オブジェクト指向分析設計

全体 outline

ここでは，オブジェクト指向分析から設計までの開発プロセスを概観し，次にそれぞれのプロセスで活用されている技術を解説する．

[開発プロセスの概観]

オブジェクト指向の分析プロセスは，大きく問題領域理解，要求分析，オブジェクト抽出に分けられる．問題領域理解では，開発するシステムを取り巻く社会環境やシステム環境，企業戦略などを理解し，問題領域の特性を分析する．要求分析では，理解する対象を絞り込んで，開発するシステムの振る舞いを定義する．この作業は，システムの境界を明確にするために重要な作業となる．また，オブジェクト抽出では，システムの振る舞いを実現するために必要なオブジェクトを抽出し，その構造，状態，オブジェクト間の相互作用を定義する．さらに，オブジェクトは抽象化されてクラスが定義され，システムのモデル化がほぼ完了する．

再利用性，拡張性，独立性を高めるためにグループ化したり，さらに抽象化を進めたり，あるいは，処理効率，拡張性，可読性の向上，システムの稼働環境への適応といった非機能的要求への対応は設計の作業となる．

[様々な方法論]

オブジェクト指向開発方法論は 1986 年のブーチによる Ada のための設計手法[{Booch}]に始まったと言える．その後，1990 年前半までに多くの方法論が提案された．これらの方法論は，大きくデータ駆動型方法論と責任駆動型方法論とに分けることができる．

1989 年から 1990 年代初頭に提案された方法論のうち，コード／ヨードン法[{Coad}]，シュレィア／メラー法[{Shlaer}]，OMT 法[{OMT}]はデータ駆動型方法論に分類できる．データ駆動型方法論は，最初に現実世界の事物や事象をオブジェクトとすることを重視する方法論であり，問題領域理解のような現実世界のモデル化にも適用しやすい．

これに対して，1989 年にカニングハム／ベックらが提案したクラス／責任／協力者(Class Responsibility Collaborator)法[{CRC}]，ワーフスブロックらによる責任駆動型設計法[{Wirfs89},{Wirfs90}]や 1992 年のヤコブソンによる Objectory 法[{Jacobson92}]，ルビン／

ゴールドバーグによるオブジェクト振る舞い分析(Object Behavior Analysis)法[{OBA}]は，責任駆動型方法論に分類できる．これらの方法論は，外部から観察されるシステムの振る舞いを定義することから始まり，そこから「××ができること」といった，システムが利用者に提供する責務を抽出し，関連の深い責務と責務を果たすために必要な情報とをカプセル化してオブジェクトを定義するという過程を経る．したがって，責任駆動型方法論は，要求分析以降のシステムのモデル化に適した方法論であると言うことができる．

設計では，システムの信頼性を高めるための手法が求められる．メイヤーは，オブジェクトの信頼性を高めるための設計手法として「契約に基づく設計(Design by Contract)[{Meyer}]」を提案した．契約に基づく設計は，オブジェクトが提供するサービスに，事前条件と事後条件という表明を取り入れ，オブジェクトの仕様をより形式的に記述する手法であり，メイヤーは Eiffel というオブジェクト指向プログラミング言語の中に表明を定義する機構を取り入れている．ヤコブソンが Objectory 法の中で提案したユースケースにも，事前条件と事後条件が導入されている．ユースケースの例を次の項で示すので，参考にされたい．

1998 年に提案されたドゥスーザ／ウィルスによるカタリシス（Catalysis）法[{Dsouza}]は，UML のオブジェクト制約言語 OCL(Object Constraint Language)を導入した方法論である．OCL を用いると，オブジェクトの構造や振る舞いの制約を記述することができる．たとえば，整数の定義を OCL で行うと次のようになる．

Integer
  global 0: Integer　……オブジェクトに共通な定数
  +(Integer) : Integer　….演算+の定義
  prev: Integer　…….演算 prev の定義（ひとつ前の整数を得る演算）
  next: Integer　…….演算 next の定義（次の整数を得る演算）
  inv　　…….オブジェクトの不変表明のラベル
    self.prev.next = self　…..ある整数に prev させ，next させた結果はもとの整数である
  & self + 0 = self　…….かつ，ある整数に 0 を+させた結果は，もとの整数である
  & self + x = self.next + x.prev　…..かつ，ある整数にｘを+させた結果は，ある整数に next させた結果と，x に prev させた結果を＋した結果と等しい．

OCL は，オブジェクトの仕様を形式的に書くための言語であり，例に示したように，オブジェクトの変わらない性質を示して，演算の制約を与えるだけでなく，属性値の制約なども記述することができる．このようにオブジェクトの制約を明記することによって，オブ

ジェクトの整合性を保ち，システム全体の信頼性を高めることができる．

以下に各プロセスで活用する技術を紹介する．

要求分析 require

[ユースケース]

