ICOT Technical Report: TR-441

TR-441

オブジェクト指向表現における再利用のための構成支援環境

片山　佳則(富士通)

December, 1988

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

オブジェクト指向における再利用のための構成支援環境

片山　佳則
Yoshinori Katayama
富士通㈱国際情報社会科学研究所
E-mail : *kata@iias.fujitsu.junet*

1. はじめに

現在,様々なソフトウェアやプログラミングに関してオブジェクト指向の考え方が導入されている.今後はさらに,研究対象としての広がりだけでなく,実用的なシステム等に対して,特徴的な概念として意識され,利用されると考えられる.

そこで本報告では,オブジェクト指向としての表現やプログラミングにおいて,各開発者が行っている様々な作業や思考活動を踏まえ,支援を必要とする部分や支援可能な機能を分析する.その上で,開発者の負担を出来るだけ軽減するような支援環境の構築を目指す.はじめに,オブジェクト指向プログラミングの利点をまとめ,次に,問題解決の例を踏まえて開発者の作業や思考活動上で妨げとなる点を取り上げる.さらに,その障害を克服することを踏まえた支援環境とその例を提示する.このような環境として既に浸透しているものの代表例として,Smalltalkのブラウザがある[1].また,プログラミングのための基本的な方法論は古くからさまざま考えられている[2,3].本報告では,これらの方法論を参考にし,ブラウザとは異なる観点で,支援を進めるような環境を考えている.その際に,知識表現として検討されている表現方法(特にpart-of表現)[4]や対象への視点の転換に注目している.本支援環境は,特にオブジェクト構造の分析やオブジェクト間の関係に重点を置いている.ブラウザが改良･編集中心の開発環境であるのに対し,この支援環境は,機能の実現関係･機能探索を重視した支援環境である.

2. オブジェクト指向プログラミングの利点

オブジェクト指向プログラミングは,生産性が良い,再利用性が高い,プロトタイプ作成が容易でありデータ構造の記述が容易である[5],また,オブジェクトの記述に異なる働きを付加したり結合できることなどから拡張性,保守性がある,さらに,分散的記述により対象を効果的に実現できるなど[6],様々な視点から注目されている(図1).

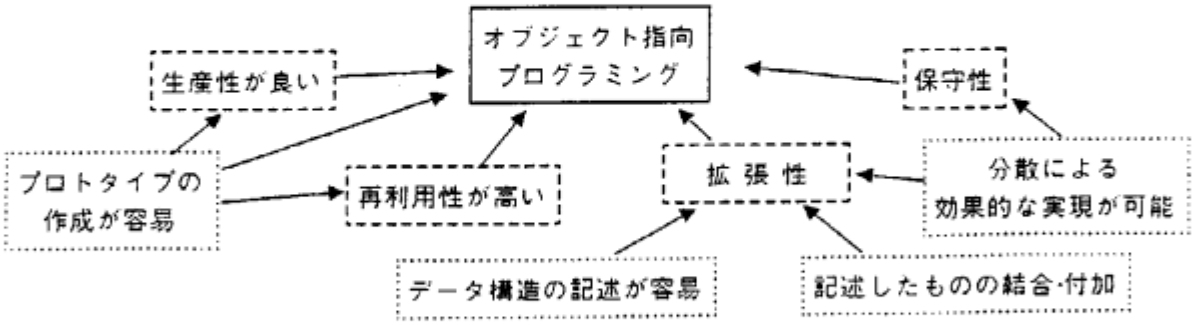


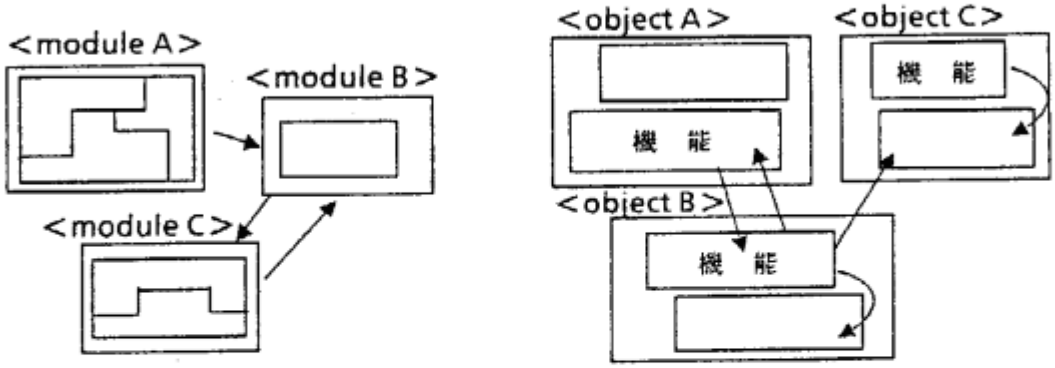
図1　"オブジェクト指向プログラミング"の注目点

また，オブジェクト指向プログラミングの利点は，クラスの根本的考えに基づき，雛形であるクラスから，インスタンスを複数作り出し，それらを利用することで得られるものである．これは逆に，各クラスに対して唯一のインスタンスしか作成しない場合は，良さが引き出せないことを示している．このことから，共通な機能は，すべて共通利用する形で処理する必要がある．

