ソフトウェア工学 101－4
（1994. 11. 18）

オブジェクトの多面性表現のための クラスバージョンの導入

志村 秀人　上田 賀一

茨城大学工学部情報工学科

オブジェクトの多面性を表現するための機構としてクラスバージョンの概念を提案する．クラスはその一側面であるバージョンの集合で表現され，オブジェクトは，所属するクラスのバージョンを遷移することにより，対象相手や状況に応じた様々な振舞いを示すことが可能となる．また，バージョンの遷移条件を記述するにあたって，クラス間の関係による制約という形式で記述する．さらに，この適用範囲を限定する概念としてフィールドを導入する．オブジェクトはフィールドに登録することによってオブジェクトグループを構成する．同様に登録された遷移条件は，そのフィールド内のオブジェクトグループにのみ適用されることになる．これらにより，オブジェクトの多面性表現を容易にすることができる．

Class version for description of object with multi-aspect

Hideto Shimura　Yoshikazu Ueda

Department of Computer and Information Sciences,
Faculty of Engineering, Ibaraki University

This paper proposes a language with a concept "class version" as a mechanism to describe various aspects of object. This language represents a class as a set of version, and enables an object to behave depending on circumstances by transition of its class version. This transit condition is described in form of relational constraint between classes. And this language introduces "field" concept which sets applied bounds of transit condition. The transit conditions registered into field are applied to object group which is constructed by registering objects into the same field. Our language make it easy to represent various aspects of an object.

1 はじめに

対象世界のモデル化を行なう際にカプセル化や情報隠蔽，クラスの継承による再利用など，オブジェクト指向の有効性はこれまでに多々論じられ，確認されている．

一方で，オブジェクトに内在する多面的な性質やオブジェクトの性質の動的な変化など，従来のオブジェクト指向の概念では表現しにくい事象があることもまた事実である．

そのためオブジェクトの多面性を記述する新たな枠組が必要となり，研究が行なわれてきている．

特にオブジェクトの多面性について着目している研究に文献 [1][2] などがある．

また，場の概念についての研究では文献 [3] などが挙げられる．

本稿ではオブジェクトの多面性表現のための新たな機構としてクラスバージョンの概念を提案する．さらに，バージョンの遷移条件の記述としての関係記述と関係適用範囲を限定するフィールドの概念について述べる．

2 オブジェクトの多面的側面

オブジェクトは本来，状況に応じて異なる振舞いをする多面的な存在である．

ここでは次のような状況を考える．

(1) 相対するオブジェクト (メッセージのやりとりを行なうオブジェクト) に呼応して，振舞いが変わる．

(2) 自分の存在する場によって振舞いが変わる．

具体的な状況例として人の立場を取り上げることにする．

まず，(1) の例として，A 君 (オブジェクト A) は，B さん (オブジェクト B) と接する時は，側面 1 の顔で接する (図 1)．C さん (オブジェクト C) と接する時は，側面 3 の顔で接する (図 2)．

ここでオブジェクト B とオブジェクト C は，それぞれ異なるクラスに属している．オブジェクト A は相対するオブジェクト B,C に呼応してそれぞれに異なる振舞いを示すことになる．

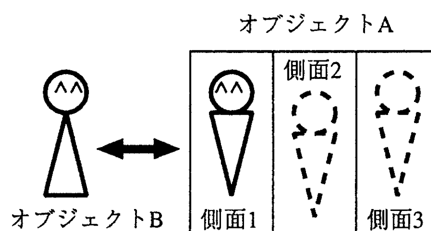

図 1: 状況に応じた振舞い (1)

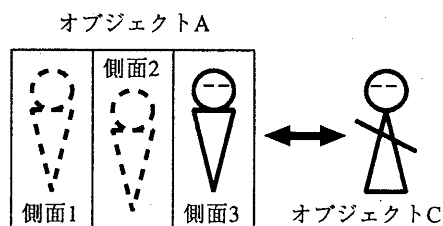

図 2: 状況に応じた振舞い (2)

次に，(2) の例として，D 氏 (オブジェクト D) は，E 大学の講師であるため E 大学内においては講師として振舞う (図 3)．またさらに，F 大学の学生でもあるため F 大学内においては学生として振舞う (図 4)．

ここでオブジェクト E とオブジェクト F は，それぞれ異なる場 (フィールド) であると考えられる．オブジェクト A は自らが存在する場 B,C に呼応してそれぞれに異なる振舞いを示すことになる．

