Rational® XDE™ モデル構造ガイドライン (J2EE™ 用)

Rational Software ホワイト・ペーパー
TP 154, 05/03

目次

1. 概要

このホワイト・ペーパーは、Rational XDE™ Java Platform 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 でどのような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項では、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。

このバージョンのホワイト・ペーパーでは、タグ・ライブラリー開発については触れていません。

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。

複数人で開発される J2EE 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。
メモ :XDE の「プロジェクトの作成 (create project)」ウィザードを使用すると、プロジェクト作成時に複数のモデルが自動的に作成されます。実際には、WSS AD XDE を使用して *エンタープライズ・アプリケーション・モデリング・プロジェクト* を作成すると、多くのモデルを含む、この複合プロジェクト構造の大部分が自動的に作成されます。また、XDE では、モデルのコンテンツの迅速な作成を可能にするモデル・テンプレートも提供しています。

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) - “EAR 配置モデル” (Java:EAR 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” (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:*ベンダー固有のドメイン・モデル・ファイル*)
EJB プロジェクト (XDE EJB モデリング・プロジェクト)	EJB プロジェクトは、EJB の実装に必要なリソースを含みます。ここに含まれる要素は EJB モジュール (.EJB-JAR ファイル) としてパッケージされ、配置されます。 個々の EJB または EJB の集合には、それぞれ別の EJB プロジェクトを定義することができます (1 つの EJB プロジェクトには Java コード・モデルを最大 1 つしか含められません)。生成される EJB-JAR それぞれに対して EJB プロジェクトを作成することを推奨します。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。 [2]	- “EJB コード・モデル” (Java:EJB Code Model) - “EJB 配置モデル” (Java:EJB Deployment Model)
Web プロジェク	Web プロジェクトは、アプリケーションの Web リ	- “Java コード・モデル” (Java:Java

ト (XDE Web モデリング・プロジェクト)	ソースを表します。ここに含まれる要素は Web アーカイブ・ファイル (WAR ファイル) にパッケージされ、配置されます。 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 WAR 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。[3]	1.3/1.4 Code Model) - “JSP タグ・ライブラリー・モデル” (Web:JSP Tag Library Model)[4] - “仮想ディレクトリー・モデル” (Web:Virtual Directory Model)[5] - “Web 配置モデル” (Web:Web 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。

[2] XDE を WSAD で使用していて、EJB (モデリング) プロジェクトをさらに作成する場合、ウィザードで EAR のホストとなるアプリケーション・プロジェクトが要求されます。ここでは同じアプリケーション (モデリング) プロジェクトを再利用する必要があります。

[3] XDE を WSAD で使用していて、Web (モデリング) プロジェクトをさらに作成する場合、ウィザードで EAR のホストとなるアプリケーション・プロジェクトが要求されます。ここでは同じアプリケーション (モデリング) プロジェクトを再利用する必要があります。

[4] プロジェクトごとに複数のタグ・ライブラリー・モデルを作成できます。実際、.tld ファイルそれぞれに別個のモデルが必要です。このバージョンのホワイト・ペーパーには、タグ・ライブラリー開発に関する説明はありません。

[5] XDE Web プロジェクトは、複数の仮想ディレクトリー・モデルを持つことができます。

ここまでで示したプロジェクトとモデル構造の例を**図 1**に示します (モデルには固有の名前を記述します)。

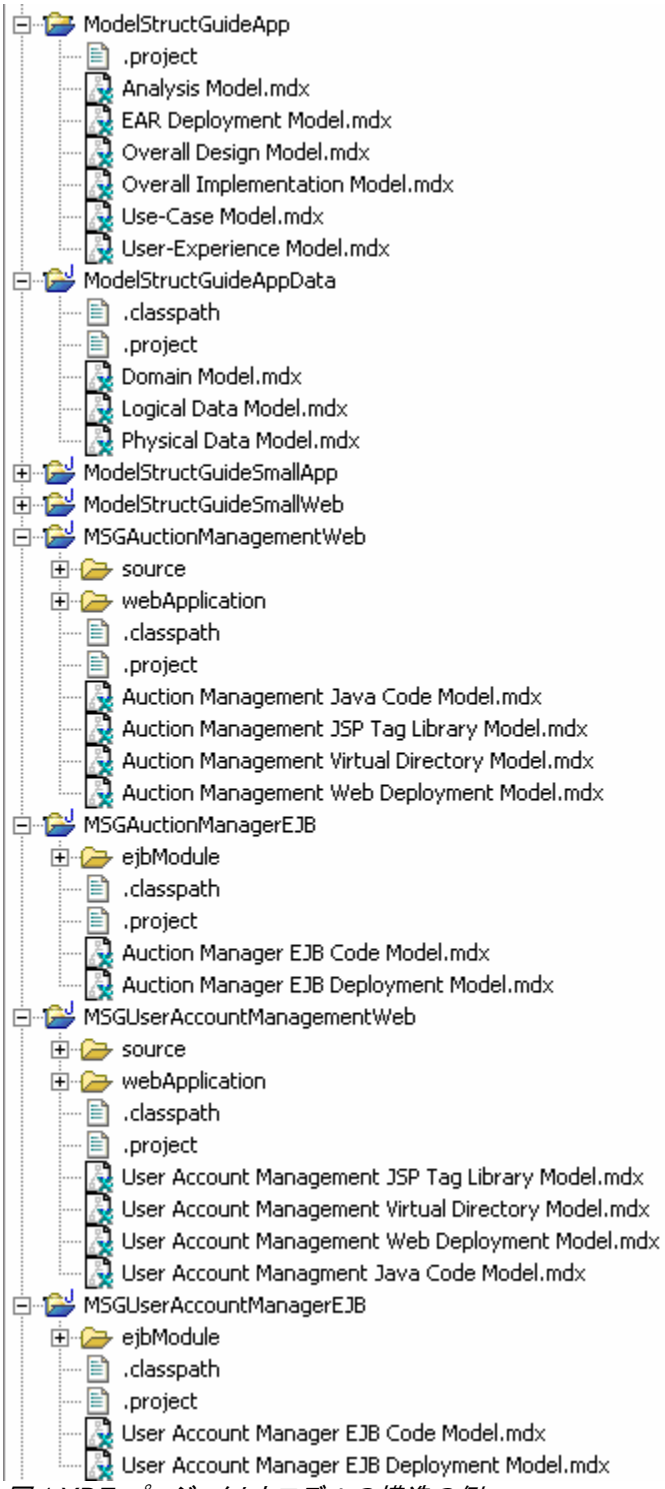

図 1:XDE プロジェクトとモデルの構造の例

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、EJB コード・モデルと仮想ディレクトリー・モデルの要素への参照で保守されます。
- 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 EJB 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、EJB 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*) - “EJB コード・モデル” (Java:EJB Code Model) - “EJB 配置モデル” (Java:EJB Deployment Model) - “EAR 配置モデル” (Java:EAR 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 Web アーカイブ・ファイル (WAR ファイル) にパッケージされ、配置されます。	- “Java コード・モデル” (Java:Java 1.3/1.4 Code Model) - “JSP タグ・ライブラリー・モデル” (Web:JSP Tag Library Model)[6] - “仮想ディレクトリー・モデル” (Web:Virtual Directory Model)[7] - “Web 配置モデル” (Web:Web Deployment Model)

