特集2

システム開発の設計・仕様を図示

統一モデリング言語

UMLを使い始めてみよう

UMLは、システム開発の全工程をカバーする表現力を持つ表記法で、ISO（国際標準化機構）やJIS（日本産業規格）で規格化されています。図の種類は多いのですが、よく使われる2割の表記要素で8割のことは表現できると言われています。まずはUMLを知って、今日から使い始めてみましょう。

オージス総研　森 三貴

UMLとは、Unified Modeling Languageの略称で、表記法の一種です。UMLの日本語名称は「統一モデリング言語」ですが、あまりそう呼称している人を見かけたことはありません。「ユー・エム・エル」と呼ぶのが一般的でしょう。

1980年代から1990年代にかけて、いくつものオブジェクト指向の開発方法論が提唱されました。UMLは、Booch法やOMT（Object Modeling Technique）法やOOSE（Object Oriented Software Engineering）法といったオブジェクト指向の開発方法論の表記法を統一したものです。

バージョン

現在、UMLの仕様は、OMG（Object Management Group）というIT技術の標準化を推進する非営利団体によって策定されています。最新バージョンは「UML2.5.1」です。ISO（国際標準化機構）やJIS（日本産業規格）においては、UML1.4.2が「ISO/IEC

表1　図の種類

図の名称		説明
構造図	クラス図	クラスの仕様（属性や操作など）やクラス間の関係を示す
	オブジェクト図	オブジェクトの持つ値やオブジェクト間の関係を示す
	パッケージ図	クラスなどのモデル要素を分類するパッケージ内容やパッケージ間の関係を示す
	コンポーネント図	コンポーネントが持つインタフェースやコンポーネント間の関係を示す
	配置図	システムの物理的な構成を示す
	コンポジット構造図*1-1	クラスやコンポーネントの内部構造などを示す
	プロファイル図*1-1	UMLを拡張（他の図に適用）するためのステレオタイプやプロファイルを定義する
振る舞い図	アクティビティ図	業務フローやフローチャートといったアクション（作業や処理）の流れを示す
	ステートマシン図*1-2	一つのオブジェクトやシステム全体などの状態や状態遷移を示す
	ユースケース図	システムが提供するサービス（機能）を示す
相互作用図	シーケンス図	オブジェクトなどの相互作用（メッセージのやり取り）を時系列に示す
	コミュニケーション図*1-3	オブジェクトなどの相互作用（メッセージのやり取り）を関係に着目して示す
	相互作用概要図*1-1	シーケンス図やコミュニケーション図といった相互作用図の流れを示す
	タイミング図*1-1	状態遷移のタイミングやメッセージ（相互作用）との対応などを時間軸で示す

*1-1　UML2.xで追加された。
*1-2　UML1.xでは「ステートチャート図」と呼称されていた。
*1-3　UML1.xでは「コラボレーション図」と呼称されていた。

19501:2005」および「JIS X 4170:2009」にて、UML2.4.1が「ISO/IEC 19505-1:2012」と「ISO/IEC 19505-2:2012」にて規格化されています。

これからUMLを学び使っていく場合、大まかにUML2.0以降の表記を知っていれば、十分に活用できるようになります。というのも、ちまたのモデリングツールや認定試験などもUML2.0以降の表記をベースにしたものが多いからです。

図の種類

UMLでは、業務分析から要求定義、設計、実装までの幅広い工程をカバーする表現力を持ち、視点の異なる図を**表1**のように用意しています。

UMLの図は「構造図」と「振る舞い図」に大別されます。振る舞い図の中でも相互作用に関するものは「相互作用図」に分類されます。なお、「プロファイル図」は、UMLそのものを拡張するときに使うものなので、UMLの図として紹介されないことが多々あります。

UMLを活用する効果

UMLを活用する効果は、主に次の二つです。

1. 複雑なものを簡潔なモデルとして整理・可視化できる
2. 標準化された表記法を使うことで他者と情報共有がしやすくなる

1.は、いわゆるモデリングの効果です。複雑かつ大規模なシステムを、簡潔なモデルとして整理・可視化することで、次のような効果が得られます。

・複雑さを軽減できる
・分析や設計など開発の早い段階で検証しやすくなる
・変更の影響範囲を局所化しやすくなる
・部品化（再利用）しやすくなる

2.は、UMLでモデリングする効果です。独自の表記ルールで可視化した場合は、凡例を用意するなどして図の読み方を説明しないとモデル（可視化したもの）の内容を理解できません。ですが、UMLは前述したようにISOやJISで標準の表記法として規格化されています。そのため、UMLで可視化すれば表記ルールについての説明をしなくともモデルの内容を理解できる人が多く存在します。そして、表記要素（点線の矢印など）の意味がUMLの仕様として定義されているので、文章よりも簡潔明瞭に表現できます。

各図の紹介

それでは、各図の概要を紹介します。よく使われる図については、表記ルールについても少し紹介します。

ユースケース図

ユースケース図は、システムが提供するサービス（機能）を示すものです。**図1**では、「チケットを予約する」「予約をキャンセルする」「チケットを購入する」というサービスを会員が使えることを表現しています。

ユースケース図の表記ルールについて少し紹介します。表記に使う要素（以降、モデル要素）で、主要なものは「アクター」「ユースケース」「関連」です。

■アクター

アクターは、システムの外部要素を表現するもので、人型のアイコンで表記します。したがって、次に示すものがアクターとなり得ます。

・システムの利用者
・外部システム（開発対象外の関連するシステム）

なお、組み込みシステムのユースケース図では、ソフトウエアの機能を明らかにするために、アクターとしてセンサーやボタンといった「ハードウエア」を抽出することがあります。その場合、「システムの利用者」などのシステムの外部要素であるアクターを「主アクター」、「ハードウエア」相当のアクターを「副アクター」と呼ぶことがあります。

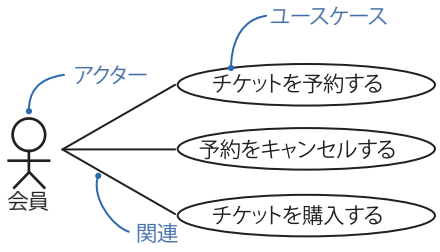
■ユースケース

ユースケースは、システムが提供するサービス（機能）を表現するもので、だ円で表記します。

■関連

関連は、アクターとユースケースをひも付ける実線です。アクターは、いずれかのユースケースと必ず関連でひも付けられ

図1　ユースケース図の表記ルール



ます。ユースケースもいずれかのアクターと必ず関連でひも付きます。アクターがシステムの利用者の場合は、どのユースケースを利用できるのかを表現します。アクターが開発対象外の関連システム（既存システム）の場合は、どのユースケースが外部システムと連携するのかを表現します。

◆クラス図◆

クラス図は、クラスの仕様（属性や操作など）やクラス間の関係を示すものです。**図2**では、「会員」「予約」「座席」というクラスがあり、「予約」クラスは「予約番号」や「予約日」といった属性と「予約をキャンセルする」という操作を持ち、1回の予約で会員が「1」つ以上の座席を予約できることを表現しています。

分析レベルのクラス図の代表的なモデル要素

クラス図のモデル要素（表記ルール）についても少し紹介します。図2に登場する主要なモデル要素は「クラス」「関連」です。図2は、いわゆる分析レベルのクラス図です。分析レベルのクラス図では、問題領域（システムなど）で扱う概念（もの、こと）をクラスとして抽出することもあり、概念モデルと称される場合もあります。

分析レベルのクラス図で使うモデル要素は、設計レベルのクラス図でも使います。よって、分析レベルのクラス図で使う代表的なモデル要素を紹介した後に、設計レベルのクラス図で使う代表的なモデル要素についても紹介します。

■クラス

クラスは、長方形で表現します。クラスを「3」つの区画に分ける場合、上の区画に「クラス名」、真ん中の区画に「属性」、下の区画に「操作」を書きます。属性や操作の区画は省略できるため、クラス名のみの「1」つの区画でクラスを表現することもあります。

