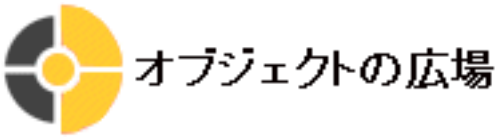


［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

ＩＮＤＥＸ

written by

（株）オージス総研
オブジェクト第一事業部
開発技術コンサルティング室
羽生田栄一

HOME TOP

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

1．オブジェクト指向って何？

オブジェクト指向とはObject-Orientedの訳で略してOO(オーオー)といいます。モノ中心にシステムを捉える、対象に則して設計する、という意味が込められた用語だといえるでしょう。オブジェクト指向プログラミングそのものは研究テーマとしては終息しつつありますが、オブジェクト指向モデリング技術やデザインパターン等のオブジェクト概念とソフトウェア工学の交差する部分には非常に広大な研究領域が広がっています。ぜひ、この分野に入ってこられる若い皆さんには、原理的な視点と役に立つ視点の交差する部分で研究者と現場の開発者とが協力しながら新しいアイデアを実践に結び付けていけるような活動を期待します。

私たちは日常生活をものやひとに囲まれてそれらとのやりとりを通して生きています。そこで起こる悩みや問題も基本的には、それらものを使い慣れた道具としさまざまなひととパートナーとしてお互い協力することで解決していくわけです。そのとき私たちは、こういう目的にはこの道具が使いやすい、この仕事にはあのパートナーと組むとやりやすい、と具体的な固有名を出して仕事の組み立てを頭の中でイメージしています。このような具体的な問題解決の要素となるものやひとのことを抽象化してオブジェクトと呼びます。

その際、ものやひとをその構造や機能や能力に応じて階層的に分類し、いつでも必要に応じて使い分けられるように概念整理しておくと便利です(ひとを使い分けるという言い方は倣岸ですが、いまのオブジェクト指向では基本的にものとひとの区別はしていません。その線で研究を進めていけば、エージェント指向に新しい展望が開けるかもしれません)。さまざまなオブジェクトを構造や機能によって類別し型として概念化したものをクラスと呼びます。クラスはその類似性と差異性にもとづいて適切な階層に分類されることになります。この階層のことをクラス継承階層と呼びます。

また仕事全体を進めていく上で、ものやひとどうしでお互いに「ひと仕事」を依頼しあいます。この仕事の依頼のことをメッセージと呼んでいます。依頼メッセージはできるだけ仕事仲間の間で共通化しておいた方が何か便利です。新人が入ってきても同じメッセージで仕事を依頼できれば、その業界で仕事の進め方を共通化したり要員が足りないときに新しいメンバーに来てもらっても仕事を遅滞なくスムーズに進めることができるようになります。同じ意味の仕事には同じ依頼メッセージを割り振るという命名規則のことをポリモーフィズムと呼んでいます。

以上のような考え方で対象を捉える仕方をオブジェクト指向モデリングといいます。それをソフトウェアの作り方の作法として全面的に採用したものがオブジェクト指向ソフトウェアであり、それを実際の計算機上でも採用したものがオブジェクト指向コンピューティングです。

このように、オブジェクト指向とはひとことでいうと、私たちの日常的な世界観にしたがって、対象をもの(概念のような抽象的な実体も含む)やひとの集合として認

識し、それらの相互作用でさまざまな現象が説明できる、だからソフトウェアの構成法もそれに従って行うべきだという考え方です。非常に素朴で素直でそれでいて強力な「ソフトウェア・アニミズム」とでも呼ぶべき世界観です。

Index Next

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

２．オブジェクト指向コンセプト早わかり

オブジェクト指向は1967年にSimulaというプログラミング言語に対して導入されたのが最初ですが、徐々にユーザーインターフェース設計、OSの構成原理、データベース設計やシステム設計一般、さらには要求分析といったシステム開発の上流工程に対しても適用されるようになってきました。このことは、オブジェクト指向がある工程や分野の特性を反映しているというよりも、問題領域やシステムといった複雑な対象に人間が対処するためにごく自然な一般的視点であり技法である、ということがいえると思います。

状態機械としてのオブジェクト

オブジェクト指向の基本は、内部状態をもつオブジェクトの集団を用いて、それらオブジェクトのイベントのやり取りによって各オブジェクトの内部状態を更新していき、その副作用としてアクションを行うという計算モデルです。いってみれば、イベントをお互いに授受する有限状態機械の集合としてシステムを構成するとき、その各状態機械のことをオブジェクトと呼ぶわけです。