このように，オブジェクトは状況に応じて異なる振舞いを示すことが求められる．

3 クラスバージョン

3.1 クラスバージョンの概念

前述のような状況に応じた振舞いを記述するために，クラス内に複数のバリエーションを許すことを考える．

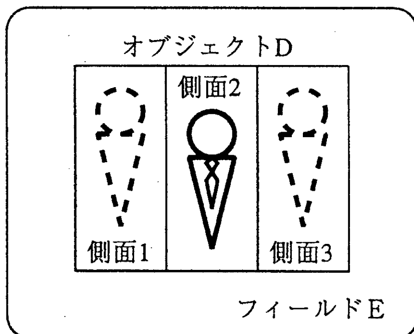

図 3: 状況に応じた振舞い (3)

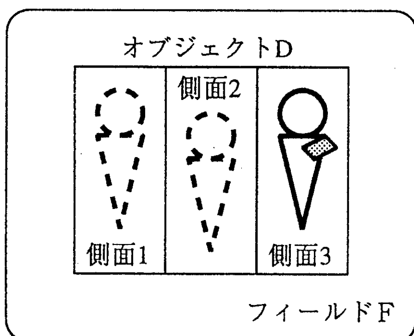

図 4: 状況に応じた振舞い (4)

クラスはバリエーションの集合で表され，そこから生成されたインスタンスは，所属するクラスのバリエーションを実行時に遷移することによって，相手や状況に応じて振舞うことが可能となる．

クラスのバリエーションは，継承を用いて差分のみを記述することにより，クラス階層のような構造を形成する．つまり，継承を時間的経過による発展と捉え，すなわちこれをリビジョンと考えると，クラス内においてリビジョンとバリエーションによるバージョン階層が形成されることになる．これをクラスバージョンと呼ぶ．クラスバージョンは各クラス毎に存在し，生成されたインスタンスはある時点においていずれか (または複数) のバージョンに属することとなる (図 5参照)．

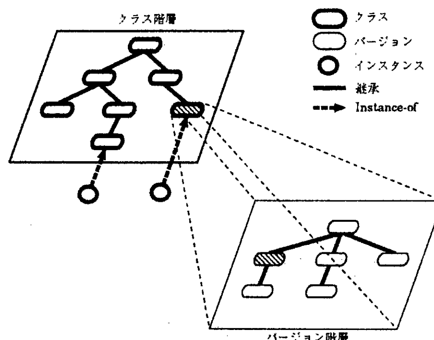

図 5: クラス階層とバージョン階層の対応

なお，インスタンス生成時において，どのバージョンで生成するかを明示的に指定しなくてもよい．この時には，デフォルトのバージョンで生成される．このデフォルトのバージョンをデフォルトバージョンと云い，各クラスに一つ設定される．通常，最後に定義されたバージョンがそのクラスのデフォルトバージョンとして設定されるが，これは変更可能である．

もし，バージョンをあくまで実装上のバリエーションと考えるなら，より抽象的なクラス階層の設計とそのクラスの内部，すなわち実装に結び付いたバージョンの設計とを分けて考えることが可能である．またその場合，クラスを利用する側にとってバージョン階層はカプセル化されるため，メソッドがどのように実装されているかについて考える必要がなくなる，という利点がある．

3.2 バージョンの継承の種類

バージョンの継承には，次の四通りの方法がある (図 6参照)．

(1) クラス越境バージョン継承
スーパクラス内のバージョンから直接継承してきた場合であり，スーパクラスの持つ性質の内，ある特定の性質しか継承されないことになる．

(2) 一般継承
クラス内にスーパバージョンを持たない．サブ

クラスの一バージョンとして，基本的にスーパクラスのすべての性質を継承する．

(3) 一般バージョン継承
クラス内のスーパバージョンからの継承．

(4) 同バージョンリンク
スーパクラス内のあるバージョンをリンクして同じ性質を持たせたもの．

3.3 バージョン探索経路

クラスバージョンの概念を導入した場合，実行時においてどのバージョンの属性およびメソッドが参照，実行されるのか，クラス階層に加えてバージョン階層を考慮した探索を行なう必要が出てくる．

