模型构造指南
针对 Rational Software Modeler、Rational Systems Developer 和 Rational Software Architect（面向“传统 RUP”）

白皮书

作者 Bill Smith，模型驱动开发，IBM Rational Software

版本 7.0
2006 年 3 月 24 日

目录

1. 简介

目标读者

本白皮书旨在支持 Rational Software Architect（RSA）、Rational Systems Developer（RSD）和 Rational Software Modeler（RSM）产品（统称为 “ **RS*x*** ”）的用户，尤其面向那些想要将 Rational Unified Process（RUP®）中的建模指导信息应用于其 RS*x* 实践的用户。如果您是 Rational Software Modeler（RSM）的用户，您将发现此白皮书很有用，但同时应当注意：某些章节所反映的功能只在 RSA 和 RSD 中提供。

本白皮书着重介绍如何在 RS*x* 中创建一组新的模型。如果您刚接触 RS*x*，但已经是 Rational Rose 或 Rational XDE 的用户，并且打算从这些产品导入模型，则还会发现此白皮书是重构所导入模型的指导信息来源。

本白皮书假定您对 RUP 和 UML 有一定程度的了解。它还假定您熟悉 RS*x* 的基本概念和 “ 操作理论 ”，您可以在 “ **The New IBM Rational Design and Construction Products for Rose and XDE Users** ” 中获取这些内容。

目的

RUP 描述了一组模型（例如用例模型、分析模型和设计模型），它们代表关于系统的问题域和解决方案域的良好定义的透视图。这组模型的实用性已在许多现实项目中得到了验证。即使您不遵循 RUP，这些模型结构仍然值得考虑。本白皮书描述了如何使用 RS*x* 实现它们。

虽然也可以考虑其他建模方法，但是支持这些方法的模型构建指导信息超出了本白皮书的讨论范围。例如，可以使用 “ 面向服务的体系结构 ” 的 “ 业务驱动开发 ” 建模方法。它可以从一个业务流程模型（可能表示为 “ UML 2 活动 ” 模型）开始，然后径自执行下去以指定一组服务的合同，这些服务会使业务流程中的任务自动执行。该方法所建议的模型结构与本白皮书中所述的那些有着很大的不同。

重要的是理解本白皮书中所述的项目和模型结构是指南，而非规定。至于是否决定在 RS*x* 中对特定 RUP 工件建模，是您自己的开发流程所要考虑的事项。该决策通常视具体的项目而定。此外，您应当认识到 RUP 并非是一组一成不变的流程规则。它是一个流程*框架*，可在其中阐明流程定义。此类流程定义可以从非常正式到非常不正式。

您使用 UML 建模的方式也可以从非常正式到非常不正式。您可以将您的模型视作要在构造期间严格遵循的正式体系结构图。也可以将您的模型视作用于提出设计大致轮廓的草图，而一旦项目转入实施阶段，就可以考虑将这些模型废弃。RS*x* 能够在这些流程和建模范围中的任何一端对您提供支持。本白皮书中的指南并无意束缚您的思想。它们只是为了帮助您了解如何使用 RS*x* 的功能以促成最适合于您的流程。

请注意，RS*x* 使您能够不仅仅将模型用作蓝图，还能将其用作规范，可从中自动生成实施的重要组成部分。这是通过 “ 模型到模型 ” 和 “ 模型到代码 ” 转换实现的。使用转换来执行 “ 模型驱动开发 ”（MDD）这一做法带来了对模型结构的特别关注。如果您要使用模型和转换来执行 “ 模型驱动开发 ”，则还应当在 developerWorks® 的 Rational “ design and construction ” 区域中查找特定于 RS*x* MDD 的资源。

范围

本白皮书介绍了如何在 RS*x* 中表示 RUP 模型工件。它还提供了有关这些工件的内部组织结构指南。其目的*不是*为了

- 重申 RUP 模型工件的概念基础或提供这些工件的详尽描述
- 描述用于指定相关 RUP 工件的详细语义或图示内容的过程或技巧。

关于如何对 RUP 工件内容进行定义、开发和建模的与工具无关的信息，请参阅 RUP。

关于用于开发 RS*x* 模型内容的特定于工具的方法的信息，请参阅

- 产品文档（教程、样例、联机帮助）
- RUP 配置中的工具向导（本白皮书是其组成部分）
- developerWorks 上与 RS*x* 相关的资源

印刷约定

对从 IBM Rational Rose 或 XDE 迁移而来的 RS*x* 用户有用的讨论以侧栏形式呈现，它们位于带边框且背景为浅灰色的文本框内：

> ***XDE/Rose***
> 对先前的 XDE 或 Rose 用户有用的讨论。

白皮书的结构

“ 基本概念和术语 ” 一节将确立工作词汇表，并提供一些关于如何在 RS*x* 产品中实施模型的常规信息。

接下来的 “ RUP 模型到 RS*x* 模型的映射 ” 一节将讨论 RS*x* 如何支持由 RUP 定义的模型类型。

随后的几节提供了关于构造各种类型的模型的指导信息。其中的几节讨论了使用模型时，根据在流程、建模方法和体系结构控制方面所需的严格程度，所能够采用的不同方式。

最后将讨论与在团队中使用模型有关的问题。这涉及了一些策略，这些策略用于管理规模，以及支持在团队成员间共享模型，以便最大程度地降低文件争用与合并。

2. 基本概念和术语

模型

在 RUP 中，模型被定义为 “ 基于特定透视图的问题域或解决方案域的完整规范 ” 。问题域或系统可以由一些模型来指定，这些模型代表了关于域或系统的不同透视图。 RUP 提议了一组特定的模型：

- 业务用例模型
- 业务分析模型
- 用例模型
- 分析模型（可以包含在设计模型中）
- 设计模型
- 实施模型
- 部署模型
- 数据模型

RUP 与工具无关。对于 RUP 来说，模型可以是白布或白板上的一副图画、可以是建模工具中的内容，甚

至可以是头脑中的图像。从 RUP 的观点看，模型是一个逻辑概念。

在 RS*x* 环境中，我们可以从逻辑角度讨论模型，但也可以从物理角度对它们进行讨论。假设您有多个团队正在处理两个应用程序，其中一个团队由三个分析人员组成，他们负责处理考勤卡管理应用程序，而另一个团队由五个分析人员组成，他们负责处理呼叫中心应用程序。两个团队现在的工作都是捕获需求，并使用 RS*x* 进行用例建模。以 RUP 的观点表达，您应当说一个团队正在构建 " 考勤卡应用程序的用例模型 "，而另一个团队正在构建 " 呼叫中心应用程序的用例模型 "。但是如果这些团队使用的是 RS*x*，则认识到他们的模型具有物理表现形式是很重要的。这就是下一节的主题。

建模文件

RS*x* 模型以文件的形式保存。（ 在 Eclipse 术语中，文件被视为一种 " 资源 "[1]，所以如果您在本白皮书或其他来源中遇到 " 建模资源 " 这个术语, 则它的含义就是 " 建模文件 "）。从广义上讲，RS*x* 支持两种建模文件：

- **预实施建模文件**在主机操作系统的文件系统中存储为单独的文件。它们的文件扩展名为 " .emx "。这些文件包含：
    - UML 语义元素（类、活动、关系...）
    - 描述 UML 语义元素的 UML 图（还*可能*描述对诸如 Java、C++ 或 DLL 的其他语义域中内容的可视引用）

- **实施建模文件**在 Eclipse 工作空间中存储为 Eclipse 项目。这些项目包含：
    - 实施工件（Java 源代码、Web 页面、基于 XML 的元数据文件...）
    - 描述并直接反映实施工件的图文件。

  模型语义位于实施工件自身之中。例如，Java 实施的语义模型被序列化并存储为一组 Java 源代码文件。每个图位于项目中其自己的物理文件中。图文件可以具有各种扩展名。最常见的是 " .dnx "。实施建模图通常使用 UML 表示法，但是也可以使用其他表示法（例如用于数据可视化的 IDEF1X 或 " 信息工程 " 表示法，或用于设计 Web 层的 IBM 专用表示法）。

本白皮书的重点是关于如何组织 " 预实施 " 模型的内部结构。**在本白皮书的其余部分中，术语 " 建模文件" 特指 " 预实施"建模文件**（扩展名为 .emx 的文件）。有关如何对实施项目的内容进行组织的指导信息，可从其他来源（例如 Rational Software Architect、Rational Application Developer 和 Rational Web Developer Community Edition 的联机帮助）获取。

建模文件未必包含某个（逻辑）模型的所有信息。实际上，建模文件通常仅包含模型的一部分信息。在上面提供的示例中，我们有一个由三人组成的团队，该团队负责处理考勤卡应用程序的用例模型。该团队可以选择在物理上将其用例模型分割成三个建模文件，从而使每个团队成员能够处理用例中的不同部分，而不会争用同一个文件。本白皮书的最后一节讨论与分割模型和管理建模文件相关的问题。

模型类型

在 RUP 中，模型具有特定的类型，例如用例模型、分析模型或数据模型。在 RS*x* 中，建模文件不属于 " 强类型 "，但是您可以遵循关于使用多个 " 松类型 " 建模文件的约定。如果您希望遵循此约定，则可以按下列任一方法确定松类型：

- 先建立一个 " 空白 " 建模文件（见下），然后会根据您对该文件的命名方式以及放入其中的内容类型（包括对其应用的 UML 概要文件）确定其类型。

[1]在 Eclipse 中，资源是文件，但是还包括 Eclipse 环境中的其他属性和行为。此处描述的建模文件被 Elipse 视为 " 资源 "。

- 创建一个建模文件，它基于代表一种特定模型类型的预定义 “ 模板模型 ”。 RSx 产品为本白皮书中所述的模型类型提供了一组缺省的模板模型。您也可以自行创建模板模型（请参阅产品帮助、论坛以及 developerWorks 上的其他资源）。

无论哪种方法，RSx 中的建模文件 “ 类型 ” 实际上只与命名约定和文件内容相关。例如，该工具允许包含用例模型的文件同时包含用于实现用例的类（从逻辑角度来看，RUP 可视作分析或设计模型的一部分）。

这些指南确实建议您将 RSx 建模文件视作具有类型的文件。

工作空间、项目和项目类型

熟悉 Eclipse、WebSphere Studio 产品或 Rational Application Developer 的读者可能已经知道：文件位于项目中，项目可具有各种类型，而项目是在工作空间中进行分组和管理的。RS*x* 建模文件与其他文件一样位于项目中。

出于本次讨论的目的，并非 RSx 和 Rational Application Developer 中提供的所有项目类型都需要加以详细说明。我们主要对两类项目感兴趣：

- **UML 项目**
- **实施项目**，此类项目包括专门的项目类型，例如企业项目、EJB 项目、Web 项目以及 C++ 项目

我们先前说过 RS*x* 支持两种建模文件：

- 扩展名为 .emx 的文件，其中包含了用于以实施之上的抽象级别（需求、分析和设计）进行建模的 UML 模型
- Eclipse 项目，其中包含了实施语义（通常被序列化为源代码文件工件）和反映这些语义的图。

将模型分配给项目的规则很简单：

a) 将 “ 预实施 ” 建模文件放置在 UML 项目中

b) 实施模型照管其自身，这是因为实质上：

[实施模型] = [实施项目]

此规则有几个例外情况。以下 UML 建模文件是可放在特定于语言的实施项目中的候选对象：

- 设计 “ 草图模型 ”（将在后面一节中讨论）。
- 带有时序图的模型，用于描述将针对项目代码执行的测试

> ***XDE/Rose***
> Rose 和 XDE 操作理论包括以迭代方式优化设计模型的做法，其目标是达到与代码等同的抽象级别，然后使用 “ 代码 — 模型 ” 同步技术使该模型的语义与代码本身保持一致。例如：在 XDE 中，实施模型并不仅仅以代码和图的形式存在于项目中，还以 “ 代码模型 ” 文件的形式存在，这些文件独立于实施工件保存，它们实质上代表其语义的冗余副本。
>
> RSx 的操作理论鼓励使用抽象级别高于代码且与平台无关的模型（即诸如 “ 企业 IT 设计模型 ” 之类的设计模型）并使用转换从这些模型生成代码。然后，在代码抽象级别上，RSA、RSD 和 RAD 允许您绘制以 UML 表示法表达的代码语义图，而无需使用实施抽象级别的独立保存的语义模型。
>
> 请注意，RSx 允许您以代码抽象级别定义 UML 模型并从这些模型生成代码，实际上您应当采取这种用法。但是 RSx 不提供将此类模型与代码自动保持同步的技术。

概念回顾

下面的图总结了前面的讨论内容。这些图反映了先前描述的场景，在该场景中，我们有两个团队，一个处理呼叫中心应用程序，另一个处理考勤卡管理应用程序。

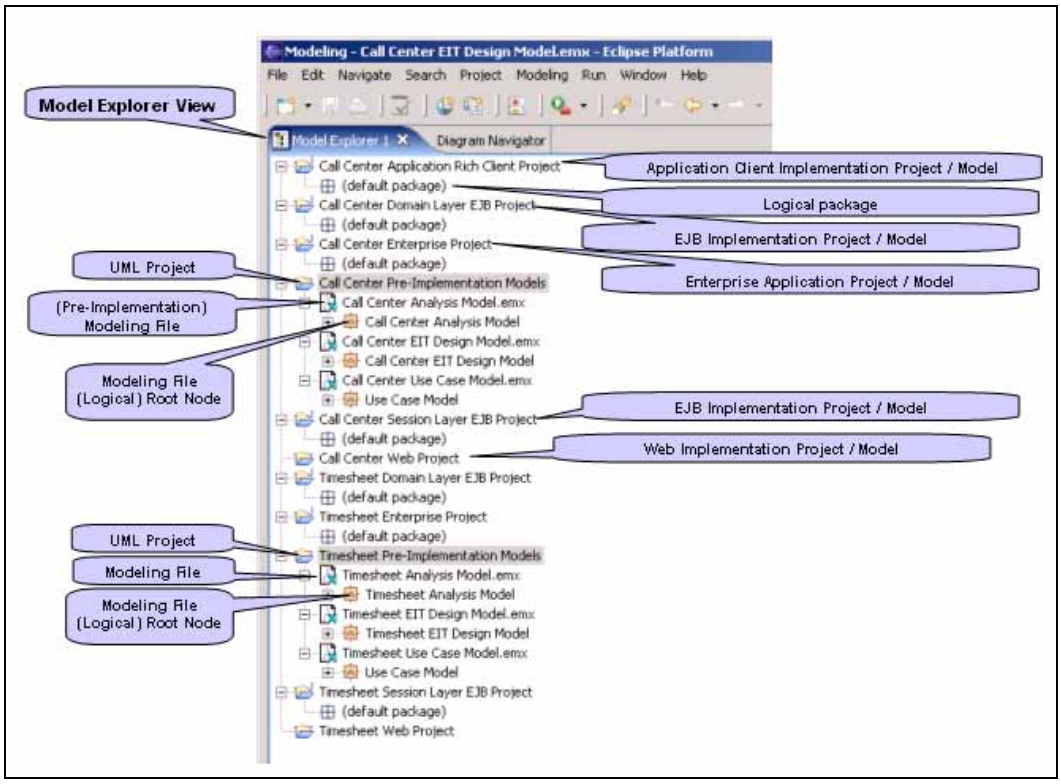


图 2-1

RSX 提供了一个浏览器视图[2]，该浏览器提供了模型的物理和逻辑综合视图。在该浏览器中，您可看到工作空间中的项目（称为顶级节点），而在每个项目中可看到属于该项目的资源。因此在**图 2-1** 中，我们看到“ 模型浏览器 ”中描述的一组项目，这些项目对应于我们场景中的两个应用程序。我们可看到 UML 项目已用于预实施模型，并且可看到一组解决方案对应类型的项目已用于这些实施模型。

> ***XDE/Rose***
> 与 RSA 模型浏览器不同，Rose 和 XDE 中的模型浏览器仅提供模型的逻辑视图。请注意，RSA 模型浏览器提供的资源视图不是 Eclipse 导航器视图提供的“ 纯 ”物理视图。虽然一些物理资源在模型浏览器中可视，但它们主要用表示资源的*逻辑*视图的图标进行表示。