オブジェクト指向概念により，ある種の問題解決を行うことの利点は，問題が持っている対象やモデル(概念)をオブジェクトとしてより自然に表現できることである．これにより，これまで扱えなかった機構(対象主体の処理等)や扱いが容易でない形式(能動的データの形式等)を簡単に構築できる．さらに，プロトタイプレベルだけでなく，計算能力と対象にできる情報・知識の量等から，より実際的な問題に対して，ボトムアップにも，トップダウンにも開発/応用できる[7]．例えば，プログラミングも含めた設計自身を，機能のもっとも抽象的な概念レベルの記述から始められ，各開発ステップを通じてリファインしていくことで，いくつかの小さく簡単なものに分解した記述に移行できる．この場合の各要素は，実装可能な抽象化の十分低いレベルであり，そうなるまで各ステップが繰り返される．これは，オブジェクトとして管理対象モジュールを小さくできることにも拠る．

拡張性は，(1) オブジェクト自身が単純な設計であることから，仕様の変更への対応が容易にできる，(2) 各オブジェクトが自主的なモジュールとして分散的に記述できるため，変更への対応を関連するモジュール単位あるいはモジュールが持つ機能単位にできる，ことである．

簡単なモジュラープログラミングでも，サブルーティンのようなそれぞれの部品の集まりとして考えられる．この技術から，結果的には各部分モジュールが一貫した内容のもとに組織化される．これは，モジュラー開発の本質が，新しい問題をいくつかのサブ問題へ分解し，そのそれぞれを解くことであるから明らかである．しかし，このプログラミングはオブジェクト指向のように個々のモジュール内の記述に対して考えられる拡張性や再利用性などの利点を持っていない．一般的なモジュラリティとしては，これらのことも考慮していなければならない(図2)．



(1) 簡単なモジュラープログラミング　　(2) オブジェクト指向プログラミング

図2　プログラミング形式

再利用性は，応用に対して，これまでにある部分を利用するソフトウェア生産性の能力である．このためモジュール分解として分割された個々の自主的モジュールは，その拡張のためにはopenであり，それを他のモジュールから利用するためにはclosedでな

ければならない．これが情報隠蔽[8]に結び付く．オブジェクト指向でのモジュラリティは，各モジュールがwell-definedで情報隠蔽としても適切な記述を与えることから，再利用性と拡張性を達成するための手本となるものである．

3. 問題解決のためのオブジェクト指向とその問題点

問題解決のためにオブジェクト指向を利用することを，プログラミングの観点で考えた場合，基本形態はやはりオブジェクトへのメッセージ送信で実装することだけである．このことから，オブジェクト指向におけるプログラミングは，問題解決に関するオブジェクトとメッセージをそれぞれ考えることと，さらにオブジェクトとメッセージの結びつきを考えることが基本になる．したがって，オブジェクト指向アプローチの基本的ステップは，ある解決すべき問題に対して，それを解くために必要なオブジェクトを設定することから始まる．そして，それらのオブジェクトが応答すべきメッセージを考え，最終的に，与えられた問題を解くための一連のメッセージを確立することである．これらの個々のステップに対して，前節で挙げたオブジェクトの持つ利点が利用される．逆に，オブジェクトへのメッセージ送信の形態が基本であるから，これらの利点が素直に導けるといえる．つまり，問題解決のためのオブジェクトや各応答メッセージ，一連のメッセージの組をどのように決定するかによって，前節で挙げたオブジェクトが持つ利点の活用可能性が決まる．

以下では，オブジェクト指向としてこれまでに挙げた利点を踏まえて，問題解決の簡単な実例を用い，開発者の障害となる点を明らかにする．

3.1 記述の容易性/生産性に関する問題点

はじめに，問題解決の例として，一般的なハノイの塔を取り挙げ，オブジェクト指向の生産性や，記述の容易性を踏まえて，そのプログラミングのステップを分析する．

[第一ステップ] 問題設定および問題解決のために必要となるオブジェクトを挙げる．

この例では，オブジェクトとして主に考えられるものは以下の3つである．

・円盤　　・柱　　・円盤を動かす動作主(全体の操作を管理する)

この他にも，円盤の数を数えるオブジェクトや，柱が持っている円盤を表現する配列を表すオブジェクトなど細かく挙げることができる．このように一般には，解決したい問題の中では明らかに示されないオブジェクトを用意しなければならない場合がある．

[第二ステップ] 問題解決のために各オブジェクトが応答しなければならないメッセージを挙げる．

柱： 最上段にある円盤を答える．円盤の枚数を答える等

円盤： どの柱にあるかを答える．大きさを答える等

動作主： 与えられた初期状態に対して問題解決を行う．

一枚の円盤を動かす．　　複数枚の円盤を動かす．

動作した状態を指定位置に示す等

配列： データを配列に格納する．

配列内のデータの状態を答える等

このステップで挙げるメッセージは，すべてのオブジェクトに共通なメッセージの応答(new等)は対象外とし，各オブジェクトが問題解決のために応答すべきものである．

[第三ステップ] 問題解決のための一連のメッセージの組を決定する．

これまでに挙げたオブジェクトやメッセージを用いて,問題解決のためのメッセージ列を考える.例えば,動作主では以下のようなメッセージ列が考えられる.

<動作主>　動作を行う動作主を作成する.
　　初期状態の設定を行う.
　　初期状態に対応して必要な円盤や柱のオブジェクトを作成.
　　問題解決を行う.
　　・動かしたい円盤が一枚の時,目標とする柱に円盤を動かす.
　　・動かしたい円盤がN枚の時,
　　(N-1)枚の円盤を補助の柱に動かす.
　　N番目の円盤を目標とする柱に動かす.
　　(N-1)枚の円盤を補助の柱から目標とする柱に動かす.