[6] プロジェクトごとに複数のタグ・ライブラリー・モデルを作成できます。実際、.tld ファイルそれぞれに別個のモデルが必要です。このバージョンのホワイト・ペーパーには、タグ・ライブラリー開発に関する説明はありません。

[7] XDE Web プロジェクトは、複数の仮想ディレクトリー・モデルを持つことができます。

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。

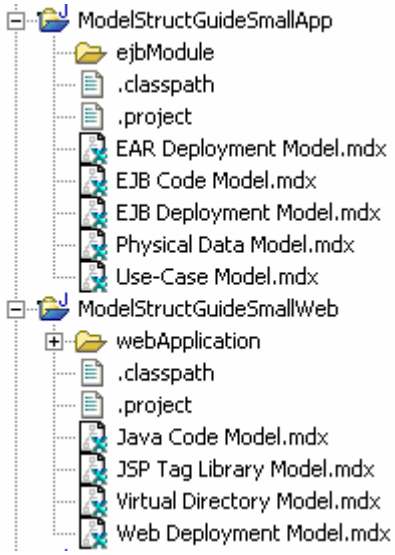

図 2:小規模の XDE プロジェクトとモデルの構造の例

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 Java 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・ペーパーで取り扱う例には、図 1 で示したプロジェクトとモデルの構造を使用します。複数の EJB プロジェクトと Web プロジェクトが定義されている点に注意してください。複数の EJB プロジェクトと Web プロジェクトの論理的根拠については、実装サブシステムの項を参照してください。

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:全体レベルの設計モデル [各 XDE ラウンドトリップ ・モデル・ファイルの設計クラス (下記を参照)]
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:全体レベルの実装モデル XDE ラウンドトリップ ・モデル[8] - EJB プロジェクト:EJB コード・モデル - Web プロジェクト:Java コード・モデル - Web プロジェクト:JSP タグ・ライブラリー・モデル - Web プロジェクト:仮想ディレクトリー・モデル
配置モデル	XDE 配置モデル[9] - アプリケーション・プロジェクト:EAR 配置モデル[10] - EJB プロジェクト:EJB 配置モデル - Web プロジェクト:Web 配置モデル

[8] 説明を簡潔にするために、このホワイト・ペーパーでは、これらの XDE モデルを表すために「XDE ラウンドトリップ ・モデル」を使用します。

[9] このホワイト・ペーパーでは、XDE モデルの説明を簡潔にするために「XDE 配置モデル」を使用します。

[10] EAR 配置モデルは、個々の XDE 配置モデルを「横断」しています。ここには、配置ノードとその接続を記述した図が含まれます。また、個々の配置モデルに定義されている、個々のアーカイブ・ファイルを配置ノードにマッピングするための図も含まれています。

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。

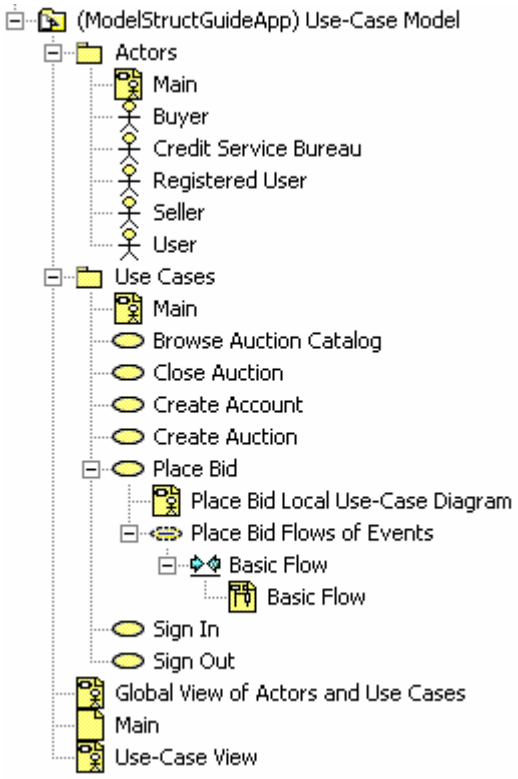

図 3:"Use-Case Model" の構造

「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図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インスタンス (例: **アクター**と**ユースケース**の相互作用) が含まれています。ユースケース・コラボレーション・インスタンスを分析モデルおよび設計ユースケースの実現の項で説明している**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。「ユースケース・モデル」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 Flows 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作成して、パッケージ内のモデル要素を整理することもできます。

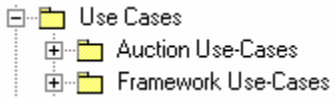

図 4:Use Cases パッケージの階層化

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**を作成するが、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[11] 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、設計モデルの項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。

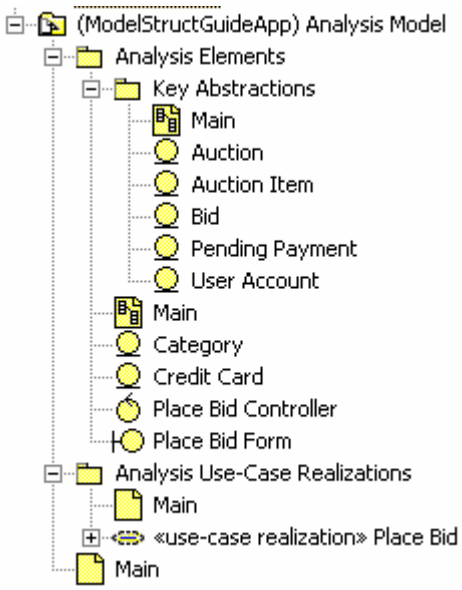

図 5:分析モデルの構造

「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

[11] 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「設計モデルのパーティション」は、XDE ラウンドトリップ・モデルの 1 つにあるパッケージである可能性があります。

