解説

オブジェクト指向モデリング言語UML(1)

羽生田栄一

(株)オージス総研オブジェクト第1事業部技術コンサルティング室

UMLとは

UMLとは，Unified Modeling Languageの略で，統一モデリング言語を意味している．オブジェクト指向で業務を分析したり，システムを開発したり，ソフトウェアモジュールを設計したりする際のダイアグラムの描き方・記法を定めた米国のオブジェクト技術標準化団体OMG (Object Management Group) の標準である[4]☆1．オブジェクト指向分析・設計の標準表記法といえる．

オブジェクト指向技術を利用してシステムを開発する際には，やはりオブジェクト指向の考え方に基づいて対象業務を分析したりシステムを設計することが望ましい．この作業では一般に，システムを図解してモデル化する．こういったモデルの図解の仕方をそのモデル構成要素の定義とともに統一していこうという動きが1990年代後半に起こり，多数のソフトウェアベンダと開発技法研究者の共同作業の結果定義されたのがUMLである．過去，さまざまあったオブジェクト指向に基づくモデル記述法をうまくまとめて共通化したものである．

UML成立の経緯

オブジェクト指向開発手法の第一歩は，Adaコミュニティの Boochによって踏み出され，以後それはBooch法として，Adaに限らない一般的な設計技法としてまとめられた[1]．また，Smalltalkコミュニティからは，CRCカード法やリスポンシビリティ駆動設計技法が生み出された．

その後，80年代後半から続々といわゆる「方法論」と称されるさまざまなオブジェクト指向開発手法が発表された．Shlaer/MellorやCoad/Yourdonのオブジェクト指向分析・設計方法論が著名である．さらに，RumbaughがGEの同僚とOMT（Object Modeling Technique）法を発表[5]し，業界での主流の1つを形成した．しかし一方で，OdellはIE（Information Engineering）のオブジェクト指向版を発表したり，Jacobsonがユースケース概念を初めて導入したObjectory法を紹介[2]し，それぞれに熱心な信奉者が従うという百家争鳴の状況が生まれていた．

もともとオブジェクト指向分析設計手法の標準化に着手しようというのは，Rational SoftwareのBoochが1993年に発案した．このとき，Boochは，Rumbaughなどに声をかけ，OOPSLA'93において標準化を呼びかけたが，大半の参加者からまだ機が熟さず標準化には弊害があると指摘され流れてしまった．その後，1994年10月にOMT法の提唱者RumbaughがGEからRational Softwareへ移籍したのを契機に，Boochとの間で統一方法論（Unified Method）と呼ぶUMLの前身の共同開発が始まり，一気に動きが加速した．

統一方法論は第0.8版としてOOPSLA'95で発表され大いに盛り上がり，オブジェクト指向コミュニティ内で標準の第1候補としての地位を確保した．この時点で，反対する人たちもあり，Henderson-Sellersを中心にOPEN（Object-Oriented Process, Environment, Notation）連合を形成していた．しかし，エリクソンのソフトウェア関連子会社のObjectoryがRational Softwareに買収され，Jacobsonが同社に参画して事態は一変．Jacobsonが交渉役を買って出るとともに，Jocobsonの提案で標準化の対象をオブジェクト指向手法全般からオブジェクト指向モデリングのグラフィカル言語に限定することとし，開発プロセス等の手法の内容は自由に定義できるようにした．これが功を奏し，より広く賛同者を集めた．

最終的に，1997年11月にUML1.1はOMGによってオ

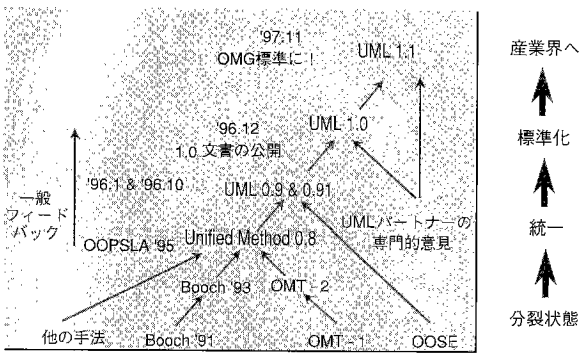

図-1 UMLの発展

☆1 1997年1月に米Rational Softwareなど11社がUMLをOMGに提案し，1997年11月に承認された．バージョン1.1である．

ブジェクト指向モデリング言語標準として正式承認された．

UMLの特徴

UMLでは，既存のほとんどの方法論をサーベイし，開発プロセスとは独立した記法の部分に限定して，オブジェクトモデリングに有用と思われるダイアグラム記法をできるだけ取り込んでいる．さらに，メタモデルによってダイアグラムの見た目（図の文法）とは独立にそのセマンティクス（図の意味）を与えている．モデリング言語と呼んでいることから分かるように，UMLはプログラミング言語と同様に，記法（文法）とその意味（セマンティクス）をともに定義している．これによって，単なるダイアグラムの描き方の統一ではなく，オブジェクト指向モデルを表現したダイアグラムに対する同一の解釈を保証している．