第三ステップにおいて,第一ステップや第二ステップで挙げたオブジェクトやメッセージの中で,不必要なものが出てくる.これは,前の各ステップでは問題解決と切り離した状態で,対象表現自体の構造を考えていたために起こる.基本的には,第一ステップで挙げたすべてのオブジェクトを用い,各オブジェクトの機能の境界を決定し,第二ステップで挙げたすべてのメッセージを実現させ,一連のメッセージ列を構成してプログラミングを行うことも可能である.このようなプログラミングの違いは,対象表現自体の自然で素直な構造と開発者が問題解決のために描く自然で素直な構造との差である.この関係は,カテゴリ[9]と類比させて考えられる(図3).

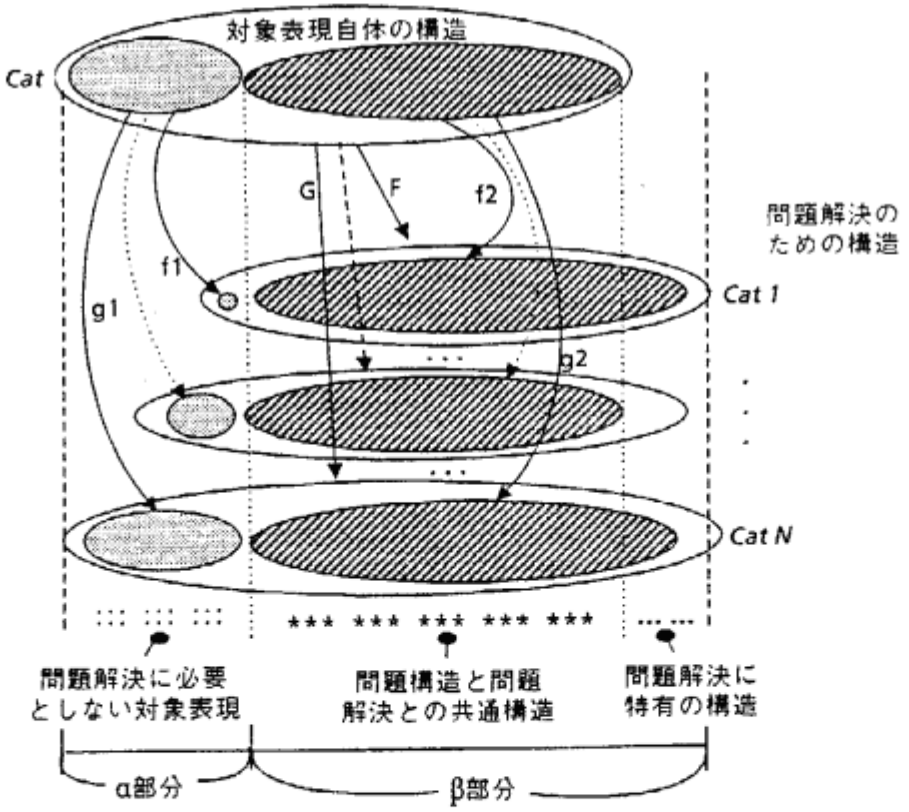


図3　カテゴリと類比

対象表現自体の構造をカテゴリ*Cat*に対応させ,プログラム上の問題解決のための構造をカテゴリ*Cat N*($N \geqq 0$)に対応させる.この場合*Cat*と*Cat N*との関係をfunctorとして

表せる(図中のF, ...,G).さらに,問題解決に直接必要としない表現構造(α部分)を除いて,各カテゴリのサブカテゴリを考えると,*Cat*のサブカテゴリと*Cat N*のサブカテゴリの間もfunctor(図中のf2, ...,g2)が考えられる(partial functor). これらのpartial functorには,互いにnatural transformationに相当する自然な対応がある. このようなpartial functorは,多数考えられる. これが,プログラム上での問題解決の構造を決定するための妨げの一つになる.

表現の観点では,α部分のサブカテゴリも含めて同形対応になるようにプログラミング側の構造を決めることが望まれ,操作や振る舞いの観点では,β部分のサブカテゴリが同形対応になることが望まれる. これらの対応関係をどう決めるかは,開発者に委ねられている. これらも妨げの一つである. さらに,対象表現自体の構造において,*Cat*をα部分とβ部分の各サブカテゴリに分類できない. したがって,この同形対応の関係も容易に決められない. これも,構造を決定するための妨げとなる. ここで上げた障害をどう処理するかにより,問題解決に対するプログラムとしての素直さや自然さの割合が決まる.

先の例で取り上げたハノイの塔のプログラムも,これらの障害の処理に応じて,様々なオブジェクト構成が実現できる.

3.2 拡張性/保守性に関する問題点

次に,オブジェクト指向の拡張性や保守性の観点として,先のハノイの例において,動作主の行う動作を視覚的に表示するための拡張を考える. これは,各オブジェクトがメッセージの一部として受け取るパラメータに対する操作を,いくつか付加することで実現できる. つまり,根本となるアルゴリズムは変わらず,オブジェクトが受け取ったメッセージに対して処理する機能を付加してやればよい. この編集操作の対象は,円盤や柱のオブジェクトを表現していても,動作主を実現しているオブジェクトに限られる. このことから,改良が容易に行える. この処理のように,オブジェクト指向プログラミングの特徴の一つが,すでにある解に対する付加を容易に行え,拡張できることである. ただし,この場合,対象とする機能の実現方法や,実現のためのオブジェクト間の関係等が明確でなければならない. これらの情報が一つのカギである.