在图 1-2 中，我们展示了“ 考勤卡 ”用例模型如何在内部组织为多个软件包，这些软件包代表问题域的某些功能内聚子集。

[2]在 V6.x 中，这称为“ 模型浏览器 ”。而在 V7.0 中，模型出现在通用浏览器中。本白皮书中的插图基于 V6.0，因此称为“ 模型浏览器 ”。

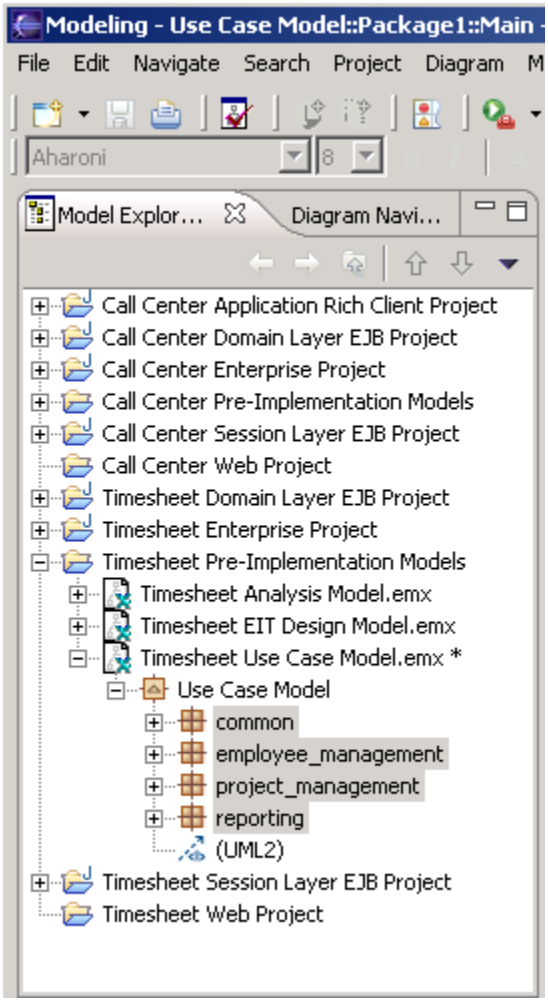


图 2-2

在**图 2-1 和 1-2** 中，每个预实施模型都位于一个单独的建模文件中。在图 1-3 中，我们展示了“ 考勤卡 ”用例模型如何重构为多个建模文件，这些建模文件对应于问题域的那些相同子集。请注意每个建模文件的根是如何命名的，以便在组成整个用例模型的所有建模文件中都保持一致的名称空间。

3. RUP 模型到 RS*x* 模型的映射

下面的表展示了最常用的 RUP 模型如何按照约定映射到 RS*x* 建模文件“ 类型 ”。该映射通常是直接的，但要将本白皮书用作通过 RSx 来实践 RUP 的指南，该映射却是关键所在。表中提及的 RSx 建模文件类型将紧接在该表之后加以讨论。后面的几节将提供关于各种建模文件类型的内部组织指南，并说明将它们存放在哪些种类的项目中。后面的这些讨论将按照此处列出的 RSx 建模文件类型出现。

RUP 模型	***RSx* 建模文件类型**
用例模型	基于“ 用例模型 ”模板的建模文件 （备用方法：先创建一个空白建模文件，然后按 RUP 用例模型指导信息来限制内容）
分析模型	分析模型 （备用方法：先创建空白建模文件，然后按 RUP 分析模型指导信息来限制内容） （备用方法：在设计模型中使用“ 分析 ”软件包）
设计模型	对于多层式业务应用程序：基于“ 企业 IT 设计模型 ”模板的建模文件 （备用方法：先创建空白建模文件，然后按 RUP 设计模型指导信息来限制内容） 对于其他类型的应用程序：创建空白建模文件，然后按 RUP 设计模型指导信息来限制内容 对于设计“ 草图 ”：空白建模文件 可选的补充：用作“ 实施概要模型 ”的其他空白建模文件
实施模型	包含实施工件和图文件的 Eclipse 项目
部署模型	创建空白建模文件，然后按 RUP 部署模型指导信息来限制内容

RS*x* 建模文件类型

空白建模文件

RSx 提供了创建“ 空白模型 ”的选项（文件→新建→UML 模型→空白模型）。空白模型是一种不基于模型模板的建模文件。它没有应用特殊的概要文件，并且除了一个单独的“ 主 ”（自由格式）图之外，没有缺省内容。**您可以将空白建模文件用作任何类型的模型的起点。**通过选择对它的命名方式、您在其中定义的内容以及对其应用的概要文件，就可以用空白建模文件来构建用例模型、分析模型、设计模型、部署模型或任何其他类型的 RUP 模型。

用例建模文件

RSx 提供了创建基于模型模板的“ 用例模型 ”文件的选项。该模板提供了***图 3-1*** 中所述的缺省内容。（关于如何使用“ 构建块 ”内容和搜索字符串的说明不在本文档的讨论范围之内。模板包含指示信息，您会发现它们大多简单明了。）

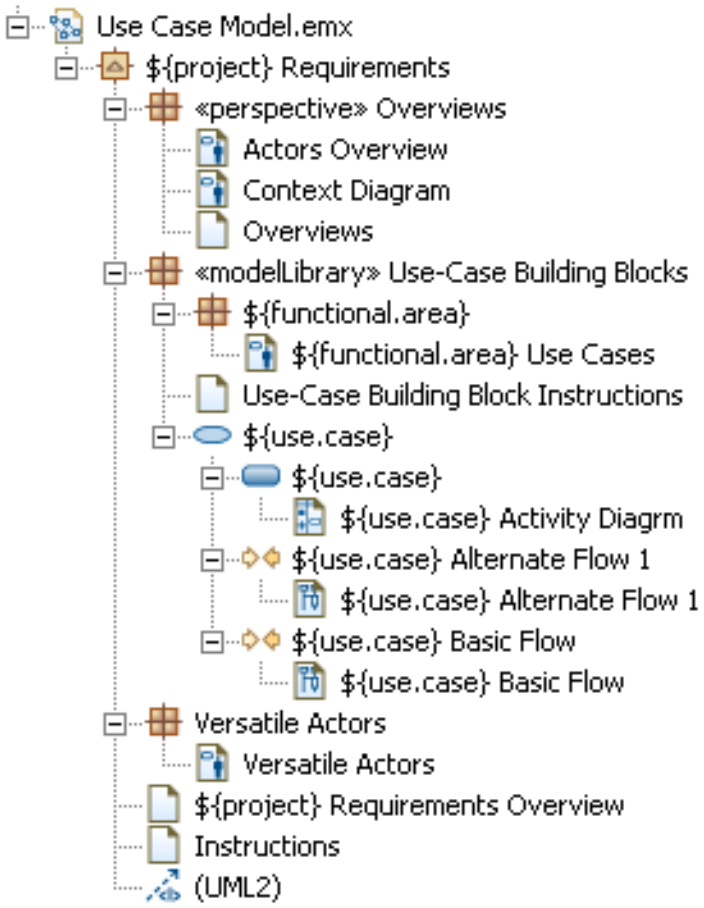


图 3-1

分析建模文件

RSx 提供了创建基于模型模板的“ 分析模型 ”文件的选项。该模板提供了***图 3-2*** 中所述的缺省内容。此外，还对从该模板创建的模型文件应用了一个“ 分析 ”概要文件：

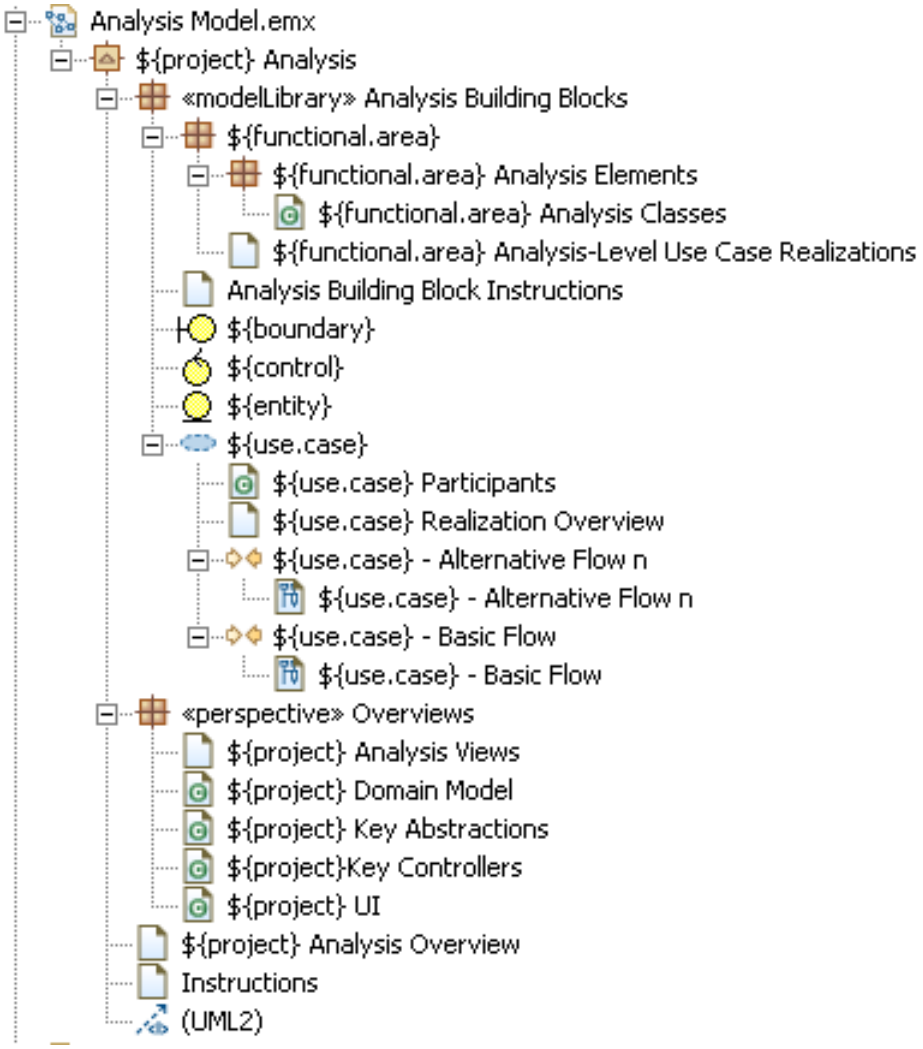


图 3-2

企业 IT 设计建模文件

RSx 提供了创建基于模型模板的“ 企业 IT 设计模型 ”（EITDM）文件的选项。该模板提供了**图 3-3** 中所述的缺省内容。此外，“ EJB 转换 ” 概要文件[3] 将应用于从此模板创建的模型文件。在针对业务应用程序和使用 RSx 代码生成转换以支持此类应用程序的创建时，该模板是适用于设计（也可选择用于分析）的模板。

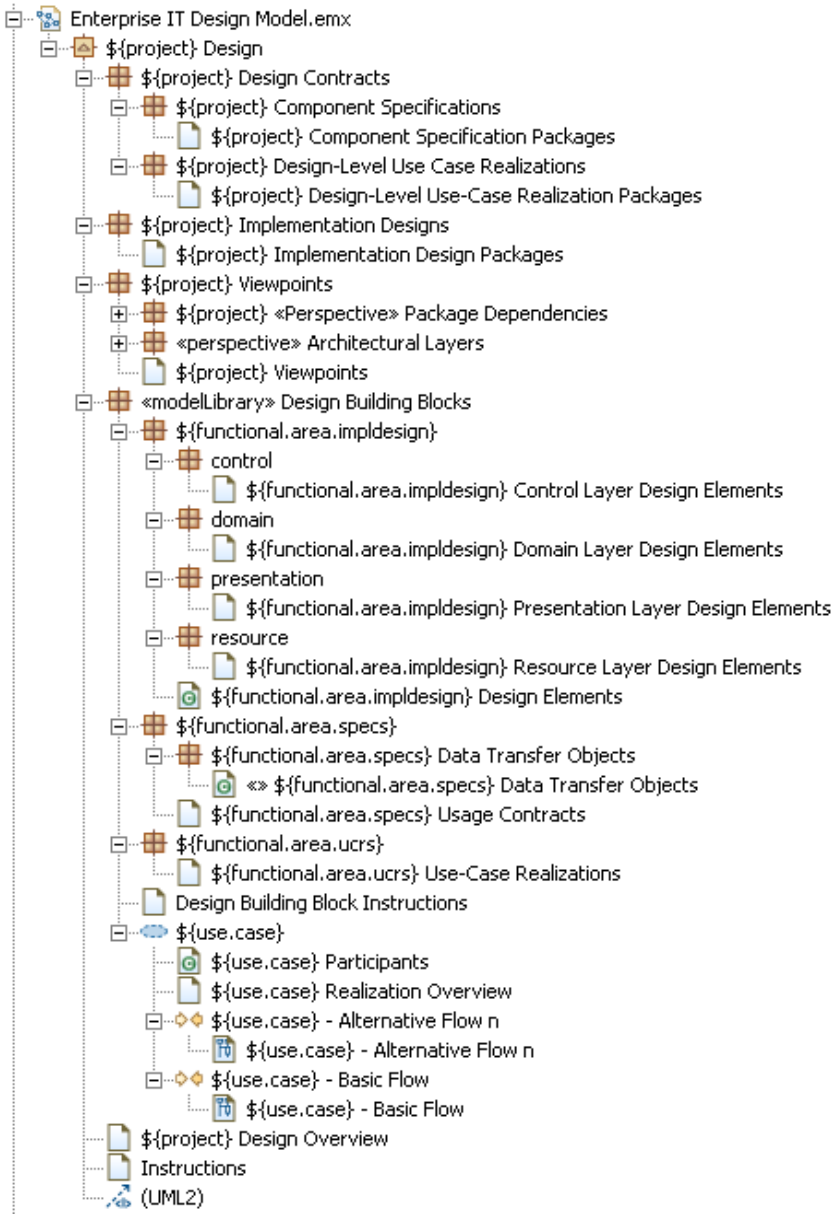


图 3-3

[3] 这组支持转换的概要文件作为“ EIT 设计模型 ” 模板的一部分提供，可能会在发行产品更新时演变。

实施概要建模文件

作为设计模型的一部分，您还会发现定义“ 实施概要模型 ”以捕获实施组织方式的高级别视图是很有用的。“ 实施概要模型 ”将在设计阶段的早期使用（在代码的生成或编写完成前），以表示实际的 RSx 或 Rational Application Developer 项目和文件夹／软件包，您期望的代码和相关文件（元数据、部署描述符等）在这些项目和文件夹／软件包中。您也可以使用此模型显示这些项目和软件包之间预期的依赖关系，这在确定系统构建需求时可能很有用。“ 实施概要模型 ”也可以用来保存解决方案体系结构的非正式概念图。

实施模型

如前所述，在 RSx 中，实施模型由一个项目构成，该项目包含实施工件和（可选）描述这些工件的图[4] 。

“草图”模型

如“ 基本概念和术语 ”一节中所述，您可以将设计模型当作在系统生存期进行维护并用于支持／加强体系结构控制的正式体系结构图。或者，您可以将它们当作草图，这些草图用于提议设计，并帮助澄清和传达该设计，但一旦实施开始就可以废弃。 RSx 对这两种方法都支持。其功能一般不针对两种方法中的某一种，但是您关于如何使用设计模型的选择的确有助于确定您将使用哪些 RS*x* 功能以及如何使用这些功能。当此区别在本白皮书中提出的指南环境中变得很重要时，将使用“ 草图模型 ”这个术语来表示正以可废弃程度更高的方式使用模型。

[4] 要创建这些图，您应当使用“ 文件→新建→类图 ”来创建一个图，在该图中可通过 UML（或其他）表示法构建代码“ 视图 ”，而不是使用“ 文件→新建→UML 模型 ”来创建模型。 每个图都单独保存为一个扩展名为 .dnx 的文件，并且会受到与代码文件相同的版本控制。这些图不包含任何语义信息，而只是表示法。所有相关的语义信息都位于代码本身之中。 当您更改其中某个图的内容（例如类名或操作签名）时，实际上是更改了底层代码本身。当您（使用文本编辑器）在代码中进行了此类更改时，更改后的代码所在的图会自动更新。

4. 组织模型内部结构的一般指南和方法

组织 UML 模型内容的基本工具是软件包。 UML 软件包有两个主要用途：

- 将模型信息分区、组织并标注
    - 将与问题域或解决方案域中的某个特定主题相对应的元素分组
    - 分离不同类型的模型信息（例如接口、实施、图等）
    - 将元素分组来定义并控制它们对其他元素的依赖关系
    - 对在同一模型上提供多个备用视图的图进行分组
- 建立名称空间
    - 用于模型元素
    - 用于从模型元素生成的实施工件（这可能涉及模型和实施语言名称空间之间的映射）
    - 用于一组复用

通常，RUP 已经为各种模型类型提出了具体的封装策略。这些策略反映在本白皮书的特定于模型类型的部分。 RSx 还引入了一些其他的组织工具，如下所述：

使用“ 透视图”软件包表示视点

如果希望看到以多种方法组织的元素，您可以使用描述备选组织方案的图来创建更多软件包。这一方法可以在任何需要表示（关于已超越了模型封装方案的模型内容的）特定视图的地方使用。 RSx 通过提供“ 透视图 ”软件包构造型作为其 UML “ 基本概要文件 ”的一部分，来支持此方法。您可以认为“ 透视图 ”软件包通常相当于用于系统工程或 IEEE 1471 – 2000 “ 视点 ”的 RUP。

不要将语义元素（类、软件包、关联等）放到“ 透视图 ”软件包中。请只将特定的图放置于其中，这些图根据备选的组织关注点或应用程序视点来描述视图。将“ 透视图 ”构造型应用于软件包有以下几个作用。它直观地将该软件包识别为表示某个特定视点。它还支持模型验证规则，当语义元素置于“ 透视图 ”软件包时，该规则将向您发出警告。它还用作 RSx 转换应当绕开的软件包的标识符。