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。
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、設計モデルの項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。

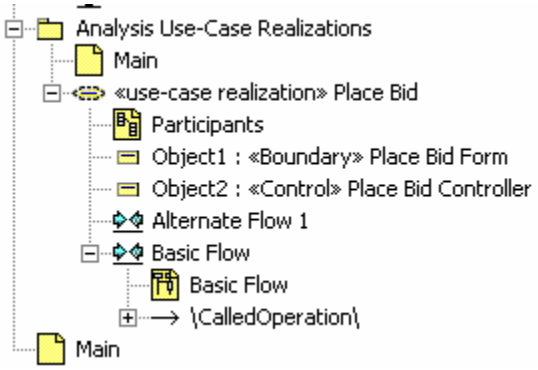

図 6:「分析ユースケースの実現」のパッケージ構造

「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。例えば、XDE EJB パターンを使用して、EJB を指定するクラスを作成できます。

「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 XDE 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・モデルの詳細については、実装モデルの項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。小規模なプロジェクトの構造の例については、XDE プロジェクトの構造の項を参照してください。

「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。

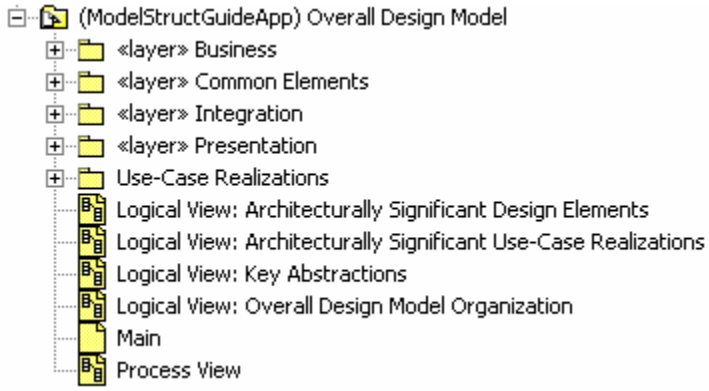

図 7:全体レベルの設計モデルの構造

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、設計レイヤーの項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。設計ユースケースの実現の項では、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、分析モデルの項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** (「設計モデル」の「Use-Case Realizations」パッケージに含まれ、設計ユースケース実現の項で取り上げています) は、これらのパッケージに含まれる設計要素に基づいて記述されています。

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。

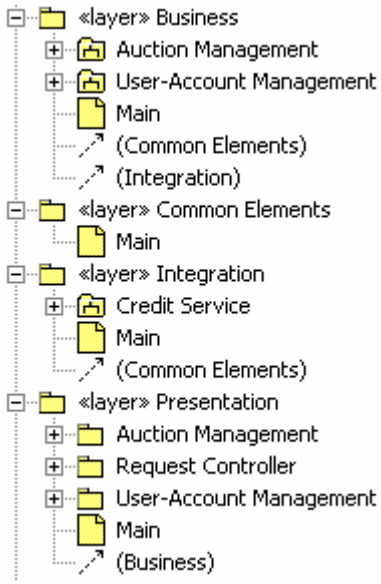

図 8:設計パッケージ内の分類例

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。J2EE 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、HTML ページ、Java Server Pages (JSPs)、サーブレットがあります。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「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項で説明しています。

「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項で説明しています。

「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。

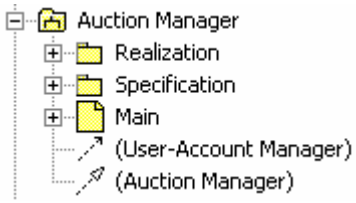

図 9:設計サブシステムの構造

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。[12] 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。

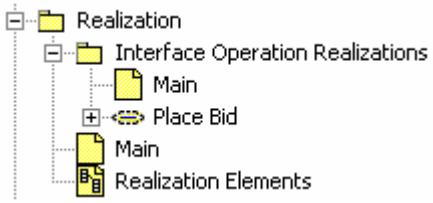

図 11:設計サブシステム実現の例

「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、「Realization」パッケージ、もしくは、ラウンドトリップ・エンジニアリングに参加する個々の XDE コード・モデル内に常駐します。詳しくは、XDE ラウンドトリップ・モデルの項を参照してください。

[12] 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[13]「Interface Operation Realizations」パッケージの例を図 12 に示します。

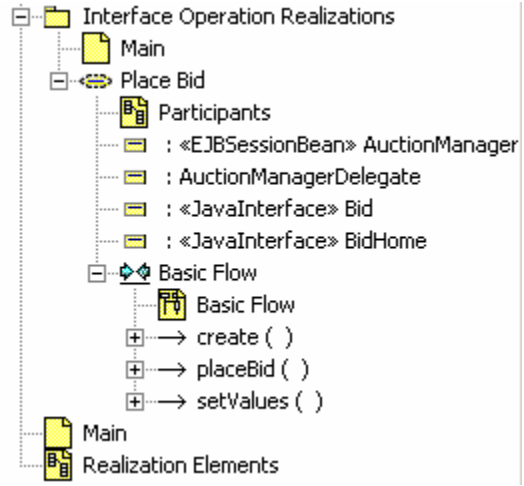

図 12:Interface Operation Realizations パッケージの例

分析レベルの**ユースケースの実現** (分析モデルの項で説明) と設計レベルの**ユースケースの実現** (設計ユースケースの実現の項で説明) 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を含むクラス図と (図 12 の「Participants」図)、これらの要素がサブシステム・インターフェース操作を共同して実行する際の相互作用の方法を示した相互作用図 (図 12 の「Basic Flow」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。

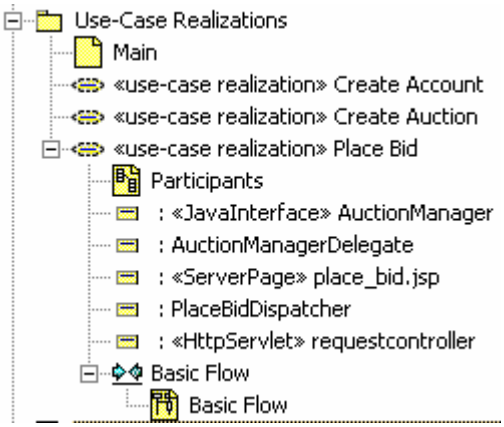

図 13:設計ユースケース実現の構造

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。

[13] 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・インスタンス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、**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、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・モデルの項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。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。

