ソフトウェア工学 84－1
（1992. 3. 3）

# 要求分析段階のモデル表現技法

片山 佳則

（株）富士通研究所
国際情報社会科学研究所

ソフトウェア開発環境に対するオブジェクト指向概念の導入を様々な局面ごとに検討している．ここでは，要求分析段階のモデル構築に対するオブジェクト指向概念導入の問題点を取り上げ，記述されるモデルの理解性と変更／改良性に焦点を当て，モデルの定義と実現の区別にポイントをおいたモデル表現方法を示す．モデル定義では，オブジェクトの関係に重点をおき，特に階層関係を明確に定義するための改良を行う．モデルの実現では，これまでのオブジェクト指向モデルに対して具現指向と呼ぶモデルの実現方法を示し，２つの実現方法を融合して要求分析段階のモデル構築を行なうことを提案する．

# An Object-Oriented Modeling Method for Requirement Analysis

Yoshinori KATAYAMA

FUJITSU LABORATORIES LTD.

140 Miyamoto, Numazu-Shi, Shizuoka, 410-03 Japan

The purpose of this paper is to propose an approach to a modeling method for requirement analysis level. Object-Oriented Methodology provides a very powerful paradigm for software composition and reuse. They can be useful for modeling and simulation. However, several problems arise along with object-oriented concept. We attacked these problems and propose a frame work for modeling technique.

In this paper, we improve on description as an inheritance relationship, and enlarge embodiment representation for part's relation.

## 1．はじめに

ソフトウェアの設計方法論として，SA/SD, JSDから，オブジェクト指向分析法等様々なものがある．対象としている開発領域のモデルを作成する方法は，ソフトウェア開発の各過程に対して，多数提案されている．その中でオブジェクト指向設計技法に注目しても，Boochのオブジェクト指向開発[1]の改良や拡張だけでなく，様々な方法が提案／考察されている．本稿では，オブジェクト指向概念を導入し，要求分析過程のモデルを作成することを目指している．ここでは，モデル作成を進める上で，現状のオブジェクト指向概念の導入方法において，モデル作成者が直面するいくつかの問題を取り上げ，それらに対処できるようにオブジェクト指向概念との結び付きを考え，モデルを作成する方法を提案する．

要求分析過程においてモデル作成を進めるシステムに必要とされる様々な性質の中では，モデルの理解性及びモデルの改良/変更/拡張性が，開発領域側からも後のプログラム実現側からも重要になる．そこで，理解性と変更，改良性に焦点を当て，オブジェクト指向概念と関連づけた新たなモデル定義およびモデルの実現方法を示す．

本稿では，要求分析過程におけるモデル表現技術に注目しているため，モデルとは，一般のソフトウェア開発過程における概念モデルに対応する．

## 2．モデル構築アプローチ

要求分析段階でのモデル構築は，ソフトウェア開発過程に組み込まれる初期場面であり，そのモデルの記述内容には次の2側面がある．

(1) 要求対象が何であるかを定義することにポイントを置く側面

(2) 定義の実現内容や実際のプログラムへの変換過程にポイントを置く側面

しかし，現在のモデル構築技法は，結果としてのプログラムの生産性や効率に重点が置かれているため，(1)のモデル定義でなく，その実現内容や効率的なプログラムへの変換側である(2)に高い比重が置かれている．このことは，ソフトウェア開発過程における，様々な場面で再利用を進めるための課題として，どのような情報をどのような形で残すべきかが注目されていることからも言える．これは，モデル構築の考え方が，実際のプログラムへの変換のための方法として解釈され，結果的に実現されたプログラムの情報だけが残される形になっていることを示している．プログラムの実現は，プログラミング方法論だけでなく問題領域側の概念にその対象が広げられてきている．要求分析段階でのモデル構築には，モデル定義である(1)にポイントを置いたモデル表現方法を確立させる必要がある．オブジェクト指向概念は，定義とその実現内容を区別してモデル構築を進められるため，この方向に則したものといえる．

