Rational Software Modeler と Rational Software Architect のためのモデル構造ガイドライン (2004 リリース)

ホワイト・ペーパー

Bill Smith、Model Driven Development、IBM Rational Software

V1.0
2004 年 9 月 8 日

目次

1. はじめに

対象読者

このホワイト・ペーパーは、Rational Software Architect (RSA) 製品の使用にあたって、Rational Unified Process (RUP®) のガイドラインを適用したいと思っている RSA ユーザーを対象としています。Rational Software Modeler (RSM) のユーザーにも役立ちますが、RSM には存在しない機能 (RSA だけに用意されている機能) に関する説明が含まれていることに注意してください。このホワイト・ペーパーでは、RSA で新しいモデル・セットを作成する場合を主に取り上げています。RSA を使い始めたばかりのユーザーであっても、以前に Rational Rose または Rational XDE を使用していて、それらの製品からモデルをインポートしたのであれば、インポートしたモデルの再編成のための参考資料としてこのホワイト・ペーパーを利用できます。

このホワイト・ペーパーでは、UML に関する幅広い知識を持ち、RSA の操作 (特にモデリング、変換、ビジュアルなコード編集) に関する基本的な概念や理論を幅広く理解している読者を想定しています。

目的

RUP で記述しているモデル・セットは、システムの問題/解決領域で入念に定義したさまざまなパースペクティブ（観点） を反映しています。このモデル構造セットの利便性は、数多くの実際のプロジェクトで実証されており、RUP などの正式なプロセスに準拠するかどうかにかかわりなく、検討するだけの価値があると言えます。このホワイト・ペーパーでは、RSA を使用して RUP モデルの成果物を実際に作成する方法を解説します。

表題が示しているとおり、このホワイト・ペーパーで取り上げるプロジェクトとモデルの構造は、あくまでもガイドラインであって、絶対的な規則ではありません。RSA で特定の RUP 成果物をモデリングするかどうかは、それぞれの開発プロセスで検討しなければならない事柄であり、プロジェクトごとに決定内容が違ってくる場合もあります。また、RUP はプロセスに関する厳密な規則集ではないという点も押さえておいてください。むしろ、これはプロ

セスに関する「枠組み」であり、この枠組みの中で、非常に正式なプロセス定義からおおざっぱなプロセス定義に至るまで、さまざまなプロセス定義を策定できます。

UML についても、非常に正式な使用法もあれば、おおざっぱな使用法もあります。UML モデルを正式なアーキテクチャー図面のように扱って、実際の作成段階でも厳密にそのモデルに従って作業する場合もあれば、設計のおおまかな概略を示したスケッチ程度に見なして、プロジェクトが実装の段階に入ったら破棄するような場合もあるわけです。RSA では、このようなプロセスやモデリングの両極端のケースに対応できます。その観点からしても、このホワイト・ペーパーで紹介するガイドラインは、自由な発想を妨げるものではなく、むしろそれぞれの状況でベストと思えるプロセスに合わせて RSA の各機能を活用する方法を考え出すためのものであると言えます。

さらに、RSA では、モデルを単なる青写真として使用するだけでなく、実装を自動生成するためのソリューション仕様書として使用することも可能です。そのために活用できるのが、RSA のモデルからモデルへの変換機能とモデルからコードへの変換機能です。Model-Driven Development (MDD) を実施するために RSA の変換機能を使用するには、モデル構造に関する特別な注意点について検討しなければなりません。MDD を実施するために RSA のモデルと変換機能を活用する場合は、developerWorks に用意されている各種の RSA MDD 固有の資料もご覧ください。

範囲

このホワイト・ペーパーでは、RUP のモデル成果物を RSA で表現する方法について解説し、そのような成果物の内部編成構造のためのガイドラインを紹介します。以下のような内容は、取り上げません。

- RUP のモデル成果物の概念的な基盤に関する細かな説明
- RUP の成果物に関する詳細なセマンティクスや図表を指定するためのプロセスやテクニック

RUP 成果物の定義方法、開発方法、モデリング方法に関する一般的な （ツールに依存しない） 情報については、RUP のマニュアルを参照してください。

RSA モデルの開発に関するツール固有のテクニックについては、以下の資料を参照してください。

- 製品の資料 (チュートリアル、サンプル、オンライン・ヘルプ)
- RSA 固有の RUP 構成に含まれているツール・メンター (このホワイト・ペーパーもその一部)
- developerWorks の RSA 関連資料

表記上の規則

IBM Rational Rose または XDE から移行する RSA ユーザーにとって特に役立つ情報を網かけのテキスト・ボックスとして記載しています。

> *XDE/Rose*
> 旧バージョンの XDE または Rose のユーザーに役立つ説明。

このホワイト・ペーパーの構成

この後の『基本的な概念と用語』のセクションでは、重要な用語について説明し、RSA 製品でモデルを実装する方法の概略を示します。

次に、『RUP モデルと RSA モデルの対応関係』のセクションでは、RUP で定義されているモデル・タイプを RSA がどのようにサポートしているのかを取り上げます。

その後のいくつかのセクションでは、RSA で各種タイプのモデルを構成するためのガイドラインを示します。特に、プロセス、モデリング方法、アーキテクチャー制御などに関してどれほどの厳密さを望むのかに応じて、同じタイプのモデルでもさまざまな方法を採用し得ることについて説明しています。

最後に、モデルをいくつかのモデリング・ファイルに分割することに関連したさまざまな問題点を取り上げます。特に、規模を管理したり、チーム・メンバー間でのモデルの共有を可能にしたりして、ファイルの競合やマージの際の不整合を最小限に抑えるための戦略についても触れています。

2. 基本的な概念と用語

モデル

RUP では、問題/解決領域を特定の観点 (パースペクティブ) から完全に記述したものをモデルといいます。問題領域またはシステムを記述する際に、その領域またはシステムのさまざまなパースペクティブに対応するモデルをいくつか使用する場合もあります。RUP では、以下のようなモデル・セットを処理します。

- ビジネス・ユース・ケース・モデル
- ビジネス分析モデル
- ユース・ケース・モデル
- 分析モデル (設計モデルの中に含めることもある)
- 設計モデル
- 実装モデル
- 導入モデル
- データ・モデル

RUP そのものは、ツールとは無関係です。RUP に関する限り、ナプキンやホワイトボードに描いた図であれ、モデリング・ツールで作成したファイルであれ、単に頭の中に思い描いたイメージであれ、すべてがモデルになります。つまり、RUP の観点からすれば、モデルとは論理的な概念です。RSA の文脈では、モデルについて論理的な観点から取り上げることもあれば、物理的な観点から取り上げることもあります。

たとえば、2 つのアプリケーションに関して 2 つのチームが作業しているとしましょう。1 つは、作業時間表管理アプリケーションの作業をしている 3 人の分析担当者からなるチーム、もう 1 つは、コール・センター・アプリケーションの作業としている 5 人の分析担当者からなるチームです。どちらのチームも、さまざまな要件を整理している段階であり、ユース・ケース・モデリングのために RSA を使用しています。RUP の観点からすれば、1 つのチームは「作業時間表管理アプリケーションのためのユース・ケース・モデル」を構築しているのであり、もう 1 つのチームは「コール・センター・アプリケーションのためのユース・ケース・モデル」を構築していることになります。しかし、RSA を使用している場合は、それぞれのモデルに物理的な側面があることを押さえておかなければなりません。その点を次のセクションで取り上げます。

モデリング・ファイル

RSA では、モデルをファイルとして保持します。(Eclipse の用語では、ファイルは「リソース」と見なされるので[1]、このホワイト・ペーパーや他の資料で「モデリング・リソース」という言葉が出てきたら、それは「モデリング・ファイル」と同じ意味の言葉です。)最も広い意味で、RSA は 2 種類のモデリング・ファイルをサポートしています。

- 「実装前」モデリング・ファイル。このファイルには、実装成果物を直接には反映しない概念的な UML の内容が含まれています。具体的には、モデルの UML セマンティクスと、そのセマンティクス要素を記述した UML 図の両方です。

- 実装モデリング・ファイル (Eclipse リソース)。Java ソース・ファイルや Web ページなどの通常の実装成果物であり、Eclipse プロジェクトの中に入っています。この場合のプロジェクトとは、実装モデリング・ファイルの内容の範囲全体を表したものと考えてください。モデルのセマンティクスは、実装成果物そのものの中に入っています。それぞれの図は、プロジェクト内の専用の物理ファイルの中に入っています。UML の表記法を使用した図もあれば、他の表記法 (データをビジュアル化するための IDEF1X や Information Engineering の表記法、Web 層を設計するための Rational 独自の表記法など) を使用した図もあります。3GL コード成果物を記述するためにサポートされている UML 図の種類としては、クラス図やシーケンス図があります (ただし、シーケンス図は Java の場合のみです)。

このホワイト・ペーパーで主に取り上げるのは「実装前」モデルの内部構造を編成する方法についてであり、**「モデリング・ファイル」という用語は「実装前」モデルの内容を含んだファイルを指しています。**実装プロジェクトの内容を編成するためのガイドラインについては、Rational Software Architect、Rational Application Developer、Rational Web Developer のオンライン・ヘルプなどの資料をご覧ください。

「実装前」モデリング・ファイルには、1 つのモデルに関するすべての情報が含まれているとは限りません。実際には、モデリング・ファイルにモデルのサブセットだけを含める場合もよくあります。たとえば、先ほどの例の場合、作業時間表管理アプリケーションのユース・ケース・モデルの作業をするチームには 3 人のメンバーがいます。そのようなチームでは、ユース・ケース・モデルを物理的に 3 つのモデリング・ファイルに分割し、それぞれのメンバーがユース・ケースの別々のサブセットの作業を行うようにして、同じファイルに関する作業の競合を避けようとするかもしれません。このホワイト・ペーパーの最後のセクションでは、モデルを分割して複数のモデリング・ファイルを管理することに関連した問題を取り上げます。

モデルのタイプ

RUP のモデルには、ユース・ケース・モデル、分析モデル、データ・モデルなどのタイプがあります。RSA のモデリング・ファイルにもタイプがあると言ってよいでしょう。つまり、各モデリング・ファイルに含まれているモデル (またはモデルのサブセット) のタイプが、そのモデリング・ファイルのタイプになるわけです。モデリング・ファイルのタイプを設定するには、2 つの方法があります。

- 最初は「ブランク」のモデリング・ファイルにしておき (以下を参照)、後から名前の付け方やファイルに含める内容 (ファイルに適用する UML プロファイルなど) によってタイプを設定します。

- 特定のモデル・タイプに相当する事前定義の「テンプレート・モデル」に基づいてモデリング・ファイルを作成します。いずれにしても、モデリング・ファイルの「タイプ」とは、実際にはファイルの内容に基づく便宜上のタイプにすぎません。たとえば、ツール自体は、ユース・ケース・モデルを含んだファイルに、ユース・ケースを実現するクラスを含めることを禁止するわけではありません。それでも、このホワイト・ペーパーで紹介するガイドラインでは、モデル・ファイルをタイプごとに扱うことを推奨しています。

[1] Eclipse ではリソースはファイルですが、Eclipse 環境内で付加的なプロパティーや動作を持っています。ここで説明しているモデリング・ファイルは、Eclipse によって「リソース」として取り扱われます。

ワークスペース、プロジェクト、プロジェクト・タイプ

Eclipse、WebSphere Studio 製品群、Rational Application Developer のいずれかに精通している読者であれば、各ファイルがプロジェクトに含まれていること、プロジェクトにはさまざまなタイプがあること、そして各プロジェクトがワークスペースの中で (仮想的に) グループ化されて管理されていることをご存知でしょう。RSA のモデリング・ファイルも他のファイルと同様にプロジェクトの中に入っています。

本書の目的からすれば、Rational Software Architect、Rational Application Developer、Rational Software Modeler に用意されているすべてのプロジェクト・タイプについてここで詳しく説明する必要はありません。最も広い意味で考えれば、ここで重要になるのは以下の 2 つのプロジェクト・カテゴリーです。

- **UML プロジェクト**
- **実装プロジェクト** (エンタープライズ・プロジェクト EJB プロジェクト、Web プロジェクト、C++ プロジェクトなどの特殊タイプも含む)

すでに述べたとおり、RSA は 2 種類のモデリング・ファイルをサポートしています。

- UML モデル (セマンティクス要素とそれを記述した図) を含んだファイル (実装の上にある抽象レベル (要件、分析、設計など) でモデリングを行うために使用する)
- 実装成果物 (Java コード、Web ページなど) を含んだプロジェクト (オプションとして、実装成果物を記述した図のファイルを含むこともある)

モデルをプロジェクトに割り振るためのルールは単純です。つまり、**a)** 「実装前」モデリング・ファイルは UML プロジェクトに組み込み、**b)** 実装モデルはそれ自体で独立させる、ということです。なぜならば、基本的に、以下の等式が成り立つからです。

[実装モデル] = [実装プロジェクト]

ところが、このルールにも例外があります。たとえば、以下の UML モデリング・ファイルは、言語固有の実装プロジェクトに組み込むのが適当です。

- 設計「スケッチ・モデル」(後述)
- プロジェクト内のコードに対して実行するテストを記述したシーケンス図を含んだモデル

> *XDE/Rose*
> Rose と XDE の操作理論には、設計モデルを反復的に洗練しながら最終的にコードに相当する抽象化レベルにまでもっていった後、コードとモデルの同期テクノロジーを使用してモデルのセマンティクスをコードそのものと一致させるという方法が含まれています。たとえば XDE では、実装モデルはプロジェクト内のコードおよび図として存在するだけでなく、「コード・モデル」ファイルとしても存在します。そのコード・モデル・ファイルは実装成果物とは独立して保持されるもので、本質的にはそれらの実装成果物の冗長なコピーを表しています。
>
> RSA の操作理論では、コードより抽象化レベルの高いプラットフォームに中立なモデル (つまり、Enterprise IT 設計モデルなどの設計モデル) を使用することや、変換機能を使用してこれらのモデルからコードを生成することが推奨されています。その後、コード・レベルの抽象化においてはコードの UML 図を作成できるだけで、別個に保持したセマンティクス・モデルを使用する方法は RSA では省略されています。
>
> ただし、RSA でも、コード・レベルの抽象化において UML モデルを定義し、それらのモデルからコードを生成することは不可能ではありません。実際、その種の使用法も想定されています。しかし、そのようなモデルをコードと同期させるテクノロジーを RSA は提供していません。この種の使用法は RUP の非常に「軽量な」バリエーションにだいたい対応しており、コードの生成が完了した後は、設計モデルは重要と見なされなくなります。

概念のまとめ

これまでの内容を以下の図にまとめます。コール・センター・アプリケーションの作業をするチームと作業時間表管理アプリケーションの作業をするチームがあるという、先ほどから取り上げてきたシナリオに基づく図です。

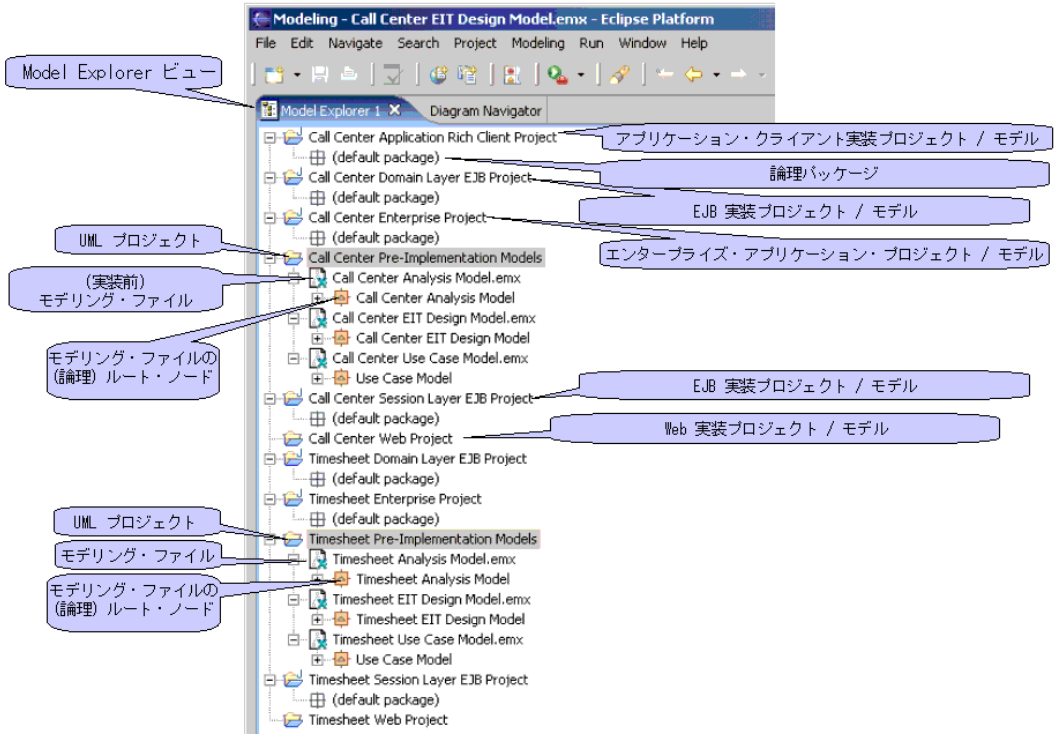


図 2-1

RSA には、モデルの物理ビューと論理ビューを組み合わせて表示する Model Explorer ビューがあります。Model Explorer では、ワークスペース内の各プロジェクトが最上位ノードとして表示され、各プロジェクトの中にそれぞれのリソースが表示されます。図 2-1 の Model Explorer では、先ほどのシナリオの 2 つのアプリケーションに相当するプロジェクト群が表示されています。ここでは、実装前モデルのために UML プロジェクトを使用しています。また、実装モデルのために、各ソリューションに適したタイプのプロジェクト群 (エンタープライズ・アプリケーション・プロジェクト、Web プロジェクトなど) を使用しています。

