Rational XDE™ モデル構造ガイドライン
(Microsoft® .NET 用)

目次

1. 概要

このホワイト・ペーパーでは、Rational XDE™ Microsoft® .NET Edition で RUP モデルの成果物を表現して構成する方法についての推奨事項を説明します。これらの RUP 成果物を XDE 内でモデルとして採用するのが適当かどうかは、当然プロジェクトに応じて異なります。このホワイト・ペーパーでは、各モデルにおいて XDE の自動化サポート機能が利用できるのかをあらかじめ知っておくことにより、そのモデルを採用するかどうかを判断する材料を提供します。

XDE ではすべての XDE モデルがプロジェクトに所属するので、『XDE プロジェクトの構造』のセクションでは、どのような XDE プロジェクトを作成し、それらのプロジェクト内でどのような XDE モデル・ファイルを作成するのが良いかを説明します。

RUP と XDE は同じ「モデル」という用語を使用していますが、RUP モデルと XDE モデル間のマッピングは、一対一対応とは限りません。『RUP モデルから XDE モデルへのマッピング』のセクションでは、RUP モデルから XDE モデルへのマッピングについて説明します。

XDE モデル・ファイル内の各 RUP モデルの成果物の構造については、それぞれ別セクションで説明しています。

2. 開発範囲

このホワイト・ペーパーで取り扱う内容は XDE モデル・ファイルの推奨構造のみであり、これに対応する RUP 成果物の開発プロセスについては取り扱いません。このホワイト・ペーパー内に示された XDE モデルを含む XDE プロジェクトの定義方法の詳しい解説も、このホワイト・ペーパーの対象範囲外です。RUP 成果物の定義方法、開発方法、モデリングの方法については RUP のマニュアルを参照してください。プロジェクトの詳細については、IDE のマニュアルを参照してください。

このホワイト・ペーパーで示されている例はあくまで内容を強調するための部分的な例にすぎず、完全な例を示すものではありません。しかし、すべての例には相互に一貫性があり、実際の XDE モデルから抜粋されたものです。

ここに示されたプロジェクトとモデルの構造はあくまでも一例にすぎず、同じ機能を実現する構造はほかに何通りでもあります。

3. XDE プロジェクトの構造

このホワイト・ペーパーの主題は、XDE モデルの構造についてです。しかし、すべての XDE モデルは XDE プロジェクトに属しているため、ここで紹介する推奨モデル構造が属しているプロジェクトの構造についても簡単に説明します。

VS.NET では 1 つのソリューションが複数のプロジェクトの組み合わせによって実現されており、各プロジェクトには 1 つ以上の XDE モデル・ファイルを組み込むことができます。このためプロジェクト構造によって、作成されるモデル・ファイルの内容だけでなく、その個数が決定されます。

.NET Enterprise アプリケーションは、そのアーキテクチャーに応じて複数のプロジェクトから構成される場合があります。XML Web サービス、Windows、Web インターフェースから実装されるアプリケーションの例では、Web サービス、Windows アプリケーション、Web アプリケーションの 3 つのプロジェクトから構成されます。その他の VS.NET プロジェクト・テンプレートについては VS.NET のヘルプを参照してください。

複数人で開発される .Net エンタープライズ・アプリケーションの場合には、以下の XDE プロジェクトとモデルを作成することをお勧めします。

XDE プロジェクト	説明	XDE モデル “<recommended model name>” (<XDE file type: model template>]
アプリケーション・プロジェクト (XDE 基本モデリング・プロジェクト)	アプリケーション・プロジェクトはアプリケーション全体を表します。アプリケーションを全体的に説明した XDE モデル・ファイルを含みます。	- “ユースケース・モデル” (Rational XDE:Use-Case Model) - “分析モデル” (Rational XDE:Analysis Model) - “全体レベルの設計モデル” (Rational XDE:Design Model) - “全体レベルの実装モデル” (Rational XDE:Blank Model) - “Deployment Model” (.Net: Deployment Model)
データ・モデリング・プロジェクト (XDE データ・モデリング・プロジェクト)	データ・モデリング・プロジェクトは、アプリケーションのデータをモデリングする際に必要なリソース、およびデータ・モデルとデータベース間のラウンドトリップ・エンジニアリングに必要なリソースを含みます。	- “論理データ・モデル” (Data:Logical Data Model) - “物理データ・モデル[1]” (Data: *ベンダー固有の物理データ・モデル・ファイル*) [1] - “ドメイン・モデル” (Data: *vendor specific domain model file*)
.Net クラス・ライブラリー・プロジェクト (XDE クラス・ライブラリー・モデリング・プロジェクト)	クラス・ライブラリー・プロジェクトは、クラス・ライブラリーの実装に必要なクラスやインターフェースなどの C# 要素を含む「C#」VS.NET プロジェクトであり、この例では Auction Manager がこれに相当します。ここに含まれる要素は .NET アセンブリーとしてパッケージ化と配置が行われます。 1 つの .NET アプリケーションには複数のクラス・ライブラリー・プロジェクトを含めることができます。すでに述べたとおりプロジェクトの個数はアプリケーションによって異なり、同じアプリケーションでも選択したアーキテクチャーに応じて変わります。	- “.Net コード・モデル” (.Net:.Net Code Model) - “.Net 配置モデル” (.Net:.Net Deployment Model)
Web プロジェクト (XDE Web モデリング・プロジェクト)	Web プロジェクトは、アプリケーションの Web リソースを表します。ここに含まれる要素は .NET アセンブリーにパッケージされ、配置されます。	- “.Net コード・モデル” (.Net:.Net Code Model) - “.Net 配置モデル” (.Net:.Net Deployment Model)

[1] Rational XDE では、複数のデータベース・ベンダー用の物理データベース・サポートを提供しています。XDE がサポートするデータベース・ベンダーごとにベンダー固有のテンプレートがあります。

	プレゼンテーション・ロジックの特定のエリアごとに個別の Web プロジェクトを定義することができます。作成する必要があるアセンブリーごとに Web プロジェクトを作成することが推奨されます。別個のプロジェクトを定義する場合は、プロジェクト名にそれぞれのコンテンツを反映させる必要があります。	

Solutions Explorer と XDE Model Explorer の各ビューを使用したプロジェクトとモデルの構成例を次の図 1 に示します。図の左側が Solution Explorer、右側が Model Explorer のビューです。

