オブジェクト・モデリングへの期待

岩 田 裕 道

1. はじめに

本稿に与えられた主題は筆者にはかなり荷が重く，いかなる読者を対象にいかなる論旨のもとにいかなるメッセージを発すればよいのか，しばらくは手が着かない状態が続いた．表題の意味するところは，技術解説でも技術報告でもなく，まして論文というスタイルとも思えない．筆者の下した結論は，（オブジェクト指向技術についての読者の関心範囲や理解がまちまちに違いないので）読者層は特に意識せず，自分のオブジェクト指向モデリングについての期待をそこはかとなく述べるしかない，ということであった．よってこの小文は，オブジェクト指向技術に関して現在筆者自身が抱いている期待を，オブジェクト・モデリングの周辺に話題を絞って，できるだけ素直に記述したものである．

オブジェクト・モデリングの普及は，企業のオブジェクト指向技術へのシフトのための重要な第１ステップである．この技術を真に生かしたビジネス・アプリケーションがなかなか増えてこない主要な理由は，そのモデリングを行うモデラがいないことだと筆者は考えている．したがってこの小文の主たる目的の一つは，オブジェクト指向技術に共感し挑戦するモデラが少しでも増えることである．そして管理者の人たちにも，それを理解し積極的に支援していただくことである．

この技術は長い歴史を持つとはいえ，ビジネス現場での適用という意味ではまだ成熟したものとは言えない．新しい技術へのシフトはまた，新たな課題への挑戦でもある．この技術の着実な普及のために解決すべき課題についても触れたい．また，統一モデリング言語 UML は，この技術の普及に弾みをつけるという意味で触れないわけにはいかない．

紙数の関係から専門用語の説明などは省略されているものが多い．また，論旨の背景などの説明もかなり省いてあり，舌足らずな箇所もあるに違いない．これらは，本文中の注や参考文献などによって補完していただくことを期待している．

2. システム開発とモデル

2.1 モデルの意味

モデルという言葉は一般に原型，模型，雛形，模範，題材といったさまざまな意味で使われるが，ここでは「直接には把握しにくい事物や事象を，図と記号を使って表現したもの」を指す[*1]．すなわち図式化された模型である．物理や化学で使う原子模型や亀の甲で描かれた分子模型などはその典型である．このようなモデルは，理解しにくい対象を図式という代替手段で表現したというだけでなく，その対象の持つ特定の視点での意味をより理解しやすくし，人間同士の正確なコミュニケーションに役立つ．

論理や集合を扱う場合に使うベン図もよく知られたモデルである．例えば，「X は

すべて Y である」という文章は図 1 のように表現されるが，これは「Y であるからといって必ずしも X であるとは限らない」という言外の意味も含めた，この文章の正しい理解を明らかに促進する．ある程度論理的なものが要求されるコミュニケーションでは，たとえコンテキスト（その場面での関心範囲）や使われる用語の意味が当事者間で同一の理解があったとしても，自然言語による言葉やテキストだけのコミュニケーションを続けると，必ずと言ってよいほど誤解が生じる．自然言語は基本的にあいまいだからである．

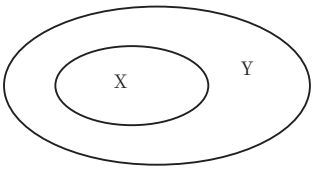

図 1　ベン図

場所や道を教えたり覚えるのに使う地図や略図も，この定義によればモデルである．ラフな殴り書きであっても，関心を持つべき重要な建物や道路の位置関係が明確でさえあれば，5 万分の 1 の詳細な地図よりはるかに理解しやすく役に立つ．このような図式モデルの一般的な役割については，文献[1]に非常に分かりやすい説明がある．一般にモデルは，元の対象に関して重要な部分のみを強調し，当面関心のない部分は切り捨てている．すなわちモデルは一種の抽象化である．理解しやすいのは，モデルが視覚化されている（ビジブル）からである．

システム開発の場面では，従来から知られている ER 図（実体関連図），DFD（データフロー図），STD（状態遷移図）などがモデル図式の代表例である．しかし，本稿ではオブジェクト・モデルを中心に扱う．後述するように，このモデルは，システム開発の対象ビジネス領域を互いに異なる視点でモデル化した，複数の図式の集合であり，オブジェクト指向技術を使って作られる．

システム開発，特にオブジェクト指向技術を用いるシステム開発，でのモデルの意味（目的と役割）は，次の三つである．

・コミュニケーション手段
・開発成果物
・再利用の対象

コミュニケーション手段

これは，モデルが一般的に持つもっとも重要な役割である．オブジェクト指向(OO)システム開発はモデル駆動であり，システム化の対象であるビジネス領域をモデル化（図式化）して，これを洗練し詳細化していく．そこで作られるモデルは，開発チーム内，チーム間，およびビジネス領域の担当者との間での正確なコミュニケーションを促進するための主要な手段となる．モデルはまた，個人が頭の中で漠然と理解している事柄をより明確にするための，つまり自己内省（リフレクション）のための手段でもある．

コミュニケーション手段としてのモデルが有効に働くには，その表記法（言語）が統一されていなければならない．そうでないと，それぞれ自国語しか知らない日本人とアラビヤ人とドイツ人が話し合うようなハメになりかねない．しかし，そのような場合でもモデルは図が主体であるから，表記法が多少違っても自然言語より理解しやすいことは言うまでもない．後述する UML（統一モデリング言語）の制定は，従来から最大の障害であった図式表記法の不統一，という問題を解消する点で非常に意義がある．従来のシステム開発でも図式は使われていたが表記法はまちまちであり，主要なコミュニケーション手段は，自然言語で書かれた膨大なテキストであった．自然言語の持つあいまいさは周知の事実であり，特に国際化の叫ばれる現代にあっては，自然言語主体の仕様の情報交換は（多量の翻訳作業を伴うという意味で）もはや時代遅れである．

モデルによる自己内省やコミュニケーションは，それを学習するうちに個人の思考手段として定着し，例えば大部分の日本人が日本語で思考するように，開発者がシステム化の対象をモデル図式で思考するようになる．モデル駆動の開発が普及し定着して，システム開発の常識になるには，上記の意味での学習が多くの開発者に必要となるが，それには一連のモデル表記法（つまり言語）が違和感なくなじむものであることが望ましい．また，図式によるモデルは，特定の視点での分かり良さを促進する一方，厳密さが要求されるアルゴリズムや，複雑な条件を伴うアクション・シーケンスなどの記述には向かない．それには形式言語などの別の手段が必要になる．

開発成果物

モデルはコミュニケーション手段であると同時に，重要な開発成果物である．手段とはつまりツール（道具）ということであり，本当の目的は別にあってそれを成し遂げるために使うツールだという意味である．一般的な意味でのモデルは大抵そうである．前にあげた原子模型や分子模型，あるいはベン図や略図はすべてツールである．これらのモデルはそれ自体が目的ではなく，何かを説明するための手段であり，目的を遂げたらその場では用済みになる――後でまた使うことはあるが．

OO システム開発におけるオブジェクト・モデルは，従来のシステム開発用語でいう外部仕様や内部仕様に相当するものであり，主要な開発成果物である[*2]．したがって保守の対象となる．成果物という意味では，重要なのは紙に描かれたハードコピーとしての図式の方ではなくて，計算機が扱う電子化された（machine　readable な）モデルの方である．モデリングは，モデルを連続的に詳細化していく作業であり，どの段階のモデルを成果物として識別するかがはっきりしないように見えるが，分析段階のモデルと設計段階のモデルの双方を成果物として保守の対象にすべきであろう．前者は外部仕様に相当し，後者は内部仕様に相当するからである．

保守段階におけるシステムへの変更・追加が，対象ビジネス領域に関するものか，設計に依存するものか，あるいはプログラミング上の問題かによって，それぞれ影響する成果物が異なる．これらの成果物が，常時一貫してシステムの最新状態を表しているように維持する必要がある．仕様に不備や変更がある場合でも，仕様書を保守せずソフトウェア・コードだけを最新状態にして済ましてしまう傾向が見られるが，こ

れはまったく逆である．その積み重ねが後でどのような結果を生み出すかは，ここで説明するまでもない．これでは仕様書が単なる開発の「手段」に成り下がってしまい，成果物としての価値を失い膨大なゴミと化す．モデルは重要な開発成果物であり，システムの変化を常時正しく反映している必要がある．

再利用の対象

再利用はOOシステム開発におけるモデルの重要な役割である．オブジェクト・モデルは，データとその操作とが一体化され自己完備したオブジェクトを基本単位として表現される．そのためモデルおよびその一部や，これらに対応する実装プログラムが，再利用しやすいという特徴を持つ．後でまた触れるが，最近，ビジネス・オブジェクト，コンポーネント，パターンといった考え方が出てきているのも，この再利用しやすいという特徴による．

