] \\
& \quad + (\tau_\alpha[\alpha, \mu_1] + \tau_\beta[\alpha])\diamond\tau_\alpha[\alpha]
\end{aligned}
$$

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

$$
\begin{aligned}
= & \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] + \\
& ((\tau_\alpha[\alpha, \mu_1] + \tau_\alpha[\alpha, \mu_2])\triangleleft\tau_\alpha[\alpha]) \\
& \qquad \triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] \\
& \quad + (\tau_\alpha[\alpha] + \tau_\beta[\alpha])\diamond\tau_\alpha[\alpha] \\
= & \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] + \\
& ((\tau_\alpha[\alpha, \mu_1] + \tau_\alpha[\alpha, \mu_2])\triangleleft\tau_\alpha[\alpha]) \\
& \qquad \triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha] \\
& + (\tau_\alpha[\alpha] + \tau_\beta[\alpha])\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]\diamond\tau_\alpha[\alpha]
\end{aligned}
\qquad \text{(式 20b)}
$$

(4) 抽出されたクラス構造

(式 20b)は，図 15 で示す Decorator パターンがもつクラス構造に対応している．

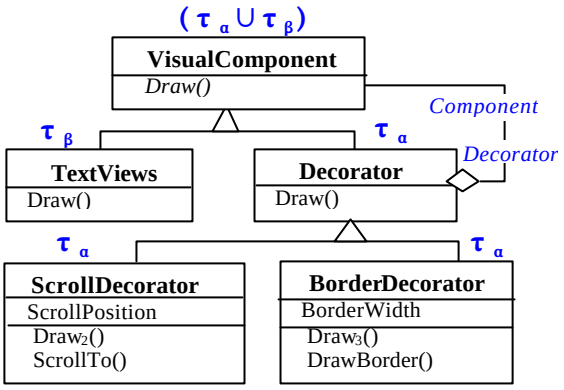

図 15 構成演算後の配置構造～Decorator パターン
Fig.15 The class structure extracted by applying the configuration operation that corresponds to the Decorator pattern

5.5 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 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 をもつ構成子は，短い存在寿命 τ_β をもつ構成子の代理（あるいはバックアップ）として機能させたい．ここで，短い存在寿命 τ_β をもつクラスは，長い存在寿命 τ_α をもつクラスの属性 image をそれと同じ存在寿命をもつメソッド 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 21a)にクラス構成演算を適用すると(式 21b)に変形できる．

$$
\begin{aligned}
& \Psi_0 = \{\ \tau_\alpha\{\alpha \mid \mu_1, \mu_2, \mu_3\}, \\
& \qquad \tau_\beta\{\beta \mid \mu_1, \mu_2, \mu_3\}\ \} \qquad \text{(式 21a)} \\
\Rightarrow & (\tau_\alpha[\alpha \mid \mu_1, \mu_2, \mu_3] + \\
& \quad \tau_\beta[\beta \mid \mu_1, \mu_2, \mu_3]) \\
& \triangleleft(\tau_\alpha\supset\tau_\beta)\ [\mu_1, \mu_2, \mu_3] \qquad \text{(式 21b)}
\end{aligned}
$$

(4) 抽出されたクラス構造

（式 21b）は，図 16 で示す Proxy パターンがもつクラス構造に対応する．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もつが，構成演算による方法では，メッセージ交換は，メソッドのも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．

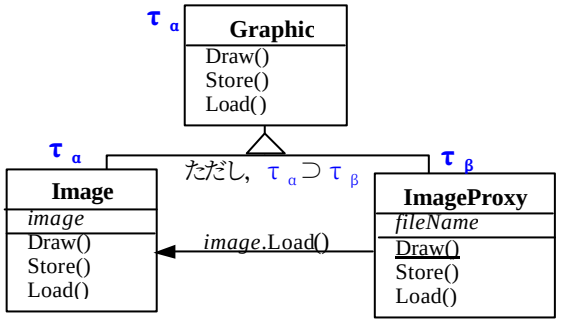

図 16 構成演算後の配置構造～Proxy パターン
Fig.16 The class structure extracted by applying the configuration operation that corresponds to the Proxy pattern

5.6 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もった新たな構成子集合 δ を介して行いたい．

(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もと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もつ分布関数の代数和である存在寿命をもったクラス$(\tau_\alpha \cup \tau_\beta \cup \tau_\gamma)[\delta]$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．

$$\Psi_0 = \{\tau_\alpha\{\alpha\}, \tau_\beta\{\beta\}, \tau_\gamma\{\gamma\}\} \Rightarrow (\tau_\alpha[\alpha] + \tau_\beta[\beta] + \tau_\gamma[\gamma]) \diamondsuit (\tau_\alpha \cup \tau_\beta \cup \tau_\gamma)[\delta] \quad \text{(式22)}$$

6．類似研究

(1)メタパターンに関して

本研究と類似する研究として，Preeが提唱しているメタパターンがある[9),10)]．メタパターンの考え方は，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，構成属性と呼ばれるメタ属性を与えることで，Templateクラスと Hookクラスの構成関係を決定してゆく考え方である．Templateクラスを固定して，Hook メソッドをオーバーライトし，適応性のあるクラス構造を構成するための各種のメタパターンを定めている．メタパターンの構成属性として次がある．

①Template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?

