Diretrizes da Estrutura do Modelo
Para Rational Software Modeler, Rational Systems Developer e Rational Software Architect
(Orientação “RUP Tradicional”)

White Paper

Bill Smith, Model Driven Development, IBM Rational Software

V7.0
24 de março de 2006

Índice

1. Introdução

Público-alvo

Este documento foi projetado para oferecer suporte aos usuários dos produtos RSA (Rational Software Architect), RSD (Rational Systems Developer) e RSM (Rational Software Modeler) (coletivamente "**RS*x***"), especialmente aos interessados em aplicar a diretriz de modelagem encontrada no RUP® (Rational Unified Process) para uso do RS*x*. Se você for um usuário do RSM (Rational Software Modeler), achará o documento útil, mas observe que algumas seções refletem os recursos disponíveis somente no RSA e no RSD.

O documento se concentra na criação de um novo conjunto de modelos no RS*x*. Se você for inexperiente no RS*x*, mas tiver sido um usuário do Rational Rose ou do Rational XDE e planeja importar modelos desses produtos, poderá também encontrar no documento uma fonte de orientação para reestruturar seus modelos importados.

O documento presume que você tenha algum conhecimento do RUP e do UML. Presume ainda que você esteja familiarizado com os conceitos fundamentais e com a "teoria das operações" do RS*x*, sobre os quais poderá ler em "**Os Novos Produtos de Design e Construção IBM Rational para Usuários Rose e XDE**"

Finalidade

O RUP descreve um conjunto de modelos, por exemplo, modelos de caso de uso, modelos de análise e modelos de design, que representam perspectivas bem definidas sobre domínios de problema e solução de sistemas. O utilitário desse conjunto de modelos foi comprovado em muitos projetos reais. Mesmo que você não siga o RUP, essas estruturas de modelo merecem ser consideradas. Este documento descreve como percebê-las durante o uso do RS*x*.

Outras abordagens sobre modelagem também poderão ser consideradas, mas a orientação de estruturas de modelo para suportá-las está além do escopo deste documento. Por exemplo, uma abordagem "Business Driven Development" de modelagem para Arquiteturas Orientadas a Serviço poderia ser utilizada. Ela começaria com um modelo de processo de negócios – talvez expresso como um modelo de Atividade UML 2 – e depois prosseguiria diretamente para a especificação dos contratos de um conjunto de serviços que automatizaria as tarefas no processo de negócios. Essa abordagem sugeriria estruturas de modelo bem diferentes daquelas descritas neste documento.

É importante entender que as estruturas de projeto e modelo descritas neste documento são diretrizes, não imposições. Decidir ou não modelar um artefato RUP específico no RS*x* é uma consideração de seu próprio processo de desenvolvimento. Muitas vezes, é uma decisão específica do projeto. Além disso, lembre-se de que o RUP não é um conjunto rígido de regras de processo. É uma *estrutura* de processos na qual formular as definições de processo. Tais definições de processo podem variar de muito formais a muito simples.

A maneira de utilizar a modelagem UML pode variar também de muito formal a muito informal. Você pode optar por tratar seus modelos como desenhos de arquitetura formais que deverão ser cuidadosamente seguidos durante a construção. Ou poderá tratá-los como rascunhos que sugerem os esboços amplos de um design, mas que são considerados descartáveis depois que o projeto é enviado para implementação. O RS*x* pode auxiliar você no final de cada um desses espectros de modelagem e processo. As diretrizes neste documento não são obstáculo ao seu modo de pensar. Elas são um meio de ajudar a entender como utilizar os recursos do RS*x* para facilitar um processo que pareça mais adequado a você.

Observe que o RS*x* possibilita utilizar modelos não apenas como cópias de projetos, mas como especificações a partir das quais partes significativas de uma implementação podem ser geradas

automaticamente. Isso é realizado com o uso de transformações de modelo para modelo e de modelo para código. O uso de transformações para a prática de MDD (Model-Driven Development) introduz algumas considerações especiais relativas às estruturas de modelo. Se você for utilizar modelos e transformações para a prática de Model-Driven Development, procure também recursos específicos do RS*x* MDD na área de design e construção Rational de developerWorks®.

Escopo

Este documento descreve como representar os artefatos de modelo RUP no RS*x*. Além disso, oferece diretrizes para estruturas organizacionais internas desses artefatos. Ele *não* tenta

- reformular os suportes conceituais dos artefatos de modelo RUP, nem fornecer descrições amplas sobre eles
- descrever o processo ou as técnicas para especificar o conteúdo semântico ou diagramático detalhado dos artefatos RUP associados.

Para obter informações de ferramenta neutra sobre como definir, desenvolver e modelar o conteúdo dos artefatos RUP, consulte o RUP.

Para obter informações sobre técnicas de ferramenta específica para desenvolver o conteúdo de modelos RS*x*, consulte

- a documentação do produto (tutoriais, amostras, ajuda on-line)
- os mentores de ferramentas nas configurações do RUP dos quais este whitepaper faz parte
- recursos relacionados ao RS*x* no developerWorks

Convenções Tipográficas

As discussões de interesse dos usuários do RS*x* que estão migrando do IBM Rational Rose ou XDE são apresentadas na forma de barra lateral, em uma caixa de texto com borda e plano de fundo cinza-claro:

> ***XDE/Rose***
> Discussão de interesse dos usuários anteriores do XDE ou do Rose.

Organização do Documento

Uma seção Conceitos Básicos e Terminologia estabelecerá um vocabulário útil e fornecerá algumas informações gerais sobre como os modelos são implementados nos produtos RS*x*.

Em seguida, a seção Mapeamento de Modelo RUP para Modelo RS*x* discutirá como o RS*x* suporta os tipos de modelo definidos pelo RUP.

Em seguida, estão as diversas seções que fornecem orientação para estruturar modelos de vários tipos. Algumas dessas seções discutem as diferentes formas de uso dos modelos, de acordo com o grau de rigor desejado em termos de processo, abordagem de modelagem e controle de arquitetura.

Por último, há uma discussão dos problemas associados ao uso dos modelos em equipes. Ela trata das estratégias de gerenciamento de escala e ativação de compartilhamento de modelos entre membros de equipe para minimizar a contenção e a mesclagem de arquivos.

2. Conceitos Básicos e Terminologia

Modelos

No RUP, um modelo é definido como "uma especificação completa de um domínio de problema ou solução a partir de uma determinada perspectiva". Um domínio de problema ou um sistema pode ser especificado por vários modelos que representam perspectivas diferentes no domínio ou no sistema. O RUP propõe um conjunto específico de modelos:

- Modelo de Caso de Uso de Negócios
- Modelo de Análise de Negócio
- Modelo de Caso de Uso
- Modelo de Análise (pode estar incluído no Modelo de Design)
- Modelo de Design
- Modelo de Implementação
- Modelo de Implantação
- Modelo de Dados

O RUP é uma ferramenta agnóstica. No que diz respeito ao RUP, um modelo poderia ser um desenho em um guardanapo ou em um quadro branco, algo em uma ferramenta de modelagem ou mesmo uma imagem mental. Da perspectiva do RUP, modelo é um conceito lógico.

No contexto do RS*x*, podemos discutir sobre os modelos em termos lógicos, mas podemos também discutir a respeito deles em termos físicos. Suponha que você tenha equipes trabalhando em dois aplicativos: uma equipe de três analistas trabalhando em um aplicativo de gerenciamento de planilha de hora e uma segunda equipe de cinco analistas trabalhando em um aplicativo de central de atendimento. Ambas as equipes estão trabalhando no momento na captura de requisitos e utilizando o RS*x* para modelagem de caso de uso. Relativamente ao RUP, você diria que uma equipe está construindo "o modelo de caso de uso para o aplicativo de planilha de hora" e a outra está construindo "o modelo de caso de uso para o aplicativo de central de atendimento". Mas se as equipes estiverem utilizando o RS*x*, é importante reconhecer que os modelos delas terão manifestações físicas. Esse é o assunto da próxima seção.

Arquivos de Modelagem

Modelos RS*x* são persistidos como arquivos. (Na terminologia do Eclipse, um arquivo é considerado um 'recurso'[1]; assim, se você encontrar o termo 'recurso de modelagem' neste documento ou em outras origens, significa o mesmo que 'arquivo de modelagem'). No sentido mais amplo, o RS*x* suporta dois tipos de arquivos de modelagem:

- **Arquivos de modelagem de pré-implementação** são armazenados como arquivos individuais no sistema de arquivo do OS host. Eles têm extensões de nome de arquivo ".emx". Esses arquivos contêm
    - elementos semânticos UML (classes, atividades, relacionamentos, …)
    - diagramas UML que representam os elementos semânticos UML (e *podem* também representar referências visuais a algo em outros domínios semânticos, como Java, C++ ou DDL).

- **Arquivos de modelagem de implementação** são armazenados como *projetos* Eclipse, em um espaço de trabalho do Eclipse. Os projetos contêm
    - artefatos de implementação (código-fonte Java, páginas da Web, arquivos de metadados baseados em XML, …)

[1] No Eclipse, um recurso é um arquivo, mas tem também comportamento e propriedades adicionais no ambiente Eclipse. Os Arquivos de Modelagem descritos aqui são tratados como 'recursos' pelo Eclipse.

- arquivos de diagrama que representam e refletem diretamente os artefatos de implementação.

A semântica de modelo reside nos próprios artefatos de implementação. Por exemplo, o modelo semântico de uma implementação Java é serializado e armazenado como uma coleção de arquivos de código-fonte Java. Cada diagrama reside em seu próprio arquivo físico no projeto. Arquivos de diagrama podem ter várias extensões. A mais comum é ".dnx". Os diagramas de modelagem de implementação muitas vezes utilizam notação UML, mas podem utilizar também outras notações (por exemplo, IDEF1X ou Information Engineering para visualização de dados, ou notações de propriedade da IBM utilizadas no design de camadas da Web).

O foco deste documento é como organizar as estruturas internas dos modelos de "pré-implementação". **No restante do documento, o termo "arquivo de modelagem" é reservado aos arquivos de modelagem de "pré-implementação"** (arquivos com extensões .emx). A orientação para organizar o conteúdo de projetos de implementação pode ser encontrada em outras fontes, como a ajuda on-line do Rational Software Architect, do Rational Application Developer e do Rational Web Developer Community Edition.

Um arquivo de modelagem não necessariamente contém todas as informações de um modelo (lógico). De fato, um arquivo de modelagem muitas vezes conterá apenas um subconjunto de um modelo. No exemplo fornecido anteriormente, tínhamos uma equipe de três trabalhando em um modelo de caso de uso para um aplicativo de planilha de hora. Essa equipe pode optar por particionar fisicamente seu modelo de caso de uso em três arquivos de modelagem, para que cada membro da equipe possa trabalhar em um subconjunto diferente dos casos de uso sem disputar o mesmo arquivo. A seção final deste documento discute os problemas associados à partição de modelos e ao gerenciamento de arquivos de modelagem.