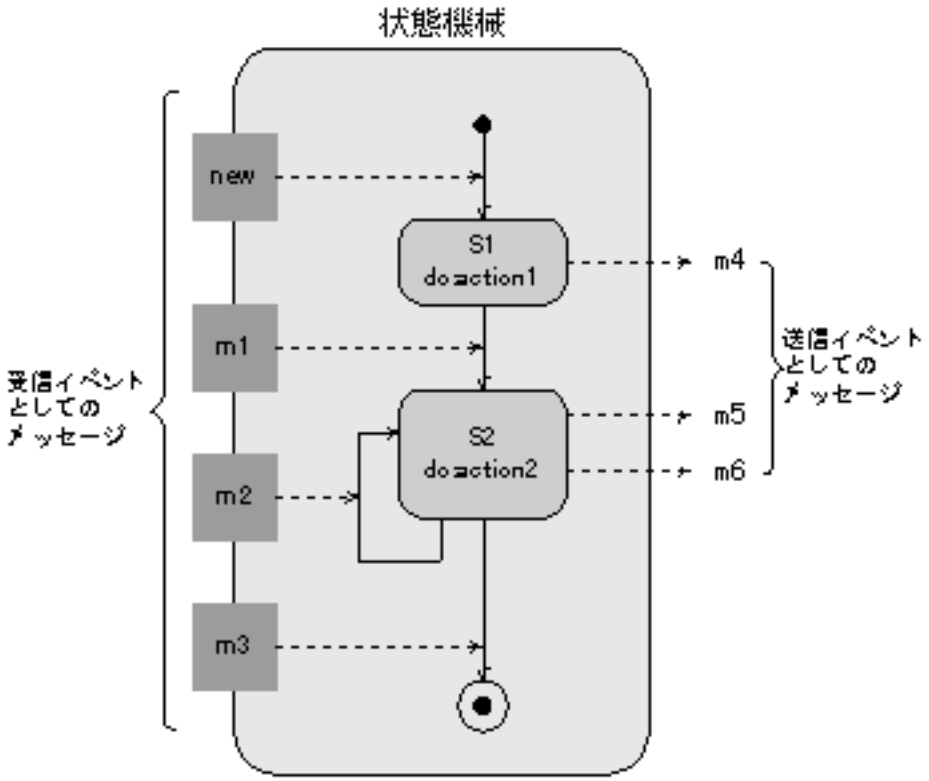


図1：イベントを送受信する状態機械としてのオブジェクト

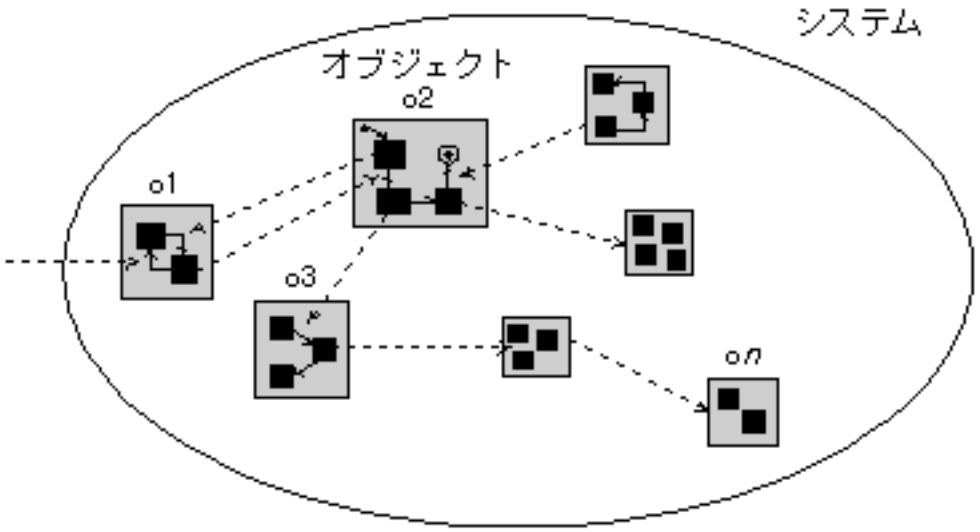


図2：イベントを送受信するオブジェクトの集合としてのシステム

ですからオブジェクトとは、メモリと外部刺激としてのイベントに反応してどんなアクションを実行するかという規則(メソッドといいます)をもった小人のような存在だとイメージすることができます。メッセージに反応して起動されたメソッドの中でさらに他のオブジェクトにメッセージを送信できます。

インターフェースと情報隠蔽

ソフトウェア工学的に見たときのオブジェクトの特徴は、メッセージ送信という間接的な手段を通してしか内部状態を参照・変更できない情報隠蔽です。受信可能なメッセージのシグニチャ(メッセージ名と各引数および戻り値の型)の集合をそのオブジェクトのもつインターフェースといいます。インターフェースを通してのみオブジェクトの提供する機能を利用できます。逆にインターフェースさえ変えなければ、その状態機械の実現方式はいかようにもできるわけです。この性質を利用して、オブジェクト指向では、他のオブジェクトに影響を与えずにあるオブジェクトの内部のデータ構造やアルゴリズムを変更することが容易にできます。

オブジェクトアイデンティティ

オブジェクトは、お互いのことが識別できなければメッセージを相手に送信することもできないわけですから、オブジェクトは、関数なんかとは違ってもしまったく同じ性質をもっていたとしても1個1個区別できる必要があります。ある人が同じ製品を2回買ったら代金は製品2個分払わなければならないのです。買った製品は1個2個と区別して数えられる必要があります。この性質をオブジェクトはアイデンティティをもつといいます。オブジェクトアイデンティティは、ソフトウェア中ではオブジェクトIDとして自動管理されます。固有名が日常世界では明示的に使われるのに対し、オブジェクトアイデンティティはあくまで同一か否かという「自己同一性」の確認に利用されるだけで、値として意識されることはありません。また、分散環境で複数のメモリ空間を移動するオブジェクトが存在する場合のアイデンティティの管理には工夫が必要になります。

