オブジェクト指向プログラミングにおけるデメテルの規則の定式化

黄 錫 炯† 辻 野 嘉 宏† 都 倉 信 樹†

オブジェクト指向プログラミングにおいて，品質の良いプログラムを作成するためには，プログラムの性質と構造などを厳密に理解した上でプログラムの複雑さを減らすための方法を考える必要がある．オブジェクト指向プログラムでは，プログラムの複雑さはクラスの間のメッセージ転送より発生するクラス間の相互依存関係に起因する．この考え方に基づいて，クラス間の不必要な相互依存関係を減らすためのプログラミングスタイルに関する規則としてデメテルの規則(The Law of Demeter)が知られている．しかし，その定義が非常に曖昧であるため，いろいろな解釈が生じる恐れがあり，実際にプログラムに適用する際には問題がある．本論文では，従来の非形式的なデメテルの規則を実際のプログラムに適用，評価するために，クラス間の関係として継承と集約，そして関連などの諸定義を行い，それらを用いてデメテルの規則を定式化する．またデメテルの規則の判定アルゴリズムと，デメテルの規則に違反したプログラムに対する変換方法について述べる．

Formulations of the Law of Demeter in the Object Oriented Programming

SUKHYUNG HWANG,† YOSHIHIRO TSUJINO † and NOBUKI TOKURA †

In the last few years, several articles have been devoted to the study of the Law of Demeter in the object oriented programming. The Law of Demeter is a style rule that aims at eliminating unnecessary coupling among classes. Although a large number of studies have been made on the informal definitions, little is known about the formulation of the Law of Demeter. In this article, we define three relationships among classes i.e. inheritance, aggregation and association, and formulate the Law of Demeter. We also propose the algorithms to decide whether a given program satisfies the law and discuss the transformation for the illegal program.

1. はじめに

近年，オブジェクト指向に対する関心が高まり，数多くのオブジェクト指向言語が提案されてきた．また，再利用性と情報の隠蔽，継承，多態性を利用したソフトウェアの構築，そして分析と設計に関するオブジェクト指向方法論の登場など，ソフトウェア工学にも新しいパラダイムの時代を迎えるようになった[1]．

しかし，オブジェクト指向プログラミング言語で作成したプログラムに対する評価や複雑さに関する尺度の提案についての研究はまだ進んでいないのが現状である．とはいえ，最近，オブジェクト指向言語のクラスの品質評価に関する研究[10]が行われている．この研究では，クラスの品質評価尺度として，クラスをモジュールと考えたモジュール強度，そしてクラス間の継承関係による結合度という評価尺度を提案している．さらに提案尺度をC++プログラムに適用した結果，多くのソフトウェア開発者が抽象データ型と継承機構を理解せずに開発に取り組んでいることが報告されている．

また，クラスの間の相互依存関係に注目して，メッセージの宛先に制限を課することに関する研究[4)~6)]が行われてきた．この研究では，クラス間の結合度を減らすための制限として，デメテルの規則(The Law of Demeter)という指針を提案している．あるクラスがこの規則に従うと，そのクラスのメソッドは制限されたクラスのメソッドのみを利用するようになり，クラスの間の結合度が低くなる．しかし，この研究では，非形式的なデメテルの規則の定義とその規則に従うようにプログラムを変換する方法のアイデアの提案だけで，具体的に与えられたプログラムがデメテルの規則を満たすか否かを判定するアルゴリズムと，違反した場合に規則を満たすように変換するアルゴリズムなどに対しては議論されていない．実際のプログラムを対象としてデメテルの規則を満たすか否かを判定するために，まず，より形式的な定義が必要となる．

本論文では，以上の研究に基づき，クラス間の相互

† 大阪大学基礎工学部情報工学科
Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University

依存関係に注目して，デメテルの規則を定式化し，与えられたオブジェクト指向プログラム（C++）に対して判定するアルゴリズムを提案する．また規則に違反したプログラムをその制限指針に従うように変換するいくつかの方法[4),5)]の基本となったリレーメソッド追加法をアルゴリズム化し，その有効性を検討する．

2. 諸 定 義

本章では，プログラム上に登場するクラスとメソッド，そしてクラス間の関係などに対する定義を行う．まず，プログラム $\mathcal{P}$ のクラスの集合を $\mathcal{C}_p$，メソッドの集合を $\mathcal{M}_p$ とする．

2.1 クラスの定義から求められる集合の定義

クラスにはデータとそれをアクセスするためのメソッドが定義されている．混同を避けるために本論文では，クラスを角の丸いボックスで表し，ボックスの最上位の欄からクラス名，データの定義，そしてメソッドの定義の順に列挙する（図1）．クラスに定義したデータをデータメンバと呼び，その型は基本型だけでなく，他のクラスの場合もある．また，クラスの継承関係によって各クラスにはその親クラス（祖先クラス）が定義される．以上のことに基づき，任意のクラス C に対して，次の集合を定義する．

