クラス構成演算によるデザインパターン導出実験

大木 幹雄 †

あらまし

ユースケースシナリオを用いたクラス構造の抽出では，分析要素（≒属性やメソッド）の識別名がもつ意味的な関連のみが分析の対象であり，分析要素がもつ存続時間(Lifetime)については何も考慮されていない．本論では，分析要素がもつ存続時間を積極的に活用して，正しいクラス構造を抽出する判断規則について述べる．分析要素はソフトウェア場と呼ぶ識別名と存続時間で張られる空間上の分布関数として捉えており，それらに幾つかのメタ特性や制約を付加して，クラス抽出の判断規則を分析要素に対する構成演算として定義する．本論の後半では，これらの構成演算を用いて，洗い出したインスタンス集合から，構造に関する幾つかのデザインパターンと同等なクラス構造を導出する．この結果は，構成演算が有効であることを示すと共に，構成演算の適用によってデザインパターンを用いずとも同等なクラス構造に到達できることを示している．

An Experiment of Design Pattern Derivation through Class Composite Operations

Mikio OHKI †

Abstract

When a novice analyst tries to extract class structures from given requirement specifications, he/she finds it is easier to extract software elements such as attributes, methods, and relationships at the primary stage and to bind them into classes than to extract the classes directly. This paper illustrates how to define the set of operations that are used to derive class structures by introducing the concepts of meta-attributes and constraints. The later part of this paper describes an experiment to validate the defined operations by deriving typical design patterns from software elements, and also describes the relationships between requirement specifications and the extracted software elements.

1. はじめに

オブジェクト指向ソフトウェア開発で最も重要で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]などが考案され，責任や利用シナリオの視点からクラスを抽出することを試みている．しかしこれらの方法とてクラス抽出の決定打ではなく，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．そのため，要求仕様から一定の手続きを踏むことでクラスを発見し，かつその正しさを検証できる何からの判断規則が必要になる．

一方，情報システム開発では，中核となる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，従来から，正しいエンティティを抽出する判断規則の研究が行われてきた．たとえば DATARUN[4]では，帳票や伝票に記載されたデータ項目から実体型(Entity Type)を抽出するため，PDG（Primary Data Generator），すなわち「データ項目の実現値を決定するきっかけ」を一つの判断規則として用いている．ある PDG によって，基本データ（≒他のデータから導出できないデータ）の実現値が同時に決定されるとき，それらの基本データの集合は，同一の実体型に属する属性候補とする判断規則である．その根拠は，実体型から一つの実体型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が同時に決定されることから，逆にPDG によって実現値が同時に定まるような基本データ集合は，実体型の属性候補となるからである．

筆者は，PDG の概念を一般化し，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 モデリングの判断規則を考案し，それが有効であるかを概念データモデリング演習授業の学生を対象にして比較実験を行った．その結果，判断規則を用いないに比較して，正しい ER モデルが導出される割合が統計的に有意な水準で向上することが実証された[5]．この比較実験の意義は，時間的な視点である「きっかけ」を基本データの分類・整

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

理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，『ひらめき」に頼らずに正しいエンティティが抽出できることを示した点にある．さらに多値度や状況数を用いると，集約，継承を含むエンティティ構造までも抽出できることを示した点にある．

本論は，比較実験で用いた判断規則をさらに発展させ，メソッドインタフェースを中心にしたクラス構造であっても正しく抽出できうることを示すものである．本論の前半では，判断規則に数学的な根拠を与えるために導入した概念と構成演算について述べる．後半では，これらの構成演算の有効性を実証するため，ユースケースシナリオ等によって洗い出されたインスタンス集合に構成演算を作用させ，代表的な『構造に関するデザインパターン」の幾つかが導き出せることを示す．

2. クラス分析過程のモデル化

クラス構造の分析の出発点は，CRC カードやユースケースシナリオを用いて，クラスや属性，メソッド等の候補を具体的に洗い出すことにある．洗い出されたクラスや属性，メソッド，洗い出しの方法には次の特徴がある．

(1) クラスや属性，メソッドは，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，場合によっては，概念的に同一のクラスが異なる識別名（用語）で認識される場合が多々ある．

(2) 洗い出しの手掛かりとして用いているものは，概念とメソッド（あるいは責任）の意味的な関係だけである．オブジェクト指向では，インスタンスはクラスによって生成されてから消滅するまでの存続時間（Lifetime）を必ずもつ．存続時間の同一性は，前述の ER モデリングにおける判断規則と同様，メソッドをクラスに割当てる手掛かりとなるが，用いられていない．