Tipos de Modelo

No RUP, modelos são de tipos específicos, como modelo de caso de uso, modelo de análise ou modelo de dados. No RS*x*, arquivos de modelagem não são "rigidamente tipificados", mas você pode seguir uma convenção de uso de vários arquivos de modelagem que são "tipificados livremente". Se você quiser seguir essa convenção, poderá estabelecer tipografia livre de duas maneiras:

- comece com um arquivo de modelagem "Em Branco" (veja a seguir) e estabeleça seu tipo conforme você o nomeia e pelo tipo de conteúdo nele colocado (incluindo quais perfis UML aplicar a ele)
- crie um arquivo de modelagem no qual basear um "modelo de gabarito" predefinido que representa um determinado tipo de modelo. Os produtos RSx fornecem um conjunto padrão de modelos de gabarito para os tipos de modelo descritos neste documento. Você também pode criar seus próprios modelos de gabarito (consulte a Ajuda e os fóruns do produto, bem como outros recursos no developerWorks).

De qualquer forma, o "tipo" de um arquivo de modelagem no RSx é de fato apenas uma questão de convenção relativa à nomenclatura e ao conteúdo do arquivo. Por exemplo, a ferramenta não impedirá um arquivo que contém um modelo de caso de uso de conter também as classes que constatam os casos de uso (o que em termos lógicos, o RUP consideraria ser parte do modelo de análise ou design).

Essas diretrizes na verdade sugerem que você trate os arquivos de modelagem do RSx como sendo de determinado tipo.

Espaços de Trabalho, Projetos e Tipos de Projeto

Os leitores familiarizados com o Eclipse, com produtos WebSphere Studio ou com o Rational Application Developer já saberão que arquivos residem em Projetos, que Projetos podem ser de vários tipos, além

de serem agrupados e gerenciados em Espaços de Trabalho. Os arquivos de modelagem do RS*x* residem em projetos assim como outros arquivos.

Nesta discussão, nem todos os tipos de projeto disponíveis no RSx e no Rational Application Developer precisam ser explicados detalhadamente. Nós estamos interessados em primeiro lugar em duas categorias de projetos:

- **Projetos UML**
- **Projetos de implementação**, que incluem os tipos de projeto especializados, como Projeto Corporativo, Projeto EJB, Projeto da Web e Projeto C++

Afirmamos anteriormente que o RS*x* suporta dois tipos de arquivos de modelagem:

- Arquivos com extensões .emx que contêm modelos UML e são utilizados para modelagem em níveis de abstração acima da implementação (requisitos, análise, design)
- Projetos Eclipse que contêm a semântica de implementação (normalmente serializados como artefatos de arquivo de código-fonte) e diagramas que refletem tal semântica.

A regra para alocação de modelos para projetos é simples:

a) colocar arquivos de modelagem de "pré-implementação" em projetos UML

b) modelos de implementação estão sob seus próprios cuidados, visto que, em essência,:

[modelo de implementação] = [projeto de implementação]

Há algumas exceções que compõem essa regra. Os arquivos de modelagem UML a seguir são candidatos a posicionamento em um projeto de implementação de linguagem específica:

- 'Modelos de esboço' de design (que serão discutidos em uma seção posterior).
- Modelos com diagramas de seqüência que descrevem os testes que serão executados em comparação com o código no projeto

XDE/Rose

As teorias de operação Rose e XDE incluem a prática de refinar de modo iterativo o Modelo de Design até que você alcance um nível de abstração equivalente ao código e, em seguida, utilizar a tecnologia de sincronização de modelo de código para manter a semântica desse modelo de acordo com o próprio código. Assim, por exemplo, no XDE, os modelos de implementação existem não apenas como código e diagramas em projetos, mas também como arquivos de 'modelo de código' que permanecem independentemente dos artefatos de implementação e, em essência, representam uma cópia redundante de sua semântica.

A teoria de operações do RSx aconselha o uso de modelos de plataforma neutra em níveis de abstração superiores ao código (isto é, modelos de design, como um Modelo de Design de TI Corporativo) e o uso de transformações para gerar o código a partir desses modelos. Assim, no nível de código de abstração, RSA, RSD e RAD de maneira simples permitem desenhar diagramas da semântica de código expressos em notações UML, dispensando a abordagem de uso de um modelo semântico persistido separadamente no nível de implementação da abstração.

Observe que o RSx não *impede* você de definir modelos UML em um nível de código de abstração e gerar o código a partir deles, e realmente esse tipo de uso é esperado. Mas o RSx não fornece a tecnologia para manter esses modelos auto-sincronizados com o código.

Conceitos em Revisão

As ilustrações a seguir resumem as discussões precedentes. Elas refletem o cenário descrito

anteriormente, em que temos duas equipes, uma trabalhando em um aplicativo de centro de chamada e outra trabalhando em um aplicativo de gerenciamento de folha de horários.

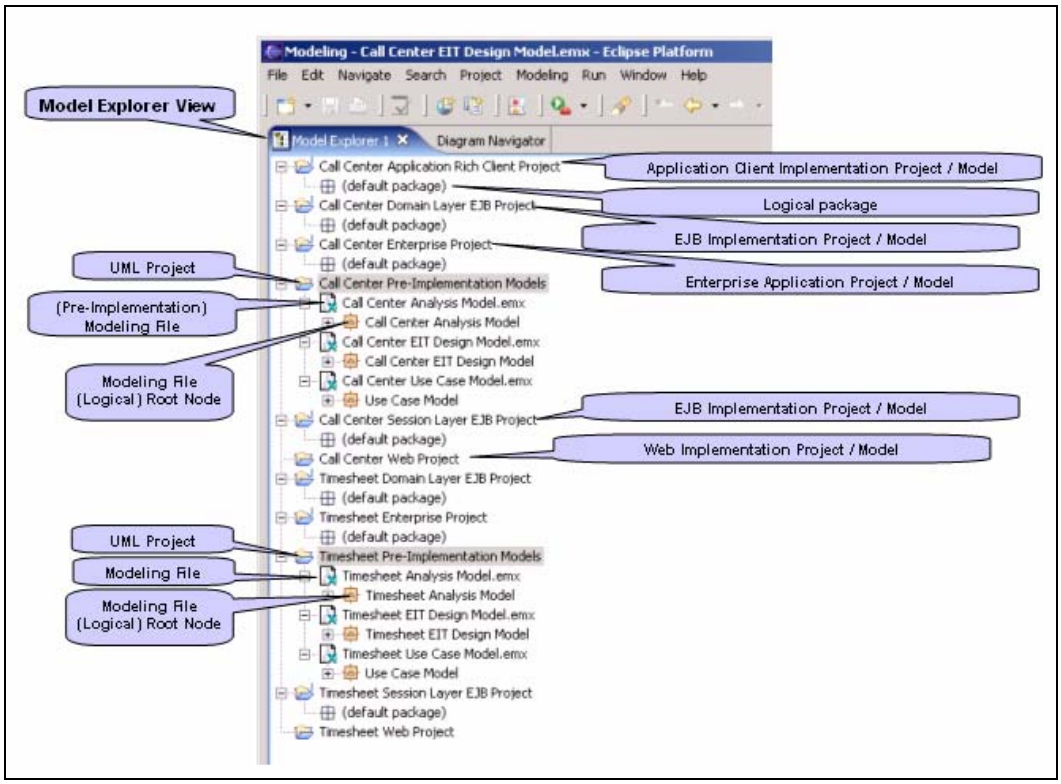


Figura 2-1

O RSX fornece uma visualização[2] do explorador que oferece uma visualização física e lógica combinada de modelos. No explorador, você vê os projetos no espaço de trabalho representados como nós de nível superior e, em cada projeto, vê os recursos que pertencem a esse projeto. Assim, na **Figura 2-1** vemos representada no Model Explorer uma coleção de projetos que correspondem aos dois aplicativos de nosso cenário. Vemos que projetos UML foram utilizados para os modelos de pré-implementação e também que uma coleção de projetos de tipos de solução apropriada foram utilizados para os modelos de implementação.

> ***XDE/Rose***
> Ao contrário do Model Explorer do RSA, os explorers de modelos no Rose e no XDE forneceram apenas uma visualização lógica dos modelos. Observe que a visualização de recursos fornecidos pelo Model Explorer do RSA não é a visualização física 'pura' fornecida pela visualização Navigator do Eclipse. Enquanto alguns recursos físicos são visíveis no Model Explorer, eles são, na maioria das vezes, representados por ícones que indicam visualizações *lógicas* dos recursos.

[2] Nas versões 6.x, esse era o "Model Explorer". A partir da versão 7.0, os modelos aparecem no explorador de propósitos gerais. A ilustração neste documento se baseia na versão 6.0 e representa o "Model Explorer".

Na figura 1-2, mostramos como o modelo de caso de uso de Planilha de Hora pode ser organizado internamente em pacotes que representam alguns subconjuntos funcionalmente coesos do domínio de problema.

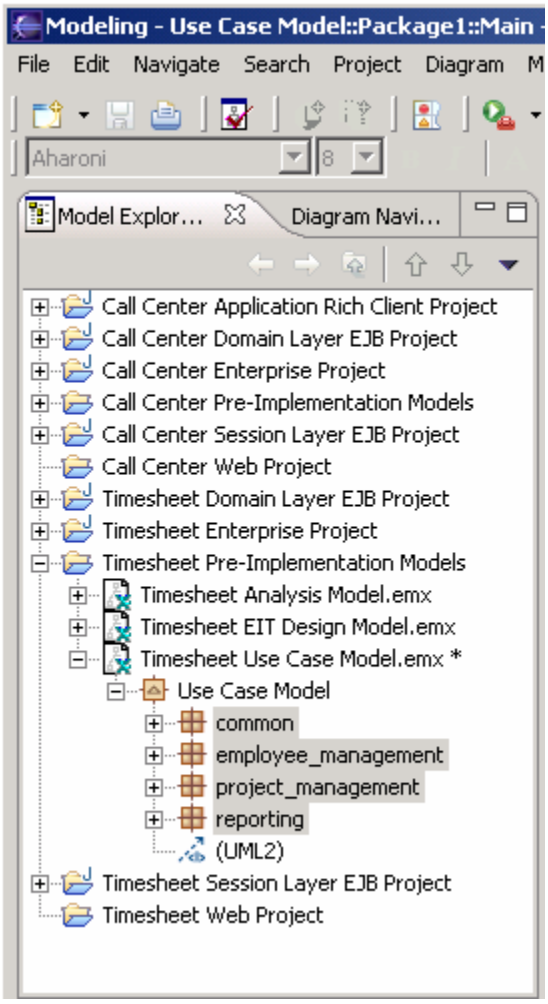


Figura 2-2

Nas **Figura 2-1 e 1-2**, cada modelo de pré-implementação reside em um único arquivo de modelagem. Na figura 1-3, mostramos como o modelo de caso de uso de Planilha de Hora pode ser refatorado em vários arquivos de modelagem que correspondem aos mesmos subconjuntos do domínio de problema. Observe como a raiz de cada arquivo de modelagem foi nomeada para manter um espaço de nomes consistente em todos os arquivos de modelagem que compõem o modelo de caso de uso completo.