## 3．要求分析段階でのモデル構築の問題

対象のモデルを素直に扱うのに適しているパラダイムとして，オブジェクト指向概念を捕え，様々な研究が進んでいる．それらの分野の代表的なものが，プログラミングやデータベースである．ソフトウェア開発では，オブジェクト指向概念が設計技法として注目されている．

オブジェクト指向概念には，抽象化や情報隠蔽など現実レベルでも必要とし，再利用を目指すには欠かせないキーワードが含まれている．このためソフトウェア開発においては，必然的に注目されるべき概念である．

オブジェクト指向概念では，対象を素直に自然に扱えることがポイントであることから，前節の(1)のモデル定義はより強調できる．これを踏まえた上で，モデル構築として問題となるポイントを把握することにより，ソフトウェア開発過程におけるオブジェクト指向概念の効果を引き出せる．

ここでは，モデル構築の問題を，新しくモデル構築を行なう場合と既に構築されたモデルを利用する場合に分けて捕えている．

### (a) 構築されたモデルの理解問題

構築されたモデルを利用するために受け継がれる情報として一般には構成された個々のクラス表現及びクラス構造（これらをクラス情報と呼ぶ）が与えられる．既に表現されているクラス情報の作成には，抽象化や情報隠蔽を実現するために様々な労力が費やされている．この主なものは，機能の分割方法である．

新たなモデル構築時には，既に構造的に表現されているクラス情報での機能の分割方法を理解しなければならない．その上で，その分割方法に従って効率的に再利用を実現する．これを困難にする原因は，情報として与えられるクラスが構造的に表現されているが，その構造の利用方法が不明

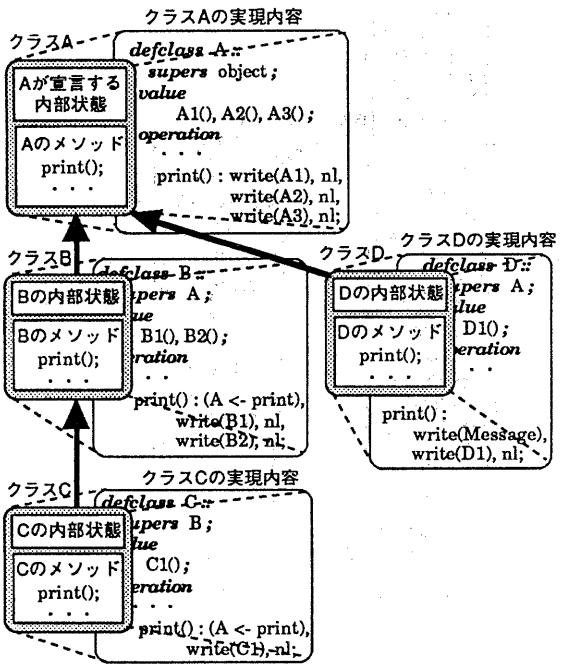



図1　クラス階層の定義情報と実現内容

確であることである．特に，現在採用されている関係構造により機能の効率的実現が進められる．しかし，それらの機能を理解するには，その機能の実現内容を詳しく分析しなければならない．

例えば，図1のようにクラス情報が実現されている場合，一般には継承やポリモルフィズムにより機能が分割され効率良くモデル構築されていると理解される．しかし，printメソッドを見ても，これらのクラスの間の結び付きをどのように利用してメカニズムが実現されているかを知るためには，各モジュールの定義情報だけでなく各メソッドの実現内容をすべて把握しなければならない．継承関係としてモジュール的に分割された機能も，何をどのように分担し，必要なメカニズムを実現しているかを把握する情報はすべて実際の実現内容を解釈しなければならない．これは，継承やポリモルフィズムを用いてモデル構築を進めた場合，前節の(2)に対する効果が記述効率という形で得られるが，それらを理解するためのモデル定義情報が不十分なことを示している．

実際には，どのような意図で各機能が分割され，クラス構造が決定されたかを詳細な実現情報を参照しなくても把握できること，さらには決断内容を柔軟に変更してモデルの動的な変更が行なえることが，クラス情報として表されているモデルの理解を容易にし利用を促進させることになる．