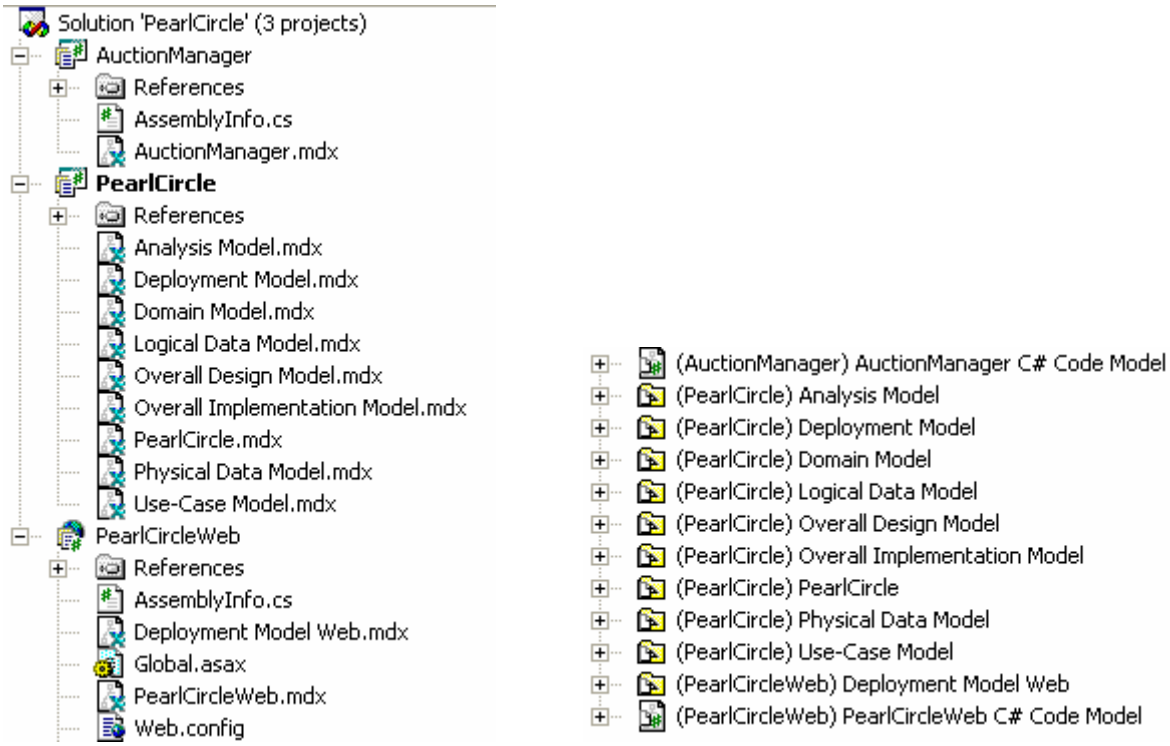

図 1:プロジェクトのモデルと構成

アプリケーションの規模が非常に小さく、1 人で開発を行う場合は、代替案として、上記のプロジェクトの構造は 2 つのプロジェクトにまとめられます。すなわち、アプリケーション規模の Web 以外の要素を含むプロジェクト、および Web 要素を含むプロジェクトの 2 つです。プロジェクトの数を減らすだけでなく、モデルの数を減らすこともできます。例えば、小規模の 1 人で開発を行うプロジェクトでは、以下のように単純化することが可能です。

- 独立した分析モデルは保守されません。分析と設計は共に XDE ラウンドトリップ・モデルで実行されます。
- 「全体レベルの設計モデル」と「全体レベルの実装モデル」は保守されません。プロジェクトの規模が小さいため、XDE ラウンドトリップ・モデルを直接見るだけで、全体像を掴むことができます。同様にユースケースの実現は .Net コード・モデルで保守されます。

- 独立した論理データ・モデルは保守されません。物理データ・スキーマは「物理データ・モデル」内で直接開発されます。

このような「小規模のプロジェクトの構造」について次の表にまとめます。

XDE プロジェクト	説明	XDE モデル “<recommended model name>” (<XDE file type: model template>]
アプリケーション・プロジェクト (XDE クラス・ライブラリー・モデリング・プロジェクト)	アプリケーション・プロジェクトは、アプリケーションの Web 以外の局面を表します。アプリケーションを全体として説明するモデル、データ・モデル、.Net 固有モデルを含みます。	- “ユースケース・モデル” (Rational XDE:Use-Case Model) - “物理データ・モデル” (Data:*ベンダー固有の物理データ・モデル・ファイル*) - “.Net Code Model” (.Net: .Net Code Model) - “.Net Deployment Model” (.Net: .Net Deployment Model)
Web プロジェクト (XDE Web モデリング・プロジェクト)	Web プロジェクトは、アプリケーションの Web リソースを表します。ここに含まれる要素は .NET アセンブリーにパッケージされ、配置されます。	- “.Net Code Model” (.Net: .Net Code Model) - “.Net Deployment Model” (.Net: .Net Deployment Model)

小規模のプロジェクトとモデルの構造の例を図 2 に示します。

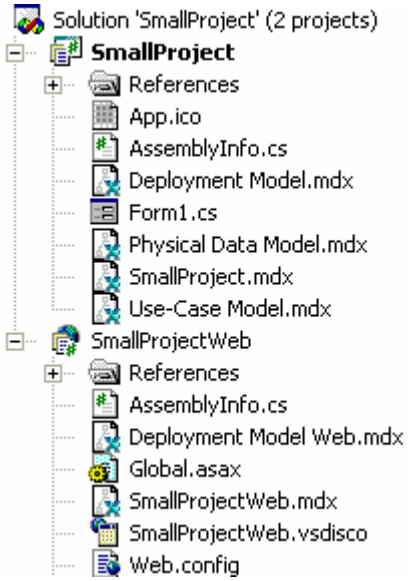

図 2:小規模の XDE プロジェクトとモデルの構造の例

実際はアーキテクチャーに基づきプロジェクトと個々のモデル・ファイルの数を選択していくことになり、プロジェクトによって選択される要素は異なる可能性があります。しかし、定義されているプロジェクトの数に関係なく、1 つのプロジェクト当たり 1 つしか XDE .Net コード・モデル・ファイルは存在しません。プロジェクトとそれに含まれる XDE モデル・ファイルの詳細については、XDE のマニュアルを参照してください。

また、*XDE モデル名が XDE プロジェクト全体を通じて重複しないようにする* ことを強く推奨します。この点は XDE モデル間の参照を解決するときに非常に重要になります。モデル間の参照とその解決については XDE のマニュアルを参照してください。

このホワイト・ペーパーでは図 1 に示した XDE モデルの構造に従って順に解説します。

4. RUP モデルから XDE モデルへのマッピング

XDE 内での RUP モデル成果物の表し方の説明に入る前に、「RUP モデル」と「XDE モデル」の違いを理解しておくことが重要となります。この 2 つのモデルは別の物であり、RUP モデルから関連する XDE モデルへのマッピングは 1 対 1 対応とは限りません (1 対 1 に近いですが、1 対 1 とは限りません)。RUP と XDE の両方で「モデル」という用語が使用されているため、最初は 2 つのモデルが同じであると仮定してしまいます。しかし、RUP モデルがプロセスにおける問題 (分析、設計、実装等) を独立させるのに対し、XDE モデルは開発における問題を独立させます (プログラミング言語のパッケージング構造と仮想ディレクトリー構造を記述するコード・モデルを独立させる、異なるプログラミング言語や開発環境に対してコード・モデルを独立させる等)。このホワイト・ペーパーでは、RUP モデルと XDE モデルの違いを明確にするために、「モデル」という用語は「RUP」または「XDE」で修飾しています。

次の表に、RUP モデルから XDE モデルへのマッピングについてまとめます。XDE モデルは、『XDE プロジェクトの構造』のセクションで紹介しています。各 XDE モデルの構造については、このホワイト・ペーパーの後のセクションで説明します。

RUP モデル	<XDE プロジェクト>:< XDE モデル名>
ユースケース・モデル	アプリケーション・プロジェクト:ユースケース・モデル
分析モデル	アプリケーション・プロジェクト:分析モデル
設計モデル	アプリケーション・プロジェクト:設計モデル
データ・モデル	XDE データ・モデル データ・モデリング・プロジェクト:論理データ・モデル データ・モデリング・プロジェクト:*ベンダー固有の* 物理データ・モデル データ・モデリング・プロジェクト:*ベンダー固有の* ドメイン・モデル
実装モデル	アプリケーション・プロジェクト:実装モデル
配置モデル	アプリケーション・プロジェクト:配置モデル