3. Mapeamento de Modelo RUP para Modelo RS*x*

A tabela a seguir mostra como os modelos RUP mais comumente usados podem ser mapeados, por convenção, para "tipos" de arquivos de modelagem RS*x*. O mapeamento geralmente é direto, mas é a chave para utilizar este documento como um guia para a prática do RUP com o RSx. Os tipos de arquivo de modelagem RSx mencionados na tabela são discutidos imediatamente após a tabela. As diretrizes para a organização interna dos diversos tipos de arquivo de modelagem, e em quais tipos de projetos mantê-los, são fornecidas nas seções posteriores. Essas discussões posteriores são apresentadas nos termos dos tipos de arquivo de modelagem RSx listados aqui.

Modelo RUP	**Tipo de Arquivo de Modelagem *RSx***
Modelo de Caso de Uso	Arquivo de modelagem com base no gabarito "Modelo de Caso de Uso" (alternativa: comece com o arquivo de modelagem em branco, restrinja o conteúdo por diretriz de modelo de caso de uso RUP)
Modelo de Análise	Modelo de Análise (alternativa: comece com o arquivo de modelagem em branco, restrinja o conteúdo por diretriz de modelo de análise RUP) (alternativa: utilize os pacotes de «análise» no Modelo de Design)
Modelo de Design	Para aplicativos de negócios de n camadas: arquivo de modelagem com base no gabarito "Modelo de Design de TI Corporativo" (alternativa: comece com o arquivo de modelagem em branco, restrinja o conteúdo por diretriz de modelo de design RUP) Para outros tipos de aplicativos: comece com o arquivo de modelagem em branco, restrinja o conteúdo por diretriz de modelo de design RUP Para 'esboço' de design: arquivo de modelagem em branco Suplemento opcional: arquivo de modelagem em branco adicional utilizado como Modelo de Visão Geral de Implementação
Modelo de Implementação	Projetos do Eclipse contendo artefatos de implementação e arquivos de diagrama
Modelo de Implantação	Comece com o arquivo de modelagem em branco, restrinja o conteúdo por diretriz de modelo de implantação RUP

Tipos de Arquivo de Modelagem RS*x*

Arquivo de Modelagem em Branco

O RSx fornece a opção de criar um "Modelo Em Branco" (Arquivo→Novo→Modelo UML→Modelo Em Branco). Um "Modelo Em Branco" é um arquivo de modelagem que não se baseia em um gabarito de

modelo. Ele não tem perfis especiais aplicados e nenhum conteúdo padrão que não seja um único diagrama “Principal” (formato livre). **Você pode utilizar um arquivo de modelagem em branco como ponto de partida para qualquer tipo de modelo**. Escolhendo como denominá-lo, que conteúdo definir nele e quais perfis aplicar a ele, você poderá utilizar um arquivo de modelagem em branco para criar um modelo de caso de uso, um modelo de análise, um modelo de design, um modelo de implantação e qualquer outro tipo de modelo RUP.

Arquivo de Modelagem de Caso de Uso

O RSx fornece a opção de criar um arquivo de “Modelo de Caso de Uso” com base em um gabarito de modelo. O gabarito contribui com o conteúdo padrão conforme representado na ***Figura 3-1***. (Está fora do escopo deste documento explicar como o conteúdo do “bloco de construção” e as cadeias de procura são utilizados. Os gabaritos contêm instruções auto-explicativas.)

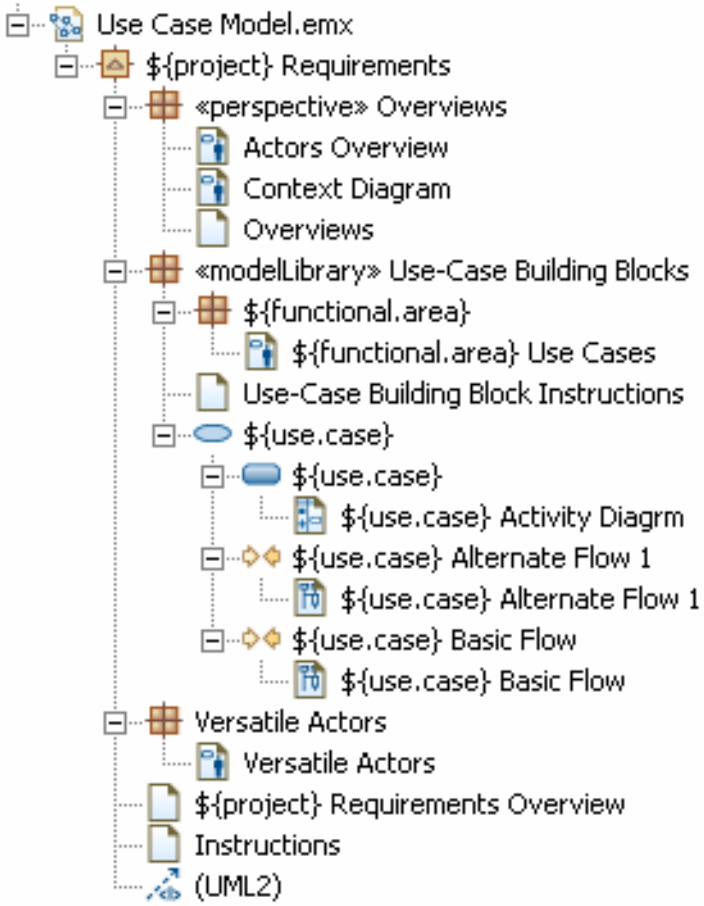


Figura 3-1

Arquivo de Modelagem de Análise

O RSx fornece a opção de criar um arquivo de “Modelo de Análise” com base em um gabarito de modelo. O gabarito contribui com o conteúdo padrão conforme representado na ***Figura 3-2***. Além disso, um perfil de “Análise” é aplicado aos arquivos de modelo criados a partir desse gabarito:

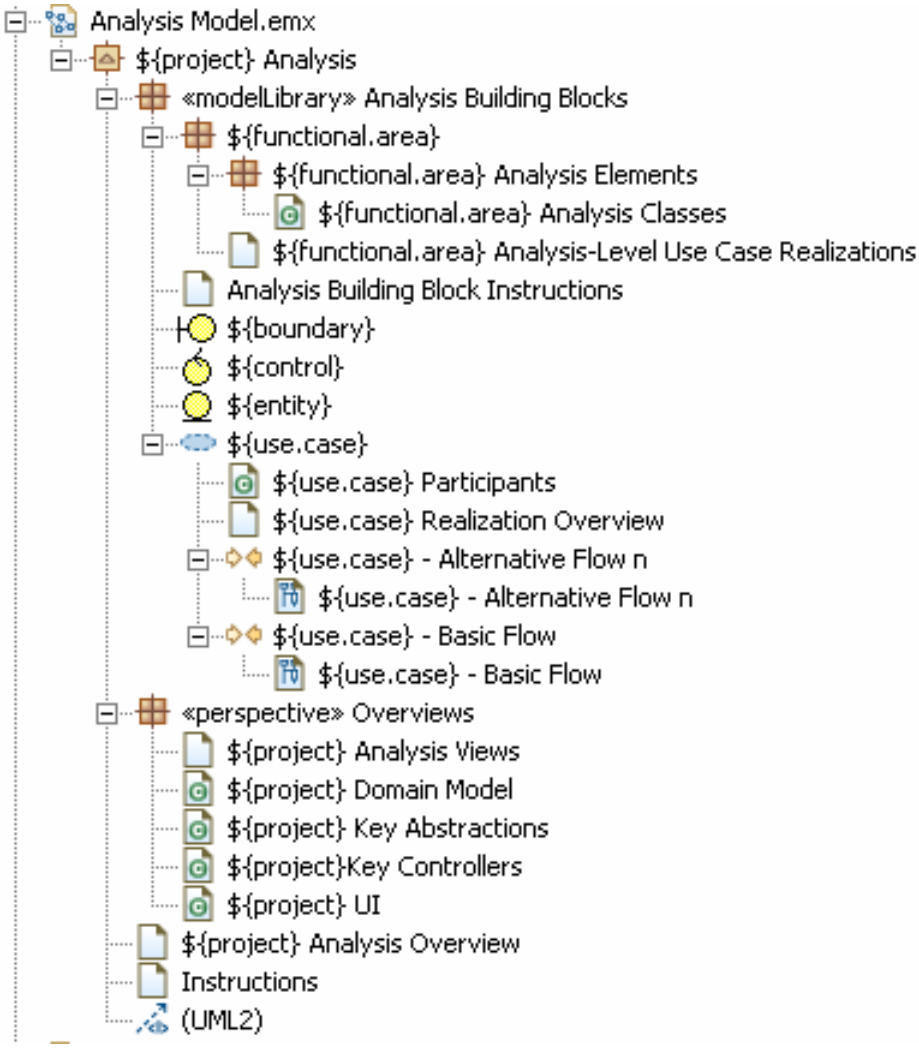


Figura 3-2

Arquivo de Modelagem de Design de TI Corporativo

O RSx fornece a opção de criar um arquivo de “Modelo de Design de TI Corporativo” (EITDM) com base em um gabarito de modelo. O gabarito fornece o conteúdo padrão conforme representado na **Figura 3-3** . Além disso, um perfil de “Transformação EJB” [3] será aplicado aos arquivos de modelo criados a partir desse gabarito. Esse é o gabarito apropriado para uso em design (e opcionalmente para análise) ao direcionar aplicativos de negócios e utilizar transformações de geração de código *RSX* para suportar a criação de tais aplicativos.

