技術レポート

UMLによるモデリング
ーオブジェクト指向開発における分析・設計方法ー

技術本部　技術研究室　山田喜彦

１．はじめに

本稿では、私が2000年度の研究テーマとしている「分析・設計フェーズのモデリング技術の研究」より、「UMLモデリング」について主要な論点を報告する。UML（Unified Modeling Language）とは、オブジェクト指向分析・設計に使用するノーテーション（記号法）と用語を定義している統一モデリング言語である。

従来、オブジェクト指向分析・設計法は数多く存在していたが、表記法としてはUMLがスタンダードとなってきた。しかし、現在においても、その表記法に基づいて、どういう開発プロセスで開発を行うかについては、まだ規格化できていない。ラショナル社（UMLを提案した会社）が提案している「RUP」(Rational Unified Process）が、事実上の標準の位置を占めているが、まだまだ今後も議論が続くものと思われる。本稿では、開発プロセスに関しての言及は最小限にとどめ、UMLを利用したモデリングの勘所を解説する。

1.1 ソフトウェア作成パラダイム

今日、システム開発を行うにあたって、以下の課題が頻繁に挙げられる。

- ビジネス変化の迅速化に対応するため、短期間でシステム構築やメンテナンスを実行する必要がある。
- 仕様があいまいな内に、開発を開始するケースが増加している。
- システムの分散化に技術的に対応する必要がある。

これに対し、以下のアプローチが有効であると考えられる。

- 標準の表記法（共通言語）の採用。
- 繰り返し型の開発プロセスの採用。
- オブジェクト指向技術の採用。
- コンポーネント技術の採用。

本章では、これらのアプローチに関し、開発工程とオブジェクト指向分析・設計を中心に解説する。さらに、今後のソフトウェア開発の標準化には、各社バラバラの方法論を採用するのではなく、共通の方法論と共通の記述法が必要であること、およびその具体的な内容を述べる。また、従来のデータ分析の手法を取り入れたオブジェクト抽出の方法を述べ、大規模なシステムでのオブジェクト指向分析・設計に利用できる方法論の手順を記す。

1.2 共通言語としてのUMLの採用

ソフトウェアの開発において、そのユーザー要件やソフトウェア仕様の記述について、OMG(Object Management Group）で規格化されたUMLを採用することは、産業界だけでなく大学、規格化団体を含めた全体的な動きとなってきている。ソフトウェアの開発の生産性をあげるには、もはや企業内の努力だけでは追い付かなくなってきており、知識や経験をさまざまな分野で公表して共有し、さまざまな開発者が利用して不十分な点をフィードバックする、というサイクルの確立が必要である。その表記の標準としてUMLが利用され始めている。

当社でも、独自の方法論や表記法を採用するのではなく、このグローバルな知識共有の流れの中で自社の開発を位置付け、実践して行くことが必要であると考える。

1.3 繰り返し型開発プロセスの採用

システムの技術革新が急速に進行している現在、技術的にインフラに完全に適合した分析・設計を、自信を持って行うことが難しくなってきている。その中で、ソフトウェ

アのアーキテクチャの確立を開発の初期段階で行うには、小規模のインプリメントを初期のフェーズにおいて実施し、頭の中で考えた解決策を実際に動作させて有効性を検証することが必要である。

また、ユーザー要件も、手作業で行っていた業務のシステム化という案件が減少し、新しいビジネスの創出に伴った案件が増加している。この場合、ユーザー要件のすべてを初期段階で固定化することは難しく、まず根幹となるシステム部分をシステム化し、その稼動状況を見ながらシステム要件を詰めて行く、という手順が必要である。

これらの問題に対処するには、従来のように、あるフェーズが完全に終了したら次のフェーズに移る、といった開発プロセスではなく、分析から統合テストといった一連の手順を繰り返し、調整を重ねながら開発を進める「繰り返し型開発プロセス」の採用が必要である。

1.4 オブジェクト指向技術の採用

プログラム言語のオブジェクト化はJavaの出現で加速され、オブジェクト指向技術を知らずして、コーディングは成り立たない時代になりつつある。設計・分析プロセスでは、なかなかオブジェクト指向化が進まなかったが、オブジェクト指向技術の標準化団体であるOMG（Object Management Group）により分析や設計における表記法であるUMLが規格化され、その加速にいっそう弾みがついた。

方法論を考えると、従来は「E-R分析」と「構造化設計」が基本的な手法であった。両者の基本的な考え方は、データ分析とプログラム分析とに分離して、システムを設計することである。この考え方は、アプリケーション構築の対象システムをクライアント／サーバー型としている場合には、それなりの有用性がある。クライアント層、プログラムサーバー層、データ層、というように、プログラムとデータが独立して構築でき、設計を別々に実施しても問題が少なかったのである。しかし、アプリケーション・レイヤが２層から３層、４層になり、Webサーバーやアプリケーション・サーバーを中心として、EJB（Enterprise Java Beans）などで、ビジネス・ロジックのコンポーネント化を推し進めるアプリケーション構築のスタイルにおいては、データとアプリケーションを統一して扱うオブジェクト指向の方法論が必要となる。

また、E-R分析は過去の分析結果を資産として再利用する考え方に欠けているようだ。つまり、パターンやフレームワークといったものを利用する方法論がなく、その表記も難しい。

今後、N層レイヤからなるシステムを構築する場合は、オブジェクト指向の分析・設計の方法論を採用し、公開されている多様なパターンやフレームワーク、といったものを利用しながら設計を進めていく手順を採用することが必要である。