以下に，バージョンの継承の種類に応じた探索方法を示す．

図6において，クラスBのインスタンスb(1), b(2), b(3), b(4) を考える．

b(1), b(2), b(3), b(4) がそれぞれ B_{v1}, B_{v2}, B_{v3}, B_{v4}のバージョンにおいてメソッドの起動が行なわれるとすると，b(1), b(2), b(3), b(4) の探索アルゴリズムはそれぞれ次のようになる．

b(1) クラス越境バージョン継承

(1) B_{v1}を探し，B_{v1}にあればそれを実行．なければ，

(2) B_{v1}はクラス A のバージョン A_{v2}から直接継承されているため A_{v2}に探しに行く．

b(2) 一般バージョン継承

(1) B_{v2}を探し，B_{v2}にあればそれを実行．なければ，

(2) B_{v1}を探し，B_{v1}にあればそれを実行．以下，b(1) に同じ．

b(3) 一般継承

(1) B_{v3}を探し，B_{v3}にあればそれを実行．なければ，

(2) B_{v3}はクラス A のいずれのバージョンからも直接継承されてはいないため，クラス A 全体から探す．

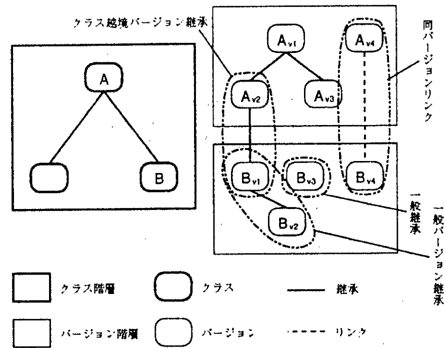

図 6: バージョンにおける継承関係および探索経路

b(4) 同バージョンリンク

(1) B_{v4}はクラス A のバージョン A_{v4}のリンクである．中身は A_{v4}に等しいが，あくまでクラス B のバージョンとして認識される．したがって，一般継承と同様の探索経路となる．

あるクラスはそのスーパクラスの特化であるため，基本的にスーパクラスのすべての特徴を引き継いでいる．故にスーパクラスへの探索が生じた場合，クラス越境バージョン継承のように継承元バージョンが特定される以外は，スーパクラスの全体から探索を行なう必要がある．

そこで，各クラスにそのスーパクラスの探索開始バージョンのリストを持たせるようにする．これにより，各クラス毎に動的に探索経路を変更することが可能となる．

4 オブジェクト間の関係

本モデルではオブジェクト間の関係によってバージョンの遷移条件を記述する．

これは，相対するオブジェクトによって自らの振舞いを変えることは，すなわち，自分と相手との関係によって自らのバージョンが変わることと考えられることに基づいている．

関係は以下のトリガを基本として，後述するフィールドによって起動される．

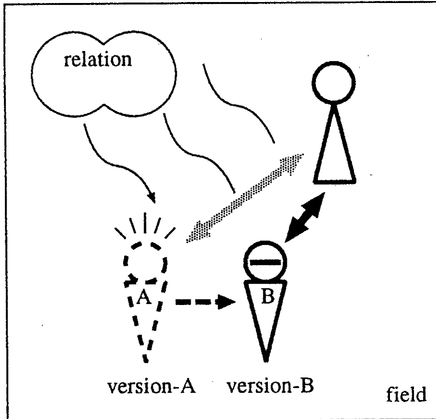

図 7: 関係によるバージョンの遷移

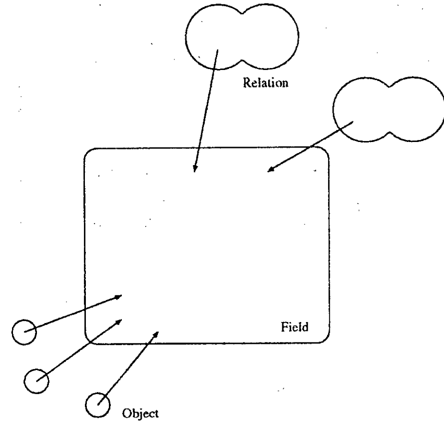

図 8: フィールドへの登録

バージョン遷移トリガ：

フィールド内のオブジェクトがバージョンを遷移した時に評価される．

メッセージトリガ：

フィールド内のオブジェクト間でメッセージのやりとりが行なわれた時に評価される．

フィールドトリガ：

フィールドの情報が変化した時に評価される (例えば新たにオブジェクトが登録されたなど)．

5 フィールド

5.1 フィールドの役割

局所的な関係を記述するには，関係の及ぶ範囲を限定する機構が必要になってくる．本モデルではこの機構として場の概念すなわちフィールドを設ける．

オブジェクトはフィールドに登録されることにより，オブジェクトグループを形成する．同様に関係を登録することにより，関係の及ぶ範囲がそのフィールドに登録されたオブジェクトのみに限定される (図8参照)．