ここで,問題解決と離れて,オブジェクト指向プログラミング自身が持つ特性を,開発者からの視点で,オブジェクトの記述に関する項目とオブジェクトの持つ機能の実行/処理に関する項目で分類すると図4のようになる.

図4　オブジェクト指向プログラミングの特性

abstraction: 複雑な対象や概念を,簡単に表現する方法である. これは,オブジェクトが持つ機能を理解するためでなく,対象とする機能の実現のためにあるもの.

encapsulation: プログラム要素が個々に記述されることで得られるものである. この単位はオブジェクトであり,持っている機能の保護と境を明確にするもの.

inheritance: クラス階層で与えられる上位クラスの属性の継承である. 多重継承

もオブジェクト指向が持つ一つの特性である.

polymorphism: 異なるオブジェクトに同じメッセージを送れ,各オブジェクトがそれぞれ自分に合った応答を行うこと.

categorize: オブジェクトを対象領域に合わせて自由に分類し様々な視点を構成できること.実行/処理や記述に対して直接関係しないが,この分類は利用・探索する立場で重要になる.

はじめの4項目は,さらに詳細に分析することで,オブジェクトやメッセージを構成している様々な要素やそれらの間の関係にブレイクダウンできる.つまり,オブジェクトが持つ本質的な特性と言える[10].最後の項目の'分類'は,各オブジェクトが持つ機能の理解や探索・再利用に関して開発者の視点に立って特徴づけができることである.

プログラミングとしてのこれらの特性は,すべてオブジェクトという対象を中心に設定することが基本になって得られたものである.

オブジェクト指向が持つ利点や特性からも,オブジェクト指向プログラミングのステップの中で,問題設定および問題解決のために必要とされるオブジェクトを挙げ,それらをどのように設定するかが重要である.

次節では,これらのことを踏まえて,オブジェクト指向における特徴の一つである分類法に焦点を当て,オブジェクト指向プログラミングとしての支援環境の提案を行う.

4. 再利用のための支援環境

オブジェクト指向プログラミングの支援環境には,開発の各段階(初期の開発段階,ある程度開発が進んだ段階等)に対応した支援が考えられる.初期段階では,これまでに表現されているオブジェクトの理解と利用のための支援や,問題解決に適切なオブジェクトを抽出する支援が必要である.しかし,オブジェクトの抽出は,容易ではない(3.1節).そこで,この段階では,表現されているオブジェクトの理解や利用のために,分類による支援環境に重点をおく.その後の段階で,オブジェクトの大枠や機能の分散方法がある程度決まったところで,問題解決としてのオブジェクト構成の適切さや各機能の実現方法に関する支援を考える.

4.1 分類にもとづく支援

本節では,再利用や拡張による開発を考慮した初期段階の支援を提案する.この段階では,各オブジェクトが実現している機能の分類に基づいた支援環境が重要になる(3.2節参照).各オブジェクトの機能分類の分類対象は,Smalltalk のブラウザのように,クラスの分類と,オブジェクトが持つメソッドの分類が適切である.ただし,機能に注目するため,メソッドの分類とともに,変数との関連性も重要である.また,利用の立場では,完全に他と交わりのない独立な分類を扱うのでなく,ある機能の実現のためのオブジェクトやメソッドの関係を,他の機能との重複を意識せずに分類することも必要である.これが機能の実現を重視した分類である(図5).ただし,実際の変更等には別の支援が必要である.

後者の機能実現の分類は,これまでの知識表現で扱われていたpart-whole関係としてオブジェクトや機能の関係を扱うことである.これは,特にオブジェクト記述において属性記述(状態変数,スロット等で呼ばれる)と機能記述(プロトコル,メソッド,スクリプト等で呼ばれる)に並ぶ,関係記述(上位,下位等)において明示的にオブジェクトのpart-whole関係を記述しない場合,特に重要な分類となる.オブジェクトの記述として,明示

(a) ブラウザでの分類　　(b) 機能の実現方法重視の分類

図5　オブジェクトや機能の分類方法
(楕円はクラスやメソッドを表す)

的でない機能実現の関係を支援環境として提供することは,機能を容易に理解させるだけでなく,新たな応用に機能を利用するためにも効果的である.

この機能の分析は,文献[11]において機能分割の考え方で用いたシンタックス情報から作成するsetを用いて行う.このsetは,オブジェクトが持つ変数(instance variable, class variable等)を基に作られる機能分類である.このset作成の段階に合わせて,機能分類に段階をつける.第一段階のsetは以下の2つの結合規則で作成されるものとする.

(a)　対象オブジェクトが持つ変数の更新を行うメソッドをその変数とともにsetとして結合する.

規則(a)は,変数の更新操作に重点を置き,機能的分類であるsetの対称として切り離せない基本的な結合関係を定めるものである.

(b)　規則(a)で結合される各メソッドに対して,メッセージ交換関係が強連結[12]となるメソッドを結合する.

