Vol. 35 No. 5　　情　報　処　理　　May 1994

解　説

オブジェクト指向分析・設計

1. オブジェクト指向分析・設計†☆

本位田　真　一††　　山　城　明　宏††

1. はじめに

ソフトウェア開発の現場において，ソフトウェア危機が叫ばれて以来久しい．その間，さまざまな概念が提案され，成果をおさめている手法もあるが，ソフトウェアの工業化に向けては課題が山積している．オブジェクト指向はオブジェクト開発における現状の課題に対して，新たな光明を与えるものとして多くの期待がある．それは，オブジェクト指向が以下のような特徴を有しているからである．

- 実世界におけるイメージによって自然にモデル化できる
- 上流工程から下流工程における成果物に一貫性がある
- 部品化再利用に適している
- 保守の局所化が図れる

こうした技術的な側面と同時に，以下のような側面も期待感の背景に存在する．提唱者のひとりである Edward Yourdon は構造化分析・設計[Yourdon 89]の大御所として広く知られている．また，「Object Oriented Systems Analysis」[Shlaer 88,92]の著者のひとりである Steve Mellor も，リアルタイムシステム向けの構造化分析・設計の大家である[Ward 85]．すなわち，実績のある彼らが提唱しているという側面も人気を高めている要因であろう．彼らがオブジェクト指向方法論を発表した背景には次の二つが考えられる．

- 構造化分析が生まれた当時（1970 年代後半）には，ワークステーションのようなシステムは影も形もなかったが，最近ではそのようなシステムの普及によって，構造化分析にとっては不向きのアプリケーション（たとえばユーザ・インタフェース）の全体に占める割合が，大幅に増加したこと
- C++ や Smalltalk などのオブジェクト指向言語が 90 年代の主流言語になりつつあること（オブジェクト指向言語向きの分析・設計手法としてはオブジェクト指向分析・設計が最も適合する）

こうした状況の下では，もはや方法論として構造化分析だけでは対処しきれなくなり，必然的にオブジェクト指向方法論を提唱するに至ったというのが素直な解釈であろう．

しかしながら，オブジェクト指向分析・設計は生まれて間もないこともあり，方法論としては未熟であり，先に示したオブジェクト指向の有する特徴を十分には生かしきっていないのが現状であり，実力よりも人気が先行している状況である．

本稿では，まず，オブジェクト指向について述べ，続いてオブジェクト指向分析・設計について，そして具体的な方法論の紹介と比較を述べる．さらに，オブジェクト指向分析・設計に関するトピックスとして，部品化再利用について論じる．最後に，今後の課題は何かについて述べることにする．

2. オブジェクト指向とは

オブジェクト指向アプローチの発想の原点は，現実のモノおよびモノどうしの関係をそのままソフトウェアで表現することによって，現実世界の仕組みをコンピュータ上で再現しようとするものである．現実に近い形でシステムの構造を表すことによって，その構造がより直感的で分かりやすくなる．現実の世界は，いろいろなモノが役割を分担しながら機能している．自分でできることは

† Object-Oriented Analysis and Design by Shinichi HONIDEN and Akihiro YAMASHIRO (Toshiba Corp. Systems and Software Engineering Laboratory).

†† (株)東芝研究開発センター　システム・ソフトウェア生産技術研究所

☆ 本稿は文献［本位田 93］に基づいている．

独立して作業を進め，自分のテリトリ以外の仕事は他にまかせて，結果だけをもらう．そこで，現実の世界のモノ（＝オブジェクト）とモノどうしのつながり（＝関係）に着目し，個々のオブジェクトに作業を割り振り，オブジェクトどうしがお互いに作業を依頼しながら機能するようにしたのが，オブジェクト指向アプローチである．

2.1 基本用語

オブジェクト指向システムの基本的な構造は，「オブジェクト」「クラス」「属性」「メッセージ」「メソッド」「関係」の6つの用語で説明できる(**図-1** 参照)．システムはオブジェクトが中心となって構成される．個々のオブジェクトはその役割(作業)を果たすのに必要な「データ(属性)」と「手続き（メソッド)」を内部にもっている．この属性とメソッドを一つにまとめた構造をカプセル化と呼び，オブジェクト指向の特徴の一つとなっている．このような構造をもつオブジェクトどうしは「作業依頼(メッセージ)」を送ることで，協調的に作業を進めることができる．メッセージを受けたオブジェクトは，作業した結果を必要に応じて他のオブジェクトに渡す．オブジェクト指向のもう一つの特徴として，「個々のオブジェクトをさらに抽象化した概念(クラス)」を取り入れている．クラスはオブジェクト（インスタンスと呼ぶこともある)という実体のテンプレートである．また，オブジェクトはお互いの位置づけを明確にする必要がある．この位置づけを表現するのが，「オブジェクトどうしのつながり(関係)」である．関係は大きく「関連」「part-of 関係」「is-a 関係」の3種類に分類できる．まず，「関連」はオブジェクトどうしの対等な参照，あるいは利用関係を示す．オブジェクト間の関連の存在は，接続されるオブジェクト間でメッセージのやりとりが行われることを示す．「part-of 関係」は一方のオブジェクトが他方の「部分」となるような構造的な包含関係を表す．言い換えれば，part-of 関係では，親オブジェクトは子オブジェクトの集約により構成される．これに対して，「is-a 関係」は，クラス間の概念的な包含関係である．「is-a 関係」においては，子は親の性質を継承し，さらに自分独自の性質をもつことができる．ここでの性質とは，属性，メソッド，関係を意味する．