1.5 ビジネスオブジェクトのコンポーネント化

ビジネスオブジェクトのコンポーネント化といっても、全てのオブジェクトがコンポーネント化の対象となるわけではなく、再利用されるオブジェクトのみを対象とする。

ビジネスオブジェクトはビジネスプロセス・オブジェクトとビジネスエンティティ・オブジェクト、ビジネスタスク・オブジェクトに分けて考えるとよい。

ビジネスプロセス・オブジェクトは、ワークフロー分析などから主に抽出されるもので、E-R分析ではイベント系エンティティとよばれていたものである。これはビジネスプロセスの標準化が進んでいない現時点では企業固有のものであり、例えば受注オブジェクトがこれにあたる。

このビジネスプロセス・オブジェクトは、他企業にとっては再利用が困難な部分である。それに対して、ビジネスエンティティ・オブジェクトはE-R分析のリソース系エンティティに近い概念で、ビジネスプロセスからは独立したものを意味する。例えば顧客オブジェクトがこれにあたる。先ほどの例に挙げた「受注」ビジネスプロセス・オブジェクトでは、顧客オブジェクトは「発注者」としてのロール（役割）を持って受注のプロセスに参加をする、というように考える。

このように、ビジネスオブジェクトの種類を確定し、その種類によって適切なインプリメント化を推進して行くことが重要である。

２．概念の説明

2.1 UMLについて

UMLは、ソフトウェアを記述するグラフィカルな言語である。1989年に設立され850社が加盟する標準化団体であるOMGによって、UMLの規格が制定された。現在、ソフトウェアの仕様を記述する上で、最も利用されている言語である。

開発方法論は一般に仕様の記述言語と開発のプロセスから構成されるが、UMLは特定の開発プロセスを前提としない記述言語である。

UMLでは、ソフトウェアの仕様を記述する際に、５つのビューから考察を始める。システムにはさまざまな側面があり、それらを中央のユースケースが中心となって結合し合い、システムの特定の側面はそれぞれのビューが表現して強調しながら、システム全体を記述しよう、という方法である。

UMLを構成する５つのビューを図１に示す。

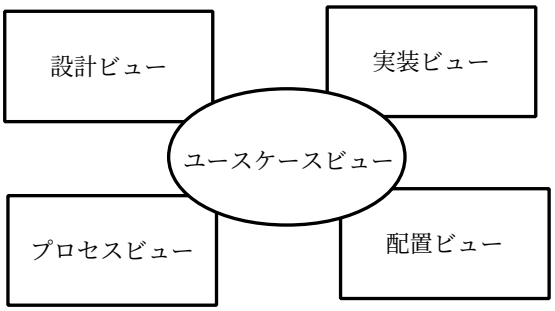

図1　UMLを構成する5つのビュー

⑴　ユースケースビュー

エンドユーザー、アナリストからみた、システムの振る舞いを記述する。

・静的：ユースケース図

・動的：相互作用図、状態遷移図、アクティビティ図

⑵　設計ビュー

システムの機能的要求、エンドユーザーへのサービスを記述する。

・静的：クラス図（オブジェクト図）

・動的：相互作用図、状態遷移図、アクティビティ図

⑶　プロセスビュー

システムの並列性や同期、スレッド、プロセスを記述する。

・静的：クラス図（オブジェクト図）

・動的：相互作用図、状態遷移図、アクティビティ図

⑷　実装ビュー

実際のシステム稼動に関する情報を記述する。

・静的：コンポーネント図

・動的：相互作用図、状態遷移図、アクティビティ図

⑸　配置ビュー

システム構成物の分散、配布、インストールを記述する。

・静的：配置図

・動的：相互作用図、状態遷移図、アクティビティ図

2.2　モデリングについて

グラフィカルな記述方法で開発対象のドメインやシステムのモデルを作成することをモデリングといい、表記法にはUMLを用いる。

オブジェクト指向開発において、このモデリングは特に重要な役割を担う。　ソフトウェアを記述する際に、対象とするユーザーの業務システムをグラフィック・モデル化したものを利用する、ということが特徴である。このため、ユーザーの要求定義とソフトウェアとの接点と対応が解りやすくなる。また、ユーザー要件を中心とした分析フェーズと、システム設計が中心となる設計フェーズのギャップが少なくなり、メンテナンス・フェーズにおいても、変更対象が分かりやすいというメリットがある。

記述方法としてのUMLは、分析フェーズからインプリメント（テスト）までの幅広い開発フェーズをカバーする。クラス図を例にとってみても、分析フェーズで作成されるものとインプリメント段階で作成されるものとは、詳しさが決定的に異なる。

モデリングでは、以下の点に注意してUMLを作成すること重要である。

①誰と対話するか意識せよ

②何を表記しようとしているか

①は例えば分析段階では顧客とのコミュニケーションを目的とするわけであるから、システム内部の構成情報は最小限にするべきである、というようなことである。②はUMLには3つの観点があり、これを開発のフェーズでどの観点で記述しようとしているのかを明確にせよ、ということである。

UMLの3つの観点とは以下のものである。

⒜概念的な観点

⒝仕様の観点

⒞インプリメントの観点

⒜はビジネスの観点と言い換えてもよい。この場合に重要なことは、クラス図において「何がクラスとして認識されているか」、「どのクラスとどのクラスが関係しているか」という記述である。これは開発者がユーザーとコミュニケーションする際の主要な観点である。