モデルは，個々のプロジェクトに限定された成果物というより，複数のプロジェクトや部門間が共有する情報だと考えるべきである．すなわち，これらは参照や再利用のための重要な素材であり，候補である．モデル全体の流用といった短絡的なことではなく，むしろその中の汎用的なパターンや，有用なモデル化の工夫が再利用の重要な候補になるからである．再利用をシステム開発の基本原理とする新しい開発方法論や技術基盤の提案が，最近さかんに行われている．コンポーネント・ベース開発とか，パターン/フレームワークによる開発などと言われているものである．これらのベースとなっているモデルは，すべてオブジェクト・モデルである．

2.2 モデル，アーキテクチャ，ユーザインタフェース

F.P. Brooks, Jr. は，1975年に“Mythical Man-Month”（邦題「ソフトウェア開発の神話」）というエッセイを出し，1995年にその増補版[2]を出したが，その中で，アーキテクチャ（本稿でいうモデルとアーキテクチャを含んだもの）の重要性を一貫して主張している．

すなわち「すべてのソフトウェア構築には，本質的（エッセンシャル）作業と偶有的・副次的（アクシデンタル）作業がある．前者は抽象的なソフトウェア実体を構成する複雑な概念構造体を作り上げることであり，後者はそうした抽象的実在をプログラミング言語で表現し，それをアベイラブルなメモリー量やスピードの制約内で機械語に写像することである．過去の大きな収穫のほとんどは，偶有的・副次的作業を困難にする障害を除去することにより得られた.」と述べている．そして，この「複雑な概念構造体」（つまり原著者が言うところのアーキテクチャ）を創り上げる作業を克服する「銀の弾丸」（即効薬）はないと述べている．

モデルとアーキテクチャ

上記の文献では，モデルとアーキテクチャを区別せず，併せてアーキテクチャと言っている．これは，その著者が「本質的な作業はアーキテクチャ・チーム（アーキテクト）の担当で，偶有的・副次的作業は実装チーム（インプリメンタ）の担当だ」としていること，また「外部仕様の作成はアーキテクトの仕事だ」とも述べていることから考えると，著者のバックボーンがOS/360というシステム・ソフトウェアの開発

だったからであろう．ビジネス・アプリケーションの場合には，モデルとアーキテクチャを区別するべきである[*3]．両者は本質的に視点や目的が異なるものであり，したがってそれを創り出すスキルが異なる．

モデルとアーキテクチャは，情報システムの構築を企業あるいは部門全体でグローバルかつ戦略的な観点で捉えるか，その一部の限定されたビジネス領域に対する個別のシステム開発という範囲で捉えるかにより，その目的や関与する人の層が異なる．グローバルな観点でのモデルは，企業モデルあるいはビジネス・モデルと言われるものであり，アーキテクチャは例えば情報システム・アーキテクチャ計画[*4]と呼ばれるものがそれに相当する．本稿ではこのようなグローバルなコンテキストには触れない．後者の観点でのシステム開発を対象とする．

基本的に，モデルは，これから作成しようとしているシステムのユーザやビジネス担当者の要求定義に基づいて，特定のビジネス領域（広義には問題領域とも言う）を抽象化したものである．オブジェクト・モデルの場合には，その主たる要素は対象ビジネス領域に所在するオブジェクトである．ただし，実際にはモデルは詳細化されて，設計段階では，計算機処理に都合がよいように人工的なオブジェクトが付加される．とはいえ，モデリングの主眼は，あくまでも対象ビジネス領域の抽象化・詳細化であり，そこにある情報の構造，情報の間の関係，および情報の操作(生成，保管，消去，更新，加工，伝達）とそれらに関係する活動，を表現することにある．したがってモデラの視点は対象ビジネス領域である．

アーキテクチャとは，もとはギリシャ語に語源を持つ建築用語であるが，広義には「複雑な構造（システム）の構成要素とそれらの関係」のことを指す．例えば Architecture of the Universe（万物の構造）などという．この定義によれば，広義にはモデルもアーキテクチャの一種だということになるが，本稿でいうアーキテクチャとは，「モデルを計算機上にのせ，一つの情報処理システムとして構成するための，情報技術にもとづく規範や仕掛け」のことを指す． 前述の F.P. Brooks の言葉を引用すれば，「複雑な概念構造体」の内部機構や全体の機構を与えるものである．実装とは基本的に独立したものと考えるのがよい．アーキテクトの視点は情報技術であり，計算機システムである．

アーキテクチャを設計するキーは，存在する多種多様な情報技術や技術規範の選択肢の中から，そのアプリケーションの特性やユーザ要求にかなう技術要素を適切に選択し，場合によってはそれを工夫することである．これらの要素あるいはその集合は，例えば，アプリケーションのモジュール分割（階層化やパーティショニング），処理方式の決定，機能やデータの物理的分散配置，独立した要素間の交信方式などを含む制御機構の設計，さらにはフェールセーフ技術，トランザクション処理技術，セキュリティ機構，そのシステムの稼動時の管理・運用に関わる技術，基盤としてのネットワーキング技術など多方面に及ぶ．

情報技術の中には，構造モデルや制御モデルとして，実世界に見られる構造や行動パターンのメタファになっているものが多い．また，OO 技術と直接関係がないものも多い．技術要素の選択に当たって，国際標準や業界標準への準拠性も重要なキーになる．

アーキテクチャは，拡張性や保守性に優れたアプリケーションを作り上げる重要な決め手である．プログラミングの巧拙によるバグの数の多少や性能の良し悪しは，単にプログラムの変更により対処できるが，良くないアーキテクチャによる品質の悪さは，修復のコストが格段に高くつく．場合によっては，システムの作り直しになりかねない．

限定された情報技術選択肢の中でアーキテクチャ的に閉じていたメインフレーム優勢の時代と違い，オープン・システムではその選択肢の範囲が格段に広く，大きな多様性を持つ．アーキテクチャ設計の重要性はますます高くなっている．

優れた品質のアプリケーションとは，対象ビジネス領域を適切に反映し，拡張性・保守性にすぐれ，変化に強い長持ちするアプリケーションであり，したがってその担い手はモデルとアーキテクチャである．本稿のテーマとややそれるが，さらにこれらと同様に重要なのはユーザインタフェース（つまり GUI）である．

ユーザインタフェース

オープン・システムの浸透と PC の圧倒的な普及で，情報システムに触れるユーザの層が広がった．現代のビジネスマンはすべて，何らかの情報システムのユーザである．子供や家庭の主婦でさえユーザになり得る．ユーザインタフェースは，彼らの「電子的な仕事場」（ワークプレース）である．このワークプレースは，ユーザの仕事場にある情報やそこで行うアクションのメタファの集合であるから，ユーザの心の中の仕事場モデルを素直に反映したものであることが望ましい．さらに，ユーザインタフェースは彼らにとって，情報システムの事実上の顔であり，システムに実際に触れて確かめることのできるすべてである．ユーザインタフェースには，システム全体が仮想的に凝縮されているのである．

ワークプレースはユーザのビジネスに直接依存する．オフィス支援のアプリケーションでは，これはいわゆるデスクトップ・メタファになるが，業務系や情報系のアプリケーションでは，銀行の窓口，製造会社の生産現場，倉庫係，旅行代理店の窓口，病院の診察室など，それぞれの対象ビジネス領域に応じて固有のメタファがある．さらに経営者，管理者，担当者といった層の違いからも，それぞれメタファは異なるであろう．

操作性，すなわちユーザインタフェースが彼らにとって違和感なくなじむかどうか（認知心理学的・人間工学的に適切なメタファが設計されていること）は，従来以上に重要な品質特性となる．逆説的にいえば，ユーザインタフェースが優れていると，システム自体の多少の性能の悪さや信頼性の低さ，あるいは機能の欠如といった欠点が隠れてしまう――美人は七難隠すということわざがある．ユーザの心の中のモデルの表出には，オブジェクト・モデリングが援用できるであろう．

ユーザインタフェースの設計は本来，情報技術以外に心理学，人間工学，医学，言語学あるいは美術的センスなど，広範な分野にわたる学際的な作業である．情報処理業界において，現在のところその重要性に関する認識は低いが，将来は，今以上に重要な側面になると思われる．いわゆる人工物（artifact）が備えるべき望ましいユーザインタフェースの特性について，文献[1]に分かりやすい説明がある．