UMLには次の特徴がある．

- 既存のオブジェクト指向手法ではまだ取り上げておらず，表記法が明確でなかった並行処理や並列システム，分散システムを記述するモデル要素を導入したことで，近年のシステム開発の高度な要求にも応えることができる．
- 開発手法や開発プロセスではなく，あくまでも開発時の成果物である分析・設計モデルの記法とダイアグラムに標準化の対象を絞っているため，UMLを採用したからといって特定の手法やプロセスに縛られることがなく，分野やプロジェクトの都合に応じて自由に選択できる．

UMLのドキュメント構成

UMLは次のドキュメント群によって定義されている．

- UMLサマリー：UMLの概要記述と残りのドキュメントへの導入．
- UMLセマンティクス：UMLに基づくモデルを解釈する基礎となるメタモデルを，UML記法と自然言語によって記述．
- UML記法ガイド：UMLによるモデル記法の使用法を例を用いて解説．
- プロセス固有の拡張：開発プロセスやビジネスモデリングへの応用を考慮し，UMLを拡張する際の拡張メカニズムとそれに基づく追加アイコン案を提示．
- CORBAファシリティIDL記述：UMLモデル情報の相互運用性を確保するためのIDLに基づくインタフェース定義を解説．
- UML標準要素：UMLで既定義の標準要素（ステレオタイプ，タグ付き値，制約）を解説．
- オブジェクト制約言語OCL仕様：オブジェクト指向制約言語OCL（Object Constraint Language）を解説．

UMLの言語構造：セマンティクスとメタモデル

UMLでは単なる図の記法だけでなく，その図の表現するモデルがどのような意味を持つと解釈すべきかというセマンティクスも，UMLモデル要素に対するUML自身によるメタモデル記述という形で定義している．今までのオブジェクト指向方法論は，その中で使うモデルの意味については例を示してユーザに推測してもらうという，大雑把な定義しかとってこなかったため，人によって微妙な解釈の違いが生まれ，誤解の原因にもなっていた．そこでUMLでは，Pascalの処理系をPascal自身で自己循環的に定義するのと同じアプローチで，UML自身をUML記法でメタモデルを与えることで定義している．とはいえ，最終的にはUMLの各モデル要素の意味を人間に分かる形で与えなければならない．UMLではこの部分を，自然言語と一部OCLと呼ばれる1階述語論理に似た制約記述言語で定義している．すなわち，メタモデルは，自然言語による概説，UMLクラス図とモデル辞書による抽象構文定義，OCLによるモデル構成規則，自然言語による動的振る舞いに関するセマンティクス記述という4つのセクションから構成されている．また説明の便宜上，UMLメタモデルは，3つの論理的なパッケージ：基礎，振る舞い要素，一般メカニズムから構成されている．各パッケージの構成を図-2に示す．

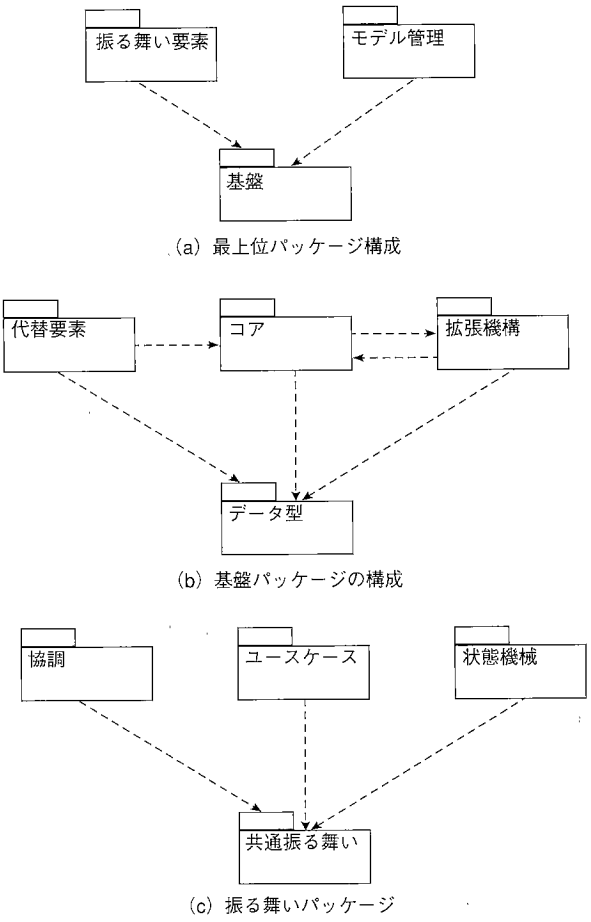

図-2 UMLメタモデルのパッケージ構成

メタモデルの一例として，基礎::コア::バックボーンのUML記述を図-3に示す．

メタモデルを一般のUMLユーザは意識する必要はないといえるが，UMLをサポートするCASEツールベンダやコード生成やリバースエンジニアリングツールの開発者，あるいは上級ユーザたとえばステレオタイプを使ってUMLのモデル要素を拡張して使いたいというユーザはメタモデルの関連部分を理解する必要があるだろう．

