04 モデリング
～UMLの根底となる考え方～

UMLの目的はシステムなどをモデリングすることです。物事を抽象的に捉えることであるモデリングについて学んでいきましょう。

目的に合わせて分かりやすく表現するモデリング

UMLはモデリングを行う言語です。モデリングとは物事を抽象的にすることですが、ここではモニタリングについて、もう少し詳しく説明していきます。

例えば、「品川から新宿に電車で移動したい」という人がいるとします。移動するために、移動経路の情報が必要なのですが、品川から新宿までのエリアが詳細に記載された地図では情報が多すぎます。電車での移動に絞った場合、詳細な地形の情報を知る必要はありません。よって、必要な情報を強調し、更に地図を抽象化した図の方が良いです。さらに、品川から新宿までは山手線一本で行けるので、山手線の外回りの路線図があれば十分です。

モデリングとは抽象化することですが、このように**目的に合わせて不要な情報**を削り、分かりやすく表現することともいえます。

■電車移動のために地図をモデリングした路線図

実際の地形情報　簡略化した地図上に路線を表示　路線図

システムのモデリング

モデリングとは物事を抽象化して分かりやすく表現することでした。それでは、システムをモデリングするとはどういうことでしょうか？

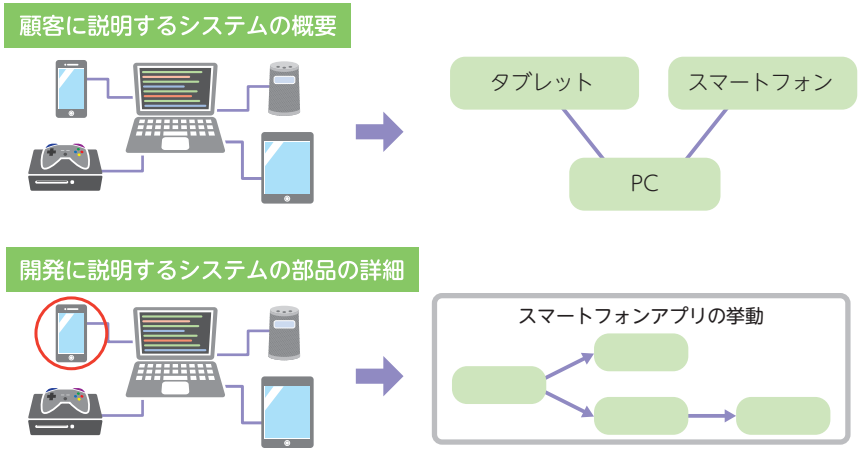
モデリングを行うためには、まずモデリングを行う目的が必要です。よって、まずシステムをモデリングする目的を考えます。システムをモデリングする目的は様々なものが想定できますが、主には次の2つが考えられます。

・顧客にシステムの概要を説明するため
・開発者にシステムの詳細を説明するため

顧客はシステムの詳細なプログラムの説明などよりも、システムの使い方などの情報を求めていることが多いです。よって、システムの利用方法という観点で不要な情報を削って、システムをモデリングしていきます。

一方、開発者はシステムの一部を担当することが多いので、当面システム全体の概要などよりもシステムの一部のプログラムに関わる情報を求めていることが多いです。よって、一部のプログラムに関わる情報以外を削って、システムをモデリングしていきます。このようにすることで、**顧客にとって分かりやすい図**と**開発者にとって分かりやすい図**を表現することができます。