2.3 開発チームメンバの役割

文献[2]の初版（増補版でも）の中で F.P. Brooks が深い考察を与えているように，言うまでもなく，システム開発活動は「人間のチームワークによる創作活動」である．チームワークでは，そのメンバの役割（ロール：role）が重要である．これはよくオーケストラに例えられるが，別の卑近な例でいえば次のようなことである．

野球チームには投手，捕手，内野手，外野手という役割があり，サッカーチームには FW，MF，DF，GK という役割がある*5．各メンバはいずれかの役割を担当し，状況に応じて互いに連携しあうことにより勝利を追求する．役割の遂行にはそれ固有の「知識と技術とその適用能力」（つまりスキル）が欠かせない．すぐれたメンバは，そのメンバに割り当てられた役割固有のスキルレベルが優れていると同時に，他のメンバの役割とそのスキルレベルを理解して，臨機応変に互いに協調できる（連携プレー）能力を持っている．すなわち，役割を果たすためのスキルと他との協調性とを併せ持つ．そして，草野球や草サッカーのチームならいざ知らず，プロならば自分の役割に誇りと自信を持ち，そのスキルを極めるべく絶えず努力している．

システム開発活動もチームワークであるという意味では，これとよく似ている．システム開発チームが持つべき主要な役割は，大別して次の三つである*6．

・モデラ…モデリングを行う
・アーキテクト…アーキテクチャを設計する
・インプリメンタ…設計されたモデルとアーキテクチャを実装する

個々の開発活動においては，これらの役割はいずれも等しく必要であり，何らの上下関係もない．それぞれの役割に必要なスキルが異なるだけである．各メンバはいずれかの役割を担当し（場合によっては兼任し），状況に応じて，他の役割を持つメンバと協調する．

モ デ ラ

システム開発の生産性やその成果物の品質を抜本的に改善できる可能性を秘めた，現在もっとも有力な技術はオブジェクト指向技術である．ただしこの技術は，歴史はあるものの実用技術としては比較的新しく，十分には成熟していない．特に，ビジネス・アプリケーションでの適用実績が少なく，この技術が広く普及し実効あるものとなるには，ビジネス現場での適用によるフィードバックが必要である．

OO 技術へのシフトのキーポイントは，システム化対象を捉える見方の転換であり，それがパラダイムシフトと言われるゆえんでもある．文献[2]の後半で，オブジェクト指向技術の成長が遅い原因がさまざまな角度で考察されているが，そこに「OO は本来デザインの一種であり，デザインの原則を教えるべきだったのに，特定のツールや言語を使用することであると教育してきてしまった」という主旨のコメントがある．ここでいうデザインは分析・設計双方を意味している．オブジェクト指向技術の推進のキーは，オブジェクト・モデリングを行うモデラの育成である．

世の中には，従来型の技法でシステム分析を担当し要求仕様や外部仕様を作成してきた，潜在的な OO モデラがすでにいる．このような人たちの多くは，対象ビジネス領域に関する知識をすでに持ち，オブジェクト・モデリングへの最短距離にいる．し

かし，OO 技術への転換は，相当の投資を伴う大きなシフトであるが故に，みんなが一斉にシフトするのは無理がある．原理をわきまえ，この技術の現状・利点・課題を正しく認識し，問題に応じて柔軟に対処できる，指導的な役割を果たすモデラの育成が急務である．

オブジェクト・モデリングはまた，データベース・モデリング技術(データの視点)とプログラム設計技術（処理の視点）の双方の側面を融合した，新しいモデリング技術にもとづいている．いずれの技術背景を持つモデラも，一方の技術にのみ捕われることなく，柔軟な視点を持って事に当たる必要がある．

前にも触れたように，設計モデリングの段階では，アーキテクチャや計算機処理上の都合で人工的なオブジェクトが追加される．この段階はモデラとアーキテクトの共同作業になるが，その主担当はモデラである．アーキテクトとインプリメンタの持つスキルの違いについて，以下手短に触れる．

アーキテクトとインプリメンタ

アーキテクトには，広範かつ多様な情報技術要素およびその標準に関する知識と，その応用能力が要求される．同時に，これらの技術を実装する主要な実装製品(DBMS，ミドルウェア，OS など）に関する基本的な知識（詳細な使用法ではなく，主要な機能，利点や短所，設計思想など）も不可欠である．一人の人間がこれらすべての分野の技術や製品知識を身につけるのは不可能なので，各開発チームには，分野別に専門性を身につけた複数のアーキテクトが必要であり，したがってアーキテクチャ設計は，各専門家の集まりである支援チームが主体になって行うことになる．

インプリメンタには，各種の実装製品，実装用ツール，プログラミング言語などに特化した，詳細な知識，プログラミング技術，実装設計技術が要求される．既存の主要なアルゴリズム，データ構造，アクセス方式などに関する知識とその応用能力も必要になる．

アーキテクトもインプリメンタも，モデルを理解し，必要ならばそれを拡張し改良することができなければならない．モデルもアーキテクチャも，それを実装する具体的な製品とは独立していることが原則である．設計されたモデルとアーキテクチャを実装するための個々の製品の選択は，実際にはアーキテクトとインプリメンタの共同作業となるが，主担当はアーキテクトである．論理的なモデルから実装へのマッピングを行う実装設計は，インプリメンタが担当する．

3. オブジェクト・モデリング

3.1 OO 技術は総合ビタミン剤

オブジェクト・モデルは，オブジェクト指向技術を用いて作成される．このモデルは複数の図式で表現され，モデリング・ツールを使って生成される．従来の分析・設計技法とオブジェクト指向技術によるそれとの違いを特徴づける主要な特性は，カプセル化と継承と多相性である[*7]．したがってオブジェクト・モデルとは，これら三つの特性をすべて備えたモデルのことである．これらの概念の詳細については他書をあたってほしい（例えば文献[3],[4],[13]の前半，[21]）．

オブジェクト・モデリングを実行するに当たり，これらの特性を別々に捉えるというよりも，1セットになった原理として理解し，その真の意義をしっかりと身につけることが肝要となる．この原理を実効あるものとするための核となる考え方を強いて上げれば，次の三つになるであろう．

・カプセルとインタフェースによるブラックボックス化
・共通性の着目
・再利用

カプセルとインタフェース

カプセル化は，抽象データ型，情報隠蔽，モジュール性という特徴を併せ持つ．継承の概念は，クラス(オブジェクトのテンプレート)の階層を導出させる．多相性は，基本的には振る舞いを総称的に扱えるようにする機構である．3.1節の冒頭であげた三つの特性のうち，もっとも重要な特性はカプセル化である．ブラックボックス化されたカプセルであるオブジェクトが，公開されたインタフェースを介して他のオブジェクト（カプセル）と交信する．各カプセルの中の情報は，隠蔽されているので直接アクセスできず，メッセージを介して間接的にアクセスする．これらのカプセルの候補が，問題領域にあるオブジェクトである．オブジェクト・モデリングでは，これらのカプセル（したがってクラス）を決め，そのインタフェースを決めることが，もっとも重要な作業目的となる．最終成果物であるアプリケーション・ソフトウェアは，結果としてこれらのカプセル（クラス）の階層化された集合として構成される．計算機の動作原理と密着したアルゴリズム（手続き）は，各カプセルの中に（個々のメソッドとして）隠蔽され，陽には見えない仕組みになっている．

抽象データ型，情報隠蔽，モジュール性，継承，多相性などの技術は，これまでそれぞれ個別に提案されすでに存在していたものである．オブジェクト指向技術は，これらを同時に取り込んで一つの技術パッケージとしているにすぎない．すなわち「オブジェクト指向アプローチの魅力は，総合ビタミン剤のようなもので，これらを一挙に（技術者の再教育により），全部手に入れられることである.」[2]

共通性の着目

クラスは，問題領域にある情報や活動を概念単位として切り出し，一つのカプセルとしてモデル化したものである．クラス階層は，個々のクラスの特性（属性，関連，操作）の共通な部分をくくり出した汎化クラスを識別したり，あるいは，既存のクラスの特性をその一部に持つ特化されたクラスを識別することにより得られる.つまり，汎化と特化はクラスの特性の共通性に着目している．

操作の多相性は，クラスの中に定義された個々の振る舞い（メソッド）における主体，対象，手段などを捨象して，それらに共通な動作やサービスに着目して総称的な（多義的な）操作を見出すことにより得られる．例えば，「ドライブする」「歩く」「電車で行く」という具体的なメソッドに対して，これらを総称して多相的な操作「移動する」を定義できる．多相性の意義は，呼び出し側が同一の総称的な操作名を使って，具体的なメソッドは，実行時のコンテキストに依存して実行させることができること