これらのあいまい性をモデル化し，存続時間をクラス抽出の手掛かりとするため，以下の概念を導入する．

2.1 ソフトウェア場の概念

ソフトウェア場とは，判断規則を演算として定義するために導入した概念である．すなわち，図1で示すとおり，具体的な属性やメソッド（以後，クラスの構成要素であることから「構成子」と呼ぶ）がまとまりを形成して，クラスとなる過程をモデル化するために導入した概念である．分析者が構成子間の意味的なつながりを判断してクラスを抽出する過程は，構成子が構成子同士に働く力の場によってまとまりを形成する過程として解釈する．

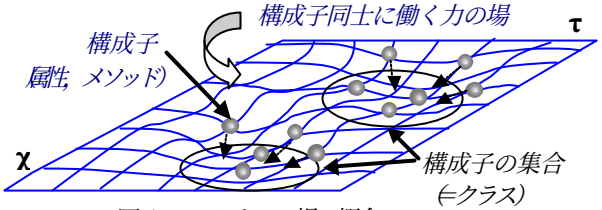

図1 ソフトウェア場の概念

2.2 分布関数としての構成子

洗い出された構成子は，識別名と存続時間をもつ．そこで，ソフトウェア場は識別名軸χと存続時間軸τによって張られる空間で記述されるものとする．また，分析過程で不確定な存在としての構成子は，χ，τ空間に確率的に分布する関数として表現する．議論を簡単にするために，構成子は分析によって識別名，存続時間が確定しているのとし，図2で示す矩形状の分布関数φで表現する．τ_1，τ_2 は構成子の生成，消滅時間を示し，φは構成子の存在確率を示す．φは存続時間内で1,それ以外は0の値をとるものとする．

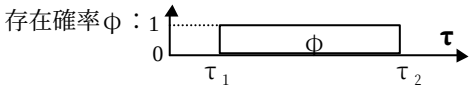

図2 τ軸方向の構成子分布関数の形状

構成子を分布関数としてとらえると，構成子を分類し束ねてクラスを抽出する操作の原動力，すなわち構成子間に働く力は,分布関数が重なり合う部分の比率として定義できる．するとクラスを抽出する操作は，『分布関数が完全に重なり合う構成子を選択して束ねる操作』として定義でき，構成士間に働く力の大きさによって，クラスを形成するか否かが機械的に判断できることになる．

3. 構成子のメタ特性と構成子間の制約

オブジェクト指向の基本的な特徴から，構成子に次のメタ特性をもたせる．

3.1 構成子がもつメタ特性

(1) 多値度υ

構成子のインスタンスが生成されるとき，同時に決定される個数を示す．構成子が属性のときは，実現値の数を意味し，構成子がメソッドのときは，同一インタフェース名で実装されるメソッドの数を意味する．

(2) 状況レベルν

クラスの継承レベルに対応する概念である．同じ存続時間をもつ構成子は，同じ状況レベルνに配置される．継承が深まるにつれ，高い継承レベルに配置される．

3.2 構成子間の制約

オブジェクト指向では，継承や関連を構成する際に働く制約が構成子毎に存在する．これらの制約を以下のとおりに整理する．

(1) 排他制約π

構成子を集めてクラスに格納するとき，同じ識別名で配置できる構成子の数を規定する制約である．排他度は，同じχ座標に配置できる構成子の数を示すものである．構成子が属性のときは，同じ識別名をもつ構成子は，0, 1以外の排他度をもつことはできない．すなわち，同じ識別名をもつ属性は，状況レベルがいかに異なっていても，1つ以上存在し得ない（public属性は，異なる状況レベルに重複して配置できるが，情報隠蔽の原則に反

することから，本論では除外して考える)．

構成子がメソッドのときは，同じ識別名をもつ構成子がクラス内に存在しても，状況レベル，すなわち継承レベルを変えれば複数個配置できる．

(2) 多値度制約υ

構成子の実現値が多値であるとき，単値構造の構成子と混在してクラス内に存在できないとするオブジェクトの基本的な特徴に根ざした制約である．制約を適用するときの優先度は，多値度制約は排他制約より低い．

4. 継承と集約の構成演算

前節で述べた構成子に付与した特性，および制約を用いて，クラスの集約と継承を構成する規則を形式的に定義すると以下のとおりになる．定義に先立って，表記規則中に現れる記号は，それぞれ次を意味するものとする．