> *XDE/Rose*
> RSA の Model Explorer とは違って、Rose や XDE の Model Explorer はモデルの論理ビューしか提供しません。RSA の Model Explorer が提供するリソースのビューは、Eclipse Navigator のビューが提供する「純粋に」物理的なビューではないことに注意してください。一部の物理リソースは Model Explorer にも表示されますが、大部分はリソースの「論理的な」ビューを示すアイコンによって表現されます。

図 2-2 は、作業時間表管理のユース・ケース・モデルをパッケージの形で内部編成して、問題領域を機能面から分割したサブセットにそれぞれ対応させる方法を示した一例です。

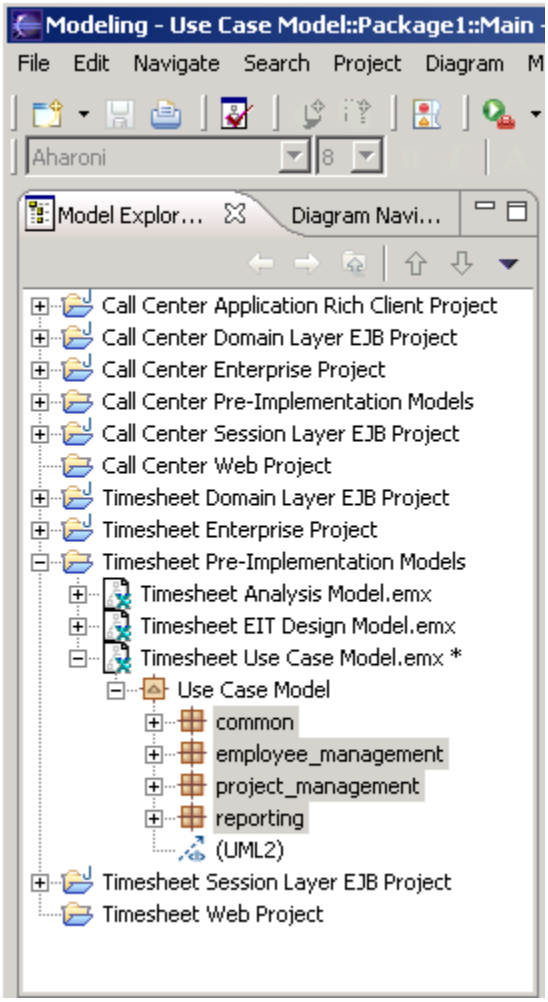


図 2-2

図 2-1 と**図 2-2** では、実装前モデルがそれぞれ 1 つのモデリング・ファイルに入っていますが、それを複数のモデリング・ファイルに分割することも可能です。たとえば、Timesheet (作業時間表管理) ユース・ケース・モデルを 4 つのモデリング・ファイルに分割し、それぞれを問題領域のサブセット (「common」、「employee_management」、「project_management」、「reporting」) に対応させるという方法があります。その場合、各モデリング・ファイルのルート・ノードには、そのユース・ケース・モデル全体を構成するすべてのモデリング・ファイルに関して一貫した名前空間を保持できるような名前を付ける必要があります。たとえば、その 4 つのモデリング・ファイルのルート・ノードに、「timesheet.requirements.common」、「timesheet.requirements.employee_management」、「timesheet.requirements.project_management」、「timesheet.requirements.reporting」という名前を付けるといった具合です。

3. RUP モデルと RSA モデルの対応関係

最もよく使われる RUP モデルと RSA モデル・タイプとの対応関係を以下の表にまとめます。この対応関係は基本的に単純明快ですが、RSA で RUP を実施するためのガイドラインとしてこのホワイト・ペーパーを活用するために、この対応関係を押さえておくことは重要です。この表で示す RSA モデル・タイプについては、すぐ後の部分で解説します。各種モデル・タイプの内部編成の方法や、各種モデル・タイプを組み込むプロジェクトの種類に関するガイドラインについては、後のほうのセクションで取り上げます。そのガイドラインも、ここに挙げる RSA モデル・タイプに基づいています。

RUP モデル	RSA モデル・タイプ
ユース・ケース・モデル	ユース・ケース・モデル
分析モデル	分析モデル (設計モデルに «analysis» パッケージとして組み込むこともできる)
設計モデル	n 層のビジネス・アプリケーション: エンタープライズ IT 設計モデル 他のタイプのアプリケーション: 設計モデルとして使用するブランク・モデル 設計「スケッチ」: 設計「スケッチ」モデルとして使用するブランク・モデル オプションの補足要素: 実装概要モデルとして使用するブランク・モデル
実装モデル	実装成果物と図ファイルを含んだ実装プロジェクト
導入モデル	導入モデルとして使用するブランク・モデル

RSA のモデル・タイプ

ブランク・モデル

RSA には、「ブランク・モデル」を作成するためのオプションが用意されています (「File」→「New」→「UML Model」→「Blank Model」)。「ブランク・モデル」とは、モデル・テンプレートに基づかないモデリング・ファイルです。特別なプロファイルは適用されていませんし、1 つの「Main」という名前の自由形式の図以外にデフォルトの内容もありません。**ブランクのモデリング・ファイルは、あらゆるタイプのモデルを作成するための出発点として活用できます。**ブランクのモデリング・ファイルにどんな名前を付けるか、どんな内容を定義するか、どんなプロファイルを適用するかによって、ユース・ケース・モデル、分析モデル、設計モデル、導入モデルなど、あらゆるタイプの RUP モデルを作成できます。

ユース・ケース・モデル

RSA には、モデル・テンプレートに基づいて「ユース・ケース・モデル」ファイルを作成するためのオプションが用意されています。このテンプレートのデフォルトの内容は、***図 3-1*** のとおりです。(「構成材料 (Building Block)」の内容や検索ストリングの使用方法に関する説明は、本書の守備範囲から外れています。テンプレートには説明や指示が含まれているので、ほとんどはそれで十分理解できます。)

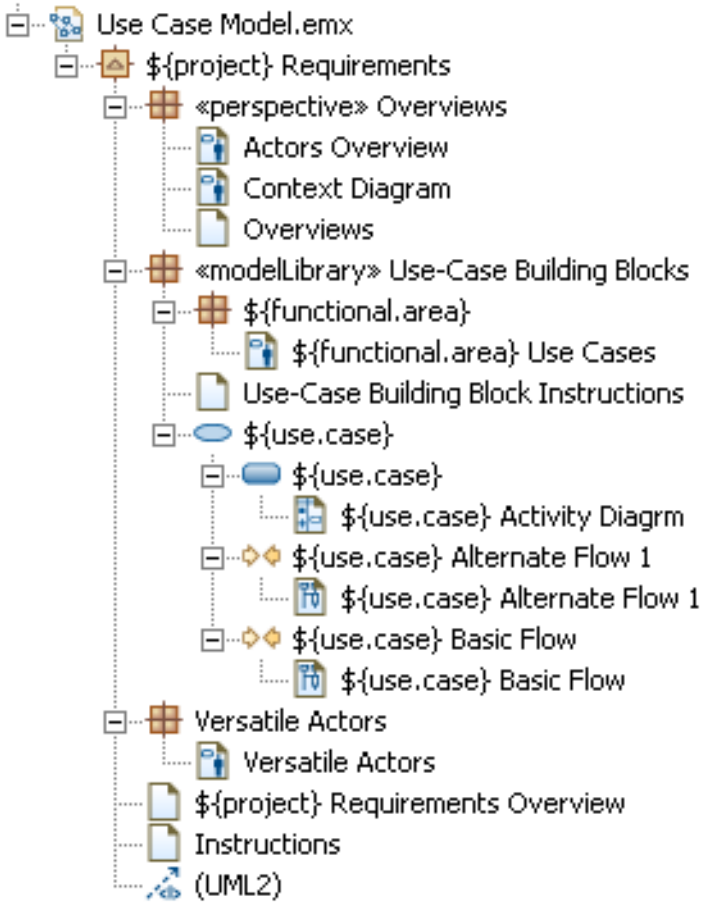


図 3-1

分析モデル

RSA には、モデル・テンプレートに基づいて「分析モデル」ファイルを作成するためのオプションが用意されています。このテンプレートのデフォルトの内容は、*図 3-2* のとおりです。さらに、このテンプレートから作成されたモデル・ファイルには「分析」プロファイルが適用されます。

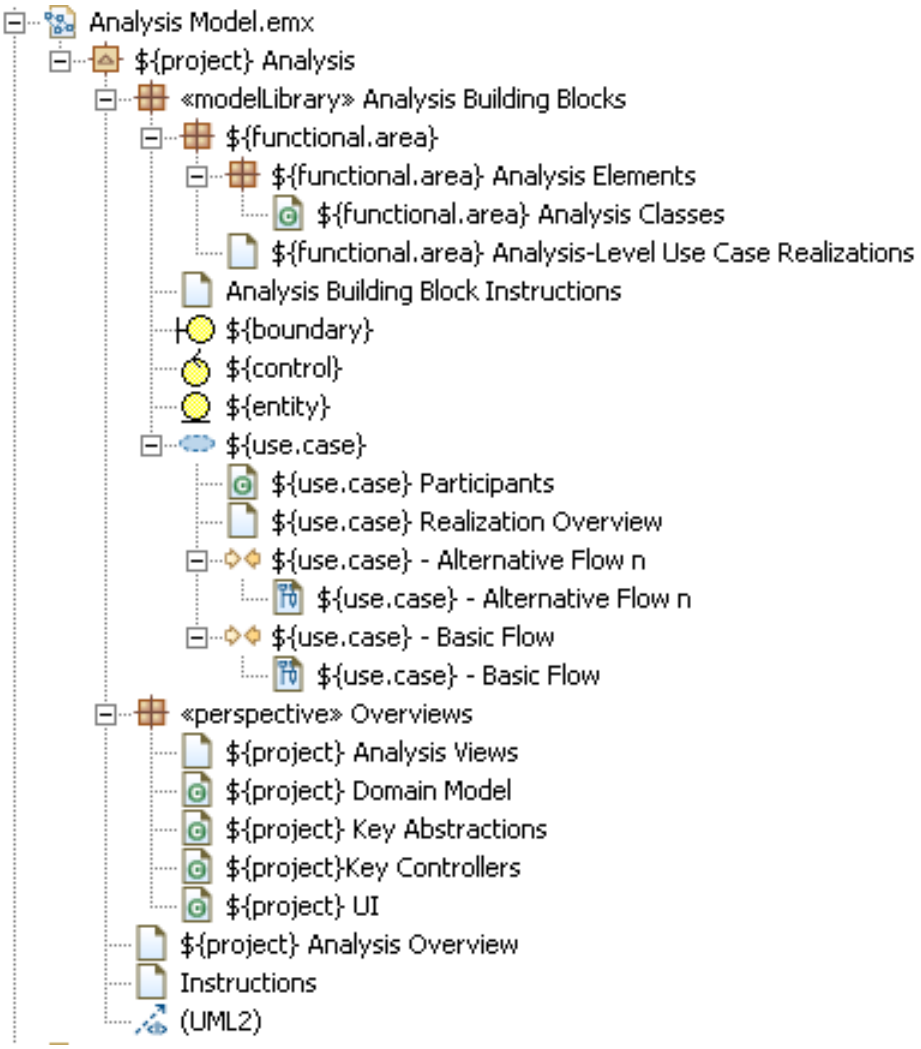


図 3-2

エンタープライズ IT 設計モデル

RSA には、モデル・テンプレートに基づいて「エンタープライズ IT 設計モデル」(EITDM) ファイルを作成するためのオプションが用意されています。このテンプレートのデフォルトの内容は、**図 3-3** のとおりです。さらに、このテンプレートから作成されたモデル・ファイルには「EJB 変換」プロファイル[2]が適用されます。目指すものがビジネス・アプリケーションであり、そのようなアプリケーションの作成をサポートするために RSA のコード生成変換機能を使用する場合は、設計用 (さらに、場合によっては分析用) としてこのテンプレートが適しています。

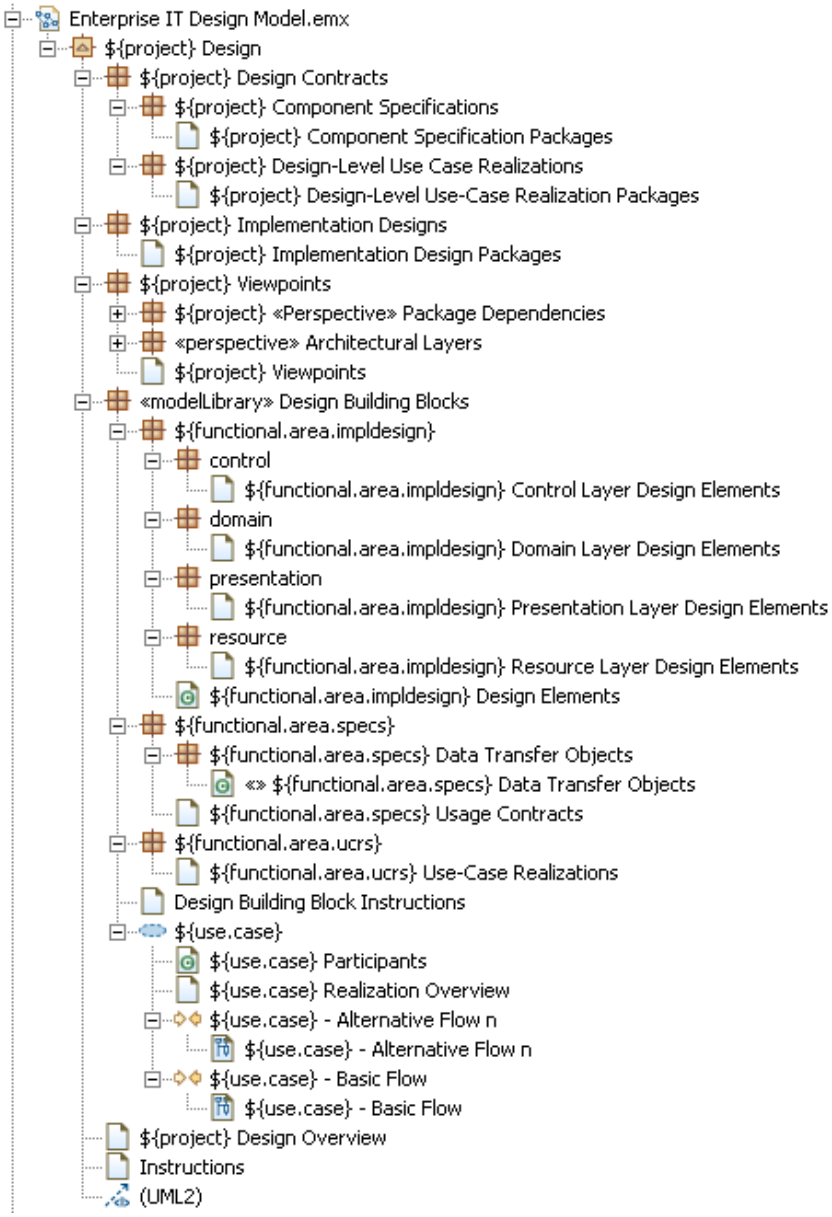


図 3-3

[2] EIT 設計モデル・テンプレートの一部として提供される、変換機能を可能にするプロファイル・セットは、製品のアップデート版がリリースされるたびに機能強化されているようです。

実装概要モデル

設計モデルの一部として、実装の編成方法の概要を取り込むための「実装概要モデル」をさらに定義するほうがよい場合もあります。「実装概要モデル」を使用するのは、設計の初期段階です。つまり、コードや関連ファイル (メタデータ、導入記述子など) を入れる実際の RSA プロジェクトやフォルダー/パッケージのためのコードの生成や記述を行う前に使用します。また、そのようなプロジェクトやパッケージの依存関係を示して、システム構築の要件を洗い出すためにこのモデルを使用することもできます。さらに、実装概要モデルは、ソリューション・アーキテクチャーのおおまかな概念図を入れておく場所にもなります。

実装モデル

すでに述べたとおり、RSA の実装モデルは、実装成果物と (オプションとして) その成果物を記述した図を含んだプロジェクトに相当します[3]。

「スケッチ」モデル

『基本的な概念と用語』のセクションでも述べたように、設計モデルを正式なアーキテクチャー図面として扱い、システムが存在する限り、アーキテクチャー制御のサポートや実施のために使用していく場合もあれば、単に説明や検討のための設計案を示したスケッチ程度に扱い、実際に設計から実装を作り出す作業が始まった時点で破棄するような場合もあります。RSA は、その両方の使用法をサポートしています。基本的には、その 2 つの方法に基づいて各種の機能が振り分けられているわけではありませんが、設計モデルをどちらの方法で使用するかに応じて、RSA のどの機能をどのように活用するかが決まってきます。このホワイト・ペーパーでは、ガイドラインを示すときにその区別が重要になってくる場合は、「スケッチ・モデル」という表現によって、おおざっぱな方法でモデルを使用することを示しています。

[3] これらの図を作成するには、「File」→「New」→「UML Model」を使用してモデルを作成するのではなく、「File」→「New」→「Class Diagram」を使用して、コードの「ビュー」を UML（またはその他の）表記で構築できる図を作成してください。個々の図は拡張子 .dnx のファイルとして別々に保持され、コード・ファイルとほとんど同じバージョン管理が可能です。これらの図には表記が含まれるだけで、セマンティクス情報はまったく含まれません。関連するセマンティクス情報はコードそのものの中にあります。 クラス名や操作のシグニチャーなど、これらの図にある要素を変更すると、実際には背後にあるコードそのものを変更していることになります。同じような変更を （テキスト・エディターを使用して） コードに加えると、変更したコードに対応する図が自動的に更新されます。