である．これにより，呼び出し側を変えずに，新たに具体的なメソッドを追加したり，既存のメソッドを置き換えることができる．モデリングの観点では，これも共通性の着目である．

パターンやフレームワークは，複数の問題領域にしばしば現れる典型的な構造や振る舞い様式に着目し，そこに所在する複数のオブジェクトとそれらの静的・動的な関係を，まとめて定型化したものである（分析パターンという）．また，分析モデルを計算機上に写像する際に有用な，オブジェクト指向設計での定石となるべき典型的な構造や制御機構を定型化したパターンは，設計パターンと呼ばれる．これらは，多数の実例の中の類似性に着目して，それらの共通部分をモデル化したものである．すなわち共通性の抽出である．

このように考えると，共通性への着目はオブジェクト指向の基本だということになる．が，一方このことは，複数の概念の総称概念を識別したり，総称名を付けるといったことがしばしば発生することにもなる．したがって，クラスおよびそれらの総称クラスの命名の仕方が重要になる．総称的な名前は，類似した特徴や機能の表現を統一できるという利点を持つ一方で，あいまいさを助長するという欠点を持つからである[6]．モデルを分かりやすくし，正確な意味を伝えるために，クラス特に人工的な抽象クラスの命名は慎重に行う必要がある．モデルは単に一つのプロジェクト内の共有物にとどまらず，広く再利用される可能性があるからである．

先に述べたカプセルとインタフェースによるモジュール性の実現は，すぐれたソフトウェア構造を導き出す第一条件である独立性[*8]を促進する．インタフェースさえ変更がなければ，クラスの実装の変更や一部の置き換えが，その外部には何らの影響も与えないからである．すなわち変更の影響が局所化される．オブジェクト・モデリングは，互いに密結合した事項が凝集し自己完備した多くのオブジェクトが，互いにメッセージ・パシングにより疎結合した世界を生み出す．かくして，最終成果物であるソフトウェアを柔構造にする．

共通部分をくくり出して他と異なる部分を切り分けたり，密接に関連するものをまとめてくくり出すといった認知行為は，日常われわれの思考の中で，意識しているいないに関わらず絶えず行っている．オブジェクト指向技術の持つさまざまな特性は，われわれが自然に備えている認知行為をシステム開発の中で有効に働かせるための，一連の技術的手段を与えているのであり，結果としてそれが，優れたソフトウェア構造の構築に導いていると考えることができる．

再　利　用

カプセルとインタフェースに基づくモジュール性により，個々には内部が複雑な部品（カプセル）でも，円滑なインタフェースが設計されていれば，これらを組み合わせてさらに手の込んだ構造が組み立てられる．この構造をさらにインタフェースで覆って隠蔽し，一つの大きなカプセルとして他と組み合わせれば，もっと複雑な構造が作れる．これは，カプセルの集約階層を作り出し，部品の再利用の典型となる(図 2)．また特性の継承は，重複排除による再利用効果をもたらす．

しかし，ソフトウェア工学でいう再利用は，このようなソフトウェア実体としての

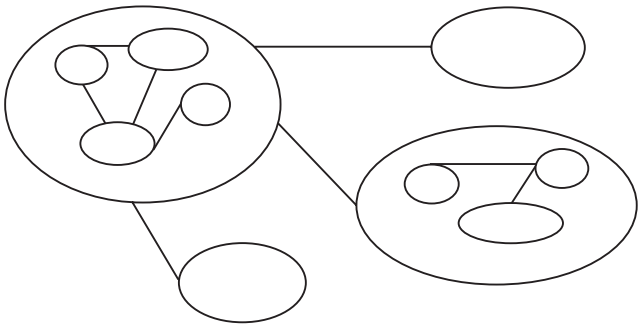

図 2　カプセルの集約階層

部品の再利用にとどまらない．仕様，モデル，技法，パターン，フレームワーク，アーキテクチャなど，システム開発活動に関連するあらゆる成果や手法が再利用の対象になる．オブジェクト指向システム開発のベースは，図式化された大小さまざまなモデルであり，これらは視覚的であるために理解しやすく，それぞれ再利用の単位になり得る．オブジェクト・モデリングにおける再利用のキーは，内外のプロジェクト成果物としてのモデルにとどまらず，文献などに見られる模範的なモデルやパターンを広く参照して，それらをまね，かつ手元の問題に応用することである．再利用とは，重複の排除や手数を減らす行為というより，巨人（先人の為した多くの実証済みの成果）の肩の上に乗ってさらに高みを目指す（付加価値を与える）行為である．

再利用はオブジェクト指向固有の技術ではなく，従来のシステム開発でも一部では行われてきた．今になって多くの話題をもたらしているのは，オブジェクト指向技術のさまざまな特性が，全体として，再利用を促進させる技術的手段を提供していて，従来より格段に再利用を進めやすいからである．とはいえ再利用は，個々のプロジェクトの課題であると同時に，企業や部門といったレベルで対処すべき組織的な課題でもある．このような組織的な再利用を実現するには，そのための専門スタッフ組織と技術基盤（つまり情報システム）が必須である．

ソフトウェア技術の発展の歴史は，再利用の歴史だと捉えることもできる（図 3）．入出力ハンドラ，OS，標準ライブラリ，ミドルウェアへと，基本的なものから順にユーザ・コードから切り離されてファームウェア化（再利用）され，現代は最後の砦であるアプリケーション部分の再利用へと向かっている．したがって，上述した再利用への挑戦は歴史的に必然な流れに沿ったものでもある．

3.2　OO 技術の適用

繰り返すが，オブジェクト・モデリングは，開発の対象である問題領域をオブジェクト指向技術を用いて複数の視点から抽象化し，それらをいくつかの整合された図式として表現する．このモデルは，問題領域を素直に反映しかつ変化に強いソフトウェア構造を生み出すのに貢献する．このモデルは，オブジェクト指向技術を反映したデータベース（OODB や ORDB）やオブジェクト指向プログラミング言語（OOPL）とよくマッチする．

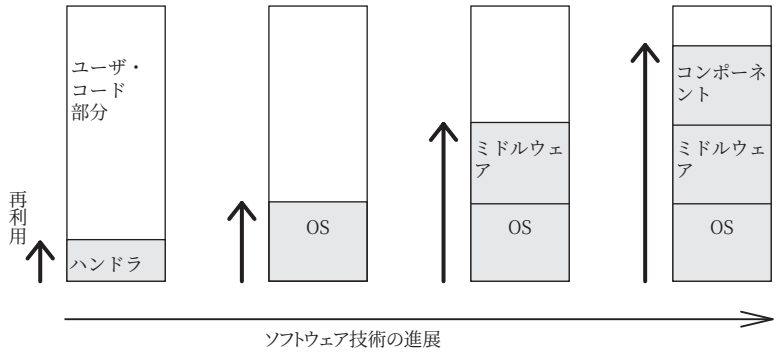

図 3　ソフトウェア再利用の流れ

とはいえ，この技術は万能ではなくまた未成熟である．モデリング技術だけでなく，それを支援したり実装するツールや製品についても同様である――特に実行時の効率とのトレードオフや，関係 DB などの従来技術との親和性など．ものによってはすでに 30 年以上もの長い経験の積み重ねを有する現行の開発技術に比べて，ビジネス現場での実績という意味ではこの技術はまだ日が浅い．したがって経験の積み重ねと使い分けが必要である．実際の開発場面での適用は，組織が保有するこの技術のスキルや対象業務の規模や特性を勘案して，慎重に行う必要がある．開発経験を重ねる度にじわじわと漢方薬のように効果が現れる技術である．特に，この技術を初めて採用しかつ白紙から開発する場合には，納期や出荷までの開発コストという側面だけで見れば，従来型の方がたぶん適しているであろう．

それにも関わらずこの技術が話題になるのは，言うまでもなく，従来のシステム開発を抜本的に改革する可能性をこの技術が持っているからである．安易にこの技術に期待して，断片的な成果に一喜一憂し，過大あるいは過小評価するといったことは避けねばならない．マネジメントの観点からは，確固とした方針のもとに，この技術を着実にものにしていく姿勢が望まれる．

多様な方法論

周知のように，1994 年に J. Rumbough と G. Booch が，互いのモデリング技法を統合する作業を開始し，翌年には I. Jacobson がそれに加わり，その成果として 1996 年に UML 0.9 が発表された．1997 年にはさらに OMG がこれを標準図式言語 UML 1.1[*9] として制定した．当初彼らは方法論（開発プロセスを含むもの）の統一を目指したが，結果として方法論は統一せず，モデル図式記法の強化と統一を行った．