2.2 基本理念

オブジェクト指向の概念として

- 抽象化
- カプセル化
- 継承
- 状態

がある．これらについては，前節でも少し触れたが，本節では，オブジェクト指向の概念がどのような効果をソフトウェア開発にもたらすのかの観点で整理する．

2.2.1 抽象化

オブジェクトによって現実の世界を抽象化することの効果は，2点あげられる．まず，第1に実世界を自然に表現できることである．実世界との比喩を用いると必要な機能や振舞いを類推できる．第2に，分析の焦点を明確化することである．実世界の重要な側面だけをモデル化することは，以降の分析で注目すべき焦点を明確にできることを意味している．

抽象化には，現実世界のモノをオブジェクトとして抽象化するという意味のほかに，同じ性質をもつオブジェクト群をさらにクラスとして抽象化する，あるいは，クラス群をさらに抽象的なクラスとして抽象化する，という意味がある．このようなモノの抽象化階層を作り上げることによって，モノの構造を体系的にとらえることも可能になる，これによって，実世界を自然に把握することができる．

2.2.2 カプセル化

カプセル化は，オブジェクトの内部にデータを隠蔽し，外部には手続きの仕様のみを見せることで，オブジェクトの仕様と実装を分離するアプ

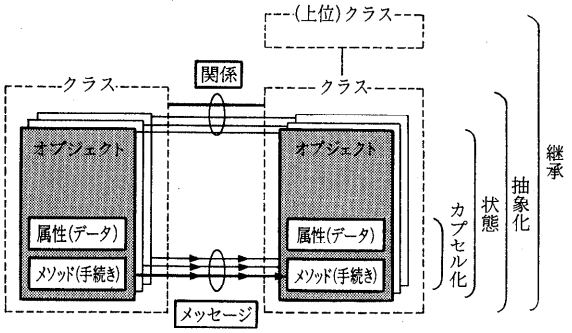


図-1 オブジェクト指向システムの基本的な構造．「オブジェクト指向システム開発」(日経 BP 社) より転載

ローチである．このカプセル化による最も大きな効果はオブジェクトの提供者とオブジェクトの利用者の切り分けを明確にできる点である．具体的には，以下の2点があげられる．オブジェクトの利用者はオブジェクトに定義された手続きの仕様さえ知っていれば，オブジェクトの内部のデータ構造あるいは，手続きの実装を知る必要がない．また特定のデータ構造にアクセスするための手続き群は，オブジェクトのひな型である「クラス」という単位の中にまとまって格納されている．そのため，必要な処理を行いたい場合の手続きの特定が比較的容易である．一方，オブジェクトの仕様と実装の分離は，オブジェクトの提供者がオブジェクトの内部の変更，たとえば内部データ構造や手続きの実装の変更を行っても，その影響を利用者のソースに及ぼさない点で，提供者にとっての変更のしやすさが向上する．

2.2.3 継　承

オブジェクト指向においては，上位クラスの概念を下位クラスが引き継ぐとともに，下位クラスでは上位クラスに存在しない新たな性質を追加できる，継承というメカニズムを取り入れている．これによって，既存の資源を有効に再利用して生産性を向上させる手段を提供している．継承の効果としては以下の2点があげられる．第1の効果は，クラスを体系化する手段として有効な点である．複数のクラスに共通の性質を一つの上位クラスに抽象化することで，上位クラスは個々の下位クラスの分類の観点を明示できる．また，下位クラスでは，上位クラスに対して追加された分のみの性質が記述されるので，何が個々の下位クラスの特徴であるのかが理解しやすい．第2の効果は，既存のクラスの拡張のしやすさである．既存の資産をそのまま受け継いで，それを拡張する形で新たなクラスを定義できる（これを差分プログラミングという）ので，新たなクラスの定義が容易に行える．

2.2.4 状　態

オブジェクトに定義されている手続き群の利用可否は，実際には内部の属性値によって決まる場合がある．そこで，オブジェクトを特定の内部データ値によって複数の異なる状態に分け，オブジェクトの状態ごとに，その時点で利用可能な手続きを明示することが必要である．状態の概念をオブジェクトの仕様に取り入れることの効果は，オブジェクトの提供者が，メッセージの正しい受信手順（メソッドの利用手順）を状態遷移図として利用者に提供できることである．

3. オブジェクト指向でのモノ作り

オブジェクト指向によるソフトウェア開発においては，従来のライフサイクルモデルと同様に分析フェーズ，設計フェーズ，プログラミングフェーズが存在する．オブジェクト指向ソフトウェア開発においては，それぞれ，オブジェクト指向分析，オブジェクト指向設計，オブジェクト指向プログラミングと呼ぶ．英語表現ではそれぞれ Object-Oriented Analysis（略して OOA), Object-Oriented Design（略して OOD), Object-Orineted Programming（略して OOP）となる．これらの用語では，OOP が最も早く生まれ（1980 年代前半），OOD が次に（1980 年代半ば），そして最後に OOA（1980 年代後半）が誕生した．この順序はオブジェクト指向に対するニーズの高まり，人口の増加，普及度にともなうものである．限られた愛好家や研究者によるモノ作りの形態から，製品開発を行う複数のソフトウェア技術者による大がかりなモノ作りの形態に変化した．その結果，上流工程に対する要請が高まり，それに応ずる形で生まれてきたといえる．したがって，前者の人たちにとってのモノ作りとは依然として OOP のみである．

