ソフトウェア工学 83－2
（1992. 2. 6）

# オブジェクトインタフェースの整合化方式の提案

岸本 芳典* 小高 信人† 本位田 真一‡
ksmt@ipa.go.jp taka@ipa.go.jp honiden@ipa.go.jp

新ソフトウェア構造化モデル研究本部
情報処理振興事業協会

**Abstract**

オブジェクトは一般に部品化／再利用に適していると認識されているが、既存のオブジェクト同士が連携するためには相手オブジェクトの提供するメソッド・インタフェースを予め知っていなければならず、連携相手に合わせた修正が必要となる。そこでオブジェクトのメソッド・インタフェースをその連携相手に適合化して動的に変更し、オブジェクト間の結合／連携を容易化するインタフェース整合化方式を提案する。本方式では、個々のオブジェクトのインタフェース仕様情報を持ちかつメソッドの変更機能を持つメタ・オブジェクトと、オブジェクトの組合せ情報と適合化候補の選択機能を持つ環境によって、インタフェースの整合を図る。本稿では、整合化方式の概要を述べる。

# Reconciliation Model of Object Interfaces

Yoshinori KISHIMOTO* Nobuto KOTAKA† Shinichi HONIDEN‡
ksmt@ipa.go.jp taka@ipa.go.jp honiden@ipa.go.jp

Laboratory for New Software Architectures
Information-Technology Promotion Agency, Japan
Shuwa-Shibakoen 3-Chome Bldg. 3-1-38 Shibakoen Minato-Ku Tokyo 105, Japan

**Abstract**

Objects are considered to be suitable for reusable software components. However, existing objects that have been developed independently don't know their method interfaces each other. So, these objects cannot communicate with each other without modifying their method interfaces. In this paper we propose a dynamic method adaptation model that consists of agents and their environment. An agent consists of an object and its meta-object. A meta-object holds information about method interfaces of corresponding object and has ability to alter interfaces. Environment knows combination of objects and has ability to choose an appropriate one out of adaptation candidates offered by a meta-object.

* （株）日立製作所より出向
† 富士通エフ・アイ・ビー（株）より出向
‡ （株）東芝より出向

*Also with Hitachi Ltd.
†Also with Fujitsu Facom Information Processing Corp.
‡Also with Toshiba Corp.

# 1 動機と目的

## 1.1 オブジェクト指向の課題

オブジェクト指向技術は、プログラムとデータとをカプセル化するデータ抽象化機構と、類似した性質を他のオブジェクトから受け継ぐ継承機構などのため再利用性が高いと考えられている [Nierstrasz89] [Wegner90]。種々提案されているオブジェクト指向言語では、用意されている豊富なライブラリや以前に作成したオブジェクトを利用して新たなオブジェクトを作ることにより容易にいわゆる差分プログラミングができる。しかしながら既存のオブジェクト同士を利用して新たなソフトウェアを作る場合には、オブジェクトが提供するメソッドに合わせたメッセージを送る、あるいは送られるメッセージに合わせてメソッドを変更する、などのメソッド整合化作業が必要である。ここで「既存のオブジェクト」とは、独立して作成されたオブジェクトのことを意味し、それぞれの作成時点では相手が提供するメソッドはもちろん、相手の存在さえも互いには知らないオブジェクトである。これはいわゆる「知人問題 (acquaintance problem)[Tsichritzis89]」の一つである。

知人問題には二種類の課題があると考えられる。一つは相手オブジェクトの存在そのもの（相手の名前）をどうやって知るかということであり、もう一つはその相手オブジェクトとの交信をどうやって確立するかということである。メソッドの整合化はこの後者に相当する。知人問題はオブジェクトの開放性 [Meyer88] に起因する。すなわち、オブジェクトの作成時点で全ての相手オブジェクトや用途を知ることは不可能であり、従って将来の拡張・変更は不可避である。この問題は、システムが大規模化・分散化するに伴っていっそう顕在化するものと考えられる。特に分散環境においては、

- 相手オブジェクトのバージョン変更
- ハードウェアの障害による動的機能代替
- 相手オブジェクトの移動 (migration) にともなう相手の変更
- 自オブジェクトの移動にともなう環境変化

などのオブジェクトの外的要因によって交信相手が変り得るが、このような時でもシステムを再構築することなく新しい環境に適応してシステムが継続して動けることが要望される。オブジェクトが移動した場合の対応として、