使用主题图创建特定关注点的自更新描述

在“ 一般 ”图中，您可以手动放置要描述的元素。与其相反，主题图的内容由针对现有模型内容运行的查询确定。要创建主题图，请选择“ 主题 ”模型元素，然后根据其他元素与主题元素的关系类型，定义您希望在图中出现的其他元素。随着模型的语义内容的更改，主题图也相应地调整。

通过浏览图检验模型

浏览图不是 *专门*用于模型组织的工具。其目的是使模型内容的发现和了解变得更容易，而不必手动构建图。但是在模型组织的环境中，对它们有所认识是有益处的，因为它们可以减少您构建持久图的需要。接着就可以减少模型的大小、降低模型的复杂性，使它们更易于组织。

浏览图有点像主题图，但有一个最大的区别，就是浏览图永远都不是持久的，它们总是即时生成的。要生成浏览图，请选择模型元素（从图中或模型浏览器中），然后使用上下文菜单来“ 探索浏览图 ”。这样生成的图将选中的元素描述为“ 中心点 ”，其中相关元素围绕中心点呈辐射状布局。当然，然后您就可以选择该浏览图中的一个相关元素，使其成为另一个浏览图的中心点，而且只要您愿意可以继续这么做。

图间浏览

在 RSx 中有两种用于图间浏览的机制：

- 可以将图节点从模型浏览器拖动到另一个“ 主机 ”图中。然后您可以双击主机图上生成的图标来打开引用的图

- 无论何时在模型中创建新的 UML 软件包，总会自动创建“ 主 ”图（自由格式图）。缺省情况下，该“ 主 ”图创建为软件包的“ 缺省 ”图。您可以将该图重命名为“ 主 ”之外的名称，它还是会被当作“ 缺省 ”图。您也可以选择软件包中的另一个图，并使它成为该软件包的“ 缺省 ”图。“ 缺省 ”图的作用是：如果您将软件包本身放入某个其他“ 主机 ”图中，就可以双击该软件包来打开其缺省图。

这些机制支持以下组织**指南**，这些**指南**可以应用于任何类型的模型：

1. 构建每个建模文件的“ 主 ”图（或其他缺省图）来描述
    a. 建模文件中的每个顶级软件包
    b. 位于建模文件的根软件包中的任何其他图的图标（换言之，不要描述缺省图本身的图标）
2. 构建每个顶级软件包的“ 主 ”图（或其他缺省图）来描述
    a. 它直接包含的软件包
    b. 它直接包含的任何其他图的图标
3. 对每个后续低级别的软件包重复该模式

5. 用例模型的内部组织指南

用例模型高级别组织

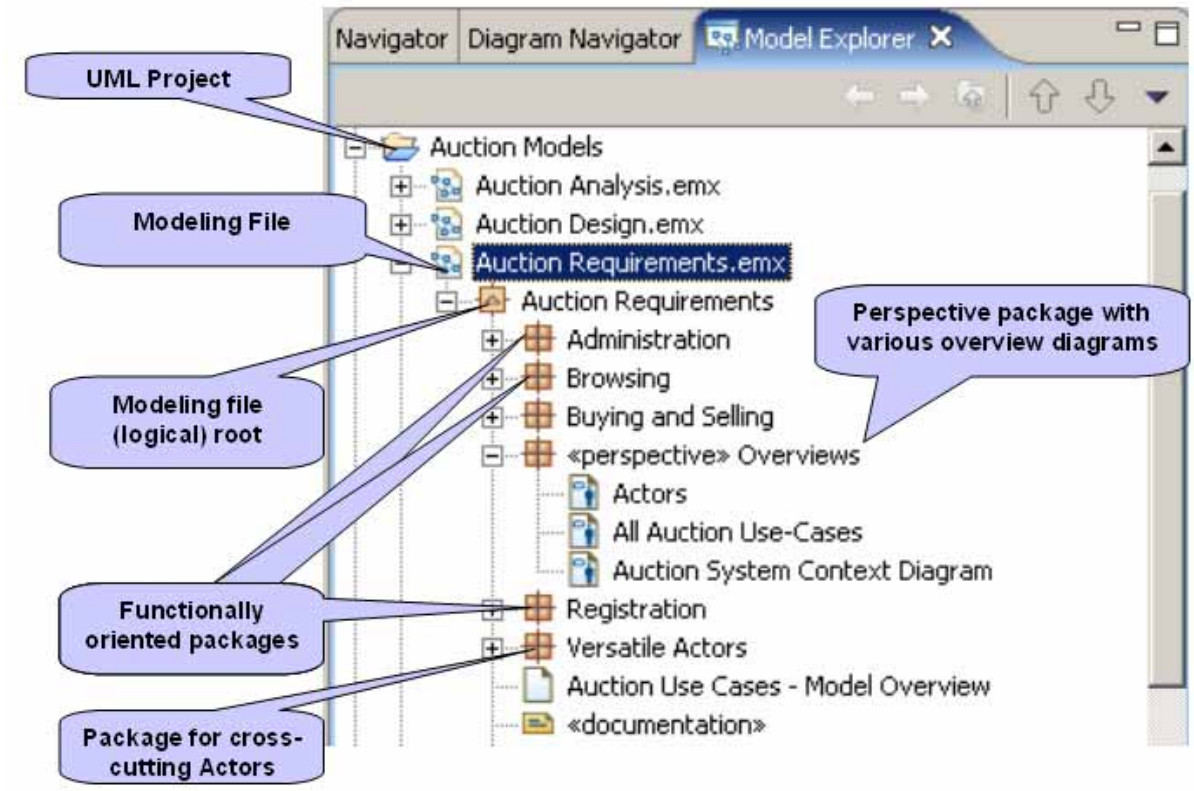


图 5-1

图 5-1（见上）说明了用于构建用例模型的以下指南：

1. 使用顶级软件包来确立以功能为导向的分组。理由：

   - 在一个团队的人员处理用例模型的情况下，这通常能够针对关注的“ 分工 ”问题建立良好的映射关系。而且，如果后来由于文件争用成为了问题，您决定将用例模型分成多个建模文件，则它可以帮助您很好地做到这一点（您只要为每个顶级软件包创建一个独立的建模文件即可）。

   - 与其他组织方法相比，这通常可针对最终实施的组织建立更好的映射关系。如果您将使用转换来为每个后续低抽象级别提供种子值，则这很重要。尤其是如果您要根据用例模型在分析模型中生成种子值内容，您就会希望用例模型的封装结构能够很好地映射到目标分析模型的期望封装结构上。接着，您又希望分析模型的封装结构很好地映射到设计模型，设计模型的封装结构很好地映射到将组成实施的那组项目上。这些映射越简单，配置从一个抽象级别到下一个级别的转换所需的工作就越少。

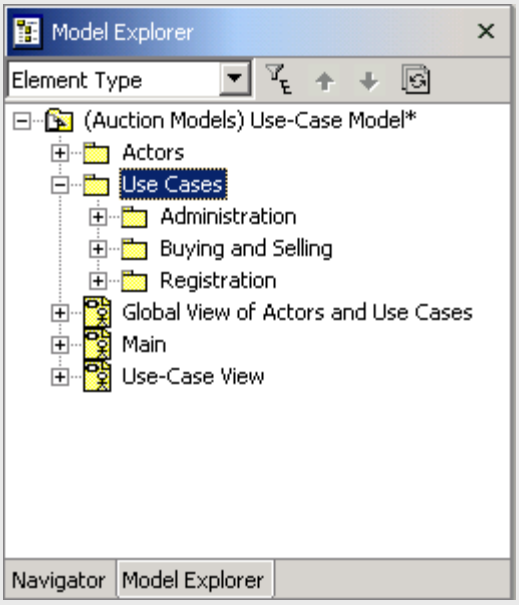
2. 使用另一个顶级软件包来捕获“ 充分授权 ”或“ 万能 ”的参与者。

3. 使用“ 透视图 ”软件包中的图来捕获用例的高级别交叉视图。理由：

   - 提供交叉视图和“ 体系结构上有重要意义 ”的用例的视图，同时保持模型的语义元素组织为以功能为导向的分组。

XDE/Rose

RSX 指导信息在一定程度上修订了针对用例模型高级别组织的传统指导信息：为参与者创建一个软件包，为用例创建另一个软件包。然后，按照要求的模型规模和复杂性，您应使用低级别软件包来确立以功能为导向的分组，如以下基于 XDE 的示例中所示：



用例模型内容

本文档并非有关如何编写好的用例或好的用例建模应该如何及不该如何的详细教程，这些内容已超出了本文档的讨论范围。但是，我们在下面对用例模型中除了参与者和用例之外还可以包含的内容进行了简要讨论。

- **建议：**在模型根位置创建“ 主 ”图，它描述模型的其他软件包，并支持下寻到那些软件包及其各自的“ 主 ”图

- **建议：**在每个用例软件包中，包含一个图来描述该软件包用例、它们之间的任何关系以及它们的参与者。（如果用例的数量很大，使用多个图可能比较合适。）

- **建议：**在每个用例的“ 文档 ”字段描述每个用例的主流程和备用流程[5] （请参阅**图 5-2** ）

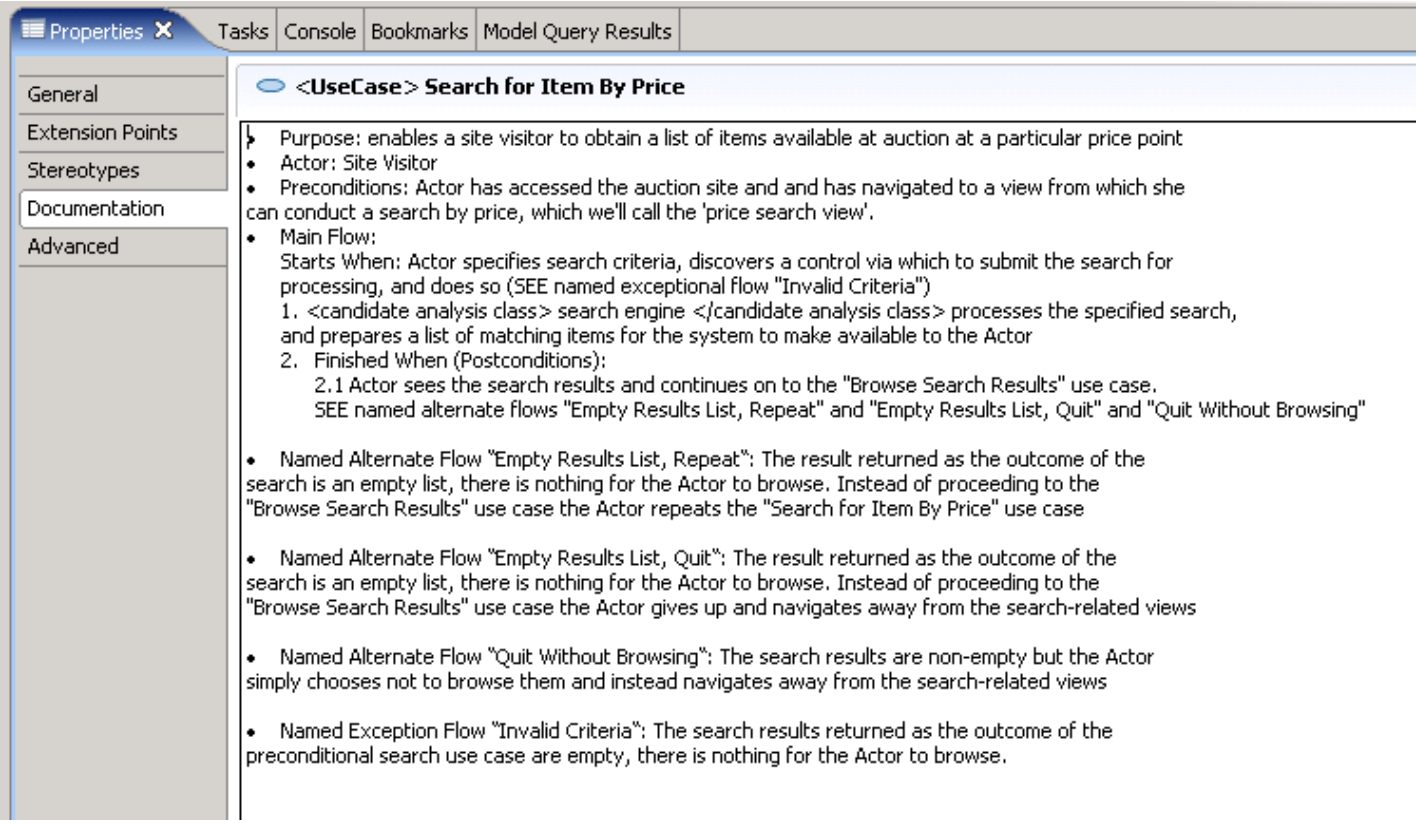


图 5-2

- **可选：**请添加“ 活动 ”图并对该图进行构建，使其反映用例的整体活动流程，这适用于当用例的复杂度使得有必要这样做的情形。（请参阅**图 5-3** ）**理由：**这有助于显示与每个流程（主流程和备用／例外流程）对应的条件，并有助于确保所有不同的流程最终重新合并到一起。（在 RS*x* 中添加活动图将使活动自动添加到用例中，并且图位于活动下）。

- **可选：**对用例的每个已命名流程（主流程、备用流程和例外流程）的“ 黑匣 ”实现进行建模：向用例添加一个协作发生；向用例添加一个对应于其主流程的交互实例，以及为每个已命名备用流程和例外流程添加一个交互实例；为每个交互实例构建一个时序图（或者可以是通信图）。这些用例协作实例不应与“ 分析模型 ”中所述的分析级别的用例实现相混淆，或者与“ 设计模型 ”中所述的设计级别的用例实现相混淆。那些是用例的“ 白匣 ”实现，它们描述解决方案的内部元素之间的交互。这里为用例模型提议的协作发生在严格意义上是参与者和系统之间的“ 黑匣 ”交互中。（请参阅**图 5-3** ）**理由：**这向非技术性的项目干系人提供了关于系统用户将如何与系统进行交互的高级别概况。它还可以帮助您确定作为实施的一部分而必需的各种视图（屏幕或页面）。它还通过将名称分配给语义模型元素（即分配给协作发生），正式确定用例的各种流程（场景）的命名。

[5]此用例描述示例中所示的格式编排是通过使用支持 RTF 的编辑器为用例描述创建文本“ 模板 ”，然后将该模板复制并粘贴到用例描述字段中来完成的。

XDE/Rose
在 UML 1.x 中，您应当已将“ 协作实例 ”（而非“ 协作发生 ”）用于本用途。

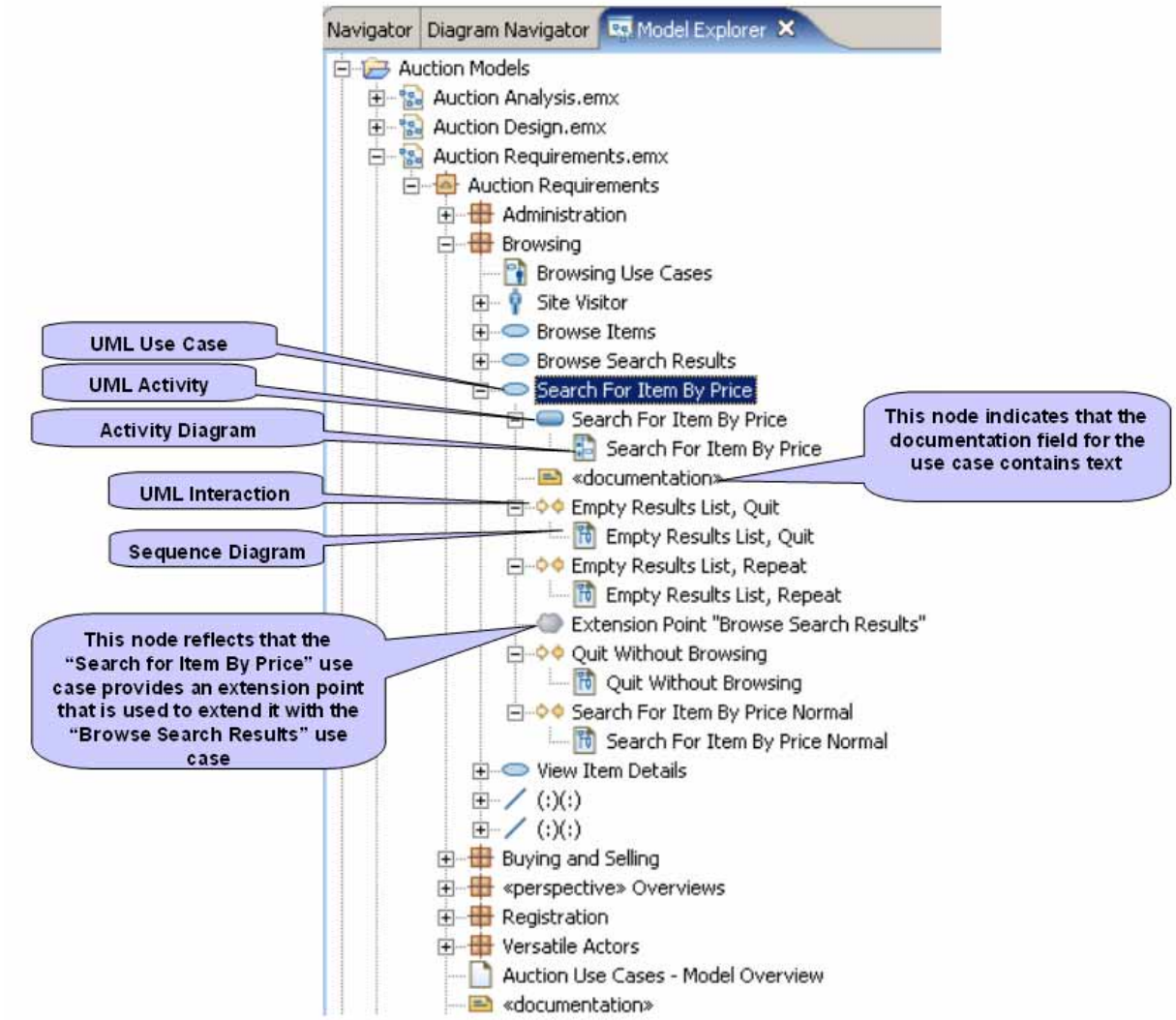


图 5-3

- **可选：**如果您遵循 RUP 指导信息来确定体系结构的“ 在体系结构上有重要意义的 ”视图，特别是如果您要保留“ 软件体系结构文档 ”，则请添加顶级“ 透视图 ”软件包以包含一些用例图，这些用例图描述在体系结构上有重要意义的用例。您可能需要将软件包命名为“ 体系结构的用例视图 ”。