5. ユースケース・モデル

「Use-Case Model」の推奨構造を図 3 に示します。

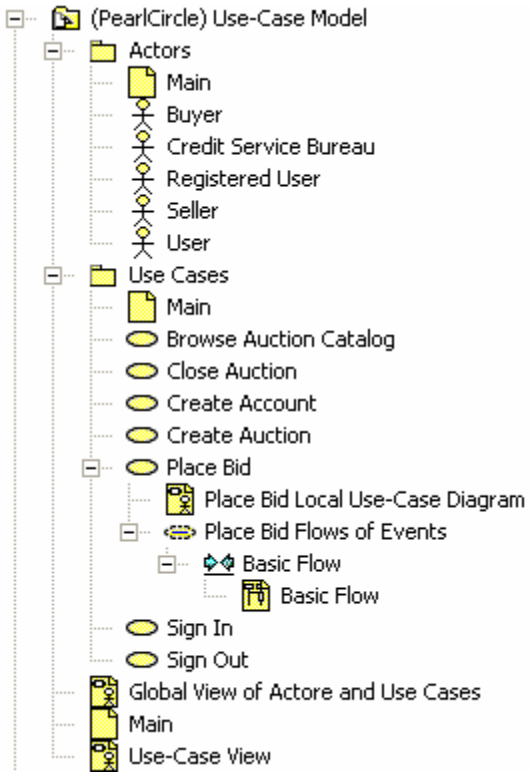

図 3:「Use Case Model」の構造

「Use-Case Model」は「Actors」と「Use Cases」という 2 つのパッケージから構成されます。

アクターと**ユースケース**を含む**ユースケース・モデル**図に加えて、**ユースケース**の別の側面を明確にするために他のダイアグラムを使用することもできます。図 3 で示したように、**ユースケース・モデル**の**ユースケース**・モデル要素の「下」に以下の補足的なモデル要素を含むこともできます。

- 「Place Bid Local Use-Case Diagram」図には、「Place Bid」**ユースケース**およびこの**ユースケース**に参加する**アクター**が含まれています。
- 「Place Bid Flows of Events」コラボレーション・インスタンスには、ユースケース定義書に記述されているイベント・フローをグラフィカルに説明した相互作用図 (**アクター**と**ユースケース**の相互作用) が含まれています。このコラボレーション・インスタンスを『分析モデル』および『分析モデル』のセクションで説明している**ユースケースの実現**と混同しないように注意してください。「Use-Case Model」内のコラボレーション・インスタンスは厳密に「ブラック・ボックス」であり、アプリケーション内の要素の相互作用は表しません。
- 「Place Bid Flow of Events」アクティビティー・グラフは、ユースケース定義書に記述されているイベント・フローをグラフィカルに説明したアクティビティー図を含みます。

図 3 で示されている例において、図 3 の「Global View of Actors and Use Cases」図には、すべての**ユースケース**と**アクター**およびその関係が示されています。「Main」図のように「Main」図が置かれているパッケージの要素しか含まれていないものとは異なります。**アクター**と**ユース・ケース**が多いときは、「Global View of Actors and Use Cases」図の情報を複数の図で表現することができます。

「Use-Case View」図はソフトウェア・アーキテクチャーのユースケース・ビューを表します。アーキテクチャー・ビューについては RUP のマニュアルを参照してください。

次の図 4 に示すように、*Actors* と *Use Cases* のパッケージの下にさらにパッケージを階層的に作成してモデル要素を整理することもできます。

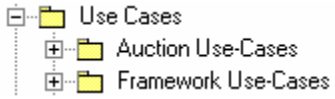

図 4:Use Cases パッケージの階層化

6. 分析モデル

分析モデルには、**分析クラス**と分析**ユースケースの実現**が置かれます。

メモ :独立した**分析モデル**と**設計モデル**を保守するかどうかはプロジェクトによって異なります。独立した**分析モデル**を保守しない場合には、**分析クラス**を適当な**設計モデル**のパーティション[2] へと移動し、洗練します。. 別の方法を取る場合には、**設計モデル**内に**分析クラス**と分析**ユースケースの実現**を作成し、そこから適切な設計の形へと展開していくこともできます。XDE において**設計モデル**がどのように表されているかの詳細については、『設計モデル』のセクションを参照してください。

分析モデルの推奨構造を**図 5** に示します。

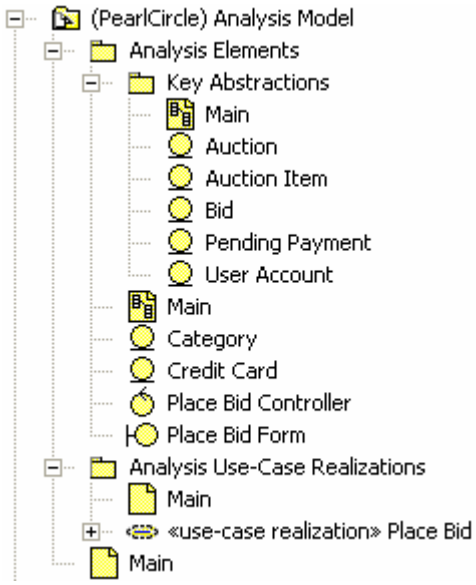

図 5:分析モデルの構造

「Analysis Elements」パッケージには**分析クラス**が含まれています。**分析クラス**のインスタンスは、「Analysis Use-Case Realizations」パッケージ内の図にあります。

分析クラス (図 5 の「Key Abstractions」パッケージを参照) をさらに分割していく場合には、「Analysis Elements」パッケージ内で、**分析クラス**に加えてパッケージを定義することもできます。こういった付加的な分割はオプションで、特に独立した**分析モデル**を保守しない場合はその必要性が低くなります。このような場合、**分析クラス**は「一時的なもの」(つまり、設計要素に展開されるまでの期間のみ存在する) と考えることができるため、その編成は重要とは見なされません。中心となる抽象概念の**分析クラス**が唯一の例外となる可能性があります。

図 5 で示したように、「Key Abstractions」パッケージには、システムの中心となる抽象概念を表すと考えられる**分析クラス**が含まれています。前述したとおり、このパッケージはオプションです。代替案として、「Analysis Elements」パッケージ内のクラス図上で中心となる抽象概念を表す方法があります。ただし、独立したパッケージを作成することにより、中心となる抽象概念としての**分析クラス**のより厳密な分類が可能になります。実際、プロジェクトによっては、独立した**分析モデル**を完全に保守しなくても、中心となる抽象概念の**分析クラス**を保守する場合もあります。このような場合、保守される**分析クラス**を含む独立したパッケージを定義することが有用です。

[2] 後の項で明らかになりますが、技術固有の要素の設計はラウンドトリップ・モデルで行われるため、正しい「全体レベルの設計モデルの分割」は、XDE ラウンドトリップ・モデルの 1 つにパッケージを 1 つ置くことかもしれません。

メモ :抽象概念はまた「全体レベルの設計モデルの構造」の「Logical View: Key Abstractions」図にも表示されます。詳しくは、『設計モデル』のセクションを参照してください。

「Analysis Use-Case Realizations」パッケージには、分析レベルの**ユースケースの実現**が含まれます。これには、「Analysis Elements」パッケージ内の**分析クラス**からどのように**ユースケース**を実行するかの説明があります。それぞれの分析**ユースケースの実現**は、**ユースケース・モデル**内の**ユースケース**を実現し、実現した**ユースケース**と同一名を持ち、図 6 で示した構造になるはずです。

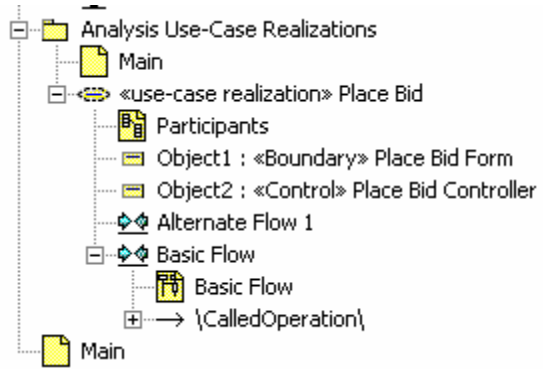

図 6:「分析ユースケースの実現」のパッケージ構造

「Participants」図には、**ユースケースの実現** (つまり、相互作用図にそのインスタンスが登場する**分析クラス**) に参加する**分析クラス** (「Analysis Elements」パッケージからの) と、相互作用図に記述されたコラボレーションをサポートする関係が示されます。