属性

属性は、次のような書式で記述します。分析の初期段階では、型を省略することもあります。

```
属性名：型
```

なお、型の記載例を**図3**に示しました。

操作

操作は、次のような書式で記述します。分析の初期段階では、引数や型を省略することもあります。

```
操作名(引数名 ： 型, 引数名 ： 型) ： 戻り値の型
```

こちらの型の記載例も図3に示しています。

■関連

クラス図での関連は、クラスとクラスの間に実線を引いて表現し、「関連の両端にあるクラスのオブジェクト間に永続的な関係がある」ことを意味します。オブジェクトとは、クラスの実体（インスタンス）のことです。

そして、関連の両端には、「多重度」や「関連端名」を付加できます。分析レベルのクラス図では、関連端名を省略することが多々ありますが、多重度は書くことが多いです。

関連端名

関連端名は、あるクラスから見た、関連先のクラスの役割や立場を表します。例えば、図2では、「予約」クラスから見た「座席」クラスは「予約した席」であることを表しています。

多重度

多重度は、あるクラスの「1」つのオブジェクトから見て、関連先のクラスのオブジェクトがひも付く可能性がある数のことです。次のような書式で記述します。

```
下限値..上限値
```

例えば、ひも付くオブジェクト数の下限値が「0」で上限値が「1」の場合は「0..1」と書きます。また、「＊」は複数を意味します。「1..＊」と書いてある多重度は、下限値が「1」で上限値が複数なので、「1」以上という意味になります。そして、「0」以上の「0..＊」は「＊」と省略して書いてよいルールとなっています。

図2では、「予約」クラスと「座席」クラスの間にある関連にはクラス「座席」側に「1..＊」という多重度が書いてありますが、これは、ある「1」つの「予約」オブジェクトにひも付く「座席」オブジェクトの数が「1」以上を表しています。以上、多重度の書き方を表にまとめると**表2**のようになります。

図2　クラス図の表記ルール（その1）

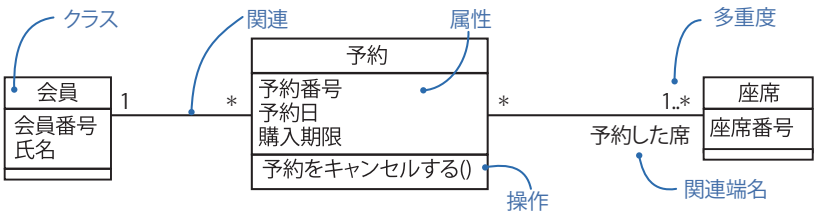


表2　多重度の書き方

多重度	意味
1	必ず1
0..1	0～1
＊または 0..＊	0以上
1..＊	1以上

図3　クラス図の表記ルール（その2）

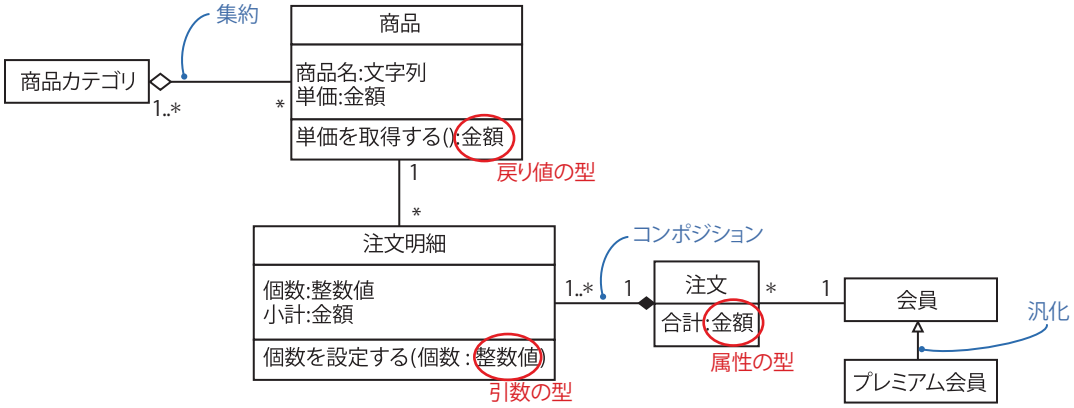
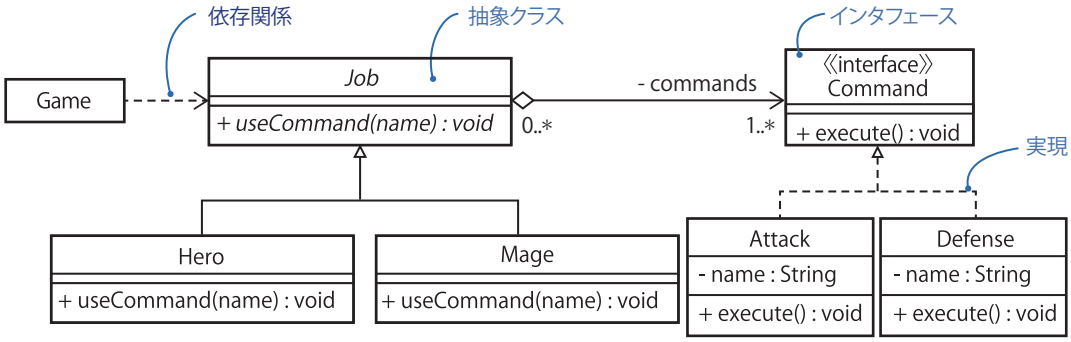


図4　クラス図の表記ルール（その3）



分析レベルのクラス図で使うモデル要素を、さらにもう少し紹介します。図3で新たに登場した主なモデル要素は「集約」「コンポジション」「汎化」です。

■集約

集約は、関連の一種です。よって、関連にできること（関連端に多重度を書くなど）は同じように可能です。関連との違いは、関連の両端にある片方のクラスが「全体」で、もう片方が「部分」という意味を持つことです。

全体側の関連端に白抜きのひし形を付加して表現します。例えば、先ほどの図3では「商品カテゴリ」クラスと「商品」クラスは全体と部分という関係があります。「商品カテゴリ」クラスが全体、「商品」クラスが部分に相当します。

■コンポジション

コンポジションは、集約の一種です。したがって、コンポジションは、集約と同様に関連にできること（関連端に多重度を書くなど）は同じように可能です。そして、ひし形が付いている方が全体という意味を持ちます。

集約との違いは、「全体側のクラスのオブジェクトが部分側のクラスのオブジェクトを占有するという意味を持つ」ということです。全体側の関連端に黒く塗り潰したひし形を付加して表現します。例えば、図3では「注文」クラスのオブジェクトが「注文明細」クラスのオブジェクトを占有する（同時に別の「注文」オブジェクトと共有されることがない）ことを表しています。

■汎化

汎化は、オブジェクト指向の基本概念である「継承」を表現するUMLのモデル要素名です。継承とは、一般的な要素（スーパークラス）と特殊な要素（サブクラス）の分類関係のことです。サブクラスは、スーパークラスの性質（属性、操作、関連）を引き継ぎ、サブクラスに特化した性質を追加できます。表記は、実線と白抜きの三角形を使い、スーパークラス側に三角形を付加します。

例えば、図3では、「会員」がスーパークラス（「プレミアム会員」より一般的）、「プレミアム会員」がサブクラス（「会員」より特殊）という関係があり、「プレミアム会員」は「会員」の性質を引継ぎ、「プレミアム会員」に特化した性質を追加できることを表しています。

設計レベルのクラス図の代表的なモデル要素

ここから、設計レベルのクラス図で使うモデル要素を少し紹介します。**図4**に登場する主要なモデル要素は「依存関係」「抽象クラス」「インタフェース」「実現」です。また、属性や操作に付加できる「可視性」、関連に付加できる「誘導可能性」についても紹介します。

可視性・誘導可能性

設計レベルのクラス図では、実装につなげられるように分析レベルのクラス図より詳細な情報を属性・操作や関連に追加します。

可視性は、属性や操作の先頭に「+」や「-」を付加しているもので、「+」がpublic（すべてのクラスに公開。アクセス可）、「-」がprivate（そのクラス以外には非公開。他のクラスはアクセス不可）を表しています。可視性の種類はほかにもありますが、まずは、この2種類のみ紹介します。