6. 分析模型的内部组织指南

分析模型表示解决方案中的“ 第一步 ”。它是从需求进行到最终设计的铺路石，将重点放在捕获业务领域的相关信息，并显示已接近业务的高抽象级别的候选解决方案元素。它是分析类和分析级别的用例实现所在之处。您应当从对用例实现进行建模（主要使用时序图）的过程入手，来*发现*解决方案需要哪些类 – 尤其是这些将是与您在时序图中发现的所需生命线相对应的类。也可以应用一些凭经验得出的方法来根据用例模型的内容提议分析模型内容。这些内容将在本节的后面部分谈到。

在 RUP 中，分析模型是否应独立于设计模型进行维护，是一个特定于项目的决策，是您根据自己是否相信维护独立分析模型的价值与投入的时间相称而作出的决策。如果创建了独立的分析模型，但是不保留，则将把分析类移入设计模型并进行优化。或者分析模型可能逐渐地演变成设计模型[6]。从特定于产品的角度讲，以下是一些您可能要考虑的选项：

1. 创建基于“ 分析模型 ”模板的分析模型，该分析模型位于一个或一组建模文件中。接着，基于“ 企业 IT 设计模型 ”模板，使用手工过程或自动转换在另一个模型文件（或一组模型文件）中创建优化版本的分析元素，然后废弃分析建模文件。这将让您选择是继续保留独立的分析模型还是将其废弃。
2. 根据您应用了分析概要文件的“ 企业 IT 设计模型 ”模板，在一个或一组建模文件中进行分析级别的建模。这样您就可以使用分析类开始对用例实现进行建模，然后随着时间的推移对它们进行优化，从而使设计接口能够扮演行为中的角色。
3. 第二个和第三个选项的结合是将某类分析模型保留在与设计模型相同的建模文件中。要做到这一点，您应当将分析内容分离到应用了关键字“ 分析 ”的软件包中。这提供了可能，以将分析级别的工件与其更优化的设计级别的工件保留在同一个建模文件中。

在使用 RSx 转换生成实施时要注意的事项是：那些转换在许多情况下可以接受分析级别的元素作为输入，从而为您省去了一些手工将那些元素优化为设计元素的步骤。以此方式使用 RSx 时，首选上述的第二个或第三个选项。封装为 RSx 一部分的标准代码生成转换将绕过有“ 分析 ”关键字的模型软件包。

[6]实际上，RUP 会指示下列选项：在设计模型中创建分析类和分析级别的用例实现，然后将它们直接演变为其设计格式。用这种方法，当设计模型“ 被发现 ”时，您可以用保留一些“ 纯分析 ”透视图的方法来创建软件包。

分析模型高级别组织

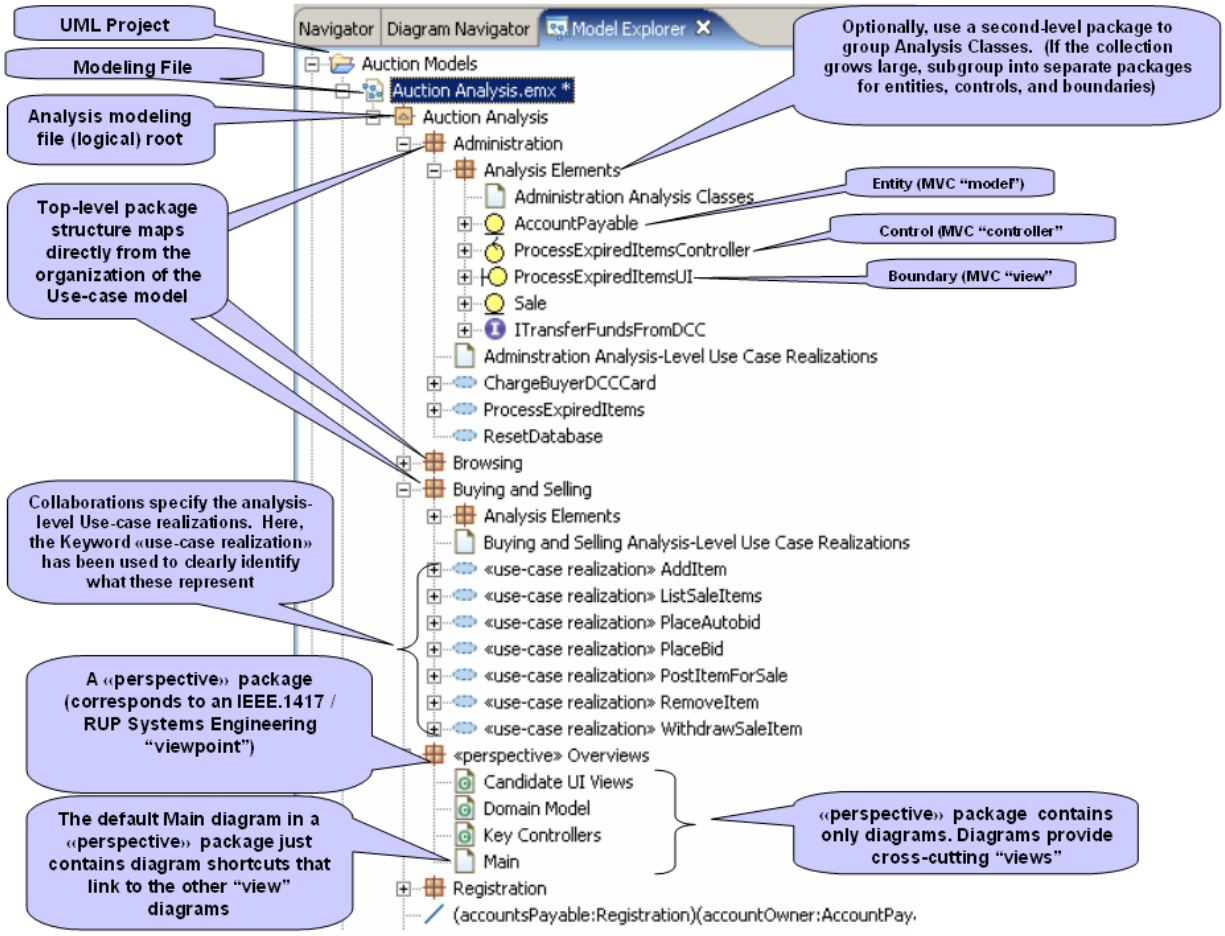


图 6-1

图 6-1 （见上）说明了用于构建分析模型的以下指南：

1. 使用顶级软件包来为分析类确定以功能为导向的分组。理由：与用例模型的理由相同。

2. （可选）在顶级软件包内使用子软件包来收集并组织分析类。

3. 使用“ 透视图 ”软件包中的图来捕获分析元素的备用、高级别或交叉视图。理由：为不同的项目干系人提供不同的透视图，同时保持模型的语义元素组织成以功能为导向的分组。

在下面的**图 6-2** 中描述了在此方法的基础上稍加变化而得到的方法，它显示了使用顶级软件包将用例实现从分析类中分离出来。在该顶级软件包中有一组以功能为导向的子软件包，它们与那组顶级软件包相匹配。用这种方法分离用例实现使您能够重构包含分析类的软件包结构，而*未必*会影响用例实现的组织。（特别是如果分析模型将*在原处*演变为设计模型，则类的软件包组织可能会演变，这样它就不再与原来用于用例的软件包相匹配。）

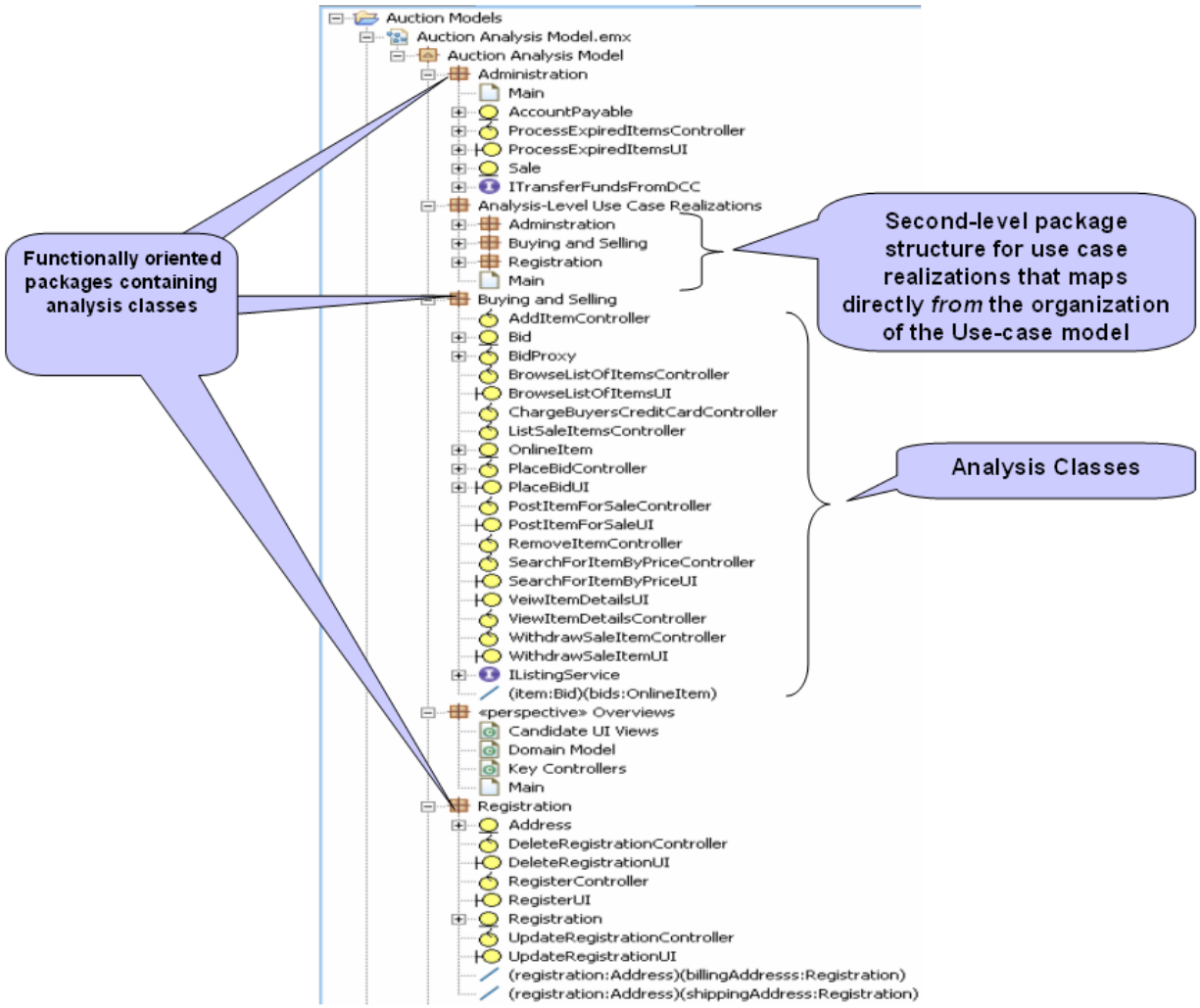


图 6-2

根据您的情形，可能有理由使用一个命名约定，该约定预计了由多个独立组创建的模型内容的合并和复用，甚至包括不同（合作伙伴）业务中的组。如果这是一个问题，请考虑使用反向因特网域名称空间约定，如下面的**图 6-3** 中所述。请注意，这对分析建模*本身*可能不是一个大问题，但是如果您采用的是让分析模型*在原处*演变为设计模型的方法，并且期望以设计级别进行复用或业务集成，则您可能需要提前进行计划。采用这种方法的另一个潜在的优势是：因为它可以针对从分析／设计中生成的代码的组织来建立良好的映射关系，因此它可以简化代码生成转换的后续配置。

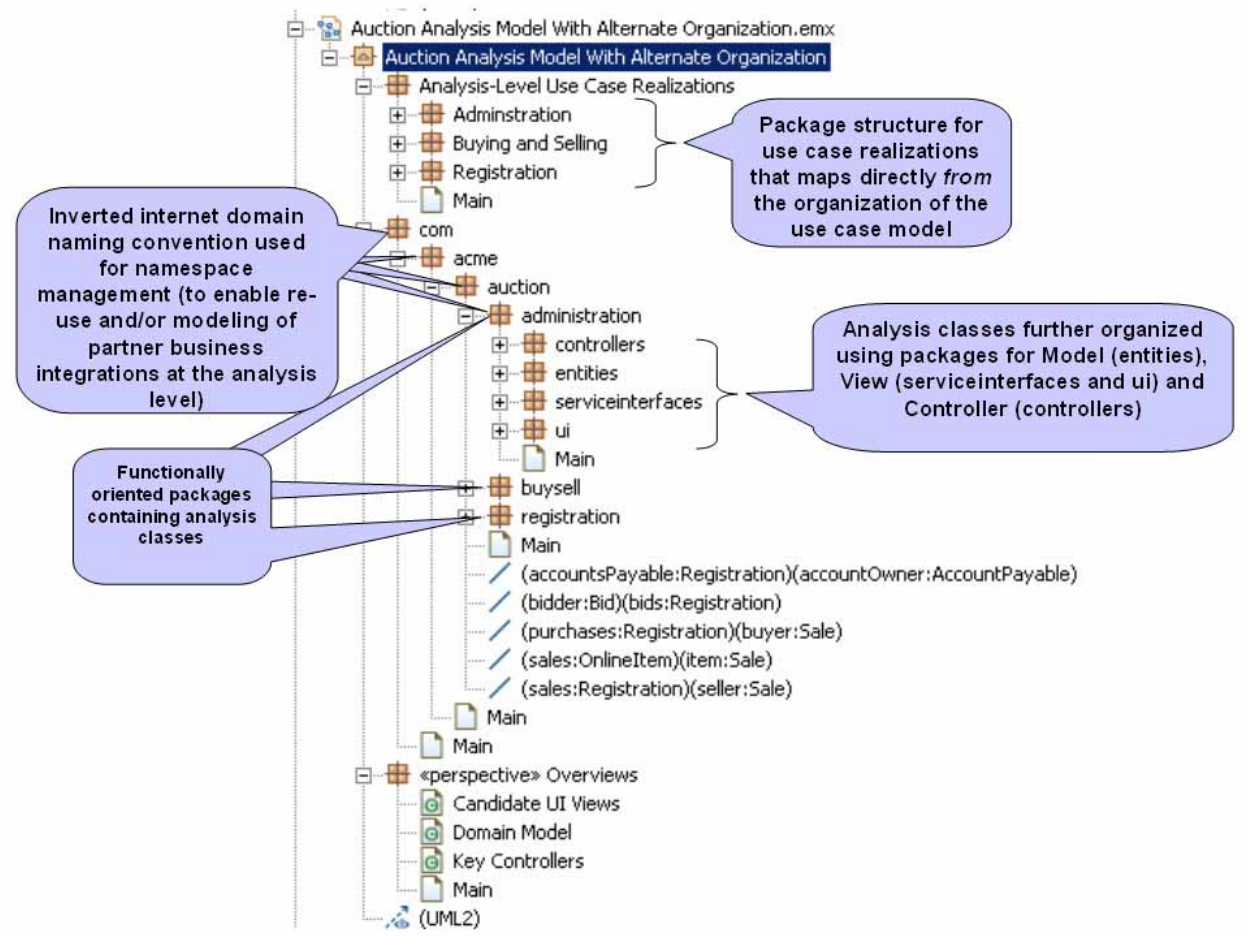


图 6-3

分析模型内容

发现分析类是什么有多种方法。一种方法是开始绘制用于提议用例实现的时序图。使用这种方法时，您会发现需要什么生命线，而且一般每条生命线都与一个可能的分析类相对应。当您使用这种方法发现类时，可以在分析模型的用例实现软件包中创建它们，但是不应该将它们留在那里。您应当“重构”模型，以将分析类移动到以功能为导向的软件包，如早先在分析模型的高级别组织指南中所述（请参阅***图 6-1***）