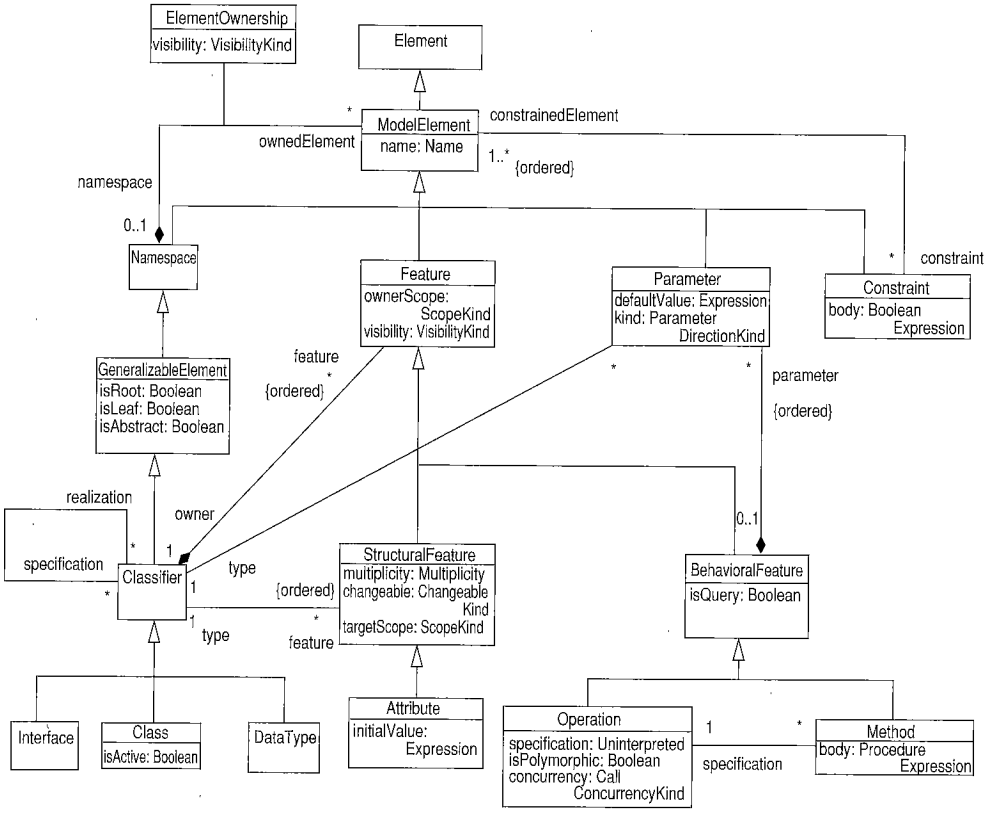

図-3 UMLメタモデルの記述例

構造	クラス／オブジェクト	共通の構造と挙動を持つオブジェクトの集合に対する定義
	インタフェース	オブジェクトの外部公開部（公開操作集合）
	コラボレーション	オブジェクトの集団の協調
	ユースケース	名前の付いたシステムの振る舞い．コラボレーションによって実現される
	プロセス／スレッド	アクティブなクラス．他のアクティブなクラスと並行動作可能
	コンポーネント	再利用可能な部品．論理的な部品と物理的な部品の両方を表現
	ノード	ソフトウェア（コンポーネント）が配置され実行されるハードウェアデバイスの抽象化
振る舞い	相互作用	オブジェクト間のメッセージ交信による情報の伝搬や状態変化のトリガ
	状態マシン	オブジェクトのライフサイクルとそれにともなうアクション／アクティビティ
その他	パッケージ	モデル要素のコンテナ（容れ物）
	ノート	コメント，説明，注釈等のテキスト記述単位

表-1 UMLモデル要素の分類

オブジェクト制約言語OCL

OCL（Object Constraint Language）はモデル記述のための形式言語であり，モデルが整合的であるためにモデル要素や関係の間で満たすべき命題を式として定義できる．

OCLはIBMとObjecTimeによってUMLの構成要素としてOMGに提案されたもので，UML1.1から導入された．UMLのセマンティクス記述にも積極的に使われている．OCLによってUMLのメタモデル定義も以前に比べて整合的なものになったと評価できる．OCLはメタモデルに対してだけでなく，一般の分析設計者がモデルを正確に表したいときに，従来のビジュアルなモデル記述に注釈として追加する形でモデルの厳密性を増すのに使うこともできる．

たとえば，分析モデルにおいて，単に「会社」クラスと「人間」クラスの間に多対多の「雇用」関連が存在するという以上の詳細な制約の記述（この場合は，従業員の年齢に関するビジネスルール）を追加できる．

例：会社.従業員.forAll(人間 p | 18 >= p.age and p.age <= 65)