⒝は、あるクラスが果たすべき責務というものが何か、ということである。具体的にはクラス図の各クラスのメソッド定義において、あるクラスが、どのようなサービスを他のクラスに提供しようとしているかに着目する。これは、システムの設計フェーズでの観点である。また、クラス間のリレーションも、この観点から解釈がなされる。例えば、従業員のクラスからプロジェクトのクラスにリレーションがあれば、従業員がプロジェクトに関する質問に答える責務がある、ということが読み取れる。

⒞は、クラスの構造と仕様をプログラムにインプリメントし、実際に動作するプログラムを作成するための観点である。クラスの内部に主要な関心があり、クラス間のリレーションに対し、あるクラスが別のクラスに対するポインターを持っている、というように解釈する。

2.3　データ分析とオブジェクト分析

データ分析のE-R手法は時代遅れとなってしまったとはいえ、エンティティの捕捉などの部分は、オブジェクト分析でも十分利用することができる。

一般に、オブジェクト分析におけるクラスの捕捉はユースケース記述において、名詞・名詞句をクラス候補して考えながら分析するのが典型的な手順である。（筆者は後で述べるがCRCカードを主体とする方法を推奨する。）

本文書では、まず、オブジェクトを以下の４つに分類する。

①ビジネスエンティティ・オブジェクト(リソース系)
②ビジネスプロセス・オブジェクト（イベント系）
③ビジネスタスク・オブジェクト（一時的）
④ビジネスイベント・オブジェクト

これらのオブジェクト分析と E-R 分析の関係を表１に示す。

まず、ビジネスエンティティ・オブジェクトの見分け方を述べる。エンティティは一般に「可視であれ不可視であれ、管理対象として補足したい存在」であり、identifier を必ず持つ。identifier は「XX 番号」や「XX コード」という名称であり、エンティティを識別するものである。エンティティの名称は、identifier から番号やコードを除いたものを採用する。例えば、「従業員コード」という identifier があると、「従業員」というエンティティが捕捉される。

次に、ビジネスプロセス・オブジェクトの見分け方を述べる。ビジネスエンティティとの違いは「XX する」のような表現を持つことである。例えば「受注」は「受注する」という表現ができるのでイベント系エンティティである。この「受注」には、必ず「受注年月日」のようなイベントが発生する日付を持つことが特徴である。

このビジネスプロセス・オブジェクトには、ビジネスエンティティ・オブジェクトがロール（役割）をもって関係を持つことが特徴である。

ビジネスタスク・オブジェクトは、ビジネスプロセス・オブジェクトの内で、変化しやすいもの、例えば、ある企業内のビジネスルールや慣習のようなものを分類する。顧客審査や仕入先選定も分類に含まれる。

ビジネスイベント・オブジェクトは「在庫の数量がある値以下になった」など、オブジェクトの状態が変化することによって発注のプロセスが起動するといった場合に、きっかけとなる発生イベントである。

以上のビジネスオブジェクトは、分類ごとに以下の性格を持つ。ビジネスエンティティ・オブジェクトは、業務の変化に影響されにくいものである。メソッドの量は少ないが属性は豊富で寿命は長く、RDB に保持されるような永続オブジェクトとなることもある。ビジネスプロセス・オブジェクトは、業務そのものといってもよく、業務変化に伴って変更されていくものである。このオブジェクトはプロセスの性格上、関連するビジネスエンティティ・オブジェクトがある役割をもって参加して、初めて意味をもつオブジェクトである。プログラム実装上は、抽象オブジェクトと具象オブジェクトの組み合わせでオブジェクトを構成すると、変化に強いものにすることができる。ビジネスタスク・オブジェクトは、業務よりさらに細かなルールを指す。明文化された業務形態ではなく、担当者のポリシーや慣習などで、常に状況に応じて変化する。ビジネスイベント・オブジェクトは、プロセスオブジェクトと似た性格を持つ。

なお、この他には関連のクラスもある。この扱いは、ビジネスエンティティ・オブジェクトに準じる扱いとする。

３．ドメインのモデリング

開発の初期作業は、ほとんどが分析作業である。この作業の目的は、ソフトウェアのモデリングではなく、ユーザーの業務をモデリングすることにある。これは、ドメインのモデリングと一般によばれ、システム対象の業務をシステム化の観点からモデル化を行うものである。

ドメインのモデリングは、まずビジネスのワークフロー表現を中心とするビジネスワークフローとユースケースによるモデリングによって、ドメインの動的な側面を記述する。そして、クラス図によってドメインの静的な側面を記述し、合わせて分析作業の成果とする。

3.1 ビジネスワークフロー

ビジネスワークフローは大規模な開発において、ドメインのモデルの動的側面を記述するのに適した記述方法である。これは必須の記述ではないが、業務が受注・発注・受払いといった「モノ」の流れを中心にモデリングする必要のある場合に有効である。

UML で記述する場合は、アクティビティ図を利用する。しかし、実際にツールで作図をしてみると、厳密性を要求しすぎていて（例えば同期の表現）、実際にはビジネスのワークフローを表現しにくい。今後の UML 改定時に改善される予定であるが、現時点では Xupper（ケン・システム開発株式会社の上流 CASE ツール）など、専門のツールの利用を推奨する。

表１　オブジェクト分析と E-R 分析の用語対応

オブジェクト分析	E-R分析
ビジネスエンティティ・オブジェクト	リソース系エンティティ
ビジネスプロセス・オブジェクト	イベント系エンティティ
ビジネスタスク・オブジェクト	なし
ビジネスイベント・オブジェクト	なし