3.1 分析フェーズとは

分析フェーズとは，一言でいえば要求モデルを作ることである．要求モデルの定義は分析の目的によって多少異なっている．システムの機能を確立することを目的とした場合には，要求モデルとは発注者が望むシステムの理想像である．一方，実世界の理解を踏まえてシステムの機能を確立することを目的とした場合には，要求モデルとは，実世界のモデル化である．オブジェクト指向分析は後者を重視している．一般に，発注者が出す要求は不完全，曖昧で冗長である．受注者は発注者とのコミュニケーションによって要求の不完全さ，曖昧さや冗長さをなくすと同時に受注者による問題領域の理解を深める．そして，発注者の意図した要求仕様（要求モデル）を作成し，何をシステムとして実現するのかを明らかにする作業が

モデリングである．要求モデルを記述する際には，対象（実世界，システム）の正しい把握が必要となる．そのための具体的手段として，次の三つを明確にすることになる．「対象の静的な構造」，「対象におけるイベントの実行順序（動的な振舞い）」，「対象におけるデータ変換（機能）」である．これらの3項目が最終的なプログラムの情報に反映されるためである．オブジェクト指向はこの3項目をオブジェクトの観点でモデル化する．すなわち，対象の静的な構造についてはオブジェクトモデル，対象におけるイベントの実行順序（動的な振舞い）については動的モデル（あるいは状態モデル），対象におけるデータ変換（機能）については機能モデル（あるいはプロセスモデル）である．

3.2 設計フェーズとは

設計フェーズでは，分析フェーズで得られた要求モデルから実現方式を決定し，実現に関する情報を生成し，プログラミング・フェーズに渡すことを目的とする．したがって分析フェーズとプログラミングの中間に位置し，ソフトウェア開発の中心部分に位置する．設計フェーズは論理設計（あるいは基本設計）および物理設計（あるいは詳細設計）の二つから構成される．

オブジェクト指向では，問題領域に関するオブジェクトが分析フェーズにおいて抽出され，設計フェーズに渡される．論理設計フェーズでは，このオブジェクトに対して，実現に関する情報がつけ加えられる．コンピュータで実現するためには，ユーザ・インタフェース部分，データ管理部分，タスク管理部分の追加が必要である．論理設計フェーズではこれらに関する情報を新たに作成する．次に，物理設計フェーズでは，実現マシンと言語を意識し，実現する上での制約を考慮する．さらに性能面についても踏まえた実現方式を決定し，プログラミングへの情報（詳細設計仕様書）を作成する．

3.3 ラウンドトリップ型の開発形態

現実のソフトウェア開発においては，どうしても設計フェーズから分析フェーズへの手戻りが頻繁に発生する．従来技術，たとえば構造化分析・設計においては分析と設計との間にギャップが存在し，そうした手戻りに関するコストは低くはなかった．

オブジェクト指向開発における一つの特徴は，従来と比較して，分析フェーズが設計フェーズに近づき，プログラミング・フェーズが設計フェーズに近づき，3者の距離が近いことである．その結果，分析フェーズ，設計フェーズを行ったり来たりする形態(ラウンドトリップ型と呼ぶ)が容易になり，オブジェクト指向分析・設計では手戻りに関するコストは低くなる．さらに，この特徴を活かして，積極的に手戻りを推進するのがラウンドトリップ型であり，極論すれば分析フェーズと設計フェーズを同時に行うことになる．オブジェクト指向開発を支援するオブジェクト指向 CASE においては，このラウンドトリップを支援する必要があり，そのためには，きめの細かい仕様化プロセスの定式化が必要となる[Honiden 93]．

4. 種々の手法

現在，オブジェクト指向分析・設計には数多くの手法が提案されている．これらの手法は以下のように三つに分類することができる．

- オブジェクトモデル・アプローチ：
  Shlaer & Mellor 方法[Shlaer 88,92]，Coad & Yourdon 手法[Coad 91a,91b]，OMT[Rumbaugh 90]，Booch 法[Booch 82,86,93]，RDD[Wirfs 90]，HOOD[Hood 89]
- 動的モデル・アプローチ：
  OBA[Rubin 92]，OOSE[Jacobson 92]
- 機能モデル・アプローチ：
  OOSD[Wasserman 90]，Martin & Odell 手法[Martin 92]，OOS[Bailin 89]