- 移動前と同等の交信を維持する
- 移動後の環境に適合して新たな交信を開始する

の２通りが考えられる。従来から行なわれているオブジェクト移動の研究では、オブジェクトのアイデンティティ保存やオブジェクト間交信の透明性などの前者を対象としたものが中心である。しかしながら、オブジェクトが移動してきた環境において、環境に元からいたオブジェクトと移動したオブジェクトとが新たに交信を始める場合や、あるオブジェクトが出ていった環境に残された他のオブジェクトが残りのオブジェクト同士で新たに交信を始める場合については考慮されていない。また従来のオブジェクト移動では、ハードウェア障害のように交信が継続できない状況には対処できない。このような場合には、既存の未知のオブジェクト同士の交信が必要となる。

メソッド整合化方法は、これを部品合成問題ととらえてオブジェクトの組合せ時に予めオブジェクトを修正する方法と、メソッドの動的束縛問題ととらえてオブジェクトの実行時に整合化する方法とが考えられる。我々は、後述する理由から、これを動的束縛問題ととらえ、後者のアプローチをとる。

メソッドの整合化は、あるオブジェクトから送られたメッセージがそれを受けたオブジェクトには理解できない時に必要となる。既存オブジェクト間の整合化には種々のレベルが考えられる。

1. メソッド適合 (method adaptation)
   受け手オブジェクトの持つメソッドで解釈できるようにメッセージを変換する
2. メソッド合成 (method composition)
   メッセージを理解できるようなメソッド（機能）を新たに追加する
3. オブジェクト適合 (object adaptation)
   メッセージの理解に必要な属性の変更・追加も含めてメソッドを新たに追加する
4. オブジェクト合成 (object composition)
   メッセージを理解できるようなオブジェクトを新たに作る

整合化は上述の順に難しくなる。一方、整合化を動的に行なうためには、

1. 変更方法（内容）を決定する
2. 変更をオブジェクトに反映させる

の二種類の技法が必要である。後者についてはリフレクション技法 [Maes88] や Music（後述）の機構などで実現できるが、前者については個々の既存オブジェクトの持つ意図、あるいは既存オブジェクトの組合せにおける各オブジェクトの用途などに基づく判断機構が必要である。

## 1.2 他の研究アプローチと問題

オブジェクトの再利用性・拡張性向上、動的メソッド変更・獲得を目的とした研究は種々行なわれている。そのいくつかについて以下に考察する。

*Music*[Watari90] [Honda90] [Honda91] は、計算が進むにつれて、オブジェクトが新しい属性／動作を獲得する「オブジェクトの成長」を基礎とした新しいオブジェクト指向データ／計算モデルである。オブジェクトがどのような実行環境で使われるかは事前には決定できないため、オブジェクトは環境に依存した動作を必要に応じて環境から獲得できる必要がある。本モデルは、オブジェクトの成長メカニズムと獲得した属性／動作の使い分けメカニズムから成る。オブジェクトの生成は、オブジェクトはテンプレート（クラスに相当）を用いて生成する。オブジェクトの成長は、スロット（オブジェクトの属性）に指定するテンプレートに対して、新たなスロット／制約の追加を可能にする。これにより開放システムにおけるオブジェクトの動的な性質を記述可能となる。

*CAM(Class Attachment Model)*[Umemura91] は、インスタンスの生成プロセスを、実体の生成とクラスの付与との2ステップに分離することにより、オブジェクトの振舞いを定める前にオブジェクトの実体を生成するモデルである。これにより、表現力が向上し、次第に多面的になるモデルの記述に有効である。本モデルでは、オブジェクトを、データそのもの (entity) と、メソッド (class) とに分離し、クラスとエンティティーとの対応をフェイス (face) により指定する。フェイスには、クラスが仮定している変数名と、エンティティーが持っている変数名との対応関係が指定されており、これによりクラスからエンティティーが参照できるようになる。従ってオブジェクトのインタフェースは、クラスのみによって決まる。クラスが必要とするデータ（変数）は予めエンティティーに備わっていなければならない。フェイスは、対応させたいエンティティーとクラス（の各々の変数名）に基づいて作成する必要がある。

*ABCL/R*[Watanabe88] は、リフレクション機能を持ったオブジェクト指向言語である。各オブジェクトに対応して存在するメタ・オブジェクトが指定した他のオブジェクトが持っているメソッドを実行時に問い合わせ、リフレクションによりそのメソッドを動的に獲得できる。

Music や CAM 、ABCL/R は、オブジェクトのインタフェースを動的に変更（追加）する基盤機構となり得るが、変更内容を決定するための機構については未解決である。