3.2 ユースケース

オブジェクト分析では、ユースケースを利用するのが一般的である。

ユースケースは、ユースケース図とユースケース記述からなり、システム外部にあるアクター（通常は人間）が、システムと対話をする際の一連の出来事が記述される。ユースケースは主に、UML の対話的な側面を表現する図である。しかし、本来静的であるシステムが外部刺激を受けて変化することを記述する、という意味で、動的な側面を記述する役割も持っている。

また、ユースケースはシステム要求の（記述という面からの）最小単位であり、一連のユースケース記述によって、ある具体的な状況でシステムが果たすべき機能要求を明らかにしている。

このユースケース記述は、クラスの選定やクラスの仕様の概要が確定した際には、その機能テストにも利用される。

注意すべきことは、ユースケースは従来の意味での仕様記述ではなく、ある目的を持ってアクターがシステムを利用する際の、標準的（および例外的）なアクションの手順を明らかにし、記述することが主眼であることである。仕様としての静的な記述というより、アクターがシステムに対して行う具体的なアクションと、それに対するシステムの応答という動的な記述といえる。

ユースケースは、全てのビジネスプロセスに関して作成されなければならない、というものではない。このユースケースとクラス図（必要に応じて相互作用図とシナリオ）の作成初期、または静的仕様記述で総合的に要求記述ができればよい。

ユースケースにおける見逃せない重要な点は、ユースケースがシステムを分解する機能を果たす点である。そしてこのユースケースが開発プロセスの主導的な役割を果たす。オブジェクトの分析はユースケース単位、または、そのいくつかのまとまりで記述することが多く、スケジュール管理やテスト計画などもユースケースを単位とすることが基本である。

3.2.1 アクターの決定

アクターとは、オブジェクトのロール（役割）、またはシステムの外のオブジェクトであり、直接にユースケースと対話をするものを表す。

具体的には、システムを利用するユーザーや、ユースケースと対話するソフトウェア・サブシステムであり、さらにはハードウェアの一部を表すこともある。

受発注のシステムでは、販売員や店員、顧客はアクターである。出荷というユースケースを考えているときには、トラックの配送システムはソフトウェアとしてのアクターとして表現される。アクターはシステムを利用したり、保守したりする「人間」であったり、あるシステムからデータを受け取る外部のシステムであったりもする。

3.2.2 ユースケースの作成

ユースケースは、システムが提供する最小単位の機能である。典型的な例は、「受注を登録する」とか「与信のチェックをする」というものである。

まず、各アクターごとに基本的で、まとまりのある作業の単位を考え、それをユースケースとして考える。アクターがシステムを利用する際に、どのような作業を行うか、どんな情報を入力したり、利用したりするのかを考える。受発注では、「受注」や「与信のチェック」、「在庫のチェック」などがそれにあたる。

3.2.3 ユースケース記述の作成

各ユースケースに関しては、ユースケース記述で、アクターの具体的なシステムとの相互作用を明確化する。

ユースケース記述は、システムの外部とシステムのユースケースという基本的な単位の間で、どのようなやり取りが想定されるかを具体的に示すものである。これは仕様、というよりシステムの典型的な対話を具体的にインスタンスへ割り当てながら記述するもので、システムのイメージを作成することが本筋であると思われる。

ユースケース記述のフォーマットは、UML では規定されていない。ここでは、よく利用される一般的なフォーマットを示した。各記述はイベントフローとよばれるアクターとシステムの相互作用の順序（シークエンス）を持った記述からなる。イベントフローは、さらにメインとなるイベントフローと例外的なイベントフローを分けて記述する。各イベントフローによって複数のシークエンスが表現されるが、その中の１つのシークエンスをシナリオとよぶ。

シナリオの条件判断が複雑に分岐する場合は、アクティビティ図による記述が効果的である。

図２にインターネットによる注文の登録のユースケース図を示す。

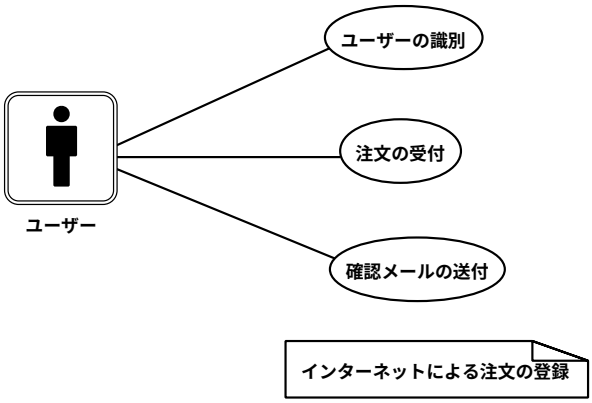

図２　ユースケース図（インターネットによる注文の登録）

この場合のユースケース記述は、次の流れで行われる。

〈ユーザーの識別〉

・ユースケース名：ユーザーの識別
・目的：ユーザー認証の実行
・アクター：WWW から本を購入するユーザー
・事前条件：ユーザーは未登録
・メインイベントフロー：
　－アクターはインターネットで書籍を注文しようと、本システムの WWW サイトを開く。
　－システムは識別のためにユーザー ID とパスワードの入力画面を表示する。
　－アクターが初めてのユーザーなので、ユーザー ID とパスワードの登録がまだされていない。そのためシステムはユーザー登録をするように促す。
　－アクターは、登録用のボタンをクリックする。
　－システムは登録用の画面を表示する。
　－アクターは自分の情報を入力する。
　（メールアドレス、ユーザー ID、パスワード、住所、氏名、性別、年齢、クレジットカード情報を入力する。）
　－受け付けのメッセージが表示され、認証が終了する。
・例外イベントフロー：クレジットカードの有効期限が過ぎている場合は、再入力が促され、変更するための画面が表示される。
・事後条件：ユーザーの認証が終了
・備考：ユーザー認証情報
　－本情報は、保存され、次回の注文時にはユーザー ID とパスワードを入力するだけで、識別される。
　－同じ PC からアクセスする場合は Cookie の機能により、前回登録のユーザー ID は自動的に識別される。