「flow」相互作用インスタンス (「Basic Flow」および「Alternate Flow 1」) は、イベントの**ユースケース・フロー**を表したシーケンス図を含みます。重要なイベントのユースケース・フロー 1 つに対して相互作用インスタンスが 1 つあるはずです。相互作用インスタンスのシーケンス図は、関連する**ユースケース**の実行に参加する**分析クラス**間のフローを記述しています。

7. 設計モデル

RUP **設計モデル**は複数の XDE モデルで表されます (「全体レベルの設計モデル」および独立した XDE ラウンドトリップ ・モデル内に常駐しているラウンドトリップ設計要素 (ラウンドトリップ設計要素は、ラウンドトリップ・エンジニアリングに関与する詳細な設計要素を指します))。こうすることで、個々のラウンドトリップ・モデルに用意された自動化機能を活用できます。

「全体レベルの設計モデル」は、アプリケーションの設計を全体として表し、複数のラウンドトリップ・モデルにわたり使用される要素を含みます。ここには、個々のラウンドトリップ・モデルの編成を導き出す論理的なパーティションと、あらゆる要素をまとめ上げる**ユースケースの実現**が含まれています (**ユースケースの実現**は異なるラウンドトリップ・モデルからの設計要素間のコラボレーションを表しています)。「全体レベルの設計モデル」には、ラウンドトリップ設計要素を参照する図が含まれています。個々の XDE ラウンドトリップ・モデルについては、『実装モデル』のセクションを参照してください。

別の可能性としては、**設計モデル**と**実装モデル**を同じ XDE コード・モデル内で表すことができます。これは、対象の実装言語が 1 つで、チームの規模が小さい場合にのみ可能です。

「全体レベルの設計モデル」の保守はオプションですが、図の編成、抽象レベルの引き上げ等は有効であり、また、これにより、どのような実装メカニズムを採用するか検討している段階時に設計要素を置く場所を確保できます。

「全体レベルの設計モデル」の推奨構造を図 7 に示します。

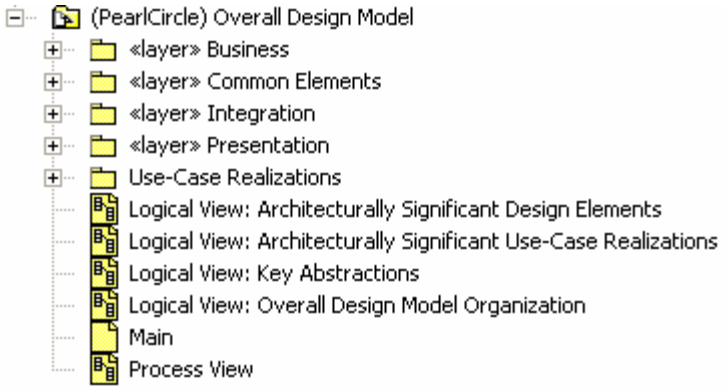

図 7:全体レベルの設計モデルの構造

この全体レベルの設計モデルには以下のパッケージが含まれます。

- «layer» パッケージには、システムの設計要素 (**設計クラス**、**インターフェース**、**設計サブシステム**) が含まれます (もしくは、これらの設計要素を参照する図が含まれます)。この構造は、『**設計レイヤー**』のセクションで説明する分割ストラテジーを反映しています。
- 「Use-Case Realizations」パッケージには設計レベルのユース ケースの実現が含まれます。『設計ユースケース実現』のセクションでは、ユースケースの実現の内部構造をさらに詳細に取り上げています。

アーキテクチャー・ビューを表す図は、その名前に「View」を含んでいます。アーキテクチャー・ビューについては RUP のマニュアルを参照してください。

「Logical View: Key Abstractions」図には、システムの中心となる抽象概念が含まれます。これらの中心となる抽象概念を保守するために、次のようなオプションが用意されています。

- 完全な**分析モデル**が保守されます。この場合、「Logical View: Key Abstractions」図には、システムの中心となる抽象概念を表す**分析モデル**からの**分析クラス**が含まれます。
- 部分的な**分析モデル**、具体的には中心となる抽象概念のみが保守されます。この場合、「Logical View: Key Abstractions」図には、システムの中心となる抽象概念を表す**分析モデル**からの**分析クラス**が含まれます。
- **分析モデル**をまったく保守しません。この場合、中心となる抽象概念を表す**分析クラス**を「Key Abstractions」と呼ばれる**設計モデル**内のパッケージで保守することができます。

分析モデルの詳細については、『分析モデル』のセクションを参照してください。

7.1 設計レイヤー

«layer» パッケージには、**分析クラス**から展開したシステムの設計要素 (例えば、**設計クラス**、**インターフェース**、**設計サブシステム**) が含まれます。«layer» パッケージの設計要素をさらに内容に応じてサブパッケージに分類することも可能で、定義できるサブパッケージの個数に制限はありません。設計**ユースケースの実現**はこれらのパッケージに含まれる設計要素に準じて記述されます。設計**ユースケースの実現**は、『設計ユースケース実現』のセクションで説明する「Overall Design Model」の「Use-Case Realizations」パッケージに含まれます。

設計モデルの内容をさらに区分する方法は無数にあります。このセクションで説明する分類例を**図 8** に示します。

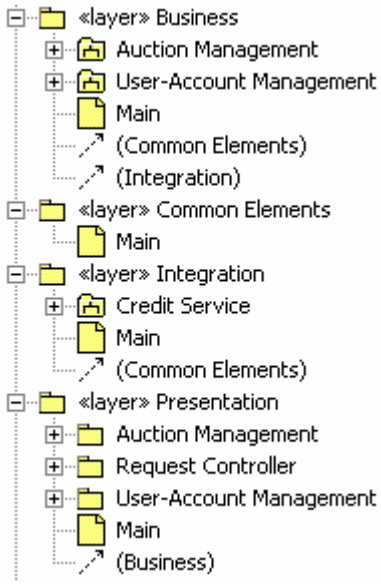

図 8:設計パッケージ内の分類例

この例では設計モデルの構成要素をまず機能別レイヤーに応じて分類し、さらにビジネス機能に応じて細かく分類しています。第 2 レベルのレイヤー・パッケージは、エンド・ユーザーとのインターフェースをつかさどる機能を担っています。

「Presentation」レイヤー・パッケージは、エンド・ユーザーとの対話を処理する機能を担っています。.Net アプリケーションにおいて、「Presentation」レイヤー・パッケージ内に常駐する設計要素には、Active Server Page (ASP.NET) が含まれます。関連性の高い**ユースケース**同士をグループ分けするために、「Presentation」レイヤー・パッケージをさらにサブパッケージに分類することもできます。**図 8** の例では、「Auction Management」パッケージがこれに該当します。

「Business」レイヤー・パッケージにはビジネス・プロセスに関わるすべての機能がまとめてあります。このホワイト・ペーパーで紹介している「全体レベルの設計モデル」の構造において、「Business」レイヤー・パッケージは、複数の設計サブシステム・パッケージで構成され、主要なビジネス機能ごとに 1 つのパッケージを持ちます (**図 8** の「Auction Management」および「User Account Management」サブシステム・パッケージがこの例に当たります)。**設計サブシステム**の各パッケージについて詳しくは、『**設計サブシステム**』のセクションで説明しています。