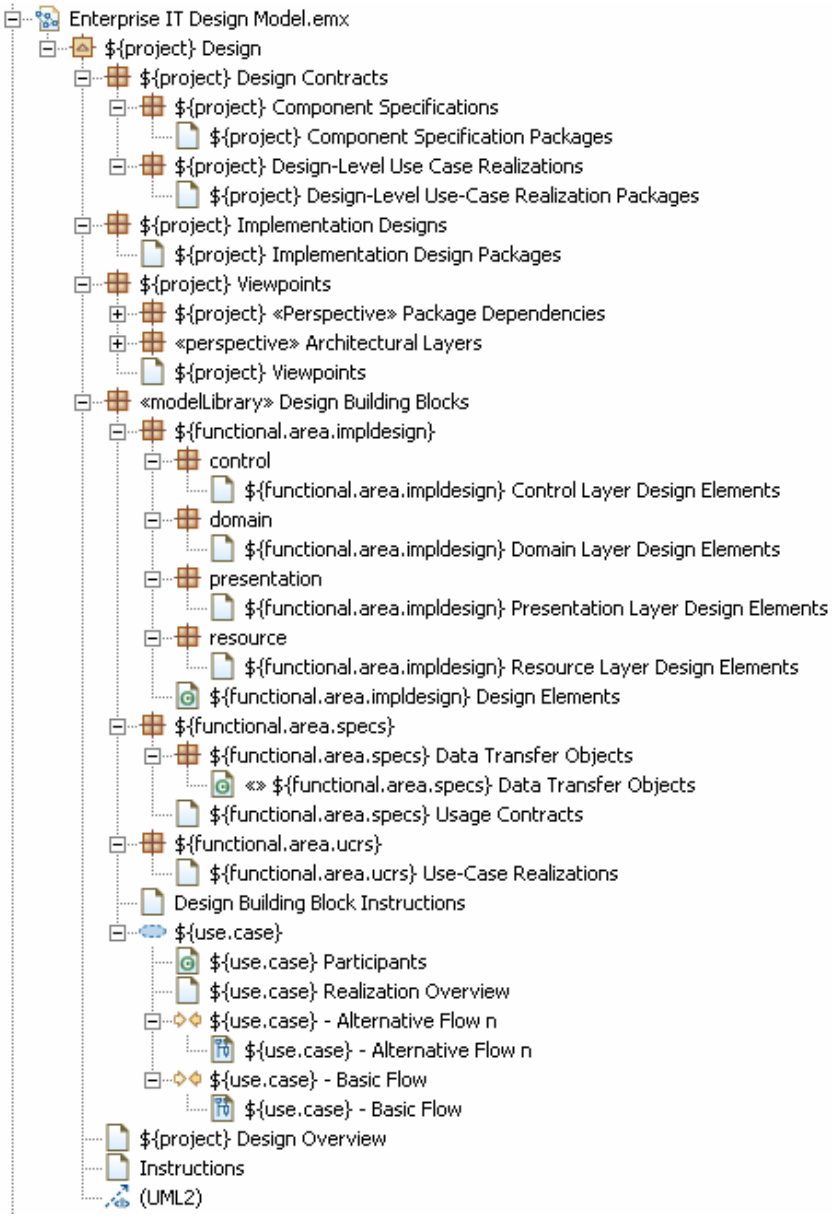


Figura 3-3

[3] O conjunto de perfis de ativação de transformação fornecido como parte do gabarito de Modelo de Design EIT provavelmente evoluirá à medida que atualizações são liberadas para o produto.

Arquivo de Modelagem de Visão Geral de Implementação

Como parte de seu modelo de design, além disso, você talvez ache útil definir um "Modelo de Visão Geral de Implementação" para capturar uma visualização de alto nível de como a implementação será organizada. O "Modelo de Visão Geral de Implementação" seria utilizado anteriormente na fase de design – antes de o código ter sido gerado ou gravado -- para representar projetos e pastas/pacotes reais do RSx ou do Rational Application Developer nos quais você espera que o código e os arquivos relacionados (metadados, descritores de implantação, etc.) residam. Você também poderia utilizar esse modelo para mostrar as dependências previstas entre esses projetos e pacotes, que podem ser úteis na identificação dos requisitos de build do sistema. O Modelo de Visão Geral da Implementação também pode ser um local para manter diagramas conceituais informais da arquitetura da solução.

Modelo de Implementação

Conforme indicado anteriormente, no RS*x*, um modelo de implementação consiste em um projeto contendo os artefatos de implementação e (opcionalmente) os diagramas que representam esses artefatos[4].

Modelos de "Esboço"

Conforme observado na seção "Conceitos Básicos e Terminologia", você pode tratar modelos de design como desenhos formais de arquitetura que são mantidos durante a existência do sistema e utilizados para suportar/impor o controle de arquitetura. Ou, pode tratá-los como esboços que servem para sugerir um design e ajudar a esclarecê-lo e comunicá-lo, mas que são descartáveis uma vez iniciada a implementação. O RSx suporta as duas abordagens. Seus recursos geralmente não têm como objetivo uma abordagem ou outra, mas as opções que você faz sobre como utilizar modelos de design certamente ajudam a determinar quais recursos do RS*x* utilizar e como utilizá-los. Quando a distinção for importante no contexto das diretrizes apresentadas neste documento, o termo "modelo de esboço" será utilizado para indicar que um modelo está sendo utilizado da maneira mais 'descartável'.

[4] Para criar esses diagramas, em vez de utilizar Arquivo→Novo→Modelo UML para criar um modelo, você utiliza Arquivo→Novo→Diagrama de Classe para criar um diagrama no qual possa compor 'visualizações' de código em notação UML (ou outra). Cada diagrama individual é persistido como um arquivo separado, com uma extensão .dnx, e pode ter versão controlada quase do mesmo modo que um arquivo de código. Esses diagramas não contêm informações semânticas, apenas notação. Todas as informações semânticas relevantes residem no próprio código. Quando você altera algo como um nome de classe ou uma assinatura de operação em um desses diagramas, está de fato alterando o próprio código de base. Ao fazer tais alterações no código (utilizando um editor de texto), os diagramas nos quais o código alterado aparece são atualizados automaticamente.

4. Diretrizes e Técnicas Gerais para Organização de Estruturas Internas de Modelos

A principal ferramenta para organizar o conteúdo de modelos UML é o pacote. Pacotes UML são úteis a dois propósitos principais:

- particionar, organizar e identificar informações de modelos
    - agrupando elementos que correspondem a um assunto específico no domínio de problema ou solução
    - separando diferentes tipos de informações de modelo, como interfaces, implementações, diagramas, etc.
    - agrupando elementos para definir e controlar suas dependências de outros elementos
    - agrupando diagramas que fornecem visualizações alternativas no mesmo modelo
- estabelecer espaços de nomes
    - para elementos de modelo
    - para artefatos de implementação gerados a partir de elementos de modelo (isso pode envolver mapeamentos entre espaços de nomes de idioma do modelo e da implementação)
    - para uma unidade de reutilização

Tradicionalmente, o RUP propõe estratégias específicas de pacote para vários tipos de modelo. Essas estratégias são refletidas nas seções específicas do tipo de modelo deste white paper. O RSx também apresenta algumas ferramentas adicionais de organização descritas aqui:

Representar Pontos de Vista Utilizando Pacotes de «perspectivas»

Nos casos em que é desejável ver os elementos organizados em mais de uma maneira, você pode criar pacotes adicionais com diagramas que representam os esquemas organizacionais alternativos. Essa mesma técnica pode ser útil sempre que existe a necessidade de representar uma determinada visualização no conteúdo do modelo que recorta o esquema de pacotes do modelo. O RSx suporta essa técnica oferecendo um estereótipo de pacote de «perspectivas» como parte de seu 'perfil básico' UML . Você pode considerar um pacote de «perspectivas» em geral como equivalente a um "Ponto de Vista" RUP para Engenharia de Sistemas ou IEEE 1471- 2000.

Não coloque elementos semânticos (classes, pacotes, associações, etc.) nos pacotes de «perspectivas». Coloque neles apenas diagramas que representem visualizações com base no ponto de vista do aplicativo ou de interesse organizacional alternativo. A aplicação do estereótipo de «perspectiva» a um pacote tem muitos efeitos. Identifica visualmente esse pacote como representante de um determinado ponto de vista. Suporta também uma regra de validação de modelo que avisa quando elementos semânticos são colocados em um pacote de «perspectivas». É útil também como designador de pacotes que devem ser ignorados pelas transformações do RSx

Criar Representações de Atualização Automática de Interesses Específicos Utilizando Diagramas de Tópico

Ao contrário de diagramas 'normais' nos quais você coloca manualmente os elementos que deseja representar, o conteúdo de um Diagrama de Tópico é determinado por uma consulta executada no conteúdo do modelo existente. Para criar um Diagrama de Tópico, selecione um elemento de modelo 'de tópico' e, em seguida, defina quais outros elementos você deseja que apareçam no diagrama com base nos tipos de relacionamentos que eles têm com o elemento de tópico. Conforme ocorrem alterações no conteúdo semântico do modelo, os Diagramas de Tópico são ajustados adequadamente.

Examinar Modelos por meio de Diagramas de Navegação

Diagramas de Navegação não são *especificamente* uma ferramenta de organização de modelos. Sua finalidade é facilitar a descoberta e o entendimento do conteúdo do modelo sem ter que compor

manualmente os diagramas. Mas, no contexto da organização de modelos, é bom estar atento a eles, uma vez que podem diminuir a necessidade de compor diagramas permanentes. Isso, por sua vez, poderia reduzir o tamanho e a complexidade dos modelos, deixando-os mais fáceis de organizar.

Diagramas de navegação são um pouco parecidos com diagramas de tópico, mas a diferença principal é que os Diagramas de Navegação nunca permanecem, são sempre gerados dinamicamente. Para produzir um Diagrama de Navegação, selecione um elemento de modelo (em um diagrama ou no Model Explorer) e utilize o menu de contexto para "Explorar no Diagrama de Navegação". Isso produzirá um diagrama representando o elemento selecionado como o 'ponto focal' com elementos relacionados apresentados em um layout radial em torno do ponto focal. É claro que, em seguida, você pode selecionar um dos elementos relacionados nesse Diagrama de Navegação e torná-lo o ponto focal de outro Diagrama de Navegação, e continuar dessa maneira, desde que deseje.

Navegação entre os Diagramas

No RSx, há dois mecanismos de navegação entre diagramas:

- É possível arrastar um nó de diagrama do Model Explorer para algum outro diagrama do 'host'. Em seguida, você pode dar um clique duplo no ícone resultante, no diagrama do host, para abrir o referido diagrama
- Sempre que você criar um novo pacote UML em um modelo, um diagrama (de formato livre) "Principal" será criado automaticamente. Por padrão, esse diagrama "Principal" é criado como o diagrama 'padrão' do pacote. Você pode renomeá-lo para algo diferente de "Principal" e ele ainda será tratado como 'padrão'. Também é possível selecionar um diagrama diferente no pacote e torná-lo o diagrama 'padrão' para esse pacote. O propósito do diagrama 'padrão' é este: se você colocar o próprio pacote em outro diagrama do 'host', poderá dar um clique duplo no pacote, o que abrirá seu diagrama padrão.