5.2 フィールドにおける関係の適用

まず，フィールド上でのオブジェクト間のメッセージ送信を考える (図9参照)．このメッセージ送信をトリガとして適用される関係の例を次に挙げる．

図9において，フィールド f にオブジェクト a とオブジェクト b，関係 r が登録されている．この時，オブジェクト a からオブジェクト b へメッセージm が送信されたとする．このメッセージ送信がトリガとなり，関係 r が評価されオブジェクト b はバージョンを遷移する．

フィールドは登録されたオブジェクトと関係のリストを持っており，まず，フィールドに登録された時点でオブジェクトの各メソッドに:around メソッド[1]が定義される．

オブジェクト a からメッセージmが送信された時，それが評価される前，すなわちオブジェクト b のメソッドが実行される前に，総称的ディスパッチにより:around メソッドが起動される．このメソッドは，オブジェクト a，b 間で適用可能な関係を評価し，その内容に応じて，オブジェクト b のバージョンが遷移したり，スロットの値を変更したりした後，実際にメッセージが送信される．場合によっては，メッセージが送信されない場合もある．

[1]この:around メソッドは各 (基本) メソッドに先駆けて実行され，基本メソッドの実行制御を行なうことができる (文献 [5] 参考)

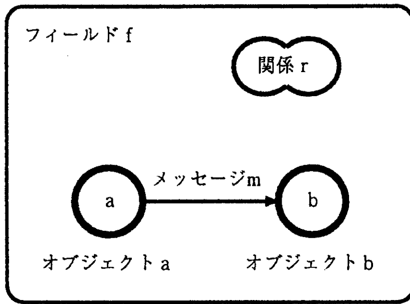

図 9: フィールド上での関係が適用される
メッセージ送信

このようにフィールド内における関係の適用により，バージョンの遷移や制約が実現される．

6 FBDL : Flexible Behavior Description Language

現在，前述のクラスバージョン，関係，フィールドの概念を採り入れたプログラミング言語FBDLのプロトタイプ作成を行なっている．FBDLはCLOS上にその拡張として構築される．

表6にFBDLの関数の一部を示す．

6.1 記述例

「人」,「子」,「親」をモデル化する場合を考える (図10参照).

(a) クラス「子」とクラス「親」をクラス「人」のサブクラスとしてモデル化する．
「子」のインスタンスとして生成されたオブジェクトは「親」にはなり得えない．

(b) 「人」の属性として「子」と「親」を考える．「子」か「親」かで振舞いを替えるようをメソッド内部に記述する他なく，メソッドは複雑なものとなる．

(c) 「人」をクラスとし，そのバージョンとして[人]を作る．さらに継承させてバージョン[親][子]を作る．

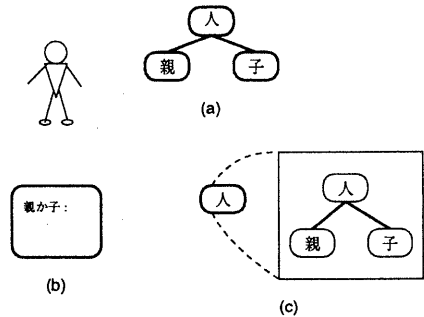

図 10: 人-親-子モデル例

クラス「人」のインスタンスは，内部状態または相手に応じて「親」または「子」または「人」のバージョンに切り替わり，それぞれの振舞いを示すことが可能となる．

バージョンを用いたFBDLによる(c)の記述例

```
;;; human クラスの human バージョン
(defclassv (human human) ()
  ((name    :accessor name
            :initarg :name)
   (sex     :accessor sex
            :initarg :sex)
   (age     :accessor age
            :initarg :age))

;;; human クラスの parent バージョン
;;; (human human) を一般バージョン継承
(defclassv (human parent) ((human human))
  ((child  :accessor child
           :initarg child)))

;;; human クラスの child バージョン
;;; (human child) を一般バージョン継承
(defclassv (human child) ((human human))
  ((father :accessor father
           :initarg :father)
   (mother :accessor mother
           :initarg :mother)))
```

表 1: FBDL 関数 (一部)