この分類は，**3.1** で述べた三つのモデル（オブジェクトモデル，動的モデル，機能モデル）のどのモデルの構築に重点を置く（あるいは最初に構築する）かによる．まず，オブジェクトモデル・アプローチは，最初に実体のあるオブジェクトを見つける手法である．オブジェクト指向方法論として比較的よく耳にする手法の多くはこの分類に属す．このアプローチでは，まず，与えられた要求仕様から，名詞とそれに対する動詞を探し，初期のオブジェクトモデルを構築する．名詞がオブジェクトあるいは属性となり，動詞がメソッドあるいは関係になる．次に，オブジェクトあるいはシステムの動作を把握する．最後に，オブジェクト内の機能を明らかにする．以上が基本的な流れであるが，詳細は各手法によって異なっている．

オブジェクトモデル・アプローチは完成度の高い要求仕様書の存在を前提としている．しかし，大規模なシステムでは，そうした要求仕様書が存在しないことが多い．そこで，最近注目されているのが動的モデル・アプローチである．このアプローチでは，最初に把握するのは実体ではなく，まずシステムに何が起こるか（振舞い）を把握することから始まる．そして，この振舞いを割り付けるという観点でオブジェクトを認識するのが基本的な考え方である．

なお，機能モデル・アプローチは，機能モデルの構築を重視する手法が属す．したがって，構造化分析・設計手法をベースにオブジェクト指向の考え方を導入した手法もここに属す．そうした手法は，構造化手法の発展形と位置づけられ，オブジェクト指向方法論として認知されないことも多い．

ここでは，正統派のオブジェクト指向方法論として，オブジェクトモデル・アプローチと動的モデルアプローチに属す手法のいくつかを紹介する．

4.1 オブジェクトモデル・アプローチの事例

Shlaer & Mellor 手法は1988年に提唱された手法である[Shlaer 88]．さらに1992年には，その完成版を公表した[Shlaer 92]．Shlaer & Mellor 手法はデータベースのモデル化技術である意味モデル[Hull 87]の発展形である．したがって，意味モデルと似ており，木目の細かい関係定義，属性定義を意味モデルから継承している．また，大規模なシステムにも適用可能なように，いくつかのモデルおよびさまざまなドキュメントが用意されており，それらの関係も明らかにされているのが特徴である．設計段階への連結度が低いのが難点である．

OMT は Rumbaugh らが General Electric 社 (GE社) において開発した手法である[Rumbaugh 90]．この OMT は，図表やモデルが詳細に決まっており十分な記述能力を有している点と，作業プロセスを具体的に指示している点が他の手法と比べて優れている．しかし，設計フェーズに関しては分析フェーズと比べてあまり十分とは言い難く，どちらかと言えば分析寄りの手法である．また，用意されたドキュメントだけでは，大規模システムの記述には不十分である．

Coad & Yourdon 手法は，構造化分析・設計において教祖のひとりである E. Yourdon が提唱していることもあり，注目されている．Coad & Yourdon 手法は1990年に教科書において，公表され，すぐに改訂版が出版され[Coad 91a]，設計について論じた教科書も合わせて出版された[Coad 91b]．設計との連結については，分析フェーズでは問題領域のオブジェクト，設計フェーズではユーザインタフェース関係，データ管理，タスク管理のオブジェクトを定義するとしており，分析フェーズと設計フェーズの連結性は高い．動的モデルの記述に関しては，表現形態はフラットな状態遷移図であり，またその作成に関する手順も明確にされておらず，十分とは言えない．

Booch 法は元来 Ada のための設計方法論としてまとめられた手法[Booch 82] を，オブジェクト指向分析・設計手法へ発展させたものであり，どちらかと言えば設計寄りの手法である[Booch 86, Booch 93]．この手法の特徴は図の種類の多さであり，これらを用いると設計に必要なほとんどの図を書くことができる．しかし，これらの表記法の使い方について明確な指針はなく，分析・設計者に委ねられている．したがって，実際に運用する際には詳細な手順を確立しておく必要がある．

4.2 動的モデル・アプローチの事例

OBA (Object Behavior Analysis)[Rubin 92]は Rubin と Goldberg が提案している手法であり，動的モデル・アプローチの代表例である．最初に認識した振舞いを割り付ける際には，その振舞いをだれが始め (initiate)，だれがそれに加わる (participate) かを理解する．この2種類のプレイヤ，initiator と participants は，システムの役割を演じ，オブジェクトとして認識される．そして，これらの役割に対して，振舞いの責任を与える．さらに，オブジェクトとして詳細化していく．以上が基本的な流れであるが，現時点では論文のみの公表であるため，詳細は必ずしも明確ではない．方法論としても未熟であり，今後の洗練化作業に期待したい．

Jacobson の OOSE (Object-Oriented Software Engineering)[Jacobson 92]も動的モデル・アプローチの一つと考えられる．OOSE では，ユースケースの概念が大きな特徴である．システムの機能とは，多くのユースケースから構成される．それぞ

れのユースケースは，システムにおいて，一連のイベントの流れを記述したものと考えることができる．こうしたユースケースに基づいて，オブジェクトとメソッドを抽出する．本手法はそもそもSimula の影響を受けて構築され，20 年かけて改良が加えられてきた．多くのリアルタイムシステムへの適用実績からも，本手法の実用性は評価できる．今後のオブジェクト指向方法論に対して，OMT とともに影響力を発揮していくことが予想される．