① $\tau_\alpha\{\alpha, \beta\}$:存続時間が同一である構成子α，βを要素とする集合．存続時間を識別する記号としてτ_αを用いる．

② $\{\alpha^*\}$:多重度が1以上の構成子αの集合．

③ $(\tau_\alpha \cup \tau_\beta)$:存続時間の代数和．

④〔 〕 :個々の構成子を列挙した集合．

⑤ $\tau_\alpha[\alpha]$:存続時間の識別子記号がτ_αである構成子を束ねて構成したクラス．

⑥$[\beta] \triangle [\alpha]$: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．

⑦$[\beta] \diamond [\alpha]$: 構成子αをもつクラス[α]は構成子βをもつクラス[β]を集約要素としてもつ．

⑧$[\beta]^* \diamond [\alpha]$: 構成子αをもつクラス[α]は構成子βをもつクラス[β]と0..*の集約関係をもつ．

⑨$[\alpha]+[\beta]$：クラス構造は構成子αをもつクラス[α]と構成子βをもつクラス[β]から記述される．

4.1 クラス継承の構成演算

前節で述べた構成子間の排他制約πを用いると，(χ, τ)空間に配置した構成子の集合から，クラス継承を構成する次の演算が導ける．

構成演算①】 構成子の分布関数集合から，存続時間が異なるものの同一の識別子名αをもつ構成子を取り出し，それらの分布関数の代数和を上位クラスに配置して，継承を構成する演算である．次の変換規則として表わせる．

$$\left\{\tau_\beta\{\alpha,\beta\}, \tau_\gamma\{\alpha,\gamma\}\right\} \xrightarrow{\pi} (\tau_\beta[\beta]+\tau_\gamma[\gamma]) \triangle (\tau_\beta \cup \tau_\gamma)[\alpha]$$

【構成演算②】 構成演算1と同様であるが，上位クラスに抽象メソッドを配置し，下位クラスでメソッドの再定義する演算で，次の変換規則として表わせる．

$$\left\{\tau_\beta\{\alpha,\beta\}, \tau_\gamma\{\alpha,\gamma\}\right\} \xrightarrow{\pi} (\tau_\beta[\alpha,\beta]+\tau_\gamma[\alpha,\gamma]) \triangle (\tau_\beta \cup \tau_\gamma)[\alpha]$$

構成演算①,②のもつ意味は，図 3 で示すとおりである．

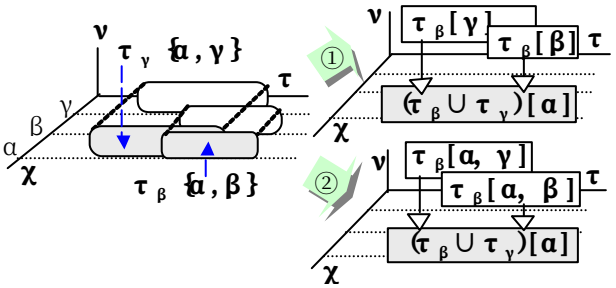

図 3 継承を構成する2つの演算

4.2 クラス集約の構成演算

存続時間が等しいが構成子集合の中で多重度が異なる構成子が混在したとき，多値度制約υから集約関係にある 2 つのクラスを構成する演算である．識別子依存(Composition 集約)関係にある関連がこれに相当し，図 4 で示す変換規則として表せる．

$$\left[\tau_\alpha\{\alpha, \beta^*\}\right] \xrightarrow{\upsilon} \tau_\alpha[\beta]^* \diamond \tau_\alpha[\alpha]$$

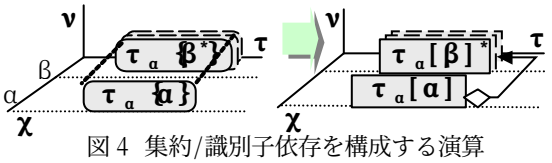

図 4 集約/識別子依存を構成する演算

5. デザインパターンによる検証

前節までに述べた構成演算を用いて，識別名と存続時間が洗い出されたメソッド集合から実際に正しいクラス構造が抽出されるかを実験する．事例として，構造に関する典型的なデザインパターンを取り上げる．