■顧客向けの図と開発者向けの図



05 オブジェクト指向の3大要素

オブジェクト指向をより効果的に活用していくために、3大要素と呼ばれるものがあります。ここではオブジェクト指向の3大要素について学んでいきましょう。

カプセル化（パッケージ）

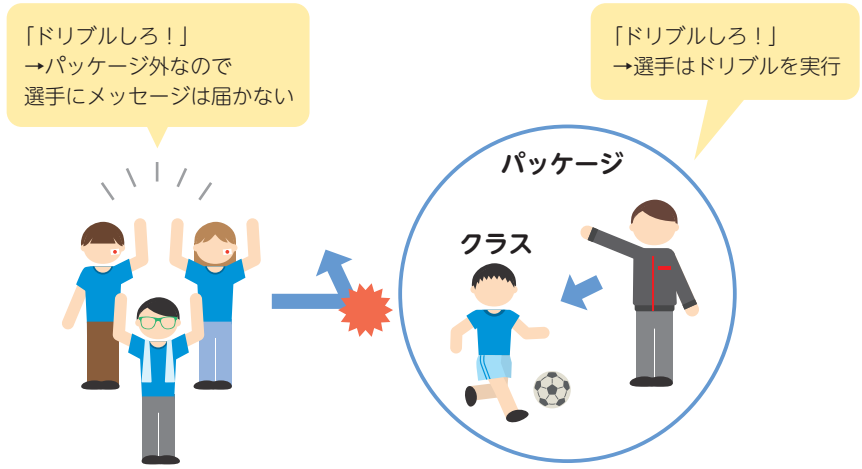
オブジェクト指向では、インスタンス間でメッセージを送りあい、振る舞いを連携させることでシステムを実現させていきます。メッセージを送りあうということは、依存関係（2-03参照）があるということです。依存関係が煩雑になると保守性が下がります。

例えば、選手インスタンスが他の選手インスタンスや監督インスタンスからだけでなく、観客インスタンスなど他のクラスのインスタンスからメッセージを受け付けることが可能であったとすると、監督も観客も選手に対して依存している関係になります。この時、選手クラスの振る舞いにバグが見つかり修正などを行った際に、選手クラスにメッセージを送っている全てのクラスを確認、修正する必要が出てきます。

そこで、乱雑にメッセージを送りあわないように、**パッケージ**という機能でメッセージを制限していきます。メッセージが頻繁に行われるクラスのグループをパッケージとしてまとめ、パッケージの外のクラスからのアクセスを制限します。例えば、選手クラスと監督クラスは同じパッケージに入れますが観客はそのパッケージに入れません。このようにすることで、観客からは選手に直接指示を出して利用できなくなります。

このようにオブジェクト指向において、**クラスの利用され方を制限する機能をカプセル化**といいます。カプセル化を行うことで、不要なアクセスを制限してシステムのメンテナンス性を高めることができます。更に、利用できる機能を絞ることでシステムの使い方を分かりやすくできます。

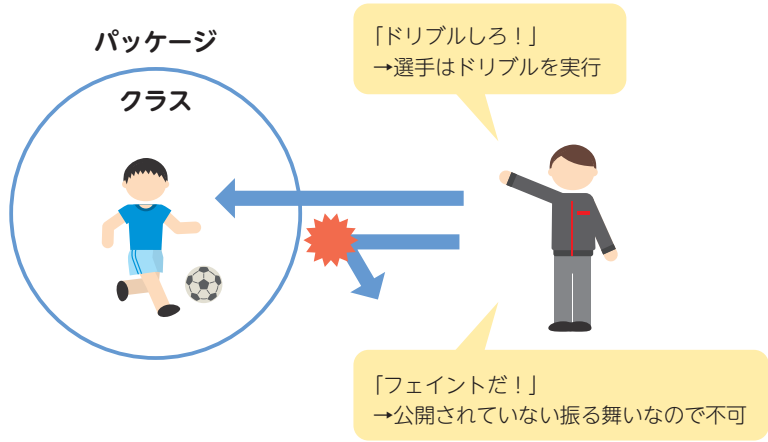
■ パッケージによるカプセル化のイメージ



カプセル化（アクセス修飾子）

カプセル化のもう1つの機能は振る舞いを非公開に設定する方法です。例えば、選手クラスの「ドリブル」の振る舞いは他のクラスに公開してメッセージを受け取りますが、「フェイント」の振る舞いは非公開で自分のクラスの中で必要に応じて自分の判断で使う振る舞いになっている、というような形です。

■ アクセス修飾子によるカプセル化のイメージ



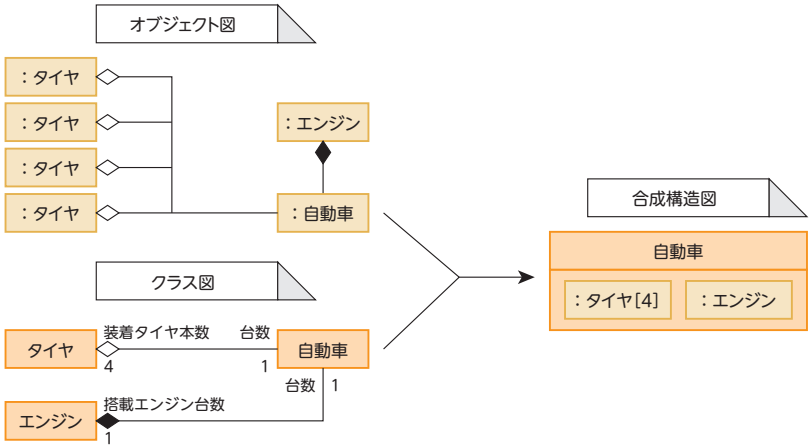
05 合成構造図
～図と図を構成する要素図形～

クラスやオブジェクトの構造関係について図示するのが合成構造図です。ここでは、合成構造図について学んでいきましょう。

合成構造図とは

合成構造図は構造図の一種です。**クラスやオブジェクトの関係**を表せます。合成構造図はオブジェクト図とクラス図を合成し、視覚的に構造を分かりやすくした図といえます。