誘導可能性は、関連（集約やコンポジションを含む）の関連端に付加するものです。表記は、誘導可能な方向に矢印を付加します。誘導可能とは、「一方が他方を参照できる」といった意味であり、厳密に言うと「一方が他方を属性として保持する」を意味します。例えば、p.27の図4では「Job」と「Command」の間に集約があります。「Command」側に矢印が付加されているので、「Job」は「Command」を属性として保持することを表しています。ここで言う「属性として保持する」については、依存関係を紹介した後に少し補足します。

■依存関係

依存関係は、クラスとクラスの間に破線矢印を引いて表現し、「一方が他方を一時的に利用する」（属性や操作などにアクセスする）ことを意味します。例えば、p.27の図4では「Game」が「Job」を一時的に利用することを表しています。

なぜ、これを「依存関係」と呼ぶのかといえば、利用する側は利用される側の属性や操作にアクセスしたり、利用される側に変更があればその変更の影響を受けたりするなど、利用される側に何かと依存する関係だからです。

ここで、先ほどの「属性として保持する」を補足します。誘導可能性を付加した関連は、属性を別の形式で表現したものとして捉えることもできます。そして、誘導可能性の矢印が付加されている側を属性として保持する意味になります。例えば、**図5**では「Command」と「Job」の間にある集約に誘導可能性が付加されています。そこにある関連端名の「commands」は属性名であり、関連端名の前にある「-」は属性の可視性です。

なお、関連で表現すると、多重度の上限が「1」より大きい場合、型について複数オブジェクトを格納できるもの（配列やコレクションクラスなど）であることまでしか分かりません。型は、実装で決めればよい場合もありますが、設計で決めた方がよければ図5のように「ノート」で補足を入れます。ノートは、すべての図において、どのモデル要素にも付加できるものです。補足を入れたいときに使います。

ちなみに、関連と属性のどちらの表現を使うのがよいのかは、ケースバイケースです。ただ、ユーザー定義の型（p.27の図4の「Command」など）は関連で示し、データ型（図4の「String」など）は属性で示すことが多いです。

■抽象クラス

抽象クラスは、クラスの一種ですが、次の点が異なります。

・直接インスタンス化できない
・操作の仕様（操作名、引数、戻り値の型）だけを定めた「抽象操作」を持てる

抽象クラスの表記は、クラス名を斜体（イタリック体）にして表現します。また、抽象操作は、操作名など操作全体を斜体（イタリック体）にします。**図6**のように、抽象クラスや抽象操作は、斜体（イタリック体）で表現せずに「{abstract}」を付加して表現することもできます。

なお、抽象クラスは、直接インスタンス化できないため、サブクラスが必要となります。そのため、抽象クラスのサブクラスのことを「具象クラス」と称することがあります。

抽象クラスは、抽象的な概念であることを示すために、分析レベルのクラス図で使うこともあります。しかし、オブジェクト指向の基本概念である「ポリモーフィズム」を実現する仕組みの一つでもあることから、設計レベルのクラス図で使うことの方が多いという印象です。

例えば、p.27の図4の「Job」が抽象クラスです。職業（Job）そのものの実体は存在しないけれども、職業（Job）の具体的な実体として勇者（Hero）や魔法使い（Mage）が存在することを表しています。また、職業（Job）は抽象操作「useCommand(name)」を持ち、その具体的な実装は勇者（Hero）と魔法使い

図5　関連は属性の別表現

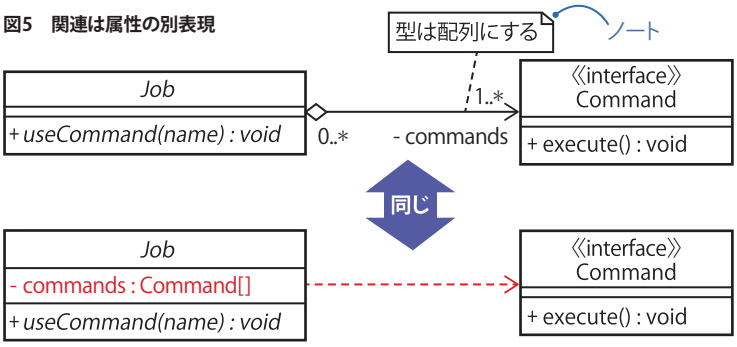
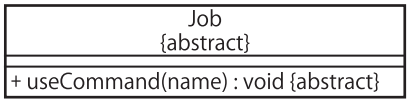


図6　抽象クラス表記のバリエーション



(Mage)がそれぞれ持つことでポリモーフィズムを実現していることを表しています。

■インタフェース、実現

インタフェースは、外部から利用するために公開している仕様です。ポリモーフィズムを実現する仕組みの一つでもあります。インタフェースには次のような特徴があります。

・直接インスタンス化できない
・操作は仕様(操作名、引数、戻り値の型)だけ持てる(実装を持たない)

インタフェースは、長方形で表現し、インタフェース名の上に「《interface》」を付加します。インタフェースはクラスとは別物です。また、インタフェースは実装を持たないため、インタフェースに定義されているすべての操作を実装するクラス(インタフェースを実現するクラス、実現クラス)を用意する必要があります。インタフェースと実現クラスは「実現」という関係で示します。「実現」は、白抜きの三角形と点線で表現し、インタフェース側に三角形を置きます。

例えば、p.27の図4の「Command」がインタフェースです。コマンド(Command)そのものの実体は存在しないけれども、勇者(Hero)や魔法使い(Mage)が持つ具体的なコマンド(Command)として攻撃(Attack)や防御(Defense)があることを表しています。そして、コマンド(Command)は抽象操作「execute()」を持ち、その具体的な実装は攻撃(Attack)と防御(Defense)がそれぞれ持つことでポリモーフィズムを実現していることを表しています。

なお、UMLの仕様では「インタフェースが持てる操作は仕様(実装を持たない抽象操作)のみ」です。したがって、UMLの表記ルールとしては、インタフェースが持つ操作を斜体(イタリック体)にしません。しかし、モデリングツールがインタフェースの操作を斜体(イタリック体)にすることがあります。

それから「インタフェースに属性を定義できるか否か」については、UML1.xでは「インタフェースは属性を定義できない」でしたが、UML2.xから「インタフェースは属性を定義できる」に変わりました。いずれにしても、UMLは特定のプログラミング言語に依存しないため、UMLとプログラミング言語のインタフェースでは定義が異なります。加えて、UMLの可視性とプログラミング言語のアクセス修飾子など、ほかにも異なることがあります。

したがって、システム開発の設計においてUMLを使う場合、次の二つがポイントとなります。

・UMLとプログラミング言語との対応(マッピング)ルールを設ける
・モデリングツールの表記がUMLの仕様と多少異なっていても気にしない

モデリングツールは、設計のクラス図からソースコードを自動生成する機能を持っているものを使えば、開発効率が上がります。

◆オブジェクト図◆

オブジェクト図は、オブジェクトの持つ値やオブジェクト間の関係を示すものです。**図7**では、「座席」クラスの具体的な実体として「E7」～「E9」といったオブジェクトがあり、山田さんが「E7」と「E8」の座席を予約していることを表現しています。

■オブジェクト

オブジェクトは、長方形で表現します。長方形内に次の書式で名前を記述し、名前に下線を付けます。

```
オブジェクト名:クラス名
```

オブジェクト名とクラス名は、それぞれどちらか一方を省略できます。よって、オブジェクトの名前表記は次のようなバリエーションがあります(**図8**)。

・オブジェクト名のみ
・オブジェクト名とクラス名
・クラス名のみ

オブジェクト名のみなのか、クラス名のみなのかを識別するには「:」の有無に着目します。「:」があればクラス名のみで、「:」がなければオブジェクト名のみだと解釈できます。