論理データ・モデルの推奨構造の例を**図 14** に示します。

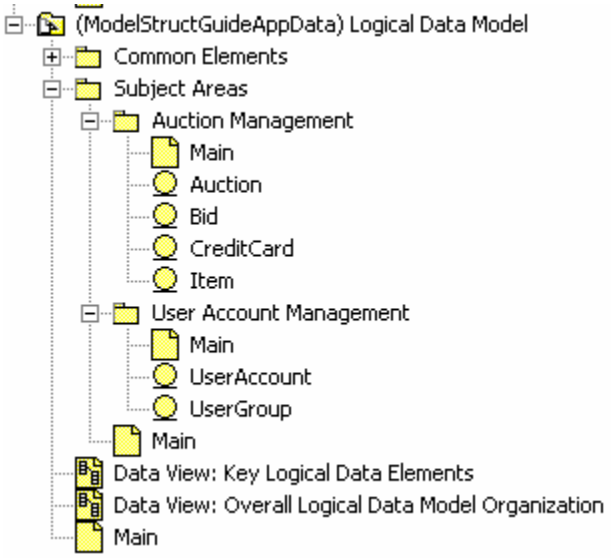

図 14:論理データ・モデルの構造

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。

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。

物理データ・モデルの推奨構造の例を**図 15**に示します。

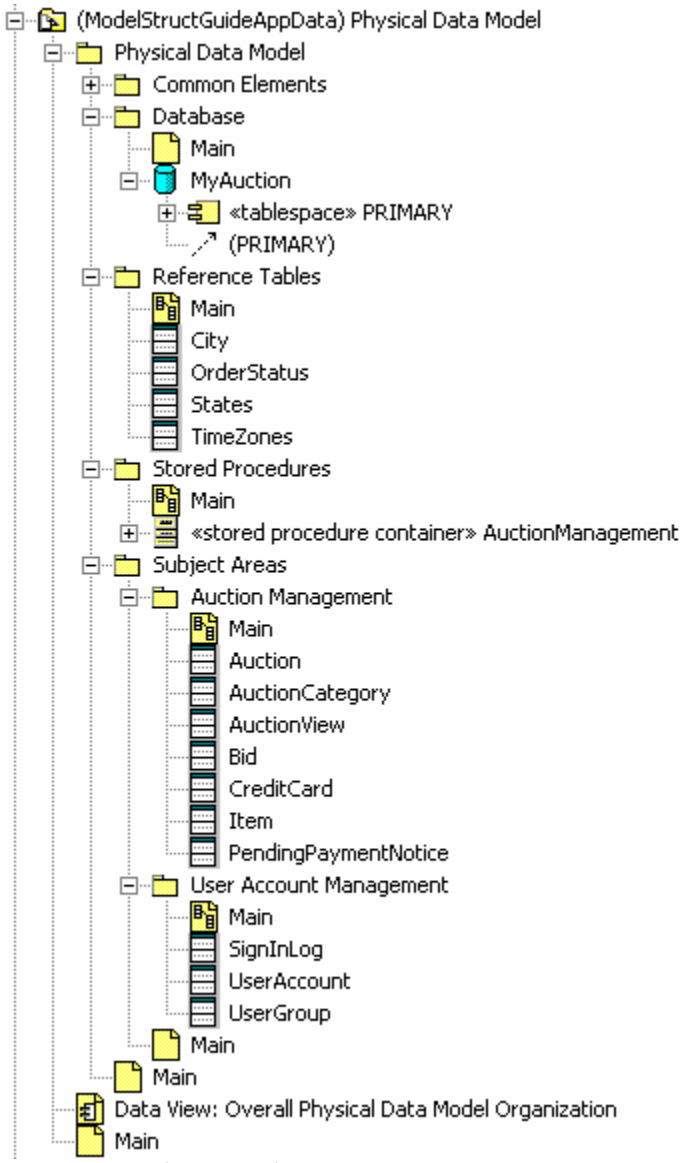

図 15:「物理データ・モデル」の構造

「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[14]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[15] 。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[16]
- 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[17] 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

---

[14] 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。

[15] 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。

[16] 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。

[17] 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。

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。

図 14 にドメイン・モデル[18]の推奨構造の例を示します。

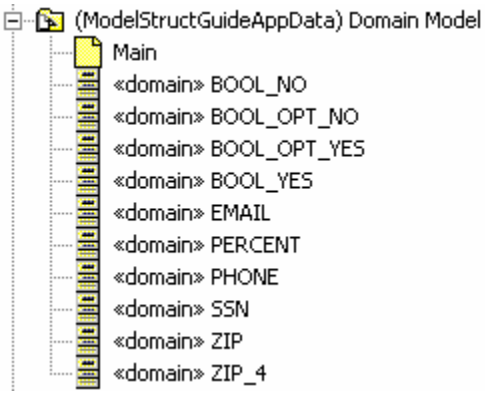

図 16:「ドメイン・モデル」の構造

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 **実装モデル**は、「全体レベルの実装モデル」と複数の XDE ラウンドトリップ・モデルの複数の XDE モデルで表わされます[19]。複数のラウンドトリップ・モデルを使用することで、様々な実装上の問題 (例えば、仮想ディレクトリー構造に対して Java のパッケージング構造を記述する場合など) を最も効率良く表すことができます。また、個々のラウンドトリップ ・モデルに対して提供される自動化機能を使用することができます (すなわち、XDE では、実装技術特有のモデル要素の作成やラウンドトリップ・エンジニアリングを使用したソース・コードとモデル要素の同期化を行う自動化機能が提供されています)。

「全体レベルの実装モデル」は、アプリケーション全体としての実装を示し、複数のラウンドトリップ・モデルに渡り使用される要素を含んでいます。これには、ラウンドトリップ設計の要素を参照する図は含まれますが、通常、モデル要素自体を「所有」することはありません。「全体レベルの実装モデル」は XDE ラウンドトリップ・エンジニアリングには参加しません。

「全体レベルの実装モデル」の保守は必須ではありません。しかし、統合の単位 (RUP では**実装サブシステム**と定義されています) などの実装の全体構造やアーキテクチャーの実装ビューの文書化の際に役立ちます。**実装サブシステム**の詳細については、実装サブシステムの項で説明しています。

XDE ラウンドトリップ・モデルの数は、定義されている XDE プロジェクトとモデルの編成に依存しています (詳しくは、XDE プロジェクトの構造の項を参照してください)。しかし、個々の**実装サブシステム**にそれぞれ別のプロジェクト (およびラウンドトリップ・モデル) を定義することもできます。XDE における**実装サブシステム**の表し方については、実装サブシステムの項を参照してください。対象となる実装言語が 1 つでチームが小規模の場合は、