「Integration」レイヤー・パッケージは、データベースや外部システムを含むバックエンド・リソースへのアクセス機能を提供します。このホワイト・ペーパーで紹介している**設計モデル**の構造において、「Integration」レイヤー・パッケージも複数の設計サブシステム・パッケージで構成され、外部システム単位で 1 つのパッケージを持ちます (**図 8** の「Credit Service」サブシステム・パッケージがこの例に当たります)。**設計サブシステム**の各パッケージについて詳しくは、『**設計サブシステム**』のセクションで説明しています。

「Common Elements」レイヤー・パッケージには、複数のレイヤーで共通して使われる要素を集めます。

なお、以上で説明した構成も、違う分類方針に従う場合は全く違う構成になり得るということを、最後に改めて強調しておきます。

7.2 設計サブシステム

設計サブシステムは「全体レベルの設計モデル」のサブシステム・パッケージによって表されます。それぞれの設計サブシステム・パッケージは同じ構造でなければなりません。**設計サブシステム**をどの程度詳細にするかによって、構造の仕様は異なってきます。

より正式で厳密な**設計サブシステム**の構造例を**図 9** に示します。

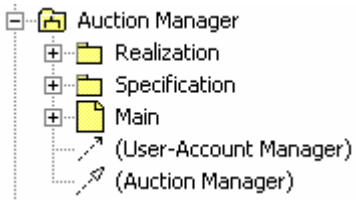

図 9:設計サブシステムの構造

この設計サブシステム・パッケージの構造は、設計サブシステム・パッケージ内に「Specification」と「Realization」を別パッケージとして定義することをサポートしています。この構造は、「*UML Components: A Simple Process for Specifying Component-Based Software* (J. Cheesman、J. Daniels 著)」に影響されています。これらのパーティションを含まない単純化された設計サブシステム・パッケージの構造は、このホワイト・ペーパーで定義している他のモデル・ファイルの構造に影響を与えることなく使用できます。「Specification」と「Realization」の各パッケージについて以下に解説します。

7.2.1 サブシステムの仕様

「Specification」パッケージには、**設計サブシステム**のインターフェースの記述が含まれます。[3] 図 10 にサブシステム仕様の例を示します。

図 10:設計サブシステム仕様の例

7.2.2 サブシステムの実現

「Realization」パッケージには、**設計サブシステム**仕様の実現方法を記述した要素が含まれます。設計サブシステム・パッケージの「Realization」パッケージの構成例を**図 11** に示します。

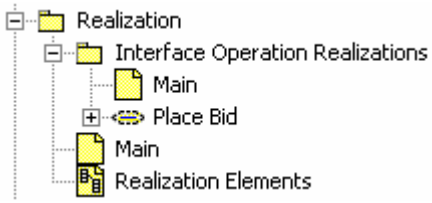

図 11:設計サブシステム実現の例

「Realization Elements」図には、サブシステムを実現する設計要素への参照が含まれます。設計要素自体は、ラウンドトリップ・エンジニアリングに参加する .Net コード・モデル内に常駐します。詳しくは、『実装モデル』のセクションを参照してください。

[3] この簡単な例では、インターフェース専用の独立したパッケージの必要性について疑問を感じるかもしれません。しかし、実際のプロジェクトでは、このパッケージにサブシステムを説明した文書への参照を含むことがあるため保守する価値があります。特に操作の事前条件や事後条件などのインターフェースの制約となるものは重要となります。

「Interface Operation Realizations」パッケージには、サブシステム要素が (「Specification」パッケージ内の) **設計サブシステム**・インターフェースの重要な操作を実現する方法を示すコラボレーション・インスタンスが含まれます。主要なサブシステム・インターフェース操作にはそれぞれ 1 個ずつのコラボレーション・インスタンスがあります[4]。「Interface Operation Realizations」パッケージの例を図 12 に示します。

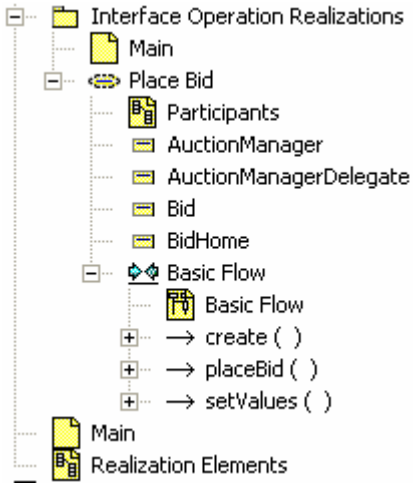

図 12:Interface Operation Realizations パッケージの例

分析レベルの**ユースケースの実現** (『分析モデル』のセクションで説明) と設計レベルの**ユースケースの実現** (『分析モデル』のセクションで説明) と同様に、各インターフェース操作の実現は、実現に参加するサブシステム要素 (分析モデル の「Participants」図) を含むクラス図と、これらの要素がサブシステム・インターフェース操作 (分析モデル の「Basic Flow」図) を実行する際の相互作用の方法を示した相互作用図を含みます。

7.3 設計ユースケース実現

「Use-Case Realizations」パッケージには設計レベルの**ユースケースの実現**が含まれます。各**ユースケースの実現**は、それぞれ**ユースケース・モデル**内の**ユースケース**に関連付けられ、その**ユースケース**と同一名を持ち、図 16 で示した構造を持つはずです。

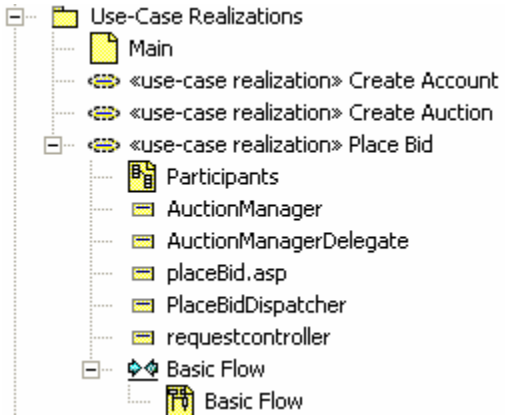

図 13:設計ユースケース実現の構造

ユースケースの実現の「Participants」図は、**ユースケースの実現**に参加する設計要素 (**ユースケースの実現**の相互作用図にインスタンスが表示される設計要素) と相互作用図に記述されたコラボレーションをサポートする関係を示しています。

[4] このレベルでは、すべての操作を定義する必要はありません。単純な操作については、独立したコラボレーションを必要としない場合があります。

「Basic Flow」図は、対応する**ユースケース**の実行中にそこに参加している設計要素の間に生じたフローを記述する相互作用図です。**ユースケース**に含まれるイベントの各フローに対して相互作用インスタンスが 1 つずつあるはずです。

重要な点は、**ユースケースの実現**図に、独立した XDE ラウンドトリップ・モデルに物理的に常駐する設計要素への参照が含まれることがあることです (通常は含まれます)。**ユースケースの実現**では、異なるラウンドトリップ・モデルの要素によるコラボレーションが行われます。

8. データ・モデル

RUP **データ・モデル**は、以下の複数の XDE モデル・ファイルで表されます。

- **論理データ・モデル** (オプション)。データベースの論理設計のアプリケーション非依存のビューである論理データ・モデルを表します。
- **物理データ・モデル**。データベースのベンダー固有の物理データ・モデルを表します。データベースのテーブル固有の特性を定義する詳細なモデル要素を含みます。「Physical Data Model」XDE モデル・ファイルには、ベンダー固有のデータベースにテーブルを実装するためのデータベース固有の実装の成果物も含まれます。
- **ドメイン・モデル** (オプション)。「物理データ・モデル」を通して一貫しているデータ・タイプを定義する際に使用できるデータベースのベンダー固有のデータ・タイプを表します。