4. モデルの内部構造を編成するための一般的なガイドラインとテクニック

UML モデルの内容を編成するために主に使用するのは、パッケージです。UML のパッケージには、2 つの大きな目的があります。

- モデル情報の分割、編成、ラベル付け
    - 問題/解決領域の特定のテーマに基づいて各要素をグループ化すること
    - インターフェース、実装、図などのタイプごとにモデル情報を分類すること
    - 要素同士の依存関係を定義して制御するために各要素をグループ化すること
    - 同じモデルに関する複数の代替ビューを示した図をグループ化すること

- 名前空間の確立
    - モデルの各要素のための名前空間
    - モデルの各要素から生成される実装成果物のための名前空間 (場合によっては、モデルと実装の言語名前空間同士の対応関係も必要)
    - 再利用の単位のための名前空間

従来、RUP では、各種モデル・タイプに応じたパッケージ戦略を提唱してきました。本書のモデル・タイプ固有のセクションは、そのような戦略を反映しています。そのほかに、RSA では編成用のツールをさらに用意しており、その点についてこれから取り上げます。

«perspective» パッケージを使用してビューポイント（視点）を表現する

各要素を複数の方法で編成するのが望ましい場合は、代替の編成スキームを記述した追加のパッケージ（内容は図）を作成できます。このテクニックは、モデルの内容に関して、モデルのパッケージ・スキームを超えたビューを記述しなければならない場合にいつでも活用できます。RSA では、このテクニックをサポートするために、UML の「基本プロファイル」の一部として «perspective» パッケージ・ステレオタイプを用意しています。この «perspective» パッケージは、Systems Engineering または IEEE 1417 の「ビューポイント」にほぼ相当する RUP 機能と言えます。

«perspective» パッケージには、セマンティクス要素 (クラス、パッケージ、関連など) は入れません。むしろ、代替の編成条件やアプリケーションのビューポイントに基づいてビューを記述した図だけを入れます。«perspective» ステレオタイプをパッケージに適用することによって、いくつかの目的を達成できます。まず、特定のビューポイントを表現したものとしてそのパッケージをビジュアルに識別できます。また、«perspective» パッケージにセマンティクス要素が含まれている場合に警告を出すモデル検証ルールをサポートできます。さらに、RSA の変換機能がバイパスするべきパッケージであることを示すマーカーのような役割も果たします。

トピック・ダイアグラムを使用して特定の条件に関する自己更新型の記述を作成する

記述したい要素を手作業で追加していく「通常の」図とは対照的に、トピック・ダイアグラムの場合は、既存のモデルの内容に対するクエリーの実行によって内容を決定します。トピック・ダイアグラムを作成するには、「トピック」のモデル要素を選択してから、そのトピック要素との関連の種類に基づいて図の中に組み込む他の要素を定義します。モデルのセマンティクスの内容が変われば、それに応じてトピック・ダイアグラムも変わっていきます。

ブラウズ・ダイアグラムによってモデルを検討する

ブラウズ・ダイアグラムは、モデルの編成用に特化したツールではありません。この目的は、手作業で図を構成しなくてもモデル内容の発見と理解を可能にすることにあります。しかし、モデルの編成についても、ブラウズ・ダイアグラムを活用すれば、永続的な図を構成する必要が少なくなる可能性があります。そうなれば、モデルの全体的なサイズが縮小したり複雑さが解消したりするので、編成作業がいっそう容易になります。

ブラウズ・ダイアグラムは、トピック・ダイアグラムと似ているところがありますが、主な違いは、ブラウズ・ダイアグラムは永続しないという点です。つまり、常に何かの処理の実行中に生成されます。ブラウズ・ダイアグラムを生成するには、図または Model Explorer からモデル要素を選択してから、コンテキスト・メニューで「Explore in Browse Diagram」を選択します。その選択した要素を記述した図が「中心点」として生成され、その周りに関連要素が放射状に表示されます。もちろん、そのブラウズ・ダイアグラムに表示されている関連要素のいずれかを選択すれば、その要素が別のブラウズ・ダイアグラムの中心点になります。つまり、このような操作をいくらでも続けることができます。

図同士の間のナビゲーション

RSA には、図同士の間のナビゲーション・メカニズムが 2 つ用意されています。

- Model Explorer から 1 つの図ノードをドラッグして別の「ホスト」図にドロップできます。ホスト図の上にアイコンが表示されるので、そのアイコンをダブルクリックすれば、参照先の図を開けます。

- モデルの中に新しい UML パッケージを作成すると、「Main」という名前の自由形式の図が自動的に作成されます。デフォルトでは、この「Main」図がパッケージの「デフォルト」図になります。その図の名前を「Main」以外に変更しても、その図は引き続きデフォルトとして扱われます。さらに、パッケージの中の別の図を選択して、その図をパッケージの「デフォルト」図にすることもできます。いずれにしても、「デフォルト」図の目的は、パッケージ自体を別の「ホスト」図の上に置いた場合に、そのパッケージをダブルクリックすることによって、その「デフォルト」図を開けるようにすることです。

これらのメカニズムから、どんなタイプのモデルの編成にも適用できる以下の**ガイドライン**を導き出せます。

1. 各モデリング・ファイルの「Main」図 (または他のデフォルト図) を作成して、以下のものを記述します。
    a. モデリング・ファイル内の各最上位パッケージ
    b. モデリング・ファイルのルート・パッケージに存在する他のあらゆる図のアイコン (言い換えれば、デフォルト図自体のアイコンは記述しません)

2. 各最上位パッケージの「Main」図 (または他のデフォルト図) を作成して、以下のものを記述します。
    a. そのパッケージに直接含まれているパッケージ
    b. そのパッケージに直接含まれている他のあらゆる図のアイコン

3. 下位のパッケージに関してこの手順を繰り返していきます。

5. ユース・ケース・モデルの内部編成のためのガイドライン

メモ: これ以降のモデル・タイプ固有のいくつかのセクションでは、RSA に用意されているオークション・ショーケースのサンプルに基づくいくつかの例を取り上げながらガイドラインを示していきます。その他の編成上の詳細や図の内容については、サンプルをご覧ください。

ユース・ケース・モデルのおおまかな編成

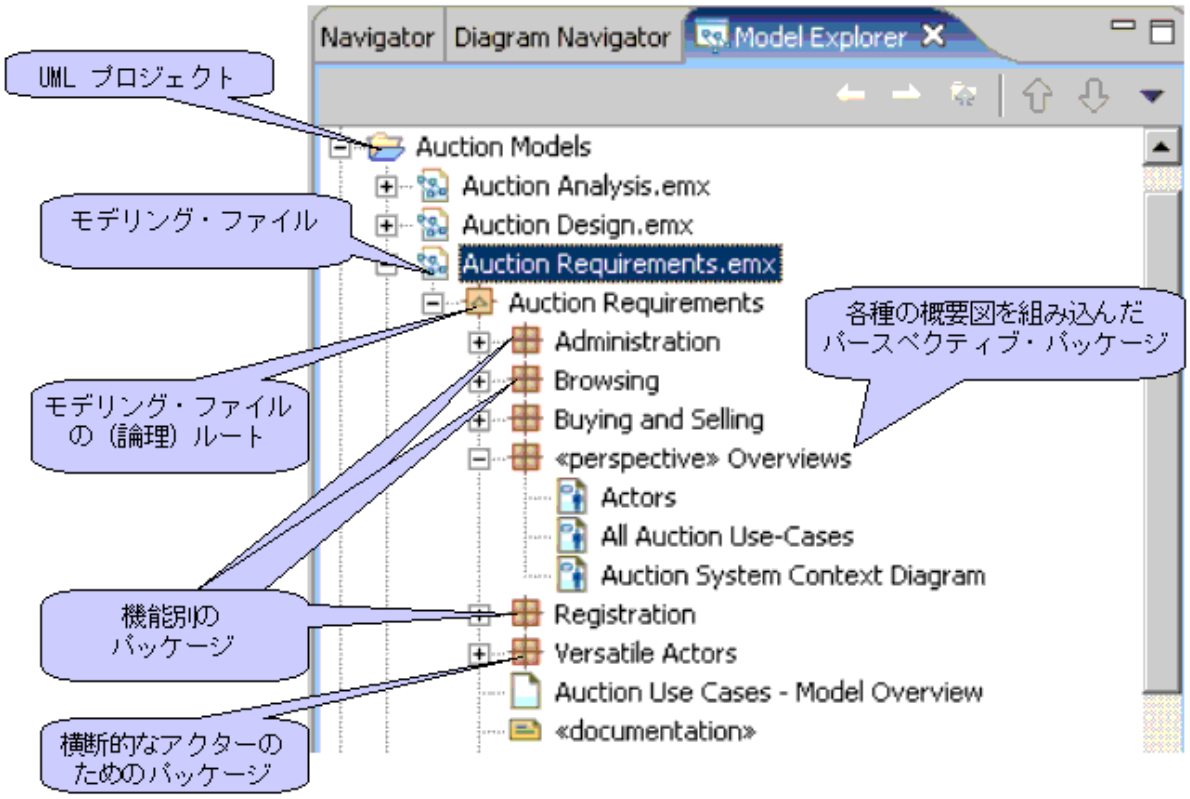


図 5-1

図 5-1 のユース・ケース・モデルの編成は、以下のガイドラインに基づいています。

1. 最上位のパッケージを使用して、機能別にグループ分けを行います。**理由は以下のとおりです。**

   - このグループ分けは、ユース・ケース・モデルの作業をチームを行うときの作業分担にほぼ相当します。また、ファイルの競合を避けるために後からユース・ケース・モデルを複数のモデリング・ファイルに分割する場合にも容易に対応できます (最上位のパッケージごとに別のモデリング・ファイルを作成するだけですみます)。

   - 他の編成方法に比べて、最終的な実装の編成内容との対応関係がすっきりします。特に、変換機能を使用して、下位の抽象レベルを段階的にシードしていく場合は、この点が重要になります。たとえば、ユース・ケース・モデルに基づいて分析モデルの中にシード内容を生成する場合は、ユース・ケース・モデルのパッケージ構造が対象の分析モデルの望ましいパッケージ構造によく対応するようにしておく

ということです。それをさらに進めて、分析モデルのパッケージ構造が設計モデルのパッケージ構造によく対応し、設計モデルのパッケージ構造が実装のプロジェクト・セットによく対応するように考えておきます。この対応関係がシンプルであればあるほど、1 つの抽象レベルから次の抽象レベルへの変換内容を構成する手間を省けるようになります。

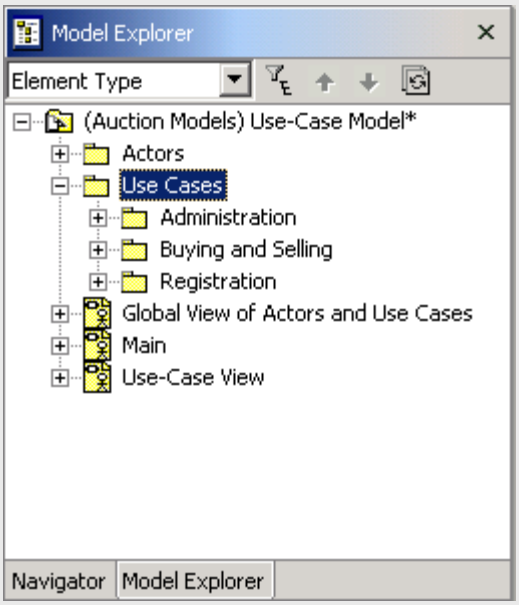
2. 別の最上位パッケージを使用して、用途の広い (汎用性の高い) アクターを取り込みます。

3. «perspective» パッケージの中の図を使用して、ユース・ケースの概略的な (横断的な) ビューを取り込みます。**理由は以下のとおりです。**

   - モデルのセマンティクス要素を機能別に編成する一方で、「アーキテクチャーの観点からして重要な」ユース・ケースの概略を示す横断的なビューを用意できます。

XDE/Rose

RSA のガイドラインでは、アクター用に 1 つのパッケージを作成し、ユース・ケース用にもう 1 つのパッケージを作成するというユース・ケースの上位レベルの編成に関する従来のガイドラインが少し改訂されています。そのため、必要なモデルのサイズや複雑さによっては、機能指向のグループ分けを実現するために下位レベルのパッケージを使用する必要があります。XDE ベースの下記の例を参考にしてください。



ユース・ケース・モデルの内容

ユース・ケースの良い書き方やユース・ケース・モデリングに関してすべきこと/すべきでないことを詳しく取り上げるのは、本書の守備範囲から外れていますが、アクターとユース・ケース以外にユース・ケース・モデルの中に含められる内容について、ここで簡単に触れておきます。

- **推奨:** モデルのルートに「メイン」の図を作成して、モデルの他のパッケージを記述し、それらのパッケージとそれぞれの「メイン」の図に対するドリルダウンをサポートします。

- **推奨:** 各ユース・ケース・パッケージに、そのパッケージのユース・ケースとユース・ケース同士の関係とユース・ケースにかかわるアクターを記述した図を組み込みます。(ユース・ケースの数が多ければ、複数の図を作成したほうがよい場合もあります。)

- **推奨:** 各ユース・ケースのメインのフローと代替のフローを「Documentation」フィールドに記述します[4]。(**図 5-2** を参照してください。)

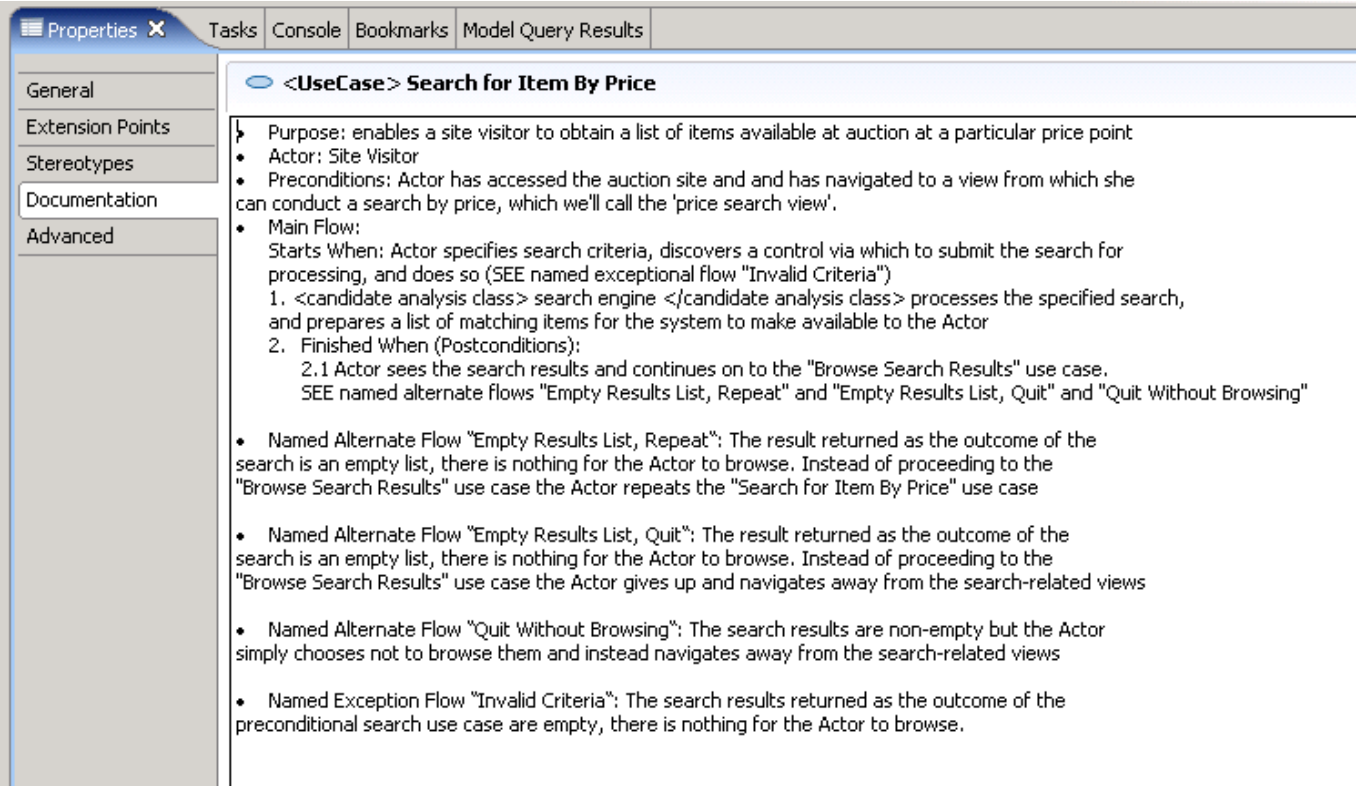


図 5-2

- **オプション:** ユース・ケースが複雑な場合は、必要に応じてアクティビティー図を追加し、ユース・ケースのアクティビティー・フロー全体を記述します。(**図 5-3** を参照してください。)**理由は以下のとおりです。**このようにすれば、メインのフローや代替/例外のフローのそれぞれに対応する状態を示して、各種のフローをすべて最終的に一本化するのに役立ちます。(RSM/RSA でアクティビティー図を追加すると、ユース・ケースにアクティビティーが自動的に追加され、そのアクティビティーの下に図が組み込まれます。)

- **オプション:** ユース・ケースに含まれる名前付きのメイン/代替/例外フローのそれぞれに関する「ブラック・ボックス」的な実現内容をモデリングし、ユース・ケースにコラボレーション事例を追加し、そのコラボレーション事例にユース・ケースのメイン・フローに対応する相互作用インスタンスと名前付きの代替/例外フローのそれぞれに対応する相互作用インスタンスを追加し、それぞれの相互作用インスタンスのシーケンス図 (またはコミュニケーション図) を作成します。このユース・ケースの相互作用インスタンスは、分析モデルで記述する分析レベルのユース・ケース実現内容や、設計モデルで記述する設計レベルのユース・ケース実現内容と混同するべきではありません。それらはユース・ケースの「ホワイト・ボックス」的な実現内容であって、ソリューショ