[18] Within XDE, several vendor databases are supported, including DB2, Oracle, Sybase, and SQL Server.When creating a Domain XDE Data Model, the Database Designer will create the Domain XDE Data Model by selecting the appropriate vendor database.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。

[19] XDE ラウンドトリップ・モデルは合成 (ハイブリッド) モデルです。RUP **設計モデル**と RUP **実装モデル**の両方を表すために使用されます。XDE ラウンドトリップ・モデルのモデル要素は、RUP **設計クラス** (物理クラスへ直接マッピングされるクラスは**設計クラス**とみなされます) と物理的な実装ファイルを表します。XDE ラウンドトリップ・モデルの構造は、物理ディレクトリー構造を表します。

前述したとおり、1 つの XDE ラウンドトリップ・モデルを使用して RUP **設計モデル**と**実装モデル**の両方を表すこともできます。

「全体レベルの実装モデル」の例を図 17 に示します。

図 17:全体レベルの実装モデルの構造

図 17に示すとおり、「全体レベルの実装モデル」には図のみが含まれます。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・ビューについて詳しくは RUP のマニュアルを参照してください。

「Implementation View: Deployable Implementation Elements」図は、XDE 配置モデル内のノードに配置されるアーカイブ・ファイルを参照します。アーカイブ・ファイル自体は、物理的には配置モデルに含まれます。詳しくは、配置モデルの項を参照してください。

「Implementation View: Implementation Subsystems」図は、アプリケーションの実装サブシステムを参照します。図の上で、**実装サブシステム**間に依存関係を表す線を記述することもあります。これらの依存関係は、**実装サブシステム**の統合される順番を決定する**実装サブシステム**のインポートを表します。XDE における実装サブシステムの表し方については、実装サブシステムの項を参照してください。

「Implementation View: Implementation Model Structure」図は、**実装モデル**とそれらの関係を表すために使用されているすべての XDE モデルへの参照を含みます。
メモ :個々の**実装サブシステム**に対して、それぞれ別々のプロジェクトが定義されている場合、「Implementation View: Implementation Model Structure」図のコンテンツは「Implementation View: Implementation Subsystems」図と重複するため、省略することもできます。実装サブシステムの詳細については、実装サブシステムの項を参照してください。

9.1 実装サブシステム

RUP **実装サブシステム**は統合の単位です[20]。そのため、J2EE モジュールとうまく協調できます (**実装サブシステム**は J2EE モジュールにパッケージして、配置できます)。個々の J2EE アーカイブに対して XDE プロジェクトを定義することでも XDE プロジェクトの構造をセットアップすることができるため、識別した**実装サブシステム**を使用して、詳細な設計と実装をサポートするためにどの XDE プロジェクトを定義すべきかを決定することができます。具体的には、XDE において RUP **実装サブシステム**は XDE プロジェクトとして表すことができます。**実装サブシステム**を XDE プロジェクトを使用して表しているガイドラインでは、このアプローチが採用されています。

これらのガイドラインのサンプル用の**実装サブシステム**は以下のとおりです。

- 「全体レベルの設計モデル」内の各設計サブシステム用の実装サブシステム。例えば、次のようなものです。User Account Manager および Auction Manager
- Auction Management のプレゼンテーション要素用の実装サブシステム
- User Account Management のプレゼンテーション要素用の実装サブシステム

[20] **実装サブシステム**の識別のためのガイドラインは、このホワイト・ペーパーでは対象外となります。詳しくは、RUP を参照してください。

関連する XDE プロジェクトとモデルを図 18 に示します。

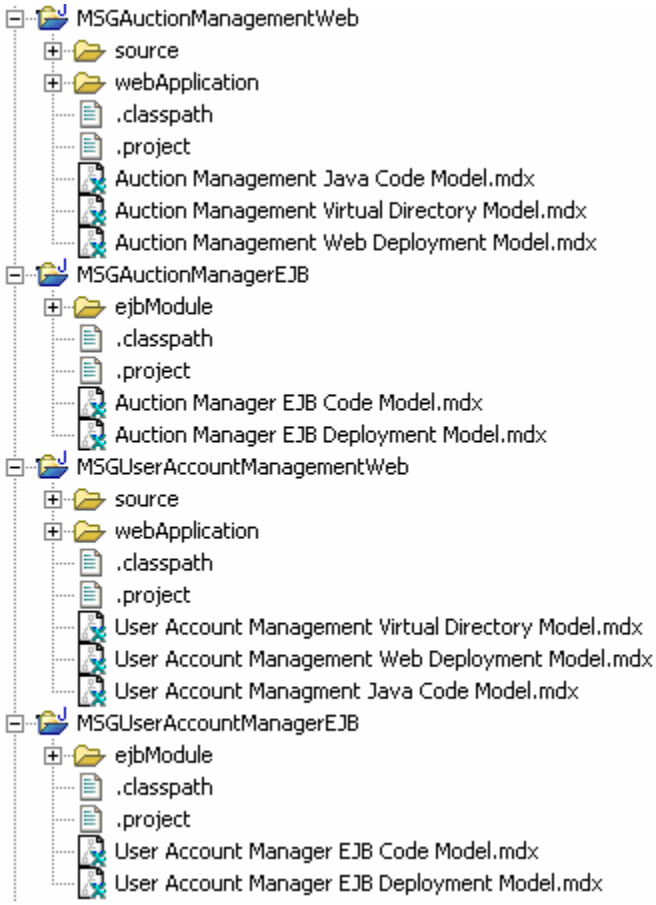

図 18:XDE プロジェクトとしての実装サブシステム

複数の Web と EJB プロジェクトが定義されている場合、プロジェクトとそこに含まれるモデル・ファイルに固有の名前を付けることを強くお勧めします。これにより、モデル内の図をより理解しやすくなり、モデル間の参照が解決されます。図 18で示す例では、**実装サブシステム**の名前がプロジェクト名およびそこに含まれる個々のモデル名で使用されています。

プロジェクトが小規模の場合、XDE では RUP **実装モデル**を XDE モデル内のパッケージとして表すこともできます。**図 19**では、XDE パッケージを使用して個々の実装サブシステムを表した場合の例 (「auctionmanager」パッケージおよび「useraccountmanager」パッケージ) を示しています。