UML 1.1 のサマリー文書[9]では次のように明言している．「プロセスはその性格上，組織，慣習，および与えられた問題領域に沿って仕立て上げられるべきものである．あるコンテキストではうまく機能しても，別のコンテキストでは災いになりかねない．特定のプロセスの選択は，問題領域，実装技術，チームのスキルなどに依存して大きな多様性を持つ．」すなわち，開発プロセスを含む方法論の標準化は事実上意味がなく，不可能なのである．ただし，プロセス・フレームワーク（個々のプロジェクトが，その中でインスタンス化できるような標準フレームワーク）の採用の可能性はある[*10]．

OO 分析・設計方法論（オブジェクト・モデリング技法）は現在多くのものが提案されていて，代表的なものだけでも十指に余る．オブジェクト指向の開発プロセスや開発方式（工法）が多様にある上，各方法論が重視する視点も違うからである．前者については次項で触れるが，後者は例えば，分析重視，設計重視，実装基盤重視，構造化技法や DOA（データ中心アプローチ）との親和性重視などの違いである*11．

図式の表記法が統一されたとはいえ，その使い方や選択肢が多様な幅を持つために，さらに方法論が多様化することも考えられ，なかなか集約されそうには思えない．上記の各モデリング技法の詳細については，注＊11 に挙げた参考文献をご覧願いたい．日本ユニシスが開発し，現在社内で使われているオブジェクト・モデリング技法 BOAD の紹介が，当「技報」に掲載されているので，あわせて参考にしていただきたい．

上述したプロセス・フレームワークは，システム開発プロセスと方法論のコンテキストにおいて，本質的ないし共通なものと各プロジェクトに特化した部分とを明確にし，これらを体系化することに貢献するものとして期待される．

開発プロセス

OO 技術は開発方式や開発プロセスとは，基本的に独立している．現在支配的なウォータフォール方式による開発でも，十分利用可能である．この方式の欠陥を補うために，従来からスパイラル方式，ラウンドトリップ方式，RAD といった方法が提案されてきたが，いずれも広く普及するには到っていない．OO 技術がしばしばこれらの新しい方式と関連づけて語られる主要な理由は，この技術とそれらとの間に親和性があるからである．この種の方式はいずれも，作業の繰り返しを許す反復的（イテラティブ）アプローチ，段階的に機能を追加しながら提供していく増殖的（インクリメンタル）アプローチ，およびプロトタイピングのいずれかあるいはその組み合わせである*12．それぞれの特徴や概要については，例えば文献[21]の 6 章を参照願いたい．

これらの新しい開発方式について総じて言えることは，従来のウォータフォール方式の持つ欠点を補うために，要求や仕様の変更・追加を必要な都度随時取り入れられること，また開発チームの内外でシステムの出来映え（品質や性能）が開発中にも見えるようにすること，を重視して，外部の変化に柔軟に対処し，作業の進捗や成果物の完成度をビジブルにしようとしていることである．このような開発方式を採用しかつ，モデル駆動による開発や再利用の促進による迅速な開発を行うためには，

① 開発作業を一貫して支援する各種 CASE ツール
② これらのツールを組み込みかつツール間の情報交換をスムーズに行える開発基盤
③ 再利用ライブラリと再利用支援システム

が必要である．

新しいオブジェクト指向リポジトリが，これらの要請にまとめて応える基盤として期待されている[22]．リポジトリを活用することによって，さらに次のようなことが可能になるであろう．すなわち，

① 開発プロジェクトの構造（役割，組織，作業，成果物，規則，リソースなど）

に関する情報をモデル化して格納して公開し――プロジェクトのビジビリティ

② さまざまな変更を反映してこれらの情報を最新状態に維持し――構成管理

③ 開発作業や成果物の進捗をビジブルにし――リスク管理や進捗管理

④ この開発基盤を再利用支援システムと連携させ

結果として，

⑤ プロジェクト・メンバやマネージャの役割を情報・機能双方の側面から総合的に支援する．

上述した新しい開発方式については，従来型のウォータフォール方式でこれまで用いられてきたさまざまな管理手法，例えば計画・立案手法や，チェックリスト，チェック・マトリクス，レビューなどによるリスク管理や進捗管理など，に対応するガイド的な手法が確立されているとは言い難い．現在のところ，各組織が個別に工夫して対応する必要がある．この点についても，実際の開発経験からのフィードバックが必要である．

アプリケーション・システムを一括して再利用するソリューション・パッケージ（いわゆる ERP パッケージ）や，個々の単体としてのコンポーネントを選択して再利用しこれらを組み合わせるスタイルの開発（コンポーネント・ビルディング）が，最近話題になっている．この種の方式は究極のソフトウェア再利用であり，F.P. Brooks も前掲の文献[2]の中でその期待を熱っぽく述べている．その主旨は，これらのアプローチが彼の言う本質的作業を無しで済ますことになるからだというのである．開発プロセスのいくつかは簡略化できるであろう．が，これらのアプローチがモデリング作業を不要にするわけではない．対象ビジネス領域のモデル（as-is あるいは to-be）の存在が前提となり，このモデルと上記の素材が持つモデルとの整合性を確認する必要がある．前に触れたグローバルな意味でのモデルとアーキテクチャが必要なのは言うまでもない．

さらにこれらと関連してビジネス・オブジェクトという概念がある．コンポーネントにしろビジネス・オブジェクトにしろ，あるいはそれらの集約としてのソリューション・パッケージにしろ，それが豊かな市場性をもたらすには，個々の単体としてのコンポーネントのインタフェースの標準化が必須である．OMG はその重要性を早くから認めて，1994 年からビジネス・オブジェクトの標準化に取り組んでいる*13．現在のところまだその成果は出ていない．ビジネス領域の多様性，インタフェース定義を簡便にするための高位言語，分散環境での実行時の相互運用，典型的なオブジェクトの選別と体系化といった，多くの課題を解決する必要がある．部分的な標準は近い将来制定されるかもしれないが，全体として依存できるような成果はすぐには期待できないであろう．

3.3 ブラックボックス化

すでに述べたように，オブジェクト・モデリングさらにはオブジェクト指向技術のキーポイントはブラックボックス化である．これは品質の良いソフトウェアを迅速に生産する技術として期待されているが，一方，それが当たり前の技術として定着するには，いくつかの技術上・管理上の課題がある．ここでは，オブジェクト・モデリングの課題面に焦点を当て，特にブラックボックス化というコンテキストに絞って考察

したい.

対応すべき主要な課題は次の三つであろう.

・的確なカプセルの識別とインタフェースの定義
・テストと品質保証
・ツールの普及と再利用の仕組み

的確なカプセルの識別とインタフェースの定義

問題領域の中からどのようなクラスを抽出し，その特性（属性，関連，操作）をどのように捉え，インタフェースをどう定めるかは，作業する人間の能力に大いに依存する．既存の各種モデリング技法は一般的なガイドは与えるものの，これらを決めてはくれない. 何に着目してどんな共通性を認識し(つまり本質的な構造を見抜くこと)，それをどのようにモデル表現するか（その構造を表現すること）は，関与する人間に大きく左右される．その意味では，モデルは，問題領域に対する一つの「解釈」であると捉えることもできる.

各モデラの有する技術の背景にも依存する．データベース技術，プログラミング言語，人工知能，あるいは構造化技法や DOA などの分析・設計技法といった，技術のバックグラウンドが違うことで，モデル化の視点が微妙に異なってくる．問題領域を正しく把握するために，オブジェクト・モデリングでは，対象ビジネス領域の専門家や実務者の参画（JAD）が必要である．問題領域の本質的な構造の把握といった極めて人間依存的な側面は，モデラの資質と，JAD などの実務を通じての学習に期待するしかないが，そのモデル表現については，模範的なモデルを多く見聞して知識を蓄え，それらを手元の問題に応用することにより技術を高めることができる．パターンやフレームワークはそのような意味でも役に立つ.

システム開発業務というビジネス領域を例に採れば，プロジェクトを構成する一連の作業の構造は，いわゆるワーク・ブレークダウン・ストラクチャ（WBS）として，典型的には図 4 のようなモデルとして表現できる．各作業とその構造は，「アクティビティ」クラスのインスタンスとして生成される．規模によってはこのままでも十分利用できるが，さらにこれを特化して，例えば当社が社内で利用している Team-Method の構造をモデル化してみる．この構造はプロジェクト，フェーズ，モジュール，タスクグループ，タスクという階層構造になっているが，これは例えばおよそ図 5 のように特化できるであろう．この場合図 4 のような総称的なモデルは，慣れないと最初からすぐには作れない.