特に要求分析段階では詳細なモデルの実現方法にまで立ち入らないで，モデル定義の情報で理解できるように定義内容の充実を進めることが必要である．

(b) 抽象的なクラス表現とモデル構築の問題

オブジェクト指向設計技法としては，オブジェクトを要求内容における名詞から導く[1]あるいは必要とされている操作の分析から解析的に導く[2]などいくつかの方法が示されている．これらのモデル構築は，モデル定義と実現を区別して扱っていない．また，抽象的なクラスを定義しながらモデル構築を進める点は，モデル定義をあいまいにし，現実の問題領域における要求分析段階では大きな負荷となる．

対象としている問題領域における機能の分割方法と分担方法が明確に分析されていれば，実現内容をベースにモデル定義としての構造を決定できる．しかし，モデル定義に関する構造には対象問題領域における概念構造も考慮しなければならない．この両方の構造が反映できるようにモデルの定義内容を充実させる必要がある．

また，抽象的なクラス表現を実現するために，開発領域における機能の分割方法や分担方法を分析すること自体は，モデル構築として重要な過程でもある．

従って，クラス情報としての抽象的表現を考慮してモデル構築を進める形だけでなく，モデル構築の個々の要素であるモデル定義とその実現内容の記述の中から抽象的なクラス表現の実現に発展させる形態が必要である．これにより，モデル構築の結果としてクラス構造を組み立てる．

クラス情報は，モデル定義とその実現の結果である．モデル構築過程において，すでに記述されたクラス情報で用いている抽象化と異なる抽象化の立場を取るモデルの構築には，クラス情報の再検討が必要になる．これらを判断するためにも，実現内容と離れてモデルを理解できるためのモデル定義情報が重要になる．これは，記述されたモデルの順応性につながる．

モデル構築の側面として，ここに示したいくつかの問題点を考慮することで，定義と実現を区別したモデル表現方法が得られる．これが記述されたモデルの理解性と変更，改良性を向上したモデル表現方法となり，オブジェクト指向概念を導入

した要求分析におけるモデル表現のベースになる．

## 4．新たなモデル構築方法

オブジェクト指向概念におけるモデル表現は，クラスと継承階層による表現（クラス指向と呼ぶ）が一般的である．ここでは，2節に示したモデル構築の2側面を踏まえて，前節の要求分析段階のモデル構築問題に対処するために，モデルの記述形態に関する改良と記述されたモデルの動作メカニズムの拡張を示す．

具体的には，オブジェクト間の関係を明確にさせるモデル定義と，個々の実体を優先させ，その記述内容を役立たせるための新たなメカニズムを導入（具現指向と呼ぶ）する．具現指向は，抽象的なクラス表現の前段階となるモデル構築として捕えられる．

これまでにも，関係を規定した特殊な意味でインスタンスを用いてソフトウェア開発環境を提案しているものもある[3]．また，デリゲーションベース言語Self[4),5)]は，具現指向のみを強く支持したモデル表現方法といえる．

ここでのモデル表現技法では，オブジェクトの関係を重視し理解性を向上させるためにモデル定義を改良し，クラス指向＋具現指向により，互いに補完関係を保ってモデルの実現を行なう．

要求分析の段階では，クラス指向も具現指向もどちらも対象としている問題領域の分析に重点があり，概念単位を検討する段階である．従って，計算機側になるプログラム技法やプログラムとしての問題解決方法とは異なる次元である．特にこの違いにポイントを置いた新たなプログラミングパラダイムの提唱も進められている[6]．

### 4．1　継承関係におけるモデル定義の改良

モデル定義において，オブジェクトの継承階層を明確にさせるために図2に示す改良を行なう．

この改良では，これまでモデルの実現内容に組み込む形で表現されていた関係を，モデルの実現内容から分離してモデル定義として記述する．これにより，継承を用いた機能の利用関係を明確にし，モデル定義レベルで理解性の向上を図る．

図2(b)では，クラスBのprintメソッドがクラスAのprintメソッドを継承することを示し，クラスCではクラスBのprintメソッドを継承することをモデル定義において明示している．これらは，各モデルが持っているメソッドに関して継承の動作を