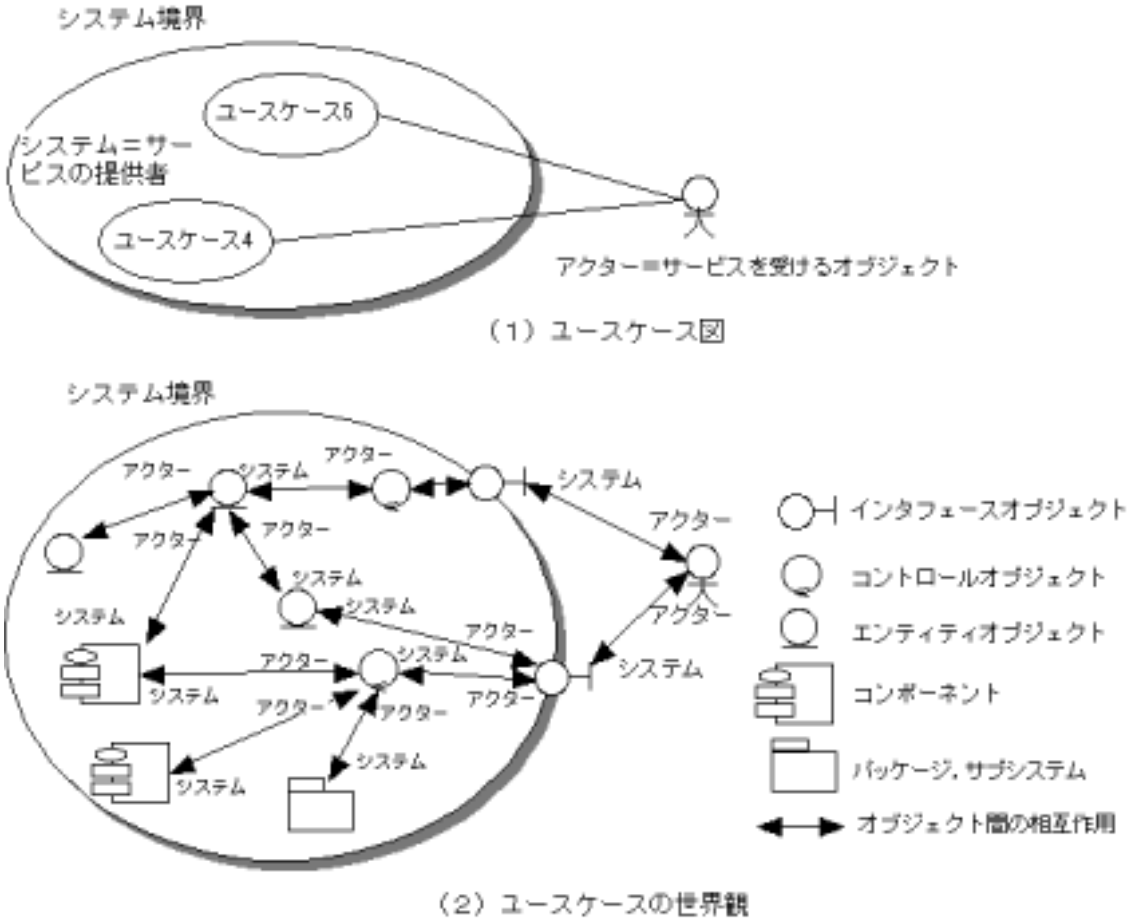

図 1　ユースケースの世界観

ユースケースとは，サービスを提供するシステムと，システムの外部にいて，システムからサービスを受ける利用者や他のシステムとの間の相互作用を時系列で記述した要求仕様の記述形式である[{Jacobson92}]．また，そのような形式で表現された要求仕様を指す．ユースケースでは，システムからサービスを受けるオブジェクトをアクターと呼ぶ．システ

ムに定義されるユースケース群の全体像と，それぞれのユースケースとアクターとの関係はユースケース図によって記述される．図１（１）にユースケース図の例を示した．ユースケースのもっとも基本的な構成は，名前，目的，アクター，事前条件，事後条件，基本系列，代替系列からなる．

事前条件とは，そのユースケースが起動されるために必要な条件であり，事後条件とは，ユースケースが起動され，基本系列に定義された相互作用が実行された後に成立する条件である．これらの項目は，「事前条件が満たされたもとでユースケースが起動されれば，必ず事後条件を満たした状態が成立するようにシステムは動作する」というユースケースの表明である．ユースケースの目的には，事前条件が満たされているシステムの状態を事後条件を満たす状態へ遷移させる意味を定義する．基本系列とは，事前条件が成立しているもとで起動され，事後条件を成立させるまでのアクターとシステムとの間の相互作用の記述である．基本系列の項目番号は，時系列で発生する事象の順番を表す．各項目で予想される例外事象がある場合は，別途，代替系列に記述することになっている．

ユースケースの定義を一般化すると，サービスを提供するオブジェクトとサービスを受けるオブジェクトとの間の相互作用を定義する目的で用いることができる．このとき，システム境界は，オブジェクトのカプセルの殻とみなされる．もともと，ヤコブソンが提案したユースケースは，開発するシステムだけにとどまらず，オブジェクト間の相互作用を記述する世界観を持っていた．図１（２）にユースケースの世界観を示した．

ユースケースには事前条件と事後条件が定義されているため，オブジェクトやコンポーネントの仕様を記述する目的に適用されるだけでなく，テストケースを生成する情報源にも活用されている．

ヤコブソンは，図１（２）の世界観のもとで，オブジェクトをインタフェースオブジェクト，コントロールオブジェクト，エンティティオブジェクトの三種類に分類し，オブジェクト抽出の指針も示している．インタフェースオブジェクトとはアクターとシステムとの相互作用の仲介をするオブジェクトである．ユーザインタフェースとなるオブジェクトはその代表例である．コントロールオブジェクトは，繰り返しや条件分岐などの制御を行って，システムの内部に格納されているオブジェクトと協調してインタフェースオブジェクトの責務遂行の支援を行う．エンティティオブジェクトは，問題領域に属するオブジェクトなど，インタフェースオブジェクトの機能に左右されて振る舞いを変えないオブジェクトである．