規則(b)は,機能を実現するために,強く協力し合っているメソッドを結合する規則である.この二つの規則により,強い連結関係で機能を実現しているset(機能的分類)が得られる.このsetにより,対象オブジェクトが持つ機能の大枠を示せる.これが第一段階のsetである.また,規則(a)と(b)で分類されたsetごとに,別のオブジェクトとのメッセージ交換を調べることで,それらの結びつきを一種の関係情報として得られる(図6a).基本的には,これらの関係も一つの機能分類に包含される.そこで,この別のオブジェクトに対するメッセージ交換は,オブジェクト間の機能実現の関係として,set作成時に区別する.呼び出す側のメソッドを持つオブジェクト(またはset)を全体側に対応させ,呼び出される側のメソッドを持つオブジェクト(またはset)を部分側に対応させて,互いのオブジェクトにpart-whole関係としての関係づけを行い,複数のオブジェクト間での機能実現の関係を導く(図6b).このオブジェクト間の機能実現の関係は,オブジェクトの枠を外して,機能に重点を置いたset間の関係(機能分類自体の関係)としても示せる(図6c).ただし,この機能間の関係は,メッセージの送信先がシンタックスレベルで明らかなもの(擬変数や変数にBindされるものが明らかなもの)に限られる.

オブジェクト間の機能実現関係や,setによる機能ごとの実現関係の情報は,オブジェクトがもつ機能の実現方法を細かく理解できるだけでなく,機能的にどのオブジェクトまでを再利用するか,どのオブジェクトの機能を変更して再利用するかなどの検討情報としても役立つ.

第一段階のsetにより,オブジェクトが持つ機能の大枠を把握でき,オブジェクトごとの再利用を検討できる.しかし,これらの規則では,機能として分類されないメソッドが多数現れる.そこで以下では,さらに再利用可能性を柔軟な機能分類として検討する

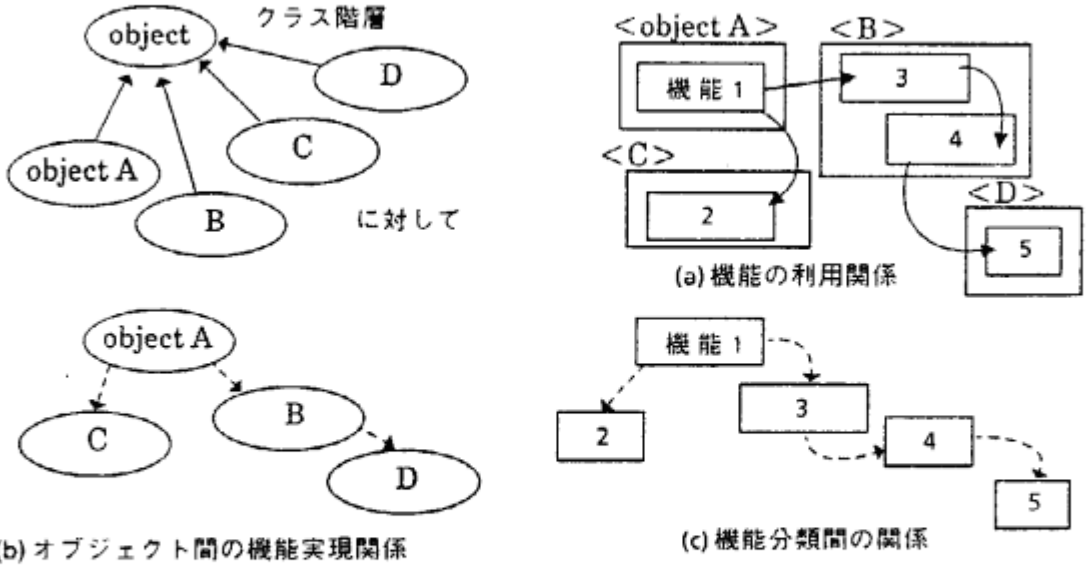


図6　機能分析の分類における関係表現

ように,これまでの規則(a)と(b)に次の規則(c)を付加して機能分類としての第二段階のsetを考える.

(c)　メソッド間に強連結な関係を意識せず,メッセージ交換関係のあるメソッドを結合する.

規則(a)と(b)では,機能を実現している核になる部分をsetとして抽出できる.しかし,オブジェクトが持つ機能実現のためのメソッド間の関係としては,規則(b)の制限が強すぎる.オブジェクトが機能の実現のために用いるものを,独立性を意識せずsetとして分類する立場では,関係するすべてのメソッドを結合する必要がある.この規則(c)の追加によって,これらのメソッドの結合を実現する.

この規則(c)を加えた場合も,別のオブジェクトに対して行われるメッセージ交換には,それらのオブジェクト間に第一段階と同様の関係づけを行う.このようにして,オブジェクトの機能実現に関するpart-whole関係を柔軟に実現し,分類による再利用のための支援情報として用いる.

第二段階のsetによって,機能実現のために必要なメソッドがすべてsetや,set間の関係として示される.したがって,各機能を実現しているsetでの支援が行え,どのsetを再利用するかなどの機能レベルの再利用を実現できるようになる(図7).

オブジェクトにおける再利用や拡張の長所は,ここで示した機能実現関係におけるあるオブジェクト自体の交換や追加,または,setとして表される細かい機能の実現方法の交換や変更,追加を自由にできることである.したがって,このような操作を容易に行えるように本節で示した支援情報を提示することが,初期の開発段階で必要な支援環境である.

4.2 オブジェクトの構成と機能の実現に関する支援

次に,ある程度開発が進んだ段階での支援として,問題解決に対するオブジェクト構成の適切さや,各オブジェクトが持つ機能の実現方法に関する支援環境を考える.