用于发现分析类的另一个有用方法是：根据以下凭经验得出的规则，用类来为分析模型提供种子值：

- 对于（用例模型中的）每个用例，向分析模型添加一个“控制”类。“控制”类代表与用例相关联的业务逻辑。（稍后在设计中，它们还将映射到诸如会话管理之类的对象。）
- 对于（用例模型中的）每个参与者／用例关系，向分析模型添加一个“边界”类。“边界”类代表解决方案和人类参与者之间或解决方案和某个外部系统之间的接口。对应于人类参与者“边界”类可能最终会映射到设计和实施中的一个或多个用户接口工件。对应于外部系统的“边界”类可能最终会映射到设计和实施中的某类适配器层。
- 通过诸如 CRC 卡分析或用例描述的字分析等过程，识别其他“控制”类（动词）和“实体”类（名词）。

当您使用该种子值方法来识别分析类时，可以将类直接放置到以功能为导向的软件包中，如早先在分析模型的高级别组织指南中所述（请参阅***图 6-1***）

无论您如何来发现分析类，都可以几乎肯定地认识到：需要对原始功能软件包的组织作出更改。

可选：使用分析类软件包中的二级软件包进一步组织那些软件包的内容（请参阅**图 6-4** ）

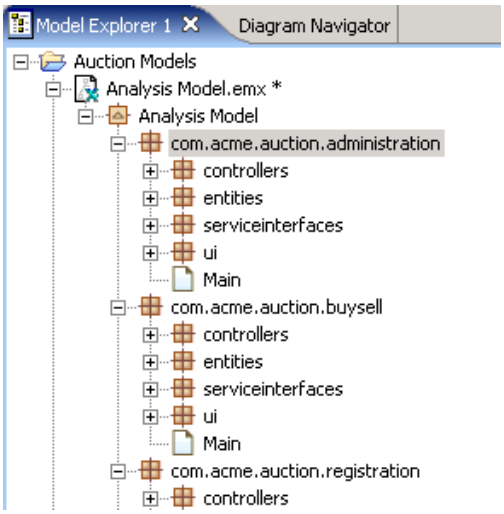


图 6-4

建议：分析模型应当包含分析级别的用例实现，这些用例实现描述了如何根据分析类执行用例。每个分析用例实现（由 UML 协作表示）实现用例模型中的一个用例，并具有与该用例相同的名称。请参阅**图 6-5** 。对于您认为应当作为分析级别实现建模的每个已命名用例流程[7] ，请添加时序图（该时序图将自动添加拥有的交互）。**图 6-6** 显示了在您创建时序图时将添加到模型中的语义内容类型。（请注意，您可以过滤“ 模型浏览器 ”视图中的任何 UML 元素类型，从而隐藏**图 6-6** 中所述的许多“ 杂讯 ”）

[7]先前已在“ 用例模型“ 中建立

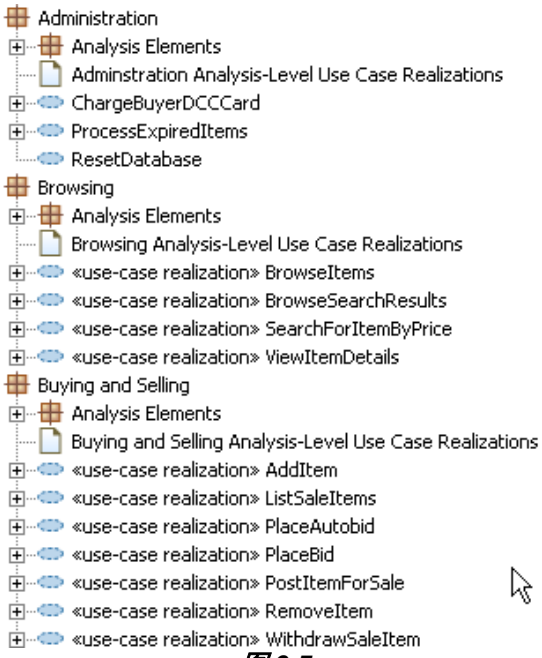


图 6-5

可选：一旦为用例流程创建了时序图，您就可以在“ 模型浏览器 ”中选择其拥有的 UML 交互，并向其添加通信图。新的通信图将自动用参与时序图的分析类实例进行填充。

建议：从每个用例实现（UML 协作）创建实现依赖关系，并从用例模型创建相应的用例（请参阅**图 6-6** ）。因为您可以使用诸如“ 主题图 ”和“ 可跟踪性分析 ”等功能来了解模型中的可跟踪性关系，所以您实际上无需保留永久图来描述可跟踪性关系，因此建议您使用某种“ 废弃 ”图来创建关系，例如：

- 向“ 协作 ”添加自由格式的图
- 将“ 协作 ”拖动到它上面
- 将用例拖动到它上面
- 绘制依赖关系
- 最后，（在模型浏览器中）将图从“ 协作 ”中删除

建议：为每个用例实现包含一个“ 参与者 ”图，以显示参与该实现的分析类（即：实例出现在描述用例实现的交互图中的分析类），以及支持交互图中所述协作的那些类之间的关系。请参阅**图 6-6**

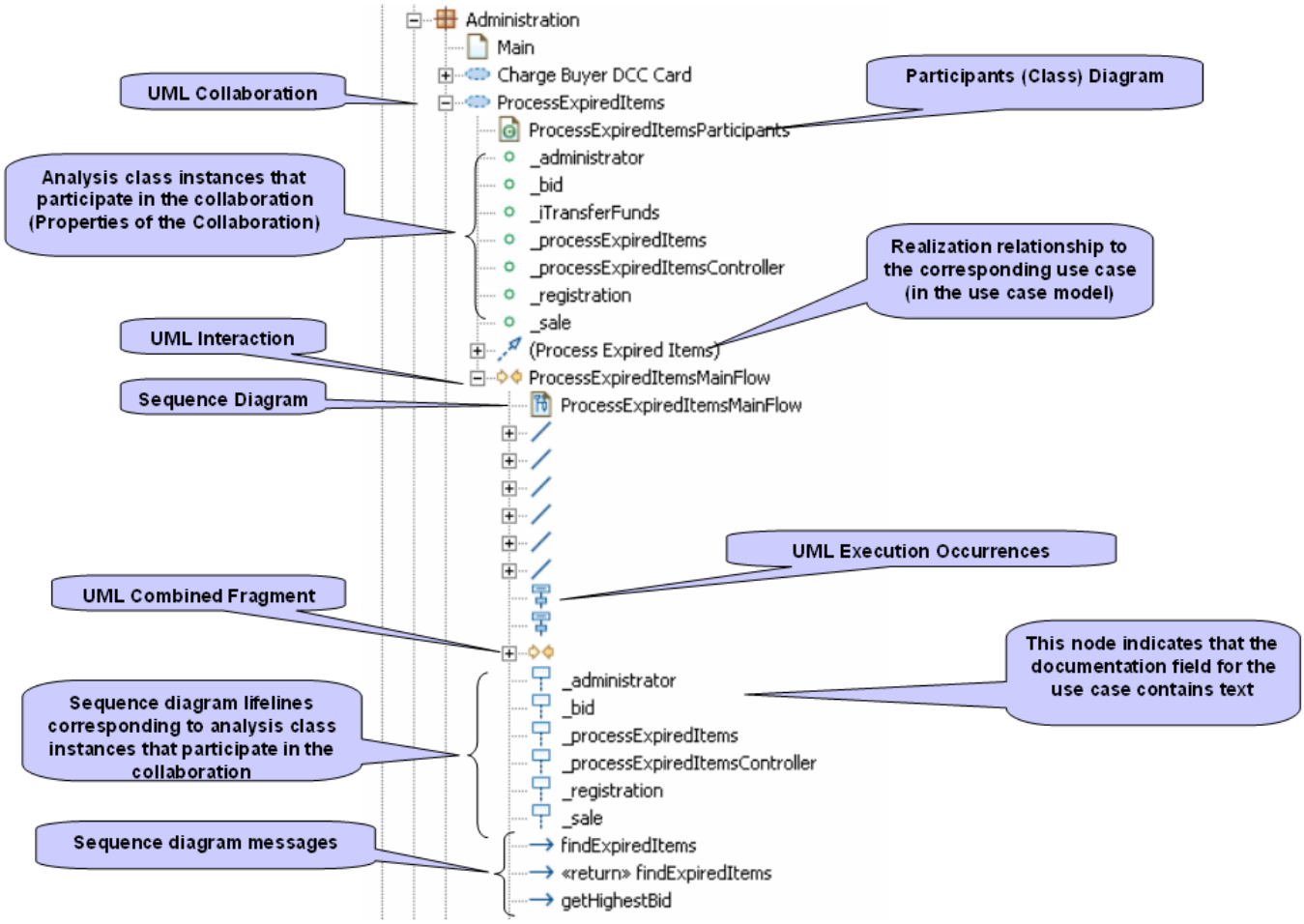
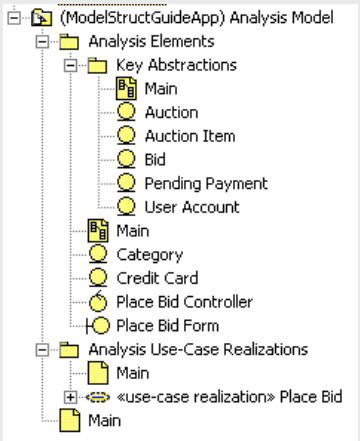


图 6-6

XDE/Rose

下面所示的一般情况下建议的**分析模型**结构针对 RSx 进行了修改，将重点放在了分析类的以功能为导向的软件包组织上。同时请注意，在“ 透视图 ”软件包中，已不再使用“ 关键抽象 ”软件包（将有损以功能为导向的封装方法），改为使用“ 关键抽象 ”图。



7. 设计模型的内部组织指南

设计模型高级别组织

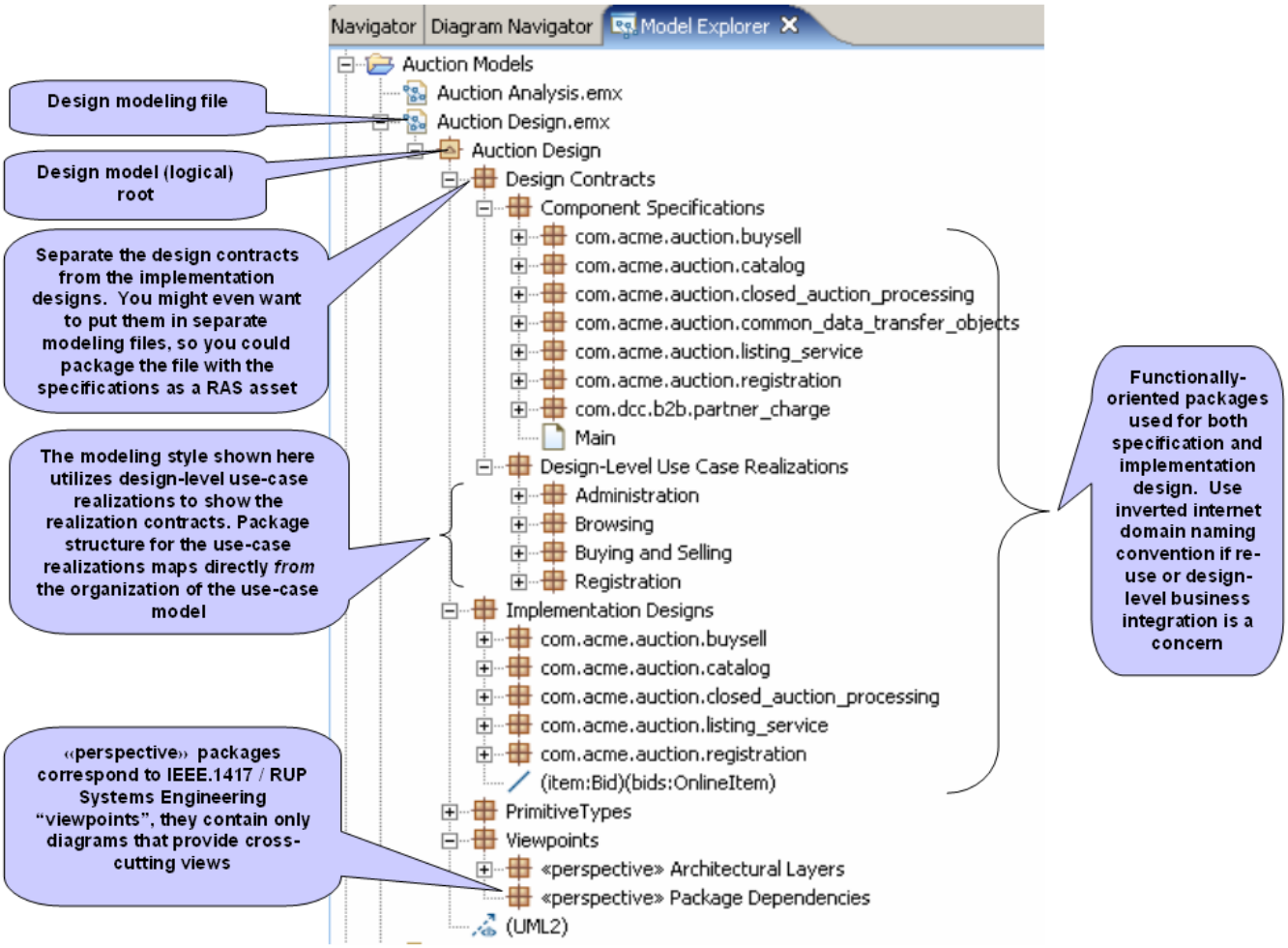


图 7-1

图 7-1（见上）说明了用于构建设计模型的以下指南：

1. 将规范从实施设计中分离出来。该图显示了使用顶级 " 设计合同 " 和 " 实施设计 " 软件包来实现这一点。

2. 使用低级别软件包来确立以功能为导向的分组。例如，您可以从分析期间所使用的组织入手，让它随着您作出关于分析类如何映射到实际设计类、组件和服务的决策进行演变。（任何初始的组织方案都可能在设计期间演变 — 请参阅下面的深入讨论）。

   现在可能该对子系统说上两句。在 UML 2 以前的版本中，子系统是一种专门的软件包类型。在 UML 2 中，子系统是一种专门的组件类型，组件中可以包含软件包。因此，在 UML 2 中，" 子系统 " 组件是

软件包的可行的组织／名称空间替代方案，但 UML 2 关于何时该使用子系统、何时该使用软件包却很模糊。建议：使用各个详细程度级别的软件包（例如：特定应用程序的设计子系统），并保留子系统来代表体系结构的企业范围视图中的所有应用程序（例如 CRM 或 SCM）。

> ***XDE/Rose***
> *进行本次编写工作时，我们期望 Rose 和 XDE 模型导入工具会提供选项，来将 UML 1.x 子系统映射到 UML 2 子系统，或映射到应用了“ 子系统 ” 关键字的软件包。*

3. 设计元素的组织在演变中可能背离系统用例在用例模型中或分析模型中（如果保留着独立的分析模型）的组织方式。使用软件包将设计合同进一步细分成设计元素规范（用途合同）以及设计级别的用例实现（实现合同），并为用例实现保留软件包子结构，该结构将继续反映用例本身的组织。

4. *请考虑：*对于组成功能区域规范和实施设计的元素，使用体系结构层作为其二级组织方案的基础（并请参阅下面的深入讨论）。

5. 在已将语义模型元素分组的 UML 组件和软件包中，放置一些图来提供特定于该分组的视图。本指南针对的是该分组是基于以功能为导向的部分业务领域、基于体系结构层还是基于您所拥有的其他内容。请使“ 缺省 ” 图的名称与软件包或组件本身的名称相同，并对其进行构建以显示软件包内容的概要。这使一些图接近于它们描述的内容，使浏览和了解模型变得更加容易。

6. 您可能需要在设计模型中使用反向因特网域名称空间。理由：
   - 基本上，原因亦是：这样做从特定于语言的实施的角度看很重要：
     a. 涉及集成工作的场景，其中涉及了多个模型驱动的应用程序（特别对于合作伙伴公司）
     b. 复用场景
   - 这可能会简化转换到实施的后续配置（源到目标的位置和名称映射）。