OCLのもう1つの使い方は，RDBにおけるSQLのようにクラス図で示されたオブジェクトモデルに対する検索操作いわゆるナビゲーション言語としての利用である．ナビゲーションの対象は「フラットセット」であり，フラットセットの要素が1つの場合，その集合を要素と同一視（シングルトンと呼ぶ）する．また，セレクタが何重にも適用されて，結果が入れ子の集合になった場合，一番外側の括弧を残し後はすべて取り除いてしまった集合と解釈する．この規則により，複数の関連をまたいでナビゲーションを行っても集合が入れ子にならず，集合演算や選択操作を簡易化できる．

例：Set{ 1 } = 1

例：Set {Set {りんご, オレンジ} Set{バナナ}} = {りんご, オレンジ, バナナ}

またOCLの文法では，セレクタとして，属性名や操作名，関連名，ロール名，を自由に使って，クラス図を介してモデルを自由に参照できる．

例：従業員
　　self.雇用者.資本金

例：会社.全従業員->size　全従業員の人数が返される

例：会社.全従業員 -> select (役職 ="マネージャ" and 年齢 < 30)

UMLのメタモデル

UMLでは，メタモデルによって4階層の言語構成を規定している．

- モデル要素：モデルを構成する基本要素（クラス，プロセス，パッケージ等）を表す．

• 関係：基本要素間の「接続」（クラス間の関係やパッケージ間の依存関係等）を表す．
• メカニズム：モデル要素や関係に対し，注釈を付けて修飾したり，拡張する手段を提供する．
• ダイアグラム：上記のものをある視点からまとめて提示する図式（クラス図，シーケンス図，配置図等）を提供する．

「モデル要素」層では，UMLで用いられるモデルを構成する基本概念を定義する．モデル要素は大きく**表-1**の9種類に分類でき，これらの要素を関係付けてより大きなモデル概念を定義していくことで最終的に，オブジェクト指向ダイアグラムが定義される．

基本的なモデル要素の間を結ぶのが「関係」層で規定される各種関係である．これらを用いてグラフを作成することで各種ダイアグラムが構築される．関係には，**表-2**に示す4種類があり，これらは同じモデル要素に対して複数回，適用できる．

UMLにおける記法メカニズム

UMLは注釈と拡張という2種類の汎用的な記法メカニズムを提供する．注釈によって既存のモデル要素や関係に対してコメントを付加できる．注釈は仕様定義と修飾に分かれる．拡張は注釈と違って実際に既存のモデル要素や関係に対して実際に性質や規則等のセマンティクスを追加する．UMLは，制約とステレオタイプとプロパティという3種類の拡張メカニズムを提供する．制約とプロパティはモデル表現をより厳密かつ詳細に表現する．ステレオタイプはユーザがUMLの記法を拡張する自由度を与えてくれる．

(1) 注釈メカニズム：仕様定義と修飾

「仕様定義メカニズム」を使い，各モデル要素のテキスト記述ができる．クラスに限らず，属性や操作，関連やパッケージ等の任意のモデル要素に対して仕様定義ができる．

一方，「修飾メカニズム」を使って各モデル要素を表す基本シンボルに対してさらにグラフィカルな装飾を追加できる．属性や操作の可視性を表わすためにクラスシンボル内で各属性や操作の前に置かれるシンボル，+，#，-，はこの修飾メカニズムの適用例である．ユースケースのアクタを表す人型のシンボルもこの修飾シンボルの一種である．

(2) 制約と制約拡張メカニズム

「制約」は，既存のモデル要素自体あるいはモデル要素間で常に成立する論理式のことで，中括弧{ }の中に定義する．UMLでは，制約表現に特定の言語を指定していないので，自然言語や数式を使う．今後，厳密なモデル記述を望む設計者は，UML1.1で導入されたOCLを制約表現に使うのが望ましいだろう．

UMLダイヤグラム

UMLでは**表-3**に示す10種類のダイアグラムをあらか

———	関連（集約は関連の特殊形）	2つのインスタンス間の意味的な接続関係
——▷	汎化関係	モデル要素およびそれと置換可能なサブモデル要素との間の関係
- - - ->	依存関係	別のモデル要素を使用するという関係
- - - -▷	洗練関係	異なる抽象レベルの間の対応関係

表-2　UML関係の分類

構造図	クラス図	モデルの組み立て部品の集合：クラスと関係クラスの構造とクラス間の関係を表現
	オブジェクト図	システムのある時点でのスナップショット
	パッケージ図	モデルの構成の管理，クラス図の特殊形
振る舞い図	ユースケース図	システムのコンテキストと外部機能の設定
	シーケンス図	相互作用するオブジェクトの時間順序系列すなわち，オブジェクトの集団のメッセージ送信の時系列表現．
	コラボレーション図	オブジェクトの集団の相互作用の直接表現：オブジェクトの集団の接続トポロジーとメッセージおよびスレッドの順序表現
	ステートチャート図	1つのオブジェクトのライフヒストリ，すなわちあるクラスに属するオブジェクトのライフサイクル表現を提供する
	アクティビティ図	1つのインタラクション全体における手続きの制御フロー．ステートチャート図の双対表現でワークフローに焦点を当てる図
実装図	コンポーネント図	ソフトウェア・ユニット間の依存関係を示すもので，ソフトウェアモジュールの構成や版管理も表現できる．
	配置図	コンポーネントやオブジェクトの，計算ノード(ネットワーク上のCPUやプロセス，デバイス等の抽象化）上をまたがる分散配置や相互作用を定義．ネットワークやデバイスの物理トポロジとコンポーネントやスレッドの割付け表現を行う．こちらが，ランタイムシステムの構成管理のための図といえる．