以下に自動販売機の代金投入に関する要求を用いて，ユースケースの例を示そう．

●ユースケース名：代金投入
●アクター：自動販売機に代金を投入する者
●目的：投入された金額で販売が可能な商品をアクターに知らせる
●事前条件：代金投入が可能であり，かつ，自動販売機が受け付けられる最大投入金額が与えられている．
●基本系列：
1. アクターは硬貨投入口から貨幣を投入する．
2. 自動販売機は，代金投入の受付を中断し，投入された貨幣の正当・不当判定を行う．
3. 自動販売機は正当な貨幣だけを受付け，残高表示器に表示していた額に，新たに投入された貨幣の金額を合計して表示する．
4. 自動販売機は，投入された額に応じて販売可能な商品の販売可能表示ランプを点灯する．
5. 自動販売機は，代金投入の受付を再開する．
●事後条件：販売可能な商品の販売可能表示ランプが点灯しており，代金投入が可能である．
●代替系列：
・ 基本系列 2 において，貨幣が投入されたことによって，それまでに投入された代金の合計金額が最大投入金額を超えた場合，自動販売機は，新たに投入された貨幣を釣銭口から返却する．
・ 基本系列 2 において，投入されたものを正当な貨幣と認識できなかった場合，自動販売機はそれを釣銭口から返却する．
・ 基本系列 3 において，自動販売機が投入された貨幣によって商品が選択された後，釣り銭が払えない可能性のある場合，自動販売機は今投入された貨幣を釣銭口から返却し，釣り銭切れランプを点灯する．
・ このユースケースを実行中に，緊急停止信号が発せられた場合は緊急停止ユースケースを実行する．実行後，この自動販売機は停止状態となる．

例に示したように，ユースケースの各項目は自然言語で記述されるため，開発者とユーザとの相互理解も円滑に進み，ユーザの要求を的確にシステムへ反映することができると言われている．また，最近では，ユースケースはオブジェクト指向システムに限らず，要求記述形式として開発関係者に浸透し始めている．

問題のモデル化 modeling

[ユースケースから抽出するオブジェクト]

先に示したユースケースの記述から，責任駆動型手法に則って，オブジェクトを抽出してみよう．

自動販売機には，貨幣の正当性を判定する責務がある．また，そこで求められた貨幣の金額は，それまでに投入された金額に足し込まれ，新しい合計金額が求められなければならない．ここから，投入された貨幣の正当性を判定し，金額を求めるインタフェースオブジェクトや，投入された貨幣の合計金額を記憶する責務を持つエンティティオブジェクトが必要であることがわかる．オブジェクトに記憶されている合計金額は，残高表示器というインタフェースオブジェクトを介してアクターに伝えられる．同様にして役割を抽出すると，図 2 に示すオブジェクトを抽出することができる．図は，UML のパッケージの表記をオブジェクトの代わりに用い，メッセージの流れを矢印の上に表した．