(1) Adapter パターンの導出

Adapterパターンとは，識別名と存続時間 (実装の決定される時間)が異なるインタフェース集合$\left[\tau_\alpha\{\alpha\}, \tau_\beta\{\beta\}\right]$があったとき，新たに，識別名は既存のインタフェース集合と一致するが，存続時間が異なるようなメソッド$\tau_\gamma\{\alpha,\beta\}$を追加したとき，形成されるクラス構造である．メソッド$\tau_\gamma\{\alpha,\beta\}$を追加後のメソッド集合を$\Psi_0$とすると，前述の構成演算②を用いて展開し，各構成関係の項を整理すると 式 2)が導かれる．(式 2)で記述されるメソッドの配置構造は，図 5 で示すとおりであり，Adapter パターンのもつクラス構造と同等な構造になる．

$$\begin{aligned}\Psi_0 &= \left[\tau_\alpha\{\alpha\}, \tau_\beta\{\beta\}, \tau_\gamma\{\alpha,\beta\}\right] \\ &\Rightarrow (\tau_\alpha[\alpha] + \tau_\gamma[\alpha,\beta]) \triangle (\tau_\alpha \cup \tau_\gamma)[\alpha] \\ &\quad + (\tau_\beta[\beta] + \tau_\gamma[\alpha,\beta]) \triangle (\tau_\beta \cup \tau_\gamma)[\beta] \\ &= \tau_\gamma[\alpha,\beta] \triangle ((\tau_\alpha \cup \tau_\gamma)[\alpha] + (\tau_\beta \cup \tau_\gamma)[\beta]) \\ &\quad + \tau_\alpha[\alpha] \triangle (\tau_\alpha \cup \tau_\gamma)[\alpha] + \tau_\beta[\beta] \triangle (\tau_\beta \cup \tau_\gamma)[\beta]\end{aligned} \quad \cdots\cdots\cdots\cdots \text{(式 2)}$$

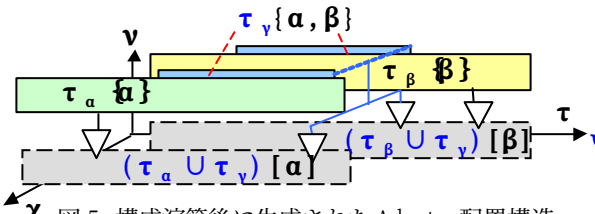

図 5　構成演算後に生成された Adapter 配置構造

(2) Bridge パターンの導出

Bridge パターンは,同じ識別名で異なる存続時間をもつメソッドが複数個洗い出されたとき，すなわち，複数の異なる実装を異なる時点で行うことが想定されるメソッド集合を洗い出したとき，形成されるクラス構造である．Adapter パターンと同様にして，（式 3)で示す配置構造が導き出される．これは図 6 で示す Bridge パターンのもつクラス構造と同等な構造になる．

$$\Psi_0 = \{\ \tau_\alpha\{\alpha\},\ \tau_\beta\{\alpha^*\},\ \tau_\gamma\{\alpha^*\}\ \}$$
$$\Rightarrow (\tau_\alpha[\alpha],\ (\tau_\beta[\alpha]+\tau_\gamma[\alpha])\triangleleft(\tau_\beta\cup\tau_\gamma)[\alpha]^*)$$
$$= (\tau_\beta[\alpha]+\tau_\gamma[\alpha])\triangleleft(\tau_\beta\cup\tau_\gamma)[\alpha]^*\diamond\tau_\alpha[\alpha]$$
$$\text{ただし}(\tau_\beta\cup\tau_\gamma)=\tau_\alpha \quad \cdots\cdots\cdots\cdots \text{(式 3)}$$

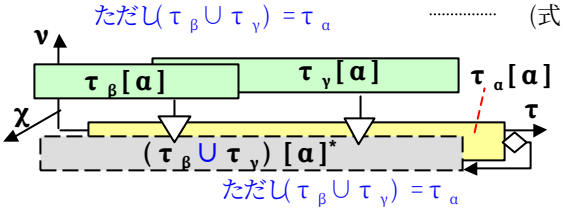

図 6　構成演算後に生成された Bridge 配置構造

(3) Composite パターンの導出

Composite パターンは，メタパターンの研究でも扱われるパターンであり，インスタンス構造を参考に洗い出されたメソッド集合が再帰構造をもつのが特徴である．前例と同様にメソッド集合に構成演算を作用させ，最終的に（式 4)で示す配置構造が導出される．これは，図 7 で示す Composite パターンのもつクラス構造と同等な構造をもつ．

$$\Psi_0 = \{\ \tau_\alpha\{\alpha\},\ \tau_\beta\{\alpha\},\ \Psi_0\diamond\tau_\gamma\{\alpha,\mu_1..\mu_n\}\}$$
$$= \{\ \tau_\alpha[\alpha],\ \tau_\beta[\alpha]\ ,\ \Psi_0\diamond\tau_\gamma[\alpha,\mu_1..\mu_n]\}$$
$$\triangleleft\ (\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\alpha]$$
$$\Rightarrow \tau_\alpha[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\alpha]$$
$$+\ \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\alpha]$$
$$+[\Psi_0]\diamond\tau_\gamma[\alpha,\mu_1..\mu_n]\triangleleft(\tau_\alpha\cup\tau_\beta\cup\tau_\gamma)[\alpha]$$
$$\cdots\cdots\cdots \text{(式 4)}$$