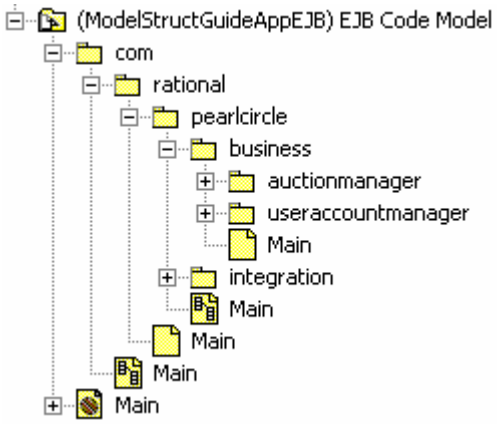

図 19:パッケージとしての実装サブシステム

9.2 XDE ラウンドトリップ・モデル

この項では、以下の XDE ラウンドトリップ・モデルの例を取り上げます。

- (EJB プロジェクト) EJB コード・モデル (EJB プロジェクト: EJB コード・モデルの項を参照)
- (Web プロジェクト) Java コード・モデル (Web プロジェクト: Java コード・モデルの項を参照)
- (Web プロジェクト) 仮想ディレクトリー・モデル (Web プロジェクト: 仮想ディレクトリー・モデルの項を参照)

通常、XDE ラウンドトリップ・モデルを構成する場合、「全体レベルの設計モデル」(設計モデルの項で説明) で示したように、モデルの構造を論理構造にできる限り近づけることが推奨されます。このときに、当然実装の制約を考慮しなければなりません。**設計モデル**と**実装モデル**の構造をできる限り近づけることで、両モデル間の追跡可能性が暗黙的になり、保守がより容易になります。**設計モデル**と**実装モデル**の間のマッピングを保守および管理する (アーキテクチャーの保守と管理の一部として) 必要があるため、これが重要になります。

XDE ラウンドトリップ・モデルの一部である**設計クラス**は、以下のいずれかの方法で作成することができます。

- EJB やサーブレット等の実装技術に依存した要素を作成するための XDE の自動化機能の使用。
  これらの要素は、最初から作成するか、**分析クラス**のような技術に依存しない要素から変換します。
- 既存の実装の XDE リバース・エンジニアリング
- 手動作成

9.2.1 EJB プロジェクト:EJB コード・モデル

EJB コード・モデルには、EJB を実装するために必要な Java リソースが含まれます (実装 Bean クラス、ホーム・インターフェース・クラス、リモート・インターフェース・クラスなど)。

EJB コード・モデルの Source Root プロパティーには、ソース・コードを置くディレクトリーを設定する必要があります。例えば、EJB コード・モデルの Source Root プロパティーに EJB プロジェクトの「ejbModule」サブディレクトリーを設定することができます[21]。

[21] XDE の「新規プロジェクト (create project)」ウィザードを使用してプロジェクトを作成すると、プロジェクト・サブディレクトリーは自動的に生成され、コード・モデルの Source Root プロパティーは自動的に設定されます。

EJB コード・モデルの例を図 20 に示します。

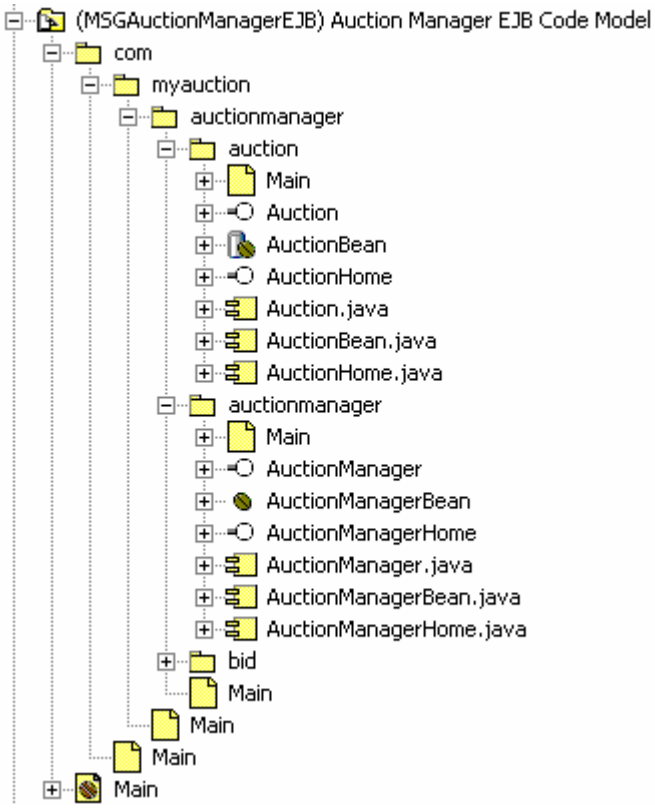

図 20:EJB コード・モデルの例

図 20 で示されている EJB コード・モデルには、Auction Manager **設計サブシステム**を実装する Java リソースが含まれています。これは、設計レイヤーの項の「全体レベルの設計モデル」にある「Business」レイヤー・パッケージの「Auction Manager」**設計サブシステム**・パッケージに対応する実装です。

図 20で示されているとおり、EJB コード・モデルの構造は、最初の Java パッケージ名にはドメイン名を使用するという規則に従っています。サンプル・アプリケーションのドメイン名は「www.myauction.com」です。したがって、実装要素を含むパッケージは、「com」パッケージの中の「myauction」パッケージに置かれています。結果として、「myauction」パッケージ内のすべての Java 要素は「com.myauction」が先頭に付く完全修飾名を持つことになります。例えば、「auction」パッケージの完全修飾名は「com.myauction.business.auctionmanager.auction」になります。ドメイン名を最初の Java パッケージの名前とする規則に従っていれば、サード・パーティーの Java クラス・ライブラリーを組み込んだとしても Java クラス名が固有となることが保証されます。

「myauction」パッケージの構造には、「全体レベルの設計モデル」(設計モデルの項で説明) の構造が反映されています。Auction Manager **設計サブシステム**を含む設計レイヤーのためのパッケージ (「business」パッケージ) と Auction Manager **設計サブシステム**を表すパッケージ (「auctionmanager」パッケージ) があります。EJB コード・モデルには関連するモデル要素を収集するためのパッケージも定義されています (「auctionmanager」、「auction」、「bid」パッケージ等)。
メモ :Java プログラミング言語ではパッケージ名にスペースを入れることができないため、Java パッケージの名前が関連する「全体レベルの設計モデル」パッケージの名前と必ずしも同じとは限りません。