*Vieu*[Hailpern90] は、あるオブジェクトが他のオブジェクトに対して公開するインタフェースを相手によって変更することを目的としている。オブジェクトのメソッド毎にそのメソッドを使用可能な相手オブジェクトを指定するビュー (view) を定義しておくことにより、オブジェクトに対するアクセスを制限する。

View では、必要なメソッドは予めオブジェクトに備わっていることを前提としているため、交信相手に適合してインタフェースを動的に変更することはできない。

*OCD(Object Complex Descriptor)*[Morley91] は、複数のオブジェクトの複合体 (object complex) を単位としたオブジェクトの再利用を行なう機構を提案している。オブジェクト複合体はオブジェクトの設計者が持つオブジェクトの概念構造に相当する。OCD では、オブジェクトの再利用は具体的なオブジェクト単位ではなく、オブジェクト群が表現する概念を単位として行なわれることに基づいている。

この方式は、知人問題の第一の課題に対するものであり、第二の課題に対しては未解決である。

一方従来からのソフトウェア・モジュールの再利用、合成に関する研究も種々行なわれている [Wileden90] [Purtilo91] [Kishimoto91a] が、いずれもモジュールインタフェースのアダプタを静的に生成するものである。

## 1.3 目的の例示

本研究では、既存オブジェクトの動的整合化モデルを確立することを目的とし、その第一歩として動的なメソッド適合化モデル OMEGA[1]を提案する。動的メソッド適合化を行なうためには、オブジェクト自身が（可能な限り）自主的に適合化方法を判断し、自身を変更することが重要である。そこで OMEGA モデルでは、各オブジェクトにリフレクション機能を持ったメタ・オブジェクトを対応させ、メタ・オブジェクトの自律的計算によってメソッドの適合化を図る。すなわちオブジェクトとそのメタ・オブジェクトとの複合体が一種のエージェントを構成する。

この最終イメージを簡単な例で示す。システムは、前述のエージェント、すなわちオブジェクトとメタ・オブジェクト、およびオブジェクトの組合せや適合化候補の選択などをメタ・オブジェクトに教える環境から構成する。図1にメソッドの適合化過程を示す。この例では、双方向リンクを持ったn分木オブジェクト DLNT(Double Linked N-ary Tree) と、単方向リンクを持った2分木オブジェクト SLBT(Single Linked Binary Tree) 、および木オブジェクトを使用するクライアント・オブジェクト C があり、C は元々同じノード上の SLBT にメッセージを送っていたが使用環境の変化、例えば C が他のノードに移動してしまったため、C は移動後のノードにある DLNT にメッ

[1] OMEGA: Object MEthod Gap Adaptation

セージを送るようになった[2]ものと仮定する。

(A). C はメッセージ insert_left（SLBT には理解できる）を DLNT に送る。

(1) DLNT は C から送られたメッセージが理解できないので、DLNT のメタ・オブジェクト mDLNT にメソッド整合化を依頼する。

(2) mDLNT は、C のメタ・オブジェクト mC にメッセージの意図を質問する。

(3) mC は、DLNT にメッセージを送った理由を環境に対して問い合わせる。

(4) 環境は、DLNT は SLBT の代わりに使われたことを返答し、mC はこれを mDLNT に伝える。

(5) mDLNT は、SLBT のメタ・オブジェクト mSLBT に対してメソッド insert_left の機能を質問する。

(6) mSLBT は、メソッドの機能、パラメータ仕様などを返答する。

(7) mDLNT はメソッド insert_left を代行可能な DLNT のメソッドとして insert を発見し、メッセージ insert_left を insert に変換する方法が2種類あることを知る。mDLNT は現在の DLNT の使用環境においてどちらが適切かを環境に対して問い合わせる。

(8) 環境は一方を選択する。

(9) mDLNT は、選択に基づいて DLNT にアダプタ・メソッドを追加する。

(B). DLNT は、メッセージ insert_left を受理し、メソッド insert を実行する。

ここで前述のようにオブジェクト C が他ノードに移動したことを仮定した場合には、ステップ (5) において mDLNT は他ノード上の mSLBT と遠隔交信しなければならないが、これは適合化のために1回だけ必要なものであって、オーバヘッドはさほど問題にならないと考えている。また C が移動するにあたって、予め C の交信相手（SLBT）のメソッド情報を一時的に取り込んで移動するようにすれば、ステップ (5) において mDLNT は mSLBT ではなく mC とのみ交信すればよいため、遠隔交信は必要なくなる。