7. *考虑*使用在目标实施平台中有效的软件包名称，以避免名称空间映射的负担和可能发生的混淆。（在很大程度上，这只是意味着“ 在名称中不要使用空格或除下划线以外的其他标点符号 ”。）

8. 对软件包名称使用小写，以使它们更容易与软件包中的类名区分。

9. *考虑*对接口和实现它们的组件或类使用不同的名称。将 ILoan 和 Loan 或者 Loan 和 LoanImpl 用作接口和实施的名称。这在模型中确实并非必需，但在生成的代码中通常是个不错的主意，所以这是您可以让自己省去一些后续转换配置工作的又一个地方。

10. 在以下场景中，任何分析级别内容（不会从中生成代码）应在构造型为“ 分析 ” 的软件包中进行隔离[8]。
    - A) 您选择了不使用独立的分析模型，而是使用分析级别的内容填充设计模型，并以分析抽象级别保留该内容，同时还在同一模型中创建设计级别的内容，并且
    - B) 您将从 EIT 设计模型驱动模型到代码的转换

11. 使用“ 透视图 ” 软件包中的图来捕获设计元素的高级别交叉视图。理由：提供交叉视图、“ 在体系结构上有重要意义的 ” 内容的视图，以及吸引不同类型的项目干系人的视图，同时保持模型的语义元素组织成以功能为导向的分组。

[8] 转换时将绕过此类软件包。

设计模型的封装结构将随时间而演变，认识到这一点是很重要的。最终，组织方式应该对应于您将体系结构构建为组件和服务的方式。然后，用于对设计进行*最后阶段*组织的这一方法一般具备封装可复用资产的最佳潜力以及从设计到这一组项目和文件夹的最直接映射，这些项目和文件夹将存放从设计生成的实施工件（代码、元数据、文档）。

但是，*初始*组织应当或多或少地直接对应您用于用例模型然后又在分析期间修订的组织方法[9]。实际上（如前一节“分析模型的内部组织指南”中所述），您可以选择让您的分析模型在原处演变为设计模型。换言之，设计的初始组织往往会将高耦合及低耦合的多组业务问题组织在一起，并分离出交叉元素或可复用元素。这种用于初始组织的方法经证明很有效，因为

- 如果您希望使用从分析或用例模型内容生成设计模型内容的转换，则源软件包到目标软件包的映射将简单而直接。
- 基于软件包的功能高耦合和低耦合的初始组织方法显然是映射到最终面向组件的组织的最佳选择，这意味着它会减少作为设计过程的一部分而必须执行的重构量
- 软件包的低耦合具有潜在能力，可以改进团队工作流，并且在将设计分解成多个建模文件时，可以为复用提供便利

当然可以使用其他方法，而且在一些情况下作为*最后阶段*组织还是建议使用的：

- 如果您的目标是基于 J2EE 的 Web 应用程序（包括 EJB），则设计的组织就可以预计 RSA 和 Rational Application Developer 关于 J2EE 项目的约定。[10] 特别是您可能会选择定义与体系结构层（展示和业务，其中业务进一步分层为会话和域）相对应的顶级设计包。这显然不是与平台无关的方法，因此建议只有在您知道所设计的解决方案将不会在 J2EE 之外的平台上实施时才使用这种方法。
- 更一般地讲，在构建多层应用程序时通常出现的情况是开发人员专长和分工对应于展示和业务层，因此您可能再次选择使用与那些体系结构层相对应的顶级软件包。但是在组织用于支持特定*业务功能*的类以便支持特定*体系结构*时，请谨慎。它会使两者中的任何一个都会变得更难更改。
- 如果您有理由使用“不以组件／服务／子系统为导向”的组织方法，您应该仍然能够通过在配置代码生成转换时做一些额外工作，将设计的组织映射到一组目标项目和文件夹。一种特殊类型的称为“映射模型”的同类模型可以用来定义特别复杂的映射。

设计模型内容

对于设计模型中应该有什么内容并没有必须遵守的规则，但是以下建议可能会证明是有用的。

[9] 已经发现分析类的封装通常会有重大的重构，这是为了更好地支持复用和未预料到的功能需求。

[10]不严格地讲：每个系统或应用程序或大的子系统一个企业项目，对于每个企业项目，展示层一个 Web 项目，以及每个组件或子系统多个 EJB 项目，其中 EJB 项目一般与组件或小的子系统相对应，而且一般独立的 EJB 项目用于会话（会话 EJB）和域（实体 EJB）。请参阅本白皮书的第 9 部分获得更多信息。

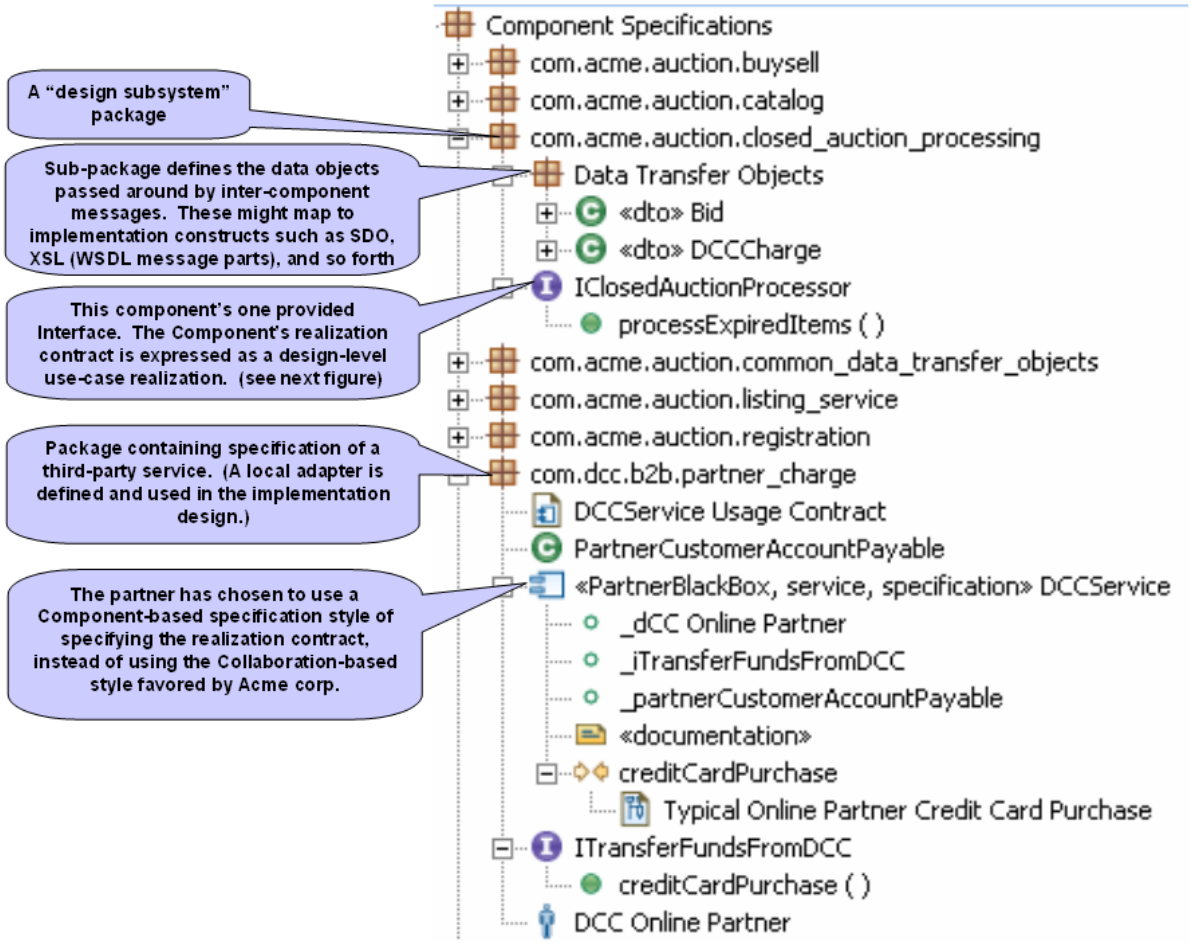

A “design subsystem” package
Sub-package defines the data objects passed around by inter-component messages. These might map to implementation constructs such as SDO, XSL (WSDL message parts), and so forth
This component’s one provided Interface. The Component’s realization contract is expressed as a design-level use-case realization. (see next figure)
Package containing specification of a third-party service. (A local adapter is defined and used in the implementation design.)
The partner has chosen to use a Component-based specification style of specifying the realization contract, instead of using the Collaboration-based style favored by Acme corp.
Component Specifications
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
Data Transfer Objects
«dto» Bid
«dto» DCCCharge
IClosedAuctionProcessor
processExpiredItems ( )
com.acme.auction.common_data_transfer_objects
com.acme.auction.listing_service
com.acme.auction.registration
com.dcc.b2b.partner_charge
DCCService Usage Contract
PartnerCustomerAccountPayable
«PartnerBlackBox, service, specification» DCCService
_dCC Online Partner
_iTransferFundsFromDCC
_partnerCustomerAccountPayable
«documentation»
creditCardPurchase
Typical Online Partner Credit Card Purchase
ITransferFundsFromDCC
creditCardPurchase ( )
DCC Online Partner

图 7-2

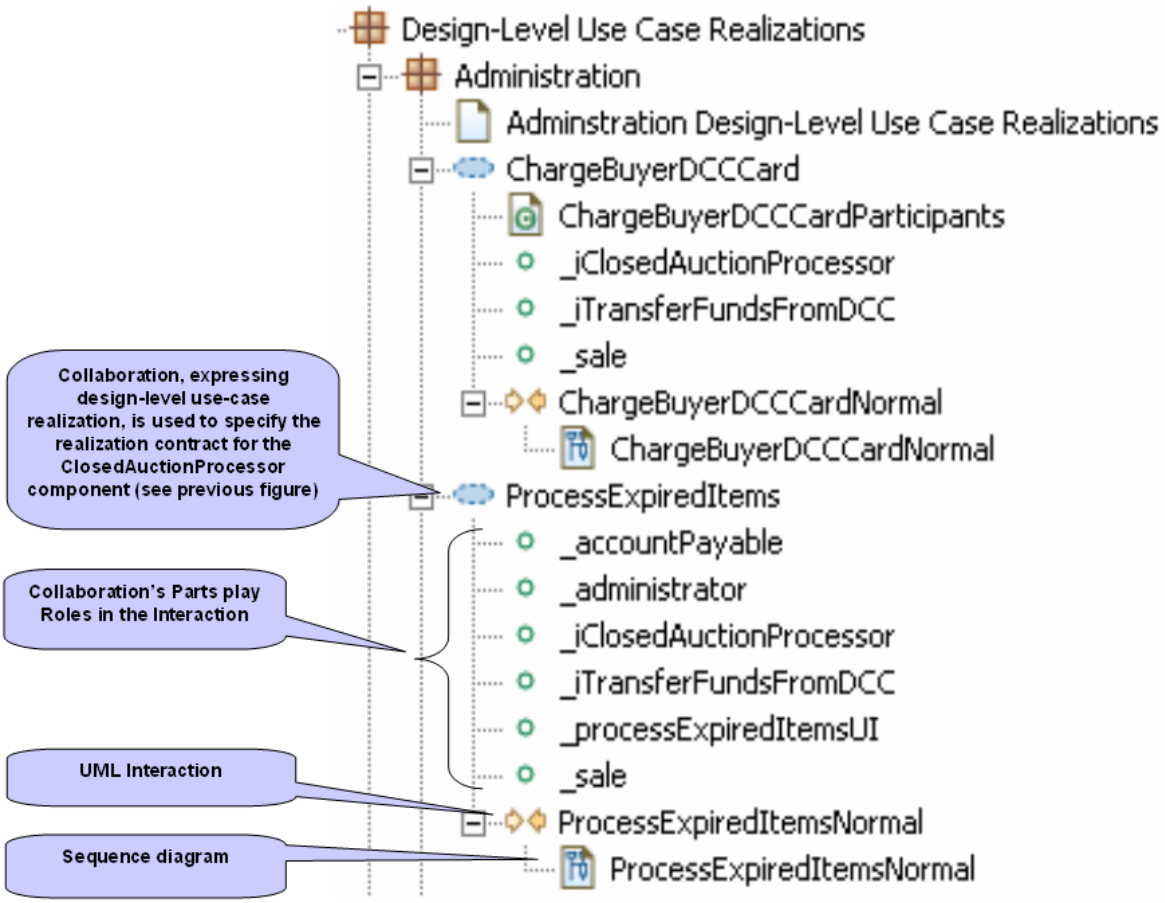


图 7-3

图 7-2 和**图 7-3** 遵循**图 7-1** 中所述的组织结构，并描述了如何指定设计合同。

- “ ClosedAuctionProcessor ”组件的用途合同表示为单个接口[11] (**图 7-2**)。对应的实现合同由表示为“ 协作 ”的单个设计级别用例实现来指定[12] （**图 7-3**）。请注意，分析级别的用例实现显示分析类之间的协作，而设计级别的实现则显示抽象级别较低的设计元素之间的协作[13]。如果期望独立于设计模型的实施设计子集对规范子集进行封装，则设计级别的用例实现应该在角色中只使用分析或设计规范元素，而决不使用实施设计元素，这一点很重要。
- 第三方“ DCCService ”的用途合同和实现合同一起装在一个软件包中[14]。我们再次看到用途合同由单个接口构成，但是在这种情况下，实现合同用“ 规范 ”组件表示（**图 7-2**）。（否则，实现合同的规范就是相同的，都用行为来表示 – 在此例中，交互名为“ creditCardPurchase ”）。另一个使用组件而不使用协作的示例显示在**图 7-4**

[11] 组件当然可以有多个已提供的接口，本示例只有一个接口纯属偶然

[12] 其他组件可能参与多个系统用例，因此它们的实现合同可能位于多个用例实现中。在这样的情况下，您也可以在与组件的接口相同的软件包中包含称为“ {组件名称}使用的地方 ”的图，在图上您可以放置到组成那些用例的用例实现的各种图的链接。

[13] 另一个可能的区别是：设计级别实现中的某些“ 参与者 ”图可能是描述组件连线的组件图，而不是（或者同时是）为分析级别用例实现提议的参与者类图。

- [14] 请注意，这里假设的情形是 DCC 公司向 Acme 公司提供 UML 规范，然后 Acme 将该规范合并到其设计模型中。那是反向因特网域空间可以派上用场的场景类型。

- 操作在接口中定义，接口可以由 “ 规范 ” 组件实现（如果使用），或者由实施设计中用于实施接口的分类器实现。

- 数据传输对象（将用作所提供操作的参数类型，并且可以映射到诸如 XML 模式或 SDO 之类的实施构造）的规范也可以包含为用途合同的一部分。对于设计为不可分发的组件，您可以选择将数据传输对象指定为用作操作参数的类型的规范，也可以选择不这样做。对于可分发的服务（例如 Web service），强制规定服务的操作不能引用本地地址空间中的对象，因此必须使用 DTO。

XDE/Rose
在先前版本的 UML 中，用例实现的指导信息是每个用例使用一个协作实例，而且每个重要的实现流程使用一个交互和时序图。

在 RSx 中，您通常只能使用一个交互和图，因为 UML 2 时序图现在支持备用执行路径的表示法。

同时，在 UML 2 中已经没有 “ 协作实例 ” 了。取而代之的是 “ 协作使用 ”，它要求将 “ 协作 ” 作为其类型。因此在 RSx 中，使用 “ 协作 ” 代表用例实现。）