個々の問題分野に特化した有用なパターン，フレームワーク，コンポーネント，あるいは汎用的なそれらが多く出回ってくると，優れた使いやすいものがどんどん洗練されて流通し活用され，そうでないものが次第に淘汰されるであろう．その決め手は，共通性に着目したカプセルとインタフェースの良否である．旧来のプログラミングに関しては，優れたプログラムを書くためのきめ細かなガイドを与える著名な文献として B.W.Kernighan ほかの「ソフトウェア作法」[24]や「プログラム書法」[25]があったが，オブジェクト・モデリングに関してこれらに相当する有益な文献が現れることを期待したい*14.

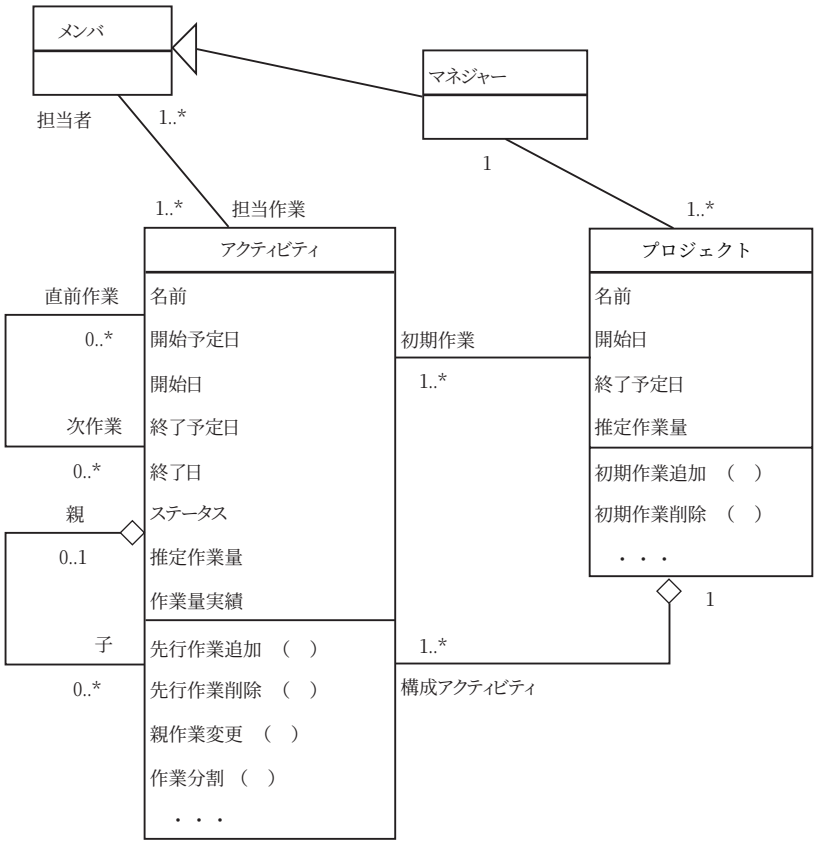

図 4　WBS のモデル

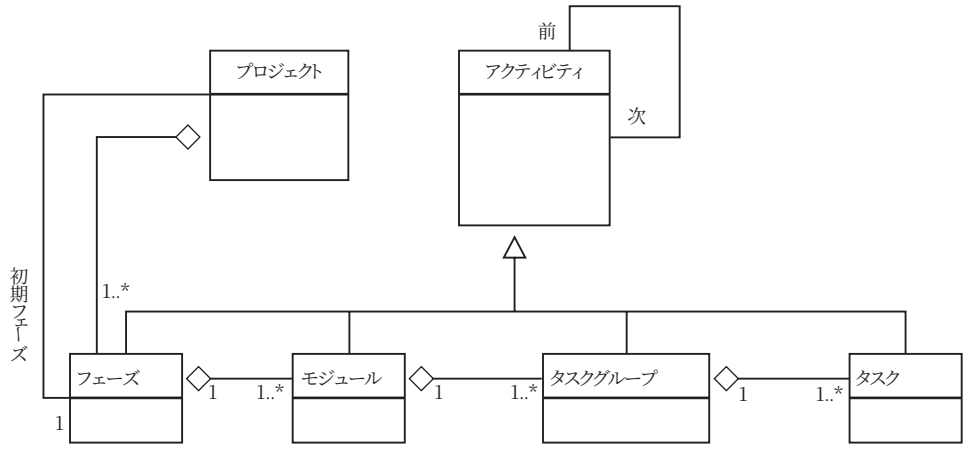

図 5　特化された WBS モデル

テストと品質保証

カプセル化されたクラスの集合としてのソフトウェアのテストは，従来のような一連の手続きの集合としてのソフトウェアのテストとは，そのテスト方法が異なる．オブジェクト指向の場合，公開インタフェースを介して行うメッセージ・パシングの全体が，そのソフトウェアの動的な要件を必要十分にカバーしているかどうか検証することになる．単体としてのクラスの正当性確認，すなわちそのインタフェースと実装とのマッチングのテストは比較的簡単であろう．したがってテストの主眼は，手続きのトレースではなくメッセージ・パシングのトレースになる．

再利用を進めると，部品をブラックボックスとして組み込んでいるために，従来型のテスト方法との違いの落差が大きい．個々の部品としての信頼性が実証済みであること，そのインタフェースのセマンティクスが明解で使う側に誤解がないこと，が要求される．これらの要件が満たされていればよいが，さもないと，最終製品であるソフトウェアがカオスになりかねない．いったん不具合が発生すると，その発生源の追求が著しく困難になる可能性がある．オブジェクト指向によるソフトウェアは，部品としてのカプセルの集まりであるから，これらのカプセル自体の信頼性と，インタフェースが円滑に作動することが大前提になる．

ソフトウェア全体を対象とする最後の大仕事（従来型のテスト）で検証を行うだけでは，非常な危険を伴う．この技術の実績の積み重ねがない初期の段階では，特にそれが言える．従来のような一回限りのテストだけには頼れない．インクリメンタルな方式やプロトタイピングを採用して，最初は小規模なものあるいは主要部の品質を実際に確認しながら，段階的に拡大していくアプローチがここでも必要になる．

ツールの普及と再利用の支援

開発を支援するツールの重要性についてはすでに触れたが，品質保証という意味では，OO 開発におけるテスト手法の確立と，それを支援するテスト・ツールやデバッギング・ツールが従来以上に重要になる．一般に旧来の CASE ツールは，ツールが使用するデータ格納基盤の非互換，およびツール間での情報交換に難があった．IRDS などのリポジトリ標準や CDIF などのデータ交換標準はあったが，普及するには到っていない．

システム開発という業務を多様な側面で支援し，他の業務システム（例えば生産管理や流通システムや CAD/CAM）のように近代化された情報システムとして実現するには，

① 懐の深い汎用かつ拡張性のあるリポジトリ
② モデルおよびそのインスタンスであるデータやオブジェクトがツール間で交換できること
③ これらのツールがリポジトリに統合されること

が必要である．特に②について，情報交換のための形式とプロトコルを定める実効性のある標準が是非とも望まれる[*15]．

また再利用を組織的に実現するために，技術基盤の確立と確固としたマネジメント・ポリシーが欠かせない．前者は，再利用を企業規模で組織的に行うための情報モ

デル（オブジェクト・モデル）とアーキテクチャ，およびそれをもとに実装された情報システムであり，後者は，再利用を計画し各プロジェクトを支援する専門組織の設定と，さまざまな政策の策定である(例えば生産性の尺度基準，再利用に関する報償，部品の流通や情報伝達の仕組み)．支援組織は，再利用促進のための計画と実行，有用な部品の収集・保守・情報伝達，現場プロジェクトへの技術的支援などを行う．

4. UML への期待

UML 制定の最大の便益は，開発者のみならず情報処理業界全体のコミュニケーション手段が統一されることである．UML は，マイクロソフトや IBM を初め有力なベンダの多くが支持を表明していて，OO の世界では現在のところこれに匹敵する図式表記法はないといってよい．ISO でも近くこれを国際標準として格上げする可能性がある．すでに各種の情報処理関連の書物や雑誌（例えば文献[28]）で紹介され，また論文や記事にも UML に準拠した図式が使われ出している．各種の図式化ツールやモデリング・ツールも，UML 図式を採用するべく拡張されている．したがってこの図式表記法は，少なくともオブジェクト指向のコンテキストでは，今後相当早いペースで普及すると思われる．

次項で触れるが，UML を構成する図式の中には，必ずしもオブジェクト指向と密結合していない図式もいくつかあり，これらは一般的なコンテキストにおいても十分通用する．よって，従来型のシステム開発でも利用できる．言い換えれば，図式表記を必要とするあらゆる場面で利用されるようになると思われる．開発者にとって今後この図式表記法は，好むと好まざるとに関わらず必須知識となるに違いない．