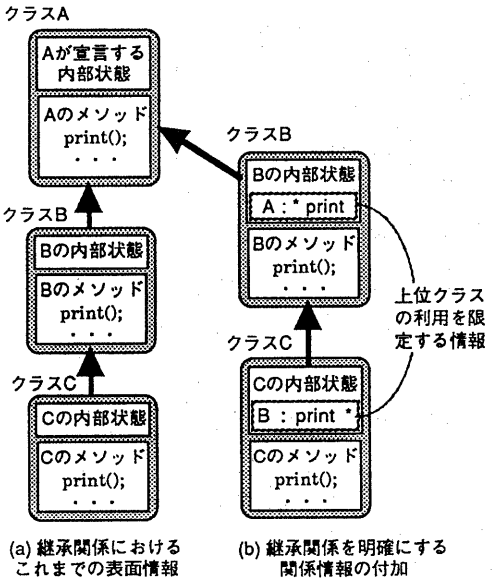



図2　関係表現における継承の明確化

各モデル定義で規定することである．部分的メソッドの協調利用は，階層関係としての定義でなく，この実質的関係定義で実現する．

これまでの階層関係では，サブクラスの解釈は上位クラスから下位に向けて，その実現内容を詳しく見ることで行なっていた．このモデル定義の改良により実現内容に立ちいらずに，下位側から主体的にサブクラスを解釈できるようになる．

### 4．2　クラス指向のモデルの実現

これまでのクラス指向では，モデル構築を進める段階で以下のことに意識が集中される．

- ○複数の対象をスコープに入れる：共通性を捕える，集合の要素的感覚
- ○常に共有性の意識を持つ：振る舞いや状態を抽象的に捕える．
- ○周囲の表現方法からの制約：これまでの細分化，概念モジュールの分割方法
- ○直接持つべき動作内容：実際に扱う状態や動作だけを考慮するのでなく，概念的関連を考慮した機能分割

これらを意識しながら対象問題領域を分析し，クラス表現や階層関係を実現する．モジュール的側面を強調すると，次の2局面により捕えられる．

(1) 内包的：対象領域で扱うものを的確に表す
(2) 外延的：別の領域からも扱えるように抽象化／構造化する

前記の4つの項目は，クラス指向におけるモデ

ルの実現では，この2局面を常に意識している必要があることを示している．このためには，問題領域に強く依存したタームを用いている所と，問題領域に依存しないタームの利用部分が適切に区別されているなど，十分に対象問題領域の抽象化の分析が進められている必要がある．

モデル構築におけるこのような分析の必要性は，新規の場合に限らずモデル作成者に対する多大な負荷である．これらの分析の負荷を縮小するには，このような抽象化を前提にしないで部分的にでも具体的対象を捕まえてデフォルト的振る舞いを表現できるようなモデル実現方法の導入が必要になる．これが具現指向である．具現指向は特に (1) の内包的局面に重点をおいた対象問題領域の分析を行なうことである．

問題領域における対象の局所部分をモデルとして実現する場合や，機能分解された局所部品のモデルの実現を行なう場合に，(2) の外延的局面を考慮するには，対象としている実現内容の具体的な情報だけでなく，より広範な知識が必要となる．この抽象化は，局所部分のモデル構築に対して思考のベクトルを変えることになり，モデル構築を困難にする．

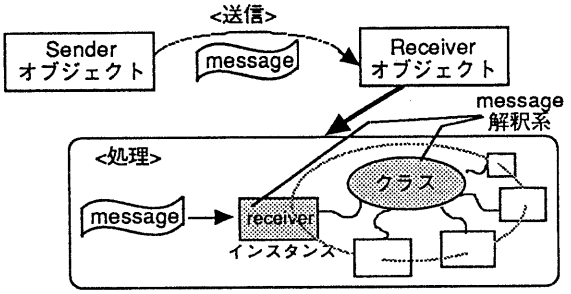
### 4．3　具現指向のモデルの実現

具現指向によるモデルの実現は，必要なオブジェクトをプロトタイプとして表現し，請け負う形で処理を進める方法である．クラス指向のように，インスタンスを捕えながら常に抽象レベルのクラスを表現し，処理はメッセージを受け取った側の内部表現に任かす方法とは異なる．