XDE モデル・ファイルを分割することで、「全体レベルの設計モデル」、**データ・モデル**、物理データベースの間でサポートされる自動化の柔軟性が最適化されます。

これらの XDE モデル・ファイルについては、次項以降で詳しく説明しています。

8.1 論理データ・モデル (オプション)

論理データ・モデルは、データベースの設計に対して重要となる主要なエンティティーと関係を表す、独立した論理データ表現を作成する必要があるプロジェクトにおいて使用されることがあります。データベースの設計チームによっては、この代わりに初期の物理データベースの設計構造を XDE 物理データ・モデル内に直接作成するために、**設計モデル**内の永続的な**設計クラス**を**データ・モデル**内のテーブルに変換することもあるため、XDE 論理データ・モデルの作成はオプションとなります (下記の『物理データ・モデル』のセクションを参照してください)。

XDE 論理データ・モデルは、必要に応じて問題領域パッケージに分割することができます。問題領域パッケージは、エンティティー・クラスの論理的なグループ分けを定義します。XDE 論理データ・モデルは、複数の問題領域に渡るモデル要素を含む「Common Elements」パッケージも含む場合があります。

名前に「View」が含まれる図は、アーキテクチャーのデータ・ビューを文書化するために使用されます。「Data View: Overall Logical Data Model Organization」図は、XDE 論理データ・モデルの主要なパーティション (パッケージ) として表現されているとおり、論理データ・モデルの高レベルのデータ編成を文書化するために使用されています。「Data View: Key Logical Data Elements」は、**データ・モデル**の重要な論理要素として文書化に使用されます。論理データ・モデルが保守されていれば (すなわち、別の「論理データ・モデル」がある場合)、この図には XDE 論理データ・モデルからの要素が含まれます。アーキテクチャー・ビューについては RUP のマニュアルを参照してください。

図 14 に XDE 論理データ・モデルの推奨構造の例を示します。

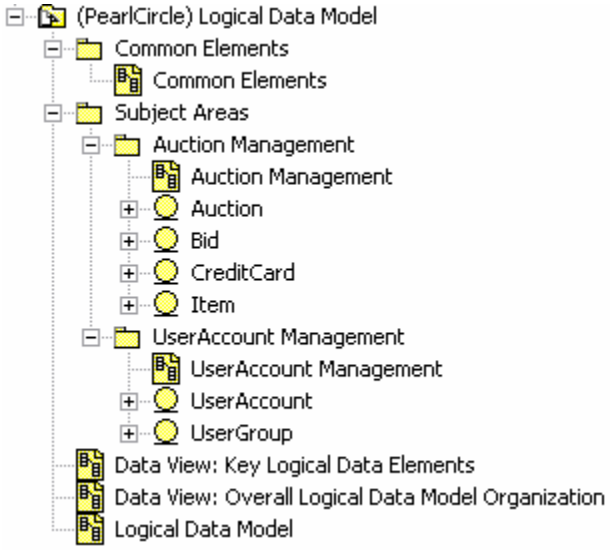

図 14: XDE 論理データ・モデルの構造

この例には、「Auction Management」と「User Account Management」の 2 つの問題領域パッケージがあります。それぞれの問題領域パッケージには、まとまって論理データ・モデルを構成するエンティティー・クラスが含まれます。共通点はありますが、**設計モデル**内のパッケージ構造への直接的なマッピングはありません。

8.2 物理データ・モデル

物理データ・モデルには、XDE Data Modeler のフォワード・エンジニアリング機能を使用してデータベースを実装する際に用いられる詳細なデータベース表とストアード・プロシージャーの設計が含まれます。物理データ・モデルは、データベースの物理ストレージ構成を定義する際に使用されるモデル要素からも構成されます。一般的に、モデル要素には、対象となるストレージ・メディアのデータベース表の物理レイアウトを構成するデータベースと表スペースが含まれます。

XDE 物理データ・モデルを作成する際に、データベース設計者は適切な対象データベースを選択する必要があります。サポートされるデータベースには、DB2 MVS、DB2 UDB、Oracle、Sybase、および SQL Server が含まれます。XDE は、XDE モデル・ファイル名を選択されたデータベースのデフォルトとします。このホワイト・ペーパーの「物理データ・モデル」の例では、XDE モデル・ファイル名が「Physical Data Model」に更新されています。データベース設計者は、「物理データ・モデル」を作成するときにデフォルト名を受け入れることもできます。

図 15 に XDE 物理データ・モデルの推奨構造の例を示します**エラー! 参照元が見つかりません。**す。

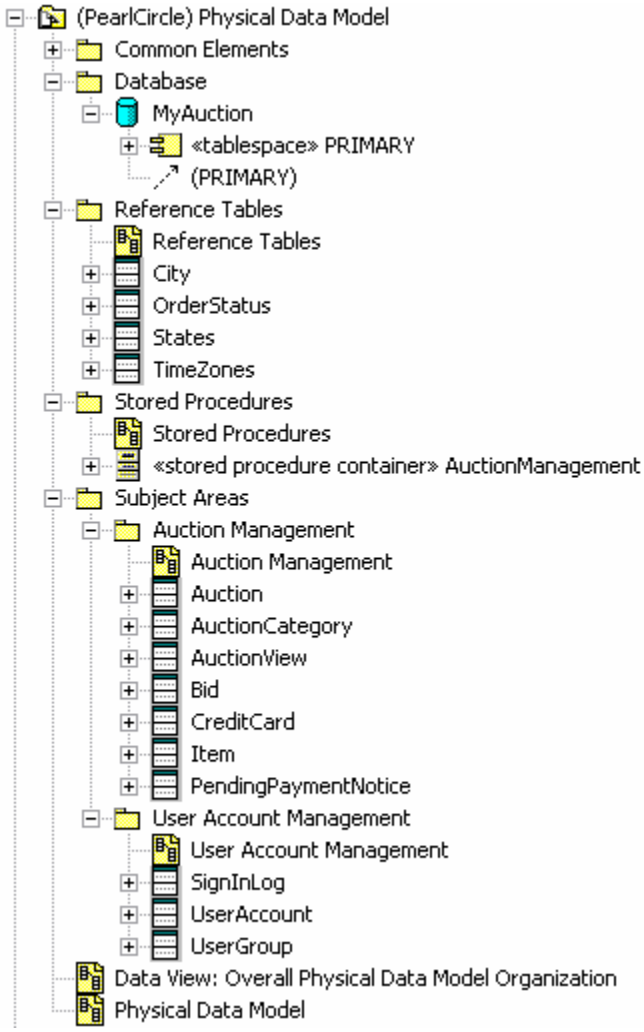

図 15 : XDE 物理データ・モデルの構造

「Common Elements」パッケージには、問題領域に渡って使用されるデータベース表とビューが含まれます。

「Database」パッケージには、データベースの物理ストレージ構成を定義するモデル要素が含まれています。ここには、対象となるストレージ・メディアのデータベース表の物理レイアウトを構成するデータベースと表スペースが含まれます。表スペースは、データベース内の表を論理的にグループ分けする際に使用されます。表スペースを定義する際は RUP をガイドラインとして参照してください。「Database」パッケージは、アプリケーションの複雑さの度合いによって、必要に応じて詳細なパッケージに分割することができます。

図 15 に示す例では、「Database」パッケージには、MyAuction という単一のデータベース、それに関連付けられた表スペースの PRIMARY、およびと表の実現関係が含まれています。表スペースには、データベース・プロジェクトに応じて任意の名前を付けることができます。MyAuction データベースに関しては、PRIMARY という 1 つの表スペースのみが定義されています。フォワード・エンジニアリングが実施されると、データベースの表スペースを用いて実現関係を通じてデータベースへリンクされた表が (データベースもしくは DDL 内に) 作成されます。