4.1 UML のスコープ

UML という統一図式化言語の制定の意義は正確には二つある．一つは，システム開発者に成果物の仕様化，視覚化，作成，文書化のための標準記法を提供し，彼らの正確なコミュニケーションを促進することである．もう一つは，CASE ツール・ベンダに向けて，OO ビジュアル・モデリング・ツールの相互運用を促進することである．これらはそれぞれ別項で触れる．

UML 1.1 が定めている図式は図 6 のように 10 種類ある（ただし現在も改訂中のため，変わる可能性がある）．これからも分かるように，UML が対象とする開発作業の範囲は，最上流のビジネス・モデリングから分析，設計，および実装にまで及んでいる．さらには，カバーするアプリケーション分野についても，事実上すべての分野のすべてのスケールのソフトウェア開発を対象としており，非ソフトウェア・システムのモデル化をも支援しようとしている．

2.1 節でも触れたように，図式モデルは理解性を高めるが，厳密さが要求される仕様の表現には適さない．しかし UML 1.1 には，副作用を伴わない制約や式の厳密な記述を支援する OCL（Object Constraint Language）という形式言語が定められていて，モデル図式中に注釈の形で埋め込むことができる．OCL はまた UML 表記法自身の意味規則の記述にも使われている．さらに，副作用を伴うアクション記述用の形式言語の標準化も検討されている[29]．これにより状態図の中のアクションやクラス図の中の操作の厳密なアクションが記述できることになる．図式の理解性と形式言語

分　類	図式名	用　途
静的構造図	クラス図 （Class Diagram）	クラスとそれらの静的な関係を表現する.
	オブジェクト図 （Object Diagram ）	インスタンスの図. ある時点でのクラス図のスナップショットを表す.
振る舞い図	ユースケース図 （Use Case Diagram ）	外部から見たシステムの機能を表現する. 企業のビジネス機能も表現できる.
相互作用図	シーケンス図 （Sequence Diagram ）	オブジェクト間で交わされるメッセージ交信のシーケンスを表現する （時系列重視）.
相互作用図	協調図 （Collaboration Diagram ）	特定の目的のために協調するオブジェクト群の相互作用を表現する （役割重視）.
	状態チャート図 （Statechart Diagram ）	一つのオブジェクトの状態遷移を表現する.
	アクティビティ図 （Activity Diagram ）	アクション・シーケンス, ワークフロー, プロセスフローを表現する. 企業レベルのビジネスプロセスも表現できる.
実装図	コンポーネント図 （Component Diagram ）	ソースコードや実行時モジュールなどソフトウェア・モジュールの依存関係を表現する.
	配置図 （Deployment Diagram ）	実行時のコンポーネントのノード（プラットフォーム）への配置と相互作用を表現する.
	パッケージ図 （Package Diagram ）	モデルの構成を体系的に表現する（補助図式）.

図 6　UML 1.1 の図式構成

の厳密性を兼ね備えることにより，UML 表記による論理的なモデルがより詳細かつ正確となり，実装作業とのギャップを埋めるのに貢献する．

その他の主な特徴は以下の通りである．

- 特定のプログラミング言語や開発プロセス/方法論とは独立である．
- 表記法自体を拡張したり特化できる機構を持つ．
- スレッドとプロセス，分散と並行性，コラボレーションなど細かな制御機構のモデリングが可能．
- 表記法のシンタクスとセマンティクスを与えるためのメタモデルがあり，そのシンタクスの記述に UML 表記法によるモデル表現を使用し，既述のように意味の細則の記述に OCL を使用している．

4.2　開発メンバにとっての UML

開発者にとっての UML 制定のメリットについてはすでに述べたが，一方その汎用性の故に実際の利用に当たっては，採用する個々の開発プロセスに応じたガイドが必要となる．UML は広範なスコープをカバーしようとしているために，図式の種類が多く，各図式を表現するために使う要素（概念）が多様であり，その上拡張機構や特化機構を持っている．個々の開発プロジェクト・メンバは，これらすべてを使わなければならないわけではないし，十分な理解もないまま野放しにして勝手気ままに使うとかえって混乱する．プロジェクト固有の実状に応じて，その使用法を規定して使い分ける必要がある．

UML の拡張機構や特化機構は，採用する開発プロセスの違いやモデリング技法の違いに対応して，多様なローカル UML を生み出す可能性がある．これらが既存のモデリング技法と合体したり，あるいは UML の独自の拡張や特化を陽に打ち出す新たな技法も出てくるかもしれない．したがって，方法論がさらに多様化する可能性がある．

開発者は，多様な側面を持つ UML を正しく理解し，誤りなく使用できることがまず必要である．実はそれだけでもかなり大変である．その表記法を学ぶだけでなく，手元の問題に最適な形でそれを利用せねばならない．他人の創ったモデルを理解し，模範となるモデルをまねることから始め，それを応用するという態度が肝要である．

企業におけるモデリング技術の育成という観点では，単に UML 表記法の概要を多数の人に一斉に講習し，後は自己学習に期待するというのもそれなりに有効ではあるが，真に有効なのは，UML の精神を掴みそれを知悉する指導的な役割を持つキーマンをまず育成し，徐々に増やしていくことである．まだ UML そのものが成熟していない状況では，一斉教育にだけ頼るのは問題がある．その意味でも，3.3 節で触れた「ソフトウェア作法」のような「モデリング作法」の登場を期待したい．

4.3 CASE ツールへの影響

図式表記法の統一は，CASE ツール間の情報交換を従来より容易にする．UML メタモデルの存在は，各ツール自身のモデルとアーキテクチャ（つまり仕様）の標準化を促し，ツール開発をも容易にする．このメタモデルを拡張あるいは特化して必要な操作（メソッド）を付加することで，ツール自体のモデリングが容易に行えるからである．

UML は，モデルあるいはモデル要素間の「依存（dependency）」という総称的な概念を導入していて，これによりアプリケーション自体のモデルとは別の視点で要素間のさまざまな関係をモデル化できる．例えば分析・設計段階の特定のクラスと，特定のプログラミング言語でのその実装クラスとを関係付ける（implement 関係）．あるいは，特定のモデル要素とそれを詳細化したモデル要素とを関係付ける（refinement 関係）．この二つの例は，いわゆる追跡可能性（traceability）を陽に支援するモデリング要素を標準として導入することによって，モデリング・ツールやリバースエンジニアリング・ツールがそれを実装するよう促していることになる．

UML には，開発者への標準図式記法を提供することに加え，CASE ツールの標準化や互換性の向上および機能強化を促すことにより，トータルとして OO システム開発を支援して行くという強い意志が感じられる．しかしまだ生まれたばかりであり，これを支援するツールも出始めたばかりである．企業の実際のビジネス場面での適用を重ねて，これをフィードバックし，さらに実効あるものにしていく必要がある．

オブジェクト・モデリングの浸透はそのための第一歩でもある．オブジェクト・モデリングへの最短距離にいる人たち（潜在モデラ）が，この技術の有する可能性に一人でも多く共感し挑戦することを期待したい．

5. おわりに

本文中でも確認したように，システム開発活動は本質的には人間のチームワークに

よる創造活動である．関与する人間が鍵を握っている．たとえ OO 技術がこの活動を大きく改革する可能性を持つとはいえ，この事実は変わらない．文献[13]に，われわれ技術者にとって非常に頭の痛い次のようなコメントがある．「良い設計は優れた設計者から生まれるのであって，優れたツールから生まれるのではない．優れたツールのマイナス面の働きは，良くない設計者がひどくまずい設計を，彼が以前にやった以上に迅速にやってしまうのを助けてしまうことである．」とはいえ，こんなことで憂鬱になる必要はない．モデリングの第一歩は，問題領域を素直に理解することであり，そこにある情報や活動の真の意味や本質を見分けることである．

＊ 1　数学のモデル理論では，形式的体系（formal　system）の表現で使われる記号に対して，適当な数学的対象（例えば集合概念）を対応させることによって得られるその体系の具体例のことをその体系のモデルといっている．つまり，定理（theorem）の集合としての理論（theory）に対する「解釈」として扱われている．そして，与えられた論理体系に対し健全でしかも完全となるような解釈を見出すことが主な目標だとしている――岩波「情報科学事典」.