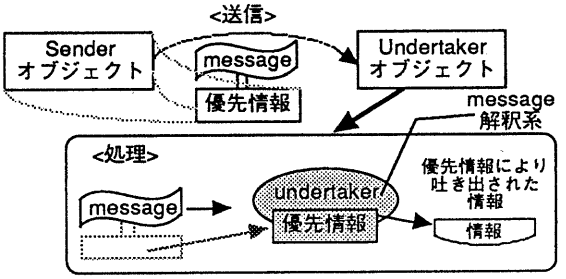
具現指向は，抽象的なクラスとしてのモデル構築を省く．これにより，抽象的クラスの定義と利用情報としての問題の一部を排除する(4.5節参照)．

これまでにも，対象をプロトタイプとして捕え，差分のある対象を扱う場合に複製を作成する方法[4]が提案されている．具現指向は，この考え方をベースにして簡潔さと順応性のあるデリゲーション[7]の改良形式で，規定された処理を請け負って実行するオブジェクト(undertaker)を実現する方法である．デリゲーションと一般のメッセージ送信とは，情報の共有に関して大きな違いがある．

通常のクラス指向においては，図3 (a)に示すように，メッセージ送信先を指定し，パラメータとともに主導権を受け渡す．メッセージを受け取ったオブジェクトは，自分の内部表現及び関係するクラス表現を参照しながら処理を実行する．




(a) 通常のメッセージ送信とその処理




(b) 具現指向におけるメッセージ送信とその処理

図3　具現指向メカニズムの概念図

具現指向におけるメッセージ送信は，常に送信する側の情報が付加される．そのメッセージの処理は図3 (b)に示すように付加される情報を優先した上でメッセージを受け取ったundertakerオブジェクト自身が解釈し実行する．

デリゲーションと具現指向およびクラス指向の違いを，疑似変数selfの扱いと内部状態の遺伝方式の観点でまとめた結果を表1に示す．

表1　メカニズムの違い

| | 疑似変数self にbindされる | 優先される 内部状態 |
|---|---|---|
| デリゲーション | 元のdelegetee object | delegetee object の内部状態 |
| 具現指向 | receiver object | sender object の内部状態 |
| クラス指向 | receiver object | receiver object の内部状態 |

通常，selfなどの疑似変数には，具現指向やクラス指向のようにメッセージのreceiverオブジェクトが代入される．デリゲーションとしてのメッセージ送信では，receiverの変わりに元のdelegeteeが用いられる[8]．疑似変数の扱いに関して各メカニズムの分類は（<デリゲーション>, <具現指向,クラス指向>）となる．この疑似変数の扱いの違い

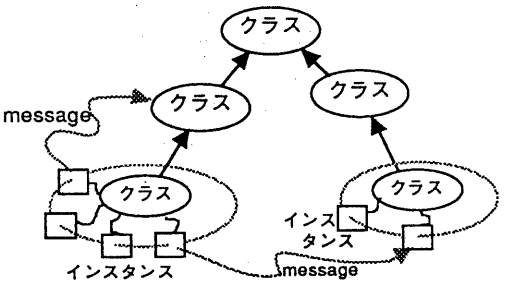
がメッセージ処理の際に優先する情報の違いとして現われる．

メッセージ処理としてこれまで素直に受け入れられているのは，クラス指向のメカニズムである．しかし，デリゲーションの様に，元のdelegeteeの内部状態を優先することが内部状態の記述を膨らませることがなく，順応性のあるオブジェクトを実現できる(4.5節参照)．具現指向のメッセージ送信の処理は前述のようにクラス指向と異なり，送り手優先であるため，デリゲーションに近い．内部状態の優先に関する各メカニズムの分類は（<デリゲーション,具現指向>,<クラス指向>）となる．

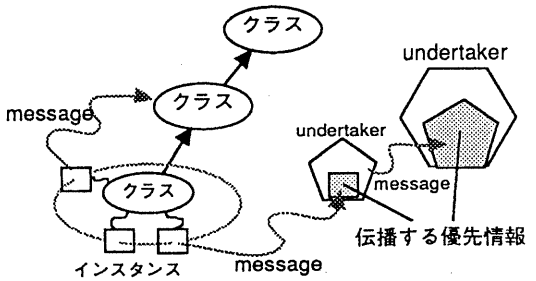
このように具現指向は，メカニズムとしてはデリゲーションとクラス指向の中間である．