3.3　静的仕様記述の作成

ユースケース記述と並行して、従来の記述によって仕様作成を行う。各ユースケース単位に以下の記述対象について作成する。

・機能要求以外の事項（レスポンス、スループットの要求等）
・静的な制約（オブジェクト間の参照制約や依存性の記述等）
・約束事（ビジネス的な内容、ルール等）
・システムの時間制約による自動的起動タスク。

なお、少量であればユースケース図のノートやシナリオの中で記述してもかまわない。ここでは Contact（契約）に基づき、システムの振る舞いを記述する場合のレイアウトを以下に示す。実際にはほとんどフリーフォーマットでよい。

この仕様の記述は各ユースケースで抽出されたイベントの単位で行う。

名称：
責務：
例外：
事前条件：
事後条件：
仕様内容：

3.4　オブジェクトの切り出し

オブジェクトの確定の方法を、以下に実例をあげて説明する。問題記述は仕様メモの形式で作成されたものを用いる。これは、オブジェクト切り出しを説明するためにあえて作成したもので、UML では表現されない知識を含んで文書化にしたものである。

3.4.1　問題記述（仕様メモ）の例

⑴　取引条件登録・更新

商社がメーカーに発注する際の契約情報である取引条件の登録を行う。登録手順は、まず商社コード、商社部門コード順に、担当者が担当している社外管理取引先のリスト（商社名、部門名）を表示する。次に対象とする取引先を選択すると、取引条件の中身の一覧が取引条件番号の順に表示される。ここで対象となる取引条件に関し、登録と更新を実施する。

⑵　受発注データ入力／注文訂正・取消

受発注データ入力は、商社からメーカーに新規注文を行う入力画面である。取引条件を決定し、契約番号を指定してから注文データを入力する。契約番号の指定は取引条件の選択により行い、「注文の追加」画面では納入先、支払者、需要家、オーダー区分、赤黒区分、担当者等の入力を行う。注文番号は通常は自動付番されるが、任意の番号を入力することも可能とする。注文品名は、注文明細により更新を行い、ひとつの注文について複数の品名（注文明細）を指定することが可能である。注文明細ごとに取引条件を選択し、単価、割増／割引単価等を設定する。

3.4.2　ビジネスオブジェクトの発見

⑴　発見の手順

①Identifier（「コード」「番号」「ID」を末尾に持つもの）を列挙する。
・商社コード
・担当者コード
・メーカーコード
・部門コード
・納入先コード
・支払い者コード
・取引条件コード
・品名コード

・契約番号

②Identifier の修飾子を除いたオブジェクト候補を列挙する。

・商社
・担当者
・メーカー
・部門
・納入先
・取引条件
・品名
・契約
・取引条件

③ひとつの Identifier では識別できないオブジェクト候補を除外する。

・今回は挙げていないが、例としては「在庫」がある。在庫は関連クラスである、と考える。
・ビジネスプロセス・オブジェクトを選択する。
　－方法：「XX する」というような表現がなりたち、さらに事象が起こった日付を持つものを、ビジネスプロセス・オブジェクトとする。
　－契約は「契約する」という表現がなりたち、契約日を持つのでビジネスプロセス・オブジェクトである。

④ビジネスエンティティ・オブジェクトを以下に挙げる。

・商社
・担当者
・メーカー
・部門
・納入先
・品名
・取引条件

⑤ビジネスプロセス・オブジェクトを以下に挙げる。

・契約

3.5 CRC カードの利用

CRC（Class Responsibility Collaboration）カードに、手書きで「クラス名」「クラスの果たすべき責務」「責務を果たすのに協力するクラス」を記述して、ウォークスルーしながら、クラスの適切な分割と相互作用を大まかに決める。この CRC カードは UML の正式な手法ではないが、有効性が広く認められている手法である。

CRC カードの書式（例）を図３に示す。

クラスの責務とは、クラスの属性や操作のような細かい記述ではなく、クラスの目的を高い抽象度で表したものである。サービスの集合やシステム内でのコンポーネントのロール（役割）を表す。通常は３つ程度で表し、それ以上ならクラスの分割を検討することが必要である。

CRC カードは、クラス間の相互作用の調査に有効であり、相互作用図によって、代替手段の検討が容易に行える利点がある。

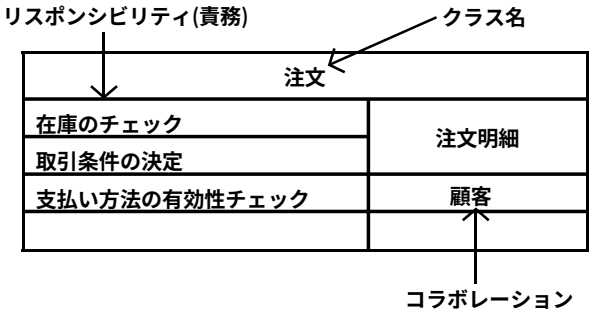

図３　CRC カードの書式（例）

前述のオブジェクトの切り出し（識別子からオブジェクトを見つけるやり方）を補助的に利用し、この CRC カードを利用することによって、オブジェクトを確定する方法を一般的に推奨している。その場合は、前述のオブジェクトの切り出しは、以下のような利用を行う。