図 20で示されているとおり、EJB コード・モデルにはソース・コード・ファイル (.java 要素) の視覚表示だけでなく、それらの実装要素を指定するクラスも含まれています。これらのクラスは、実装可能な水準まで発展し、完成度を高めた RUP **設計クラス**を表し、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レベルの設計クラスを表します。メモ :XDE では、EJB を指定するのに使用するすべてのクラスを自動的に作成するパターンを提供しています。

9.2.2 Web プロジェクト:Java コード・モデル

Web プロジェクト Java コード・モデルには、Java Web リソースが含まれています (JavaBeans、サーブレット、ヘルパー・クラス等)。

Web プロジェクトの Java コード・モデルの Source Root プロパティーには、ソース・コードを置くディレクトリーを設定する必要があります。例えば、Web プロジェクトの Java コード・モデルの Source Root プロパティーに Web プロジェクトの「Java Source」サブディレクトリーを設定することができます[22]。

Web プロジェクト Java コード・モデルの例を図 21に示します。

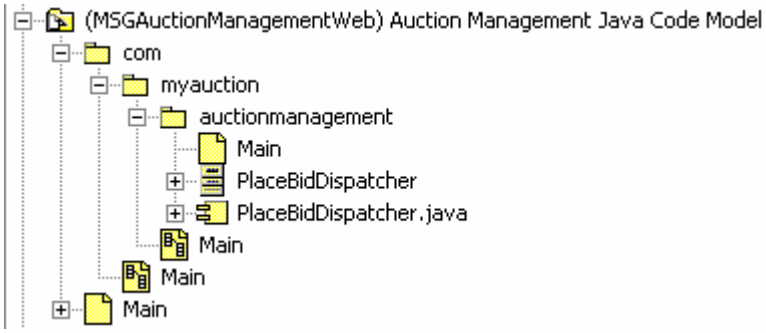

図 21:Web プロジェクト Java コード・モデルの例

図 21で示されている Java コード・モデルには、Auction Management プレゼンテーション設計要素を実装する Java リソースが含まれています。これは、設計レイヤーの項の「全体レベルの設計モデル」にある「Presentation」レイヤー・パッケージの「Auction Management」パッケージに対応する Java の実装です。

図 21で示されているとおり、Web プロジェクト Java コード・モデルの構造は、最初の Java パッケージ名にはドメイン名を使用するという規則に従っています。サンプル・アプリケーションのドメイン名は「www.myauction.com」です。したがって、実装要素を含むパッケージは、「com」パッケージの中の「myauction」パッケージに置かれています。結果として、「myauction」パッケージ内のすべての Java 要素は「com.myauction」が先頭に付く完全修飾名を持つことになります。例えば、「auctionmanagement」パッケージの完全修飾名は「com.myauction.presentation.auctionmanagement」になります。ドメイン名を最初の Java パッケージの名前とする規則に従っていれば、サード・パーティーの Java クラス・ライブラリーを組み込んだとしても Java クラス名が固有となることが保証されます。

「myauction」パッケージの構造には、「全体レベルの設計モデル」(設計モデルの項で説明) の構造が反映されています。設計レイヤー用に Auction Management プレゼンテーション要素を含むパッケージ (「presentation」パッケージ) があります。そして、Auction Management プレゼンテーション要素を表すパッケージ (「auctionmanagement」パッケージ) があります。
メモ :Java プログラミング言語ではパッケージ名にスペースを入れることができないため、Java パッケージの名前が関連する「全体レベルの設計モデル」パッケージの名前と必ずしも同じとは限りません。

図 21で示されているとおり、Web プロジェクトの Java コード・モデルにはソース・コード・ファイル (.java 要素) の視覚表示だけでなく、それらの実装要素を指定するクラスも含まれています。これらのクラスは、実装可能な水準まで発展し、完成度を高めた RUP **設計クラス**を表し、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レベルの設計クラスを表します。

9.2.3 Web プロジェクト:仮想ディレクトリー・モデル

仮想ディレクトリー・モデルには、Java のソース・コード以外の Web リソース (JSP、HTML ページ等) が含ま

[22] XDE の「新規プロジェクト (create project)」ウィザードを使用してプロジェクトを作成すると、プロジェクト・サブディレクトリーは自動的に生成され、コード・モデルの Source Root プロパティーは自動的に設定されます。

れます。XDE Web プロジェクトは、複数の仮想ディレクトリー・モデルを持つことができます。仮想ディレクトリー・モデルを複数持つと、仮想ディレクトリーを複数持つ J2EE アプリケーションを開発することができます。そのようなアプリケーションでは、結果としてできる Web サイトは、共通のルートを持たないディレクトリーへと物理的に分割されています。例えば、オンラインの小売店において www.mystore.com をカタログ・ショッピングに使用し、order.mystore.com を注文状況の監視用に使用することができます。

仮想ディレクトリー・モデルの Source Root プロパティーは、Web コンテナーに配置される Web リソースが置かれるディレクトリーに設定する必要があります。例えば、仮想ディレクトリー・モデルの Source Root プロパティーに、Web プロジェクトの「Web Content」サブディレクトリーを設定することができます[23]。

仮想ディレクトリー・モデルの例を図 22 に示します。

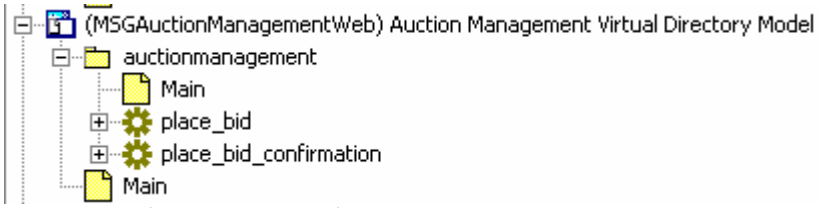

図 22:仮想ディレクトリー・モデルの構造

図 22で示されている仮想ディレクトリー・モデルには、Auction Management プレゼンテーション設計要素を実装する Java 以外のソース・コード・リソースが含まれています。これは、設計レイヤーの項の「全体レベルの設計モデル」にある「Presentation」レイヤー・パッケージの「Auction Management」パッケージに対応する Java 以外のソース・コードでの実装です。

この例において仮想ディレクトリー・モデルの構造は、「全体レベルの設計モデル」の構造 (設計レイヤーの項で説明) を反映しています。「全体レベルの設計モデル」内の各パッケージには、JSP や HTML ページのように、Web コンテナーに配置される Java 以外の要素で実装されるコンテンツを持つパッケージがあります。Web サーバーからアクセスされるディレクトリーの命名により課せられる制約により、仮想ディレクトリー下のディレクトリーの名前は、必ずしも「全体レベルの設計モデル」内のものと同一名を持つとは限りません。