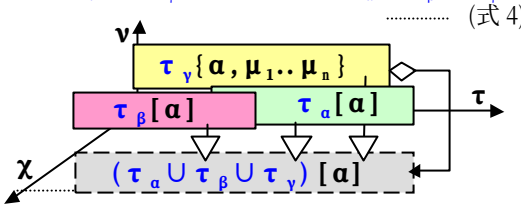

図 7　構成演算後に生成された Composite 配置構造

(4) Decorator パターンの導出

Decorator パターンも同様に，（式 5)，図 8 で示すとおり，同等なクラス構造を導くことができる．紙面の制約上，詳細は割愛する．

$$\Psi_0 = \{\ \tau_\beta\{\alpha\},\ \tau_\alpha\{\alpha,\mu_1\}\diamond\tau_\alpha\{\alpha,\mu_2\},$$
$$\tau_\beta\{\alpha\}\ \ \diamond\tau_\alpha\{\alpha,\mu_1\}\}$$
$$=\{\tau_\beta\{\alpha\},\ \tau_\alpha\{\alpha,\mu_1\}\diamond\{\tau_\alpha\{\alpha,\mu_1\},\tau_\alpha\{\alpha,\mu_2\}\}$$
$$,\ \tau_\beta\{\alpha\}\ \diamond\{\tau_\alpha\{\alpha,\mu_1\},\tau_\alpha\{\alpha,\mu_2\}\}\}$$
$$=\{\tau_\beta\{\alpha\},\{\tau_\alpha\{\alpha,\mu_1\},\ \tau_\alpha\{\alpha,\mu_2\}\}\triangleleft\underline{\tau_\alpha\{\alpha\}}\ ,$$
$$\tau_\alpha\{\alpha,\mu_1\}\diamond\underline{\tau_\alpha\{\alpha\}}\ ,\ \tau_\beta\{\alpha\}\diamond\underline{\tau_\alpha\{\alpha\}}\}$$
$$\Rightarrow\ \tau_\beta[\alpha]\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]\ +$$
$$((\tau_\alpha[\alpha,\mu_1]+\tau_\alpha[\alpha,\mu_2])\triangleleft\tau_\alpha[\alpha])$$
$$\triangleleft(\tau_\alpha\cup\tau_\beta)[\alpha]+\ (\tau_\alpha\cup\tau_\beta)[\alpha]\diamond\tau_\alpha[\alpha]$$
$$\cdots\cdots\cdots \text{式 5)}$$

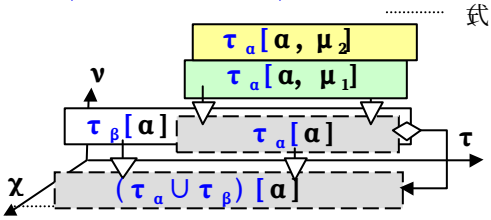

図 8　構成演算による変換後の配置構造

以上の実験から，メソッドの識別名と存続時間を正しく洗い出せば，これらのデザインパターンなしでも，構成演算により同等なクラス構造に到達できることが示せた．

6. おわりに

さらに広範囲のデザインパターンが構成演算によって導出できれば，分析者の主な作業はユースケースシナリオや帳票分析等から，存続時間も含めて，いかに構成子を洗い出すかの作業に力点が移ることになる．そのためには，要求仕様から正しく構成子を洗い出す手法や，構成子に対する役割や更新頻度等のメタ特性導入の検討が必要になる．今後の研究課題としたい．

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, "TheUnified Software Development Process, "Addison-Wesley (1999)

[3] C.Larman, "Applying UML and Patterns: an Introduction to Object-Oriented analysis and Design, " Prentics-Hall, 1998.

[4] D. Pascot, "DATARUN CONCEPT " CSA Research Pte.,(1996)

[5] 大木幹雄，秋山構平, "概念モデリングにおける判断規則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)