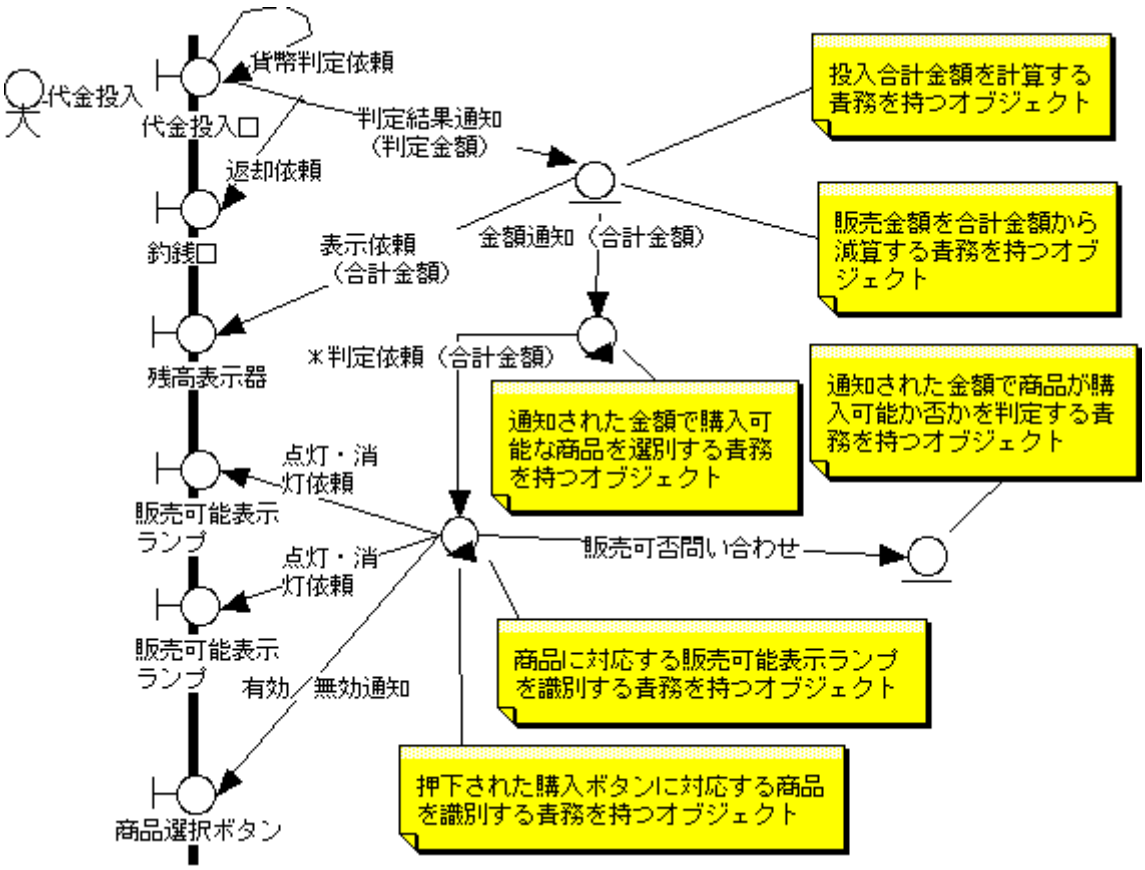

図 2　抽出したオブジェクトの例

[状態遷移と相互作用の分析]

オブジェクトの状態の変化やオブジェクト間の相互作用をブジェクトの動的な振る舞いと

言う．このようなモデルを表現するためには，相互作用図や状態図が用いられる．

UML では，オブジェクトの状態の遷移を表記する図として，ステートチャート図が提供されている．また，相互作用図としてコラボレーション図とシーケンス図が提供されている．

（1） ステートチャート図の適用

ステートチャート図は，オブジェクトがとり得る状態を列挙し，オブジェクトがそれらの状態の間を，いつ，どのような条件が満たされたときに，そして，どのような事象が発生したときに遷移するのかを分析する視点を提供する．

図 3 に特定の商品の販売状況に関する自動販売機のステートチャート図を示した．黒丸から商品販売のライフサイクルが始まり，二重丸によって完結する．四角形はオブジェクトの状態を表し，矢印は状態の遷移を，また矢印上の文字は遷移を起こさせる事象を，鍵括弧は遷移が成立する条件をそれぞれ表す．また，ステートチャート図では状態の入れ子構造も定義できる．図 3 では，自動販売機は販売を行っている間，販売可能表示ランプは点灯，消灯，点滅を繰り返すが，稼働停止はランプがどのような状態であっても発生しうることを表している．

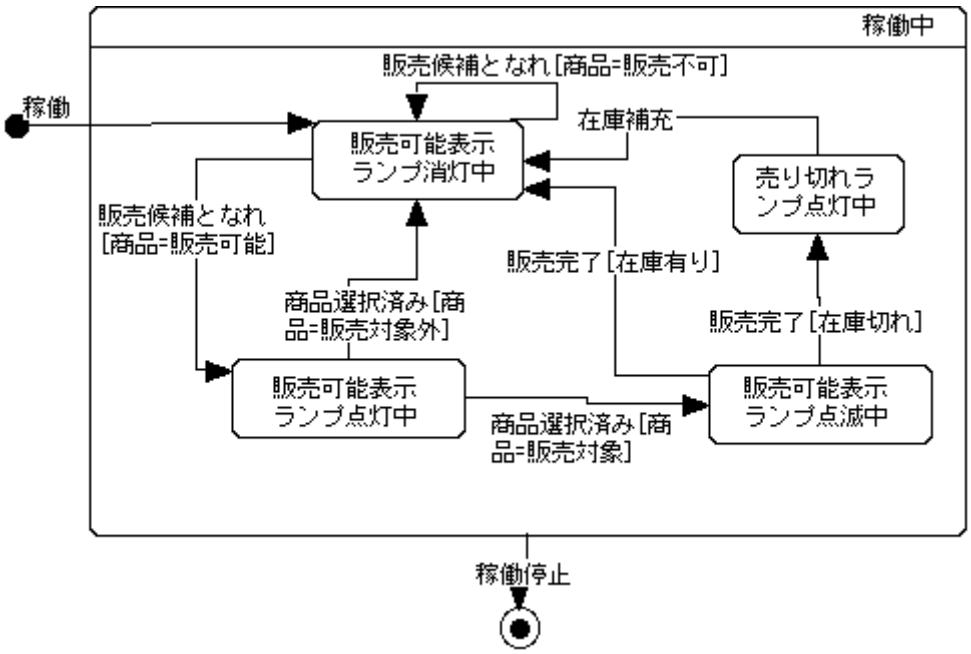

図 3　販売状況のステートチャート図

しかし，ステートチャート図は，どの事象がどのオブジェクトから通知されるかを分析す

るのには適さない．このような情報は相互作用図を用いて分析する．

（２） 相互作用図の適用

図 4 に示したシーケンス図とコラボレーション図は，代金が投入されることで販売可能な商品の販売可能表示ランプが点灯するまでのオブジェクト間の相互作用の例を示した図である．シーケンス図の横軸にはオブジェクトが並び，それぞれのオブジェクトからオブジェクトの生存を表すライフラインが縦に伸びる．縦軸はメッセージが流れる時間軸を表し，オブジェクト間の相互作用はライフラインから出てライフラインに向かう矢印で表される．送信されるメッセージは矢印の上に記述される．コラボレーション図のオブジェクトは，次に紹介するクラス図のように配置され，メッセージは，オブジェクト間の関連を介して渡される．コラボレーション図では，メッセージの順番はメッセージに付加した番号で表され，メッセージの番号の枝番でメッセージ送信の深さを示す．