■オブジェクト図／クラス図→合成構造図



また、インターフェースなどを用いた依存関係も分かりやすく表現できます。ただし、何をクラスとして何をオブジェクトとして説明するのかを目的によって使い分ける必要があります。

合成構造図で用いる要素図形の種類

合成構造図で主に用いる要素図形はパートと構造化分類子とインターフェース（2-03参照）とポートです。

合成構造図で用いる要素図形（パート）

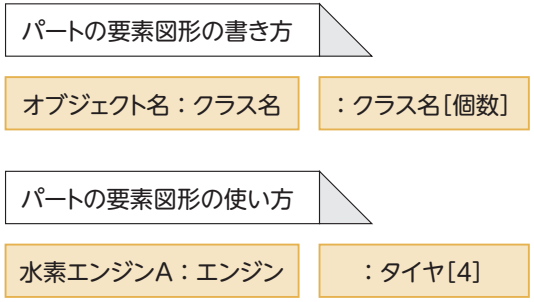
パートの要素図形は矩形で表記します。合成構造図におけるパートは基本的にオブジェクトのことを指します。要素図形の中に「**オブジェクト名:クラス名**」の形でオブジェクト名とクラス名を表記します。

例えば、エンジンのクラスにモデリングされるオブジェクトである水素エンジンAを表現する場合、「水素エンジンA：エンジン」と記載された矩形を表記します。

また、オブジェクト名が定まっていない場合は「**：クラス名**」のように省略して表記できます。

更に、同じクラスに分類される複数のオブジェクトを表現する場合は「**：クラス名[個数]**」のように表記することもできます。例えば、タイヤのオブジェクトが4つある場合はタイヤのパートの要素図形を4つ書くこともできますが、「：タイヤ[4]」と記載した1つのパートにまとめられます。

■合成構造図で用いるパートの要素図形の書き方と使い方例



07 パッケージ図
～要素の関係性を示す関連線～

要素図形間の関係性は関連線で表します。ここでは、パッケージ図で用いる関連線について学んでいきましょう。

パッケージ図で用いる関連線の種類

パッケージ図で主に用いる関連線は依存（2-03参照）とマージの線です。

パッケージ図で用いる関連線（依存）

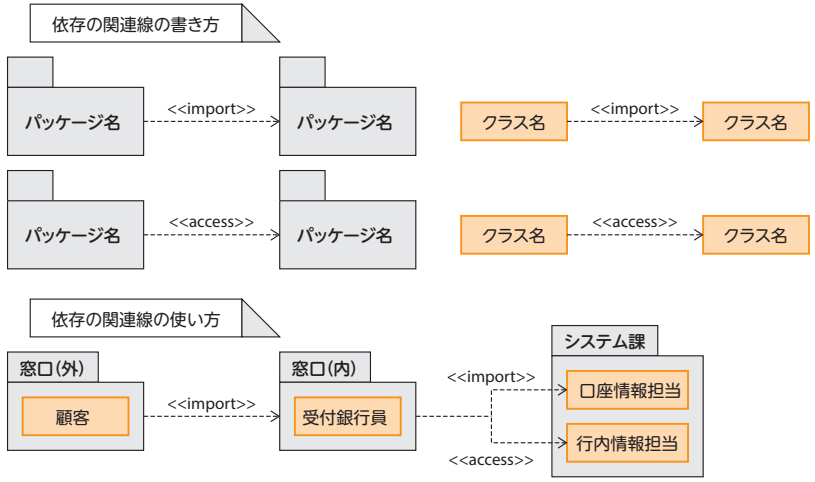
パッケージ図で用いる依存は、アクセスとインポートの2種類に分けられます。アクセスとは、**依存先にあるパッケージ内のクラスやインターフェースを、依存元のパッケージ、クラス、インターフェースで利用できるようにする**ことです。インポートとは、**依存関係にあるパッケージ内のクラスやインターフェースを、依存元と更にその依存元のパッケージ、クラス、インターフェースで利用できるようにする**ことです。インポートの方が、より依存性が高いといえます。