インスタンスとクラス

オブジェクトはその対象を構成する1個1個の具体的な構成要素ですが、よくよく観察すると類似したオブジェクトが複数存在しているのが普通ですし、将来似たオブ

ジェクトを追加する必要が出てくるやもしれません。そこで、個々のオブジェクトではなく、それらの構造や機能の共通性を抽出し型として概念化したもの、一種のテンプレートのことをクラスといいます。

クラスは、共通の属性のセットとそれに関与するメソッドのセットを記述した型です。属性とはオブジェクトの内部状態を特徴づけるパラメータです。それに対し、各属性に具体的な値をデータとして与えたものをインスタンスと呼びます。オブジェクトは何らかのクラスのインスタンスとして生成されます。インスタンスは属性値を保持していますが、メソッドのセットは持たず、自分の属するクラスを参照しているのです。

また、オブジェクト指向でもクラス概念がない、プロトタイプにもとづく言語モデルもあります。新たなオブジェクトはクラスから生成するのでなく、原型オブジェクト(プロトタイプ)からクローニングによってコピーを作り出すのです。そのためあらかじめのクラス定義に縛られることなく実行時にオブジェクトのダイナミックな振る舞いを実現するのに適しています。

汎化と継承

複数の構造の似たオブジェクトをまとめて型として扱う技術がクラスでしたが、型の種類も多くなるとそのままでは管理が大変です。そこで、今度はクラスどうしを型としての類似・差異によって分類してしまおうという発想がでてきます。各クラスの型は属性セットとメソッドセットによって特徴づけれた構造と振る舞いを表現しているわけですから、それらの共通点をスーパークラスとして抽出(汎化)し、差分だけをサブクラスとして別に(特化)してやれば、クラスの分類階層ができあがります。サブクラスに属するオブジェクトはスーパークラスに属すると見なすこともできるという意味で、これをis-a関係と考えることができます。またソフトウェア・モジュールの観点からは、サブクラスからはスーパークラスで既に定義された属性やメソッドを流用できるという意味で、この関係を継承と呼びます。サブクラスではスーパークラスに追加して必要なデータ構造や再定義したい振る舞いのみを記述すればよいわけです。異なる複数のスーパークラスがとれる場合には多重継承と呼ばれます。

クラス分類を行うことで、問題領域やシステムに含まれるオブジェクトを体系的に管理できるようになり、類似性にもとづく理解を促進し、さらにはソフトウェア・モジュールとしての再利用性を高めます。もともと継承は、同一プログラム内でのデータ構造とプログラムコードの流用を目的として考案されたものです。したがって、単にプログラムの流用率を最大化するという近視眼的な発想にもとづけば必ずしもis-a関係にならないクラス間の継承でもたまたま使えるデータ構造やメソッドがあればOKということになってしまいます。しかし、システムやプログラムの複雑さに対処するという大局的な観点からはこのようなアドホックな継承は望ましくありません。is-a関係にもとづくクラス分類という観点から継承を利用すべきです。その際の指針は、外部観測できる振る舞いにおいて下位クラスのインスタンスが上位クラスのインスタンスとみなすことができるかというLiskovの置換原理です。特に多重継承にもとづく設計は振る舞いの正確な予測がむずかしくなりがちで、後で触れるデザインパターンの観点からも継承の代わりにオブジェクト間でのメッセージ転送を重視した委譲にもとづく設計が推奨されます。

ポリモーフィズム

オブジェクトどうしの相互作用はメッセージの送受信にもとづいています。メッセージの選び方で相手にどんな仕事をしてもらうかが決まるわけです。しかし、実際には、メッセージを具体的なアクションに対応させる最終的決定権をもっているのは受信側のオブジェクトです。同じmove()というメッセージであっても、その実際上の効果は受信側のオブジェクトのもつメソッドの内容によって異なります。航空機オブジェクトであれば飛行するでしょうし、イルカオブジェクトであれば泳ぐことでしょう。メッセージを媒介することによって、送信側の意図と受信側のアクション(メソッド起動)の対応の自由度が得られるわけです。これをメッセージのポリモーフィズムといいます。従来の手続き呼び出しであれば、イルカ用の移動手続き呼び出しというように送信側で詳細を決めなければならなかったわけです。

ポリモーフィズムとは、先に情報隠蔽のところで説明した外部インターフェースと内部実装の対応を受信オブジェクト側に任せるための機構だと考えることができます。これによって、新たなクラスが追加されても反応できるメッセージさえ共通に定義しておけば、それを呼び出して使うクライアント側のプログラムはいっさい影響を受けずに済むことになります。

集約と複合オブジェクト

オブジェクトは通常複数で協調して働きますが、その協調の様子が非常に密接である場合、それらをまとめて１つのオブジェクトと見なす方が便利な場合があります。そうした名目で作り出したオブジェクトを集約といったり複合オブジェクトといったりします。ですから複合オブジェクトとその要素オブジェクトとの間には全体-部分の関係が成り立ちます。デザインパターンでもCompositeとして再帰的な複合オブジェクトの合成パターンが定式化されています。また、継承の替わりに委譲によって、機能をダイナミックに追加したり交換したりするのも、この複合オブジェクトの構成原理の応用です。