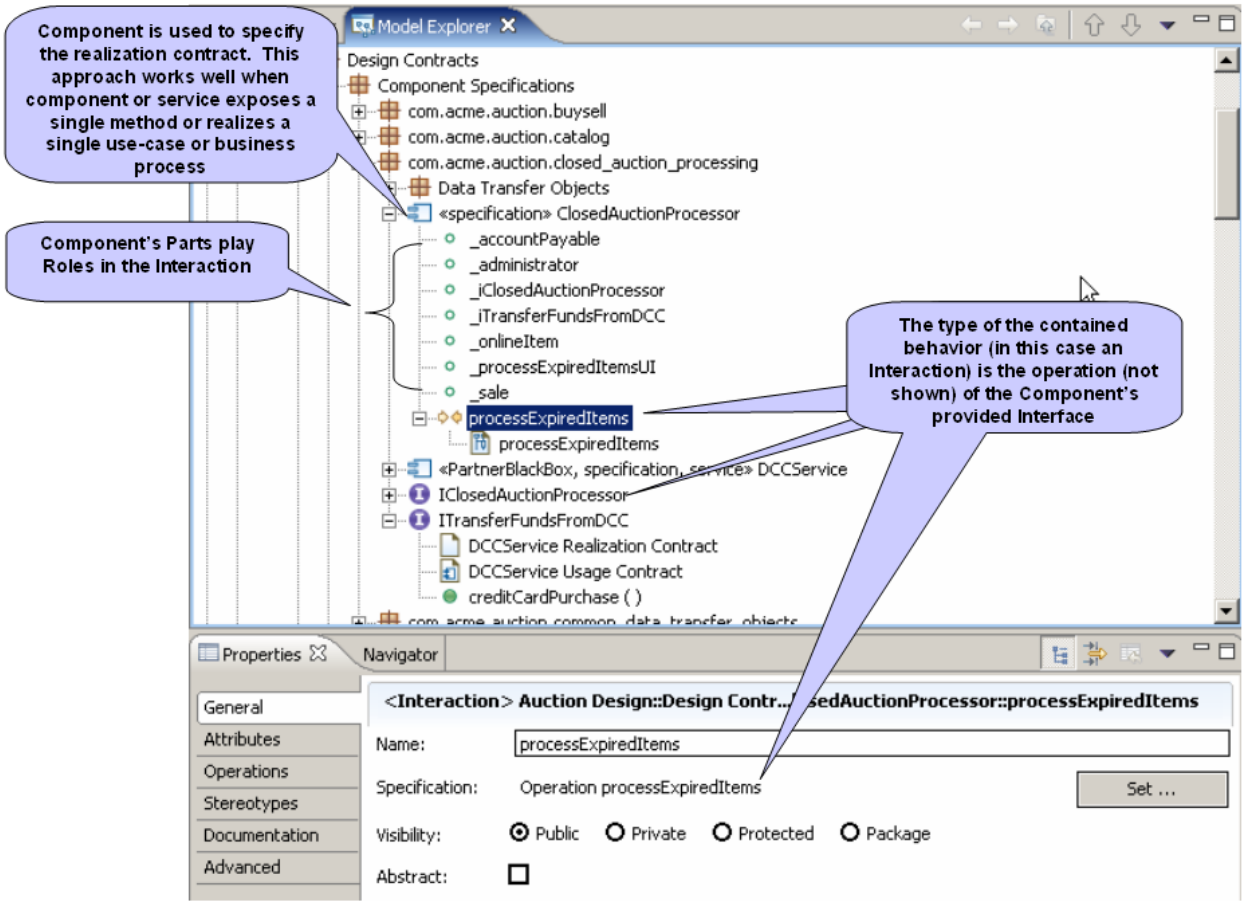


图 7-4

- 可用于指定实施设计的方法显示在**图 7-5**。实施结构是使用包含操作的简单类定义的。这种方法对使用 UML 1.x 创建的设计模型非常典型。第二种可用的方法可能更注意与 UML 2 的目标保持一致，这种方法显示在**图 7-6** 中。在这里，我们看到使用的是“ 组件 ”而不是“ 类 ”，并且看到“ 组件 ”不拥有“ 操作 ”，而是拥有行为（在本例中是“ 交互 ”）。

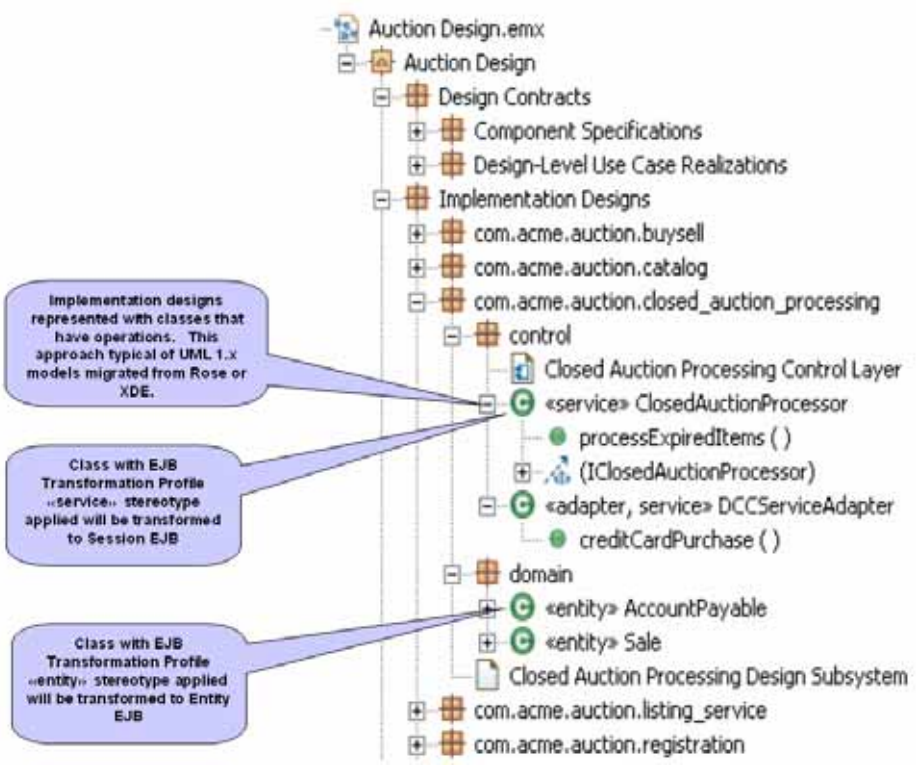

Auction Design.emx
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«service» ClosedAuctionProcessor
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
Implementation designs represented with classes that have operations. This approach typical of UML 1.x models migrated from Rose or XDE.
Class with EJB Transformation Profile «service» stereotype applied will be transformed to Session EJB
Class with EJB Transformation Profile «entity» stereotype applied will be transformed to Entity EJB

图7-5

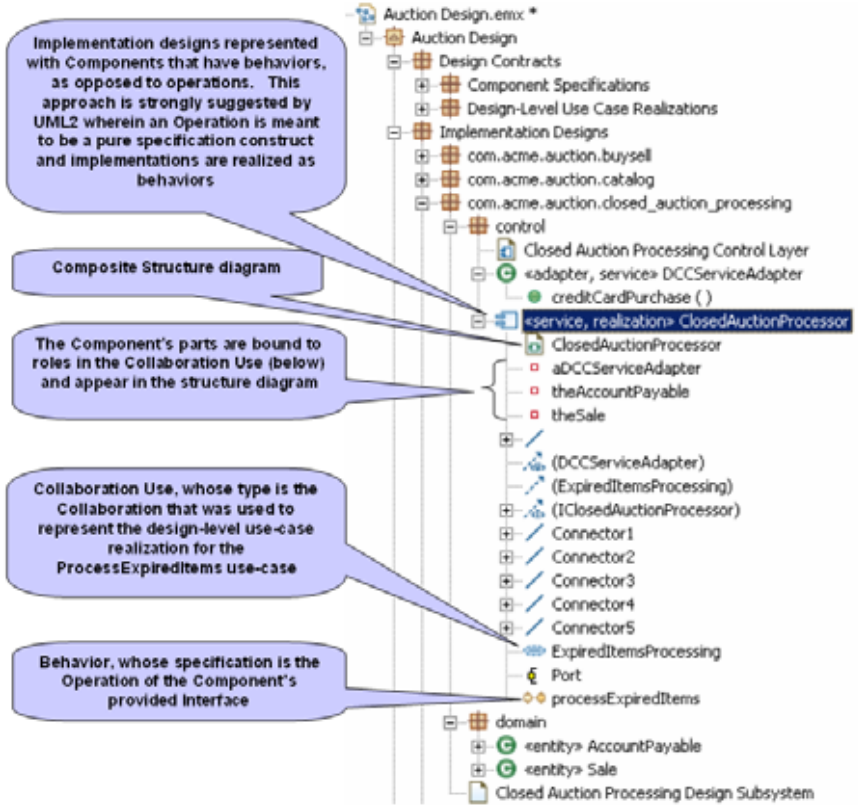

Auction Design.emx *
Auction Design
Design Contracts
Component Specifications
Design-Level Use Case Realizations
Implementation Designs
com.acme.auction.buysell
com.acme.auction.catalog
com.acme.auction.closed_auction_processing
control
Closed Auction Processing Control Layer
«adapter, service» DCCServiceAdapter
creditCardPurchase ( )
«service, realization» ClosedAuctionProcessor
ClosedAuctionProcessor
aDCCServiceAdapter
theAccountPayable
theSale
(DCCServiceAdapter)
(ExpiredItemsProcessing)
(IClosedAuctionProcessor)
Connector1
Connector2
Connector3
Connector4
Connector5
ExpiredItemsProcessing
Port
processExpiredItems
domain
«entity» AccountPayable
«entity» Sale
Closed Auction Processing Design Subsystem
Implementation designs represented with Components that have behaviors, as opposed to operations. This approach is strongly suggested by UML2 wherein an Operation is meant to be a pure specification construct and implementations are realized as behaviors
Composite Structure diagram
The Component's parts are bound to roles in the Collaboration Use (below) and appear in the structure diagram
Collaboration Use, whose type is the Collaboration that was used to represent the design-level use-case realization for the ProcessExpireditems use-case
Behavior, whose specification is the Operation of the Component's provided Interface

图7-6

8. 实施概要模型的内部组织指南

> ***XDE/Rose***
> 在 XDE 模型构造指南中，建议将实施概要模型作为向实施提供子系统级别概要的方法。每个子系统的详细信息则是在实施该子系统的项目的代码模型中指定的。

严格地讲，在 RSx 中使用实施概要模型应该不是必需的。如果遵照设计模型组织指南，那么设计模型的（最后阶段）组织自然应该构建组件（包括更强大的“ 子系统 ”和更可分发的“ 服务 ”变体）。然后，通过转换可将设计包映射到项目。例如，如果是 J2EE 实施，它们将映射到各种 Java、EJB、Web、J2EE 应用程序以及从中开发了实施的其他项目中。（那些项目实际上代表解决方案的实施模型，如本白皮书的“ 基本概念和术语 ”一节中所述。）

如果您以自下而上的角度考虑，则应当考虑如何根据项目和文件夹对实施进行组织，并将此分解到设计模型的组织中，这样就能直接完成转换映射。或者，如果您以自上而下的角度考虑，则由于设计模型的组织已随着设计工作的进展而发生演变，因此应当确定所需的那组实施模型（项目）。无论哪种方法，设计模型的最后阶段组织自然应该针对“ 项目和子项目概要 ”这一需要，即：描述设计模型中软件包的图应该等同于项目及其子文件夹的概要。

但是，您可能还是喜欢在早期阶段画出项目结构的草图，或者可能只是想看到更直观明了的项目结构描述 — 例如，代表项目和文件夹的工件用关键字明确地表示为“ 项目 ”和“ 文件夹 ”，甚至表示为“ EJB 项目 ”和“ Web 项目 ”。另一个考虑事项是描述细化的实施工件（例如 JAR）不适用于设计模型（在 Rational Software Architect 的操作理论中，这是与平台无关的）。但这样的工件完全适合包含在实施概要模型中。因此，有一些原因可能使您希望使用实施概要模型。**图 8-1** 下面描述了一个实施概要模型样本。

最后的一个想法是，实施概要模型可能是捕获解决方案的各个方面的非正式图的好地方。**图 8-2** 下面显示了拍卖系统的非正式高概念图，本白皮书的大多数样例都基于该图。

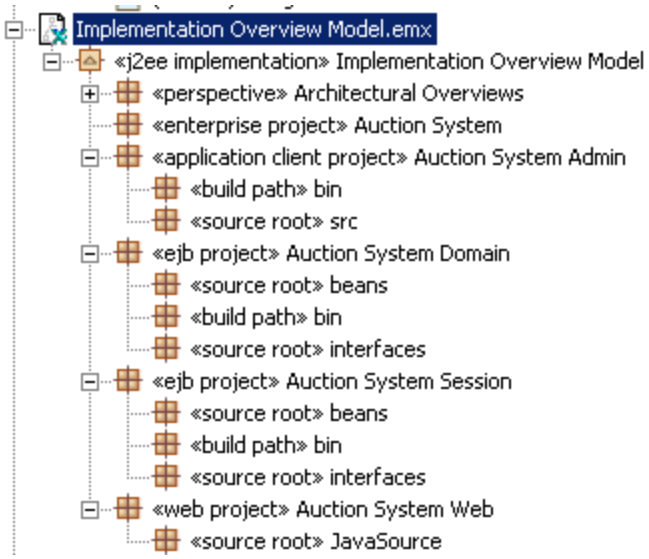

Implementation Overview Model.emx
«j2ee implementation» Implementation Overview Model
«perspective» Architectural Overviews
«enterprise project» Auction System
«application client project» Auction System Admin
«build path» bin
«source root» src
«ejb project» Auction System Domain
«source root» beans
«build path» bin
«source root» interfaces
«ejb project» Auction System Session
«source root» beans
«build path» bin
«source root» interfaces
«web project» Auction System Web
«source root» JavaSource

图 8-1

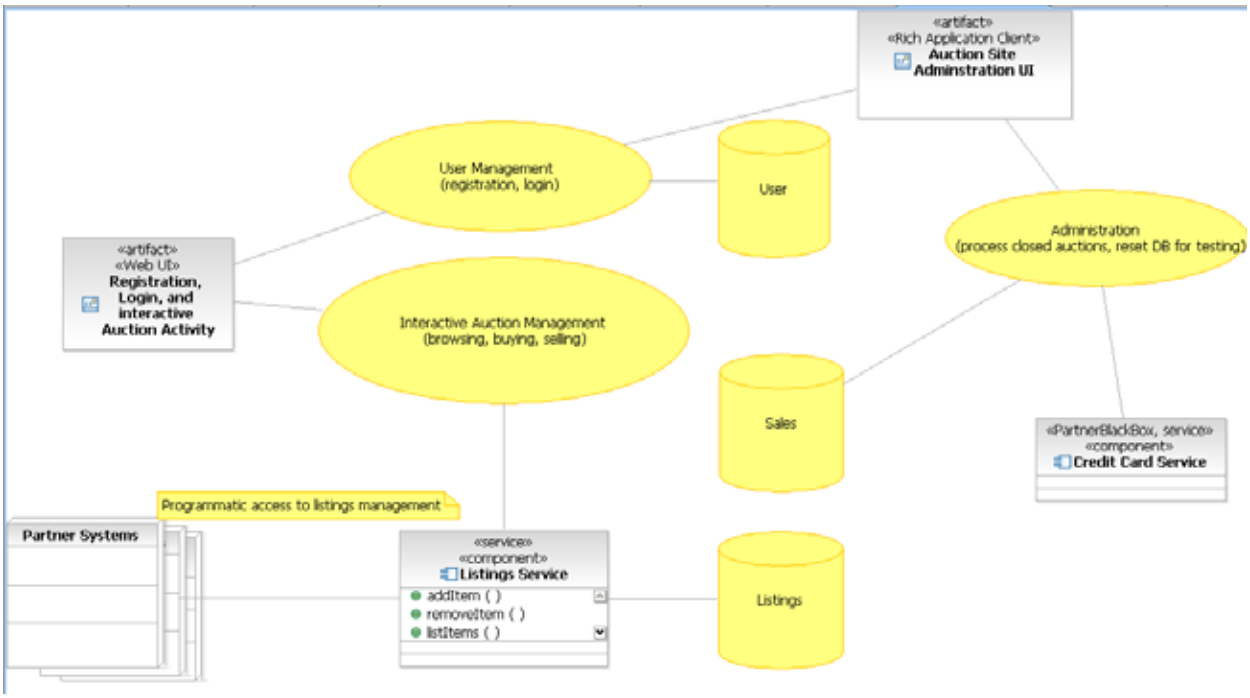

«artifact»
«Rich Application Client»
Auction Site
Adminstration UI
User Management
(registration, login)
User
Administration
(process closed auctions, reset DB for testing)
«artifact»
«Web UI»
Registration,
Login, and
interactive
Auction Activity
Interactive Auction Management
(browsing, buying, selling)
Sales
«PartnerBlackBox, service»
«component»
Credit Card Service
Programmatic access to listings management
Partner Systems
«service»
«component»
Listings Service
addItem ( )
removeItem ( )
listItems ( )
Listings

图 8-2

9. 部署模型的内部组织指南

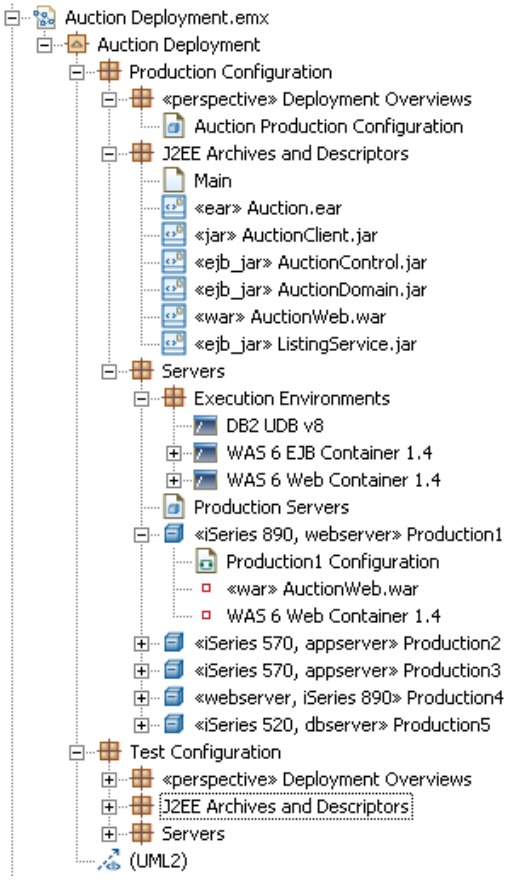


图 9-1

关于部署模型，需要阐述的内容可能比本白皮书中提到的其他任何模型都要少。您的部署建模组织和内容选择通常应该只有很少的下游含义，因此只要执行有意义的操作就可以了。在**图 9-1** 中所述的一个可用的战略和几个具有代表性的内容仍然只是为了让您思考。此示例中有几个值得注意的事项：