＊ 2　モデルが外部仕様や内部仕様に取って代わるのではなく，それらの主要な部分を構成するという意味である．実際には図式モデル以外に，用語辞書，命名規約，補足のための詳細説明，各種一覧表などを伴う．外部仕様とは，システムが何をするか（what）を問題領域の言葉で記述したものをいい，内部仕様は，それをどのように（how）計算機上に実現するかを記述したものをいう．

＊ 3　F.P.　Brooks がビジネス・アプリケーションにも関与していれば，多分同様の考察をしたであろう．近年の開発方法論では，モデリングとアーキテクチャ設計を区別する傾向が見られる．

＊ 4　情報技術を的確に選別しそれらを体系立てた，企業ないし部門レベルの情報システムの構築様式のこと．詳しくは文献[5]を参照．ガートナーグループのあるレポートでは，これに相当するものを City Plan（都市計画）といい，もう一方をアーキテクチャといって区別している．

＊ 5　監督やコーチやスコアラーは除いている．ここでは目的の達成に技術的に直接関わる役割に焦点を置いている．

＊ 6　＊5 と同様の意味でプロジェクト・マネージャやリーダや書記などは除いている．

＊ 7　ここでは分析・設計技術の側面から考察しているが，例えばデータベース技術の観点では，マルチメディアのような膨大な不定長データを扱う BLOB 型の導入とか，集約オブジェクトが陽に扱えること，またオブジェクト ID の導入といった特徴がある．

＊ 8　ソフトウェア工学でいう独立性とは，関連する複数の要素において，一つの要素の変更が他の要素に何らの影響も与えないことをいう．

＊ 9　UML のセマンティクスに関する詳しい仕様については文献[7]を参照．UML 表記法に関する市販の解説書としては，例えば文献[8]がある．文献[8]の巻末には「図書館アプリケーション」の分析・設計モデリングと Java による実装のケーススタディがある．UML は細かな点で，概念の重複や意味のあいまいなところ，不都合なところがまだあり，現在もさらに改訂中である．

＊10　現在 OMG の中で，個々の開発プロジェクトのプロセス/方法論を定義するための，標準のフレームワークとそのファシリティを定めようとする動きがある[10].

＊11　相対的にオブジェクト分析に手厚い技法として例えば，OOSE[11]や P.　Coad の技法[12]がある．前者はユースケース概念を初めて導入した技法で，ユースケース駆動とも言われる．後者は，一連の定石的なストラテジと分析パターンを与えている点に特徴がある．Booch 法[13]は比較的設計・実装段階に手厚く，多くの図式を導入している．OMT[14]や Fusion 法[15]は，分析・設計・実装の全体をほぼ等しくカバーしている．後者はイベント/アクションの記述に形式言語を導入している．IBM の VMT[16]はツールや実装基盤との結びつきが強い．従来技法との親和性を重視しているものとして，例えば Martin/Odell 法[17]や P. Sully の方法論[18]がある．また，共通に使える技法として多くの方法論でも援用されている Wirfs-Brock の技法[19]がある．

＊12　ほかに，オブジェクト指向固有の開発方式として噴水モデル（fountain model）[20]がある．

＊13　ビジネス・オブジェクトに関して，OMG における 1998 年 2 月時点での検討中の仕様について，文献[23]に紹介されている．この仕様はその後見直され，現在もまだ検討作業が続いている．

＊14　例えば設計パターンに関する文献[26]などは，そのはしりかもしれない．

＊15　OMG では，モデルの情報交換形式の標準[27]を策定中で，例えば XML をベースとする方式などが検討されている．

参考文献

［1］ 佐伯胖「新・コンピュータと教育」：岩波新書, 1997, 岩波書店.
［2］ 滝沢徹ほか訳/F.P. Brooks, Jr.「人月の神話：原著発行 20 周年記念増補版」, 1996, 星雲社；(原著：" The Mythical Man-Month, Anniversary edition ", 1995, Addison-Wesley).
［3］ 二木厚吉監訳/B. Meyer「オブジェクト指向入門」, アスキー出版, 1990；(原著：" Object-Oriented Software Construction ", Prentice Hall, 1988；現在第 2 版が出ている).
［4］ 所真理雄ほか「オブジェクト指向コンピューティング」, 1993, 岩波書店.
［5］ 手島歩三ほか「情報システムのパラダイム・シフト」(2 章), 1996, オーム社.
［6］ 木下恂「ソフトウェアの法則」：中公新書, 1995, 中央公論社.
［7］ OMG " OML Semantics, Ver. 1.1, 1 Sep. 1997 ", OMG Document : ad/97-08-04.
［8］ 杉本宣男ほか監訳/H.E. Eriksson「UML ガイドブック」, トッパン, 1998；(原著：" UML Toolkit ", Wiley, 1998).
［9］ OMG " UML Summary, Ver. 1.1, 1 Sep. 1997 ", OMG Document : ad/97-08-03.
[10] OMG " Process Working Group White Paper on Analysis & Design Process Engineering, Ver. 1.0 ", OMG Document : ad/98-07-12.
[11] 西岡利博ほか監訳/I. Jacobson「オブジェクト指向ソフトウェア工学 OOSE」, 1995, トッパン；(原著：" Object-Oriented Software Engineering ", 1993, Addison-Wesley).
[12] P. Coad ほか" Object Models : Strategies, Patterns, & Applications ", 1995, Yourdon Press；現在第 2 版が出ている.
[13] 山城明宏ほか訳/G. Booch「Booch 法：オブジェクト指向分析と設計　第 2 版」, 1995, 星雲社；(原著：" Object-Oriented Analysis and Design with Applications 2nd Ed. ", 1994, Benjamin/Cummings).
[14] 羽生田栄一監訳/J. Rumbaugh ほか「オブジェクト指向方法論 OMT」, 1992, トッパン：(原著：" Object-Oriented Modeling and Design ", 1991, Prentice Hall).
[15] 横川・ヒューレット・パッカード(株)カストマ教育センター訳/D. Coleman ほか「オブジェクト・オリエンテッド開発設計論：The Fusion Method」, 1994, トッパン；(原著：" Object-Oriented Development : The Fusion Method ", Prentice Hall).
[16] D. Tkach ほか" Visual Modeling Technique ", 1996, Addison-Wesley；(邦訳書あり).
[17] 三菱 CC 研究会ほか訳/J. Martin ほか「オブジェクト指向方法序説―基盤編」, トッパン；(原著：" Object-Oriented Method : A Foundation ", 1995, Prentice Hall).
[18] 本位田真一監訳/P. Sully「オブジェクト指向モデリング」, 1995, 日経 BP センター；(原著：" Modeling the World with Objects ", 1993, Prentice Hall).
[19] R. Wirfs-Brock ほか " Designing Object-Oriented Software ", 1990, Prentice Hall.
[20] B. Henderson-Sellers ほか" The Object-Oriented Systems Lifecycle ", CACM Vol. 33 No. 9, 1990.
[21] 岩田裕道ほか「ゼロからわかるオブジェクト指向の世界」, 1996, 日刊工業新聞社.
[22] Unisys Corp." Universal Repository : Technical Overview (Rel. 1.2) ", 1996, Unisys Corp.
[23] 岩田裕道「OMG のビジネス・オブジェクト標準化と適用上の課題」, ユニシス技報 57 号, 1998, 日本ユニシス.
[24] 木村泉訳/B. W. Kernighan ほか「ソフトウェア作法」, 1981, 共立出版；(原著：" Software Tools ", 1976, McGraw-Hill).
[25] 木村泉訳/B.W. Kernighan ほか「プログラム書法　第 2 版」, 1982, 共立出版；(原著：" The Element of Programming Style, 2nd Ed. ", 1978, McGraw-Hill).
[26] 本位田真一ほか監訳/E. Gamma ほか「デザイン・パターン」, 1995, ソフトバンク；(原著：" Design Patterns ", 1995, Addison Wesley).
[27] OMG " Stream-based Model Interchange Format : Request for Proposal ", OMG Document : ad/97-12-03.
[28] 羽生田栄一「統一モデリング言語 UML 1.1 の概要」, Dr. Dobb 's Journal Japan, 1998, 3 月号.
[29] OMG " Action Semantics for the UML : Request for Proposal ", OMG Document : ad/98-11-01.

執筆者紹介　岩　田　裕　道（Hiromichi Iwata）

1939 年生．1962 年名古屋大学理学部数学科卒業．1962 年 4 月日本ユニシス(株)入社．各種計算受託，OS 開発，言語プロセッサ開発，SE サービス，社内標準整備などに従事．現在，情報技術部技術研究開発室に所属．
著書・訳書：「ゼロから分かるオブジェクト指向の世界」，(共著，1996 年，日刊工業新聞社)，「情報システムのパラダイム・シフト」，(共著，1996 年，オーム社)，「多層型クライアント/サーバ・コンピューテイング」，(共訳，1998 年，日刊工業新聞社)