全体オブジェクトのとる属性は部分オブジェクトにも継承されるのが普通です。たとえば、自動車オブジェクトが速度100km/sであれば、その乗客の移動速度も100km/sです。さらに部分が全体に存在依存する場合をコンポジット集約と呼びます。部品が他の全体オブジェクトに共有されることはなく、特定の全体オブジェクトに固有の存在として一体化されていることを示します。集約のセマンティクスにはさまざまなものがあり、それを精密に定義しコンポーネントウェア等オブジェクト部品の組み立て技術に応用するのは今後の研究課題です。

Prev. Index Next

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

3．オブジェクト指向とモデリング技法

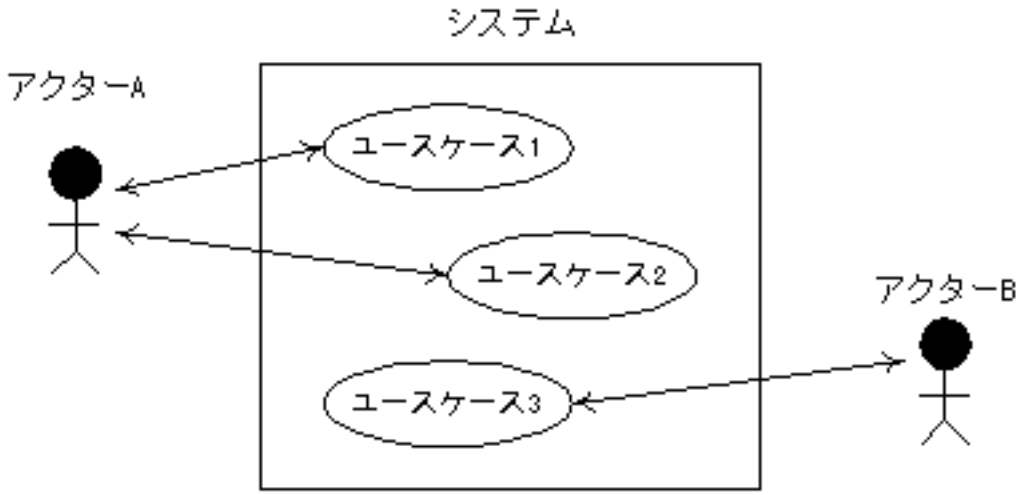
オブジェクト指向は、内部状態をもつオブジェクトの集団を用いて、それらオブジェクトのイベントのやり取りによって各オブジェクトの内部状態を更新していき、その副作用として処理を行うという計算モデルです。いってみれば、イベントをお互いに授受する有限状態機械の集合としてシステムを構成します。したがって、問題領域やシステムをモデル化するのに、静的な視点からは、各オブジェクトの内部構造および各オブジェクト間の接続構造を併せて表現する「クラス-関連図」、動的な視点からは各オブジェクトの状態機械としてのライフサイクルを外部イベントと対応するアクションとして表現する「状態遷移図」がモデル表現の基礎となります。

実際には、他にもいくつかのモデル表現を組み合わせて開発を進めていくのが普通です。一度に複数のことを考えられない、静的な構造に比べ動的な振る舞いについての推論能力が弱いという人間の性質を考慮してのことです。

問題やシステム全体をその環境中で大局的に捉えるという視点が必要です。対象領域全体をブラックボックスオブジェクトとして捉え、それが外部のどんなタイプの利用者に対しどのようなサービスを提供してくれるのか、を定義したユースケース図を描きます。利用者のタイプのことをアクター、個々のサービスのことをユースケースと呼びます。システムの利用(ユース)の事例パターン(ケース)という意味で、スウェーデンのI.Jacobson氏によって導入された概念です。ユースケースは、構造化技法における機能分割と変わりがないのではないか、本当にオブジェクト指向にフィットするのかという批判がありますが、両者の大きな違いは、機能分割が最終的に実装コードにおちるまで繰り返し適用されるのに対し、ユースケースはあくまで利用者の視点から彼らに意味のある機能やサービスを提供するというレベルで定義しそれ以下には分割しない点です。このことは、オブジェクト指向で企業活動自体をモデル化するに際し、顧客満足度という観点から企業をリエンジニアリングをする際の基本的視点としてユースケースが用いられていることからもわかります。

ユースケースはあくまでもそれを実現するオブジェクトの集団とそれらの相互作用を導き出すためのきっかけです。同じシステムがいろいろな利用者から様々な要求を出されそれに反応してある機能を発揮して処理が進んでいきます。このとき、どのような実現方式をとるかは、各ユースケースをどのようなオブジェクトの組み合わせでどのように協調動作させるかで決まります。そうした実現方式を検討するのに、通常、シーケンス図やコラボレーション図が使われます。