4.3 3大手法の比較評価

Shlaer & Mellor, Coad & Yourdon, OMT の各手法をいろいろな角度から比較しよう．まず，手法によって表記法は確かに異なるが，本質的な考え方にはほとんど差異がない．用語の使い方を比較するために，オブジェクト指向言語と意味モデルの用語を併記した（**図-2** 参照）．オブジェクト指向言語と意味モデルの両方を基礎としているが，手法によっては，いずれかを中心に考えている場合がある．たとえば，Shlaer & Mellor 手法は意味データモデルを基盤としているため，メソッドに関する記述が明確ではない．一方，Coad & Yourdon 手法はオブジェクト指向言語を基盤としているため，関連の記述が弱いことがあげられる．仕様化プロセスにおいても，作業項目の相違が明らかになる（**図-3** 参照）．また，仕様化プロセスに合わせて作成するドキュメントも併記した．一般に，ドキュメントが豊富であると仕様化プロセスが複雑になり，方法論を遂行する上での自由度も小さくなる．実用上の観点からは，仕様化プロセスとしての完成度の高い JSD が，実際には，実務にあまり適用されていないという現実を踏まえて考えなければならない．分析フェーズとしての作業項目はある順序で実施されるというよりは，むしろ並行に同時に実施するほうが自然である．いわゆるラウンドトリップ型であるため，仕様化プロセスの自由度が大きいほうが実用的であるとの意見もある．さて，ドキュメントにおいては Shlaer & Mellor 手法が豊富であることが分かる．しかも各ドキュメント間の関連が明確になっている点も評価できるが，実際の適用上の自由度は小さい．逆に，Coad & Yourdon 手法は，教科書において流れを示しているが，この流れを規定するものではないとしている．これは，他の手法と比較して大きく異なる点である．ただ，ドキュメントがきわめて簡素であることが実用上どのように影響を及ぼすかに関しては意見の分かれるところである．OMT は動的モデルに関する仕様化プロセスをきめ細かく規定している．シナリオ，イベントレース図，イベントフロー図，状態遷移図という作成手順は，モデルの作成者にとって自然である点，および表現形態として Statechart [Harel 87] を用意しており，複雑な動的モデルを記述することができる点は評価できる．

4.4 手法の今後

各手法の教科書に書かれている内容は，かなり

Shlaer & Mellor 手法	OMT 手法	Coad & Yourdon 手法	意味データ・モデル	オブジェクト指向言語
オブジェクト object	クラス class	クラス class	オブジェクト object	クラス class
属性 attribute	属性 attribute	属性 attribute	属性 attribute	変数名 variable
—	操作 operation	サービス service	—	メソッド method
イベント event	事象 event	メッセージ message connection	—	メッセージ message
インスタンス instance	オブジェクト object	オブジェクト object	実体 entity	インスタンス instance
上位型／下位型	スーパクラス／サブクラス	Gen-Spec	汎化／特殊化	親クラス／子クラス
関連付けオブジェクト	集約	Whole-Part	集約	—
関連付けオブジェクト	関連	インスタンス・コネクション	関連	—

図-2 用法の相違．「オブジェクト指向システム開発」（日経 BP 社）より転載

抽象論として描かれており，そのまま現場で役に立つほど具体化されていない．したがって，現場で適用する際には，その部門固有の問題に適するようにうまくカスタマイズする必要がある．筆者らの適用経験では，ある部門では，OMT においてデータフローダイアグラムを拡張し，別の部門では，Coad & Yourdon 手法に対して OMT のイベントトレース図を導入し，それぞれ工夫した．教科書は，あくまでも，できる限り一般性をもたせて，幅広く適用可能な方法論として記述してあるので，やむをえないところである．欲を言えば，方法論のカスタマイズ手法についても論じて欲しいところである．いずれにしても，現在，教科書として提案されている手法の分野限定化，洗練化，拡張化の流れが続いていくのか，それらの手法の統合化がなされていくのか，あるいは，まったく新しい発想でのオブジェクト指向モデルによるソフトウェア開発手法が確立されていくのか，興味のあるところである．また，既存の手法は新規開発を対象とした方法論である．再利用の観点からの方法論の見直しも必要だろう．さらに，方法論のモデルにおいて，構造的，意味的な正しさを保証する手法も必要である．形式的記述を適用した手法も提案されてきている[Embley 92][Nerson 93]が，オブジェクト指向CASEでの支援手法も含めて今後の重要な研究課題である．

手法	Shlaer & Mellor 手法	
オブジェクト・モデル作成フェーズ	オブジェクト検出 ↓ 属性 の定義 ↓ 階層 の定義 (上位型/下位型) ↓ 関連 の定義 (関連付けオブジェクトの定義)	情報構造図 (オブジェクト図) オブジェクト仕様書 関係仕様書
動的モデル作成フェーズ	状態 の定義	状態遷移図 イベント・リスト オブジェクト通信モデル
機能モデル作成フェーズ	アクション の定義	オブジェクト・アクセス・モデル アクション・データフロー・ダイアグラム 状態プロセス表 プロセス記述