2種類の違いの例として、例えば、銀行の振込業務において、顧客は受付の人を通して、システム課の人から自分の残高の確認処理を行えます。一方、窓口の人を通しても銀行の行内の情報を知ることはできません。しかし、窓口の人は銀行の行内の情報を知ることができます。いずれも受付の人とは依存関係のある機能ですが、残高の確認処理はインポート、お金の引き出しはアクセスです。

アクセスの関連線は**先端に矢印を加えた点線の上に<<access>>とステレオタイプ**を加えて表記します。

インポートの関連線は**線の先端に矢印を加えた点線の上に何も表記しないか<<import>>とステレオタイプ**を加えて表記します。

■ パッケージ図で用いる依存の関連線の書き方と使い方例



パッケージ図で用いる関連線（マージ）

マージとは、**2つのパッケージを合体させて1つのパッケージとして扱う**ことです。依存のインポートとの違いは、インポートは依存元と依存先に同じ名前のクラスがあった場合、依存元のクラスは依存先のクラスで上書きされます。一方、マージはクラスの要素が合体し、両方のクラスの属性と振る舞いを兼ね備えたクラスが生成されます。マージの関連線は**先端に矢印を加えた点線の上に<<merge>>とステレオタイプ**を加えて表記します。

例えば、会社の研究部門と開発部門を統合して研究開発部門を作る場合、2つの部門を研究開発部門にマージします。両方の部門にそれぞれ別にいた会計担当者は、統合後、両方の部門の会計担当者の業務を引き継いだ会計担当者となります。パッケージ図では研究部門のパッケージと開発部門のパッケージからマージの関連線を引き、研究開発部門のパッケージに繋ぐことで表現します。

このように別のシステムでは分割していたパッケージを新しい統合したパッケージで再利用する時などに使われます。

01 ユースケース図
～図と図を構成する要素図形～

システムで利用できる機能について図示するのがユースケース図です。ここでは、ユースケース図について学んでいきましょう。

ユースケース図とは

ユースケース図は振る舞い図の一種です。ユースケース図では**システムにはどのような機能があるのか**と、**誰がその機能を使えるのか**を表現できます。システムが備える機能のことをユースケースといい、機能を利用する誰かのことをアクタといいます。アクタは人間に限らずシステムを利用する別のシステムも含みます。ユースケース図はUMLの中でも特に分かりやすい図で、システムの利用できる機能について顧客に説明する目的で使われることが多いです。

■ ユースケース図はシステムの機能を把握できる分かりやすい図

ユースケース図の表記方法を用いて、システムについて詳細に説明することも可能ですが、詳細に表記しすぎると複雑で分かりづらくなってしまうことがあります。目的に合わせて分かりやすく表記することが重要です。実践的な例を参考にしたい方は8-06を参照してください。

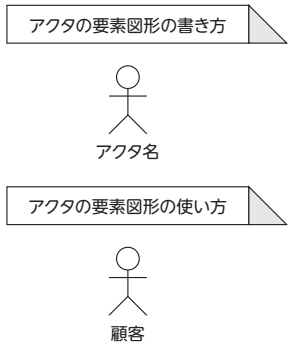
ユースケース図で用いる要素図形の種類

ユースケース図で主に用いる要素図形はアクタとユースケースです。

ユースケース図で用いる要素図形（アクタ）

アクタの要素図形は棒人間で表記します。図形の下にアクタ名を表記します。

■ ユースケース図で用いるアクタの要素図形の書き方と使い方例



ユースケース図で用いる要素図形（ユースケース）

ユースケースの要素図形は**楕円**で表記します。図形の中にユースケース名を表記します。拡張ポイントがある場合は図形の中央に線を引いて区切り、上にユースケース名、下に拡張ポイント名を表記します。

拡張ポイントとは、ユースケースの機能に含まれるクラスの振る舞いが、別の機能を利用する場合があることを明記するものです。例えば、銀行の口座引出機能は本人確認を行う振る舞いが含まれますが、この本人確認の振る舞いは運転免許証の確認機能やマイナンバーカードの確認機能などの別の機能を呼び出します。この時に本人確認の振る舞いは**拡張ポイント**と呼ばれます。