表-3　UMLのダイヤグラム

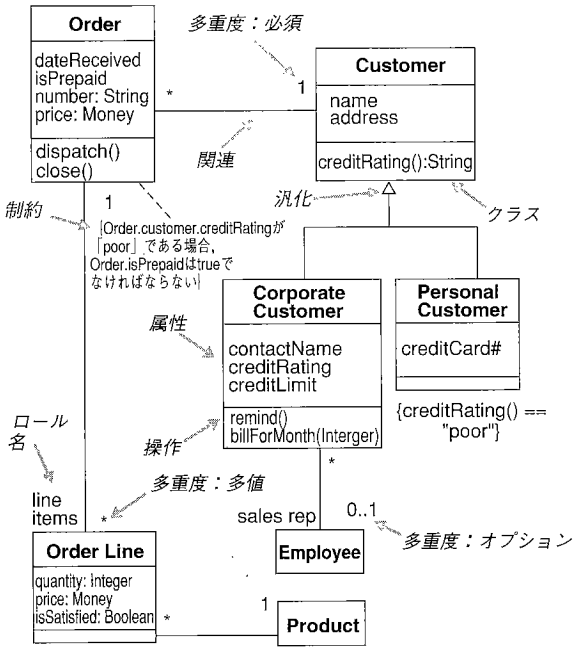

図-4　クラス図の例

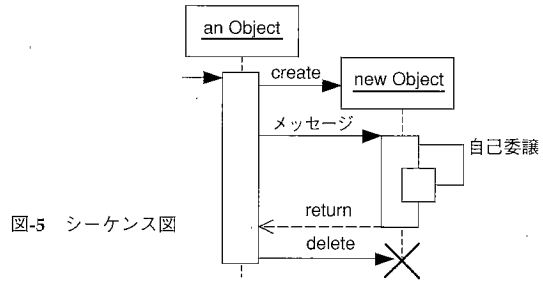

図-5　シーケンス図

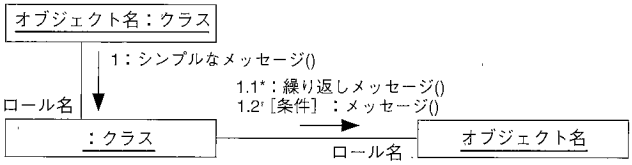

図-6　コラボレーション図

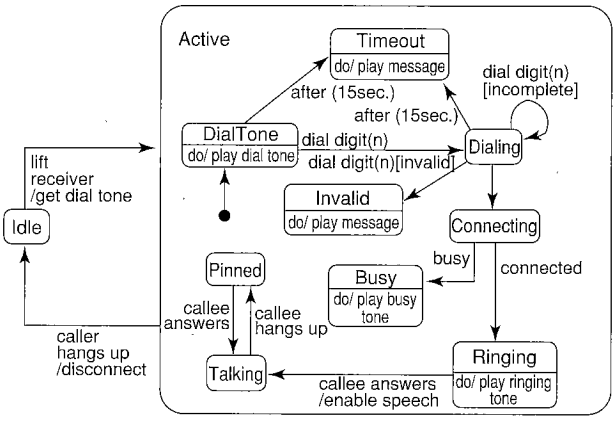

図-7　ステートチャート図の例

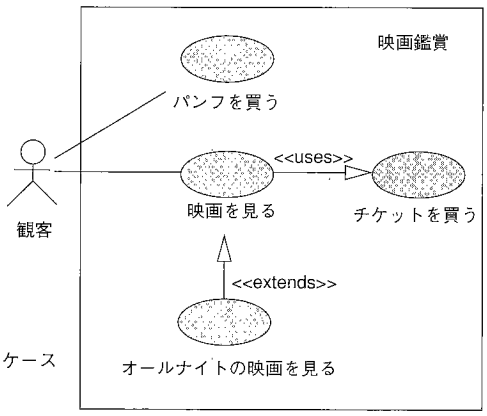

図-8　ユースケース図の例

じめ定義している．それらは，システムの静的な構造に焦点を当てた構造図と，システムの動的な変化や相互作用に焦点を当てた振る舞い図と，ソフトウェアという観点から実装時の構成や実行時の配置に焦点を当てた実装図の3つに大きく分類できる．