これらの相互作用図は，それぞれ分析する目的や内容によって使い分けることができる．たとえば，コラボレーション図を用いるとオブジェクト間の相互作用の入れ子構造を深く参照してシステム全体のメッセージの流れを把握することができる．シーケンス図は多数のオブジェクト間の相互作用を参照するというよりは，特定のオブジェクトから発信されるメッセージのタイミングを参照するのに適している．このような特徴から，コラボレーション図はオブジェクト間の複雑度を評価するために，また，シーケンス図はメソッドの複雑度を評価する目的で使うことができる．

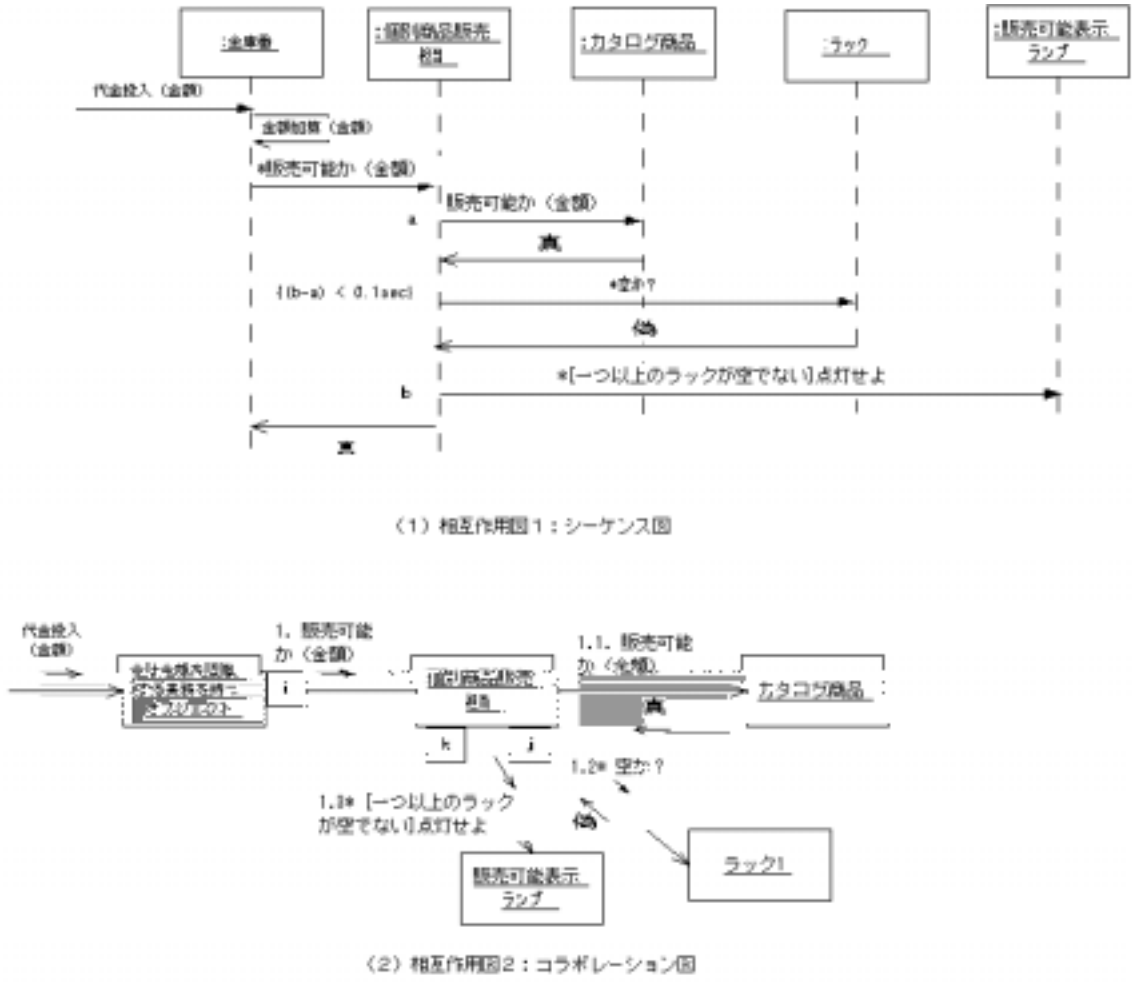

図 4　販売可能表示ランプ点灯に関する相互作用図

[静的な構造の分析]

オブジェクトの静的な構造は，クラス図を用いて分析することができる．クラス図は，オブジェクト指向プログラミング言語のクラスの定義によく対応する図である．

図 5 に商品販売に関するクラスだけを抜き出したクラス図を示した．「個別商品販売担当」は特定の商品を販売する責務を持ったオブジェクトであり，複数の「販売可能表示ランプ」，商品が格納されている複数の「ラック」，販売する商品のカタログ情報を保持する「カタログ商品」と関連を持つ．ステートチャート図に定義されていた事象は「個別商品販売担当」が受け取るメッセージとしてクラス図に定義される．「販売可能表示ランプ」は消灯／点灯／点滅の基本的な動作だけを行うが，これらの状態遷移は，シーケンス図に記述されているように，「個別商品販売担当」から指示される．そのため，「個別商品販売担当」は，「販売可能表示ランプ」と関連を持っていなければならないことがわかる．