## 4．4　クラス指向と具現指向によるモデル構築

クラス指向でのモデル構築の関係を概念的に示したのが図4 (a)である．各クラスは，自身の抽象クラスとして指定されたスーパークラスとの継承関係以外は，インスタンスによる窓口を持つことで外部のオブジェクトと動的に交信する．




(a) クラス指向の概念図




(b) クラス指向と具現指向の融合

図4　クラス指向に対する具現指向の関係

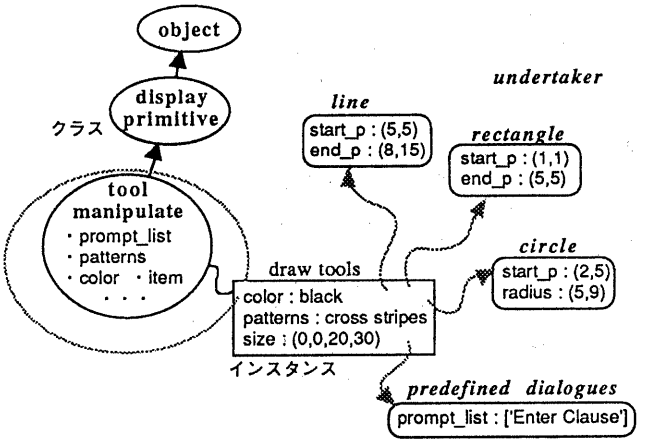



図5　Draw Toolsに関するオブジェクト

具現指向では，クラス指向のモデル構築が持つ区別された2つの関係，クラスとインスタンスの関連構造を示す"IS-A的関係"とクラスとサブクラスの関連を示す"KIND-OF的関係"を持たない．この点でも，具現指向におけるundertakerオブジェクトは，継承機能による複雑さを排除し理解性を強調できる．

図4 (b)のように具現指向におけるundertakerオブジェクトは，利用する側の持つ情報を優先し，機能を達成するまでその優先情報が伝播される．undertakerオブジェクトでは，そのメッセージ機構を拡張することで，機能継承としてのメカニズムを実現することも可能である．

コンパクトなDraw Toolsを実現するモデル表現を取り上げて，具現指向のundertakerオブジェクトの例を示す．図5に示すようにdraw_toolsオブジェクトはtool_manipulateクラスのインスタンスとしてこれまでのクラス指向に基づいて実現されている．ここで，draw_toolsの各部分機能となるline, rectangle, predefined_dialoguesなどが具現指向のundertakerオブジェクトとして個々にプロトタイプが実現されている．draw_toolsから各undertakerオブジェクトにメッセージが送信されると，draw_toolsが持つ内部状態を引き継ぎ，優先したうえで各undertakerオブジェクトの処理が実行される．例えば，lineに対してメッセージが送られるとline自身がその時点でcolorやpatternsなどの内部状態を保持していても，そのメッセージの送信元であるdraw_toolsの内部状態にあるcolorやpatternsの情報を優先して実行される．draw_toolsがstart_pやend_pの内部状態を持っていれば，draw_toolsからのメッセージを受け取った時点でline自身が持つ内部状態

は，そのメッセージ処理に対しては効力を持たない．このように，具現指向のオブジェクトは，利用しようとしている側のオブジェクトに順応する柔軟さを実現する．

### 4．5　モデル構築の融合とその効果

ここでは，クラス指向と具現指向それぞれのモデルの具体的な記述方法と２つの融合によりモデルをどのように構築するかについて述べる．

具現指向としてのモデル構築では，そのオブジェクトがメッセージ処理の段階でundertakerであることを区別できればよい．モデル構築時に，図６(a)に示したように各オブジェクトの表現にundertakerの宣言を行なうことで区別できる．クラス指向のモデル構築には，利用するundertakerオブジェクトの宣言が追加される．図に示したモデル記述の説明については文献[9]参照．undertakerオブジェクトのモデルには，現時点で必要とする具体的内容を記述する(図６(b))．

undertakerオブジェクトは，継承機能を実現していないことから，各undertakerの表現時には抽象性を意識しない．このためundertakerオブジェクトは，一部のクラスのようなフォーマットや初期化の表現をしない．この点でも，クラス指向のオブジェクトより明確に役割のみが表現される．図６の(a)と(b)を比較して，内部状態の記述(value部)を見てもこの違いが分かる．