UMLダイアグラムの分類を作業目的に合わせて細分類し，構造表現，振る舞い表現，システム要求表現，アーキテクチャ表現という目的別の観点から各ダイアグラムの特徴を解説する．

■構造を表すダイアグラム

(1) クラス図（Class Diagram）

クラス図は対象領域やシステムの静的な構造を，クラスとそれらの間の関係として表現した図である．クラスは対象を表現するモデル内で第1級の管理対象としたいものを示す．そして，クラス間にはさまざまな関係（関連，依存関係，汎化，集約）を張ることでモデルの構造を表現する．

モデルを管理するために複数のクラスをグループ化してパッケージとすることができるが，このパッケージもクラス図の構成要素となる．

(2) オブジェクト図（Object Diagram）

オブジェクト図はインスタンスレベルの静的構造を表現する図である．クラス図のバリエーションと考えられ，記法もクラス図とほぼ同様である．クラス図がシステムのとりうる構造の制約を表現する一般的なモデルであるとすると，オブジェクト図はクラス図の1事例を表したものとなり，そのシステムのある実行時点におけるスナップショットを記述したものになる．また，オブジェクト図は，後述のコラボレーション図においてメッセージ情報を取り除いたものと考えることもできる．

オブジェクトはクラスシンボルのクラス名の代わりにインスタンス名にし，アンダーラインを引いて表現する．また関連の多重度は用いず，個別に個々のインスタンスとの関連リンクを張る．

■振る舞いを表すダイアグラム

振る舞いは相互作用図とステートチャート図，アクティビティ図で表現する．相互作用図はシーケンス図とコラボレーション図からなる．

(1) シーケンス図（Sequence Diagram）

シーケンス図は，相互作用図の1種であり，あるコラボレーションに参加するオブジェクトの間でやりとりされる個々のメッセージ送受信（相互作用と呼ぶ）の時間的順序を上から下に順に示す．メッセージ送信を時間順に1つずつトレースできるため，シナリオと対応させてコラボレーションの具体的な内容を示すのに便利であり，分析で多用される．

縦棒（オブジェクト生存線と呼ぶ）でオブジェクトの存在を示し，オブジェクト間のメッセージ送信を実線矢印とメッセージ名で表して上から順番に並べる．また分析では特に，シーケンス図の左にメッセージ送信と対応させてシナリオの各ステップをテキストで記述することで，ユースケースの実現例である1本のシナリオとシーケンス図の対応付けが容易になる．

(2) コラボレーション図（Collaboration Diagram）

コラボレーションとは，1つのユースケースや1つのメソッド等を実現するに必要な一連の相互作用のまとまりを指す．コラボレーション図も，相互作用図の1種であり，あるコラボレーションに参加するオブジェクト間で送受信されるメッセージを，それらのオブジェクト間の接続関係（コラボレーションのコンテキストと呼ぶ）とともに示す．参加オブジェクトは，オブジェクト名にアンダーラインの付いたシンボルで表現し，それらを関連リンクで結合して，各オブジェクトの参照・接続関係を示す．

各リンクを流れるメッセージはリンクの脇の矢印で表し，メッセージ送信の順番を示すシーケンス番号を各メッセージの頭に振る．

シーケンス図に比べると設計を意識したオブジェクト間の接続関係が詳細に表現できるが，時間順のトレースはやりづらくなる．したがって，設計で多用する．

(3) ステートチャート図（State Chart Diagram）

ステートチャート図は，クラス（場合によってはシステム全体や各ユースケース）のとりうるライフサイクルの状態遷移図としてHarelのステートチャート記法で表したものである．オブジェクトのとりうる状態の集合，状態間の遷移とそのトリガとなるイベント，状態や遷移にともなうアクションの記述によって，オブジェクトの振る舞いを記述する．ステートチャート図では，状態の入れ子とそれに伴う遷移とアクションの継承，および状態履歴という概念が追加され，従来の状態遷移図に比べ表現力と記述の簡潔さが向上した．

ステートチャート図は各クラスごとにその振る舞いを定義するのに必要だが，実際には，ある程度複雑なライフサイクルをもつクラスのみ定義すれば十分である．

ラウンドボックスで状態を，状態間の実線矢印で遷移を，遷移の脇にイベント名を置いて，その遷移を発火するトリガイベントを示す．状態遷移および状態そのものに付随した操作実行をアクションとして付加することができる．また，状態には，入状時アクション，退状時アクション，状態内で継続的に実行するアクティビティを定義できる．

状態は入れ子にでき，上位状態の性質を下位状態は継承する．また，並行状態を破線で区切って並置したり，並行状態との入出力に対応して複数の遷移間で同期をとるためのバーを導入できる．

■システム要求定義のためのダイアグラム

(1) ユースケース図（Use Case Diagram）

ユースケースとは，システムの利用のしかたをシステム外部の視点から記述したもので，システムの利用者のタイプ（すなわちアクタ）と併せて記述する．システムに対する要求を，ユースケースという単位でカプセル化し，ユースケースと分析モデル，設計モデルをトレースすることで，システム要求から分析，設計，実装，テストといった一連の作業を関連付けて管理できる．したがって，ユースケースは，要求フェイズのみならずシステム開発の全工程で利用できる．