また、オブジェクトとクラスは、ともに長方形で表現します。

図7　オブジェクト図の表記ルール

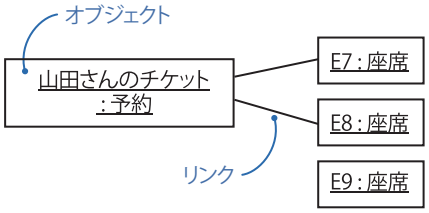
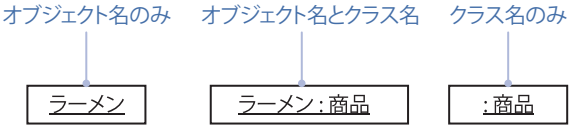


図8　名前表記のバリエーション



オブジェクトとクラスを見分けるポイントは、名前に下線が付いている（オブジェクト）か否か（クラス）です。

■リンク

リンクは、実線で表現し、オブジェクト間に何らかの関係があることを意味します。リンクを用いて、クラス図の多重度を検討できます。

例えば、**図9**を使って、オブジェクト図のリンクからクラス図の多重度を導く例を説明します。図9のオブジェクト図からは部署から見た社員について次のことが読み取れます。

・開発Tという部署から見て社員は上田、中田、下田のリンクが「3」つ
・新規の部署にひも付く社員のリンクがない

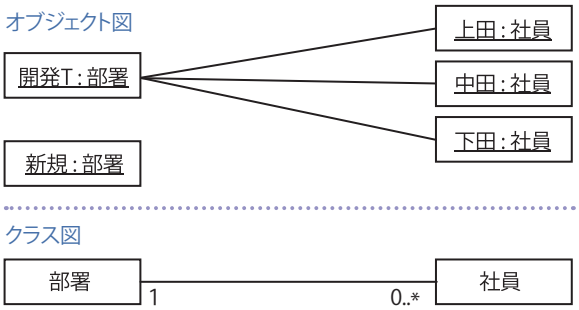
したがって、部署から見た社員がひも付く数の下限値は「0」、上限値は「3」です。オブジェクト図と完全に一致させるなら多重度は「0..3」となります。なお、部署から見た社員の上限を特に定めていない場合は、図9のクラス図のように部署から見た社員の多重度は「0..*」です。

また、図9のオブジェクト図から、社員から見た部署については、次のことが読み取れます。

・上田、中田、下田いずれの社員もひも付く部署は開発Tへのリンク「1」つのみ

したがって、社員から見た部署がひも付く数は必ず「1」です。図9のクラス図のように社員から見た部署の多重度は「1」となります。

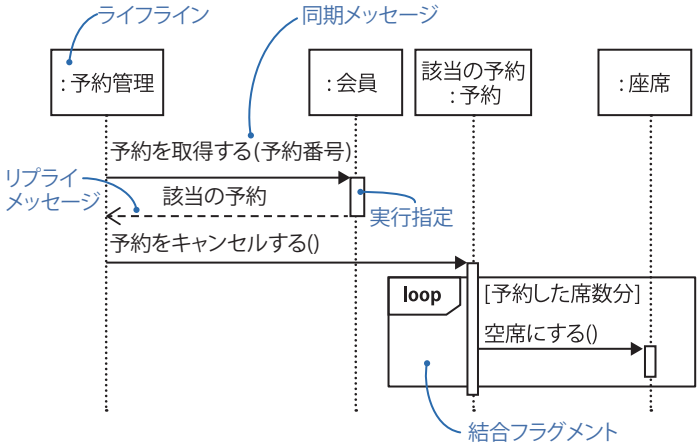
図9　オブジェクト図のリンクとクラス図の多重度



◆シーケンス図◆

シーケンス図は、オブジェクトなどの相互作用（メッセージのやりとり）を時系列に示すものです。**図10**は、まず「予約管理」から「会員」に「予約を取得する」というメッセージを送ると「会員」から「予約管理」に「該当の予約」が返ってきます。次に「予約管理」は返ってきた「該当の予約」に「予約をキャンセルする」というメッセージを送るといった相互作用があることを表現しています。

図10　シーケンス図の表記ルール



■ライフライン

ライフラインは、相互作用に参加している要素（クラスやアクターの実体など）を表現するものです。各要素とそこから下に伸びる点線を含めてライフラインと呼びます。点線が下に伸びている間は、その要素が存在していることを意味します。

ライフラインの名前は、相互作用に参加している要素がクラスの実体の場合、長方形内に「オブジェクト名:クラス名」という書式で書きます。オブジェクト名とクラス名は、オブジェクト図と同じように、それぞれどちらか一方を省略できます。

■メッセージ

メッセージには同期と非同期があります。同期メッセージは、メッセージを送る側から受ける側へ先端を塗り潰した矢印を引いて表現します。同期メッセージは、送る側が送ったメッセージの完了を待って次のメッセージを送信することを意味します。

UML2.xでは、メッセージを次の書式で記述します。

```
メッセージ名(引数…)
```

UML1.x では、次のようにメッセージに戻り値の型を表現できました。そして、UML2.x対応のモデリングツールでも表現できることがあるため、モデリングの現場ではメッセージに戻り値の型を表記することが多いです。

メッセージ名(引数…):戻り値の型

同期メッセージには、反対向きの破線矢印を引いたリプライメッセージを付加できます。リプライメッセージは、メッセージが完了したことや処理の結果を返すことを意味します。

リプライメッセージの記述は任意なので(必須ではないので)、「処理の結果を返さない場合(戻り値の型がvoidなど)はリプライメッセージを省略する」「何らかの戻り値がある場合にリプライメッセージを書く」といったスタイルガイドラインを設けると、読みやすいシーケンス図になります。

なお、UML2.xから、リプライメッセージの書式が煩雑になったため、モデリングの現場では簡略化して、戻り値の型のみや図10のように「該当の予約」といった具体的な戻り値のみを書くことが多々あります。

■実行指定

ライフライン上にある長方形を「実行指定」と呼びます。実行指定は、ライフラインがメッセージを受け取り、それを処理している期間を表すものです。例えば、図10で「:会員」というライフライン上にある実行指定は同期メッセージ「予約を取得する(予約番号)」が処理している期間がリプライメッセージまでであることを示しています。

また、実行指定の記述は任意ですが、実行指定を記述することで、その処理の中で別のメッセージを送っていることを明確に示せます。例えば、図10で「該当の予約:予約」というライフライン上にある実行指定は同期メッセージの「予約をキャンセルする()」の中で同期メッセージの「空席にする()」を送っていることを示しています。

■結合フラグメント

結合フラグメントは、繰り返しや条件分岐といった制御構造を表すものです。制御作用がある部分を枠で囲み、枠内の左上に制御構造の種類(図10の「loop」など)を書きます。loopは、与えられた条件を満たす間だけ繰り返し実行することを示します。

繰り返しの条件を記載する方法には複数あります。ただし、次の書式で「論理式がtrueの間だけ処理を繰り返す」ということを表現できることや、他の図などにおいても条件を書く表記に「[]」が使われていることから、モデリングの現場では[]だけを使って繰り返しの条件を記載することが多いといえます。

図11　結合フラグメントのopt、altと相互作用使用のref

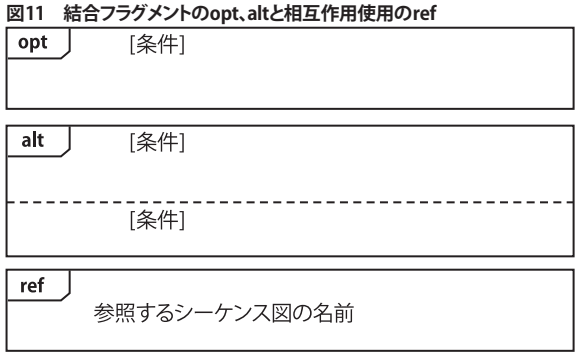
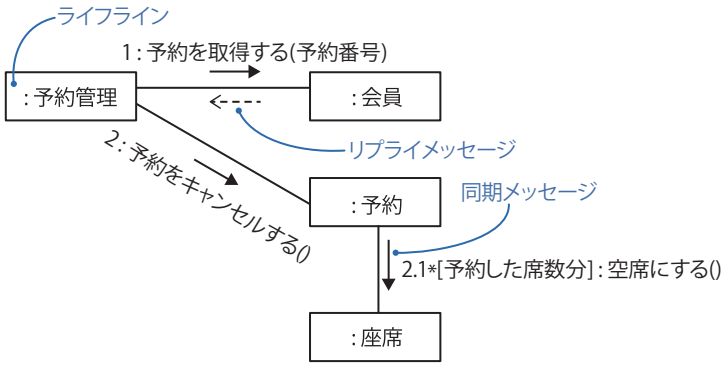