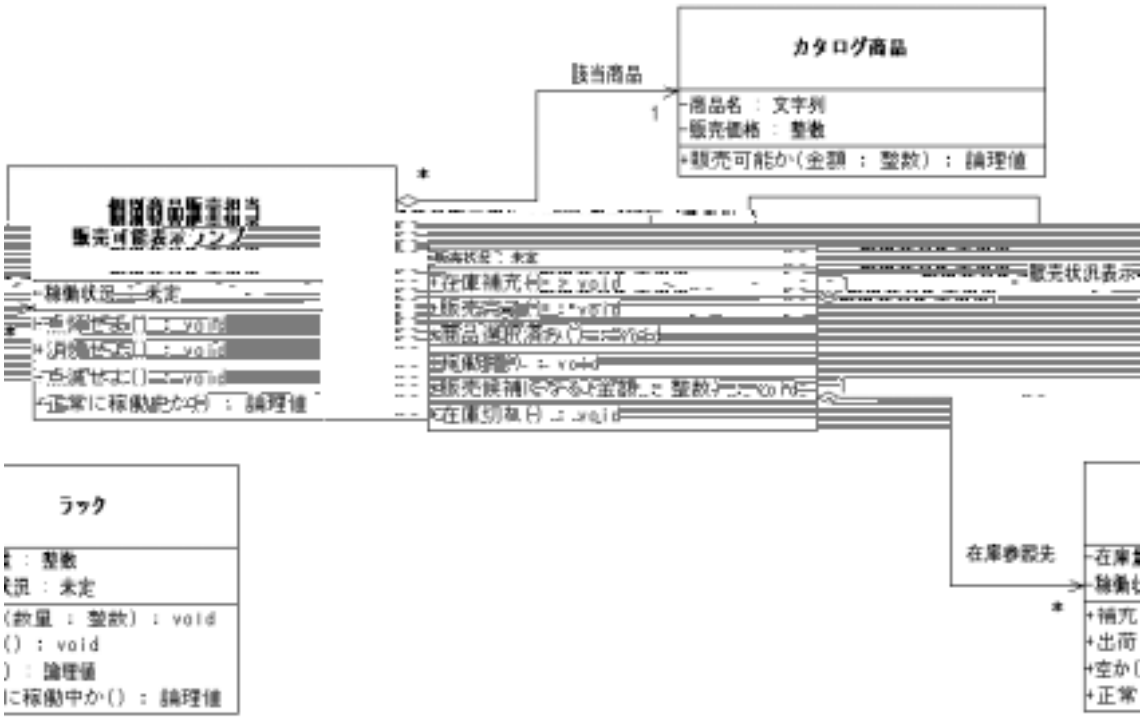

図 5　自動販売機のクラス図（商品販売に関する部分のみ抜粋）

このように，UML の個々の図式は，それぞれ異なる情報を分析する目的で使用される．言い換えると，これらの図式を用いることによって，多面的に世界を分析できるということなのである．

設計 design

[分析から設計へ]

設計では，実装に向けた様々な側面を考慮したモデルを構築しなければならない．また，分析で構築されたモデルが再利用性や拡張性を阻害しないか否かを評価し，より適切なモデルへ変換していく必要もある．

図 6 に不適切な役割分担がなされているシーケンス図を示した．この図と図 4 のシーケンス図を比較してみよう．図 6 では,「金庫番＝合計金額を認識する責務を持つオブジェクト」が，合計金額の計算から販売可能表示ランプ点灯までのすべての意思決定を行っている．複数の「ラック」や「販売可能表示ランプ」を「金庫番」が参照し，これらのオブジェクトを集約している「個別商品販売担当」の情報隠蔽を破壊している．

情報隠蔽を破壊しているモデルは，再利用性や拡張性を低下させる．たとえば，図 6 で「個別商品販売担当」より右に書かれているオブジェクトをまとめてひとつのコンポーネントとして再利用の単位にできないだけでなく，これらのオブジェクトの変更は，「金庫番」へ及ぶ可能性が高いため，保守しにくい構造となっている．

情報隠蔽を破壊しているモデルをシーケンス図で見ると，しばしば一本のオブジェクトのライフラインから多数のメッセージが様々なオブジェクトに発信されるという形で現れることが多い．