アクタを人型シンボルで，1つのユースケースを実線の楕円形で表す．

ステレオタイプが，ユースケース間の関係の分類にも適用されており，3種類のユースケース間の関係が一般の関係のサブカテゴリとして定義され，それぞれ次のようなキーワードラベルで区別される．

- 《communicates》:アクタとユースケースとの相互作用を示す．
- 《uses》：ユースケースの内部で別のユースケースのインスタンスを呼び出して使用する関係を示す．サブルーチン呼出しのように，利用する側のユースケースから共通ユースケースに白抜きの三角矢印を引く．ただし，この関係に汎化と同じ記号を使うことに以前から批判があるが，UMLの次の版では改められる予定．
- 《extends》：一般的なユースケースとそのバリエーションや例外を与えるユースケースとの関係を示す．特殊なケースのユースケースから汎用のユースケースに汎化の白抜きの三角矢印を引く．

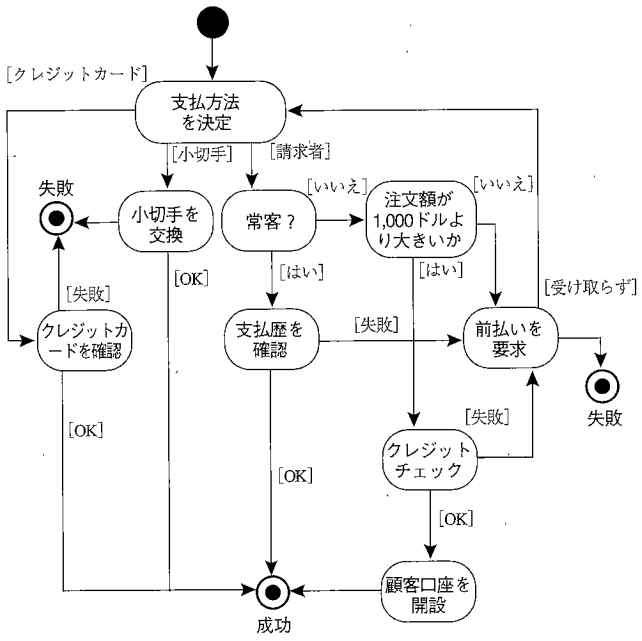

図-9 アクティビティ図の例

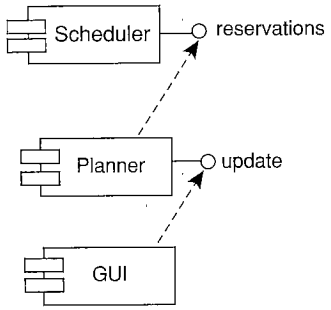

図-10 コンポーネント図の例

(2) アクティビティ図（Activity Diagram）

アクティビティ図は，Odellらのイベントスキーマ図とオブジェクトフロー図に影響を受けてUML1.0から新たに導入されたもので，ワークフローを表現するために用いる．アクティビティ図は，ひとまとまりの業務や処理を表すために関連する複数のアクティビティを時間的に順序だてて表現する．アクティビティは業務や処理を構成する1単位に相当し，状態の1種と見なされる．アクティビティの粒度は相対的である．人間のタスクにも，システムの1機能（つまり1ユースケース）にも，1メソッドの実行にも対応する．アクティビティ図を特定業務や組織のビジネスフローの記述に使うこともできれば，あるシステムの1つのユースケースに対応する処理フローの記述や，あるオブジェクトのもつ1メソッド内部のアルゴリズム記述に使うこともできる．

ラウンドボックスでアクティビティを表し，その中にアクティビティの名前を記し，アクティビティ間を矢印で

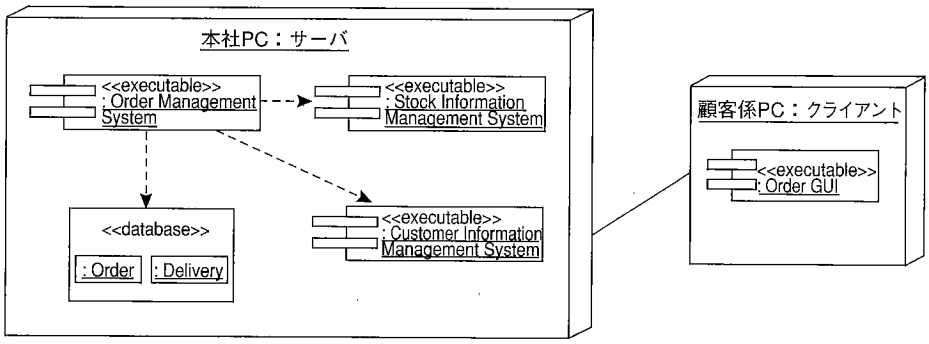

図-11　配置図の例

繋ぐことで作業の順序を表す．ある条件を満たさないと次のアクティビティに移行できない場合は，矢印にガード条件を付け，条件判断は特殊なアクティビティと見なすことができ，ひし形図形で表す．また太いバーによってそこに入力するすべての遷移の同期を表現し，そこから複数の遷移が出力される場合，それらの並行性を表す．