$OB(C)$：クラス C から生成されたオブジェクトの集合

ただし，クラス C から生成されたオブジェクトを，クラス C のインスタンスとも呼ぶ．

$MD(C)$：クラス C のデータ定義にメンバとして宣言された変数の型がクラスであるとき，それらのクラスの集合

以降，簡単のため，$MD(C)$ に該当するクラスをクラス C のデータメンバのクラスと呼ぶ．

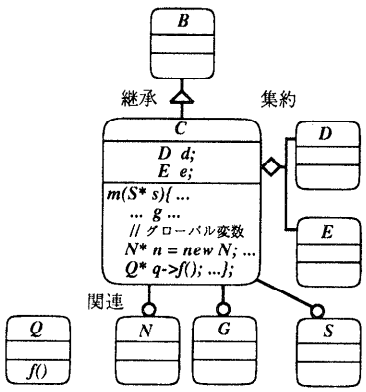


図1 クラス間の関係の例
Fig. 1 A relationship between some classes.

$MF(C)$：クラス C に定義されたメソッドの集合

ただし，$MF(C)$ と $MD(C)$ には継承されたメソッドやデータメンバのクラスは含まれない．

継承に関する定義を行うために，クラス C がクラス P を直接に継承することを $P \gg C$ または，$P \overset{1}{\gg} C$ と表し，クラス P から n 回の継承を経てクラス C が得られた場合は $P \overset{n}{\gg} C$ と表す．また，クラス C がクラス P を0階層以上にわたって継承した場合には $P \overset{*}{\gg} C$ と表す．これに基づいて，クラス C の祖先クラスの集合 $SC(C)$ を次のように定義する．

$$SC(C) \triangleq \{P \in \mathcal{C}_p \mid P \overset{*}{\gg} C\}, \text{where} C \in \mathcal{C}_p$$

また，任意のメソッド m に対して，メソッド m が定義されたクラスを $CM(m)$ と定義する（すなわち，$m \subset MF(CM(m))$）．本論文では，各メソッドはそれが属するクラス名とメソッド名および引数の型の並びの組合せによって静的に一意に特定できると仮定している．従って，以降では，簡単のため，多重定義または重複定義されたメソッドは存在しないと考える．

2.2 述語の定義

任意の2つのクラスの間にメッセージのやりとりが行われたり，一方のクラスで他方のクラスのデータメンバを参照するなどのような場合，両クラスの間には相互依存関係があるという．本論文では，相互依存関係として，グローバル変数としての参照，メソッドの仮引数としての参照，そのクラスのオブジェクトの生成，メッセージの転送のみを考える．以上のことに対し任意のメソッド m に関して，次のような述語を定義する．

$ref(m, g)$：メソッド m のボディー部でグローバル変数 g を参照する

$arg(m, s)$：メソッド m の仮引数リストにオブジェクト s がある

$contain(m, N)$：メソッド m のボディー部にクラス N のオブジェクトを生成する文を含む

$send(m, f, Q)$：メソッド m 中にクラス Q のオブジェクトにメッセージ f を送る文を含む

これらの述語は静的に型付けられた言語ではプログラムテキストから真偽を決定できる．

2.3 メソッドに関するオブジェクトのクラス

任意のメソッド m に対して，そのメソッドの内部で現れたオブジェクトのクラスの集合を次のように定義する．

・メソッド m で使われているグローバル変数のクラスの集合 $GV(m)$

$$GV(m)\triangleq\{A\in\mathcal{C}_p|ref(m,g), g\in OB(A)\}$$

・メソッド m の引数リストで使われているオブジェクトのクラスの集合 $AC(m)$

$$AC(m)\triangleq\{A\in\mathcal{C}_p|arg(m,s), s\in OB(A)\}$$

・メソッド m が属しているクラスに定義されたデータメンバのクラスの集合 $IV(m)$

$$IV(m)\triangleq\{A\in\mathcal{C}_p|A\in MD(CM(m))\}$$

・メソッド m で新しく生成されたオブジェクトのクラスの集合 $CRE(m)$

$$CRE(m)\triangleq\{A\in\mathcal{C}_p|contain(m,A)\}$$

最後に，メソッド m の供給者クラスの集合 $SU(m)$ を，

$$SU(m)\triangleq\{A\in\mathcal{C}_p|send(m,f,A), f\in\mathcal{M}_p\}$$

と定義する．つまり，メソッド m の供給者クラスは，メソッド m のボディー部で呼び出されるメソッドが定義されたクラスである．

2.4 クラス間の関係

クラスの間には以下のような3つの関係がある．図1は下記のクラス間の関係の例を示す．

・継承（*inheritance*） i

$$i=\{(B,A)|A,B\in\mathcal{C}_p\wedge A\overset{*}{\gg}B\}$$

継承はクラスの間の関係であり，各子供クラスは親クラスの特性（データメンバとメンバ関数）を共有する（図では三角形で表す）．

・集約（*part_of*） p

$$p=\{(A,B)|A,B\in\mathcal{C}_p\wedge B\in MD(A)\}$$

集約は，「全体-部分」あるいは「a-part-of」と表現される関係である．それは何かの部品を表すクラスを全体の組立を表すクラスに関連づけるものである（図では菱形で表す）．