このように、メソッドの整合化を動的（実行時）に行なう方式には、オブジェクトを事前に修正して検査する静的方式に比較して次のメリットがある。

[2] C がいかにして DLNT の存在を知るか、は知人問題の第一の課題でありここでは触れない。

1. オブジェクトの内部状態（現在の属性値）に基づいた整合化が可能である。

2. メッセージ（パラメータ）の具体値に基づいた整合化が可能である。

3. オブジェクトの修正方法を見つけるための試行錯誤に適している。

4. オブジェクトのソースコードが無くても整合化可能である。

## 2 メソッドの適合化対象

ここではまずメソッド適合化の対象要素について考察する。メソッド・インタフェースの構成要素は、メソッド識別、連携手段、パラメータ、の3つに大別できる。

### 2.1 メソッド識別

オブジェクト間のメッセージ通信では、受信したメッセージに対応するメソッドが実行される。既知のオブジェクト間の交信では、利用可能なメソッドを予め知っておき、それに合わせたメッセージを送る。しかしながら未知のオブジェクト間交信ではこれは不可能である。考えられる方法は2つある。

一つの方法は、送信側のオブジェクトは、自分が知っているオブジェクトのつもりで相手にとりあえずメッセージを送ることである。受信側のオブジェクトにとって、そのメッセージはおそらく未知のものである。偶然に自分が提供しているメソッドと一致することもあるだろうが、そのメソッドが持つ機能とメッセージが意図する機能とが一致する保証はない。すなわち、メソッド名の同名異義、別名同義の問題が発生する。従って受信側オブジェクトはメッセージの送り手の意図を考慮して適切なメソッドを選択する必要がある。

もう一つの方法は、送信側オブジェクトが予め相手の提供するメソッドの機能を調べて適切なメソッドを選択し、それに合わせたメッセージを送ることである。但し本質的には、メッセージとメソッドとの意図を照合する、という点において両者は同等の問題を持つものである。

実現方法に関しては、前者は受信オブジェクトの既存メソッドは変更せずに新たなメソッドを追加するだけでよいのに対して、後者はメッセージを送信するメソッド自体を変更（修正）する必要がある、という点が異なる。メソッドの修正には副作用（他のメッセージ通信に影響を与える）のおそれもある。従って本研究では前者のアプローチをとる。

### 2.2 連携手段

オブジェクト間の交信はメッセージ通信のみである。従って基本的な連携手段は一致している。しかしながらメッセージ通信は細部において異なることがある。すなわちメッセージの種々のバリエーションである。具体的には、同期メッセージ／非同期メッセージ、メソッドのリターン値の有無、現在型メッセージ／未来型メッセージ[Yonezawa86]、などがあり、個々のプログラミング言語により種類は異なる。

これらの種類には、本質的に互換性のないものと、代替可能なものとがある。例えば、リターン値を必要とする関数型メッセージはリターン値のない手続き型メソッドでは代行できないが、逆に手続き型メッセージは関数型メソッドで代行可能である。メッセージの種類は、先にも述べたように、プログラミング言語に依存する。従って全ての種類やそれらの間の代行可能性を予め知ることはできない[3]。代行可能性は必要に応じて追加定義しなければならない。

### 2.3 パラメータ

上述のメソッド識別と同様に、メソッドのパラメータについても送信されたメッセージに含まれるパラメータと、対応する（メソッド識別により判明した）メソッドの要求するパラメータとが一致する保証はない。具体的には、パラメータの個数、順序、型、が異なる可能性がある。

順序の違いは並べ変えればよい。個数が異なる場合には、過剰パラメータの無視、不足パラメータの補充、などが必要である。パラメータの順序や過不足を判断するためには、各パラメータの役割を知らねばならない。これには、受け手側のオブジェクトのメソッドにおける各パラメータの役割と、送り手側のオブジェクトのメッセージにおける各パラメータの役割とが必要である。但し後者は、送り手側オブジェクトが元々送ろうとしていた相手オブジェクトのメソッドにおける各パラメータの役割である。従ってこれは送り手側オブジェクトに依存する情報ではなく、元々の受け手側オブジェクトに依存する情報である。