05 シーケンス図
～要素図形を分ける区分～

区分を用いると図の中の要素図形を意味上で分類できます。ここでは、シーケンス図で用いる区分について学んでいきましょう。

シーケンス図で用いる区分の種類

シーケンス図で主に用いる区分は複合フラグメントです。

シーケンス図で用いる区分（複合フラグメント）

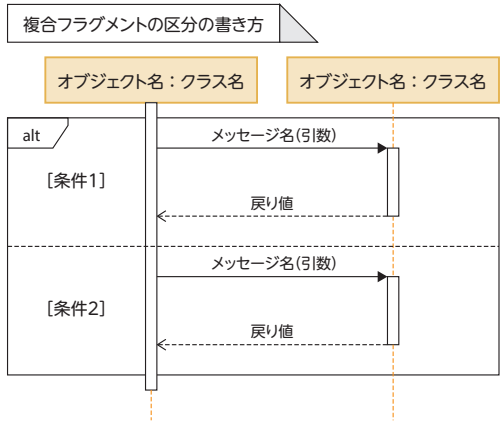
複合フラグメントの区分は**矩形**で表記します。左上に複合フラグメントの種類を表記し、中に要素図形や関連線を含められます。区分を点線で区切る場合もあります。代表的な複合フラグメントの種類は次の通りです。

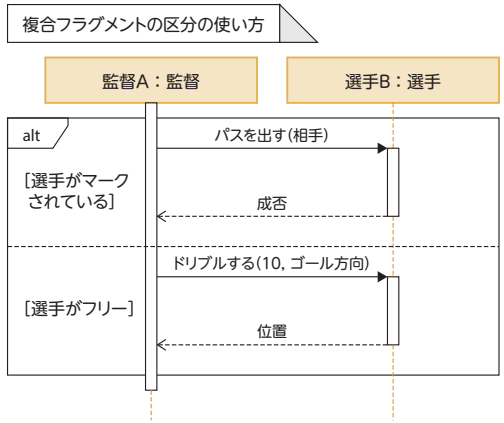
複合フラグメントはシーケンス図の表現力を上げられますが、多用すると可読性が落ちる問題もあります。複合フラグメントを多用しなければならない場合は6-06の相互作用概要図と組み合わせて使う方が良い場合もあります。

種類	表記	機能
分岐処理	alt	条件が満たされた区分が実行される処理
条件処理	opt	条件が満たされた時だけ実行される処理
弱順序処理	seq	区分毎の区分内のメッセージが順番に実行されることが保証される処理
中断処理	break	条件が満たされた時に中断される処理
並行処理	par	各区分が同時に実行される処理
強順序処理	strict	区分毎の区分内のメッセージとそのメッセージ先の処理も含め順番に実行されることが保証される処理

種類	表記	機能
ループ処理	loop	繰り返し実行する処理
排他処理	critical	他からの割り込みをされない処理
否定処理	Neg	通常は発生しない処理
評価処理	assert	属性などの値が正当であるかを評価する処理
無効処理	ignore	図が表す機能とは関係のない処理
重要処理	consider	必ず実行される重要な処理

■ シーケンス図で用いる複合フラグメントの区分（分岐処理）の書き方と使い方例





02 要件定義
～システム開発の目的を定める～

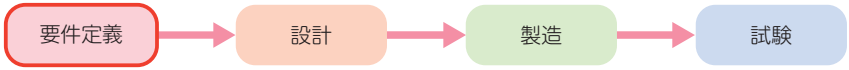
システムの開発が始まってまず行うのが、顧客の要求をシステムの仕様に落としこんでいく要件定義です。ここでは要件定義について学んでいきましょう。

要件定義とは

要件定義はシステムの開発が決まり、最初に行う工程です。システムを開発するにあたり、まず定めなければならないのは、これから開発するシステムは何のためのシステムか？ということです。例えば、受注生産の場合は顧客の要求を満たすため、自社などで開発する場合は社会の要求を満たすためなどに開発を行います。これらの顧客や社会の要求を満たすシステムの機能について定義し、分析して、その機能の中で今回の開発で実現すべき機能を割り出します。これらの機能のことを要件といいます。

このようにして、**顧客や社会の要求を満たすためにシステムに組み込む要件を分析・定義すること**を要件定義といいます。要件定義では、UMLなどを含めた要件定義書を成果物にするのが一般的です。