（a）ユースケース図



（b）ユースケースとオブジェクト集合

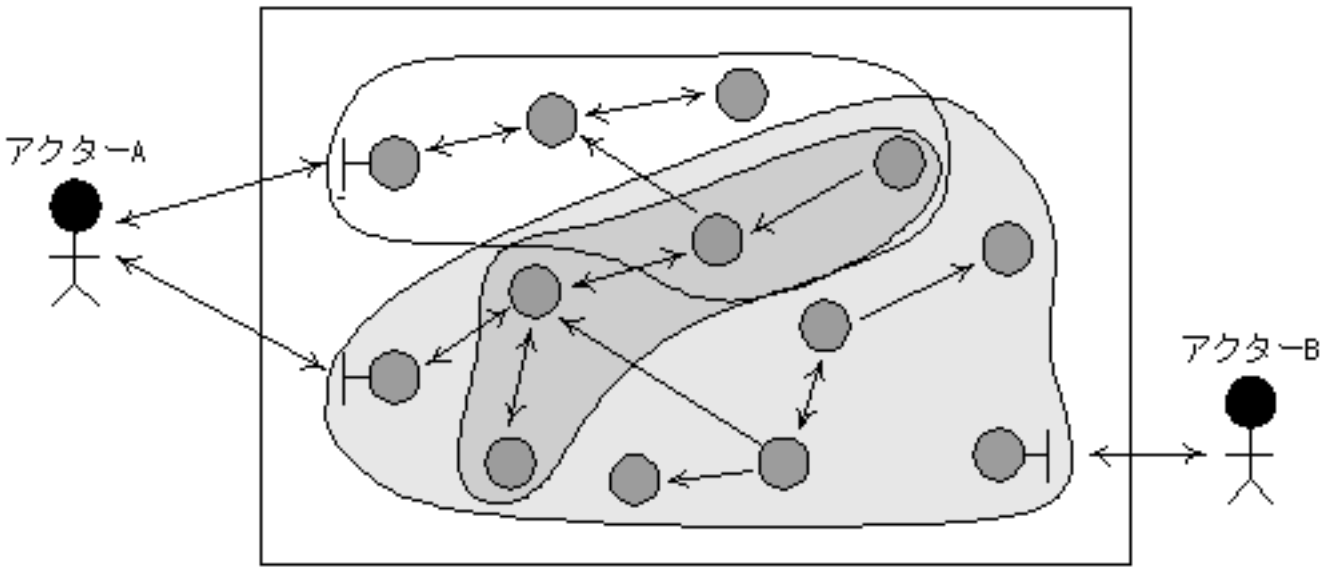


図3：システムとユースケースとオブジェクト集合

１つのユースケースはMVCアーキテクチャによって構成すると機能変更や拡張等の保守の点からも便利です。MVCとは、モデル-ビュー-コントローラの3組みでシステムを実現する構成法で、SmalltalkのGUIフレームワークとして採用されたことから広く知られるようになりました。知識(モデル)と外部との情報のやり取りの仕方(ビュー)とそれらを特定の機能を発揮させるためにコーディネートするマネージャ(コントローラ) を分離して管理するわけです。それをユースケースの構成に応用したのもやはりJacobsonn氏です。ユーザや外部システムあるいは他のユースケースとの対話を管理するバウンダリオブジェクト(論理ビュー)、ユースケースを実現するためにバウンダリや各ドメインオブジェクトとの間でのワークフロー・メッセージの流れを制御管理するコントローラオブジェクト、そしてユースケース実現に必要な問題領域の知識や情報を保持するドメインオブジェクト群(モデル)という3種類のオブジェクトカテゴリを用います。外部とのやり取りは必ずバウンダリオブジェクトを通し、またドメインオブジェクトどうしのやり取りも特定のユースケースに関する場合は必ずコントローラ経由で行うというのがMVCの基本です。こうすることで、ドメインオブジェクトが特定のアプリケーション機能（つまりユースケースやビュー）に依存した実装になることを防ぎ汎用性・再利用性を確保し、また複数の