[4] ユース・ケース記述の例に示されているフォーマットを実現するには、RTF 対応のエディターでユース・ケース記述用のテキスト「テンプレート」を作成し、ユース・ケース記述フィールドにそのテンプレートをコピー & ペーストします。

ンの内部要素間の相互作用を記述したものです。ここで示しているユース・ケース・モデルのコラボレーション事例は、アクターとシステムの間のきわめて「ブラック・ボックス」的な相互作用です。(**図 5-3** を参照してください。)**理由は以下のとおりです。**このようすれば、非技術系の利害関係者のために、ユーザーがどのようにシステムとかかわるようになるのかという意味での相互作用の概略を示すことができます。また、実装に含めなければならない各種のビュー (画面やページ) を洗い出すためにも役立ちます。さらに、ユース・ケースの各種のフロー (シナリオ) に名前を付けて、その名前をセマンティクス要素 (つまりコラボレーション事例) に割り当てることによって、名前の体系を正式に確立することにもつながります。

> *XDE/Rose*
> UML 1.x では、「コラボレーション事例」の代わりに「コラボレーション・インスタンス」を使用していました。

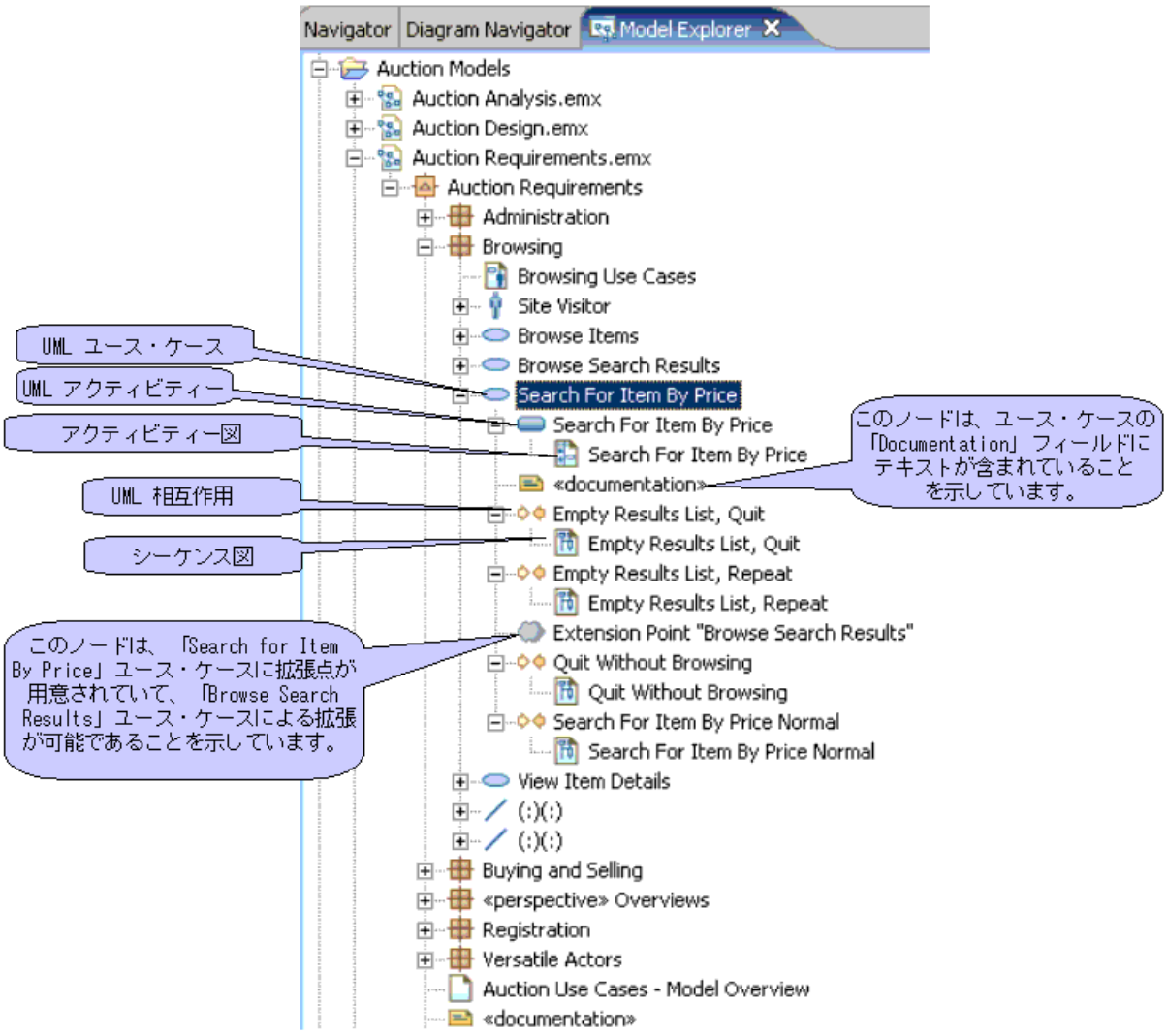


図 5-3

- **オプション:** 「アーキテクチャーの観点からして重要な」ビューを洗い出すために RUP のガイドラインを活用している場合、特にソフトウェア・アーキテクチャー説明書を活用している場合は、最上位の «perspective» パッケージを追加して、アーキテクチャーの観点からして重要なユース・ケースを記述したユース・ケース図を組み込みます。そのパッケージには、「Use-Case View of Architecture」などの名前を付けます。

6. 分析モデルの内部編成のためのガイドライン

ソリューションを映画に例えれば、分析モデルは最初のシーンのようなものです。要件から最終的な設計を生み出すための準備段階であり、主にビジネス領域に関する情報を収集し、ビジネスに近い上位の抽象レベルでソリューションの各要素の候補を示します。分析モデルには、分析のための各クラスと、分析レベルのユース・ケース実現内容を組み込みます。ソリューションに必要なクラスを洗い出す作業は、ユース・ケース実現内容をモデリングするプロセスから始めます。このプロセスには、主にシーケンス図を使用します。特に、シーケンス図で必要なライフラインを見つけ出し、そのライフラインに相当するクラスを組み込んでいきます。さらに、ユース・ケース・モデルの内容に基づいて分析モデルの内容を考えるときに適用できる経験則がいくつかあります。その点についても、このセクションの後のほうで触れることにします。

RUP で分析モデルを設計モデルから切り離して管理するかどうかは、プロジェクトごとに決定する事柄です。分析モデルを別個に管理すればそれなりの時間がかかるので、その時間に見合うだけの価値があるかどうかを検討しなければなりません。別個の分析モデルを作成するとはいえ、その分析モデルを保持しない場合は、分析クラスを設計モデルに移して加工することになります。あるいは、分析モデルを段階的に発展させて設計モデルを作り出すという方法もあります[5]。製品固有の観点から検討できるいくつかのオプションを以下に示します。

1. 分析モデル・テンプレートに基づいて分析モデルを作成し、それを 1 つのモデリング・ファイルまたは一群のモデリング・ファイルに組み込みます。次に、エンタープライズ IT 設計モデル・テンプレートに基づいて、手作業か自動変換機能によって別のモデル・ファイルまたは別の一群のモデル・ファイルに洗練された分析要素を作成してから、分析モデリング・ファイルの扱いを決めます。この場合は、別個の分析モデルを保持するか破棄するかを選択できます。
2. エンタープライズ IT 設計モデル・テンプレートに分析プロファイルを適用して、1 つのモデリング・ファイルまたは一群のモデリング・ファイルで分析レベルのモデリングを行います。この場合は、分析クラスを使用してユース・ケース実現内容のモデリングを開始してから、設計インターフェースにさまざまな動作の役割を引き継ぐために時間をかけて洗練していきます。
3. 1 番目と 2 番目のオプションを組み合わせて、ある種の分析モデルを設計モデルと同じモデリング・ファイル内に保持するという方法もあります。そのためには、分析の内容を «analysis» というキーワードの付いたパッケージに分離します。このようにして、分析レベルの成果物をより洗練された設計レベルの成果物と同じモデリング・ファイル内に保持できます。

RSA の変換機能を使用して実装を生成する場合に覚えておくべき点は、分析レベルの要素を変換機能の入力データとして使用できる場合が多いということです。そのような場合は、分析レベルの要素を設計の要素に洗練するための手作業の一部が不要になります。このような方法で RSA を使用する場合は、上記のオプション 2 か 3 が望ましいと言えます。RSA の一部として組み込まれている標準のコード生成変換機能は、«analysis» キーワードの付いたモデル・パッケージをバイパスするからです。

[5] RUP は実際には、分析クラスと分析レベルのユース・ケース実現内容を設計モデル内に作成するオプションを呼び出した後、直接それを設計の形式に発展させます。この方法では、設計モデルが「発見」されたときに、「純粋な分析」の観点をいくらか残す方法でパッケージを作成できます。

分析モデルのおおまかな編成

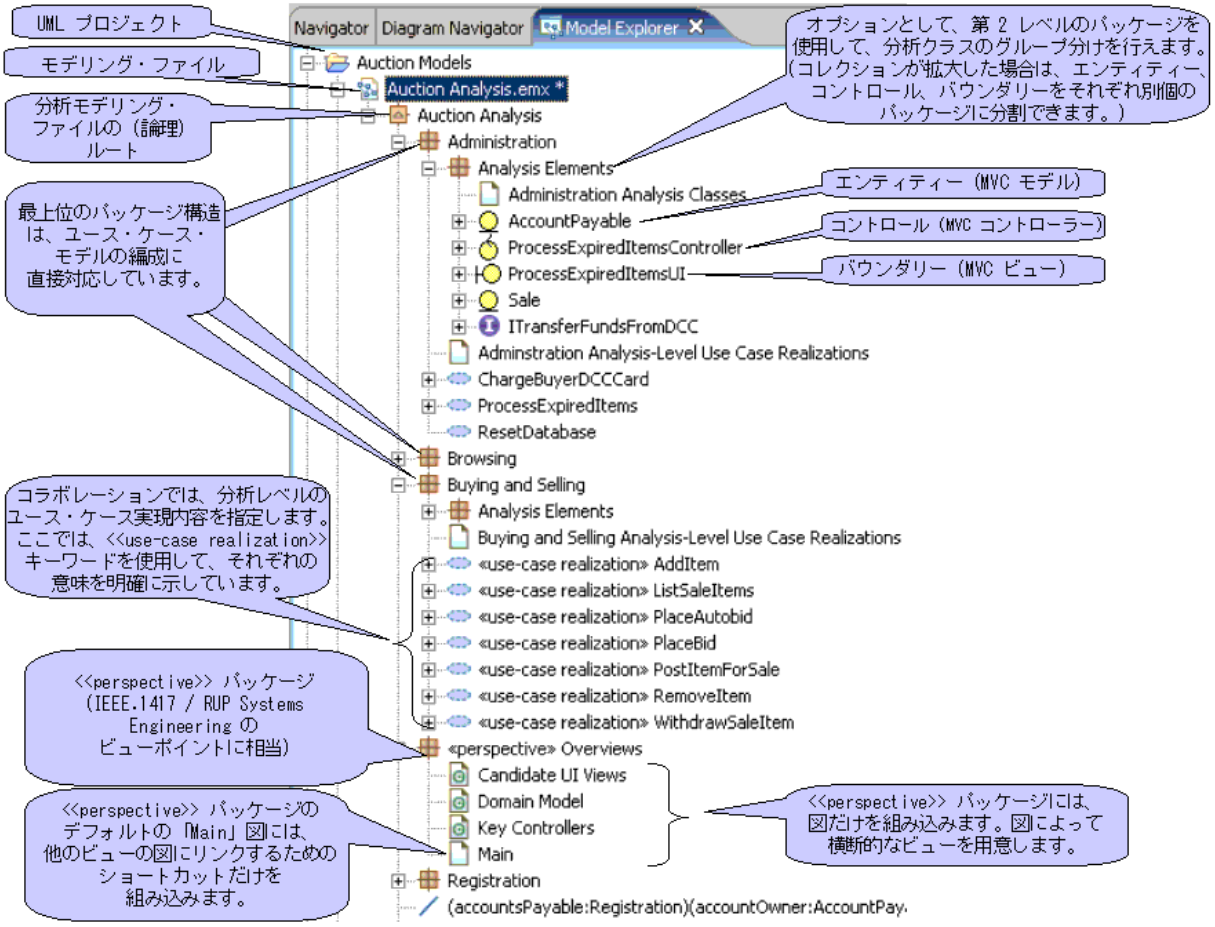


図 6-1

図 6-1 の分析モデルの編成は、以下のガイドラインに基づいています。

1. 最上位のパッケージを使用して、分析クラスを機能別にグループ分けします。**理由は以下のとおりです。**ユース・ケース・モデルの場合と同じ理由です。

2. オプションとして、最上位パッケージの中で、分析クラスを集めて編成するためのサブパッケージを使用します。

3. «perspective» パッケージの中の図を使用して、分析の要素の概略的な (横断的な) 代替ビューを取り込みます。**理由は以下のとおりです。**そのようにすれば、モデルのセマンティクス要素を機能別のグループに編成する一方で、さまざまな利害関係者のさまざまなパースペクティブ（観点）を用意できます。

この方法に少し手を加え、最上位のパッケージを使用して、分析クラスからユース・ケース実現内容を分離したのが、**図 6-2** です。ここでは、その最上位パッケージの中に、すべての最上位パッケージに対応する機能別のサブパッケージ群を組み込んでいます。このようにしてユース・ケース実現内容を分離することによって、ユース・ケース実現内容の編成に必ずしも影響を与えずに、分析クラスを含んだパッケージ構造を分割できます。(特に、分析モデルをそのまま発展させて設計モデルを作り上げる場合は、クラスのパッケージ編成がユース・ケースの元の編成とは食い違うような形で発展していく可能性があります。)

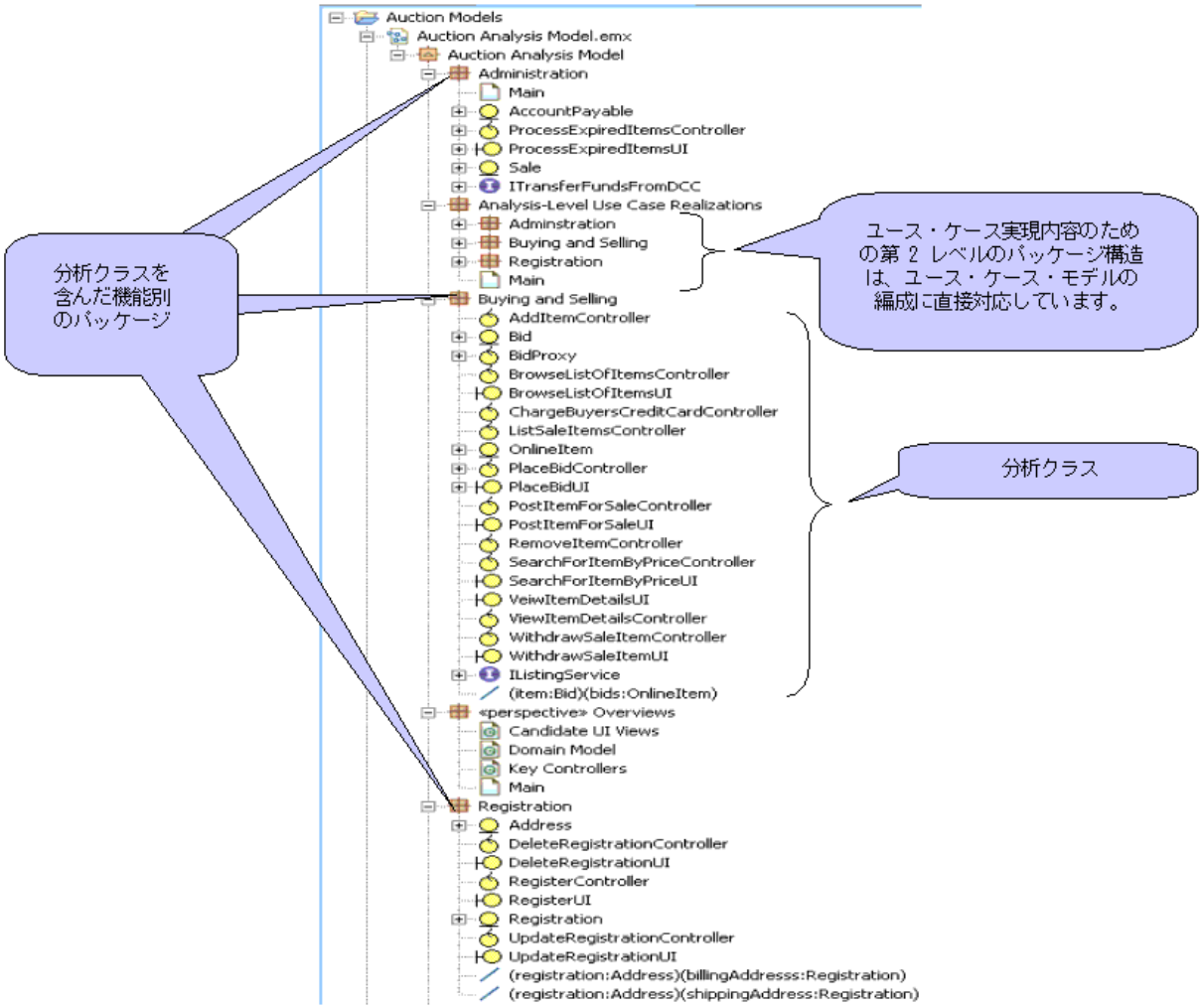


図 6-2

別のパートナー企業のグループなども含めてさまざまなグループが作業している状況では、それぞれのグループが作成するモデルの内容をマージしたり再利用したりする場面を最初から想定し、それに合わせた名前付けの体系を考えておくほうがよい場合もあります。その場合は、**図 6-3** のような、インターネット・ドメインの名前空間を反転させた名前付けの体系を使用できます。もちろん、このこと自体は分析モデリングの大きな問題ではありませんが、分析モデルをそのまま設計モデルに発展させる場合に、設計レベルでの再利用やビジネス統合を想定しているのであれば、最初から計画を立てておくのが望ましいと言えます。さらに、分析/設計から生成するコードの編成方法との対応関係がすっきりするので、コード生成変換機能を構成するときにその作業を簡略化できるというメリットもあります。