■ 要件定義工程の位置づけ



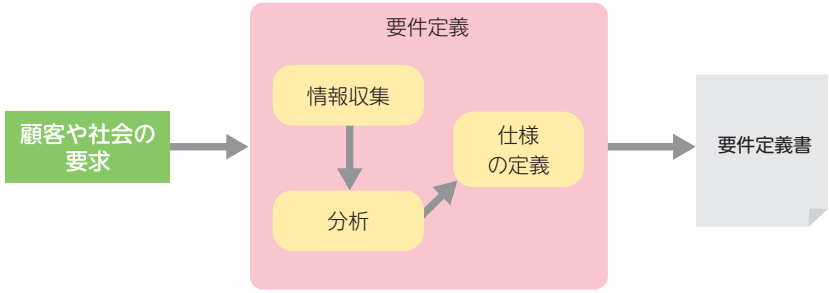
要件定義で行う作業

要件定義では、情報収集、分析、仕様の定義を行っていきます。

まず初めに情報収集の作業を行います。情報収集の情報とは顧客の要求に繋がるものです。情報収集の作業の方法は大きく分けて2つの方法があります。1つは**トップダウンアプローチ**、もう1つは**ボトムアップアプローチ**です。トップダウンアプローチとは、経営者や企画者などからシステム開発の背景や目的、概要などを情報収集する方法です。ボトムアップアプローチとは、現場などからこれから開発するシステムが実現する業務の現状などを情報収集する方法です。次に分析の作業を行います。分析では収集した情報の矛盾や重複などが無いように整理していきます。最後に、仕様の定義の作業を行います。仕様の定義では分析した結果を元に、今回の開発で作るべき機能について定めていきます。この機能が要件になります。このように情報（顧客の要求）から要件を定義していくので要件定義と言われます。

例えば、銀行の窓口業務のシステム化のプロジェクトについて考えます。まず、システム化の企画を出した銀行の担当者からトップダウンアプローチで、今回のプロジェクトの背景や目的、概要を確認します。更に、ボトムアップアプローチとして、現場での窓口業務のマニュアルの入手や担当者へのヒアリングを行います。次に、入手した情報を分析します。分析の結果、窓口業務には、振込と引出があることが分かりました。その中で、今回の案件では緊急度が高い、振込の機能を要件として開発することに決まりました。そこで、これらを資料化し、要件定義書にする、というような流れです。

■ 要件定義工程の作業手順例



05 業務分析

業務分析ではシステム化対象の業務はどのようなものかの分析を行います。ここでは、仮想のシステム開発の例で分析方法を学んでいきましょう。

業務分析とは

業務分析では、**システム化対象の業務を整理**していきます。トップダウンアプローチで収集した情報から、大まかな業務について理解します。更にボトムアップアプローチ等によって収集した情報から、業務がどのような手順で、どのようなデータを用いて行われているのか、システム化後はどのように行われるようになるのかをモデリングし図で示します。

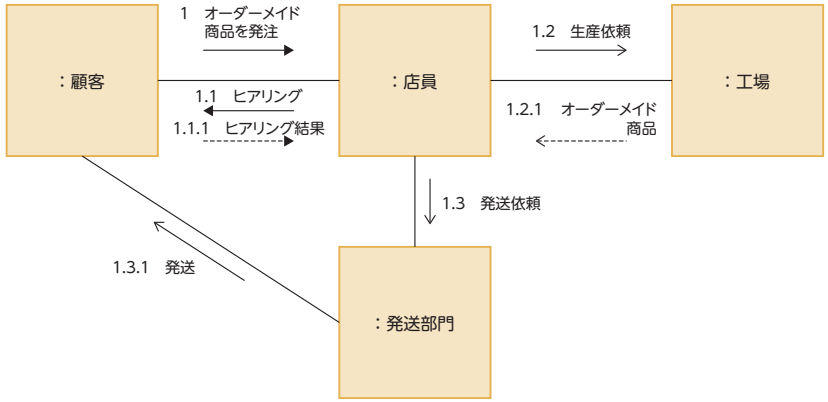
業務分析では業務の整理は業務概要図や業務フロー図等を使って行います。これらの図は詳細に記載するのではなく、概要が分かるように抽象的に表現することが多いです。業務概要図は業務の関係性を示し、業務フロー図は業務の手順を示します。

業務分析における業務概要図

業務分析における業務概要図を、コミュニケーション図（6-01参照）を用いて表現します。コミュニケーション図は通常インスタンスの相互作用のような詳細な設計時に用いられますが、ビジネスコンテキストや業務の概要を表す際にも活用できます。