本質的に素直で自然なオブジェクト構成のためには,3.1節で述べたように,

(1) 開発者が考えている対象表現自体の構造を表すカテゴリに対し,問題解決のための構造のカテゴリにはnatural transformationのあるものが多数考えられる,

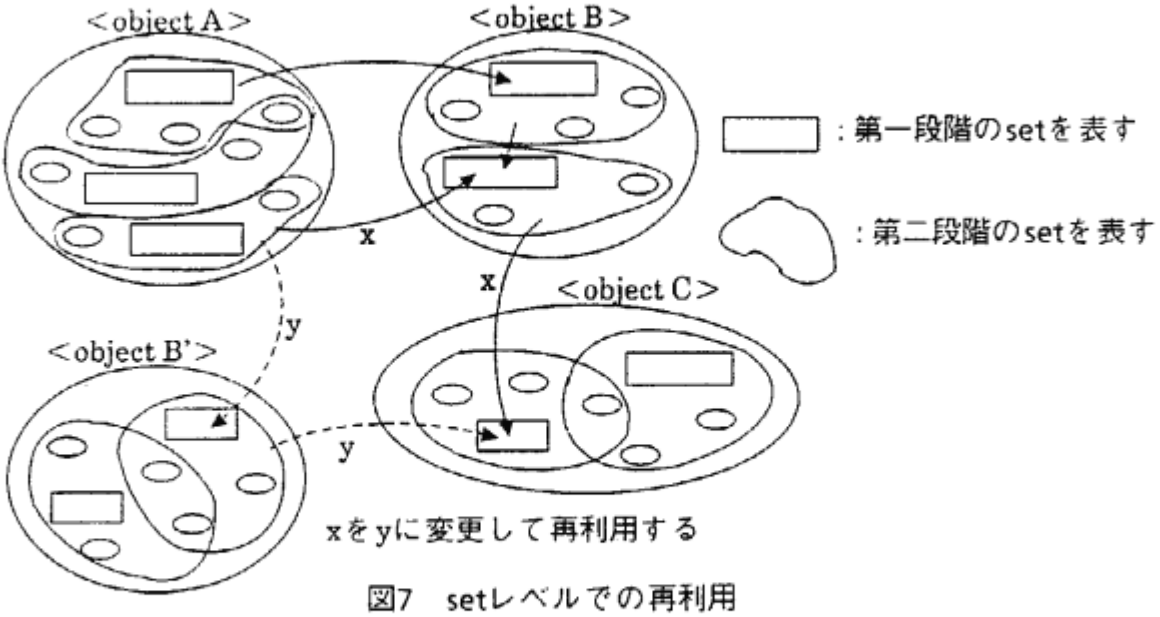


図7　setレベルでの再利用

(2) 問題解決のための構造を表すカテゴリは,一連のメッセージを構成する段階で明確になる. しかし,対象表現自体の構造を表すカテゴリは明確にできない. 特に,対象表現自体でカテゴリを図3のα部分とβ部分に分離できない.

この(1)と(2)から,問題に対して素直で自然なオブジェクト構成を唯一に決定できない.

しかし,この節で対象としている段階では,目的とする問題解決に対する一連のメッセージを情報として取り出せる. そこで,その一連のメッセージを基にして,機能的観点から問題解決として適切なオブジェクト構成の候補を指摘できる.

この構成支援のための情報として,すでに開発されている問題解決のオブジェクトに対して,前節の規則に次の規則(d)を付け加えて分類したsetを用いる.

(d)　対象オブジェクトが持つ変数の参照を行うメソッドをその変数とともにsetとして結合する.

これは,変数の更新だけでなく参照関係も結びつけて,機能的に関わりを持つすべての変数やメソッドを結合する規則である.

この規則(d)を加えて分類することで,対象機能として,変数中心に関わりのあるすべてをsetにできる. この分類情報により,機能の実現関係から大枠として考えられるオブジェクト構成の候補を分析できる. 例えば,個々に分類された機能をそれぞれ実現するオブジェクト構成とか,いくつかの分類された機能をまとめて一つのオブジェクトとするなど. これらの分析は,各分類された機能ごとに,その対象オブジェクトやそれらの関係を再度検討させることになる(図8). 検討結果が,これまでに開発したオブジェクト構成と同一であれば,問題解決に対する機能面でのオブジェクト構成として適切であったことがわかる.

問題解決の例として3.1節で示したハノイの塔のプログラム(smalltalk上のプログラムを例にする)を取り上げて,ここでの機能的観点での構成支援を例示する.

(1)　[問題解決のために動作主だけのオブジェクト(Hanoi)を考えた場合]

このオブジェクトから得られる変数やメソッドの機能的関係を図9に示す.

ここで,図中の各項目が表している機能を次にまとめる.