CRC カードでオブジェクトを切り出す場合は、オブジェクトの粒度やロールが問題になることが多い。例えば、「ヒト」の切り出しを行う場合は「人間」というオブジェクトとして捕えるのか、「従業員」としてか、「プロジェクト・メンバー」、あるいは「給与支払い対象」なのか、という問題である。これは、定義しようとしているドメインにおいて、管理対象となっているか否かで判断する必要があり、補足するには「識別子は、何があるか？」ということである。識別子を持つのであれば、「管理対象である」と判断できる。例えば人事システムでは、従業員コードが識別子であって従業員がオブジェクトなので、プロジェクト・メンバーは、その従業員オブジェクトがあるビジネスプロセスに参加した場合のロールである、と捕える。なお、ロールのクラス表現の方法については後述する。

Responsibility-driven modeling（Cockburn）を行う手順を述べる。

①Preparation（準備）
　ユースケースを幾つか選択する（シナリオとなる）。

②Invention（発案）
　カードを１群（アクターから直接、相互作用を行うもの）と２群（１群と相互作用をするもの）に分けて机上に置き、ウォークスルーを実施する。シナリオにしたがって、カードを動かし、必要あればカードの責任を示したり、新たに書き入れたり、新規のカードを追加する。

③Evaluation（評価）
　状況をさまざまに変えて、カードが対応できるか確かめる。

④Consolidation（統合）

カードを追加したり、脇に動かしたりして、デザインの改良を実施する。必要あればデザインの代替案を検討したり、名称やクラスのレベルを検討する。

⑤Documentation（ドキュメント）

キーとなるリスポンシビリティを書き留める。主要なシナリオに関して相互作用図を作成する。

3.6 シークエンス図の作成

CRC カードにて、大まかに決めたクラス（オブジェクト）と責務を、シークエンス図に記述する。これは、重要なシナリオに限定して作成する。

図４にシークエンス図を示す。

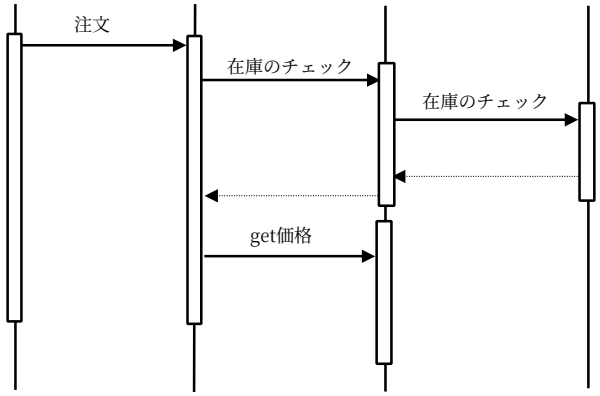

図４　シークエンス図

3.7 概念クラス図の作成

オブジェクトが確定した後、概念レベルをクラス図で表記する。

・システムが、ある一定の大きさ以上の規模であるなら、システムをパッケージに分割する。

・システムを、どの単位で分割するかについては以下のように考える。

－ユースケースの粒度が大きい場合は、１ユースケースを１パッケージとする（これは、ほとんどない）。

－ユースケースの粒度が小さい場合は、関連する複数のユースケースを１パッケージとする。(前述のユースケース記述では、「ユーザーの識別、注文の受け付け、確認メールの送付」を、まとめて「インターネットによる注文の登録」とする）

－Martin Fowler は「A４サイズの用紙１枚でシステム全体のクラス図を描けなくなったらパッケージに分割せよ」としている。

・分割は UML のパッケージ図にて表記する。

・各パッケージ内にクラス図を記述する。

・上記とは別に、パッケージ全体を統合したクラス図を作成することは、全体の整合性のチェックに有効である。

図５、図６にドメインパッケージ図と販売パッケージ中の概念クラス図をそれぞれ示す。

概念クラス図の内容に関しての注意点をまとめる。

・概念レベルのクラス図の目的は、ユーザーの世界をモデリングすることであって、ソフトウェアをモデリングするものではない。

・主要な登場人物、もの、概念、事象がなにか、その相互の関係はどうなっているのかを記述する。

・このクラス図はユーザーとの仕様の確認に利用すること（つまりコミュニケーションの手段である）と、ソフトウェアのモデリングを目的とした仕様レベルのクラス図の基本構造となる。