ここでは、システム化前のオーダーメイド商品販売の概要構成について記載します。顧客がオーダーメイド商品を発注するとスタッフと工場と発送部門が連携して商品を製造し、発送するまでの流れをコミュニケーション図で表現しております。コミュニケーション図を用いてキャンセルの流れなども1つの図で表現することは可能ですが、要件定義においては顧客に説明するために簡単な図にすることが望ましいです。よって複雑な分岐やイベント駆動処理などは記載せず、複数の図に分けるなどが一般的です。会員の認証や商品の状況確認やキャンセルについては省略し、オーダーメイド商品の販売業務の通常の流れに絞って記載しております。

■システム化前のオーダーメイド業務の関係



業務分析における業務フロー図（商品選択）

業務分析における業務フロー図を、アクティビティ図（5-02参照）を用いて表現します。ここでは、要求の収集によって得られたシステム化前のオーダーメイド商品販売の流れについて記載します。

アクティビティ図を用いてキャンセルの流れなども1つの図で表現することは可能ですが、要件定義においては顧客に説明するために簡単な図にすることが望ましいです。よって複雑な分岐やイベント駆動処理などは記載せず、複数の図に分けるなどが一般的です。ここでは基本的な流れだけ紹介します。

オーダーメイド業務追加により、顧客が選択した商品がオーダーメイド商品であった場合に実行される、新たな機能が追加されます。

従来の処理の流れは、顧客選択した商品の情報を、システムが商品購入画面に表示するだけのものでした。

06 ユースケース分析

ユースケース分析ではシステムの利用方法はどのようなものかを分析します。ここでは、仮想のシステム開発の例で分析方法を学んでいきましょう。

ユースケース分析とは

ユースケース分析とは、システムの利用者とシステム化の範囲と使い方（機能）について行う分析です。機能定義図を使って**システム化する対象範囲と使用できるユーザを明確**にします。また、オブジェクト状態図や画面遷移図によってシステムの使い方を明らかにしていきます。

加えて、機能定義図で定義した機能について、記述するユースケース記述を作成し、ロバストネス分析を行って、その後の分析や設計に繋げていくこともあります。また、画面遷移図で定義した画面に対して簡単なイメージを表す画面設計書を作成することもあります。

ユースケース分析における機能定義図

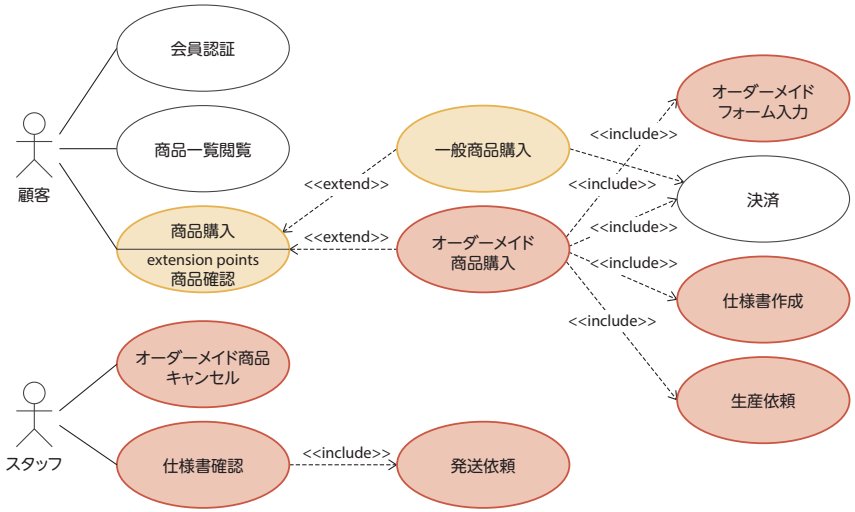
ユースケース分析における機能定義図を、ユースケース図（5-01参照）を用いて表現します。ここでは、業務概要図を参考にシステムの利用者と扱う機能について定義していきます。機能定義図から、機能の一覧と今回の開発範囲を確認できます。ここでは、オーダーメイドに関わる12の機能を抽出しました。その内、オーダーメイド商品購入などの7つ機能が新規開発対象（図中赤色）です。商品購入の機能は、新たに商品確認の拡張が行われ、一般商品購入と切り分けられるため、改修対象（図中黄色）です。それ以外の機能は従来のECサイトの機能を使えるため開発の範囲外です。