Esses mecanismos suportam a seguinte **diretriz** organizacional, que pode se aplicar a modelos de qualquer tipo:

1. Compor o diagrama Principal (ou outro diagrama padrão) de cada arquivo de modelagem para representar
    a. cada pacote de nível superior no arquivo de modelagem
    b. os ícones de quaisquer outros diagramas que residam no pacote raiz do arquivo de modelagem (ou seja, não representar o ícone do próprio diagrama padrão)
2. Compor o diagrama Principal (ou outro diagrama padrão) de cada pacote de nível superior para representar
    a. os pacotes contidos diretamente
    b. os ícones do diagrama de quaisquer outros diagramas diretamente contidos
3. Repetir esse padrão para cada nível inferior de pacotes sucessivamente

5. Diretrizes de Organização Interna de Modelo de Caso de Uso

Organização de Alto Nível do Modelo de Caso de Uso

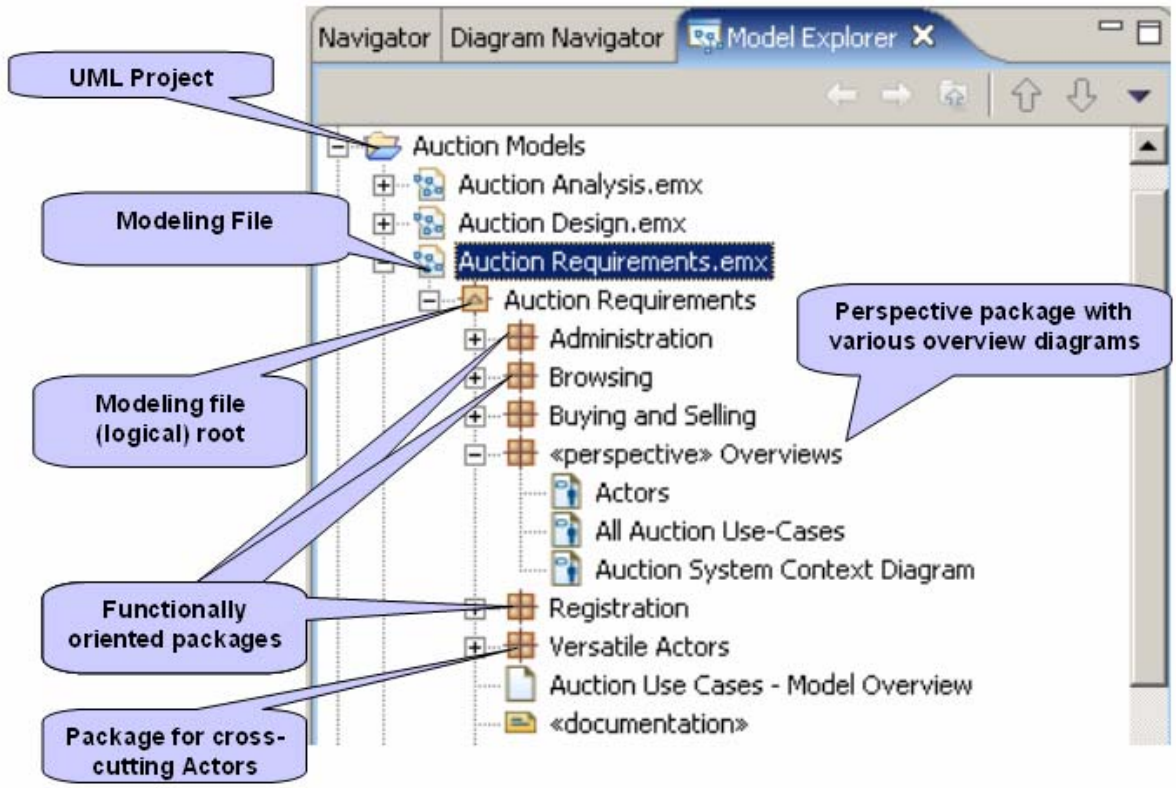


Figura 5-1

Figura **5-1** anterior ilustra as seguintes diretrizes para estruturar os modelos de caso de uso:

1. Utilizar pacotes de nível superior para estabelecer agrupamentos orientados funcionalmente. Análise racional:

    - Isso geralmente é bem mapeado para os interesses da divisão de trabalho quando uma equipe de pessoas for trabalhar no Modelo de Caso de Uso. E deixa você bem preparado para o caso de mais tarde ter de decidir dividir o modelo de caso de uso em vários arquivos de modelagem, devido à contenção de arquivo ter se tornado um problema (basta criar um arquivo de modelagem separado, por pacote de nível superior).

    - Comparada com outras abordagens organizacionais, geralmente o mapeamento desta será melhor para a organização da implementação eventual. Isso será importante se você for utilizar transformações para iniciar cada nível inferior sucessivo de abstração. Especificamente, se você for gerar conteúdo inicial em um modelo de análise com base no modelo de caso de uso, desejará que a estrutura de pacote do modelo de caso de uso seja bem mapeada para a estrutura de pacote desejada do modelo de análise de destino. Por sua vez, você desejará que a estrutura de pacote do modelo de análise seja bem mapeada para o modelo de design, e a

estrutura de pacote do modelo de design seja bem mapeada para o conjunto de projetos que consistirá na implementação. Quanto mais simples esses mapeamentos, menos trabalho será exigido para configurar as transformações de um nível de abstração para o outro.

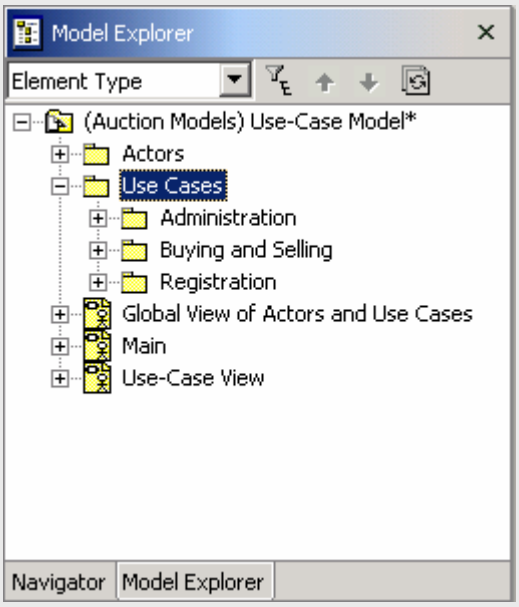
2. Utilizar outro pacote de nível superior para capturar Agentes ‘amplamente capacitados’ ou ‘versáteis’.

3. Utilizar diagramas em pacotes de «perspectiva» para capturar visualizações de corte cruzado e alto nível dos Casos de Uso. Análise racional:

   o Fornecer visualizações de recorte cruzado e visualizações de casos de uso ‘significativo do ponto de vista da arquitetura’ enquanto se mantém os elementos da semântica do modelo organizados em agrupamentos orientados funcionalmente.

XDE/Rose

A orientação do RSX revisa um pouco a orientação tradicional para organização de alto nível do modelo de caso de uso, que era criar um pacote para Agentes e outro para Casos de Uso. Assim, conforme o tamanho e a complexidade do modelo exigia, você utilizaria pacotes de nível inferior para estabelecer agrupamentos orientados funcionalmente, conforme mostrado neste exemplo baseado em XDE:



Conteúdo do Modelo de Caso de Uso

Está fora do escopo deste documento servir de tutorial detalhado sobre como gravar bons casos de uso ou os prós e contras da boa modelagem de casos de uso. Entretanto, aqui está uma breve discussão do que poderia ser incluído em um modelo de caso de uso, além dos Agentes e dos Casos de Uso.

- **Recomendado:** crie um diagrama ‘principal’ na raiz do modelo, que representa os outros pacotes do modelo e suporta pesquisa detalhada nesses pacotes e em seus respectivos diagramas ‘principais’

- **Recomendado:** em cada pacote de caso de uso, inclua um diagrama que representa os casos de uso do pacote, todos os relacionamentos entre eles, bem como os Agentes participantes. (Talvez seja apropriado utilizar mais de um diagrama se o número de casos for grande.)

- **Recomendado:** descreva os fluxos principal e alternativo de cada caso de uso em seu campo Documentação[5] (veja a **Figura 5-2**)