型については、オブジェクト型とオブジェクト以外のデータ型の場合で扱いが異なる。オブジェクト型の場合、メッセージのパラメータがメソッドのパラメータの一種（is-a 関係）であれば問題はない。is-a 関係でない場合には、オブジェクトの「互換性」が問題であるが、これはオブジェクトの属性とメソッドが互換か否かである。今（本研究では）属性については除外しているので、これはメソッドの互換性に他ならない。メソッドの互換性とはメソッドの適合化が可能か否かである。これについては、パラメータになっているオブジェクトが使われた（メッセージを送られた）時点で判断されるべきである。従ってパラメータがオブジェクト型の場合には、型の相違は考えなくてよいことになる。一方オブジェクト以外のデータ型の場合は、いわゆる型変換が可能か否かにより適合可能かどうかが決まる。

[3]先に述べた「プログラムの開放性」に対して、これは「プログラミング言語の開放性」ともいえるだろう。

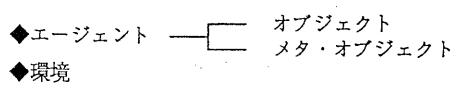
## 3 OMEGA モデル

### 3.1 モデルの構成要素

第 1.3 節で述べたように、OMEGA モデルはエージェント自身がメソッド適合化を行なうモデルである。エージェントは必要に応じて自分のメソッドに関する情報を検索したり、メソッドに関する他のエージェントからの問い合わせに応答し、さらにメソッド適合化方法を決定し、メソッド・インタフェースを変更する。このためにはオブジェクト自体に関する計算、すなわちメタ・レベルの計算が必要である。またメソッドの適合化方法は、クラスに共通とは限らず、各インスタンス・オブジェクト毎に異なることがある。そこで各オブジェクトに対応するメタ・オブジェクトを用いることにした。

エージェントがメソッド適合化を行なうためには、第 2 章で述べた種々の判断をエージェント[4]自身ができなければならない。但し全てをエージェントが自主的に判断することはかなり困難である。例えば、あるオブジェクトが従来交信していた相手オブジェクトがなぜなくなったのか、どこへ行ったか、あるいは複数の判断が可能な時にどれを選択するか、などは、エージェント自身に閉じたものではない。むしろこれらの情報、判断は、エージェントの外部から与えられるべきものである。我々はこれをエージェントの「環境」としてエージェントとは切り離すことにした。環境自体もオブジェクトとは別次元のエージェントであることが望ましい[5]が、現在の OMAGA モデルではこれは人間（オブジェクトの利用者）が代行するものとして環境自体のモデル化は行なっていない。

以上をまとめると、OMEGA モデルは以下の要素から構成される。図 2 に本モデルの構成を BNF 風に示す。




[4]具体的には、エージェントを構成するメタ・オブジェクトである。
[5]少なくとも現状の技術の組合せだけでは不可能であろう。

### 3.2 メタ情報の記述形式

メタ・オブジェクトは対応するオブジェクトの各メソッドに関する情報を持つ。この情報とは、第2章で述べたメソッド識別、連携手段、パラメータ、に関する情報で、原則的には各オブジェクトの作成時に静的に定まるものである。ただしメソッド適合化により動的に生成されたアダプタ・メソッドに関する情報は、動的に追加される。メソッドに関する情報を我々は作成者意図(*PI: Producer's Intention*)と呼ぶ。一方環境は、オブジェクトの交信相手の変更に関する情報と、適合化候補の選択情報とを持つ。この情報を我々は**環境情報**(*EI: Environmental Information*)と呼ぶ。作成者意図はメタ・オブジェクト内に保持されているが、環境情報はメタ・オブジェクトから環境への問い合わせ発生時点で作成[6]される。

作成者意図の記述形式を図3に示す。同図において「語彙」とは、メソッドの機能説明やパラメータ用途説明に用いる用語を定義するものである。語彙はオブジェクトのメソッドの説明に用いる用語であり、かつメソッドはクラス階層構造に従って継承されるため、語彙にも階層関係(図3では**include**)が定義できるようにした。これにより同義語(synonym)と異義語(homonym)が発生する可能性があるため、これらを記述可能にした。

ところで、語彙の同義語や異義語の定義は、作成者意図の記述時に必要なものと、オブジェクトの組合せ時に必要なものとがある。後者は、既存オブジェクト同士を結合する時点でメタ・オブジェクトがそれらの作成者意図を比較しても、用語の違いから対応がとれない場合に必要となる。ただし同義語の候補を絞り込むためには、用語のいわゆる「意味」や我々が一般に持っている「常識」が必要であり、メタ・オブジェクトにこの機能を持たせることは現状では考えていない。このような理由から、オブジェクト組合せ時の同義語／異義語定義は現状ではできない。