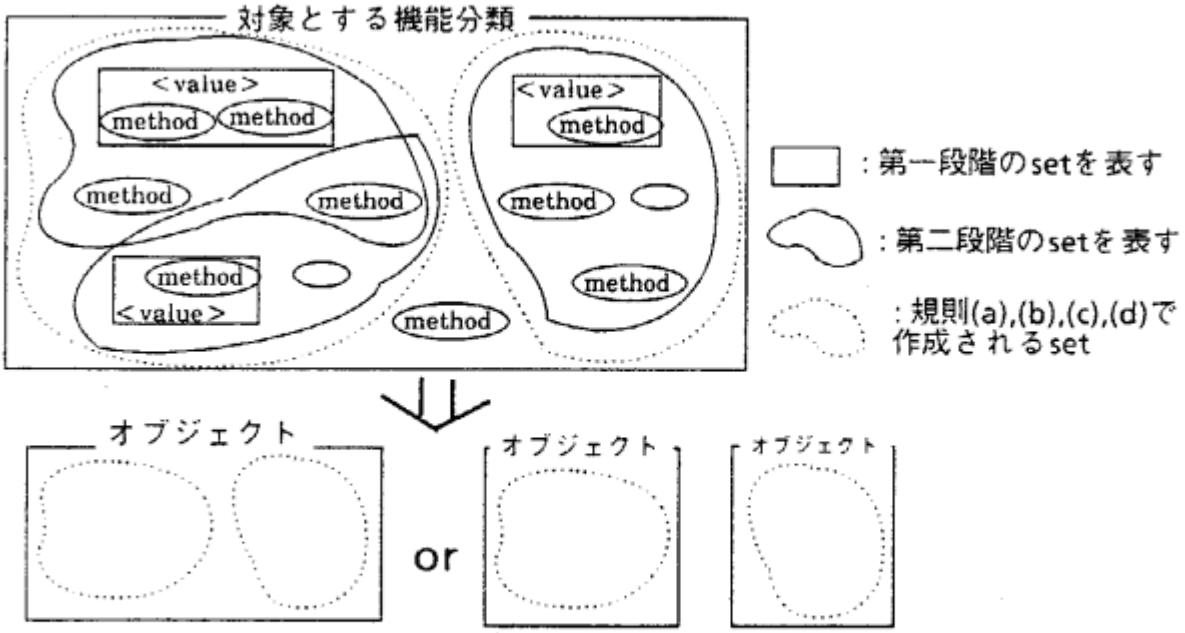


図8　分類情報によるオブジェクト構成支援

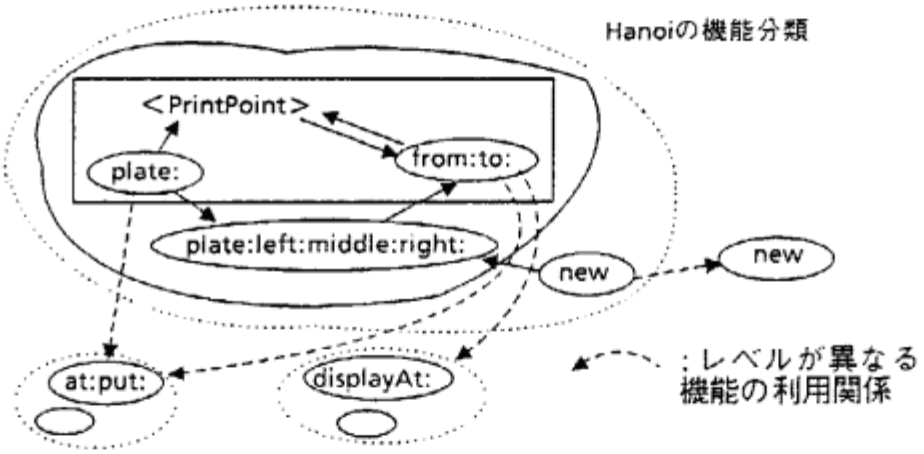


図9　オブジェクト(Hanoi)が持つ機能分類の関係

PrintPoint は印刷座標位置のデータを示すHanoiオブジェクトのinstance variableである.
plate: は印刷開始位置を決めて, 円盤や柱を特定するメソッドである.
plate:left:middle:right: はplateとして与えられる枚数の円盤を柱leftから柱rightに柱middleを補助にして移動させるメソッドである.
from:to: は 一枚の円盤を柱fromから柱toに移動させるメソッドである.

この場合には, variableが一つであることから, 分離可能な機能分類は存在しない. したがって, オブジェクト構成としては, このHanoiだけで実現することが適当である. ただし, displayAt:やat:put:を実現している機能を別の機能で変更することが容易であることがわかる. これは, 機能拡張のための支援になる. このように, ここで示した機能分類情報は, 再利用のための簡易な変更や改良についても効果がある. Hanoiが持つ機能の変更にも, 示されるsetにより段階が考えられる. 例えば, plate:left:middle:right:で実現している機能の変更から検討するなど.

(2) [問題解決のために3種のオブジェクト(Hanoi, Tower, Plate)を考える場合]

この3種のオブジェクトから得られる機能分類の関係を図10に示す．Hanoi, Tower, Plateの各オブジェクトをH, T, Pで表し，それぞれが持つ変数をH-PrintPoint, P-Name等で表している．