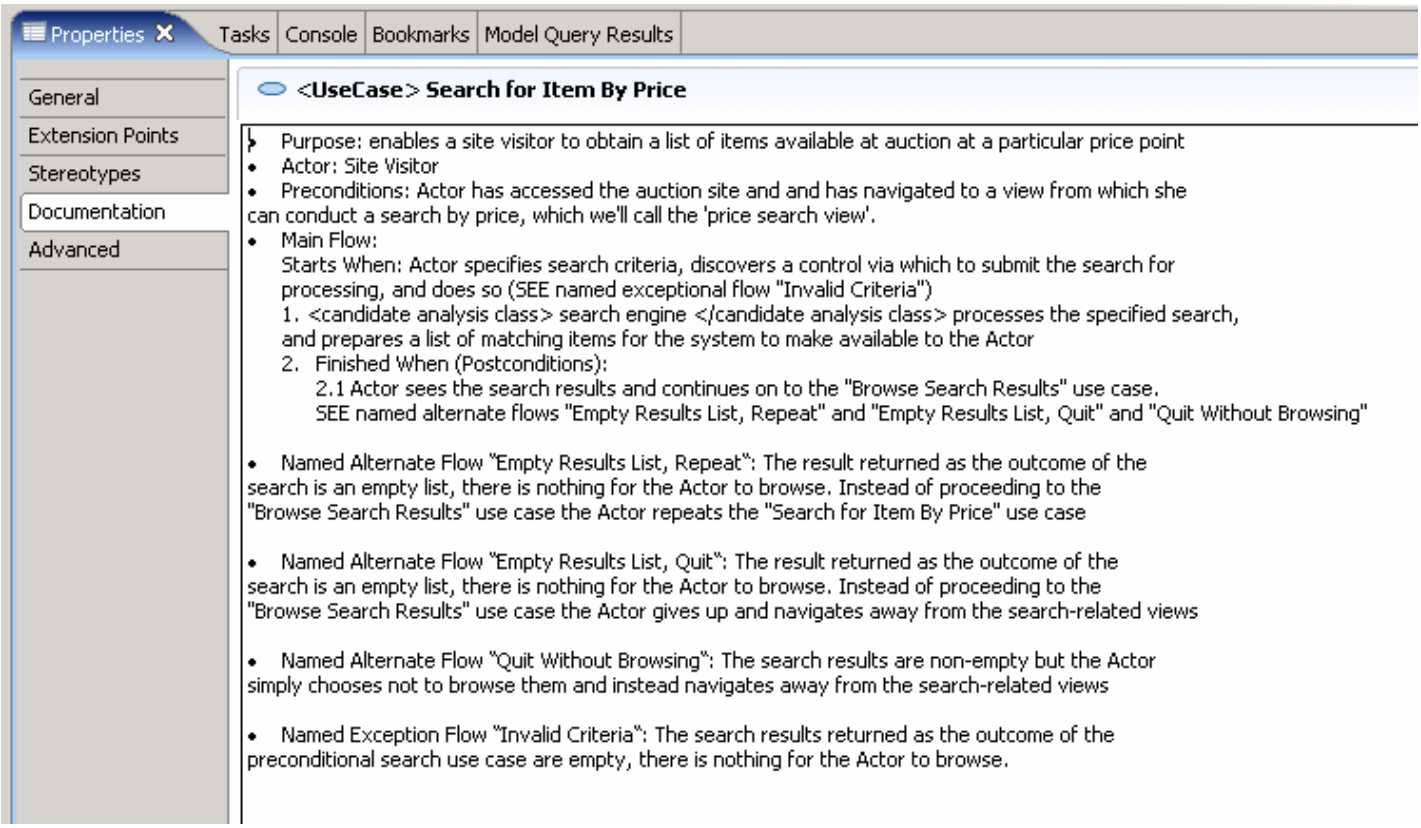


Figura 5-2

- **Opcional:** quando a complexidade de um caso de uso o garante, inclua um diagrama de Atividade e componha-o para refletir os fluxos de atividade geral do caso de uso. (Consulte a **Figura 5-3**) **Análise racional:** isso ajuda a mostrar as condições que correspondem a cada fluxo (principal e alternativo/excepcional) e ajuda a garantir que todos os diversos fluxos sejam enfim reconvergidos. (A inclusão de um diagrama de Atividade no RS*x* resultará na inclusão automática de uma Atividade no caso de uso, com o diagrama sob a Atividade).

- **Opcional:** modele uma realização de 'caixa preta' de cada fluxo nomeado (principal, alternativo e excepcional) do caso de uso: inclua uma ocorrência de colaboração no caso de uso; inclua nela uma instância de interação correspondente ao fluxo principal do caso de uso além de uma instância de interação para cada fluxo nomeado alternativo e excepcional; componha um diagrama de seqüência (ou alternativamente, um diagrama de comunicação) para cada instância de interação. Essas instâncias de colaboração de caso de uso não deverão ser confundidas com as realizações de caso de uso em nível de análise (conforme descrito no modelo de Análise) ou com as realizações de caso

[5] O formato representado no exemplo de descrição de caso de uso foi obtido com a criação do 'gabarito' textual para uma descrição de caso de uso, utilizando um editor com capacidade para RTF e, em seguida, copiando e colando o gabarito no campo de descrição do caso de uso.

de uso em nível de design (conforme descrito no Modelo de Design). Essas são realizações de "caixa branca" do caso de uso e descrevem interações entre os elementos internos de uma solução. As ocorrências de colaboração propostas aqui para o Modelo de Caso de Uso são estritamente interações de "caixa preta" entre os agentes e o sistema. (Consulte a **Figura 5-3**) **Análise racional:** isso fornece aos investidores não técnicos uma imagem de alto nível de como os usuários do sistema interagirão com o sistema. Pode também ajudar a identificar as diversas visualizações (telas ou páginas) que serão necessárias como parte da implementação. Além disso, estabelece formalmente a nomenclatura dos vários fluxos (cenários) do caso de uso, designando esses nomes aos elementos do modelo semântico (isto é, às ocorrências de colaboração).

> ***XDE/Rose***
> No UML 1.x, você teria utilizado "Instância de Colaboração" em vez de "Ocorrência de Colaboração" para essa finalidade.

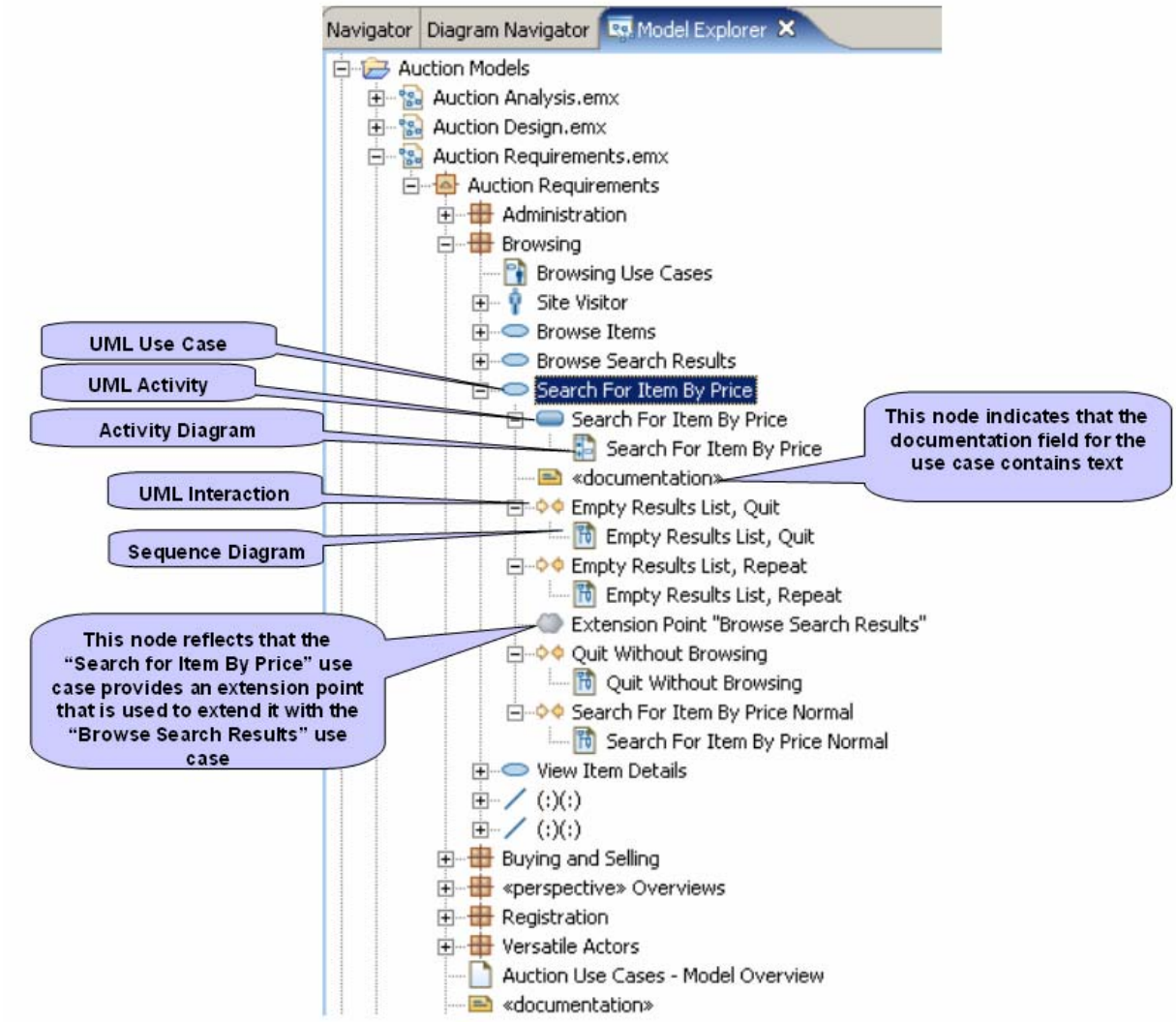


Figura 5-3

- **Opcional:** se você estiver seguindo a orientação do RUP para identificar visualizações 'arquiteturalmente significativas" de sua arquitetura e, particularmente, se for manter um Documento

de Arquitetura de Software, inclua um pacote de «perspectivas» de alto nível para conter diagramas de caso de uso que representem os casos de uso arquiteturalmente significativos. Talvez você queira nomear o pacote como “Visualização de Caso de Uso de Arquitetura”.

6. Diretrizes de Organização Interna de Modelo de Análise

O Modelo de Análise representa a ‘primeira parte’ em uma solução. É o ponto de partida para ir dos requisitos ao design final, com foco na captura de informações sobre o domínio de negócios e na apresentação dos candidatos a elementos de solução, em um alto nível de abstração próximo do negócio. É onde residem as Classes de Análise e as Realizações de Caso de Uso de nível de análise. É por meio do processo de modelagem de realizações de caso de uso (principalmente utilizando diagramas de seqüência) que você começa a *descobrir* quais classes são necessárias para a solução– particularmente, essas serão as classes correspondentes às linhas de vida que você descobre que precisa nos diagramas de seqüência. Há também algumas regras práticas que podem ser aplicadas para sugerir o conteúdo do modelo de análise com base no conteúdo do modelo de caso de uso. Isso será tratado posteriormente nesta seção.

No RUP, a decisão de manter ou não um Modelo de Análise independentemente do Modelo de Design é específica de um projeto; decisão esta que você poderá tomar com base no fato de acreditar ou não que o valor da manutenção do Modelo de Análise separado garantirá o tempo investido. Se um Modelo de Análise separado for criado, mas não mantido, as Classes de Análise serão movidas para o modelo de design e refinadas. Ou talvez o modelo de análise evolua gradualmente para se tornar um modelo de design[6]. Em termos específicos ao produto, seguem algumas opções que podem ser consideradas:

1. Crie um modelo de análise que resida em um arquivo de modelagem (ou conjunto de arquivos) com base no gabarito de Modelo de Análise. Em seguida, utilize um processo manual ou transformações automatizadas para criar versões refinadas dos elementos de análise em um segundo arquivo de modelo (ou conjunto de arquivos) com base no gabarito de Modelo de Design de TI Corporativo; em seguida, descarte os arquivos de modelagem de análise. Isso dá a você a opção de manter em andamento o modelo de análise separado, ou descartá-lo.
2. Realize uma modelagem em nível de análise em um arquivo (ou conjunto de arquivos) de modelagem com base no gabarito do Modelo de Design de TI Corporativo, ao qual você aplicará o Perfil de Análise. Dessa maneira, você pode iniciar a modelagem das realizações de caso de uso utilizando as classes de análise, e periodicamente refiná-las para que as interfaces de design assumam as funções nos comportamentos.
3. Um misto da segunda e terceira opções é manter um modelo de análise de classificações no(s) mesmo(s) arquivo(s) de modelagem do modelo de design. Para fazer isso, separe o conteúdo de análise em pacotes aos quais você aplica a palavra-chave «analysis». Isso dá a oportunidade de reter artefatos de nível de análise nos mesmos arquivos de modelagem que suas contrapartes de nível de design mais refinadas.