図12　コミュニケーション図の表記ルール



[論理式]

制御構造の種類は、loopのほかに「opt」や「alt」もよく使います。optは、特定の条件を満たす場合のみ実行することを示しています。[]内に条件を記載します。altは、複数の条件分岐を示すものです。**図11**のように分岐する条件ごとに点線で区切り、[]内に条件を記載します。

なお、結合フラグメントと同じ枠を使って、制御構造の種類を書く場所に「ref」と書いた場合は、別のシーケンス図を参照することを意味する「相互作用使用」と呼ぶものになります*1。図11のように、枠内に参照するシーケンス図の名前を書くことで、そのシーケンス図を参照することを示せます。

◆コミュニケーション図◆

コミュニケーション図は、オブジェクトなどの相互作用(メッセージのやり取り)を関係に着目して示すものです。**図12**は、図10のシーケンス図と同じ内容をコミュニケーション図で表現しています。

コミュニケーション図で表現できることは、すべてシーケンス図でも表現できます。設計ではソースコードとの対応をイ

*1　結合フラグメントと相互作用使用をまとめて「相互作用フラグメント」と呼びます。

メージできた方がよいし、より詳細に表現できるため、シーケンス図を使うことが多いです。一方で、コミュニケーション図は、オブジェクトやクラスの責務を明確にしたい場面で使用します。例えば、分析では、オブジェクト図やクラス図と見比べながら責務を明確にしたい（メッセージの向きや集中具合で責務を確認できる）ため、それらと配置が類似しているコミュニケーション図を使うことが多いといえます。

■ライフライン

コミュニケーション図のライフラインは、長方形のみ（破線なし）で表現します。

■メッセージ

メッセージのやり取りがあるライフライン間に実線を引き、その実線沿いにメッセージを書きます。同期メッセージは、シーケンス図と同じ先端を塗り潰した矢印で表現します。

メッセージの順序（シーケンス番号）を、メッセージ名の前にコロン（:）を付加して記述します。メッセージ呼び出しに階層構造がある場合は、「1.1」「1.1.1」のように、ピリオド（.）を使ってメッセージの入れ子を表現できます。例えば、図12では「予約管理」はメッセージ「予約を取得する（予約番号）」の後にメッセージ「予約をキャンセルする()」を送っていること、「予約」はメッセージ「予約をキャンセルする()」の中でメッセージ「空席にする()」を送っていることを示しています。

メッセージ実行に条件があれば、コロン（:）の前に「[条件]」のように記述します。また、メッセージの繰り返しは、シーケンス番号の後にアスタリスク（＊）を付加して表現します。例えば、図12では、予約した席数分だけメッセージ「空席にする()」を繰り返すことを示しています。

リプライメッセージも、シーケンス図と同じ破線矢印で表現します。ただ、コミュニケーション図においては、ライフライン間のメッセージが双方向になっていない（単方向である）ことを確認する際にリプライメッセージがあると分かりにくいため、リプライメッセージを省略することが多いです。

ステートマシン図

ステートマシン図は、一つのオブジェクトやシステム全体などの状態や状態遷移を示すものです。**図13**では、次のようなことを表現しています。

- **「空席あり」「満席」という状態がある**
- **「空席あり」のときは、ずっと「予約を受け付ける」という動作を継続して行う**
- **「空席あり」のときに「予約」というイベントが発生すると、「残席の数=1」の場合は「残席の数を1減らす」という動作をしてから「満席」という状態になる**
- **「満席」という状態なったら最初に「予約を締め切る」という動作を行う**
- **「満席」のときに「キャンセル」というイベントが発生すると、「予約を再開する」という動作をしてから「残席の数を1増やす」という動作をした後に「空席あり」の状態になる**

■状態、開始疑似状態

状態は、角の丸い長方形で表現し、着目しているオブジェクトやシステム全体が取り得る一つの状況を表します。例えば、図13では「空席あり」と「満席」という状態があることを表しています。

開始疑似状態は黒丸で表現し、初期状態を示すものです。図13では、初期状態が「空席あり」であることを表しています。

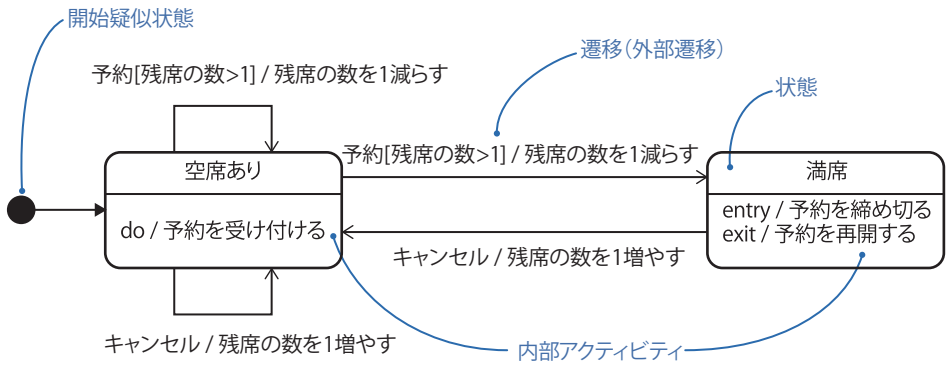
■遷移（外部遷移）

遷移（外部遷移）は、実線の矢印で表現し、ある状態から別の状態に変化することを表します。遷移には、次の書式で情報を付加します。

トリガ［ガード条件］/ エフェクト

トリガは、ある状態から別の状態に移るきっかけ（事象、イベ

図13　ステートマシン図の表記ルール



ント）のことです。開始疑似状態からの遷移以外はトリガが必要です。図13では「空席あり」の状態から「満席」の状態へと遷移するきっかけとなるイベントは「予約」であることを表しています。

ガード条件は、トリガが発生した際に状態遷移するための条件がある場合に記述します。図13では、「空席あり」の状態において「予約」のイベントが発生した際に「残席の数＝1」の場合は「満席」の状態に遷移することが決まることを表しています。

エフェクトは、状態が遷移するとき、その遷移とともに実行する動作がある場合に記述します。図13では、「空席あり」から「満席」へ遷移することが決まった後、「空席あり」状態を出てから「満席」状態に入るまでの間に、「残席の数を1減らす」といった動作を行うことを表しています。

■内部アクティビティ

内部アクティビティは、その状態にあるときに実行する動作です。次の3種類があります。

- **入場動作**
- **退場動作**
- **実行アクティビティ**

入場動作は、その状態に入ったときに必ず実行する動作です。次の書式で記述します。例えば、図13では、「満席」状態に入ったら最初に「予約を締め切る」動作を行うことを表しています。

```
entry/動作名
```

退場動作は、その状態から出ていくときに必ず実行する動作です。次の書式で記述します。図13では、「満席」状態から出ていく直前に「予約を再開する」動作を行うことを表しています。

```
exit/動作名
```

実行アクティビティは、その状態の間ずっと継続して実行する動作です。次の書式で記載します。図13では、状態「空席あり」の間ずっと「予約を受け付ける」動作を行うことを表しています。

```
do/動作名
```

◆アクティビティ図◆

アクティビティ図は、業務フローやフローチャートといったアクション（作業や処理）の流れを示すものです。**図14**は、空席を照会し、空席があればチケットを予約し、購入期限内であればチケットを購入することを表現しています。