・関連（*association*） a

$$a=\{(A,B)|\exists m\in MF(A), A,B\in\mathcal{C}_p \wedge(B\in GV(m)\vee B\in AC(m) \vee B\in CRE(m))\}$$

関連は，あるクラスとそのクラスのメソッドで参照するクラスとの関係である．ここではメソッドで参照したグローバル変数のクラスと，メソッドの引数に現れてボディー部で参照したクラス，そしてメソッドで新しく生成されたオブジェクトのクラス，という3つの場合のみを関連づけられたクラスと考える（図では円で表す）．

3. デメテルの規則の定式化

3.1 Strong/Weak version の定式化

ここでは，クラス間の相互依存関係に注目して，メッセージ転送における制限指針（デメテルの規則[4]）を示す．しかし，文献4)での制限指針は，与えられたプログラムを評価するには曖昧であるため，クラス間の関係やメッセージ転送に対する定義を用いてより明確に定式化する必要がある．まず，文献4)での定義を示す．

指針1：クラスCに属しているすべてのメソッドmは次のようなクラスのインスタンスにメッセージを送ることができる．：

・クラスC

・クラスCのデータメンバのクラス
ただし，ここではクラスCに定義されたデータメンバだけを考える．つまり，クラスCの親クラスに定義されたデータメンバのクラスのインスタンスにはメッセージを送れない．

・メソッドmの引数のクラス

・メソッドmによって直接的に生成されたオブジェクトのクラス

・メソッドmで参照しているグローバル変数のクラス

以上の指針を前章の諸定義を用いて定式化する．

[**定義1**] デメテルの規則（Strong version）

プログラム $\mathcal{P}$ は

$$\forall C\in\mathcal{C}_p[\forall m\in MF(C)[SU(m)\subset(CM(m)\cup IV(m)\cup AC(m) \cup CRE(m)\cup GV(m))]]$$

を満たさなくてはならない．

すなわち，任意のクラス C とそのクラスに属しているメソッド m に対して，メソッド m からメッセージを送れるクラスとしては，メソッド m が定義されたクラス C，クラス C に定義されたデータメンバのクラス，メソッド m の引数オブジェクトのクラス，メソッド m で生成されたオブジェクトのクラス，メソッド m で使われているグローバル変数のクラスでなければならない．

さらに指針1を少し緩めて祖先クラスに定義したデータメンバのクラスのインスタンスにもメッセージ転送ができるという次のような指針が提案された[4]．

指針2：指針1に加えて，祖先クラスから継承されたデータメンバのクラスのインスタンスにもメッセージを送ることができる．

次に指針2を具体的に定式化した結果を示す．

[**定義2**] デメテルの規則（Weak version）

プログラム $\mathcal{P}$ は，

$\forall C \in \mathcal{C}_p[\forall m \in MF(C)[$

$SU(m) \subset (CM(m) \cup TIV(m) \cup AC(m)$

$\cup CRE(m) \cup GV(m))]]$

を満たさなくてはならない．

ただし，$TIV(m) = \bigcup_{A \in SC(C)} MD(A)$

すなわち，任意のクラス C とそのクラスに属しているメソッド m に対して，メソッド m からメッセージ転送のできるオブジェクトのクラスは，クラス C とその祖先クラスに定義されたデータメンバのクラス，メソッド m の引数クラス，メソッド m で生成されたオブジェクトのクラス，メソッド m で使われているグローバル変数のクラスである．つまり，Weak version では継承関係に基づいて，祖先クラスに定義されたデータメンバのクラスへもメッセージを転送できることになる．

以上の定義から Strong version は非常に厳格であり，Strong version を満たすためにはクラスに適当な変換を行う必要があることが多い．例えば，あるクラス P に定義したデータメンバのクラスのインスタンスにメッセージを送る文を含むメソッド cm がクラス P の子孫クラス C に定義された場合には，メソッド cm をクラス C からクラス P へ移動し，クラス C にクラス P のメソッド cm を呼び出す新しいメソッドを定義する方法が考えられる．

3.2 クラスの分類

多くのクラス間のメッセージのやりとりはクラス間の関係に沿って行われる．従って，供給者クラスと依頼者クラスとの間にはどのような関係で結び付いているかを知っておくことによってデメテルの規則をより明確に理解できる．本節では，メッセージ転送が行われる時，そのメッセージに関する依頼者-供給者モデルに基づき，クラスの各メソッドに対してその供給者クラスを以下の3種のクラスに分類する[5]．

知り合いクラス（$AQ\_C(m)$）

知り合いクラスは，任意のクラスに必要によってメッセージを転送する場合，メッセージのやりとりを行っている両クラスの間にクラス関係が一番薄い関係で結ばれたクラスである．以下，知り合いクラスについて定式化すると，

$$AQ\_C(m) \triangleq \{A \in \mathcal{C}_p | A \in SU(m) \wedge (A \notin SC(AC(m)) \wedge A \notin SC(IV(m)) \wedge A \notin SC(CM(m)))\}$$