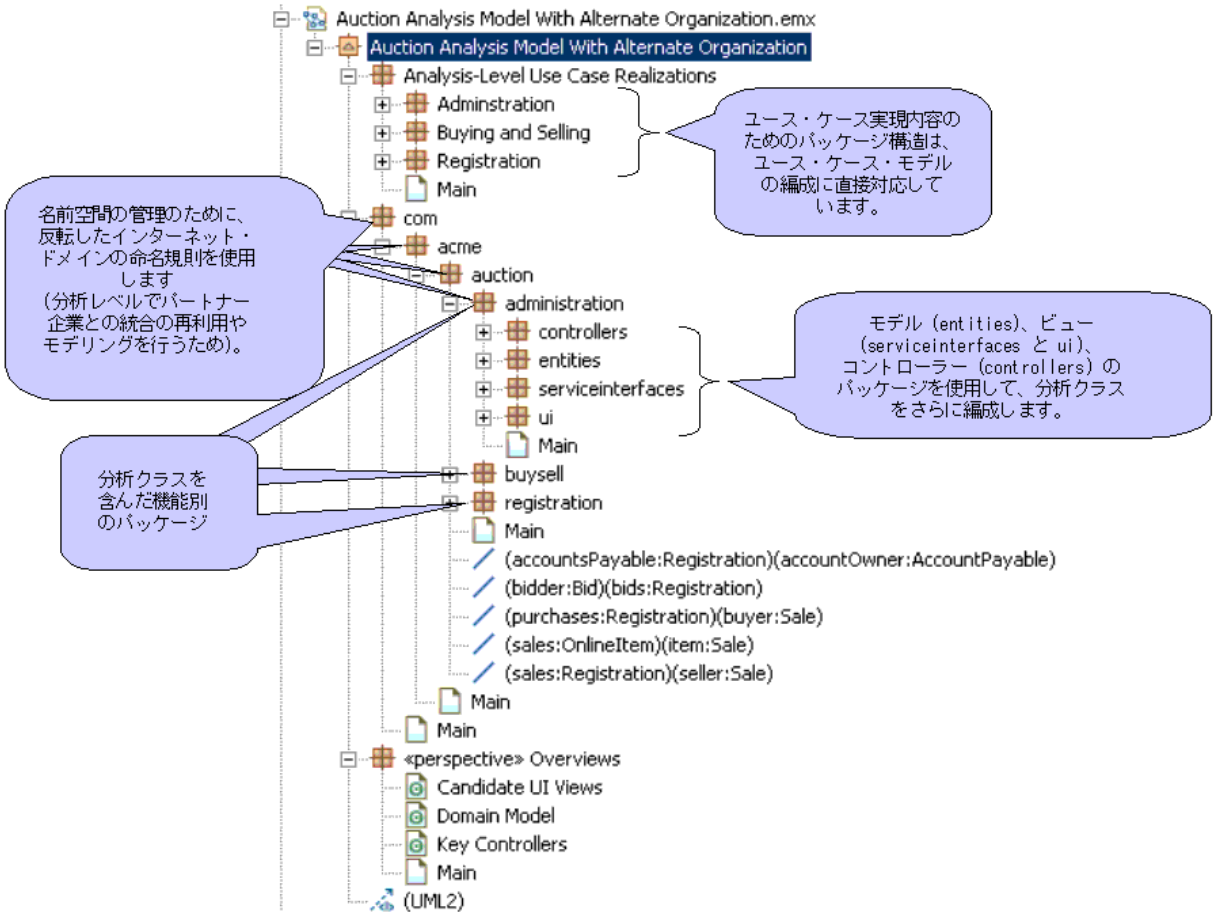


図 6-3

分析モデルの内容

分析クラスを見つけ出す方法は、いくつかあります。1 つは、ユース・ケース実現内容を示すシーケンス図を描くという方法です。そのシーケンス図から必要なライフラインを見つけ出せば、基本的にはそのライフラインが分析クラスの候補になります。このようにしてクラスを見つけ出した場合は、分析モデルのユース・ケース実現内容パッケージの中にそれらのクラスを作成できますが、それをそのままにしておくべきではありません。モデルを分割して、分析モデルのおおまかな編成に関するガイドラインのセクションですでに述べた機能別のパッケージに分析クラスを移すのが望ましいと言えます (***図 6-1*** を参照してください)。

分析クラスを見つけ出すためのもう 1 つの便利な方法は、以下のような経験則に基づいて、分析モデルにクラスを「シード」することです。

- ユース・ケース・モデルのユース・ケースごとに、それぞれ対応する «control» クラスを分析モデルに追加します。«control» クラスは、ユース・ケースに関連するビジネス・ロジックに対応します。(後の設計段階では、セッション管理などの問題点にも対応することになります。)

- ユース・ケース・モデルのアクター/ユース・ケースの関係ごとに、それぞれ対応する «boundary» クラスを分析モデルに追加します。«boundary» クラスは、ソリューションと人間のアクターとの間、またはソリューションと外部のシステムとの間のインターフェースに対応します。人間のアクターに対応する «boundary» クラスは、

基本的に設計や実装の段階でユーザー・インターフェース成果物に対応することになります。外部のシステムに対応する «boundary» クラスは、基本的に設計や実装の段階である種のアダプター層に対応することになります。

- CRC カード分析やユース・ケース記述のワード分析などのプロセスによって、その他の «control» クラス (動詞) と «entity» クラス (名詞) を識別します。

このシード方法によって分析クラスを識別する場合は、分析モデルのおおまかな編成に関するガイドラインのセクションですでに述べた機能別のパッケージに分析クラスを直接配置できます (***図 6-1*** を参照してください)。

どんな方法で分析クラスを見つけ出すとしても、ほとんどの場合は、元の機能別のパッケージ編成を変更する必要があります。

オプション: 分析クラス・パッケージ内の第 2 レベルのパッケージを使用して、パッケージの内容をさらに編成します (**図 6-4** を参照してください)。

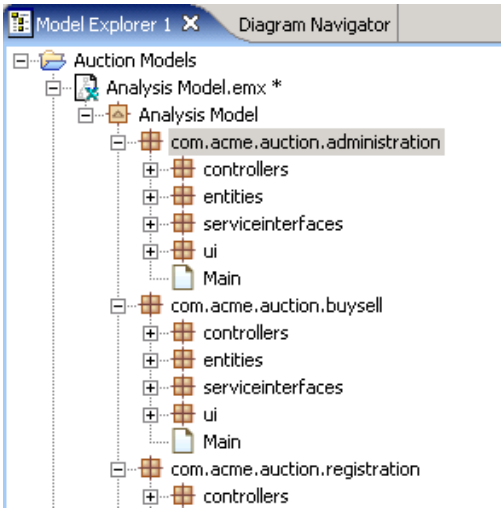


図 6-4

推奨: 分析モデルには、分析クラスの観点からユース・ケースの実行方法を記述した分析レベルのユース・ケース実現内容を組み込むべきです。UML のコラボレーションで表現する分析レベルのユース・ケース実現内容はそれぞれ、ユース・ケース・モデル内の 1 つのユース・ケースを実現したものであり、そのユース・ケースと同じ名前になります。**図 6-5** を参照してください。分析レベルの実現内容としてモデリングするべき名前付きのユース・ケース・フロー[6]ごとに、それぞれ対応するシーケンス図を追加します (そうすると、所有側の相互作用が自動的に追加されます)。シーケンス図を作成するときにモデルに追加されるセマンティクス内容のタイプについては、**図 6-6** を参照してください。(Model Explorer ビューでは、UML 要素のタイプに基づくフィルターを設定して表示内容を絞り込み、**図 6-6** にあるような雑然とした表示をすっきりさせることもできます。)

[6] ユース・ケース・モデル内に以前に確立したもの。

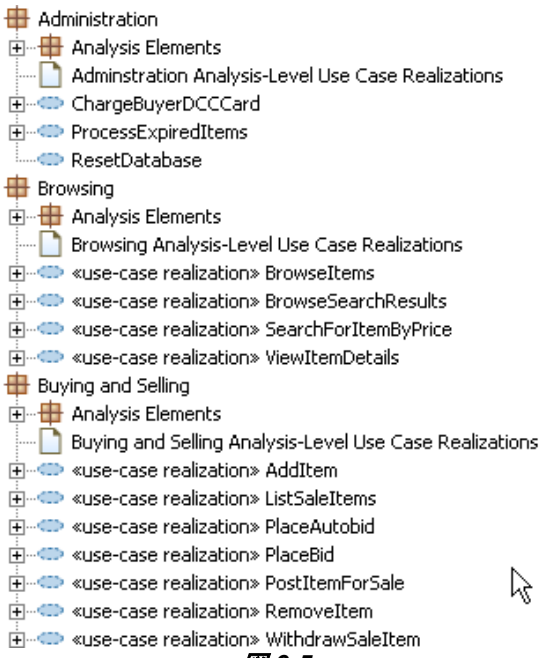


図 6-5

オプション: ユース・ケース・フローのシーケンス図を作成したら、Model Explorer でその所有側の UML 相互作用を選択し、それにコミュニケーション図を追加できます。その新しいコミュニケーション図には、シーケンス図にかかわっていた分析クラスのインスタンスが自動的に組み込まれます。

推奨: 各ユース・ケース実現内容から実現内容の依存関係 (UML コラボレーション) を作成し、ユース・ケース・モデルからそれに対応するユース・ケースを作成します (**図 6-6** を参照してください)。トピック・ダイアグラムや追跡可能性分析などの機能を使用すればモデル内の追跡可能性関係を把握できるので、追跡可能性関係を記述するための永久的な図を保持する必要はありません。むしろ、何かの「使い捨て」の図を使用してその関係を作成することをお勧めします。たとえば、以下のような方法があります。

- 自由形式の図をコラボレーションに追加します。
- その上にコラボレーションをドラッグします。
- その上にユース・ケースをドラッグします。
- 依存関係を記述します。
- 最後に、Model Explorer でコラボレーションからその図を削除します。

推奨: ユース・ケース実現内容ごとに、それぞれに対応する「参加者」図を組み込んで、実現内容に加わっている分析クラス (つまり、そのユース・ケースの実現内容を記述した相互作用図にインスタンスが登場する分析クラス) と、相互作用図に記述されているコラボレーションをサポートするクラス間の関係を示します。**図 6-6** を参照してください。

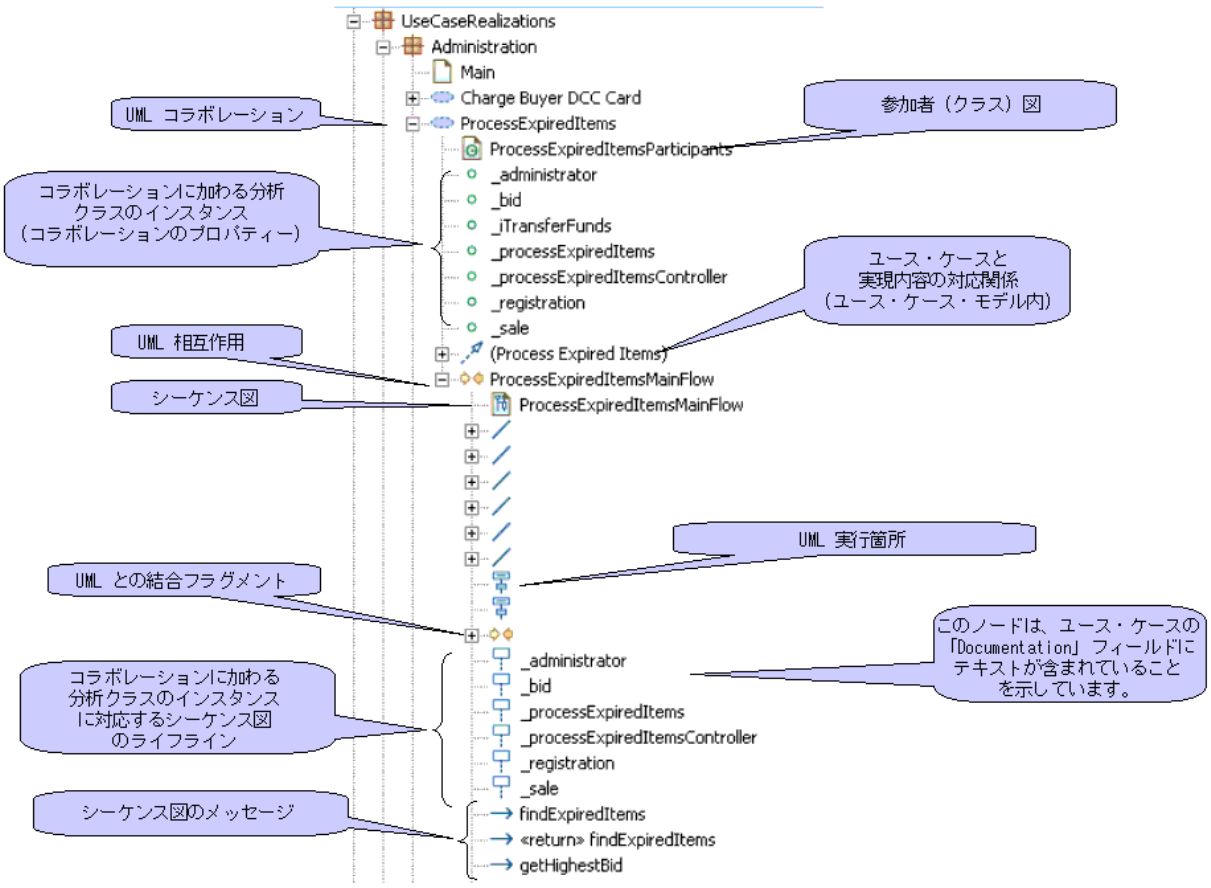
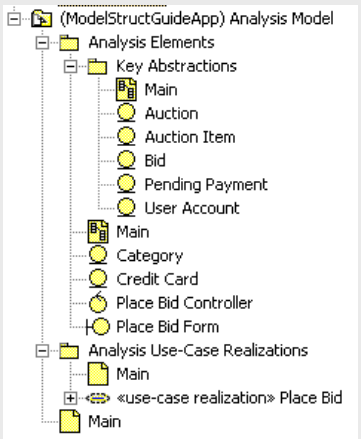


図 6-6

XDE/Rose

これまで推奨されていた**分析モデル**の構造 (下記参照) が RSA では修正され、分析クラスを機能指向のパッケージ編成にすることが強調されるようになりました。また、Key Abstractions パッケージを使用する代わりに (機能指向のパッケージ方法と矛盾してしまうので)、«perspective» パッケージ内で Key Abstractions 図 (1 または複数) を使用することになりました。



7. 設計モデルの内部編成のためのガイドライン

設計モデルのおおまかな編成

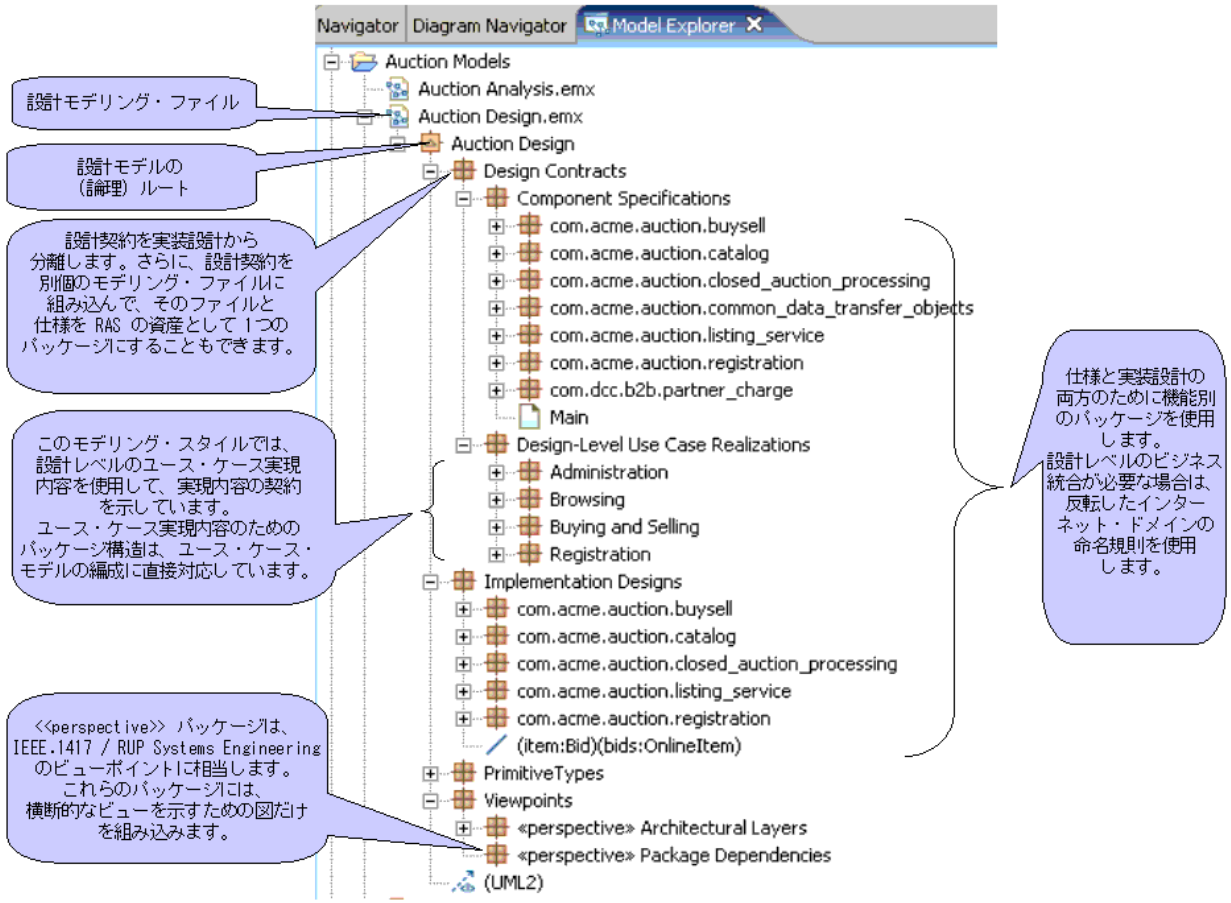


図 7-1

図 7-1 の設計モデルの編成は、以下のガイドラインに基づいています。

1. 仕様と実装設計を分離します。この図では、そのために「Design Contracts」パッケージと「Implementation Designs」パッケージを最上位に置いています。