「Reference Tables」パッケージには、アプリケーションに必要な「不変」のデータ情報を持つ静的なデータ表が含まれます。

「Stored Procedures」パッケージには、データベースのストアード・プロシージャーを表すすべてのクラス («stored procedure container »クラスおよび関連する « stored procedure» 操作) が含まれます。単一の表に

関連するストアード・プロシージャーは、「ストアード・プロシージャー中心」もしくは「表中心」のビュー[5]のどちらを表すかに応じて、それぞれ「Stored Procedure」パッケージもしくはストアード・プロシージャーが参照する表を伴う「Subject Areas」パッケージにパッケージ化することができます。

「Subject Areas」パッケージは、論理的に関連する表とビューのまとまりをグループ化したパッケージを含みます[6] 。問題領域パッケージで表と一緒にビューを作成することが推奨されます。これは編成上の理由からのみ推奨されます。ビューは、使用される問題領域に保持しておくのが有効です。こうすることで、表と同じ問題領域にビューを置くことになります。図 15 に示す例には、「Auction Management」と「User Account Management」の 2 つの問題領域パッケージがあります。アプリケーションの複雑さの度合いにより、問題領域パッケージの数が変わってきます。しかし、一般的には、論理データ・モデルの問題領域パッケージから物理データ・モデルの問題領域パッケージが「導き出され」ます。論理データ・モデルの問題領域は、物理データ・モデルの問題領域の抽象概念となります。

問題領域パッケージの表には、表のために定義された列とトリガーが含まれます。表は以下のいずれかの方法で作成されます。

- XDE クラスから表への変換機能
- 既存のデータベース機能への XDE リバース・エンジニアリング[7]
- Database Designer による手動作成

既存データベースをリバース・エンジニアリングすると、XDE 物理データ・モデル内にスキーマ・パッケージが作成されます。これらのパッケージの名前は、リバース・エンジニアリングされたデータベースのデータベース所有者[8] に基づいて付けられます。リバース・エンジニアリングされた表を「Subject Areas」パッケージ内の問題領域パッケージに移動し、リバース・エンジニアリングされたスキーマ・パッケージを削除することが推奨されます。表を問題領域パッケージへ移動すると、表が機能的に編成され、Database Designer を用いて必要に応じた表の更新を行うことができます。

名前に「View」が含まれる図は、アーキテクチャーのデータ・ビューを文書化するために使用されます。「Data View: Overall Physical Data Model Organization」図は、XDE 物理データ・モデルの主要なパーティション (パッケージ) として表現されているとおり、物理データ・モデルの高レベルのデータ編成を文書化するために使用されています。アーキテクチャー・ビューについては RUP のマニュアルを参照してください。

8.3 ドメイン・モデル (オプション)

ドメイン・モデルは、データベース用のユーザー定義のデータ・タイプを格納するために使用されるオプションの XDE モデルです。ドメインにより、Database Designer は、データベース設計を通して要素のプロパティーを再利用できます。ドメインは、Database Designer でデータベース全体に渡り一貫性を保ちながら列のプロパティーを文書化する際に使用されます。列の名前は表に定義されています。ドメインは列の *TypeExpression* を定義するために使用されます。

---

[5] 表中心のビューは、データベースの設計/操作すべてを 1 つのビューで表わすため理解が容易になります。ストアード・プロシージャー中心のビューは、ストアード・プロシージャーの検索、変更/保守を単純化します。

[6] 論理および物理データベースの問題領域パッケージを保守する必要が出るため、物理データ・ モデル内の問題領域パッケージの使用について疑問を覚える人もいます。物理データ・モデルの問題領域は、論理データ・モデル (使用されている場合) との間の一貫性を保つためにあります。また、物理データ・モデルが「大規模」で、論理データ・モデルが存在しない場合には特に有用です。そのような場合、問題領域パッケージは、クラスから表への変換によって生成された表を管理するために使用できます。

[7] 通常、データベースは一度リバース・エンジニアリングされ、その後の更新はすべて XDE の Compare と Sync 機能により同期されます。

[8] XDE において、データベース所有者は <<database>> コンポーネントのプロパティーとして取り込まれます。接続ストリングの一部となる Location プロパティーの中にスキーマ属性があります。データベースをリバース・エンジニアリングすると、通常これがデータベース所有者になります。

図 16 に XDE ドメイン・モデル[9] の推奨構造の例を示します。

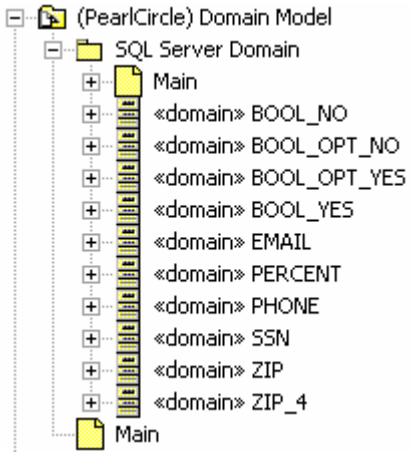

図 16: XDE ドメイン・モデルの構造

この例では、「SQL Server Domain」パッケージ内に編成された SQL Server ドメイン値を示しています。Database Designer は、数多くのドメインを定義する場合、「SQL Server Domain」パッケージ下のパッケージを用いてドメインを編成する必要があるかもしれません。

9. 実装モデル

RUP で定義される**実装モデル**には、実装要素を表す UML 要素のようなビジュアル表現からファイル・システムの物理ファイルのような物理表現に至るまでの実装要素が含まれます。XDE で**実装モデル**を取り扱うことの長所は、このような水準の異なる表現の間をラウンドトリップ・エンジニアリングにより自動的に同期をとりながら行き来できる点にあります。

XDE における**実装モデル**は、図 17 に示すような複数の XDE モデル・ファイルから構成されます。

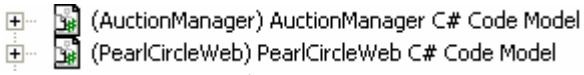

図 17:実装 XDE モデル

この例では、**実装モデル**を記述するのに、以下に示す XDE モデル・ファイルを使っています。

- “AuctionManager C# Code Model” には Auction Manager の**実装サブシステム**を構成する Microsoft® Visual C# のコード要素を含む XDE コード・モデル・ファイルがまとめられています。このモデルの要素は XDE ラウンドトリップ・エンジニアリングの適用対象となっています。

- “PearlCircleWeb C# Code Model” XDE モデル・ファイルには、PearlCircle Web の**実装サブシステム**を構成する ASP.NET C# のコード要素 (Web フォーム、Web コントロール、HTTP ハンドラー) がまとめられています。このモデルは VS.NET Web アプリケーション・プロジェクトに対応しています。このモデルの要素は XDE ラウンドトリップ・エンジニアリングの適用対象となっています。

1 つの VS.NET プロジェクトに対して XDE コード・モデル・ファイルは 1 つのみであることに気を付けてください。プロジェクトと個々のモデルの数はアーキテクチャーに基づいて選択していくことになり、プロジェクトごとに異なる場合があります。詳しくは XDE のオンライン・ヘルプを参照してください。

[9] XDE では、DB2、Oracle、Sybase、SQL Server 等複数のベンダーのデータベースがサポートされています。ドメイン XDE データ・モデルを作成する場合は、Database Designer が適切なベンダーのデータベースを選択することでドメイン XDE データ・モデルが作成されます。XDE は、選択されたデータベース・ベンダー用のドメインのデフォルト・リストを作成します。