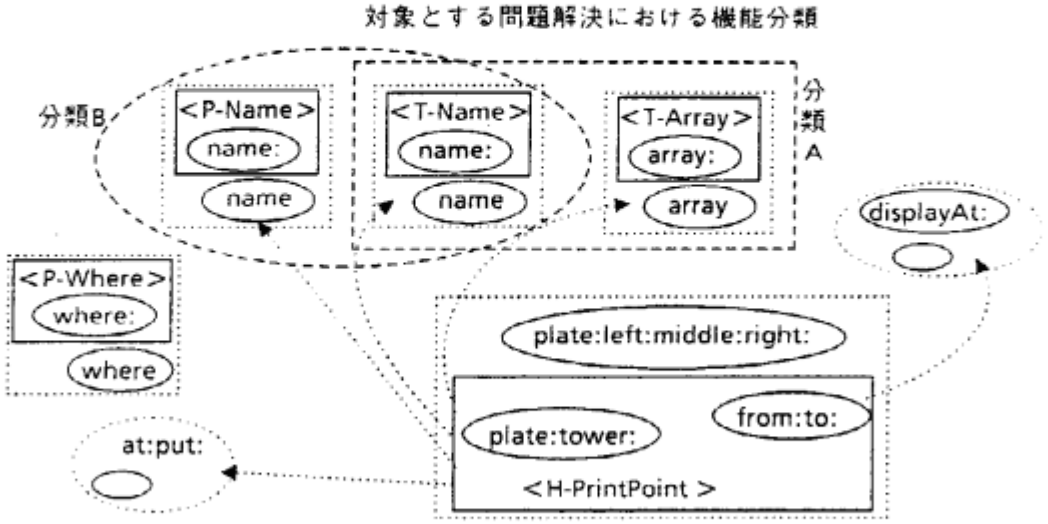


図10　3種のオブジェクト(Hanoi,Tower,Plate)が持つ変数やメソッドの機能的関係

図9と同じ変数名やメソッド名は，その機能も同じものを表している．新しいメソッドであるname:やnameなどは，':'が付いている方が変数の更新であり，付いていないものは変数の参照の機能を実現している．

図10から大きく7つの機能を実現していることがわかる．各機能分類を，オブジェクトに対応させると7つ以上のオブジェクトが必要である．しかし，<P-Where>の機能は利用されていないし，at:put:やdisplayAt:をここでのオブジェクト構成の対象外とすると，4つの機能分類からなる構成が考えられる．ここで，Towerとしての機能をまとめる場合(分類A)は，初めの構成で良いことがわかり，Name処理をまとめると考える(分類B)と，別の構成ができる．また，TowerやPlateの各機能分類の対象機能が単純であるため，(1)の例のようにHanoiオブジェクトに取り込んで構成を実現したり，Plateに関する機能分類だけをHanoiの機能分類に含めてオブジェクト構成を実現することなどが，機能分類の面から新たなオブジェクト構成の候補として考えられる．

これらの例でわかるように，本節での機能的分析による支援環境は，開発者の考えで作成されたオブジェクト構成や内部の機能記述を，各機能実現のためのオブジェクト構成として適当であるかを評価させるものである．さらに，機能の拡張や変更，改良における重要な視点をも与えることができる．

5. まとめ

本報告で提案した支援環境による効果を以下にまとめる．

○ 対象とする問題解決での機能分類，システムが備えているプロトコル等の機能分類(第4節)から，各機能の実現方法を理解し易くなる．

○ 各オブジェクトを再利用するための支援環境として，機能的視点に注目した機能の分類に基づく支援(4.1節)を提案した．これにより，クラス階層による縦の関

係でなく，機能実現の横の関係をたどり，オブジェクトが備えている機能の部分的拡張や改良が容易になる．

○ 機能的観点でのオブジェクト構成の候補が指摘される(4.2節)ことから，機能的に適したオブジェクト構成を導けるようになる．

これらの効果をさらに発展させるには，次のような点を考慮する必要がある．

機能的分類をどのレベルまで示すかによって支援の効果が異なる．対象とする問題解決に合ったレベルで分類がまとめられると，支援環境として充実する．smalltalk では，外部のオブジェクトへの呼び出しを除き，selfやsuper などの疑似変数や変数にBind されるものが明らかなものだけを分類対象としている．

特に，本報告で提案した支援環境では，一つの問題解決を対象として考えている(4.2節)．したがって，構成支援の情報も候補の提示にとどまっている．この提示を指示にするためには，複数の問題解決を対象とし，各機能分類の利用関係にもとづき，その分類をオブジェクト構成として分割するなどの規則を与え，実質的な支援に結びつける必要がある．4.1節や4.2節で示した各オブジェクトが持つべき機能の分析以上の支援は，対象表現自体での構造が得られるような対象モデルをうまく取り出す方法が必要になる．これにより，さらに効果的な支援環境を実現できる．本報告では，触れなかったが，機能分類どうしの同一性の判定も考えられる[13].

尚，本研究は，第五世代コンピュータプロジェクトの一環として行ったものである．

[参考文献]

[1] Goldberg, A. "Smalltalk-80 : The Interactive Programming Environment" Addison-Wesley, 1984

[2] Stevens, W.P., Myers, G.J. and Constantine, L.L. "Structured design" IBM Systems Journal ,Vol.13 No.2 , pp115-139, 1974

[3] Wirth, N. "Programming development by step-wise refinement" Communications of the ACM, Vol.14 No.4, pp221-227, April 1971

[4] Findler, V.N. "Associative Networks : Representation and Use of Knowledge by Computers" Academic Press, 1979

[5] 丸一威雄 "オブジェクト指向言語によるソフトウェア設計" 情報処理学会研究会報告 ソフトウェア基礎論 85-SF-13, 1985

[6] 久野靖，米澤明憲 "情報処理：大特集 オブジェクト指向プログラミング" 情報処理学会誌 Vol.29 No.4, 1988

[7] Grady, B. "Object-Oriented Development" IEEE Trans. Softw. Eng., Vol.SE-12, Feb. 1986

[8] Parnas, D.L. "Information Distribution Aspects of Design Methodology" Information Processing 71, North-Holland, 1972

[9] Coldblatt, R. "TOPOI : The Categorial Analysis of Logic" North-Holland, 1979

[10] Pinson,L. and Wiener, R. "An Introduction to Object-Oriented Programming and Smalltalk" Addison-Wesley, 1988

[11] 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" 情報処理学会研究会報告 知識工学と人工知能 50-8, 1987

[12] Harary, F. "GRAPH THEORY" Addison-Wesley, October 1972

[13] 片山佳則 "オブジェクト指向表現のための同一化による構成支援" 日本ソフトウェア科学会第5回大会論文集 pp157-160, 1988