2. 下位レベルのパッケージを使用して、機能別のグループを設定します。たとえば、最初は分析段階で使用した編成から始め、分析クラスを実際の設計クラスやコンポーネントやサービスに対応付ける方法を決めながら、その編成を発展させていくという方法もあります。(初期の編成スキームは、いずれにしても設計段階で発展していく場合が多いと言えます。その点については、以下の説明を参照してください。)

   ここで、サブシステムのことに少し触れておきたいと思います。バージョン 2 より前の UML では、サブシステムは特別なタイプのパッケージでした。UML 2 の場合、サブシステムは特別なタイプのコンポーネントであり、そのコンポーネントにパッケージを組み込むという形になります。UML 2 では、パッケージの代わりになる有

効な編成手段/名前空間として «subsystem» コンポーネントを使用できますが、サブシステムとパッケージの使い分けの方法については、UML 2 は特に具体的なことを定めていません。1 つの方法として、エンタープライズ・レベルのアーキテクチャーでは、アプリケーションの設計サブシステム程度の細分化レベルでパッケージを使用し、アプリケーション全体 (CRM や SCM など) に対応するレベルでサブシステムを使用することをお勧めします。

> *XDE/Rose*
> 執筆の時点では、Rose および XDE のモデル・インポート・ツールが UML 1.x のサブシステムを UML2 のサブシステムか «subsystem» キーワードが適用されたパッケージにマッピングするオプションを提供する予定であるとされていました。

3. 設計要素の編成は、システムのユース・ケースの編成 (つまり、ユース・ケース・モデル内での編成や、別個の分析モデルを保持する場合は分析モデル内での編成) とは別個に発展する場合が多いと言えます。そこでパッケージを使用し、設計の契約内容をさらに設計要素仕様 (使用法の契約の場合) や設計レベルのユース・ケース実現内容 (実現内容の契約の場合) に分割して、ユース・ケース実現内容のパッケージ下部構造とユース・ケースそのものの編成との対応関係を維持します。

4. 機能領域の仕様と実装設計を構成する要素の第 2 レベルの編成スキームの基礎としてアーキテクチャー層を使用することを検討します (詳細については、以下の説明を参照してください)。

5. セマンティクス・モデル要素をグループ分けした UML のコンポーネントとパッケージの中に、各グループに固有のビューを示した図を組み込みます。このガイドラインは、そのグループ分けがビジネス領域の機能別のサブセットに基づいているか、アーキテクチャー層に基づいているか、その他のものに基づいているか、という点にかかわっています。そのパッケージまたはコンポーネントと同じ名前の「デフォルト」図を作成し、それぞれのおおまかな内容を示すようにしてください。そのようにしていくつかの図の名前と内容を一致させておけば、モデルのナビゲートが容易になり、全体像を把握するのも楽になります。

6. 反転したインターネット・ドメインの名前空間を設計モデルで使用することもできます。**理由は以下のとおりです。**
   - 基本的に、言語固有の実装の場合と同じ理由です。
     a. 複数のモデル駆動型アプリケーションの統合作業がかかわっているシナリオ (特にパートナー企業がかかわっている場合)
     b. 再利用のシナリオ
   - 多くの場合、実装への変換を構成する作業 (場所や名前に関するソースと宛先の対応関係の設定) がシンプルになります。

7. 名前空間の対応関係を設定する際の手間や混乱を避けるために、対象の実装プラットフォームで有効なパッケージ名を使用することを検討します。(簡単に言えば、要は「名前に下線以外の句読点やスペースを使わない」ということです。)

8. パッケージ名には小文字を使用して、パッケージ名とパッケージ内のクラス名を区別します。

9. インターフェースとそれを実現するコンポーネントやクラスには別々の名前を使用することを検討します。たとえば、インターフェースと実装の名前に ILoan と Loan を使用するか、Loan と LoanImpl を使用する、といった具合です。モデルの中でそのようにする必要があるわけではありませんが、基本的にコード生成のためには便利な方法です。つまり、この点でも変換機能の構成作業の手間をいくらか省けるということです。

10. 以下のようなシナリオでは、コード生成に使用しない分析レベルの内容を «analysis» ステレオタイプのパッケージ内に分離するべきです[7]。

    - A) 別個の分析モデルの使用を省略する必要があり、設計モデルに分析レベルの内容を取り込み、その内容を抽象化の分析レベルに保つと同時に、同じモデル内に設計レベルの内容を作成する必要がある場合。

    - B) なおかつ、モデルからコードへの変換機能を EIT 設計モデルから実行することになる場合。

11. «perspective» パッケージの中の図を使用して、設計要素の概略的な (横断的な) ビューを取り込みます。理由は以下のとおりです。モデルのセマンティクス要素を機能別に編成する一方で、「アーキテクチャーの観点から重要な」内容や様々なタイプの利害関係者を納得させる観点を示す横断的なビューを用意できます。

設計モデルのパッケージ構造は時間の経過とともに発展していくものであると認識することが大切です。最終的な編成は、アーキテクチャーをコンポーネントとサービスに構造化する方法と対応させる必要があります。この方法で設計の「最終段階」へと発展させていけば、一般に、パッケージを再利用可能な資産にできる可能性が高まり、設計と、その設計から生成する実装成果物 (コード、メタデータ、文書) を入れる一式のプロジェクトやフォルダーとを最も分かりやすくマッピングすることができます。

とはいえ、「初期段階」の編成は多かれ少なかれ、ユース・ケース・モデルの作成に使用した編成方法に依存したものになるでしょうから、分析中に改訂していくことになります[8]。実際、『分析モデルの内部編成のためのガイドライン』のセクションで前述したとおり、分析モデルを発展させて所定の設計にしていく方法を選ぶことができます。言い換えると、設計の初期編成では凝集したビジネス事項と疎結合のビジネス事項がグループ化される傾向があり、そこから横断的な要素または再利用可能な要素を分離していきます。この方法による初期編成の作成は、以下の理由で効率的であると言われています。

- 分析モデルまたはユース・ケース・モデルの内容から設計モデルの内容を生成する変換機能を使用したい場合に、ソース・パッケージから目的パッケージへのマッピングが単純で分かりやすくなります。

- 機能の凝集とパッケージの疎結合に基づいて初期編成を作成する方法では、最終的なコンポーネント指向の編成にマッピングできる可能性が明らかに高くなります。つまり、設計プロセスで必要になるリファクタリングの量が減ります。

- パッケージを疎結合にすると、チームのワークフローが改善される可能性があり、その設計を複数のモデリング・ファイルに織り込む場合の再利用が容易になる可能性もあります。

もちろん他の方法も可能であり、「最終段階」の編成としてその方法が望ましい場合もあります。

- J2EE ベースの Web アプリケーション (EJB を含む) を設計している場合は、RSA および Rational Application Developer の J2EE プロジェクトに関する規則に従うことになります。[9]特に、アーキテクチャ

[7] そのようなパッケージは、変換機能では省略されます。

[8] 分析クラスのパッケージは、発見されるたびに大幅にリファクタリングされるのが普通です。再利用と予期されなかった機能要件のサポートを改善するためです。

[9] 大ざっぱに言うと、システム、アプリケーション、または大規模サブシステムごとに 1 つのエンタープライズ・プロジェクト、各エンタープライズ・プロジェクトに対してプレゼンテーション層用の 1 つの Web プロジェクト、コンポーネントや小さなサブシステムに全般的に対応する EJB プロジェクトと、コンポーネントまたはサブシステムごとのセッション層 (セッション EJB) とドメイン層 (エンティティ EJB) に使用される通常は別個の EJB プロジェクトという複数の EJB があります。詳細については、このホワイト・ペーパーのセクション 9 を参照してください。

一層 (プレゼンテーション層とビジネス層。ビジネス層にはセッションおよびドメインという副層がある) に対応する最上位レベルの設計パッケージを定義する必要があります。これは明らかにプラットフォームに中立な方法ではないので、設計しているソリューションを J2EE 以外のプラットフォームに実装しないことが分かっている場合にのみ採用してください。

- もっと一般的な例として、開発者の専門的な判断により n 層のアプリケーションを構築していて、作業の分担がプレゼンテーション層とビジネス層に対応しているというケースでは、それらのアーキテクチャー層に対応する最上位レベルのパッケージを使用することになります。しかし、特定の「アーキテクチャー」をサポートするために特定の「ビジネス機能」をサポートすることを意図したクラスは、注意深く編成してください。どちらも変更が困難だからです。

- コンポーネント指向、サービス指向、サブシステム指向のいずれでもない編成方法を使用する十分の理由がある場合でも、コード生成の変換機能を構成する際に余分の努力を払うことにより、設計の編成を対象のプロジェクトおよびフォルダーにマッピングできるはずです。特に複雑なマッピングを定義するためには、「マッピング・モデル」と呼ばれる特別なタイプのコンパニオン・モデルを使用できます。

設計モデルの内容

設計モデルに何を含めるべきかに関して厳格な規則はありませんが、次の提案が役立つでしょう。

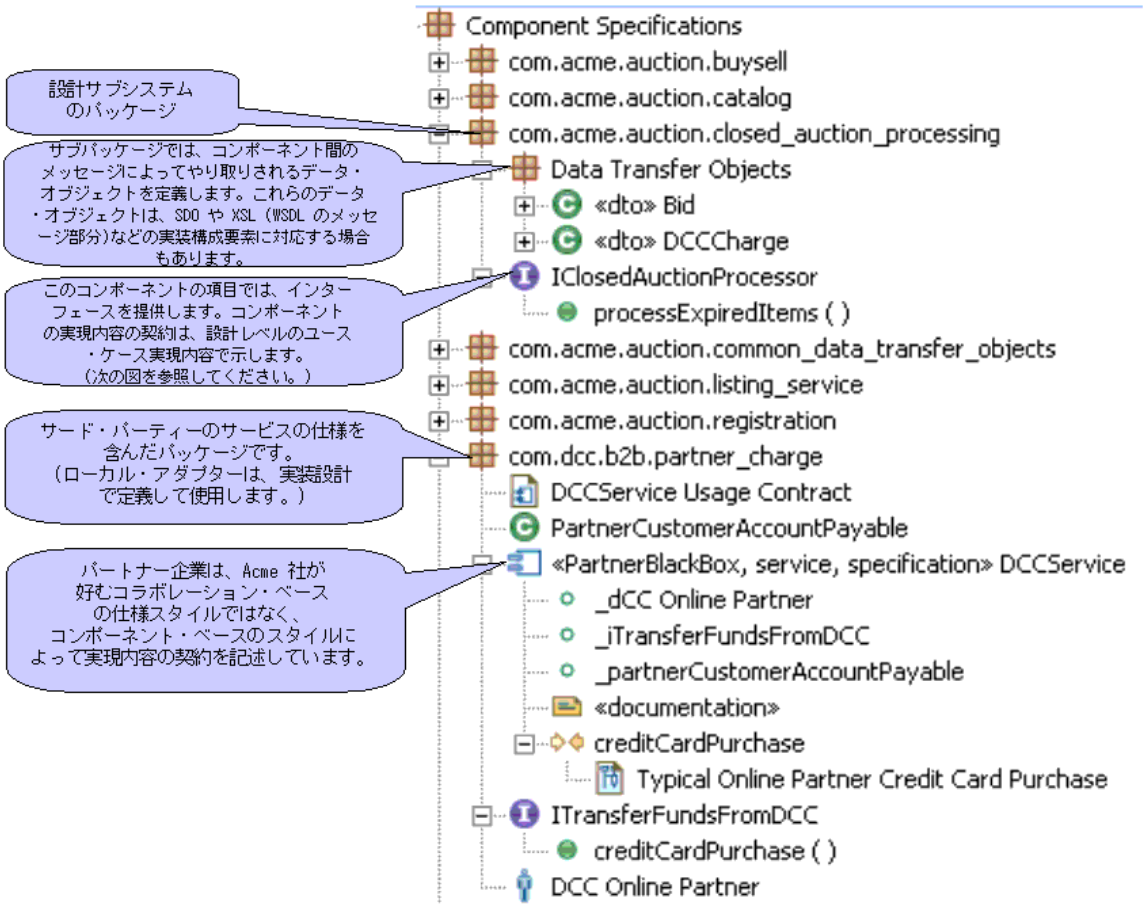


図 7-2

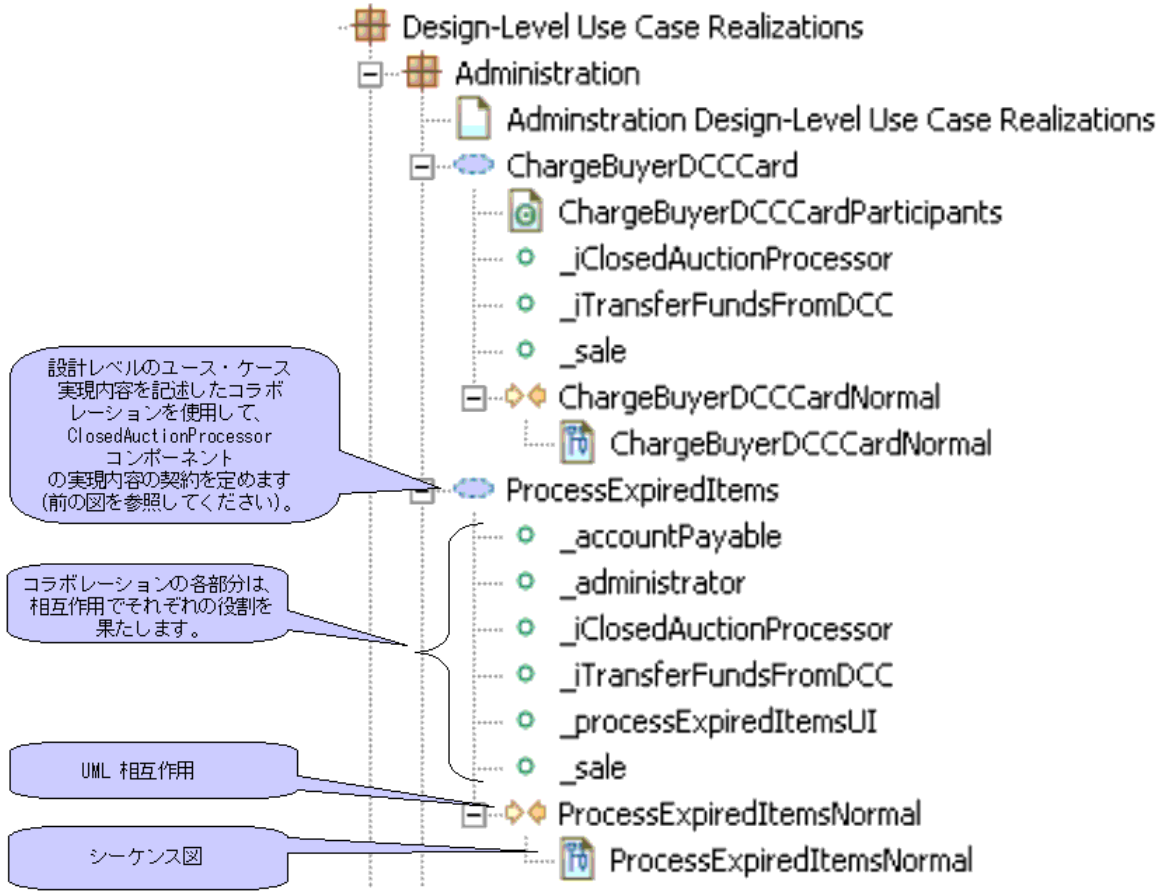


図 7-3

図 7-2 と **図 7-3** は、**図 7-1** に示されている編成の構造に従っており、設計の契約をどのように規定するかを示しています。

- 「ClosedAuctionProcessor」コンポーネントの使用法の契約は、1 つのインターフェースとして表現されています[10] (**図 7-2**)。それに対応する実現内容の契約は、コラボレーションとして表現された 1 つの設計レベルのユース・ケース実現内容により規定されています[11] (**図 7-3**)。分析レベルのユース・ケースの実現内容は分析クラス間のコラボレーションを示しているのに対して、設計レベルの実現内容は抽象化レベルの低い設計要素間のコラボレーションを示していることに注意してください[12]。設計モデルの仕様サブセットを実装設計サブセットとは別にパッケージ化したい場合は、設計レベルのユース・ケースの実現内容で役割として分析仕様要素または設計仕様要素だけを使用し、実装設計要素は使用しないようにすることが大切です。

[10] この例ではインターフェースが 1 つですが、もちろん、コンポーネントに複数のインターフェースを提供することもできます。

[11] 他のコンポーネントは複数のシステム・ユース・ケースに参加することがあるため、それらのコンポーネントの実現内容の契約が複数のユース・ケース実現内容の中に含まれることがあります。そのような場合には、コンポーネントのインターフェースと同じパッケージ内に「{コンポーネント名} 使用場所」という図を組み込んで、それらのユース・ケースの実現内容を構成するさまざまな図へのリンクをその図に配置することができます。