・CRC カードにて抽出したクラスの内、ユーザー・インタフェースやレポートなどは除外する。

・初期においては、クラスとその関係を中心にモデリン

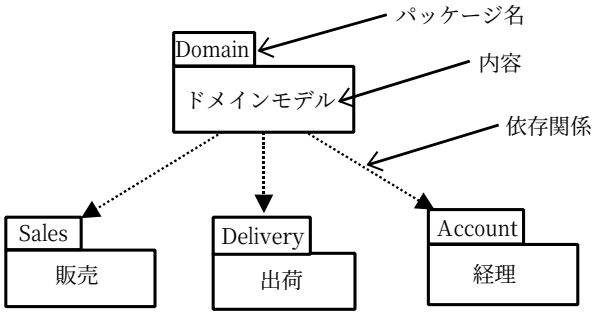

図５　ドメインパッケージ図

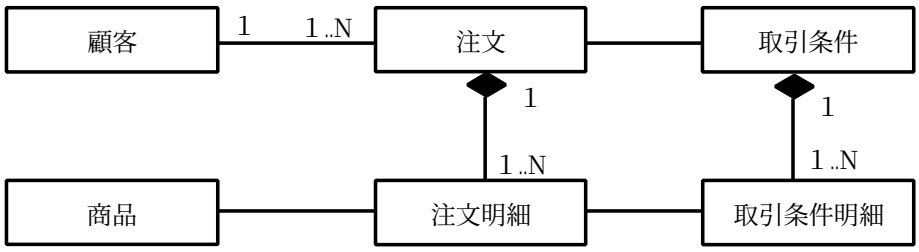

図６　販売パッケージ中の概念クラス図

グする。

・概念レベルではクラスの確定とクラス間の関係の大まかな確定が主目的であるので、属性とメソッドは必要最小限でよい（特にメソッド）。

・クラスが確定してきたら、CRC カードの「責務」を果たすために必要な属性とメソッドを順次記述していく。

・属性はそのまま実装に引き継がれるが、メソッドは実装レベルにおいて変更されることが多い。

クラス図においては役割（ロール）に注意する必要がある。

ロールの表現には「関連における表現」と「概念（クラス）による表現」がある。概念レベルでは識別子によって、ドメインにおいて管理すべき概念（クラス）を識別した後、関連によるロールの表現を行うことを推奨する。そして、設計モデルであるシステムモデルにおいては、インプリメントを考慮したクラスで表現する（設計モデルについては後述する）。

概念クラスにおけるロール表現（関連するロール）を図７に示す。

図７　概念クラスにおけるロール表現(関連によるロール)

４．システムモデリング

分析フェーズで作成したドメインモデルを、設計フェーズにおいてシステムのモデリングに発展させる。システムのモデルは、開発するソフトウェアのモデルである。

まず、概念モデルにドメインのクラスと位置付け、さらにプレゼンテーション・レイヤーを追加する。アプリケーションによっては、２つのレイヤーの中間にアプリケーション・ファサードというレイヤーも追加する。レイヤーはパッケージとして表現する。

アプリケーション・ファサードはユーザー・インタフェースのクラスからドメインを取り扱う部分を取り出したもので、アプリケーション・ロジック・レイヤーともよぶ。原則は１つのユーザー・インタフェースに対して１つ作成されるが、複数のユーザー・インタフェースを１つで扱う場合もある（受注と受注明細のような緊密な関係の場合）。ただし、アプリケーション・ファサードはユーザー・インタフェースに依存して作成されるので、ユーザー・インタフェース固有の処理や表示項目に依存性が高いという特徴があり、売上時の顧客チェックのような一連の手順はドメインに依存するものであり、次に述べるコントロール・オブジェクトとして扱う。アプリケーション・ファサードの共通ロジックがドメイン・モデル・レイヤーに構築されたものがコントロール・オブジェクトといえる。

ドメインのクラス図の中で、コントロール・オブジェクトを発見する。ユーザー・インタフェースごとに、アプリケーション・コーディネータ、ドメインクラスのリレーションをクラス図で表現しながら、主要なものはシークエンス図を描く。この過程で、クラス図の属性やオペレーションが詳細化していく。

設計レイヤーを図８に示す。

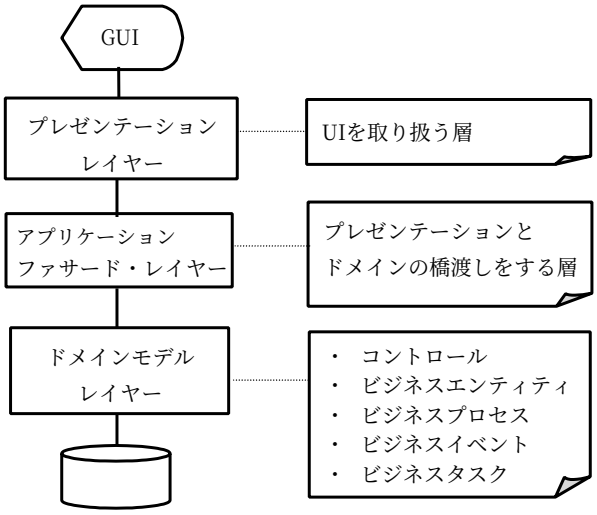

図８　設計レイヤー

システムモデルはパッケージに分割し、各パッケージの中にクラス図を記述する。

システムモデルのパッケージを図９に示す。

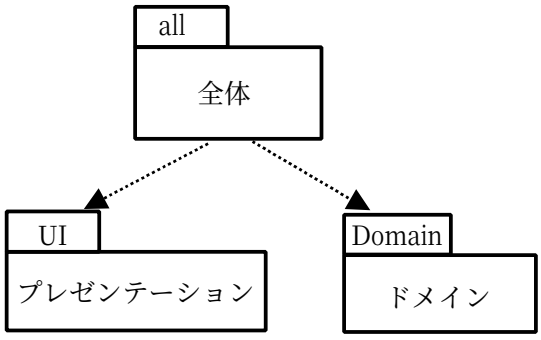

図９　システムモデルのパッケージ

・ユーザー・インタフェース・パッケージには、プレゼンテーション・レイヤーとアプリケーション・ファサード・レイヤーをクラス図として記述する。ドメインからは、必要なクラスを引用する。