この２種類のモデル構築を用いる際に重要なことは，これまでクラス指向によって表現していたどの部分を具現指向のモデル表現に移行すべきかである．

クラス指向のモデル構築から具現指向のモデル構築を区別するポイントとして以下の項目を検討している．

(1) 概念的な関係と実際に扱いたい機能から見た関係とに違いがある．
(2) モデル化対象を抽象化するベクトルが明確に定まっていない．
(3) 既にクラス指向で実現されている構造にうまく埋め込めない．
(4) モデル化時点で広範に適用できるモジュールとして考えていない．

この項目にあてはまれば，そのモジュールは，具現指向によって構築する．(1)は，概念面／機能面のどちらかに決めてクラス指向で表現すると，後に利用する際に常にこの違いを意識しなければならない．これは，記述されたモデルの理解性を欠く．このため具現指向により扱いたい具体的機能をモデルとして構築する．(2)は，抽象化のベクトルが統一されていないと(1)と同様に記述されたモデルのの理解性を欠く．具現指向により，抽象化を意識せずにモデル構築を行う．(3)は，具現指向により表現することで，実現されている構造に捕われずに，モデルを構築できる．(4)は，具現指向でモデル構築を行ない，後の利用形態などの分析によりクラス指向に移行する．

いずれの場合も後の支援環境により，必要に応じて具現指向からクラス指向への移行を検討できる．その際に具現指向で表現されていたときの情報は，移行のためだけでなく，記述されたモデルを理解するための情報として活用できる．

ここで示したモデル構築を区別すべきポイントだけでなく，特に，互いのオブジェクトが部品的関係や局所機能を受け持つ関係になるモデル構築には，具現指向のモデルの実現が適する．

具現指向のモデル構築を用いることで，呼び出し側のオブジェクトの性質や属性を部品となるオブジェクトや局所機能を

```
defclass tool_manipulate ::
  relation
    supers display_primiive ;
    undertaker
        menu_handling,
        predefined_dialogues,
        line, rectangle, circle ;
    group display_tool ;

  value
        prompt_list(), item(),
        color(black), name(),
        patterns(normal),
        brushes(), item()
        size( 2,2,30,40 ) ;

  operation
      . . .
      ( line <- start ),
      . . .
      #(prompt_list,Prompt),
      ( predefined_dialogues
          <- prompt_read(Ans) ),
      . . .
```

operation 内の <- が
メッセージ送信である

(a) tool_manipulateクラス

```
defobject line ::
  relation
    undertaker nill ;
  value
        color(red) ,
        start_p((5,5)),
        end_p((8,15)) ;
  operation
    start() :
        @(color,C), @patterns(P),
        wait_click(@name,Y1,X1,_),
        wait_click(@name,Y2,X2,_),
        line((X1,Y1),(X2,Y2),C,P),
        #(start_p,(X1,Y1)),
        #(end_p,(X2,Y2)) ;
      . . .
```

```
defobject predefined_dialogues ::
  relation
    undertaker nill ;
  value
        prompt_list(['Enter Clause']) ;
  operation
    prompt_read(Term) :
      @(prompt_list,List),
      prompt_read(List,Term) ;
    yes_no() :
      @(prompt_list,List),
      yesno(List) ;
      . . .
```