アクションは、作業や処理を構成する1単位に相当するものです。角の丸い長方形で表現します。例えば、図14では、「空席を照会する」「チケットを予約する」といった作業があることを表しています。

開始ノードは、手順（業務や処理の流れ）の始まりを示すものです。黒丸で表現します。図14では、「空席を照会する」が最初の作業であることを表しています。

アクティビティ終了ノードは、手順（業務や処理の流れ）の終わりを示すものです。中を黒く塗り潰した2重丸で表現します。

制御フローは、アクションの順序を示すものです。矢印で表現します。

デシジョンノードは、作業や処理の流れが分岐することを示すものです。ひし形で表現します。そして、ガードは、分岐の条件を示すものです。[]内に条件を書きます。図14では、「空席を照会する」の後には分岐があり、「空席あり」の場合は「チケットを予約する」の作業に進むけれども、「満席」の場合は作業が終了することを表しています。なお、分岐した流れを合流させるときにデシジョンノードと同じひし形が使えます。そのときは合流することを示しているので「マージノード」と呼びます。

図14　アクティビティ図の表記ルール

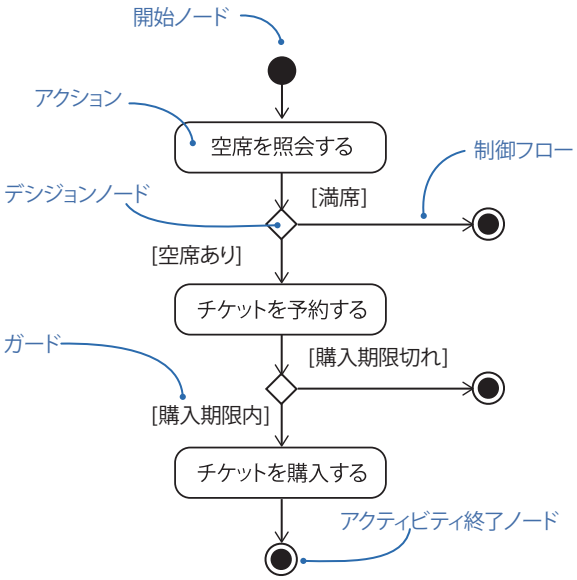


図15　パッケージ図の表記ルール

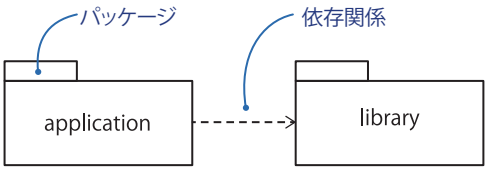


図16　コンポーネント図

図17　配置図

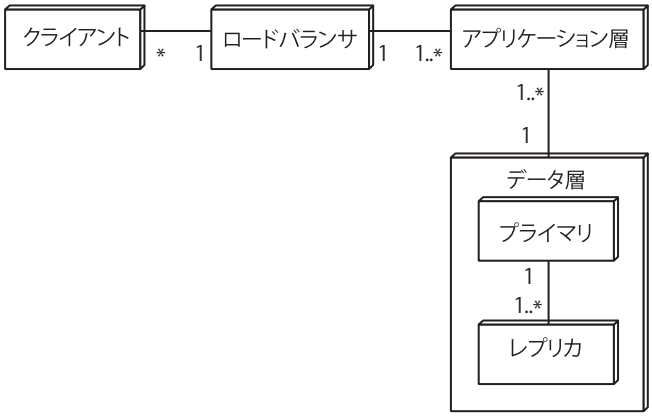
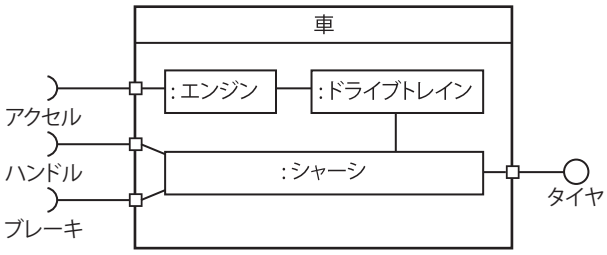


図18　コンポジット構造図



◆パッケージ図◆

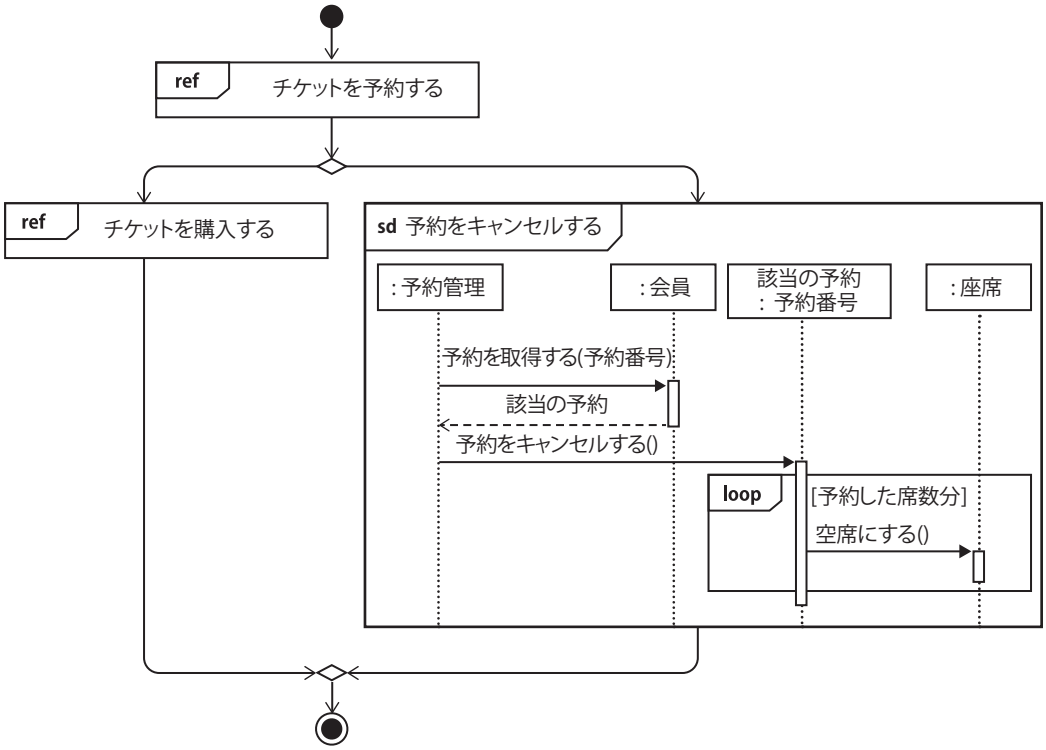
パッケージ図は、クラスなどのモデル要素を分類するパッケージ内容やパッケージ間の関係を示すものです。**図15**では、「application」「library」のパッケージがあり、applicationがlibraryを使う（libraryが変更されるとapplicationはその変更の影響を受ける）ことを表現しています。

パッケージとは、クラスなどのモデル要素を、その意味や役割に基づいてグループ化する入れ物です。規模が大きい場合は、関係性の強いモデル要素をグループ化することで、モデル全体が把握しやすくなり、メンテナンス性を高めることができます。

パッケージは、左上にタブのような小さな長方形を付加した長方形で表現します。

依存関係は、パッケージ間に破線矢印を引いて表

図19　相互作用概要図



現し、「一方が他方を利用する（利用される側に変更があれば利用する側はその変更の影響を受ける）」ことを表します。

パッケージ図にパッケージ間の依存関係を示すことにより、変更時の影響範囲を把握できます。また、システム全体のモジュール構成（ソフトウエアアーキテクチャ）を示す場合にもパッケージ図を使います。

コンポーネント図

コンポーネント図は、コンポーネントが持つインタフェースやコンポーネント間の関係を示すものです。**図16**では、「JDBCドライバ」「業務ロジック」というコンポーネントがあり、JDBCドライバが「JDBC API」というインタフェースを実現していて、業務ロジックがJDBC APIを使うことを表現しています。