OMT 手法		Coad & Yourdon 手法	
オブジェクト検出 ↓ 関連 の定義 ↓ 属性 の定義 ↓ 階層 の定義 (Super Class/Sub Class)	オブジェクト図 (データ辞書)	オブジェクト検出 ↓ 構造 の定義 (Gen-Spec, Whole-Part の定義) ↓ オブジェクトのグループ の定義 (サブジェクトの定義) ↓ 属性 の定義 (インスタンス・コネクションの定義)	オブジェクト図
イベント の検出 ↓ 状態 の定義	シナリオ イベント・トレース図 イベント・フロー図 状態遷移図 (Statechart)	↓ 状態 の定義	オブジェクト状態図
機能 の定義 操作 の定義	データフロー・ダイアグラム	サービス の定義 (メッセージ・コネクションの定義)	サービス・チャート

図-3　仕様化プロセスとドキュメント．「オブジェクト指向システム開発」(日経 BP 社) より転載

5. ソフトウェアの再利用

ソフトウェアの再利用の第一義的な目的は，当然のことながらソフトウェア生産性の向上である．すでに存在するソフトウェアを最大限利用することにより新たに作成するソフトウェアの量を減らすことを目的とする．このとき，再利用するソフトウェアは品質の保証されたものでなければならない．その結果ソフトウェア・システムの信頼性の向上を図ることができる．また，標準化の促進を図ることも大きな目的である．

一般にソフトウェア開発工程において，再利用の対象は，再利用によって生産性，信頼性の要件を満たすことのできる各フェーズのプロダクト(成果物) のすべてである．具体的には，仕様，ノウハウ，プログラム，システムと多岐にわたっている．特に，オブジェクト指向を対象とした場合には，その特徴を生かして再利用の対象はソフトウェア部品（オブジェクトに相当）とアーキテクチャ部品（パターン／フレームワークに相当）とするのが望ましい．すなわち，再利用する対象を「部品」という単位で標準化し，標準化された

部品をフレームワークに沿って再利用することによって，ソフトウェアの均質化を図るのが目的である．こうした部品化再利用における技術課題とは，一般的に，部品（再利用単位）をどうするか，部品をどのように体系化するのか，効率的でかつ簡潔な検索手法をどのように実現するのか（あるいは，検索のための情報をどのように表現するのか），修正手順をどのように定式化するのか（再利用した部品をどのように修正し，再利用するのか）である．

5.1 オブジェクト指向と再利用性

まず，オブジェクト指向が部品化再利用とどのように親和性があるのかについて論じよう．オブジェクトの特徴の中で，部品化再利用と関連するキーワードには，オブジェクトとメッセージによる形態，カプセル化，クラス階層がある．オブジェクト／メッセージによって部品の形態および部品間のインタフェースを統一できる．次に，カプセル化により，部品の独立性，強固性が高められ，クラス階層により部品が体系化され結果として検索が容易になる．同時に，差分プログラミングが可能となり，修正が容易になる．

オブジェクト指向の採用において大きな生産性向上が見込まれるのは，再利用対象である部品が用意されている場合と言える．すなわち，適切なクラス・ライブラリが用意されている場合である．適切なクラス・ライブラリとは標準化部品の集合体と言えるが，これを用意するコストは少なくない．しかし，あらかじめ用意することができれば，多くのシステム開発にとって有効であることは言うまでもない．したがって，オブジェクト指向は先行投資型開発とも言える．

また，膨大になったクラス・ライブラリを再利用するためのコストは低くはない．そこで，ある機能や振舞いを満たすクラスの集合を登録して，再利用しようとするフレームワークの考え方も，実用的な観点から，大変注目されている[Johnson 92]．しかしながら，フレームワークの表現手法，構築手法（たとえば，再利用できるようにどのように抽象化するか），再利用手法（たとえば，どのように理解し，修正するか）など，多くの研究課題がある．オブジェクト指向 CASE において，どのようにサポートしていくかについても興味深いところである．

5.2 分析におけるオブジェクトの再利用

分析フェーズにおいては，仕様が確定していない．そのため，分析における再利用の目的は仕様の確定である．既存のクラス・ライブラリの情報を参考にして，仕様を確定するのが目的である．したがって再利用の対象はプロダクトだけでなく，むしろそのプロダクトを作り出した（仕様を確定する）際に必要となったノウハウになる場合も多い．分析における再利用における特徴とは，分析中の早い段階から既存の概念との共通性を認識することであり，すなわち，新しい観点からオブジェクトを把握することである．分析段階では，与えられた問題の解決には不要，あるいは冗長な情報を含んで認識されるので，不要あるいは冗長な部分を削りなるべく高い抽象レベルで解決をはかることも必要となる．

分析における再利用が必要となる場面はさまざまであるが，その典型例として仕様変更がある．仕様変更とは，すでに存在する作業プロダクト（分析結果）において，その作業プロダクトに対応する仕様が変更されたとき，それに応ずるために作業プロダクトを変更することと定義する．したがって仕様変更は，開発中のシステムに対する要求仕様の変更，すでに運用中のシステムに対する保守要求，既存のシステムの仕様を手本に新たに類似のシステムを構築する場合などさまざまな場面で登場する．本来，従来技術と比較してオブジェクトモデルはこうした仕様変更に強い．