図 22で示されているとおり、仮想ディレクトリー・モデルには実装要素を指定するクラスの視覚表示が含まれています。これらのクラスは、実装可能な水準まで発展し、完成度を高めた RUP **設計クラス**を表し、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レベルの設計クラスを表します。EJB コード・モデルや Web プロジェクト Java コード・モデルと異なり、XDE では仮想ディレクトリー・モデル内の関連するソース・コード・ファイルの視覚表示は生成しません。関連するソース・コード・ファイルの名前は、クラスのプロパティーとして保守されます。

10. 配置モデル

RUP **配置モデル**は複数の XDE モデルで表され、配置する要素を含む XDE プロジェクトごとに「EAR 配置モデル」と XDE 配置モデルの集合が 1 つずつあります。複数の配置モデルを使用することで、XDE の配置の自動化機能を利用できます (XDE では、J2EE アーカイブ・ファイルと配置記述子の作成と改良を行う自動化機能、およびこれらのモデル要素を、アーカイブにソースがパッケージされているモデル要素と同期させる自動化機能を提供しています)。

この項では、以下の配置モデルの例を取り上げます。

- (アプリケーション・プロジェクト) EAR 配置モデル (EAR 配置モデルの項を参照)
- (EJB プロジェクト) EJB 配置モデル (EJB 配置モデルの項を参照)
- (Web プロジェクト) Web 配置モデル (Web 配置モデルの項を参照)

[23] XDE の「新規プロジェクト (create project)」ウィザードを使用してプロジェクトを作成すると、プロジェクト・サブディレクトリーは自動的に生成され、仮想ディレクトリー・モデルの Source Root プロパティーは自動的に設定されます。

以下に、XDE での配置に関する基本的な注意点を示します。

- 配置記述子は、XDE 配置モデル内で明示的にファイル (UML 成果物) として表されてはいません。代わりに、XDE 配置モデルのコンテンツで表されています。XDE では、モデルの配置記述子ファイルに記述されたコンテンツ情報を特定するために、配置モデルに含まれる要素とそれらの要素に割り当てられたプロパティー値を使用します。
- XDE では、1 つの EJB JAR や WAR のみを配置する場合であっても、すべての配置に EAR 配置モデルが使用されます。これには、すべての配置に EAR が必要となる多くのアプリケーション・サーバーの制約が反映されています。したがって、EJB JAR や WAR のみをモデリングしている場合であっても、XDE がサポートするアプリケーション・サーバーへの配置の際は EAR 配置モデルを使用する必要があります。ただし、XDE ではあらゆるアーカイブをファイル・システムに*エクスポート*できます。この機能は、XDE でサポートされないアプリケーション・サーバーに配置する際に使用することができます。アーカイブのエクスポートを行うと、サーバー固有のツールを起動して配置を完成させることができます。
- さまざまな配置環境 (テスト環境、稼働環境など) に合わせて、異なる J2EE アーカイブ・ファイルを定義できます。これらのアーカイブは、同一の XDE 配置モデル上でも、異なる XDE 配置モデル上でも定義することができます。XDE の自動化機能は両方をサポートしていますが、プロジェクトにはデフォルト配置モデルが定義され、配置モデルごとにデフォルト・アーカイブが定義されます。しかし、これらの定義は XDE 配置モデル内にモデリングされているため、変更を加えることができます。いずれのアプローチを選択したとしても、EJB 配置モデルを EJB プロジェクト内に定義し、Web 配置モデルを Web プロジェクト () 内に置かなくてはなりません。[24]

10.1 EAR 配置モデル

「EAR 配置モデル」には、J2EE アプリケーション・アーカイブ・ファイル(.EAR ファイル)、EAR 配置記述子の情報、EAR を配置するノードの視覚表示が含まれます。「EAR 配置モデル」には、EAR 内に含まれる J2EE モジュール (他の配置モデルからの) を示す図も含まれます。

「EAR 配置モデル」は、アプリケーション全体の配置構成やアーキテクチャーの配置ビューを記述する際にも使用することができます。「EAR 配置モデル」は、すべての配置ノードとその接続を示した図や、どのノードにどのアーカイブを配置するかを示した図を含むこともできます。そのような図は、すべての個々の配置モデルの要素 (ノードおよびアーカイブ) を参照することになります。

[24] EJB 配置モデルは EJB プロジェクト内になければなりませんが、これに対応する EJB コード・モデルと*同一*のプロジェクトでなくても構いません。実際、異なるコード・モデルからの EJB を 1 つの配置モデルに「混ぜ合わせたり、組み合わせたり」することができます。Web モデルについても同様です。

「EAR 配置モデル」の例を**図 23**に示します。

図 23:EAR 配置モデル

「Deployment View」図はアーキテクチャーの配置ビューを表します。ここには、配置ノードとその相互接続、また、これらのノードにどの J2EE アーカイブを配置するかの情報が含まれます。場合によっては、この図には、J2EE アプリケーション・アーカイブとそれが配置されるアプリケーション・サーバー・ノードだけが含まれます。しかし、独立した J2EE モジュール・アーカイブを特定のノードに配置する場合には、この図はどのノードにどのアーカイブを配置するかを示していなくてはなりません。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。

10.2 EJB 配置モデル

EJB 配置モデルには、EJB コンポーネント、EJB-JAR 成果物、EJB 配置記述子の情報が含まれています。EJB 配置モデルには、どの実装要素 (EJB コード・モデルからの) が EJB-JAR に含まれるかを示す図も含まれていることがあります。正確には、EJB 配置モデル内で EJB コンポーネントが実装要素を表すのに使用され、この EJB コンポーネントが EJB-JAR にマッピングされています。

EJB 配置モデルの例を**図 24**に示します。

図 24:EJB 配置モデルの例

10.3 Web 配置モデル

Web 配置モデルには、Web コンポーネント、WAR アーカイブ・ファイル、Web 配置記述子の情報が含まれて

います。Web 配置モデルには、どの実装要素 (Web プロジェクト・ラウンドトリップ・モデルからの) が WAR に含まれているかを示す図も含まれていることがあります。正確には、Web 配置モデル内で Web コンポーネントが実装要素を表すのに使用され、この Web コンポーネントが WAR にマッピングされています。

Web 配置モデルの例を**図 25**に示します。

図 25:Web 配置モデル

Rational®
the software development company

Dual Headquarters:
Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

Toll-free: (800) 728-1212
E-mail:info@rational.com
Web:www.rational.com
International Locations:www.rational.com/worldwide