ユースケースから同時に利用できることにもなります。

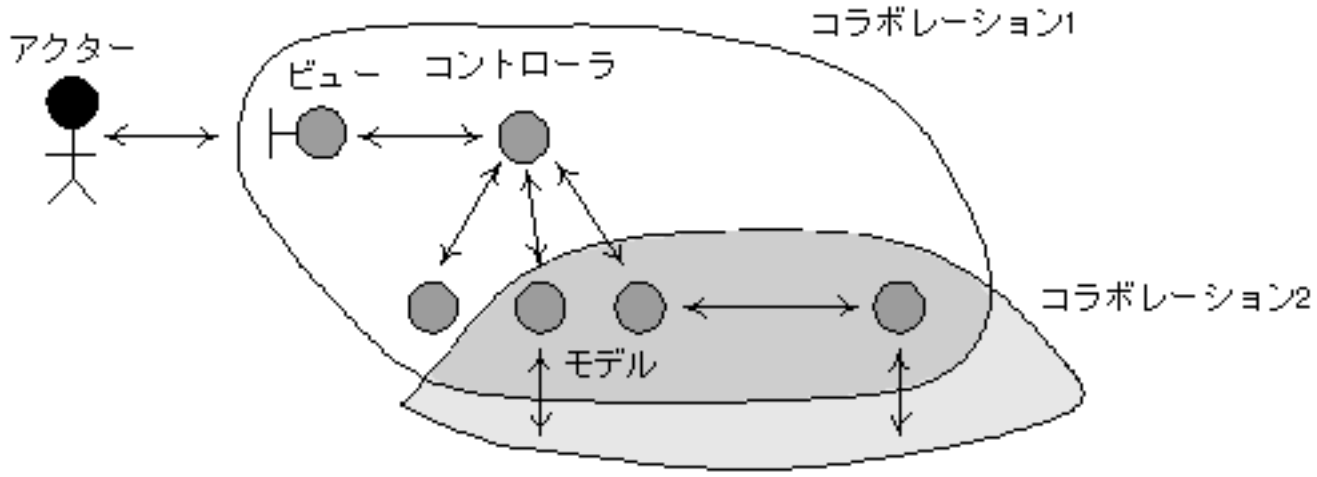


図4：ユースケースとその実現としてのコラボレーション

こうして各ユースケースごとに識別されたドメインオブジェクト群をマージし、整理してクラス分類し、各クラス間の関連や集約を識別して、問題領域を表現した分析モデルをクラス-関連図として記述します。またコントローラも場合によってはユースケース間で共有できることがあります。コントローラやバウンダリオブジェクトも整理しクラス-関連図を作成しますが、これは設計モデルを作成する際の核になります。またユースケース単位の状態遷移は結局、各ユースケースを制御するコントロールオブジェクトの状態遷移ということになります。

このようなオブジェクト指向モデルに対する様々なビューを図式表現する標準的なモデリング言語としてUML(Unified Modeling Language)が1997年12月にOMGで採択されました。今後、UMLがオブジェクト指向システム開発のさまざまな局面でドキュメント記述に利用されていくことになります。

Prev. Index Next

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

４．開発プロセスとパターン/アーキテクチャ

オブジェクト指向で開発を行う場合に基本的には、要求分析、システム分析、システム設計、オブジェクト設計、実装というサイクルをスパイラルに数回繰り返しながら、最終的なシステムに仕上げていきます。従来もこうした繰り返し型の開発がやりたかったプロジェクトも多々あったわけですが、オブジェクト指向をベースにした開発になるまではスパイラル方式を採用できなかったのが実状です。としうのも、サイクルの各フェイズで用いる概念とダイアグラムがまったく異なっており簡単には行ったり戻ったりできなかったからです。それに対し、オブジェクト指向の場合、分析、設計、実装といった各フェイズで用いられる概念も図式もともにオブジェクトモデルであり、かつUMLという標準のダイアグラムであるため、オブジェクトを単位として作業の各フェイズでのトレーサビリティを確保しやすいのです。

ところが、トレーサビリティをつけやすいといっても従来のOO開発技法にはどうしても分析と設計の間に大きなギャップがありました。従来は、分析モデルを「詳細化」して設計モデルを得るというガイダンスがなされ、分析モデルにアドホックに設計上のクラスや属性･操作を追加してお茶を濁していたのです。しかし、実際には設計では、ターゲットプラットホームの制約や効率、ミドルウェアやクラスライブラリの特性を考慮した微妙なセンスが要求されるもので、結局、能力のある設計者に任せざるを得ませんでした。

それに対し、近年、分析モデルと設計モデルの関係に対する見直しが図られています。分析モデルは設計モデル中の問題ドメインサブシステムおよびアプリケーションサブシステムの一部に埋め込まれるものだという考え方です。設計モデルを考える際に、システム構成を表現したアーキテクチャの上で個別の設計を詰めていくという方針です。アーキテクチャは問題ドメインの性格、特に制御の特性に応じて構成が変わることが知られていますが、標準的なアーキテクチャは、プラットホーム層、ミドルウェア層、問題ドメイン層、アプリケーション層、プレゼンテーション層といった構成になります。

オブジェクト設計に関しても、最近は、設計目標に応じたデザインパターンをパターンカタログから選び設計制約を満足させるという手法が提唱されています。デザインパターンとは、設計課題とそれに対する一般的な解決策をペアにして記述した解法テンプレート集です。オブジェクトの生成方式に関するさまざまなファクトリパターン、複合オブジェクトの構成方式やアクセス方式に関するパターン、複数のオブジェクト間の相互作用や通信・制御の方式に関するさまざまなパターンがあります。再帰的な集約オブジェクトの構成法であるコンポジット・パターンやサブシステムの共通APIを提供するファサード・パターン、オブジェクト間の間接的な依存関係を実現するオブザーバ･パターン、代理オブジェクトによるアクセス制御を提供するプロキシー･パターン等が代表的なものです。

実際には、カタログから選んだ解法を個別状況に合わせてカスタマイズする作業が発生しますが、カタログにどのようなパターンが載っているかあらかじめその全体

像を習得しておくことで、効率的にある程度の設計が平均的なエンジニアにも行えるという大きな利点があります。現在普及しているデザインパターンは柔軟性や拡張性や保守性を目標としたものがほとんどで、特定の機能や効率といった目標を解決するパターンの開発は今後の課題です。逆に言うと現在のデザインパターンは、設計において効率よりもソフトウェア工学上の原則を最大限引き出すためのものがほとんどです。