標準化された部品があると仮定してそのときの仕様変更の手順について考えよう．再利用との関連で述べるならば，仕様変更に対して，再利用可能部分，追加修正部分を分類，特定化し，追加修正部分の作業プロダクトを追加・修正・削除する．言い換えれば，再利用可能部分を検索するプロセス，および何をどのように修正するのかという修正プロセスから構成されるのが仕様変更プロセスである．具体的には仕様変更要求の発生にともない，まず，仕様変更要求（問題）を分類する必要がある．たとえば，処理の変更に対応した機能拡張，環境の変更に対応した機能拡張，性能／資源の制約に対応するための変更がある．仕様変更要求の発生した際には，それぞれに対して，仕様変更プロセスを定式化しておく必要がある．

仕様変更とは，当面の仕様変更だけではなく次

の仕様変更にも対処できるように変更しておかなければならない．修正された作業プロダクトが再度変更されることがある．その際にも，再変更がスムーズに行えるように，変更を行っておかなければならない．

5.3 再利用における状態の扱い

オブジェクトを再利用部品と捉えた場合に，従来の部品と何が異なるのであろうか．大きな相違点として，従来の部品の多くが静的なモジュールであるのに対して，オブジェクトは動的なモジュールであることである．動的なモジュールとは，オブジェクトが状態を有することである．この状態を有することによってオブジェクトとしての再利用性（利用されるときの変更度，および変更容易性）が低下している．これは一般に状態遷移図の再利用性は高くないことに起因する．

また，仕様変更に対する耐性が弱くなっているのも事実である．たとえば，OMT のオブジェクトモデルは仕様変更に強いのに対して，動的モデルは弱いと言える．この理由の一つとして，オブジェクトに対して複数のオブジェクトが絡むことがあげられる（他のオブジェクトとの結合度が高い）．したがって，それぞれのオブジェクトがそのオブジェクトとの関わり方をもっており，そのオブジェクトに対して複数の視点が存在することになる．その関わり方とは基本的には，メッセージ交換であり，メッセージを送るときには，相手の状態を踏まえて送るのが通常である．複数のオブジェクトがそのオブジェクトにそれぞれ送るときには，それぞれの視点でそのオブジェクトの状態を見ていることになる．したがって，視点ごとにオブジェクトの状態の形態が異なっているのである．

このような状況においてたとえば，新たなオブジェクトの追加といった仕様変更が起きたとする．新たなオブジェクトの追加によって新たな視点による状態遷移が発生することになる．状態を有するオブジェクトはそうした追加に対しても対処しなければならない．別の仕様変更としては既存のオブジェクトにおいてメッセージの交換順序の変更がある．これらの変更に対しても変更箇所を少なくしておく必要がある．そこで，複数の視点による状態の差異を最小化し，適度な抽象度でモデル化する手法が提案されている[入江 92]．

6. むすび

オブジェクト指向分析・設計の課題を普及の観点でまとめる．まず，品質評価手法の確立があげられる．オブジェクト指向モデルに適した品質評価方式は現状では確立していないが，いくつか提案されてきており，今後の研究成果に期待するところが大である．特に上流工程における作業プロダクトに対する品質評価手法が他の方法論においても確立していない状況であるため，オブジェクト指向によって，上流工程における品質評価手法の確立の可能性が生まれたと言える．次に，工数見積もり技術を含めたプロジェクト管理手法の確立である．すなわち，多人数によるラウンドトリップ型の開発形態をどのように管理するかである．ラウンドトリップとはいわゆるプロトタイピング手法である．プロトタイピング手法は管理という側面には向いていないとされてきたが，他の方法論においては，管理に向いているライフサイクルモデルの中で，必要に応じてプロトタイピングが採用されてきたので問題なかった．しかし，ラウンドトリップ型ソフトウェア開発は，ライフサイクルモデルとは直交する考え方であり，従来とは異なる新たな管理基準の設定が必要になる．

また，従来技術の蓄積をどのように継承するかについても大きな問題である．この問題を解決する一つの動きとしてラッパーの構築があげられる[Booch 93]．ラッパーとは「包むもの」という意味であり，すでに開発されたソフトウェア遺産をオブジェクト指向のインタフェースでキャストすることにより，それをあたかも一つのオブジェクトに見せかける技術である．この方法は従来手法から，オブジェクト指向アプローチに移行する過渡期の技術として注目されている．次に，ソフトウェア開発現場にどのように新しい概念を浸透させるかである．そのためには，まず，企業内において効果的な教育体系を確立させる必要がある．また，導入に当たっては，その部門において対象とするソフトウェア・システムの性質に大きく影響する．再利用性（リピート性）の高いシステム，あるいは，足の長いシステムの場合には，初期投資の性格の強いオブジェクト指向は有効である．さらに，部門における標準化戦略とも大きく絡んでくるのは言うまでもない．

参考文献

[Bailin 89] Bailin, S. C.: An Object-Oriented Requirements Specification Method, Comm, ACM, Vol. 32, No. 5 (1989).

[Booch 82] Booch, G.: Object-Oriented Design, Ada Letters, Vol. 1, No. 3 (1982).

[Booch 86] Booch, G.: Object-Oriented Development, IEEE Trans. Software Eng., Vol. SE-12, No. 12 (1986).

[Booch 93] Booch, G.: Object-Oriented Analysis and Design with Applications 2nd ed. (1993).

[Champeaux 93] de Champeaux, D., Lea, D. and Faure, P.: Object-Oriented System Development, Addison-Wesley (1993).