②Templateクラスと Hookクラスの間には継承関係があるか？

これらの構成属性にもと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もつ基本的な特徴（たとえば，存在寿命はその一つである）にさかのぼって，クラス構造の構成過程を掘り下げることはせず，現象論的に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．困難が伴うクラスそのものの抽出方法について触れていない．

ⅲ)メタパターンでは,クラスのも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，前者はクラスであるのに対して，後者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)Template メソッドと Hook メソッドとの依存関係と構成子の「独立キー」メタ特性の依存関係はほぼ等価である．

(2) デザインパターンの定式化に関して

デザインパターンの定式化に限ってみると，Temporal Logic of Actions[18)]を基盤に，ザインパターンがもつ振る舞いの定式化を試みた Mikkonen の研究[19)]がある．定式化の方法は，一定の役割をもっ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もつ関連(＝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も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変化は，考察の対象に含めていない．そのため，妥当性検証実験で用いたパターンも「構造に関するデザインパターン」の範囲に限定している． 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の関連へと視点を移してゆく点で共通点をもつ．

(3)その他

このほかに，振る舞いと実装を委譲によって分離して，デザインパターン間の関連を定める Zimmer の研究[20)]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．しかし，GoF のデザインパターンがもつ相互関連の分類（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一つのロードマップ的な役割を果たしている．

7．まとめと今後の展望

(1) 構成演算の妥当性検証

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方法に焦点を当て議論してきた．分析要素である構成子は，帳票や伝票，ユースケースシナリオから比較的容易に抽出することができることから，「ひらめき」に依存したクラス構造の抽出を，ソフトウェア場上での構成演算列の適用に置換することを試みた．さらに，構成演算の妥当性と可能性を検証するため，GoF の構造に関するデザインパターンがもつ構成子集合を想定して，それらに構成演算列を適用し，デザインパターンに対応するクラス構造が抽出できることを机上実験によって示した．その結果，正しいクラス構造の抽出問題が正しい識別名と存在寿命の抽出問題に還元できることを示した．

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は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（デザインパターンに限らず，ドメイン特有のアナリシスパターン [21] もそれゆえに価値をもつ）．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もと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ゆきたい．

(2) 要求仕様定義手法との関連

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 [22] がある）．

役割はクラスがもつメタ特性の一つとして定義できるが，クラス構造全体の拡張に伴って，役割が動的に再編成されることが生じる．これは役割が構成子集合を配置するときの制約として機能するメタオブジェクト（＝独自の存在寿命をもった分布関数）として捉えられることを示している．実際，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，役割に対して，時間に依存した「重視度」メタ特性を与えると，要求仕様に含まれる役割要素から，クラス構造の構成上の制約としての役割構造（=役割の重ね合わせ状態，および相互の関連）を抽出することが可能になる．構成子集合は，この役割構造から制約を受けながら，χ－τ空間に配置され，クラス構造を構成してゆくものとしてモデル化できる．

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，構成演算の適用を前提とする要求定義手法の研究へと発展が見込める．

8．あとがき

現在，構成子の洗い出しと構成演算にもとづくクラス構造の半自動的な抽出ツールの試作を進めている．試作ツールの詳細や構成演算の解の一意性，さらに広範なクラス構造への適用事例については，別稿で述べたい．

最後に本論をまとめるに当たって，内容の詳細な吟味にご協力いただくと同時に，貴重なアドバイスをいただいた神林 靖氏に感謝の意を表したい．

参 考 文 献

1) R.Wirfs-Brock, B.Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach," Proc of OOPSLA '89, ACM, pp. 71-75(1989).
2) I.Jacobson,G.Booch,J.Rumbaugh, "The Unified Software Development Process, "Addison-Wesley (1999)
3) 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)
4) 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)
5) D. Pascot, "DATARUN CONCEPT "CSA Research Pte.,(1996)
6) **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)
7) P. Coad, "Object-Oriented Patterns, "CACM, Vol. 35 No. 9, pp. 152-159 (1992)
8) Gamma,Helm,Johnson&Vissides, "Design Patterns: Elements of Reusable Object-Oriented Software , " Addison-Wesley (1995)
9) W.Pree, "MetaPatterns—A Means For Captureing the Essentials of Reusable Object-Oriented Desin, "Proc. Of ECOP pp.150-162 (1994)
10) W.Pree, " Design Patterns for Object-Oriented Software Development, "Addison-Wesley (1996)
11) D.Kling, "Metapatterns-An overview " www.idt.mdh.se/kurser/cd5130/msg/2002lp3/download/CD5130%20VT02%20Metapatterns.pdf
12) 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)
13) 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)
14) 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)
15) 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)
16) 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)
17) 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)
18) L.Lamport, "The temporal logic of actions, "ACM Trans. PL and Systems Vol.16. No.3 pp,872-923 (1994)
19) T.Mikkonen, "Formalizing Design Patterns, "Proc. of 20th ICSE pp.115-124(1998)
20) W.Zimmer, "Relationship Between Design Patterns, "Cop95 pp.345-364 (1995)
21) M.Fowler, "Analysis Pattern: Reusable Object Models, "Addison-Wesley (1995)
22) D.Riehle, "Describing and Composing Patterns Using Role Diagrams, "
http://www.ubilab.org/publications/print_versions/pdf/ubilab-woon-96.pdf