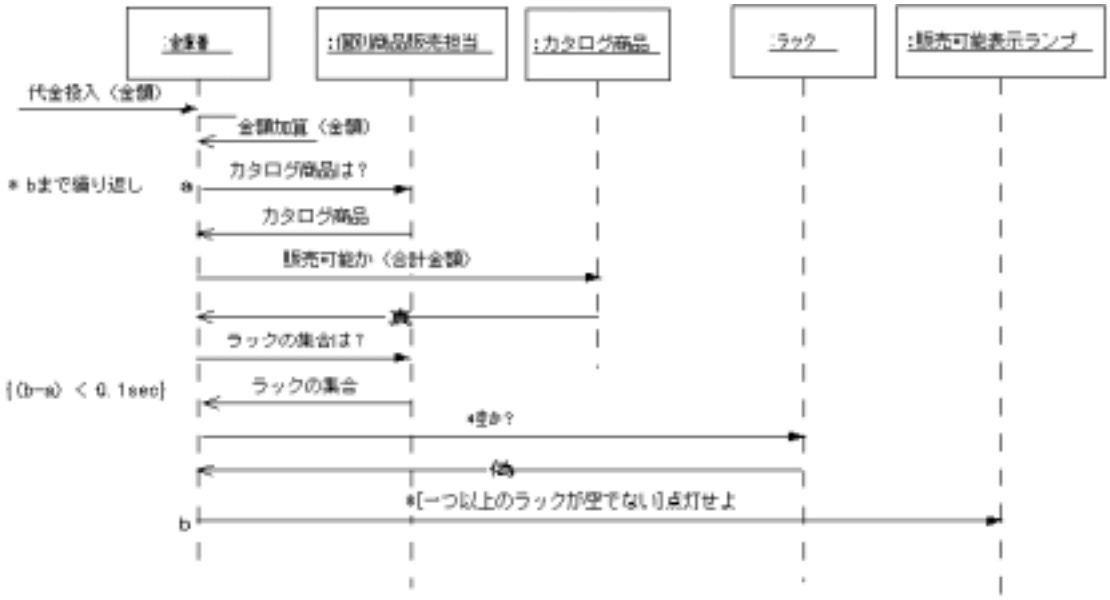

図 6　販売可能表示ランプ点灯に関する相互作用図

設計では，分析で定義された問題領域のオブジェクトを詳細化する以外に，利用者インタフェース層のオブジェクト，データ管理層のオブジェクト，タスク管理層のオブジェクトを設計する．Web のアプリケーションでは，タスク管理層にセッションの区切りや，データベースの更新のタイミングを設計する．

図 7 に分析から設計に至るモデル間の関係を示す．

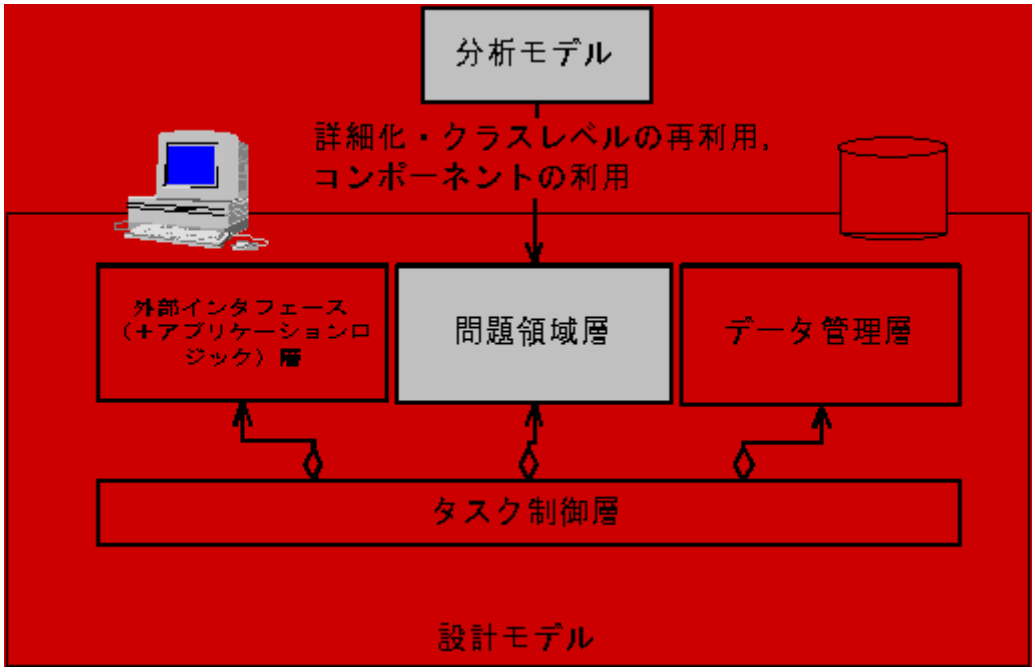

図 7　分析から設計へ．分析モデルの設計

[クラス主導型設計からフレームワーク主導型設計へ]

オブジェクト指向設計手法は，クラス主導型設計からフレームワーク主導型設計，そしてコンポーネントに基づく設計へと変遷してきた．それぞれの設計の特徴を概観してみよう．

（1）クラス主導型設計

クラス主導型設計とは，システムの枠組みをアプリケーションごとに構築し，提供されたライブラリの中から再利用可能なクラスを選択して，組み込む設計である（図 8（1））この開発形態では，アプリケーション毎に枠組みが定義されるため，類似のシステムでも，その枠組みが異なるということもあり得た．これは再利用の阻害要因であると言われている[{Garlan}]．クラス主導型において，再利用の実績があったクラスは，利用者インタフェースや配列といった汎用的なクラスであった．

（2）フレームワーク主導型設計

再利用性をさらに向上させるために導入されたのがフレームワーク主導型設計である．この設計では，類似のシステムでは，クラスを個別に再利用するだけでなく，フレームワークというシステムの基本構造を再利用する．（図 8（2））．フレームワークは，クラス間の協調動作が定義されている抽象クラスによって構成されている．開発者は，抽象クラスに定義されている協調動作を利用して，具象クラスを定義する．具象クラスにはアプリケーション固有の具体的な動作を定義する．

図 8（２）は，フレームワーク主導型設計とライブラリクラスの再利用を組み合わせた例である．アプリケーションシステム毎に開発する部分が少なくなるため，生産性が（１）よりも高くなることは容易に想像できよう．しかし，この構造で再利用したクラスが進化した場合，フレームワークを取り囲むアダプタ部分を変更しなければならない可能性もある．システムをより頑健にするためには，再利用する部品の進化から，システムを保護する機構が必要である．

再利用 recycling

（３） コンポーネントに基づく設計

再利用部品の進化に対してシステムを頑健にするために，フレームワークと再利用部品とを接続するアダプタを階層化させ，より部品に近い部分とフレームワークと近い部分とに分ける設計法が提案された．部品に近い部分を狭義のアダプタと呼ぶこともある．