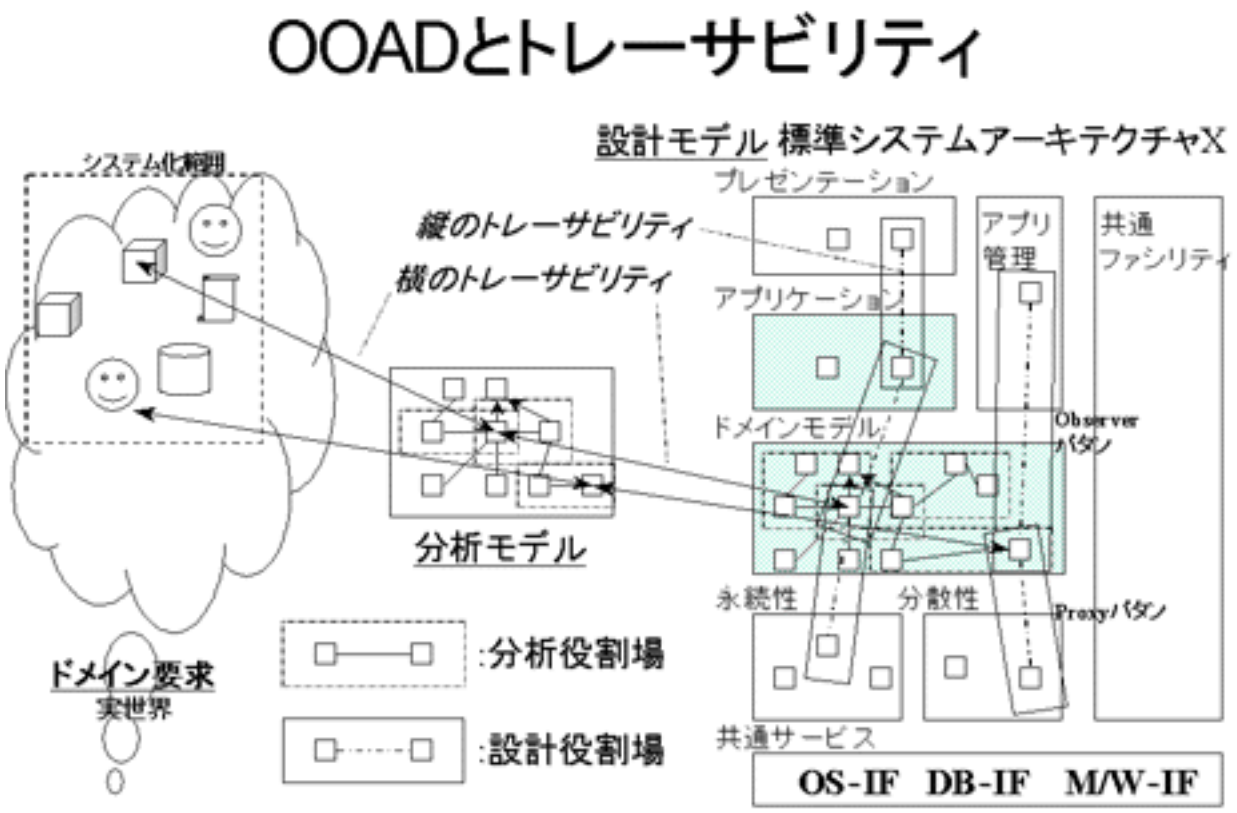


図5：システムアーキテクチャとトレーサビリティ

またデザインパターンは、アーキテクチャと組み合わせることによって、システム設計を補強することができます。つまり、問題ドメイン層を始めとする個々のレイヤー間のスムーズな接続と頑健さの増強に、デザインパターンを「糊」として使い、全体として設計モデルを構造化するという技法です。各レイヤー間のインターフェースにファクトリ、シングルトン、ファサードやオブザーバ、プロキシー･･･といったパターンが有用です。さらに各レイヤー内のサブシステム内ではほとんどすべてのパターンがオブジェクト設計に活用できます。

オブジェクトの単位でなくもっと大きなアプリケーションの単位で再利用性を提供するものにフレームワークがあります。サブクラスを定義することで実際実行可能なアプリケーションプログラムを提供するものです。汎用的な制御ロジックを抽象クラスのレベルでテンプレートメソッドとして記述しておき、変更可能部分をサブクラスのメソッドとして追加定義させる仕組みです。パラメータ部分自身がオブジェクト化されているわけです。こうすることで、ERPパッケージなどより非常に柔軟なカスタマイズを可能にしています。こうしたフレームワークはGUI分野に始まり、最近ではプロセス制御や金融派生商品リスク管理といった特定の業務ドメイン用のものも登場してきています。また、ビジネスドメインでのフレームワークの基本部品のことをビジネスオブジェクトと最近では呼びます。フレームワークやビジネスオブジェクトの設計にもデザインパターンは活用されています。

フレームワークの利用には一部コーディングが必要でしたが、継承を使わず、したがってコーディングも再コンパイルも必要なく、一切を既存部品オブジェクト(コンポーネントと呼ぶ)の組合せ(ワイヤリング)だけで済ませる方式をコンポーネントウェアと呼びます。コンポーネント実現のポイントは、通常のオブジェクトと違い、外部インターフェースとして呼び出し可能な操作のセットだけでなく、そのコンポーネントの内部状態遷移を示す外部イベントのセットも公開していることです。したがって、ソースコードを参照することなく、コンポーネントどうしで必要なメッセージのフローを組み合わせる（すなわちワイヤリングする）ことができるわけです。