また図4に図1に示した例題の使用者意図の関連部分の記述例を示す。

### 3.3 メタ・プロトコル

メソッド適合化を行なうためには、メタ・オブジェクト間、およびメタ・オブジェクトと環境との間での作成者意図および環境情報の交換が必要である。これを行なうために次のようなプロトコルを用いる。

1. メターメタ間プロトコル

   (a) 原送信先 (original-destination)
   メッセージの受信側が送信元に対して元々の送信相手を問い合わせる

   (b) メソッド仕様 (method-specification)
   あるメソッドの機能、手段、パラメータ仕様を問い合わせる

   (c) ヒント (hint)
   類似したメソッド適合化が過去にあるかを（主として同一クラスの他のメタ・オブジェクトに対して）問い合わせる

2. メター環境間プロトコル

   (a) 代用 (substitution)
   オブジェクトの元々の交信相手、および代用に付随する制約条件を問い合わせる

   (b) 候補選択 (select-candidates)
   メソッド適合化方法の候補の内どれが適切かを問い合わせる

[6]現在のOMEGAモデルでは、使用者がその時点で作成する。

## 4 考察

### 4.1 OMEGAモデルの妥当性

本モデルでは、できるだけエージェントの自主的な判断によってメソッド適合化を行なうことを目標としているが、完全な自動化を目指したものではない。すなわちエージェントに全ての知識を持たせるのではなく、必要に応じて積極的に外部（環境）に問い合わせることを許す。またエージェントの稼働環境の解釈には高度の知能／知識が必要であるため、環境自体のモデルはあまり詳細化していない。これは我々が革新的(innovative)なものではなく一歩前進的(progressive)なものを目指したからであって、その点で妥当なものと考えている。

### 4.2 OMEGAモデルの適用限界

OMEGAモデルは、第1.1節で述べたように、4つに分類したオブジェクト整合化の内のメソッド適合化にしか対処していない。従って既存オブジェクトに新たにメソッドを追加するような変更はできない。

メソッド識別については、メソッドの機能を完全に形式仕様化するのではなく、語彙定義された用語のみを用いる比較的簡単な機能説明文に頼っている。この説明文自体がメソッドの機能と無矛盾である保証はない。

また第3.2節でも述べたように、動的な語彙の動議語／異義語定義ができないため、メソッド識別やパラメータの対応付けが不十分になることがある。

## 5 まとめ

既存のオブジェクト間の交信をいかにして確立するか、という知人問題の第二の課題に対して、エージェント（オブジェクトとメタ・オブジェクトの複合体）と環境との対話により動的なメソッド適合化を行なうOMEGAモデルを提案した。

第 4.2 節で述べた問題点を解決するためには、機能の抽象化 [Matsuura92] が不可欠である。またメソッド合成やオブジェクト適合も可能なモデルに拡張するためには、設計結果として実在するオブジェクトだけでなく、設計過程での判断、失敗例、および中間生成物などのソフトウェア・プロセスを定式化 [Kotaka91b] [Kotaka92a] し、これを利用する必要があると考えている。今後これらの成果を踏まえてモデルの拡張を行なう予定である。

［謝辞］

本研究は、次世代産業基盤技術研究開発「新ソフトウェア構造化モデルの研究開発」の一環として情報処理振興事業協会が新エネルギー・産業技術総合開発機構から委託をうけて実施したものである。

## 参考文献

[Hailpern90] Hailpern, B., Ossher, H.: "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Trans. Softw. Eng.*, Vol. SE-16, No. 11, pp. 1247-1257, Nov., 1990.

[Honda90] Honda, Y., Watari, S., Osawa, E., Tokoro, M.: "A Model for Growing Objects," *7th Conf. Proc. Japan Soc. Softw. Sci. and Tech.*, pp. 277-280, Oct., 1990. (in Japanese)

[Honda91] Honda, Y., Watari, S., Osawa, E., Tokoro, M.: "Compositional Adaptation: A New Method for Constructing Software in Open-Ended Systems," *Sony Comput. Sci. Lab. Tech. Report SCSL-TR-90-012*, Jun., 1991.

[Kishimoto91a] Kishimoto, Y., Yamano, K.: "SOFTON: A Flexible Software Construction Model by Interface Mediation," *Proc. 15th Int. Comput. Softw. Appl. Conf. (Compsac91)*, pp. 479-486, Sep., 1991.