のようになる．上記の式は次のように表すことができる．

$$AQ\_C(m) = SU(m) - SC(AC(m)) - SC(IV(m)) - SC(CM(m))$$

すなわち，知り合いクラスは，引数クラスでも，データメンバのクラスでも，そして，メソッド m が属しているクラスでもない供給者クラスであり，メソッド m によって生成されたオブジェクトのクラスやメソッド m で参照したグローバル変数のクラスなどが当てはまる．

優先知り合いクラス（$PA\_C(m)$）

メソッド m を起動することによってその結果として生成されるオブジェクトのクラスと，メソッドのボディー部で用いるグローバル変数のクラスは知り合いクラスの中で最も相互依存度の高いクラスであり，これらのクラスを優先知り合いクラスと呼ぶ．優先知り合いクラスを供給者クラスとするメッセージ転送の例としては，メッセージ転送を行い，その返り値のオブジェクトにさらに他のメッセージを送るといったネストしたメッセージ転送文を使う場合と，グローバル変数を宣言してプログラムの中のどこからも用いることができるようにした場合等がある．

優先知り合いクラスは以下のように定式化できる．

$$PA\_C(m) = AQ\_C(m) \cap (SC(CRE(m)) \cup SC(GV(m)))$$

すなわち，優先知り合いクラスは，知り合いクラスのうち，メソッド m で生成されたオブジェクトのクラスとメソッド m で使われているグローバル変数のクラス，またはそれらの先祖クラスである．

優先供給者クラス（$PS\_C(m)$）

任意のメソッド m に対する優先供給者クラスは，そのメソッド m が定義されたクラス C，そしてクラス C と継承，集約，そして関連といったクラス関係で関係づけられたクラス，またはそれぞれのクラスの祖先クラスであるような供給者クラスである．以上を定式化すると，次のようになる．

$$PS\_C(m) \triangleq \{B \in \mathcal{C}_p | B \in SU(m) \wedge (B \in SC(IV(m)) \vee B \in SC(AC(m)) \vee B \in SC(CM(m)) \vee B \in PA\_C(m))\}$$

上の式は次のようにも表せる．

$$PS\_C(m) = SU(m) \cap (SC(IV(m)) \cup SC(AC(m)) \cup SC(CM(m)) \cup PA\_C(m))$$

つまり，優先供給者クラスは，優先知り合いクラス，メソッド m が属しているクラスに定義されたデータメンバのクラス，メソッド m の引数クラス，メソッド

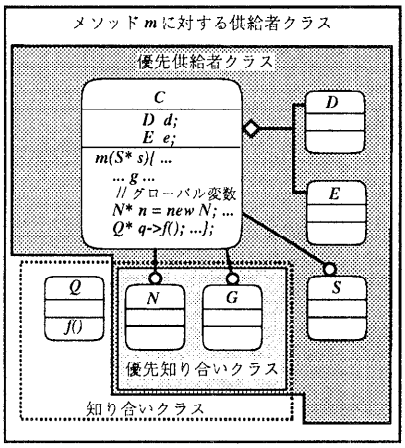


図2 クラスの分類
Fig. 2 Classification of classes.

m が定義されたクラス，またはそれらの先祖クラスであるという条件を満たす供給者クラスである．上記のクラスの分類の例を図2に示す．

3.3 Strict version の定式化

ここでは，前節で定義したクラスの分類に基づいて，任意のメソッド m からメッセージ転送のできるクラスに対する制限を定義する．

[定義3] デメテルの規則（Strict version）

プログラム $\mathcal{P}$ は，

$$\forall C \in \mathcal{C}_p[\forall m \in MF(C)[SU(m)=PS\_C(m)]]$$

を満たさなくてはならない．

すなわち，すべてのメソッドの供給者クラスは優先供給者クラスでなければならないということである．

デメテルの規則（Strict version）に従うようにプログラミングを行うと，クラス間の不必要な相互依存関係を減らすことができると言える．つまり，優先供給者クラスはメッセージ転送がなくても依存関係にあるクラスであり，デメテルの規則を満たすことによってメッセージ転送による新たな依存関係の発生をなくすことができる．また，デメテルの規則(Strict version)はプログラミング時，プログラマの助けになる．あるメソッドの実装において，プログラマはそのメソッドの優先供給者クラスの機能と仕組みだけを知っておけば良いからである．

しかし，実際には優先供給者クラスだけではなく，知り合いクラスを使わなくてはならない場合もある．例えば，実行時の効率のために優先供給者クラスでない他のクラスに直接メッセージを送る必要もある．しかし，知り合いクラスを使うことによって，知り合いクラスとその依頼者クラスとの間には余分の相互依存関係が発生し，知り合いクラスに変更があった場合にも，その依頼者クラスは影響を受ける．影響を受けるクラスはできるだけ少ないほうが望ましいので，知り合いクラスの数をなるべく少なくした方がいいと言える．

4. 判定アルゴリズム

ここまでの諸定義と定式化によって，デメテルの規則はより明確になった．また，定式化された規則を実際のプログラムに適用して，規則を満たしているかどうかの判定も可能であると考えられる．ここでは与えられたプログラムに対して，そのプログラムがデメテルの規則を満たすか否かを判定するアルゴリズムを提案する．