新たに追加された商品購入機能の拡張ポイントである商品確認時に扱う機能が分かれます。業務フロー図にもあったように、購入するのが一般商品だけであった場合は従来の処理の流れとなる一般商品購入の機能を利用しますが、購入する商品にオーダーメイド商品が含まれていた場合、オーダーメイド商品購入の機能を利用します。新たに追加されたオーダーメイド商品購入の機能は、オーダーメイドフォーム入力の機能と決済の機能と仕様書作成の機能と生産依頼の機能を含みます。

決済の機能は従来の一般商品を購入した場合にも用いられていたものを採用します。一般商品購入の機能と決済の機能は、処理は従来のものと同様ですが、従来は商品購入の機能の一部であったため、それぞれ一つの機能として独立させる改修が必要となります。

このような機能の一覧をユースケース図で表します。

■オーダーメイド業務システムの機能定義書



ユースケース分析におけるオブジェクト状態図

ユースケース分析におけるオブジェクト状態遷移図を、ステートマシン図（5-06参照）を用いて表現します。ここでは、業務フロー図を参考にオーダーメイド商品の状態を図で表していきます。オブジェクト状態遷移図から、オー

02 ソフトウェアアーキテクチャ設計

ソフトウェアアーキテクチャ設計では、実装するソフトウェアの構成を設計します。ここでは、仮想のシステム開発の例で設計方法を学んでいきましょう。

ソフトウェアアーキテクチャ設計とは

ソフトウェアアーキテクチャ設計とは、**システムを構成するソフトウェアの要素の全体像及びその関係性などを策定していく設計作業**です。具体的なクラスの設計などの前に、コンポーネントレベルでシステムのソフトウェアの構成や実行の大まかな流れなどを設計していきます。

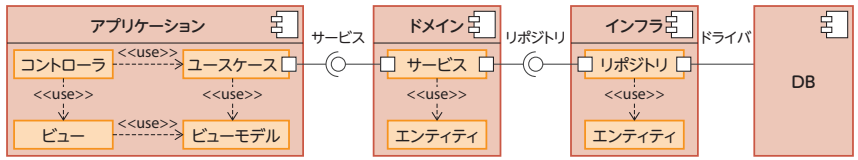
ソフトウェアアーキテクチャ設計を行わずに、クラスなどの具体的な設計に入ってしまうと、似たような機能を持ったクラスを複数作成してしまうこと、クラスの設計方法にばらつきが出てしまうこと、依存関係が複雑になってしまうことなどが起こりえます。ソフトウェアアーキテクチャ設計によって全体の構成を先に作ることで、無駄なく品質の高いソフトウェアを開発できます。

ソフトウェアアーキテクチャ設計における基本コンポーネント図

アーキテクチャ設計ではコンポーネント図（4-08参照）を用いてソフトウェアの構成及びインターフェースを表現します。

ここでは、Webアプリケーションの開発でも用いられるドメイン駆動設計というソフトウェアの設計技法に基づいて、ASP.NETのフレームワークを用いてUMLスポーツのシステムのコンポーネントの設計をコンポーネント図で示します。

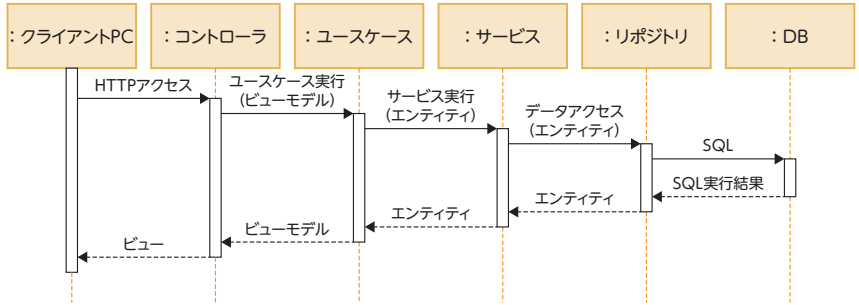
■基本コンポーネント図によってシステムのソフトウェア構成を表現



ソフトウェアアーキテクチャ設計における基本シーケンス図

アーキテクチャ設計ではシーケンス図（6-03参照）を用いて、複数の機能に共通して実行される処理の流れなどを表現します。基本コンポーネント図を元にライフラインを作成し、処理の基本的な流れを定義していきます。

■基本シーケンス図によってシステムの基本的な処理の流れを表現



まとめ

- ソフトウェアアーキテクチャ設計はソフトウェアを構成する要素の全体像を設計する
- ソフトウェアアーキテクチャ設計は既存の設計技法やフレームワークを用いて行うことがある