各パッケージについてはこの後の節で詳しく説明します。

9.1 C# コード・モデル - クラス・ライブラリー・プロジェクト用

「C# コード実装モデル」には C# で実装された要素がまとめられています。

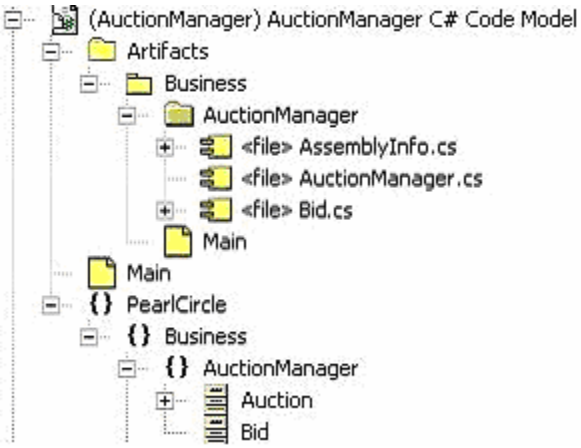

図 18:「Auction Manager C# Code Model」の構造

この例を見ると、「Auction Manager C# Code Model」の構造が「全体レベルの設計モデル」の構造 (セクション 7 で説明) を反映していることがわかります。.NET の名前空間に対応する「全体レベルの設計モデル」の各パッケージが、それぞれ対応する 1 つずつの C# (サービス・コンポーネントとその他の C# 支援クラスを含む) コードに実装されています。XDE における .NET 名前空間は、*namespace* ステレオタイプ (中括弧 '**{}**' アイコン) に従ったパッケージとしてモデリングされます。C# プログラム言語では名前空間の名称にスペースを含むことができないため、C# の名前空間名は対応する「全体レベルの設計モデル」のモデル要素名と必ずしも同一ではありません。

図 18 に示すように「C# Code Model」にはソース・コード・ファイルのビジュアル表現が含まれています (拡張子が .cs の要素)。XDE が各コード・モデル・ファイルについて生成する *Artifacts* パッケージにはソース・コード・ファイルが含まれます。ソース・コードの各ファイルは「全体レベルの設計モデル」で定義されたクラスに対応付けされています (セクション 7 を参照)。これらは、「全体レベルの設計モデル」のクラスを実装可能な水準まで発展させ、完成度を高められています (XDE の場合はラウンドトリップ・エンジニアリングが可能な形で対応がとられています)。

図 18 を見てわかるように、「C# Code Model」構造は C# 名前空間名の先頭に会社名を使用する規則に従っています。サンプル・アプリケーションでの会社名は「Pearl Circle」です。したがって、実装要素を含むパッケージは *PearlCircle* 名前空間に置かれます。その結果、実装要素はすべて *PearlCircle* 名前空間に集められることになり、*PearlCircle* 名前空間にある C# 要素の完全修飾名の先頭には常に「PearlCircle」と付くことになります。例えば、*AuctionManager* 名前空間の完全修飾名は「*PearlCircleBusiness.AuctionManager*」になります。C# 名前空間名の先頭に会社名を付けるというこの命名規則により、サード・パーティーの C# クラス・ライブラリーを組み込んでも、名前が重複することはありません。

9.2 C# コード・モデル - Web アプリケーション・プロジェクト用

XDE Web モデルには Active Server Page .NET (ASP.NET) Web アプリケーション・プロジェクトに対応する要素がまとめられています。図 19 に、ASP.NET プロジェクトの要素に対応する「PearlCircleWeb」XDE コード・モデル・ファイルの例を示します。

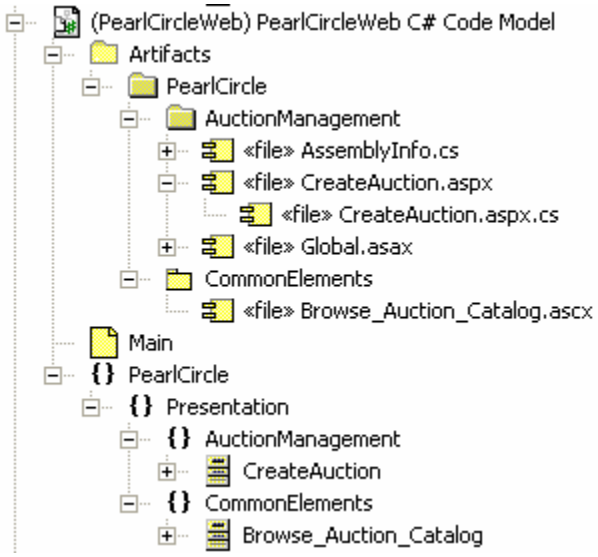

図 19:「PearlCircle Web C# Code Model」の構造

この例を見ると、「PearlCircle Web C# Code Model」の構造が「全体レベルの設計モデル」の構造 (セクション 7 で説明) を反映していることがわかります。C# (分離コード・クラスとその他の C# 支援クラスを含む) コードで実装されるコンテンツを持つ、「全体レベルの設計モデル」内のパッケージごとに .NET の名前空間が存在します。XDE における .NET 名前空間は、*namespace* ステレオタイプ (中括弧 '**{ }**' アイコン) に従ったパッケージとしてモデリングされます。C# プログラム言語では名前空間の名称にスペースを含むことができないため、C# の名前空間名は対応する「全体レベルの設計モデル」のモデル要素名と必ずしも同一ではありません。

Web フォーム、Web コントロール、HttpHandlers のような支援的 C# クラスに関連する C# の分離コード クラス (.aspx.cs または .ascx.cs) は、XDE を使えばラウンドトリップ・エンジニアリングの適用対象にできます。現在のバージョンの XDE は .aspx や .ascx の各ファイルの RTE をサポートしていません[10]。図 19 に示す「*CreateAuction*」クラスは、*CreateAuction.aspx.cs* ファイル内にある分離コード・クラスをモデリングしたものです。成果物パッケージでは、*CreateAuctiont.aspx.cs* ファイルは *CreateAuction.aspx* ファイルの下に示されます。

アーキテクチャーの点で重要な機能がすべて分離コード・クラスに含まれている場合、必要となるすべての XDE コード・モデルは自動生成されます。一方、アーキテクチャーの点で重要な機能が Web コントロール・ファイル (.ascx) に実装されているときは、このファイルに対応するクラスを手作業でモデルに追加します。後者の例は図 19の *Browse_Auction_Catalog* クラスです。このクラスは手作業で図に追加されています。

10. 配置モデル

配置モデルは「Deployment Model」と名付けられた XDE「Blank Model」ファイルで表現します。

「配置モデル」には配置環境のネットワーク構成を表すノードと接続が含まれます。また、これらのノードに配置される実装要素の識別もします。

「Deployment Model」の例を図 20 に示します。

[10] Web フォームは、単一のユニットに対し、分離コード・ファイルと .aspx ファイルの 2 つの別々のファイルで構成されます。詳細については、VS.NET の文書を参照してください。

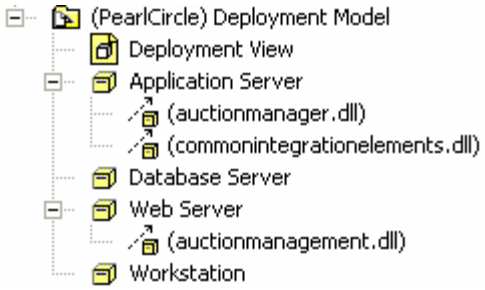

図 20:「Deployment Model」の構成

この例では、ノードとして「Database Server」、「Application Server」、「Web Server」が定義されています。auctionmanagement.dll は Web Server に配置されています。auctionmanager.dll と commonintegrationelements.dll は、Application Server に配置されています。

「Deployment Model」モデル・ファイルにはアーキテクチャー・ビュー図も含めることができます。図 20 の「Deployment View」図はアーキテクチャーの配置ビューを示します。アーキテクチャー・ビューについては RUP のマニュアルを参照してください。