まず，判定アルゴリズムを記述するために，実際のプログラムに対して次のようなクラス階層グラフ **G** を定義する．

[定義4] クラス階層グラフ

$\mathbf{G}=(V, E)$

V：クラスの集合．$E=RE \cup ME$

RE：関係辺の集合．$RE \subseteq V \times V \times R$

R：$\{i, p, a\}$ i, p, a は各々継承，集約，関連を表す．

ME：メッセージ辺の集合．$ME \subseteq V \times V \times W$

W：正整数の集合．メッセージ転送文の数を表す．

与えられたオブジェクト指向プログラムはクラス階層グラフ **G** に変換できる．クラス階層グラフ **G** は，クラスを頂点とし，RE と ME の2通りの辺をもつ．

RE の辺はクラス間の関係を表し，クラス間の関係，継承(i)，集約(p)，関連(a)のうち，いずれか1つがラベルとして付けられる．任意の2頂点 v と w との間に直接の継承関係辺が存在する時 $(w \gg v)$，$(v, w, i) \in RE$ と表す．また，n 階層に渡って継承した場合 $(w \overset{n}{\gg} v)$ には $(v, w, i^n) \in RE$ と表す．同様に任意の2頂点 v, w が集約関係で結びつけられた場合 $(w \in MD(v))$，$(v, w, p) \in RE$ と表す．関連関係についても同様である．一方，ME の辺はメッセージ転送を表す．すなわち，$m \in MF(v)$ の時，$send(m, f, w)$ であれば，$(v, w, t) \in ME$ となる．ただし，t は，クラス v のメソッド m の中に書かれたクラス w へのメッセージ転送文の総数を意味する．図3にクラス階層グラフの例を示す．以下では，前節のデメテルの規則の定義とクラス階層グラフ **G** に基づいて，判定アルゴリズムを提案する．

与えられたプログラムを上記のクラス階層グラフに

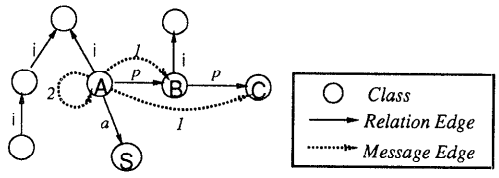


図3 クラス階層グラフの例
Fig. 3 The class hierarchy graph.

変換し，判定アルゴリズムの入力とする．アルゴリズム Demeter では，まずグラフ上の各頂点のメソッド m に対して，そのボディ部のメッセージ文から供給者クラス $SU(m)$ を求める．次にクラス階層グラフに深さ優先探索を行い，各クラス v の祖先クラスの集合 $SC(v)$ を求める．さらにすべてのクラス v について，$(v, w, p)\in RE$ または $(v, w, a)\in RE$ の関係のクラス w が存在すれば，その祖先クラスの集合 $SC(w)$ すべてと $SC(v)$ の和集合をクラス v の暫定優先供給者クラス $TPS(v)$ とする．もしすべてのクラス w について $(v, w, p)\notin RE$，かつ $(v, w, a)\notin RE$ であれば，クラス v の祖先クラスの集合 $SC(v)$ が暫定優先供給者クラス $TPS(v)$ となる．デメテルの規則 (strict version) では関連関係をメソッドごとに求めているが，本アルゴリズムでは簡単のため，クラスごとに計算している．厳密に言えば，両者は異なるが，クラス間の関係を扱うという点から言えば，これで十分である．

以上で求められた集合 $SU(m)$ と $TPS(v)$ に基づいて，定式化したデメテルの規則を用いて判定する．最後には，デメテルの規則に違反するメソッドの集合 $VD\_M(v)$ と，その違反メソッドの供給者クラスの集合 $VS\_C(m)$ を出力する．ここで，すべてのクラスに対して，集合 $VD\_M(v)$ が空集合の場合は入力プログラムは，デメテルの規則を満たすという意味で適切なスタイルのプログラムと言える．次に判定アルゴリズム Demeter を示す．

```
Demeter(G)
//入力：クラス階層グラフ G
//出力：違反メソッドの集合（VD_M(v)）と
        その供給者クラスの集合（VS_C(m)）
for each vertex v∈V[G] do
  for each method m of v do
    m のボディ部から SU(m) を求める
    VS_C(m)←φ
  VD_M(v)←φ ; SC(v)←φ ; TPS(v)←φ ;
for each vertex v∈V[G] do
  color[v]←WHITE
for each vertex v∈V[G] do
  if color[v]=WHITE then DFS(v)
for each vertex v∈V[G] do
  TPS(v)←SC(v)
  for each vertex w∈V[G] do
    if (v, w, p)∈RE∨(v, w, a)∈RE then
      TPS(v)←TPS(v)∪SC(w)
for each vertex v∈V[G] do
  for each method m of v do
    if SU(m)⊈TPS(v) then
      VD_M(v)←VD_M(v)∪{m}
      VS_C(m)←SU(m)−TPS(v)
end_Demeter

DFS(x)
color[x]←GRAY
SC(x)←SC(x)∪{x}
for each (x, y, i)∈RE do
  if color[y]=WHITE then
    DFS(y)
  SC(x)←SC(x)∪SC(y)
color[x]←BLACK
end_DFS
```