1. 生产配置的规范已经独立于测试配置的规范

2. 概要（例如集群、数据中心或企业的概要）保留在 “ 透视图 ” 软件包中

3. 在节点和工件中进行专门化和分类的时候已采用了轻量级的方法：封装和使用关键字相结合。更成熟的方法是开发特殊的 UML 概要文件定义用来描述和记录用在您自己环境中的资源类型的特殊化构造型和属性。

10. 使用建模文件表示软件体系结构文档

假如有它用于组织模型的工具（如图链接以及用跨模型引用对多个模型文件的支持），创建实际上代表 RUP 软件体系结构文档和“ 4+1 体系结构视图 ”的模型几乎是个小问题了。

在最简单的形式下，您可以按照**图 10-1** 执行一些操作。创建建模文件并用与 4+1 视图相对应的一组简单的软件包来填充它。（显示的示例没有用于流程视图的软件包，这是因为此示例中的系统以并行方式展示的信息不多。）

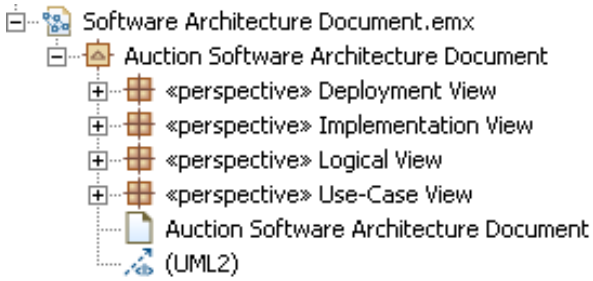


图 10-1

然后可以按照**图 10-2** 中的提议来构建缺省图。您也可以将其他注释或文本添加到该图中。

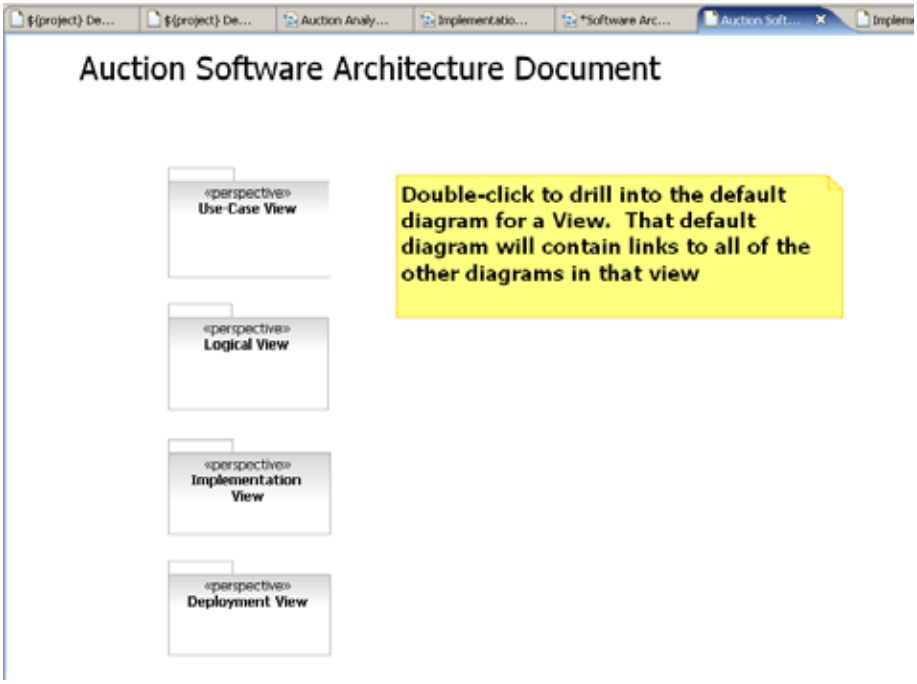


图 10-2

然后只要用以下方法在软件体系结构文档建模文件中创建图就可以了：

- 创建一些图，它们是使用其他建模文件中的 UML 语义元素构建的，用于描述在这些其他建模文件中没有的、但作为体系结构文档的一部分所需的新视图

- 创建由几何形状和／或位于软件体系结构文档建模文件的“ 即兴 ” UML 元素组成的图。（此类 UML 元素应当仅用于记录或阐述目的，应该对所述解决方案的实际实施没有语义意义）

- 创建仅包含到其他建模文件中现有图的链接的图。（该技巧在体系结构文档建模文件随其他建模文件一起分发供读者使用时能起很好的作用。如果体系结构文档将在 Web 上发布，请遵循其他方法）

11. 团队开发和模型管理注意事项

本节将介绍有关何时及出于何种原因可以选择将模型分割成多个建模文件的一些注意事项。对这些问题的更全面的处理方法可以在 RSx 联机帮助上找到。这里假定读者对下列概念有一定程度的了解：并行开发，以及归并已对某个工件的多个副本并行执行的更改。

首先快速回顾：我们在“ 基本概念和术语 ”一节讨论了 RUP 所能识别的各种模型类型，例如“ 用例 ”、“ 分析 ”和“ 设计 ”。提供了几个示例以说明在 RSx 中：

- 如果您要构建多个应用程序，则可能需要为每种类型创建多个模型（例如多个用例模型、多个分析模型等）
- 一个模型（在逻辑意义上）可以保存为一个或多个建模文件，例如应用程序“ X ”的设计模型可以保存为一个单独的建模文件，也可以保存为由多个建模文件构成的集合。

分割模型

将模型分割为多个建模文件的方法在 RSx 联机帮助中有述，这里就不再介绍了。在这里，我们感兴趣的是在何时以及出于何种原因需进行分割。在两种情况下，您可能会选择将某个特定的模型保存为多个建模文件：

1. 模型的大小已增加到无法管理的程度，或者其封装结构的深度已增加到无法管理的程度[15]
2. 您开始遇到过多用户同时提交对某一建模文件的更改，从而导致需要通过两个以上的变更添加程序执行归并[16]。（如果您觉得此表述不够清楚，请继续阅读。）

团队建模

如果您的配置管理策略支持模型的并行开发[17]，将对文件进行未协调的变更（实际上是对该文件所代表的逻辑模型或模型的一部分进行变更）。在某一时刻这些变更必须归并。如果发现某些变更彼此存在冲突，则认为该归并是“ **普通** ”的，因为必须人工决定哪个冲突变更应“ 胜出 ”。（“ 复制型 ”归并就是不存在冲突的未协调变更，因此模型归并引擎能够自动执行归并而无需人工干预。）
“ 普通 ”归并执行起来可能非常麻烦。如何才能最大程度地减少此类归并呢？

您有两种基本的“ 武器 ”可避免冲突和导致的“ 普通 ”归并。一种是“ 强体系结构 ”。另一种是“ 强所有权 ”。它们的关系很密切：强体系结构*实现*强所有权。

武器 1：强体系结构

[15] 主要当文件的大小对于用户群中正在使用的机器而言变得过大时需要分割。例如，磁盘上达到 30MB 的模型会变得非常难以在 RAM 为 1GB 的机器上进行日常使用。在此类机器上，需要对该模块进行分割，目标是任何时候 RAM 中的模型大小始终保持 5 至 10MB。替代方法是升级 RAM（这种解决方法相对廉价，却是一种非常有效的方法 – RAM 为 2GB 且没有交换文件的机器几乎在执行每项 Eclipse 操作时都要快得多，因此即使模型很大仍然能够很好地执行。）

[16] RSx 模型归并工具支持最多通过 3 个添加程序进行归并：两个“ 变更 ”添加程序，一个“ 基本 ”添加程序（也称为共同祖代）。当有两个以上的“ 变更 ”添加程序要处理时，归并将开始级联，并且从此刻起两个“ 变更 ”添加程序中的一个成为会话中先前完成的归并的结果集。

[17] “ 并行 ”开发策略的示例：

- 对于非独占的访问，可检出建模文件
- 在具有多个工作人员的开发流中以并行方式处理某个建模文件

此上下文中“ 强体系结构 ”主要指分解。此处应用的体系结构分解*原理*与驱动“ 面向对象的开发 ”、“ 基于组件的设计 ”以及“ 面向服务的体系结构 ”的原理相同：

- 最大限度地消除业务功能的耦合
- 将必须保持紧密耦合的事物归在一起。将这些分组彼此隔离
- 如果您得到的分解有大量的“ 颗粒 ”，则根据您的人员配备模型（请记住，强体系结构和强所有权紧密相关），您可能需要着手将这些“ 颗粒 ”分组成高亲缘关系聚集（就建模而言是 UML 软件包）
- *始终*有些内容是许多（某些情况下是所有）分解单元必定会触及的。将这些内容集中在一个“ 公共 ”软件包中并对每个开发迭代进行规划，使它在迭代起始处包含侧重于稳定“ 公共 ”内容的一点“ 瀑布 ”。
- 还有一个时间元素。当您对解决方案的理解从较抽象转变为较具体时，您对体系结构（和模型）的最佳组织的认识就将深化。您应当在每个阶段转换到下一个阶段（业务分析、需求、应用程序分析、高级别设计……）时规划一次模型重构（重新组织）活动。

如果您发现您的解决方案和一切内容似乎都高度地相互依赖且紧密耦合，则可能是您的体系结构需要改进，或者是问题域的性质存在某种情形使您实际上无法分解问题。在任何一种情况下，您都有多种选择：

- 将项目分配给一个很小的团队，该团队共享一个物理空间，并且其成员就各自所作的可能影响其他工件的变更，彼此非常积极地进行交流。
- 准备好执行许多普通归并

XDE/Rose

如果您是 Rose 或 XDE 用户，则可能有执行普通模型归并的经历。所幸在 RSx 中，普通归并不再像从前那样麻烦了。最大的区别在于 RS*x* 是对*一组*相关建模文件进行操作，而不是对*一系列* petal 文件或子单元进行操作。这一关键区别意味着我们在 RS*x* 中支持在物理级别执行好得多的高级别逻辑分解。

我们还支持用更好的工具进行比较归并，实际上将拥有**好得多**的归并体验。

- RS*x* 中的后端归并引擎比 XDE 后端快 10000 倍（没错，一万倍）。
- 它还实施了大量的技术，例如“ 组合增量 ”（一种分组机制，按图对变更进行归类，并将大的制图活动集中在一起以实现组操作和／或原子操作）、“ 模型完整性保护 ”、“ 原子性 ”和“ 级联增量处理 ”。这些技术使得由于归并而使文件损坏的可能性降至与只是由于编辑文件而使文件损坏的可能性大致相等的水平。
- 现在有一个直观的图归并 GUI 可用于解决布局／表面冲突

武器 2：强所有权

一旦建立了强体系结构分解，将体系结构组件的“ 强 ”所有权映射到个别工作人员或小型团队应是相当直接的（具备专门技能）。如果模型中的每个逻辑软件包（或分支）都能够由某个工作人员以独占方式操作，则对该模型执行的归并大多数是复制型归并（无论模型存储为单个建模文件还是多个建模文件）。如果每个分支可以由某个小型团队以独占方式操作，并且团队成员倾向于就自己所做的工作进行大量的非正式交流，则同样可以得出上述结论。

可以通过将模型分成多个建模文件来避免普通归并吗？回答是：“ 不可以 ”。**体系结构相互依赖性是逻辑现象，而非物理现象。**如果您将模型分成多个建模文件，元素相互依赖性的表现形式只是从文件内部引用变成了文件交叉引用。您并没有使冲突变得容易解决（事实上使冲突变得更难解决）。并且当您引入文件交叉引用时，就引入了潜在的损坏点（请参阅侧栏）。

侧栏：建模文件交叉引用

只要两个建模元素位于不同的建模文件中，并且您在它们之间创建了关系，就创建了一个“ 建模文件交叉引用 ”。因为建模文件（.emx 文件）暴露在主机操作系统的文件系统中，并且能够在 Eclipse 环境之外移动、重命名或以其他方式修改，所以这些引用代表潜在的损坏点。但是，只要您始终通过 Eclipse 环境对建模文件进行修改和变更管理，并且遵守以下指南，就不会遇到损坏。

当您处理（编辑）一组“ 封闭 ”的建模文件（即一组彼此引用的建模文件）时，应当使该封闭组中的所有建模文件都出现在工作空间中。这并不严格地表示封闭组中的所有建模文件都必须位于同一项目中，但是使用一个项目通常能够确保所有的模型都存在，这是由于在典型的 CM 工作流中，所有的模型文件都汇集到同一项目中。

现在让我们总结一下：

- 如果您缺少强体系结构，或者虽然具有强体系结构但缺少强所有权，则将会频繁地遇到普通归并，并且再如何进行模型分割也无济于事。
- 如果您具有强体系结构和强所有权，则将极大地降低普通归并的频率（但无法彻底消除）。无法彻底消除普通归并是由于组件相互依赖性始终存在。前面提到的“ 公共 ”元素就是一个例子，虽然几乎是唯一的例子。
- 将模型分割为多个文件的重要程度远比不上在逻辑上构建模型，后者能够使多个工作人员并行操作同一建模文件而不会引入冲突的变更。
- 所幸在 RS*x* 中，处理模型归并比现有的其他任何建模工具都要快得多且高效得多。

那么何时应当分割模型呢？请尽量不要这么做，除非整个模型的大小开始考验您硬件的承受能力，而您创建的建模文件通常能够同时以*独占*（即只有一个团队成员能够随时将某个文件检出）**和***隔离*（即可对文件进行大多数变更而无需访问包含相关模型元素的其他文件）的方式操作。

后记：RSx V6.x 与 V7.x 之间的差异

当 RSx 最初发行时（V6.x），不支持 Rose 和 XDE 所支持的 “ 子单元 ” 概念（子单元是包含了一部分模型的 “ 透明 ” 建模文件，这里的 “ 透明 ” 是指它不出现在模型的逻辑视图中，您将在模型浏览器视图中看到该视图）。而模型分割还被限制为定义多个顶级模型（ “ 顶级 ” 的含义是每个建模文件在模型浏览器视图中会显示为一个独立的顶级条目）。

现在，RSx 还使您能够选择现有模型的 UML 软件包，并根据该软件包 “ 建立模型 ” 。您可以将此操作看作对软件包进行 “ 分割 ” 来创建一个新的建模文件。结果就是在模型浏览器视图中，该软件包成为了新的顶级逻辑模型。原始分层包含结构的可视性将丢失，但是只要您打开一个建模文件（已经从中以该方式 “ 分割 ” 了软件包），则所有的 “ 已分割 ” 模型也会打开。这可能导致不必要的检出，以及在模型浏览器视图中发生拥塞，并且在编辑器窗格中出现大量的选项卡，这些问题一起使导航变得困难。

有了 RSx V7.0 中添加的子单元支持，使您能够将模型分割成多个文件而不会丢失分层结构的可视性，并且除去了编辑器窗格中的 “ 模型编辑器 ” 选项卡，这样就克服了其他导航问题。

但是，有一个从早期 6.x 经验中得到的关于模型构建的教训仍然适用于今天。如上所述，打开已从中分割了软件包的建模文件将导致打开所有的已分割模型。这个问题可通过提前规划分割策略来避免，而不是通过使用 “ 建立模型 ” 功能。

那些遵循 “ 建立模型 ” 功能的用户通常会先建立一个单独的建模文件，然后创建软件包来表示解决方案体系结构的组织。例如，解决方案的每个主要组件或子系统可能有一个软件包（其中 “ 组件 ” 和 “ 子系统 ” 反映的是这些术语的非正式用法，这可以回溯到计算技术的早期，而并非它们的正式 UML 语义定义）。可能还有针对 “ 公共 ” 、 “ 实用程序 ” 或 “ 框架 ” 组件的软件包。然后，随着此类模型的发展，与特定于应用程序的组件对应的软件包可以分割为多个独立的建模文件，这样构建这些组件的团队（在理论上）就可以用隔离方式处理这些模型文件。但是在实际情况下，打开这些特定于组件的模型文件中的任何一个都会导致打开原始的 “ 主 ” 模型（ “ 公共 ” 组件所在的模型），并且该模型接着将打开所有其他特定于应用程序组件的模型。其结果就是前面提到的不必要的检出和导航困难。

较好的方法是提前进行规划来为每个特定于应用程序的组件定义一个单独的模型，并为 “ 公共 ” 组件定义一个模型（也可能是多个模型，它们定义公共／可复用组件的后续层，即：反映实施体系结构的抽象层）。然后，任何特定于应用程序的组件模型都可以由拥有该模型的团队打开，并且同时需要打开的只有此特定于应用程序的模型所依赖的 “ 公共 ” 模型。