関数 (マクロ) 名	構文	意味
defclassv	(defclassv (*class-name version-name*) *slot-spec class-option*)	クラスバージョン定義
defmethodv	(defmethodv *method-name specialized-lambda-list* {*decl* | *doc*}* {form}*)	バージョンメソッド定義
relation	(relation *rel-name specialized-lambda-list* {((*trigger* '(*prog*) break?)...)}	関係生成
field	(field *field-name*)	フィールド生成
chversion	(chversion *object new-version*) (chversion *object-list new-version*)	バージョン変更

まずフィールド family を作り，親子におけるバージョン遷移関係を登録する.

```
;;; 親 -> 子
(defmethodv talk
  ((p human parent) (c human child))
  (print "My kiddy."))

;;; 子 -> 親
(defmethodv talk
  ((c human child) (p human parent))
  (print "Mam."))

;;;h は p の子供?
(defmethodv is-her-child
  ((h human) (p human parent))
  (eq h (child p)))

;;;h は c の母親?
(defmethodv is-his-mother
  ((h human) (c human child))
  (eq h (mother c)))

;;; フィールド family
(field family)

;;; 関係 親-親
(relation talk-in-family
  ((p1 human parent) (p2 human parent))
  ((mes-trigger
    '(cond ((is-her-child p1 p2)
            (chversion p2 'child))
           ((is-her-child p2 p1)
            (chversion p1 'child))))))
;;; 関係 子-子
(relation talk-in-family
  ((c1 human child) (c2 human child))
  ((mes-trigger
    '(cond ((is-his-mother c1 c2)
            (chversion c1 'parent))
           ((is-his-mother c2 c1)
            (chversion c2 'parent))))))

;;; 関係 talk-in-family の登録
(enter-relation family 'talk-in-family)
```

インスタンスを生成し family に登録する.

```
;;; 母
(setf kiku
  (make-instanse 'human
                 :version 'parent))

;;; 自分
(setf hanako
  (make-instanse 'human
                 :version 'child
                 :mother kiku))
;;; 子供
```

```
(setf taro
  (make-instanse 'human
                 :version 'child
                 :mother hanako))

;;;family field へ登録
(enter-object family '(kiku hanako taro))
```

ここでhanakoは，taroに対して［母］に，kikuに対して［子］に，バージョン遷移が起こり，次のように表示される．

```
> (talk hanako taro)
"My kiddy."
> (talk hanako kiku)
"Mam."
```

これは，非常に簡単な例だが相対オブジェクトに対する振舞いの変化を容易に記述できることがわかる．

7 まとめ

本稿では，オブジェクトに内在する概念的側面を表現し得る機構として，クラスバージョンの概念を提案した．オブジェクトは，所属するクラスのバージョンを遷移することにより，対象相手や状況に応じた様々な振舞いを示すことが可能となる．

また，バージョンの遷移条件を記述するにあたってクラス間の関係による制約という形式でこれらを記述する．この関係により，クラス間にまたがる制約，バージョン遷移条件が宣言的に記述可能となり，追加，削除，変更が容易になる．

さらに，関係の範囲を限定する概念としてフィールドを導入した．オブジェクトはフィールドに登録することによってグループを形成する．同様に関係をフィールドに登録することによって，そのフィールド内に登録されたオブジェクトグループにのみ，適用が行なえることになる．

そして，これらの概念を採り入れたプログラミング言語FBDLを提示し，簡単な例を挙げ，オブジェクトの多面性が容易に記述可能であることを示した．

8 今後の課題

現在，FBDLのプロトタイプの作成を行なっている．特に今後，制約に関する部分を強化する予定である．また並列，分散についても今後考慮する必要があると思われる．

参考文献

[1] 塚田 晴史, 杉村 利明：Multiple Objectの提唱, 情報処理学会記号処理研究会, 58-4, 1990.

[2] 塚田 晴史, 杉村 利明：MACモデル:複数観点からの分類が可能なオブジェクト, コンピュータソフトウェア Vol.11, No.5, pp.44-57(400-413), Sep,1994.

[3] 西尾 郁彦, 渡辺 豊英, 杉江 昇：オブジェクトと場に基づいた協調的プログラム言語, 情報処理学会論文誌, Vol.34, No.12, Dec, 1993.

[4] Bing Liu, Yuen-Wah Ku : ConstraintLisp: An Object-Oriented Constraint Programming Language, ACM SIGPLAN Notices, Vol.27, No.11, Nov, 1992.

[5] S.E. キーン：COMMON LISP オブジェクト指向 (CLOS), 1991, ドッパン.