(b) undertakerオブジェクト

図６　モデル表現例

受け持つオブジェクトに伝播できる．図５で示したundertakerオブジェクトである(line,rectangle,circle)と(predefined_ dialogues)は，それぞれdraw_toolsオブジェクトの部品関係と局所機能を受け持つ関係にある．それぞれ，draw_toolsオブジェクトが持つ内部状態の波及を受けて機能を実現する．このように，具現指向におけるundertakerオブジェクトに対するメッセージには，送信元が持つ性質や属性などを表した内部状態を優先させられる．具現指向におけるundertakerオブジェクトが，そのメッセージメカニズムによって実現するオブジェクトの順応性はオブジェクトの部品的関係や局所機能を受け持つ関係には効果的である．

これまでのクラス指向では，オブジェクト間の部品関係もすべて図４(a)のようにインスタンスの窓口を通じてクラス利用形式で行なわれる．つまり，インスタンスを作成し，必要な情報を与えた時点で具体的部品として意識できることになる．しかし，具現指向を用いることで，モデル構築段階で具体的部品を意識してモデルの作成を進められる．さらに具現指向におけるundertakerオブジェクトは，クラス／インスタンスの関係を持たないことから，具体的なundertakerオブジェクトの機能を利用するための，窓口的なオブジェクトの作成を削減できる．

また，具現指向のモデル構築におけるメッセージでは，送信側の内部状態を優先するため，図４(b)のようにundertakerオブジェクトがその処理を達成するまで，常に優先情報が拡張されながら伝播する．

## ５．まとめ

オブジェクト指向概念に基づくモデル表現方法としてモデル定義とモデルの実現を区別することにポイントをおいて改良を示した．３節(a)のモデルの理解問題に対しては，特に継承関係を規定するモデル定義方法を示し，３節(b)の抽象的クラスとモデル構築の問題に対して，具現指向の提案を行なった．構築されたモデルにおける継承関係のあいまいさを取り除くことは，利用側の観点では重要である．また，具現指向のモデル構築では，クラス／インスタンスの関係や継承関係を持たないことによる作成面や理解面の効果もある．さらに，undertakerオブジェクトは，その表現時には抽象性を意識しないため，明確に役割のみが表現される．特に重要な点として，部品関係や分割した局所機能関係を表すモデルの構築において，undertakerオブジェクトが順応性を持つことである．

ここで示した順応性の拡張としてundertakerオブジェクトへのメッセージに付随する優先情報の拡張が考えられる．拡張のポイントは，優先情報内に現われる名前の対応づけや振る舞いに関する情報などである．

### 参考文献


1) G.Booch " Object-Oriented Development ", IEEE Trans. of Software Engineering, Vol.12, No.2, Feb., p.211~221 (1986)

2) 片山佳則 "オブジェクト表現開発のためのクラス構成支援について" 情報処理学会知識工学と人工知能研究会 50-8 (1987)

3) 松浦佐江子，大林正晴 "ソフトウェア開発環境dmCASE" 情報処理学会論文誌 31 Vol.31, No.7, p1091~1103, 1990

4) D. Ungar & R.B.Smith " Self: The power of simplicity " Object-Oriented Programming, Systems Languages and Applications (OOPSLA) '87 Conference Proceedings, Published as SIGPLAN Notices Vol.22, No.12, p227~242, 1987

5) C. Chambers , D. Ungar & E. Lee " An efficient implementation of SELF, a Dynamically-Typed Object- Oeiented Language Based on Prototypes " Object- Oriented Programming, Systems Languages and Applications (OOPSLA) '89 Conference Proceedings, Published as SIGPLAN Notices Vol.24,No.10, p49~70, 1989

6) 小林要，木村高久，織田充 "ソフトウェアの対象モデルと実現モデルの対応構造に関する一考察" 情報処理学会ソフトウェア工学研究会 62-3 (1988)

7) H. Lieberman "Using prototypical objects to implement shared behavior in Object Oriented Systems", Object-Oriented Programming, Systems Languages and Applications (OOPSLA) '86 Conference Proceedings, Published as SIGPLAN Notices Vol.21, No.11, p214~223, 1986

8) R. E. Johnson & J. M. Zweig "Delegation in C++" Journal of Object-Oriented Programming, Vol.4, No.3, p31~34, 1991

9) 片山佳則 "多視点に基づくオブジェクト指向表現システム" 情報処理学会ソフトウェア工学研究会 73-5, p.35~42 (1990)