・ドメインパッケージは、分析のドメインクラス図を詳細化、洗練化して設計クラス図に発展させる。

4.1 設計クラス図

ドメインのクラスを利用しながら、ビューのパッケージで設計のクラス図を作成する。ファサードのクラスを、基本的にはユーザー・インタフェースごとに設計し、ドメインクラスに必要な属性、操作を追加していく。

前述したロールを設計モデルに展開するには、いくつかの方法がある。ここでは、代表的なものを述べる。

⑴ Role インタフェースによるロール表現

ロールをインタフェースとして表現し、外からはインタフェースを通して利用する。

ロールを扱う場合に、最も使用される表現である。

設計クラスにおけるロール表現（Role インタフェースによるロール表現）を図10に示す。

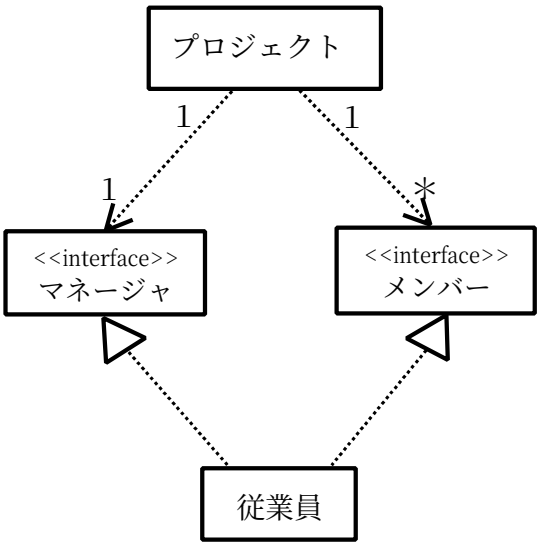

図10　設計クラスにおけるロール表現
（Role インタフェースによるロール表現）

⑵ 特徴

・あるロールを複数のオブジェクトが持つ場合にも利用できる。
・インタフェースとその実現クラスとの間に、実装の継承の必要が無い場合に有効である。
・GoF の Factory Method のような、生成パターンを利用したセットアップが必要である。

5．おわりに

UML を利用した、オブジェクト指向によるシステム分析・設計に関して述べてきた。オブジェクト指向技術はSmallTalk-80から数えて、早20年の歳月が流れた。その間において、最もセンセーショナルかつ重要な話題は UML、パターン、Java だろうと考察する。これらは記述言語、ノウハウの言語、実装の言語ともいえるもので、オブジェクト指向の３点セットと位置付けても過言ではないだろう。

この３点セットによって、企業間の壁を超えたノウハウの共有、コンポーネント・フレームワークによるソフトウェアの作成という、長年の課題にやっと取り組める下地ができた。

当社においても、最近は Web ベースのシステムを中心に、オブジェクト指向分析・設計の実践がなされるようになった。多くの開発によって技術蓄積がなされるのは、たいへん喜ばしいことではあるが、アーキテクチャの選択など設計判断の重要な点を、相互に情報共有する仕組みがないのは残念である。今後、当社が３点セットを用いて社内はもちろん、社外に向けても積極的に情報発信することが重要であると思われる。

〈参考文献〉

1．マーチン・ファウラー著，堀内一監訳：“ アナリシスパターン ”，アジソン・ウェスレイ・パブリッシャーズ・ジャパン㈱（1998）
2．マーチン・ファウラー著，羽生田栄一監訳：“ UML モデリングのエッセンス第二版 ”，㈱翔泳社（2000）
3．マーチン・ファウラー著，児玉公信他訳：“ リファクタリング：プログラミングの体質改善テクニック ”，㈱ピアソン・エデュケーション（2000）
4．クレーグ・ラーマン著，依田光江訳：“ 実践 UML ”，㈱ピアソン・エデュケーション（1998）
5．フィリップ・クルーシュテン著，藤井拓訳：“ ラショナル統一プロセス入門 ”，㈱ピアソン・エデュケーション（1999）
6．イヴァー・ヤコブソン他著，藤井拓監訳：“ UML による統一ソフトウェア開発プロセス ”，㈱翔泳社（2000）
7．エーリッヒ・ガンマ他（GoF）著，本位田真一他訳：“ デザインパターン ” ソフトバンク㈱（1995）
8．マーク・グランド著，原潔他訳：“ デザインパターン ”，㈱カットシステム（2000）
9．ゲリ・シュナイダー著，羽生田栄一監訳：“ ユースケースの適用：実践ガイド ”，㈱ピアソン・エデュケーション（2000）
10．佐藤正美：“ T 字型 ER データベース設計技法 ”，㈱ソスト・リサーチ・センター（1998）
11．Martin Fowler：“ Dealing with Roles ”，http：//www.martinfowler.com
12．Martin Fowler：“ Application Facades ”，http：//www.

martinfowler.com
13. J. Donovan Wells :“ Extreme Programming Project ”, http : //www.extremeprogramming.org/map/project.html
14. Scott W. Ambler :“ The Unified Modeling Language v1.1 and Beyond : The Technique of Object-Oriented Modering ”, http : //www.ambysoft.com
15. Alistair Cockburn :“ Responsibility-based Modeling ”, http : //members.aol.com/humansandt/techniques/responsibility.htm