配置図

配置図は、システムの物理的な構成を示すものです。**図17**では、複数のクライアントが「1」つのロードバランサに接続し、ロードバランサは複数のアプリケーション層 に接続することを表現しています。

コンポジット構造図

コンポジット構造図は、クラスやコンポーネントの内部構造などを示すものです。**図18**では、車はアクセルを踏むと「エンジン」→「ドライブトレイン」→「シャーシ」を経由してタイヤが駆動し、ハンドルを回すと「シャーシ」を経由してタイヤの向きが変わることを表現しています*2。

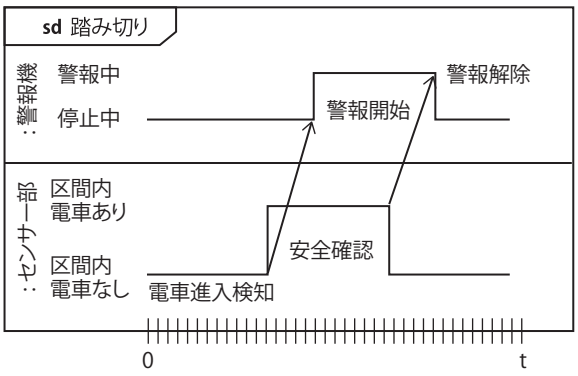
相互作用概要図

相互作用概要図は、シーケンス図やコミュニケーション図といった相互作用図の流れを示すものです。**図19**は、「チケットを予約する」という相互作用の後に「チケットを購入する」という相互作用もしくは「予約をキャンセルする」というシーケンス図の相互作用という流れが実行されることを表現しています。

タイミング図

タイミング図は、状態遷移のタイミングやメッセージ（相互作用）との対応などを時間軸で示すものです。**図20**は、踏み切りにおいて、センサー部が「区間内電車なし」状態のときに「電車侵入検知」というイベントを受信したら、警報機に「警報開始」のメッセージを投げて（警報機はイベントを受信し）、センサー部は「区間内電車あり」の状態に遷移することを表現しています。

図20　タイミング図



ここまで、各図の概要を紹介するついでに、よく使われる図では表記ルールも紹介しました。UMLの表記ルールのうち2割は紹介できたでしょう。UMLでは、よく使われる2割の表記要素で8割のことは表現できると言われています。実際にUML を使い始めてみてください。

UMLを活用するために学んだ方がよいこと

UMLを活用するために学んだ方がよいことは、次の三つです。

・表記ルール
・良いモデルの作り方
・開発におけるUMLの使い方

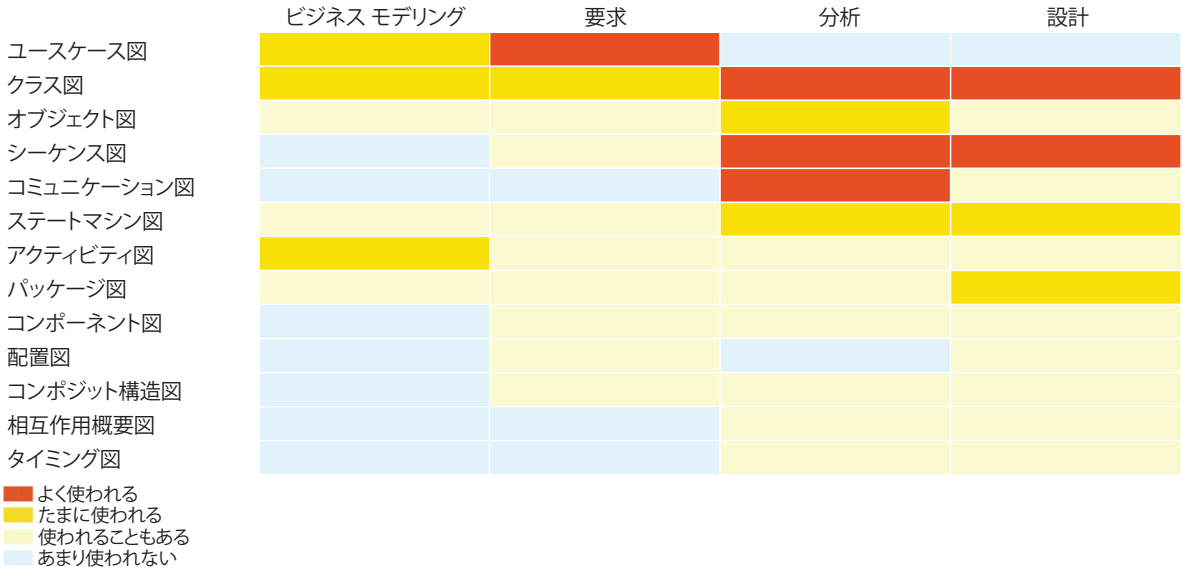
「表記ルール」は、図の表記に使う要素（モデル要素）の意味で、UMLで規定されています。まず表記ルールを知ることで、自分が作成した図を他者に理解してもらえるようになります。また、他者が作成した図を理解できるようになります。よって、表記ルールを学ぶのは、UMLを活用するための最初の一歩として肝要です。

ただ残念なことに、表記ルールに従っているだけでは「仕様変更の影響範囲を局所化しやすくなる」などの前述した「UMLを活用する効果」を得るには不十分です。前述のような効果を得るには「良いモデルの作り方」と「開発におけるUMLの使い方」が必要になってきます。そして、これらはUMLでは規定されていません。

良いモデルの作り方とは、適切にモデルとして整理するための考え方のことです。例えば、分析レベルであれば、現実世界

＊2　ここでは、車の基本構成部分から「エンジン」「ドライブトレイン」を除いた足回り部分のことを「シャーシ」と呼ぶことにしました。

図21　UMLの図がどの工程で使われているかの対応例



にあるものから不要な情報を省いて、注目したい本質のみ抽出する方法（いわゆる「適切な抽象化」）だったり、「ものーことーもの」パターンといった概念の抽出を手助けしてくれる構造パターンだったりします。

設計レベルであれば「結合度を低く」（Low Coupling）、「凝集度を高く」（High Cohesion）といった構造化プログラミングの頃から良いとされている設計の考え方であったり、分かりやすい責務を持ったオブジェクト（クラス）に部品化するといった、オブジェクト指向の考え方であったりします。また、少し話の切り口が変わりますが、分析モデルや設計モデルといった異なる工程のモデルをシームレスにつなげることも良いモデルに必要なことです。

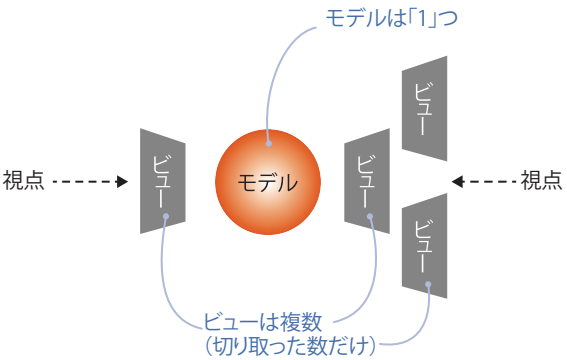
開発におけるUMLの使い方とは、開発のどの工程で、どの図を使うのか、といったことです。例えば、Javaの設計においてクラス図とシーケンス図を使う、C++の設計においてステートマシン図を使うといった事例を目にしたことがあるかもしれません。また、分析レベルのクラス図や設計レベルのクラス図といった、モデルの記述レベル（粒度）を本記事でも少し触れました。

どの図を、いつ、どう使えばよいかといったプロセスを効率良く学ぶのであれば先人の知恵を借りるとよいと思いますが、小規模であれば自分でUMLを使いながら試行錯誤するのもよいでしょう。UMLは特定の開発プロセスに依存していません。どの工程でどの図を使用するかは利用者に任されています。