[Chen 77] Chen, P.: The Entity-Relationship Approach to Logical Data Base Design, Q. E. D. Information Sciences, Wellesley, Masachusetts (1977).

[Coad 91 a] Coad, P. and Yourdon, E.: Object-Oriented Analysis second edition, Yourdon Press, 1990 (羽生田監訳: オブジェクト指向分析 (OOA), トッパン, 1993).

[Coad 91 b] Coad, P. and Yourdon, E.: Object-Oriented Design, Yourdon Press (1991).

[Embley 92] Embley, D., Kurts, B. and Woodfield S.: Object-Oriented Systems Analysis, A Model Driven Approach, Yourdon Press (1992).

[Harel 87] Harel, D.: Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming 8 (1987).

[Honiden 93] Honiden, S., Kotaka, N. and Kishimoto, Y.: Formalizing Specification Modeling in OOA, IEEE Software, 1993 January (1993).

[本位田 93] 本位田, 山城: オブジェクト指向システム開発, 日経 BP 社 (1993).

[Hood 89] Hood Working Group: Hood Reference Manual, European Space Agency (1989).

[Hull 87] R. Hull and R. King: Semantic Database Modeling: Survey, Applications, and Research Issues, ACM Computing Serveys, Vol. 19 (1987).

[入江 92] 入江, 斉藤: 仕様・運用変更を意識した複数視点による OOA, 電子情報通信学会, 知能ソフトウェア工学研究会, 92-25 (1992).

[Jacobson 92] Jacobson, I., Christerson, M. Jonsson, P. and Overgaard, G.: Object-Oriented Software Engineering, Addison-Wesley (1992).

[Johnson 92] Johnson, R. E.: Documenting Frameworks using Patterns, OOPSLA '92, pp. 63-76 (1992).

[Martin 92] Martin, J. and Odell, J.: Object-Oriented Analysis and Design, Prentice Hall (1992).

[Meyer 88] Meyer, B.: Object-Oriented Software Construction, Prentice Hall (1988) (二木監訳: オブジェクト指向入門, アスキー出版局, 1990).

[Nerson 92] Nerson, J. M.: Applying Object-Oriented Analysis and Design, Comm. of ACM, Vol. 35, No. 9 (1992).

[Rubin 92] Rubin, K. and Goldberg, A.: Object Behavior Analysis, Comm. of ACM, Vol. 35, No. 9 (1992).

[Rumbaugh 90] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: Object-Oriented Modeling and Design, Prentice-Hall (1990) (羽生田監訳: オブジェクト指向方法論 OMT, トッパン, 1992).

[Shlaer 88] Shlaer, S. and Mellor, S. J.: Object-Oriented Systems Analysis, Prenice Hall (1988) (本位田, 山口訳: オブジェクト指向システム分析, 哲学出版, 1990).

[Shlaer 92] Shlaer, S. and Mellor, S. J.: Object Lifecycles, Prentice-Hall (1992) (本位田, 伊藤監訳: 続オブジェクト指向システム分析, 哲学出版, 1992).

[Ward 85] Ward, P. T. and Mellor, S. J.: Structured Development for Real-Time Systems, Prentice-Hall/Yourdon Press, New York (1985).

[Wasserman 90] Wasserman, A. I. et al.: The Object-Oriented Structured Design Notation for Software Design Representation, IEEE Computer, Vol. 23, No. 3 (1990).

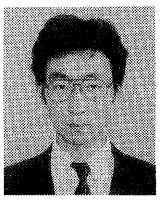
[Wirfs 90] Wirfs-Brock, R., Wilkerson, B. and Wiener, L.: Designing Object-Oriented Software, Prentice Hall (1990).

[Yourdon 89] Yourdon, E.: Modern Structured Analysis, Prentice-Hall (1989).

(平成5年9月13日受付)

本位田真一 (正会員)

1953 年生. 1976 年早稲田大学理工学部電気工学科卒業. 1978年同大学院理工学研究科電気工学専攻修士課程修了. 工学博士. 同年(株)東芝入社. 現在, 同社研究開発センターシステム・ソフトウェア生産技術研究所に所属. 1989年より早稲田大学非常勤講師を兼任. 1991年東京工業大学大学院非常勤講師. 主として, ソフトウェア工学, 人工知能の研究に従事. ソフトウェアの基礎理論に興味を持つ. 1986 年情報処理学会論文賞受賞. 著訳書「ソフトウェア開発のためのプロトタイピング・ツール」(共著), 「KE 養成講座(2)エキスパートシステム基礎技術」(共著), 「オブジェクト指向システム分析」(共訳共著) など. 日本ソフトウェア科学会, 人工知能学会, IEEE, AAAI 各会員.



山城 明宏 (正会員)

1960 年生. 1982 年北海道大学工学部電子工学科卒業. 同年(株)東芝に入社. 1991 年 Boston 大学工学部システム工学科修士課程修了. 現在, システム・ソフトウェア生産技術研究所開発主務. ソフトウェア再利用技術およびオブジェクト指向分析/設計の研究・開発・教育に従事. 著書「オブジェクト指向システム開発」(共著, 日経 BP). IEEE, ACM 各会員.