5. 変換方法のアルゴリズム化およびその有効性に関する検討

判定アルゴリズムによってデメテルの規則に違反したとわかったプログラムに対して，規則を満たすように変換するいくつかの方法のアイデア[4),5)]が提案されている．ここでは，それらの基になったリレーメソッド追加法をアルゴリズム化し，その有効性について述べる．最後にはクラス階層構造を変更する新しい変換方法を提案する．

5.1 リレーメソッド追加法のアルゴリズム化

いくつかのメッセージを並べて書く，つまり，先行のメッセージからの戻りオブジェクトにまた新しいメッセージを送る場合がある．このとき，戻りオブジェクトが依頼者側の優先供給者クラスのオブジェクトでないと，デメテルの規則に違反することになる．この場合，依頼者から供給者へメッセージをリレーする新たなメソッド（リレーメソッドと呼ぶ）を，クラス階層グラフ上の依頼者のクラスから供給者のクラスへの関係辺からなる経路上の各クラスに追加していく変換方法がある．この方法をリレーメソッド追加法と呼び，**図4** にその例を示す．

図4は図3のクラス階層を仮定している．クラス A からクラス B にメッセージ m_1 を送り，その返りオブ

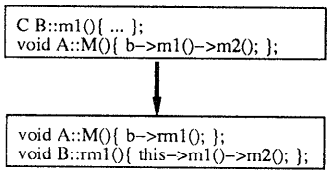


図4 リレーメソッド追加法の例
Fig. 4 Insertion of the relay method.

ジェクトにまたメソッド m_2 を送る，といったメッセージ転送が行われている．しかし，この例では，クラスCがクラスAのメソッドMの優先供給者クラスではないとすれば，デメテルの規則に違反する．

メソッドMに対して，デメテルの規則を満たすように変換を行う．まず，新しいメソッド rm_1 をクラスBに定義し，メソッドMのメッセージ文をリレーする．そして，メソッドMではクラスBにメッセージ rm_1 を送るように変更する．もしMで $->m_1()->m_2()->\cdots->m_n()$ となっている場合には，同じ方法でリレーメソッド rm_2 等を追加して以上の変換を繰り返すと，すべてのメソッドはデメテルの規則を満たすようになる．

以上の変換をアルゴリズムで示すために，まず，クラス階層グラフ上に現れるメッセージ転送経路を次のように定義する．

[**定義5**] メッセージ転送経路 mp
クラス階層グラフ上でメッセージ辺(ME)で結びつけられた任意の頂点 v_1 と v_k との間の各頂点間に存在する関係辺(RE)に沿った経路をメッセージ転送経路と呼び，$mp=\langle v_1, v_2, \cdots, v_k\rangle$ と表す($v_i=v_{i+1}$ の場合もありうる)．

次のアルゴリズムRMIはリレーメソッド追加法を実現している．

アルゴリズム RMI

1. デメテルの規則に違反するメッセージ転送文を含むメソッド m_1 を見つける．
   $C_1::m_1\{\cdots c_2->m_2()->m_3()->\cdots->m_k();\cdots\}$
   ただし，$i=1, \cdots, k$ に対して，$m_i\in MF(C_i)$ であり，$mp=\langle C_1, C_2, \cdots, C_k\rangle$ はメッセージ転送経路であると仮定する．
2. $c_2->m_2()->m_3()->\cdots->m_k()$ で $->m_i()$ はデメテルの規則を満たしているが $->m_{i+1}()$ は違反しているようなメッセージ転送(i が最小なもの)を見つける．
3. C_i に新たなメソッド rm_i を追加し，次のように定義する．
   $rm_i()\{\text{this}->m_i()->m_{i+1}()->\cdots->m_k();\}$
4. メソッド m_1 の $c_2->m_2()->m_3()->\cdots->m_k()$ の部分を $c_2->m_2()->\cdots->m_{i-1}()->rm_i()$ に書き換える．
5. デメテルの規則に違反する各メッセージ転送に対して繰り返し，以上の変換を行う．

上記のアルゴリズムでは，まず与えられたプログラムのクラス階層グラフGに対してデメテルの規則を満たすか否かをアルゴリズムDemeterで判定する．判定アルゴリズムから規則違反とわかったメソッド $m\in VD\_M(C_i)$ とそのメソッドに定義されたメッセージ転送文を用いて，判定アルゴリズムで求めた暫定優先供給者クラスの中でメッセージ転送経路上でクラス C_i と直接関係しているクラス C_j にリレーメソッド rm_j を追加する．

5.2 リレーメソッド追加法の有効性の検討

デメテルの規則に違反したプログラムは，アルゴリズムRMIによって規則を満たすように変換できるということを検討する．リレーメソッド追加法による変換では，次の定理が成り立つ．

[**定理1**] デメテルの規則に違反したプログラムはアルゴリズムRMIのステップ1の条件を満たす時，またその時に限りRMIによって規則を満たすように変換される．