[Kishimoto91b] Kishimoto, Y., Kotaka, N., Honiden, S.: "Reconciliation of Object Interfaces," *Proc. 43rd Annual Convention IPS Japan*, 3K-5, pp. 5:435-436, Oct., 1991. (in Japanese)

[Kotaka91a] Kotaka, N., Kishimoto, Y., Honiden, S.: "Formalization of Object-Oriented Analysis," *Proc. 43rd Annual Convention IPS Japan*, 3K-6, pp. 5:437-438, Oct., 1991. (in Japanese)

[Kotaka91b] Kotaka, N., Kishimoto, Y., Honiden, S.: "Specification Process Modeling in OOA," *TOOLS 6 (Proc. 6th Tech. OO Lang. Sys.)*, pp. 67-81, Prentice Hall, 1991.

[Kotaka92a] Kotaka, N., Kishimoto, Y., Honiden, S.: "Case-Study Based Specification Process Modeling for Object-Oriented Analysis," *Preprints Work Gr. for Softw. Eng., IPSJ*, Vol. 92-SE-83, Feb., 1992. (in Japanese)

[Maes88] Maes, P., Nardi, D.: *Meta-Level Architectures and Reflection*, North-Holland, 1988.

[Matsuura92] Matsuura, S., Honiden, S.: "Abstract on Formal Specification Language," *Preprints Work Gr. for Softw. Eng., IPSJ*, Vol. 92-SE-83, Feb., 1992. (in Japanese)

[Meyer88] Meyer, B.: *Object-Oriented Software Construction*, Prectice-Hall, 1988.

[Mili91] Mili, H.: "SoftClass: An Object-Oriented Tool for Software-Reuse," *TOOLS 5 (Proc. 5th Tech. OO Lang. Sys.)*, pp. 303-317, Prentice Hall, 1991.

[Morley91] Morley, D., Chiu, S., Robbins, J., Maddux, T., Voelker, G.: "Reusable Objects," *TOOLS 4 (Proc. 4th Tech. OO Lang. Sys.)*, pp. 237-248, Prentice Hall, 1991.

[Nierstrasz89] Nierstrasz, O.: "A Survey of Object-Oriented Concepts," *Object-Oriented Concepts, Databases, and Applications (ed. Kim, W., Lochovsky, F. H.)*, pp. 3-21, ACM Press, 1989.

[Nierstrasz90] Nierstrasz, O., Papathomas, M.: "Viewing Objects as Patterns of Communicating Agents," *ECOOP/OOPSLA'90 Conf. Proc.*, pp. 38-43, Oct., 1990.

[Purtilo91] Purtilo, J. M., Atlee, J. M.: "Module Reuse by Interface Adaptation," *Softw. Prac. Exper.*, Vol. 21, No. 6, pp. 539-556, Jun., 1991.

[Raj89] Raj, R. K., Levy, H. M.: "A Compositional Model for Software Reuse," *Comput. J.*, Vol. 32, No. 4, pp. 312-322, 1989.

[Sullivan90] Sullivan, K. J., Notkin, D.,: "Reconciling Environment Integration and Component Independence," *SIGSOFT90: 4th Symp. Softw. Dev. Env.*, ACM SIGSOFT S.E. Notes, Vol. 15, No. 6, Dec., 1990.

[Tsichritzis89] Tsichritzis, D. C., Nierstrasz, O. M.: "Directions in Object-Oriented Research," *Object-Oriented Concepts, Databases, and Applications (ed. Kim, W., Lochovsky, F. H.)*, pp. 523-536, ACM Press, 1989.

[Umemura91] Umemura, K.: "CAM: Class Attachment Model," *8th Conf. Proc. Japan Soc. Softw. Sci. Tech.*, pp. 213-216, Sep., 1991. (in Japanese)

[Watanabe88] Watanabe, T., Yonezawa, A.: "Reflection in an Object-Oriented Concurrent Language," *OOPSLA'88 Conf. Proc.*, pp. 306-315, Sep., 1988.

[Watari90] Watari, S., Osawa, E., Honda, Y., Tokoro, M.: "Towards Music: A Description Language for the Muse Object Model," *TOOLS 2 (Proc. 2nd Tech. OO Lang. Sys.)*, Jun., 1990.

[Wegner90] Wegner, P.: "Concepts and Paradigms of Object-Oriented Programming," *ACM OOPS Messenger*, Vol. 1, No. 1, pp. 7-87, Aug., 1990.

[Wileden90] Wileden, J. C., Wolf, A. L., Rosenblatt, W. R., Tarr,P. L.: "Specification-Level Interoperability," *Proc. 12th Int. Conf. Softw. Eng.*, pp. 74-85, 1990(also in *Comm. ACM*, Vol. 34, No. 5, pp. 72-87, May, 1991).