Um fato do qual estar ciente ao utilizar as transformações do RSx para gerar implementações é que essas transformações podem em muitos casos aceitar os elementos de nível de análise como suas entradas, livrando você de algumas etapas de refinamento manual desses elementos em elementos de design. Ao utilizar o RSx dessa maneira, as opções 2 ou 3 anteriores são preferenciais. As

[6] O RUP de fato chama a opção de criar classes de análise e realizações de caso de uso em nível de análise no modelo de design e depois evoluí-los diretamente para suas formas de design a partir daí. .Sob essa abordagem, conforme o modelo de design for “descoberto”, você poderá criar pacotes durante esse tempo, nos quais preservará algumas das perspectivas de “análise pura”.

transformações de geração de código padrão compactadas como parte do RSx ignorarão os pacotes de modelo que tiverem a palavra-chave «analysis».

Organização de Alto Nível do Modelo de Análise

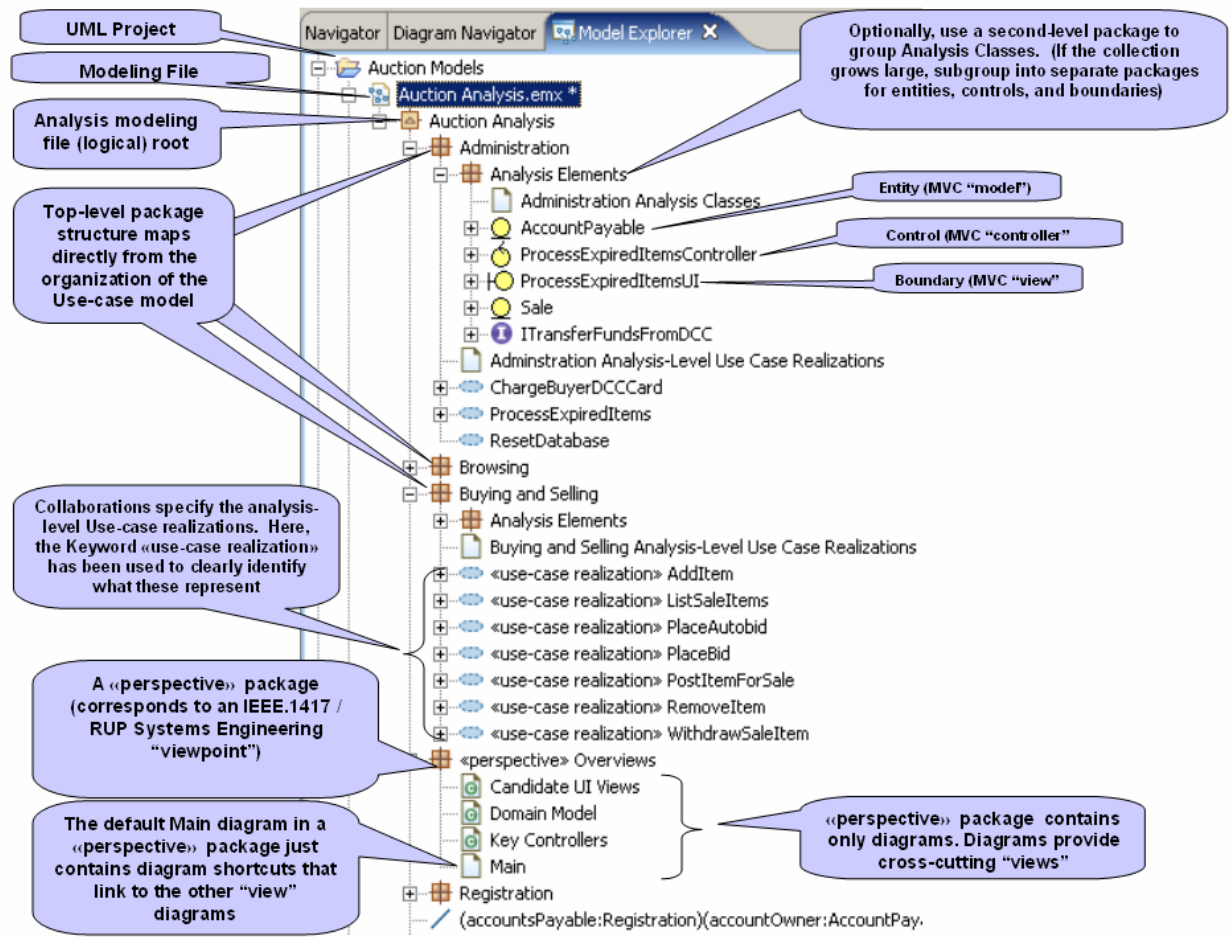


Figura 6-1

Figura ***6-1*** anterior ilustra as seguintes diretrizes de estruturação dos modelos de análise:

1. Utilize pacotes de nível superior para estabelecer agrupamentos orientados funcionalmente para classes de análise. Análise racional: a mesma do modelo de caso de uso.

2. Opcionalmente, nos pacotes de nível superior, utilize subpacotes para coletar e organizar as classes de análise.

3. Utilize diagramas em pacotes de «perspectiva» para capturar visualizações alternativas, de alto nível ou de recorte cruzado dos elementos de análise. Análise racional: forneça perspectivas diferentes para investidores diferentes, mantendo ao mesmo tempo os elementos semânticos do modelo organizados em agrupamentos orientados funcionalmente.

Uma pequena variação dessa abordagem é representada a seguir na **Figura 6-2** que mostra o uso de um pacote de alto nível para separar as realizações de caso de uso das classes de análise. Nesse pacote de alto nível, vemos um conjunto de subpacotes orientados funcionalmente que corresponde ao conjunto de pacotes de nível superior. O isolamento das realizações de caso de uso dessa maneira permite refatorar a estrutura do pacote que contém as classes de análise, sem *necessariamente* afetar a organização das realizações de caso de uso. (Especificamente, se o modelo de análise evoluir para design *em sua posição original*, é provável que a organização do pacote para classes evolua de forma que não corresponda mais à que foi originalmente utilizada para os casos de uso.)

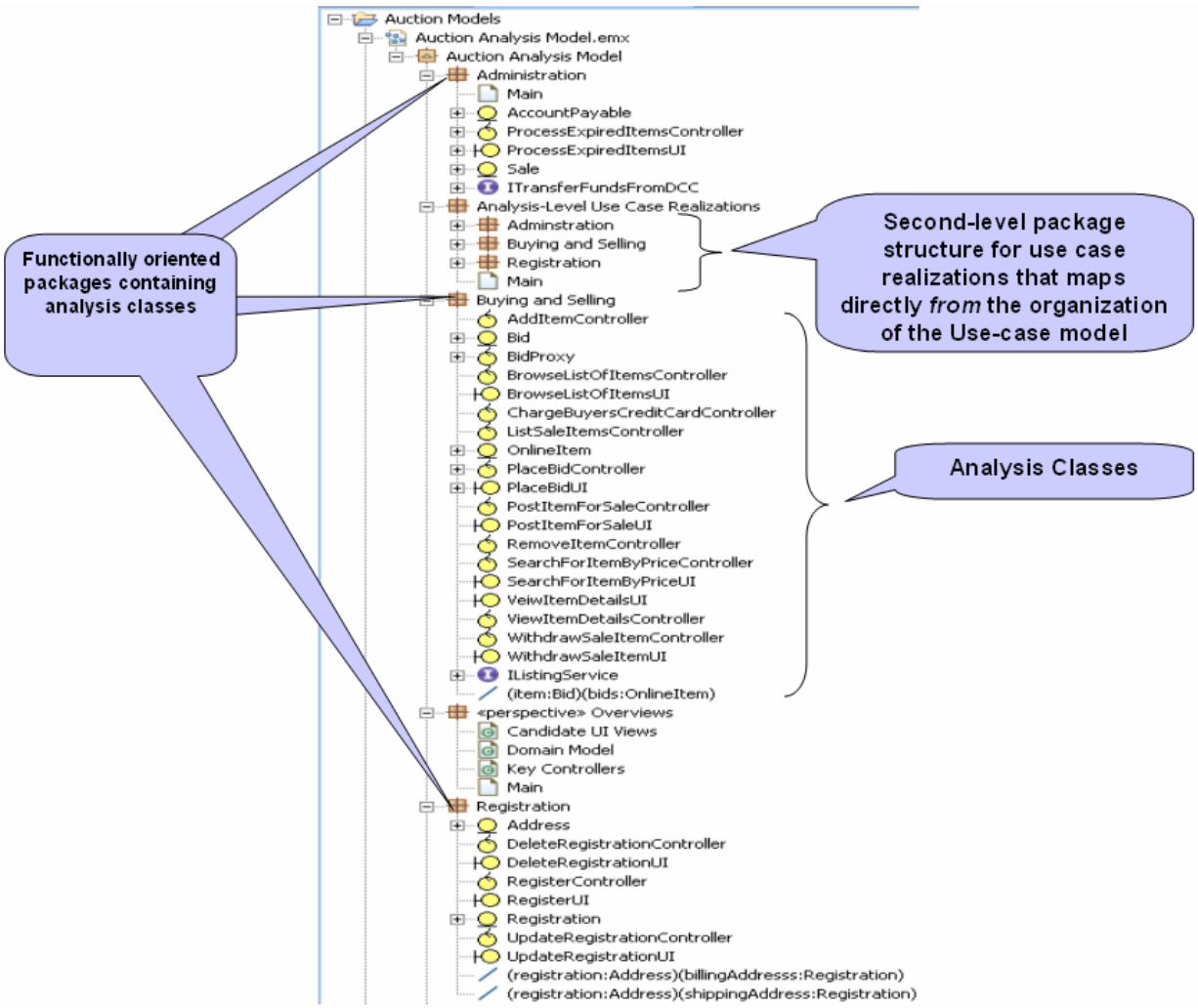


Figura 6-2

Dependendo do seu caso, pode haver motivo para introduzir o uso de uma convenção de nomenclatura que preveja a mesclagem e a reutilização do conteúdo de modelo criado por vários grupos independentes, incluindo grupos em diferentes empresas (parceiros). Se isso for do interesse, considere o uso de uma convenção invertida de espaço de nomes de domínio da Internet conforme representado a seguir na **Figura 6-3**. Observe que essa provavelmente não é a grande preocupação da modelagem de análise *por si só*, mas se você estiver utilizando a abordagem de permitir que o modelo de análise evolua para o modelo de design *em sua posição original,* e prevê a reutilização ou a integração de negócios no nível de design, talvez queira avançar no planejamento. Outra possível vantagem de adotar essa abordagem: como ela pode ser bem mapeada para a

organização do código gerado a partir da análise/design, pode simplificar a configuração subseqüente das transformações de geração de código.