アクティビティ図は状態図の特殊形である．すべての状態にアクションが張り付き，前の状態におけるアクションの終了がトリガとなって状態遷移が起こるような特殊な状態図だと考えればよい．したがって，始状態，終状態には状態図と同じ記号が使われる．

アクティビティ図に「レーン」を設定することができ，これによって，各アクティビティの実行責任主体を明示できる．さらに各アクティビティ間での成果物つまりオブジェクトのやりとり（オブジェクトフローと呼ぶ）も明示できる．したがって，アクティビティとオブジェクトフローを使って，DFD（データフロー図）の代用が可能である．

また，アクティビティ図内で明示的にシグナル送信とシグナル受信を記述することができ，どちらもアクティビティ間の遷移に割り付けられた送信シンボル（片側に三角形が突き出た記号），受信シンボル（片側が三角にくぼんだ記号）で表す．シグナルの流れそのものは送信シンボルから受信シンボルへの破線矢印で表現する．

このようにアクティビティ図は，フローチャートやデータフロー図の流れをくんだ，どちらかというと手続き指向のダイアグラムである．適切な利用を心掛けないとオブジェクト指向との整合性で問題が出る可能性がある．典型的な利用例は，ビジネスやシステムの大局的なワークフローの記述，および複雑な手続き的アルゴリズムの記述あたりだろう．

■アーキテクチャのためのダイアグラム

(1) コンポーネント図（Component Diagram）

パッケージ概念と依存関係を使って，クラスファイルやコンポーネントからアプリケーションやサブシステム，バージョンやシステムがいかに構成されるかを表現したものがコンポーネント図である．

コンポーネントとは，コンパイルやリンクや実行の単位となるものを指す．したがって，ソースファイル，実行モジュール，Windows環境のDLL（Dynamic Link Library）もコンポーネントである．たとえば，C++の場合，複数のヘッダファイルとボディファイル，1つの主プログラムファイル間のインクルード関係に基づく依存関係のグラフがコンポーネント図を構成する．また，コンポーネントをグループ化したパッケージもコンポーネントと考えられる．

さらに，コンポーネント図では，インタフェース記号とそれへの破線矢印を用いて，どのコンポーネントがどのコンポーネントの実装するインタフェースを利用するかを明示することもできる．

(2) 配置図（Deployment Diagram）

ハードウェアにどうソフトウェアを配置するかという観点からシステムを表現したものが配置図である．そのために，ネットワーク内のサーバやクライアント，プロセッサやデバイスを抽象化したノードという概念を導入し，そこにプロセス，スレッドをどう割付け，さらにコンポーネントをどう配置するかを示す．プロセス間通信やネットワーク接続等をノード間の関連として表現する．

ノードは立方体シンボルで表し，ノードの性格を強調したい場合は，ステレオタイプ<<device>>等を使う．ノード間の関連に対しても，必要に応じて，ステレオタイプ<<IIOP>>，<<TCP/IP>>，<<RS-232C>>等を付加する．

配置図において，必要に応じて各ノード内に，オブジェクト図を表示することも，コンポーネント図を表示することもできる．

分散処理で問題になるのが，実際の物理的なオブジェクトの配置（deployment）と移動（migration）である．UMLでは「ノード」によって，オブジェクトの場所やネットワークのトポロジを抽象的に議論できる．計算リソースを保有する物理要素がノードと考えられる．

配置図の中でインタフェースを使うことにより，サーバオブジェクトの位置および提供するサービスを示すことができるが，クライアントオブジェクトも明示したければ，クライアントとサーバ間に依存関係の矢印を引く．また，オブジェクトやコンポーネントのノード間での移動（オブジェクトの動的再配置，いわゆるマイグレーション）やコピー（レプリケーション機能）を示したければ，配置図内で，旧オブジェクトと新オブジェクト間で《becomes》や《copies》といったラベル付き依存関係の矢印を引けばよい．

参考文献
1) Booch, G.: Object-Oriented Analysis and Design with Applications, 2nd Ed., Benjamin/Cummings (1994).
2) Jacobson, I. : Object-Oriented Software Engineering, Addison Wesley (1992). [西岡利博ほか (監訳): オブジェクト指向ソフトウェア工学, トッパン (1995)].
3) Fowler, M. et al.: UML Distilled, Addison-Weslay (1997). [羽生田栄一(監訳): UMLモデリングのエッセンス, アジソン・ウェスレイ(1998)].
4) OMG: http://www.omg.org
5) Rumbauch, J. et al.: Object-Oriented Modeling and Design, Prentice Hall (1991). [羽生田栄一 (監訳): オブジェクト指向方法論OMT, トッパン (1992)].

(平成10年12月11日受付)