[Yonezawa86] Yonezawa, A., Briot, J., Shibayama, E.: "Object-Oriented Concurrent Programming in ABCL/1," *OOPSLA'86 Conf. Proc.*, pp. 258-268, Oct., 1986.

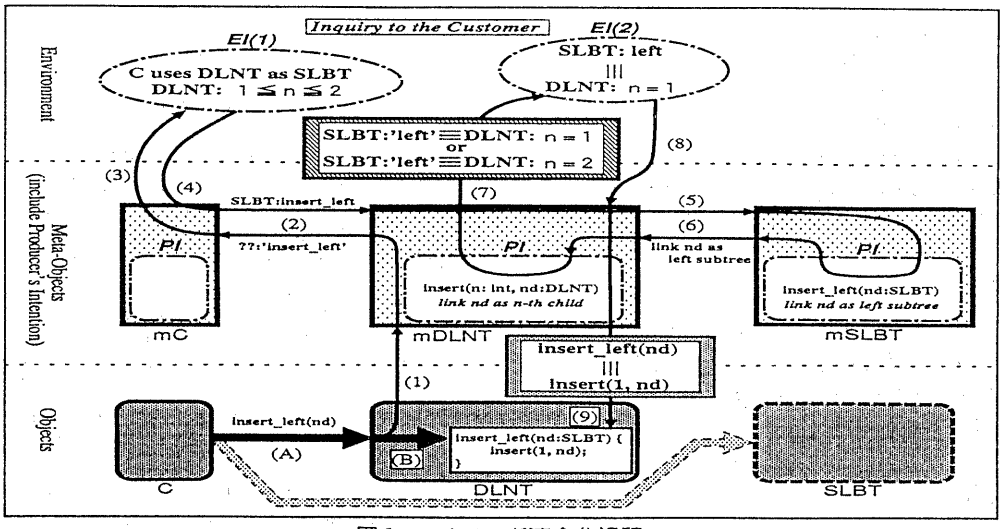



図１　メソッド適合化過程

```
OMEGA-Model ::=
    Agent    Environment
Agent ::=
    Meta-Object    Object
Meta-Object ::=
    Producer's-Intention
Object ::=
    source codes
    (in some favorite language)
Environment ::=
    substitution | selection
substitution ::=
    usage class-name
      uses class-name as class-name
      substitution-constraint ...
substitution-constraint ::=
    class-name ':' expression ';'
selecttion ::=
    select number
```

図２　ＯＭＥＧＡモデル

```
Producer's-Intention ::=
  class class-name [':' description] ';'
      vocabulary v-set ...
      method-specification ...
method-specification ::=
  method-name [':' return]
    function function-description ';'
    [parameters [number param-name ':'
                 type-name purpose ';'] ... ]
    [precondition [term '=' term] ... ]
    [postcondition [term '=' term] ... ]
vocabulary-set ::=
  vocabulary v-set
    [include v-set ['('exchange-def ')']
            [',' v-set ['('exchange-def ')']] ... ]
    [noun word [',' word] ... ';' ]
    [adjective adj-def [',' adj-def ] ... ';' ]
    [verb word [',' word] ... ';' ]
    [synonym [word '=' [v-set ':'] word] ... ]
exchange-def ::=
  word ':' word [',' word ':' word ] ...
adj-def ::=
  word '(' word [',' word] ... ')'
```

図３　作成者意図の記述形式

```
class SLBT: single-linked binary tree node;
  vocabulary binary_tree;
  insert_left  function     link nd as left subtree;
               parameters   1 nd:SLBT subtree;
               postcondition leftO=nd;
    ...
class DLNT: double-linked n-ary tree node;
  vocabulary n-ary_tree;
  insert       function     link nd as n-th child;
               parameters   1 n:integer position;
                            2 nd:DLNT child;
               precondition n>=1;
               postcondition child(n)=nd,
                            nd.parentO=this;
    ...
vocabulary   tree
  noun       tree, node, root, leaf, parent, child, value;
  adjective  elder(node), younger(node);
  verb       link, unlink, traverse, get, set;
vocabulary   binary_tree
  include    tree;
  noun       subtree;
  adjective  left(subtree), right(subtree);
  synonym    subtree=tree:child;
vocabulary   n-ary_tree
  include    tree;
  adjective  n-th(child);
```

図４　仕様記述例