この設計は，コンポーネントに基づく設計と呼ばれ，再利用部品のより高い独立性を目指したものである．最近の著しい技術進化や企業戦略上の迅速な要求変更への対応の必要性から，必須の設計技術となりつつある．システム本体と置換可能な再利用部品であるコンポーネントは，アプリケーション毎に開発される狭義のアダプタを介して結合される．そのため，両者は独立に進化可能である．図 8（３）にコンポーネントに基づく設計の形態を示した．

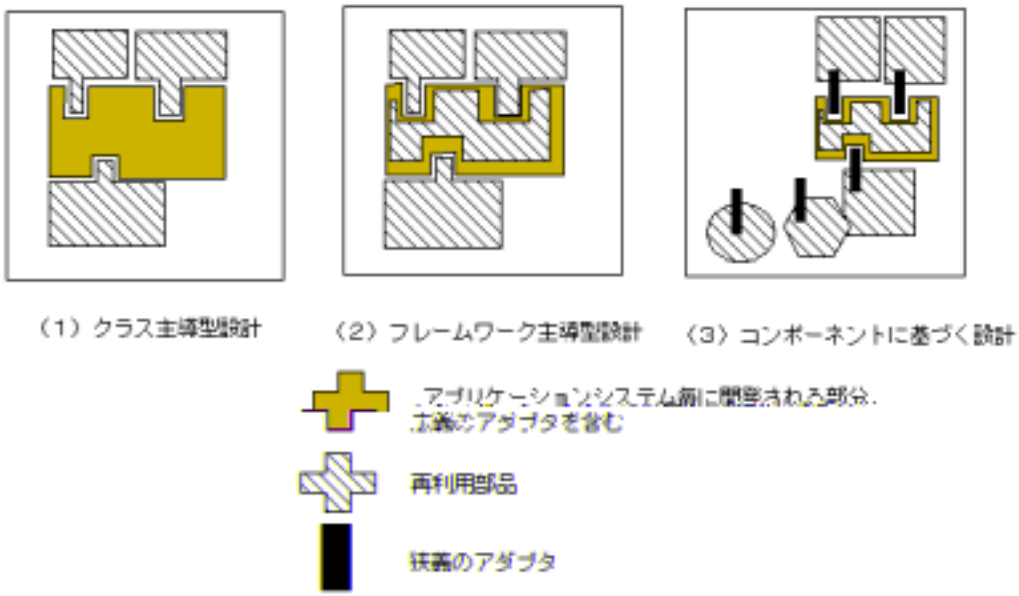

図 8 クラス，フレームワーク，コンポーネントの設計の違い

[まとめ]

オブジェクト指向分析設計方法論は，コンポーネントに基づく開発が時代の要請として必須の開発手法となるにつれて，技術者に求められる重要な知識となりつつある．再利用性の向上はソフトウェア工学の永遠の課題とも言われているが，構造化分析設計方法論からオブジェクト指向へ，そしてオブジェクト指向分析設計方法論の成熟と実用化の歴史の中で，再利用の形態はクラス主導型からコンポーネント主導型へと進化してきた．これらの再利用は成果物の再利用であるが，一方では，パターンというノウハウの再利用を進める活動も行われている．さらに，分散環境上でのシステム構築では，再利用部品の仕様からも独立な動的なシステム構成も必要となってくるだろう．今後も再利用性を向上させる努力とともに，開発方法論の検討は続けられていく．

（参考文献）

{Booch}Booch, G. : ``Object-Oriented Development,'' IEEE Transaction on Software Engineering, Vol.12, No. 2, pp.211--221 (1986).

{Coad}Coad, P. and Yourdon, E.: Object-Oriented Analysis, Prentice Hall, second edition, 1991 (羽生田栄一監訳, オブジェクト指向分析(OVA), トッパン, 1993.)

{Dsouza}D'Souza, D. F. and Wills, A. C. : Objects, Components, and Frameworks with UML, The Catalysis Approach, Addison-Wesley, 1998.

{Jacobson92}Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G. : Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, 1992. (西岡利博他監訳, オブジェクト指向ソフトウェア工学 OOSE, トッパン, 1995.)

{CRC}Beck, K. and Cunningham, W. : ``A Laboratory for Teaching Object-Oriented Thinking,'' Proc. of OOPSLA'89, ACM, pp. 1--6 (1989).

{Garlan}Garlan, D.m Allen, R. and Ockerbloom, J. : ``Architectural Mismatch: Why Reuse Is So Hard,'' IEEE Software, Nov. 1995, pp.17--26.

{Meyer}Meyer, B. : Object-Oriented Software Construction, second edition, Prentice Hall, 1997. (First Edition: 二木厚吉監訳, オブジェクト指向入門, アスキー出版, 1990.)

{OBA}Rubin, K. and Goldberg, A., ``Object Behavior Analysis,'' Communications of the ACM, Vol.35, No.9, pp. 48-62 (1992).

{OMT}Rumbaugh,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.)

{Shlaer}Shlaer, S. and Mellor, S.: Object-Oriented Systems Analysis, Prentice Hall, 1988. (本位田真一他訳, オブジェクト指向システム分析, 近代科学社, 1990.)

{UML}UML 技術情報：http://www.rational.co.jp/, http://www.rational.com/

{Wirfs89}Wirfs-Brock, R. and Wilkerson, B.: ``Object-Oriented Design: A Responsibility-Driven Approach,'' Proceedings of OOPSLA, ACM, pp. 71-75, 1989.

{Wirfs90}Wirfs-Brock, R., Wilkerson, B. and Wiener, L. : Designing Object-Oriented Software, Prentice Hall, 1990.