Prev. Index Next

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

５．技術者としての心得

優れたエンジニアと優れた研究者ではその目指すところも心得も異なるでしょうが、共通していえるのは、好奇心と向上心と対人能力でしょう。ソフトウェア工学は実学であるべきで、研究者は現場の苦労と悩みを、現場は最新の研究テーマを知ろうと努力することが必要です。仕事や専門と関係のない本や人と知り合いになる努力を怠らず、専門へのフィードバックを心がけましょう。勉強の仕方も基本的には独学で何でも勉強できます。同好の士を募り仕事とは違うテーマで勉強会をしてください。翻訳や出版という目にみえる成果を目標にするのも得策です。

本屋も有効活用できますし、古本屋さんもあなどってはなりません。専門分野ごとの古書店の書棚には、その分野の歴史が古い本も新刊もわけへだてなく共時的に配置され、新刊本でもすぐ値くずれを起こすような本は並べられていません。情報の洪水の中で相対的によい本を目にできます。その点、Internetの世界ではまだそのような役割を果たしてくれる仕組みが存在しません。

日々の仕事の中で技術を磨いていく必要があります。その際、これぞという師匠をひとりみつけて納得できない部分はとことん聞くべきです。どんな仕事でも技術的にちょっとした新しい試みのできる部分は何個所かあるもの。またすべての仕事はOJTであるという意識をどこかで持っていた方が気が楽です。どうしてもそう思えないなら転職を考えましょう。今後ますます会社という組織を固定的にとらえる必要のない時代になっていきます。自分の仕事をしていく上でのよき秘書という意識で会社を見直すのも１つの考え方です。

開発の現場に関していうと、実際のシステム開発はオブジェクト指向だけで事がかたづくような単純な話ではありません。言語、ミドルウェア、ツール等のノウハウが必要になります。分析設計技法をメンバーに習得させる必要と、システムの基本構成を定義し作り込むアーキテクトや設計ノウハウをもったエンジニアの育成も重要です。アーキテクトと協力して開発上のリスク管理を中心にスパイラル型でプロジェクト全体を運営していくリーダーの選定･育成、そしてプロジェクトへのクライアント参加および開発側上司の理解取り付けも重要です。いわゆるプロジェクト運営能力およびそれをバックアップする組織の育成が最大の課題といいうことです。しかし、こうした問題にも開発プロセスやプロジェクト自身のモデル化ということでオブジェクト指向技術の応用が考えられます。

私が日々やっているコンサルティングという仕事は、原理的にはお客様の抱えている問題を解決するお手伝いをすることですが、その基本はそのクライアントがそもそもどのような問題を抱えているのか、その構造と意味に気づくように仕向けるという、言ってみれば精神分析医のような側面が重要です。その意味でも、モデリングは一番重要な要素になるわけです。

現在のように過去の制度・価値観が大きくゆらぎ政治経済的にも危機に瀕しているいまこそが仕事の上でも研究の上でも新しい面白いことをするためのチャンスで

す。皆さんに可能性は大いに開かれているといえます。


Prev. Index Next

オブジェクトの広場

［新人およびOO初心者に贈る「オブジェクト指向」本格入門］

6．新人のための図書ガイド

いわゆる入門書のたぐいは省略しました。なお，本節の内容をより詳しく知りたい向きはぜひ，以下のHPを参照してください．

http://www.ogis-ri.co.jp/otc/hiroba/OoBook/Beginner/index.html

オブジェクト指向概念形成史

算法表現論、木村泉、米澤、岩波書店(絶版ですので図書館か古本屋さんで探しましょう．それだけの苦労をする価値のある1冊)

特に木村泉氏の書かれた序章は、自分で抽象データ型を発見しなかったソフトウェア技術者は一度は目を通しておくべきでしょう。スパゲッティプログラムからいかにそれをほぐして構造化、データ抽象、抽象データ型とプログラムの複雑さに対処するための概念を発展させてきたかが、富士登山の代わりに擬似的な築山である「お富士さん」に登るように、パイオニアの登山者達の気持ちとその時点での眺望が追体験できます。米澤氏の手になる後半もソフトウェアの基礎理論を簡潔にまとめてあって便利。

オブジェクトモデリングの心に触れたい向きに

オブジェクトモデリング、落水浩一郎他、ジャストシステム現在はプレンティスホール(最近，出版社も装丁も替わって入手し易くなりました．)

重要ポイントは薄いこの1冊に含まれています。次に、「オブジェクト指向方法論OMT」ランボー他、トッパン、に進みましょう。

オブジェクト指向の研究テーマのヒントが欲しい人

オブジェクト指向早わかり、ウェグナー、共立

オブジェクト指向とがっぷり四つに組んで基礎（とは初歩ではない）から取り組みたい向きの必読書

バートランド・マイヤー、オブジェクト指向入門、アスキー出版

7．さいごに

なお本記事は，共立出版コンピュータサイエンス雑誌bit 1998年 5月号: オブジェクト指向技術紹介 として掲載されたものをベースに改訂補筆したものです．