(**証明**) 規則に違反するメッセージ転送を含む一連のメッセージ転送文 $c_2->m_2()->\cdots->m_k()$ を，規則に違反しないようにRMIによって変換できることを帰納法で示す．

(1) $(C_1, C_2, R)\in RE$，または $C_1=C_2$ なので，$->m_2()$ は違反していない．

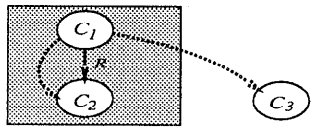
(2) $->m_j()$ (ただし，$j\leq i$) の部分では違反していないとする．$->m_{i+1}()$ が違反していれば，ステップ3，4によって，

$c_2->m_2()->\cdots->m_{i-1}()->rm_i()$
$C_i::rm_i\{\text{this}->m_i()->m_{i+1}()->\cdots->m_k()\}$

と変換される．

変換前のメッセージ転送文 $c_2->m_2()->\cdots->m_i()$ が違反していないことから，変換後のメッセージ転送文 $c_2->m_2()->\cdots->m_{i-1}()->rm_i()$ も違反していない($m_i, rm_i\in MF(C_i)$ であるから)．また，変換後の $->m_i()$ は同じクラスへのメッセージ転送なので違反していない．そして，変換後の $->m_{i+1}()$ はクラス C_i から C_{i+1} への転送なので違反していない．ゆえに，$->m_j()$ (ただし，$j\leq i+1$) は違反していないように変換される．

従って，以上の変換をすべての規則違反のメソッドに対して繰り返し行うと，与えられたプログラム $\mathcal{P}$ はデメテルの規則を満たすプログラム $\mathcal{P}'$ に変換される．




```
void C1::m1(){ ...
        c2->m2()->m3()->...->mk();
        ...
      };
```

(a) Case 1

```
x1=o->m1();
...
x2=x1->m2();
...
x3=x2->m3();
...
```

(b) Case 2

図5 変換アルゴリズムの問題点
Fig. 5 The problem on the transformation.

一方，デメテルの規則に違反するメッセージ転送文を含むメソッド $m_1 \in MF(C_1)$ に対して，$mp=\langle C_1, C_2, \cdots, C_k \rangle$ がメッセージ転送経路でないとすれば，C_i と C_{i+1} が関係辺 RE で結ばれていないような i が存在する．従って，アルゴリズム RMI を実行しても C_i のオブジェクトから C_{i+1} のオブジェクトへのメッセージ転送がデメテルの規則に反する． □

デメテルの規則に違反するプログラムに対して，規則を満たすように変換するリレーメソッド追加法をアルゴリズム RMI として定式化した．しかし，アルゴリズムのステップ1の条件を満たさない次の2つの場合，リレーメソッド追加法では変換できないと考えられる．

図5 の Case 1 でメソッド m_1 中のメッセージ転送文 $->m_3()$ が規則に違反し，クラス C_2 と C_3 との間には関係 RE が存在しない場合 ($(CM(m_2), CM(m_3), R) \notin RE$) がある．この場合はアルゴリズム RMI でのように $CM(m_3)$ にリレーメソッドを追加し，m_1 のメッセージ転送文を変更しても違反は解決しない．Case 2 のように一時的に局所変数に退避する場合は，クラス間の関係を変えないとすれば，実行時にメソッドをリレーするオブジェクトを特定できないため，実現できない．つまり，メソッド m_2 への呼出時に返りオブジェクトをそのクラスのデータメンバに格納 ($CM(m_3) \in MD(C_2)$) するなどしておけば可能であるが，これはクラス間の関係を変更することになる．

5.3 クラスの再構成

前節では，リレーメソッド追加法でクラス間の関係を変更することなしにデメテルの規則の違反を解消できない場合があることが分かった．一方で，要求仕様の変化などによって既存のクラスの構造とクラス階層を積極的に再構成する場合がある[7]．ここでは，集約関係のあるクラス間のメッセージ転送においてデメテルの規則違反が発生したクラスとそのメッセージ転送経路上のクラスとのクラス階層を変更し，規則を満たすように再構成する方法を提案する．

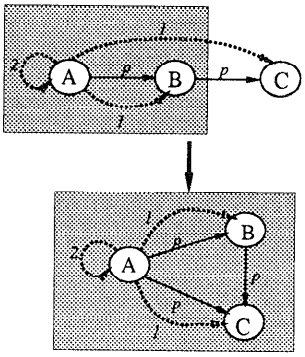


図6 クラス間の新しい関係の追加
Fig. 6 Refactoring by inserting new class relationships.

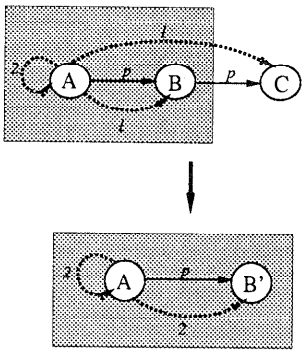


図7 クラスの吸収
Fig. 7 Refactoring by absorbing class.