参考までに、UMLの図がどの工程で使われているかの例を**図21**に示します*3。

図22は、モデルの1側面を切り取ったビューです。図を作成しながらモデルとして適切かを視点を変えながら検証する（図の整合性を保つ）ことも必要です。

図22　モデルとビュー



資格

筆者が知る限り、UMLに関する資格は、「OMG」と「UMTP」の2種類あります。

OMGは「UML 2 CERTIFICATIONS」という試験です*4。レベルは「Foundation」「Intermediate」「Advanced」の3種類です。UMLの仕様（表記ルールだけでなくメタモデルを含む）を理解していることが求められます。よって、モデリングツールを作る人やUMLを拡張したい人に向いているのですが、システム開発でUMLを使う人にはお薦めではありません。なお、2022年3月末の時点において、日本語で受験できません。

UMTPは「UMLモデリング技能認定試験」という試験です*5。レベルは「L1」～「L4」の4種類です。UMLの表記ルール・開発におけるUMLの使い方・良いモデルの作り方が問われます。よって、システム開発においてUMLを使ってモデリングする人にお薦めです。日本語でオンライン受験できます。

＊3　必ず使用する/必ず使用しないという訳ではありません。
＊4　詳しくは「https://www.omg.org/ocup-2/」を参照してください。
＊5　詳しくは「https://umtp-japan.org/about_exam」を参照してください。

表3　UMLモデリング技能認定試験のレベル（UMTP のホームページより抜粋）

レベル	内容
L1	UMLなどを使ってモデリングを行う最低限の知識を持っている
L2	UMLモデルの読み書きが普通にできる。開発範囲の一部を担当し、モデリングができ、他者のモデルの意味を理解できる
L3	実務でモデリングが実施できる。拡張性や変更容易性などの点で高品質なモデルを定義できる。特定分野の専門的な知識を備えている
L4	実践に基づいてモデリングを指導できる。L3のスキルを有し、開発プロジェクトでモデリングを一定数あるいは期間実践した経験を持つ

参考までに「L1」～「L4」の概要を**表3**に示します。（UMTP のホームページより抜粋したものです。）

◆教育◆
（書籍、Web記事、eラーニングなど）

UMLの教育は、eラーニングやオンライン研修が充実してきています。また、書籍やWeb記事から学ぶのも有効です。

■良いモデルの作り方を学べる書籍と記事

まずは、分析レベルの良いモデルの作り方を学べる書籍とWeb記事を紹介します。

書籍「UMLモデリングレッスン」

UMLモデリングレッスンは、分析レベルのクラス図を作成する必要があるモデリング初心者に、お薦めの一冊です*6（**図23**）。「ものーことーもの」パターンといった概念の抽出を手助けできる構造パターンが数多く掲載されています。加えて、モデリングのコツも丁寧に記載されています。

Web記事「技術情報/モデリング・DDD」

技術情報/モデリング・DDD*7（**図24**）の「誤解しがちなモデリングの技」（第1回～第8回）では、レビューの場にてありがちな間違いや誤解を取り上げて解説するスタイルを用いています。いずれも、正確なモデリングを理解するのを促してくれるでしょう。

なお、設計レベルの良いモデルの作り方は、一般的な設計原則（GRASPなど）やデザインパターン（GoFなど）を学ぶのがよいでしょう。プログラミング言語に依存する部分もありますので、ここでは書籍やWeb記事の紹介を割愛します。

図23　書籍「UMLモデリングレッスン」（価格：2400円＋税、ISBN 978-4822283490）

図25　書籍「UMLモデリングのエッセンス 第3版」（価格：2400円＋税、ISBN 978-4798107950）

図24　Web記事「技術情報/モデリング・DDD」

＊6　「https://www.nikkeibp.co.jp/atclpubmkt/book/08/P83490/」から購入できます。
＊7　「https://www.mamezou.com/techinfo/modeling_ddd」から閲覧できます。
＊8　「https://www.shoeisha.co.jp/book/detail/9784798107950」から購入できます。

■UMLの使い方を学べる書籍と記事

次に、開発でのUMLの使い方について学べる書籍とWeb記事を紹介します。

書籍「UMLモデリングのエッセンス 第3版」

UMLモデリングのエッセンス 第3版は、UMLの表記と使い方がコンパクトにまとまっている書籍です*8（**図25**）。UMLに関する書籍は数多く出版されていますし、筆者は数多くの書籍

を持っていますが、「どれか1冊を残してほかは捨てろ」と言われたら、この書籍だけを残そうと思います。書籍の帯に「『UMLモデリングのエッセンス』は、今でも最高のUML記法入門書だ」と書いてありますが、筆者もそう考えます。

海外書籍「The Elements of UML 2.0 Style」

The Elements of UML 2.0 Styleには、UMLのスタイルガイドラインがとてもよくまとまっています。筆者は、大規模な開発案件でUMLのスタイルガイドラインを策定するときに存在を知り、参考にしました。「線を交差させない」といったUMLに限らずに言えることから「サブクラスはスーパークラスの下に置く」「菱形を気にするな」(集約かコンポジションに迷うようなら、関連にしておけばよい。大した違いはない。)といったUMLならではのスタイルガイドラインも豊富で、とても参考になりました。

「http://www.agilemodeling.com/style/」というURLにて内容の一部が公開されています(**図26**)。どんなスタイルガイドラインが掲載されているのか気になる人は、書籍を購入する前に、ざっと目を通しておくとよいでしょう。

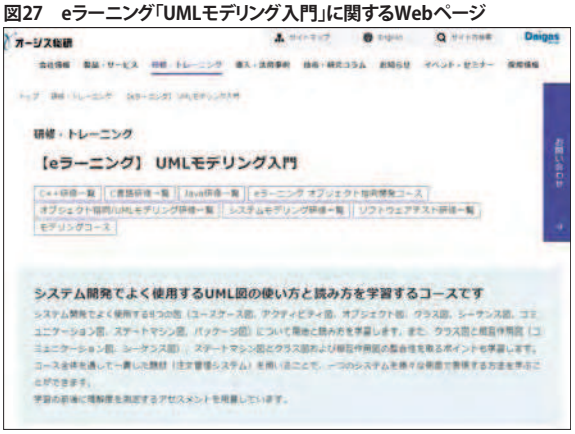
Web記事「システム開発にUMLを適用するためのFAQ」

古い記事なのですが、システム開発にUMLを適用するためのFAQは、今改めて読み返してみても「そうですね」と同意できる内容です。特に、組み込み開発系の人は、参考になるでしょう。「UMLはオブジェクト指向なしで使えるのでしょうか」「UMLには図がたくさんあるが、全部使う必要はありますか。いつ、どの図を使うべきですか」といった質問に対する回答が丁寧に記載されています*9。

■eラーニングとオンライン研修

最後に、オージス総研のeラーニングやオンライン研修を紹介します。

図27　eラーニング「UMLモデリング入門」に関するWebページ



eラーニング「UMLモデリング入門」

UMLモデリング入門では、開発での使用頻度が高い表記ルールを中心に、本特集では紹介しきれなかった表記ルールはもちろんのこと、オブジェクト指向の基本概念(カプセル化、継承、ポリモーフィズムなど)やモデリングのコツなども学べます*10(**図27**)。

オンライン研修「UMLモデリング入門－UMTP-L1認定」

UMLモデリング入門－UMTP-L1認定は、UMTPの「UML技能認定試験 L1レベル認定」テキストを用いた研修です*11(**図28**)。

筆者が所属するオージス総研では、ほかにも、プログラミング言語との対応(マッピング)ルールが学べるeラーニングや設計原則(リファクタリングを通して学ぶ良い設計)が学習できるオンライン研修なども取りそろえています。「オージス総研」や「UML」で検索してみてください。

＊9　「https://www.ogis-ri.co.jp/otc/hiroba/technical/DesignWave_UMLFAQ/」から閲覧できます。
*10　詳しくは「https://www.ogis-ri.co.jp/learning/l106114.html」を参照してください。
＊11　詳しくは「https://www.ogis-ri.co.jp/learning/c-01-03.html」を参照してください。

図26　「The Elements of UML 2.0 Style」の一部を掲載

図28　オンライン研修「UMLモデリング入門－UMTP-L1認定」に関するWebページ