[12] それに似たもう 1 つの違いとして、設計レベルの実現内容に含まれる「参加者」図の一部は、分析レベルのユース・ケースの実現内容について提案されている参加者クラス図の代わりに (または、それに加えて)、コンポーネントの配線を示すコンポーネント図のことがあります。

- サード・パーティーの「DCCService」に対する使用法および実現内容の契約は、共に 1 つのパッケージに入っています[13]。このケースでも使用法の契約が 1 つのインターフェースで構成されていますが、実現内容の契約は «specification» 図を使用して表現されています (**図 7-2**)。なお、それ以外の点では実現内容の契約はほとんど同じように規定されており、動作 (この場合、「creditCardPurchase」という相互作用) を使用して表現されています。コラボレーションの代わりにコンポーネントを使用するもう 1 つの例を**図 7-4** に示します。

- 操作はインターフェース内で定義し、«specification» コンポーネント (使用されている場合) によって実現するか、そのインターフェースを実装している実装設計内の分類子によって実現することができます。

- データ転送オブジェクト (提供されている操作のパラメーター・タイプの役割を果たし、XML スキーマや SDO などの実装構成要素にマッピングできる) の仕様を使用法の契約に含めることもできます。分散可能として設計しないコンポーネントについては、操作パラメーターとして使用するタイプの仕様としてデータ転送オブジェクトを指定しても指定しなくてもかまいません。分散可能なサービス (たとえば、Web サービス) については、サービスの操作がローカル・アドレス空間のオブジェクトを参照してはならないため、DTO を使用する必要があります。

> ***XDE/Rose***
> 以前のバージョンの UML では、ユース・ケースの実現内容のガイドラインとして、ユース・ケースごとにコラボレーション・インスタンスを使用することや、実現内容の重要なフローごとに相互作用およびシーケンス図を使用することが定められていました。
>
> Rational Software Architect では、相互作用およびシーケンス図を 1 つしか使用できないことが多くなっています。UML2 シーケンス図で代替実行経路の表記がサポートされるようになったからです。
>
> また、UML2 では「コラボレーション・インスタンス」がなくなり、「コラボレーション・ユース」(そのタイプとしてコラボレーションが必要) が導入されました。したがって、Rational Software Architect では、ユース・ケースの実現内容を表現するためにコラボレーションを使用してください。

- [13] ここでは、DCC 社が Acme 社に UML 仕様を提供し、Acme 社はその仕様を設計モデルに組み込んだと仮定しています。これは、インターネット・ドメインの名前空間を反転させて使用するのが便利なタイプのシナリオです。

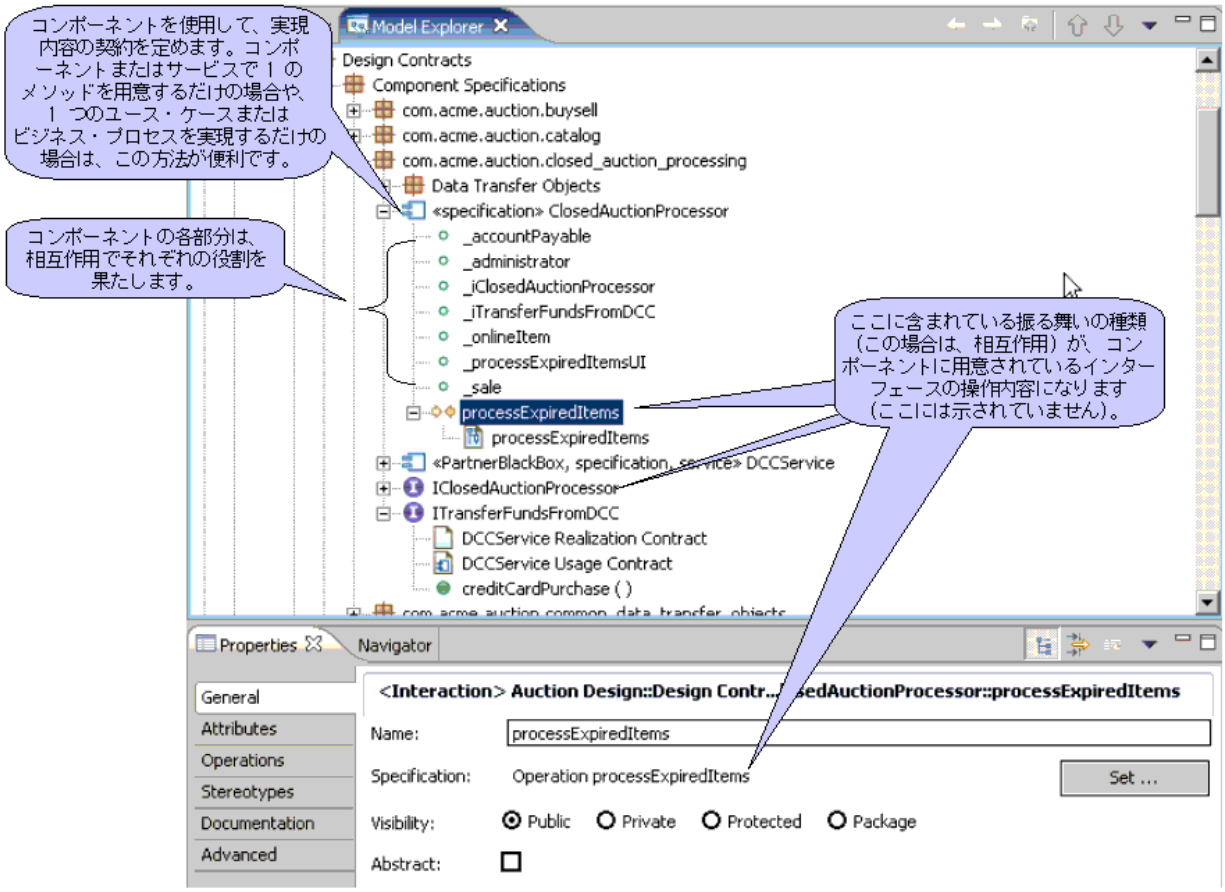


図 7-4

- 実装設計を指定するために利用できる方法の 1 つを**図 7-5** に示します。実装の構造は、操作を含む単純なクラスを使用して定義します。この方法は、UML 1.x を使用して作成する設計モデルとしてごく標準的なものです。利用できる 2 つめの方法を**図 7-6** に示します。これは、UML2 の目標に準拠しています。この方法では、クラスの代わりにコンポーネントが使用されており、コンポーネントは操作を所有するのではなく動作 (この例では相互作用) を所有しています。

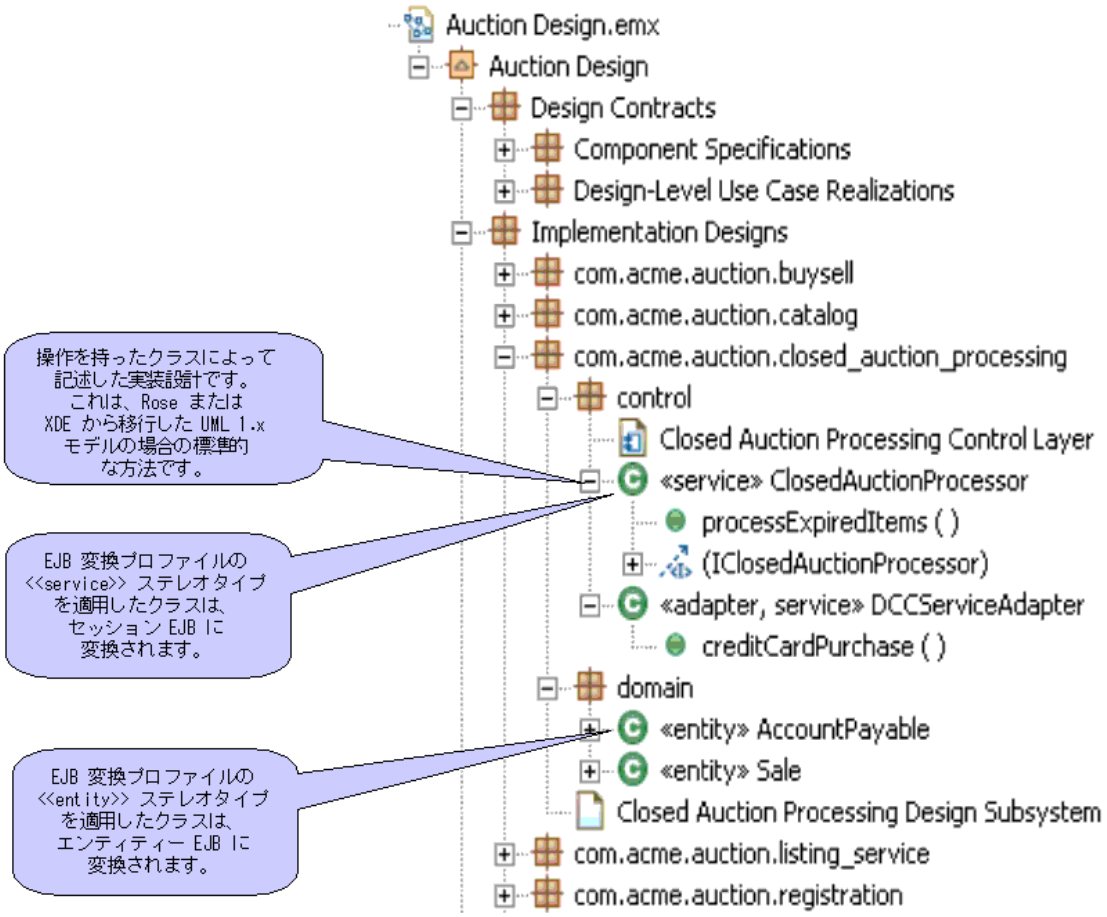

Auction Design.emx
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«service» ClosedAuctionProcessor
processExpiredItems ( )
(IClosedAuctionProcessor)
«adapter, service» DCCServiceAdapter
creditCardPurchase ( )
domain
«entity» AccountPayable
«entity» Sale
Closed Auction Processing Design Subsystem
com.acme.auction.listing_service
com.acme.auction.registration
操作を持ったクラスによって記述した実装設計です。これは、Rose または XDE から移行した UML 1.x モデルの場合の標準的な方法です。
EJB 変換プロファイルの <<service>> ステレオタイプを適用したクラスは、セッション EJB に変換されます。
EJB 変換プロファイルの <<entity>> ステレオタイプを適用したクラスは、エンティティー EJB に変換されます。

図 7-5

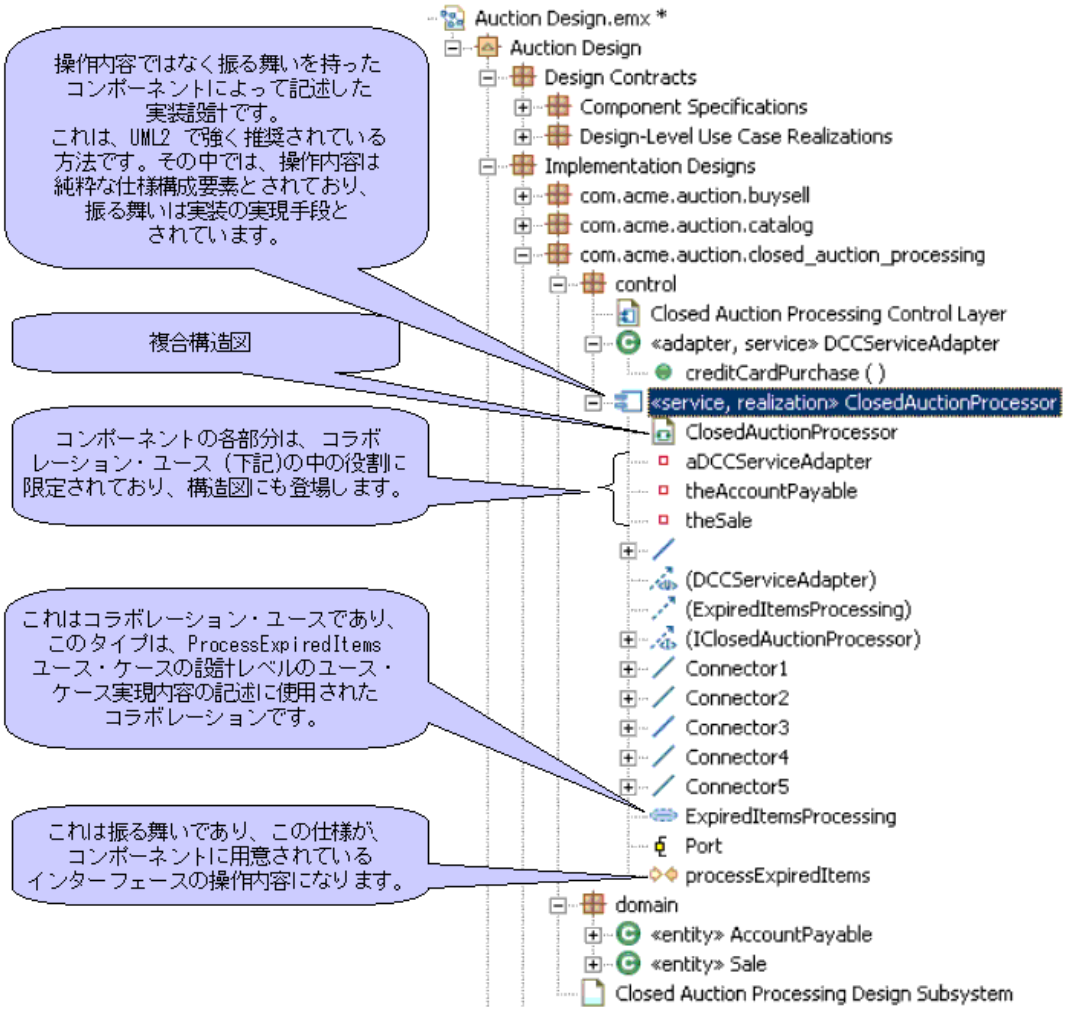


図 7-6

8. 実装概要モデルの内部編成のためのガイドライン

> *XDE/Rose*
> XDE のモデル構造ガイドラインでは、実装概要モデルは、実装のサブシステム・レベルの概要を提供する手段として推奨されていました。その上で、各サブシステムの詳細は、そのサブシステムを実装するプロジェクトのコード・モデルの中で指定していました。

厳密に言うと、RSA では実装概要モデルを使用する必要がないはずです。設計モデルの編成ガイドラインに従っていれば、設計モデルの (最終段階の) 編成は自然にコンポーネント (より大きい «subsystem» と、より分散可能な «service» のバリエーションを含む) の周りに形成されます。その後、変換機能により、設計のパッケージをプロジェクトにマッピングできます。たとえば、J2EE 実装の場合であれば、各種の Java、EJB、Web、J2EE アプリケーションや、実装が開発されている他のプロジェクトにマッピングされます。(そして、これらのプロジェクトは実際にはソリューションの実装モデルを表しています。このホワイト・ペーパーの『基本的な概念と用語』セクションで説明したとおりです。)

とはいえ、早い段階からプロジェクト構造のスケッチを作成するのを好む開発者や、視覚的に分かりやすいプロジェクト構造の表現を好む開発者もいることでしょう。たとえば、プロジェクトやフォルダーを表す成果物に «project» や «folder» といった明示的なキーワードを付けたり、さらには «EJB Project» や «Web Project» といった具体的な名前を付けたりする表現です。考慮に入れるべき別の点は、実装の成果物を (たとえば JAR のように) 具体的に表現するのは設計モデルとしては不適切です (Rational Software Architect の操作理論では設計モデルをプラットフォームに対して中立にすることになっている)。しかし、実装概要モデルであれば、そのような成果物を問題なく含めることができます。というわけで、実装概要モデルを使用したくなることもあります。**図 8-1** に、実装概要モデルの例を示します。

最後の点として、実装概要モデルは、ソリューションのさまざまな面を表す自由形式の図を入れるのに適しています。**図 8-2** は、このホワイト・ペーパーで取り上げている大部分の実例の基になっているオークション・システムの大まかな概念図です。