5.3.1 新しい関係の追加

集約関係のクラス間でデメテルの規則に違反した場合，規則違反の供給者クラスを依頼者クラスと関係づけて優先供給者クラスにする方法が考えられる．**図6** に集約関係のクラス階層上に新しいクラス関係を追加した変更例を示す．

5.3.2 クラスの吸収

集約関係では，場合によっては新しいクラスを付け加えたり，あるいは集約関係の両クラスを合併したりすると意味的により明確になることもある．特に，集約関係のクラス階層では，集約関係の2つのクラスを合併して新しいクラスに再構成することも考えられる．**図7** は上記の変換例を示す．

図7で，クラスAからクラスCへメッセージを送るのは，クラスCがクラスAのメソッドの優先供給者クラスではないため，デメテルの規則に違反している．つまり，クラスAからCへメッセージを送るためには，クラスBにリレーメソッドを追加しなければなら

ない．しかし，クラスBとCに対して合併による再構成を施し，新しいクラスB'に置き換えれば，デメテルの規則に違反したメッセージ転送は解消される．

6. まとめ

本論文では，クラス間の関係とメッセージ転送などに関する定義を行い，既に提案されたデメテルの規則をより形式的に定式化した．その結果，デメテルの規則をより一層明確に理解できるようになった．また，デメテルの規則と諸定義に基づいて与えられたプログラムがデメテルの規則を満たすか否かを判定するアルゴリズムを提案し，違反したプログラムを変換する方法をアルゴリズム化して，その有効性について検討を行った．その結果，リレーメソッド追加法が有効であるための必要十分条件が定理1から明らかになった．

従って，デメテルの規則を満たす，品質の良いソフトウェアを構築するためのより具体的な設計方法が必要である．また，デメテルの規則を満たすようにプログラムを変換する場合，クラス構造と階層の再構成に伴う適切な再構成方法が必要である．今後の研究方向としては，オブジェクト指向におけるクラスの構造および階層の再構成に関する研究などがあげられる．

参考文献

1) Henderson-Sellers, B: *A Book of Object Oriented Knowledge : Object Oriented Analysis, Design, and Implementation : a New Approach to Software Engineering,* Prentice Hall (1992).
2) Monarchi, D. E. and Puhr, G. I.: A Research Typology for Object-Oriented Analysis and Design, *Comm. ACM,* Vol. 35, No. 9 (1992).
3) 黄，梁，辻野，都倉：オブジェクト指向プログラミングにおけるデメテル法則の定式化，情報処理学会ソフトウェア工学研究会，SE 96-5 (1994).
4) Lieberherr, K. J., Holland, I. and Riel, A. J.: Object-Oriented Programming: An Objective Sense of Style, *Proc. Object-Oriented Programming Systems, Languages, and Applications Conf.,* pp. 323-334, ACM, New York (1988).
5) Lieberherr, K. J. and Holland, I.: Assuring Good Style for Object-Oriented Programs, *IEEE Software,* Vol. 6, No. 5 (1989).
6) Sakkinen, M: Commcnts on the Law of Demeter and C++, *SIGPlan Notices,* Dec., pp. 38-44 (1988).
7) Johnson, R. E. and Opdyke, W. F.: Refactoring and Aggregation, *Proceedings of International Symposium on Object-Oriented Technologies for Advanced Software (ISOTAS'93),* Kanazawa, Japan (1993).
8) Wirfs-Brock, R. and Wilkerson, B.: Object-Oriented Design: A Responsibility-Driven Approach, *Proc. OOPSLA'89,* pp. 71-75 (1989).
9) Sharble, R. C. and Cohen, S. S.: The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods, *ACM SIGSOFT Software Engineering Notes,* Vol. 18, No. 2 (1993).
10) 梁，辻野，都倉：オブジェクト指向言語のクラスの品質評価について，情報処理学会ソフトウェア工学研究会，SW 77-3 (1991).

(平成6年9月7日受付)
(平成7年2月10日採録)

黄　錫炯（学生会員）

昭和43年生．平成3年8月韓国国立江原大学自然科学部電子計算学科卒業．平成6年大阪大学大学院基礎工学研究科修士課程修了（情報工学専攻）．現在，同大大学院博士課程在学中．主にオブジェクト指向ソフトウェア工学に関する研究に従事．

辻野　嘉宏（正会員）

昭和32年生．昭和54年大阪大学基礎工学部情報卒業．昭和59年同大大学院博士課程修了．同年同大基礎工学部助手．平成元年同大基礎工学部講師．平成3年同大基礎工学部助教授（情報工学科）．工学博士．計算機言語，並行処理記述，プログラミング教育，ユーザインタフェースなどソフトウェア工学に関する研究に従事．IEEE，電子情報通信学会，日本ソフトウェア科学会各会員．

都倉　信樹（正会員）

昭和14年生．昭和38年大阪大学工学部電子卒業．昭和43年同大大学院博士課程修了．同年大阪大学基礎工学部講師．現在，同教授（情報工学科）．プログラムの技法と理論，計算機言語，VLSIの計算複雑さの理論などの研究に従事．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会，人工知能学会各会員．