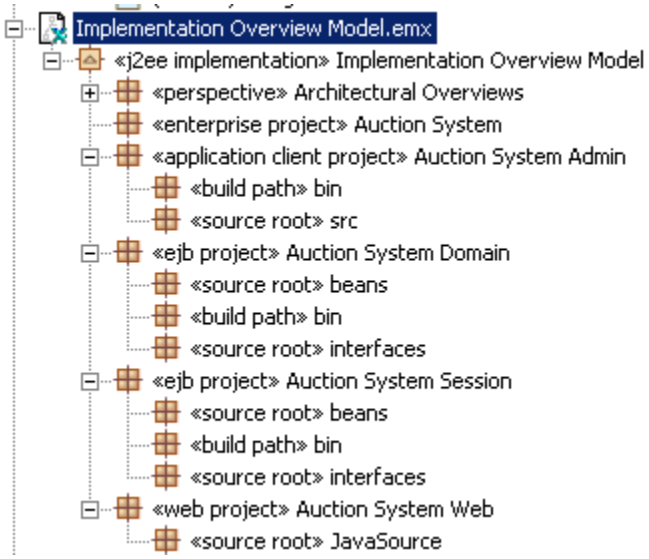


図 8-1

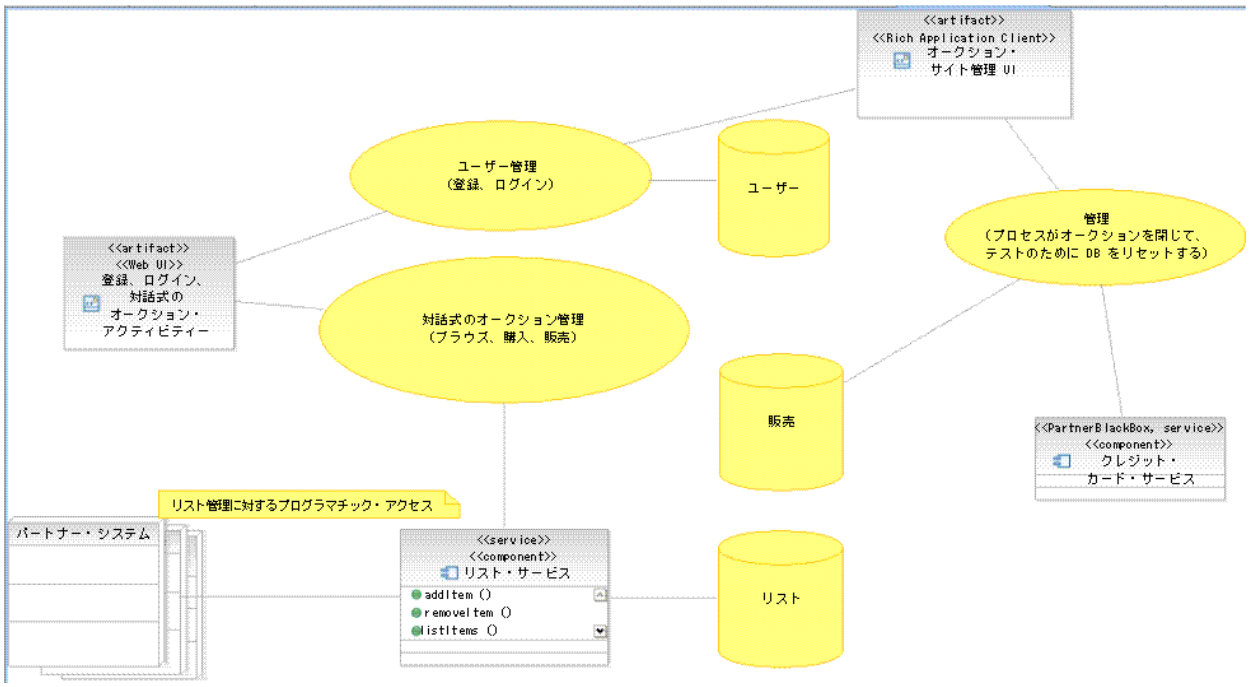

<<artifact>>
<<Rich Application Client>>
オークション・
サイト管理 UI
ユーザー管理
(登録、ログイン)
ユーザー
管理
(プロセスがオークションを閉じて、
テストのために DB をリセットする)
<<artifact>>
<<Web UI>>
登録、ログイン、
対話式の
オークション・
アクティビティー
対話式のオークション管理
(ブラウズ、購入、販売)
販売
<<PartnerBlackBox, service>>
<<component>>
クレジット・
カード・サービス
リスト管理に対するプログラマチック・アクセス
パートナー・システム
<<service>>
<<component>>
リスト・サービス
addItem ()
removeItem ()
listItems ()
リスト

図 8-2

9. 配置モデルの内部編成のためのガイドライン

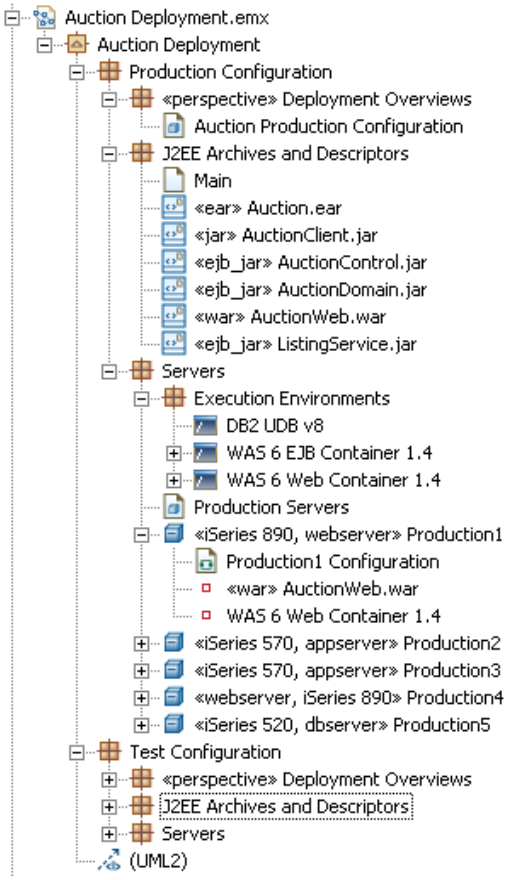


図 9-1

このホワイト・ペーパーでは、他のすべてのモデルに比べて配置モデルの必要性についてほとんど説明していません。配置モデルの編成やその内容の選択は下流に対してほとんど意味を持たないことが多いので、意味のあることだけを行えば十分です。それでも、読者に考えていただくため、利用可能な戦略と代表的な内容について上の**図 9-1** に示しておきます。この例では、以下の点に注目してください。

1. 実稼動構成の仕様をテスト構成の仕様から分離してあります。

2. 概要 (たとえば、クラスター、データ・センター、企業の概要など) は «perspective» パッケージに入れてあります。

3. ノードと成果物の特化および分類に関しては、パッケージの組み合わせとキーワードの使用という簡便な方法をとりました。より高度な方法を使えば、それぞれの環境で使用されているリソースのタイプを記述および文書化するのに適した特別なステレオタイプとプロパティーを定義する、特別な UML プロファイルを開発できます。

10. ソフトウェア・アーキテクチャー説明書を表すためのモデリング・ファイルの使用

モデルを編成するツール (図のリンクや、モデル間参照による複数のモデル・ファイルのサポートなど) が用意されていることから、RUP ソフトウェア・アーキテクチャー説明書および「アーキテクチャーの 4+1 ビュー」を実質的に表すモデルの作成はごく簡単な作業になりました。

最も単純な方法としては、**図 10-1** に倣って作業することになります。モデリング・ファイルを作成し、4+1 ビューに対応する単純なパッケージ・セットをそのファイルに追加します。(この例では、プロセス・ビューのパッケージを省略しています。この例のシステムは並行性の方法について多くを示してはいないからです。)

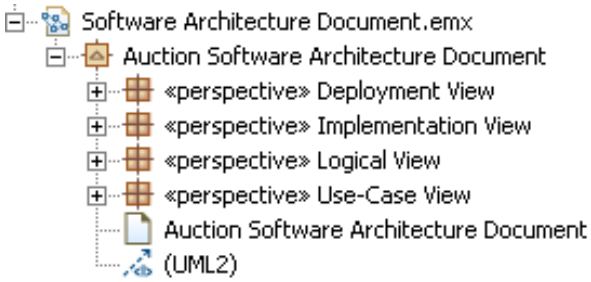


図 10-1

その後、**図 10-2** の例に倣ってデフォルトの図を作成できるでしょう。この図に注記やテキストを追加することもできます。

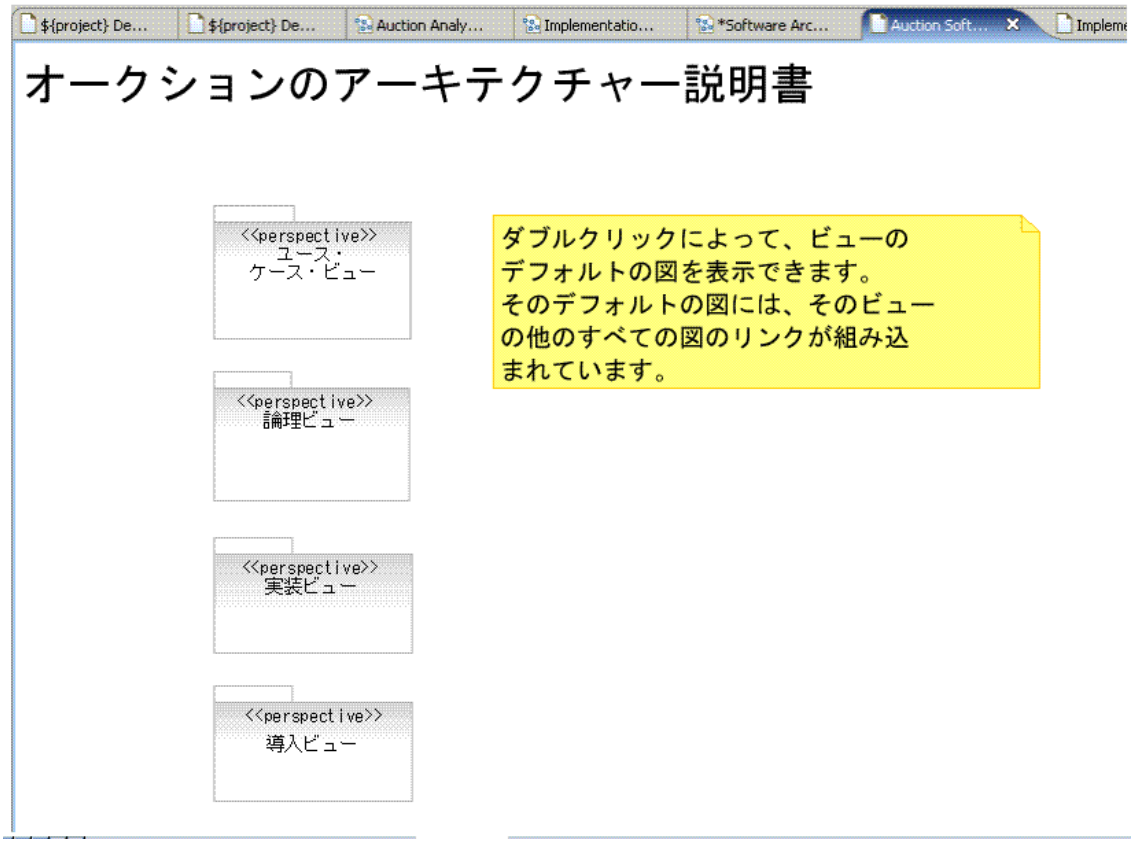


図 10-2

その後、以下の方法を用いて、ソフトウェア・アーキテクチャー説明書のモデリング・ファイル内に図をいくつか作成します。

- 他のモデリング・ファイルにある UML セマンティクス要素を使用して、それらのモデリング・ファイルには含まれていない、アーキテクチャー説明書の一部として必要な新しいビューとなる図を作成します。

- ソフトウェア・アーキテクチャー説明書のモデリング・ファイルに含まれる幾何学形状や「その場限り」の UML 要素で構成される図を作成します。(そのような UML 要素は、文書化や明確化だけを目的としているべきであり、記述しているソリューションの実際の実装に対してセマンティクス上の意味があってはなりません。)

- 他のモデリング・ファイルにある既存の図に対するリンクを含むだけの図を作成します。(アーキテクチャー説明書のモデリング・ファイルを他のモデリング・ファイルと一緒に読者に配布する場合には、この方法で問題ありません。アーキテクチャー説明書を Web で公開する場合には、上記のいずれかの方法を使用してください。)

11. チーム開発の考慮事項

ユース・ケース・モデル、分析モデル、設計モデルなど、RUP により認識される各種のモデルについては『基本的な概念と用語』セクションで説明しました。また、RSM または RSA の「実装前」モデルを 1 つ以上のモデリング・ファイルとして保持できることや、複数のユース・ケース・モデル、複数の分析モデル、その他を維持し、それらのモデルを 1 つのモデリング・ファイルまたは複数のモデリング・ファイルのいずれかとして保持できることを、例を挙げて説明しました。このセクションでは、モデルを複数のモデリング・ファイルとして保持するべきなのはどんな場合で、それはなぜかについて、いくつかの考慮事項を紹介します。これらの問題の総合的な説明については、RSA のオンライン・ヘルプを参照してください。

チームでのモデリング

複数の担当者が 1 つのモデルに対して並行して作業する必要がある場合には、そのモデルを複数のモデリング・ファイル (.emx ファイル) に分割すると効率が上がることがあります。チームで作業するには、多くの場合、独占的アクセスを取得するためにモデリング・ファイルをチェック・アウトしてから作業することになります。そうすれば、2 人以上の担当者が同じモデリング・ファイルを修正してしまい、それらの修正内容をマージする必要が生じるケースを減らせます。

そのような利点がある一方で、トレードオフもあります。複数のモデリング・ファイルを使用する場合、頻度が低いとはいえ、マージを実行する必要が時おり発生します。その上、マージでは、ファイル・グループではなく、各ファイルを個別に処理します。言い換えると、1 つのモデルを複数のモデリング・ファイルとして保管した場合、個々のファイルを論理的なモデル全体の「文脈外」で処理することになるわけです。マージ作業を効率的に実施するには、モデル全体の情報内容に多くアクセスできればできるほど有利になります。つまり、モデルを分割するモデリング・ファイルの数が少ないほど良いということです。

結局のところ、重要なのは、モデルを複数のファイルに区分することそのものではなく、区分したファイルに対して複数のチーム・メンバーが作業しても、マージの際に解決しなければならない矛盾した変更が発生しないような構造のモデルを作成することです。モデル (の論理的なインスタンス) を複数の物理ファイルに分割するのは、それによってマージを実行する必要性が顕著に低くなる場合に限ることをお勧めします。通常は、チーム・メンバーが個々のファイルを「独占的」に (つまり、1 つのファイルをチェックアウトしているチーム・メンバーはどの時点でも 1 人だけという状態で)、かつ「分離された状態」で (つまり、モデルの論理的なコンテキスト全体を構成する他のファイルにアクセスしないでも、チームの各メンバーが個々のファイルを変更できる状態で) 作業できるようであれば、

マージを実行する必要性を軽減できます。

モデルを分割する 2 つの方法

RSM や RSA では、モデルをリファクタリングする機能を使用することにより、論理的なモデル・インスタンスを複数の物理ファイルに分割するかどうかを臨機応変に決定できます。しかし、推奨されているのは「前もって計画する」ことです。このあと、2 つの方法について比較検討します。

計画的な方法: 最初からモデルを分割

開発チームにおけるファイル共有の必要性を可能な限り予測し、それに応じてモデルを論理的なサブセットに分割します。たとえば、次のようにします。

- 各ユース・ケース・モデルを複数のモデリング・ファイルに分割するには、チーム内の各分析担当者が機能面のどのような専門知識を持っているかを調べ、それに応じてモデルを分割します (要するに、モデル内で実際のユース・ケースを編成するためにパッケージを使用したであろうサブセットについて別個のファイルを作成する)。たとえば、医療サービス・アプリケーションであれば、患者登録ユース・ケースを入れるモデリング・ファイルと、注文処理ユース・ケースを入れるモデリング・ファイルを作ります。あるいは、注文処理ユース・ケースをさらに分割して、実験用の注文、放射線関連の注文、薬剤関連の注文などに分けることもできます。

- 各分析モデルを複数のファイルに分割するときに推奨される方法も、チームの各メンバーが持つ専門知識に従って分けるか、問題領域内の自然で論理的なグループに従って分けることです (この 2 つの考慮事項を併用することが多い)。

- 各設計モデルを複数のファイルに分割するときには、アーキテクチャー内の主要なサブシステム、サービス、またはコンポーネントに基づいて分けます (言い換えれば、チームや個人に実装作業を割り振る仕方に従って分ける)。

上記のそれぞれのケースで、各モデルについて、共有される要素および共通の要素や、区分 (サブシステム/パッケージ/サービス) をまたがる概要を提供する図を含む付加的なモデリング・ファイルを保持することもできます。

モデルの分割に関するその他のガイドラインについては、RSM/RSA のオンライン・ヘルプを参照してください。

臨機応変な方法： モデルのリファクタリング

RSM や RSA では、まずモデル全体を 1 つのファイルに入れて開始し、後で枝を「切り離す」ことによって追加のファイルを作成するという仕方で、1 つのモデルに対して複数のファイルを作成することができます。分割の方針を前もって計画していた場合でも、チームのワークフローを改善する新しい手段が分かったときなどに、この方法が役立ちます。

このセクションではモデルの論理的な内容に関心を向けているので、切り離した枝を「サブモデル」と呼ぶことにします。サブモデルを、それを含んでいたモデルから切り離すと、元のモデルはそのサブモデルを指す「ショートカット」を保持します。サブモデルの側では、元のモデルに対するバックポインターを保持しません。デフォルトでは、新しいサブモデル内の要素は、元の親モデルの名前空間に所属しなくなります。ただし、元のモデルの名前空間を新しいサブモデルに伝播させるオプションを、切り離し処理のときに選ぶこともできます。元のモデルから切り離すことによってサブモデルを作成する手順については、RSM/RSA のオンライン・ヘルプを参照してください。

ファイル間の参照

2 つの RSA モデリング・ファイル間で要素を相互に参照することができます。プロジェクトをまたがる参照も可能

です。このような参照は、ユース・ケース間の依存関係やクラス間の関連など、明示的に作成した関係を表すことがあります。また、RSA によって作成される参照や、非表示の参照もあります。たとえば、変換機能のアプリケーションにより追跡可能性のリンクが生成されることがあり、これらのリンクは通常のモデル要素として表示されます。別の例として、RSA の図には、その図に示されているセマンティクス要素 (どのセマンティクス要素も複数の図に含めることができる) を指す参照が含まれていることがあります。その表示要素参照は「非表示」の参照です (つまり、Model Explorer には表示されない)。

ファイル間の参照はそれぞれ、たとえば次のような場合に破損する可能性のある地点です。

- 相互参照先のファイルを 1 つかそれ以上、正しく保存できない場合 (たとえば、システムのクラッシュなど)
- ファイルをファイル・システム内で移動したとき、RSM/RSA モデル・サーバーがその移動を認識していない場合

したがって、モデルを複数ファイルに分割することを計画している場合は、ファイル間の参照が増えることを考慮に入れてください。この場合も、機能の凝集と組織単位 (パッケージやモデリング・ファイル) の疎結合を優先してモデルを編成すれば、チーム開発の効率改善に寄与します。