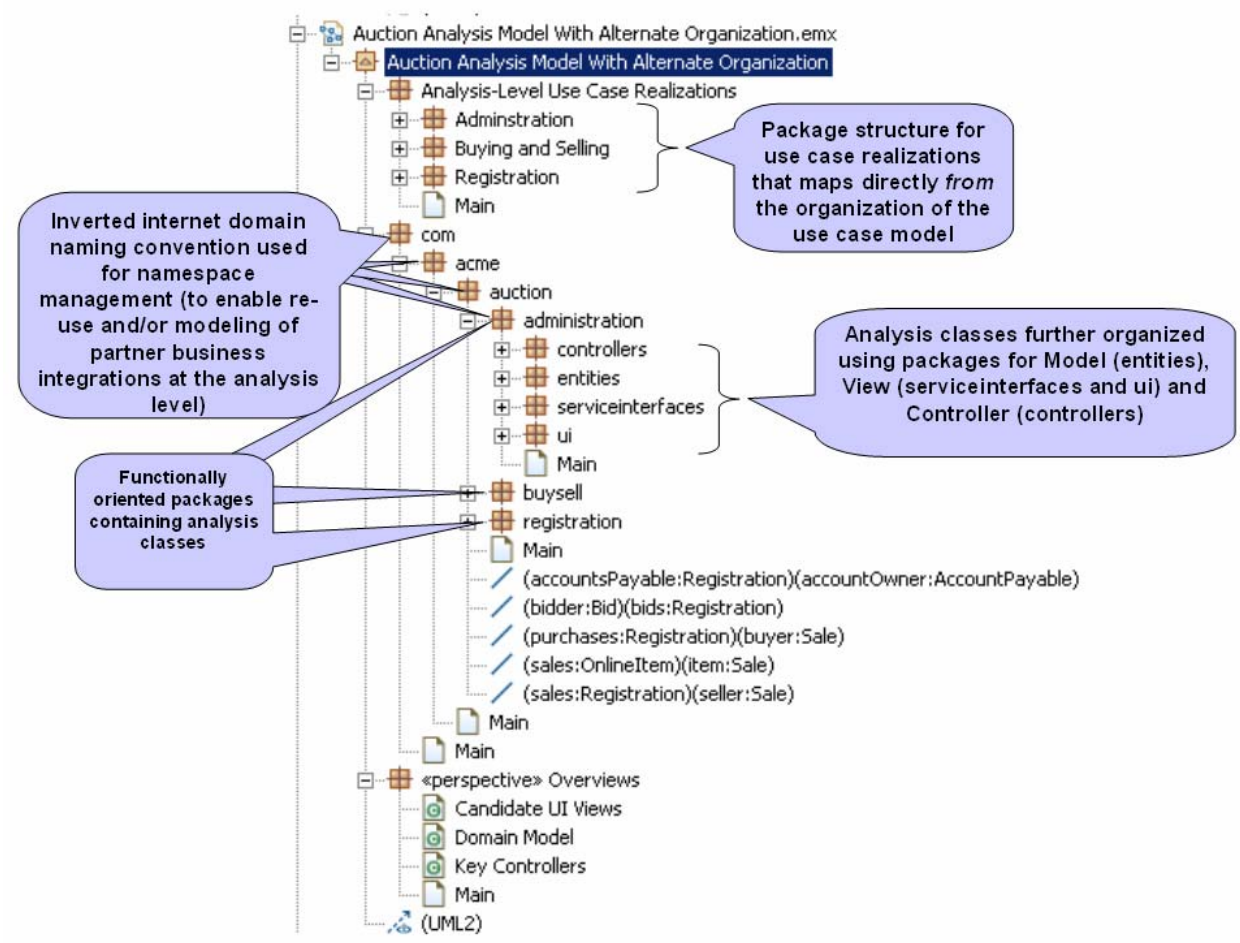


Figura 6-3

Conteúdo do Modelo de Análise

Há várias maneiras de descobrir o que são as classes de análise. Uma delas é começar a desenhar os diagramas de seqüência que sugerem realizações de caso de uso. Conforme assim o fizer, você descobrirá quais linhas de vida precisa, e cada linha de vida em geral corresponderá a uma classe de análise apropriada. Ao descobrir as classes dessa maneira, você poderá criá-las nos pacotes de realização de caso de uso do modelo de análise, mas não deverá deixá-las lá. Você deve 'refatorar' o modelo para mover as classes de análise para os pacotes orientados funcionalmente, conforme descrito anteriormente nas diretrizes de organização de nível superior do modelo de análise (consulte a ***Figura 6-1***)

Outra abordagem útil para a descoberta de classes de análise: 'iniciar' o modelo de análise com classes que se baseiam nessas regras práticas:

- Para cada caso de uso (no modelo de caso de uso), inclua uma classe de «controle» no modelo de análise. Classes de «controle» representam a lógica de negócios associada ao caso de uso.

(Posteriormente, no design, elas também serão mapeadas para assuntos como gerenciamento de sessões.)

- Para cada relacionamento agente/caso de uso (no modelo de caso de uso), inclua uma classe «limite» no modelo de análise. As classes «limite» representam interfaces entre a solução e um agente humano ou entre a solução e um sistema externo. É provável que as classes «limite» que correspondem a um agente humano sejam eventualmente mapeadas para um ou mais artefatos de interface com o usuário no design e na implementação. As classes «limite» que correspondem a um sistema externo podem eventualmente ser mapeadas para um tipo de camada de adaptador no design e na implementação.
- Por meio de um processo como análise de cartão CRC, ou análise de texto das descrições de caso de uso, identifique as classes (verbos) de «controle» e as classes (nomes) de «entidade» adicionais

Ao utilizar essa abordagem inicial para identificar classes de análise, você pode colocar as classes diretamente nos pacotes orientados funcionalmente, conforme descrito anteriormente nas diretrizes de organização de alto nível do modelo de análise (consulte a ***Figura 6-1***)

Por mais que você trabalhe na descoberta de classes de análise, é quase certo que reconheça a necessidade de alterações na organização de seu pacote funcional original.

Opcional: utilize os pacotes de segundo nível, incluídos nos pacotes de classe de análise, para organizar ainda mais o conteúdo desses pacotes (consulte a **Figura 6-4**)

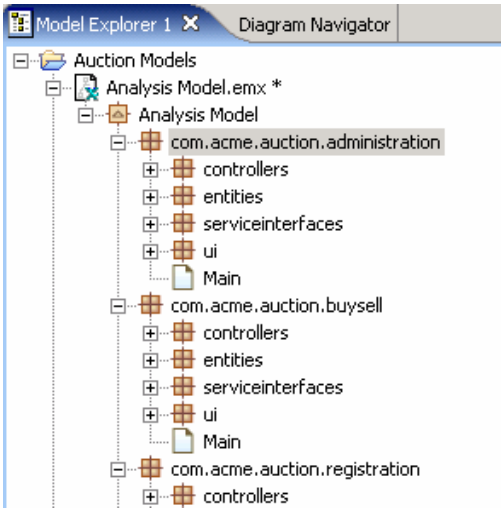


Figura 6-4

Recomendado: O modelo de análise deve conter realizações de caso de uso em nível de análise**,** que descrevem como os casos de uso são executados em termos de classes de análise. Cada realização de caso de uso de análise (representada por uma Colaboração UML) realiza um caso de uso no modelo de caso de uso e tem o mesmo nome desse caso de uso. Consulte a **Figura 6-5**. Para cada fluxo[7] de caso de uso nomeado que você achar que deveria ser modelado como uma realização em nível de análise, inclua um diagrama de seqüência (que incluirá automaticamente uma Interação própria). **Figura 6-6** mostra os tipos de conteúdo semântico que serão incluídos no modelo, conforme você criar diagramas de seqüência. (Observe que você pode filtrar qualquer tipo de elemento UML a partir da visualização do Model Explorer e, assim, ocultar boa parte da 'confusão' representada na **Figura 6-6**)

[7] Conforme estabelecido anteriormente no Modelo de Caso de Uso

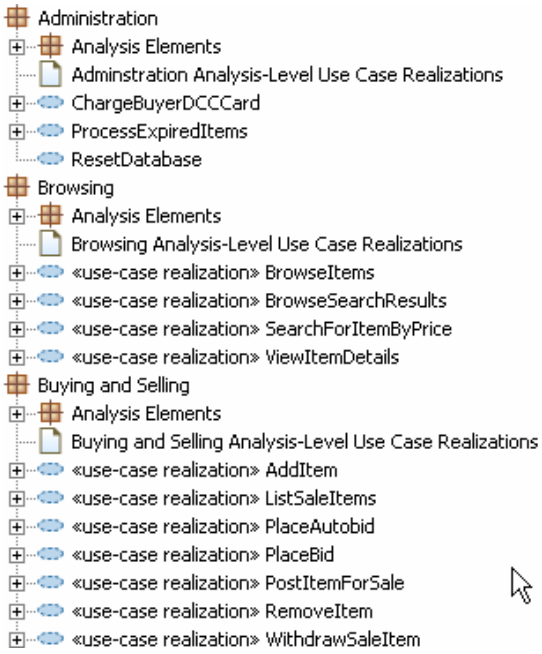


Figura 6-5

Opcional: Uma vez criado o diagrama de seqüência para um fluxo de caso de uso, é possível selecionar sua Interação UML própria no Model Explorer e incluir nele um Diagrama de Comunicação. O novo Diagrama de Comunicação será preenchido automaticamente pelas instâncias de classe de análise que participaram do diagrama de seqüência.

Recomendado: Crie um relacionamento de dependência de Realização de cada realização de caso de uso (Colaboração UML) e o caso de uso correspondente do modelo de caso de uso (consulte a **Figura 6-6**). Como é possível utilizar recursos como Diagramas de Tópico e Análise de Rastreabilidade para entender os relacionamentos de rastreabilidade em seu modelo, você não precisa realmente reter diagramas permanentes para representar os relacionamentos de rastreabilidade; por isso, recomenda-se criar os relacionamentos utilizando um tipo de diagrama 'descartável', por exemplo:

- inclua um diagrama de formato livre na Colaboração
- arraste a Colaboração para ele
- arraste o caso de uso para ele
- trace o relacionamento de dependência
- por último, (no Model Explorer) exclua o diagrama da Colaboração

Recomendado: Inclua um diagrama de "Participantes" para cada realização de caso de uso a fim de mostrar as classes de análise que participam da realização (isto é, as classes de análise cujas instâncias aparecem nos diagramas de interação que descrevem a realização do caso de uso) e os relacionamentos entre essas classes que suportam a colaboração descrita nos diagramas de interação. Consulte a **Figura 6-6**

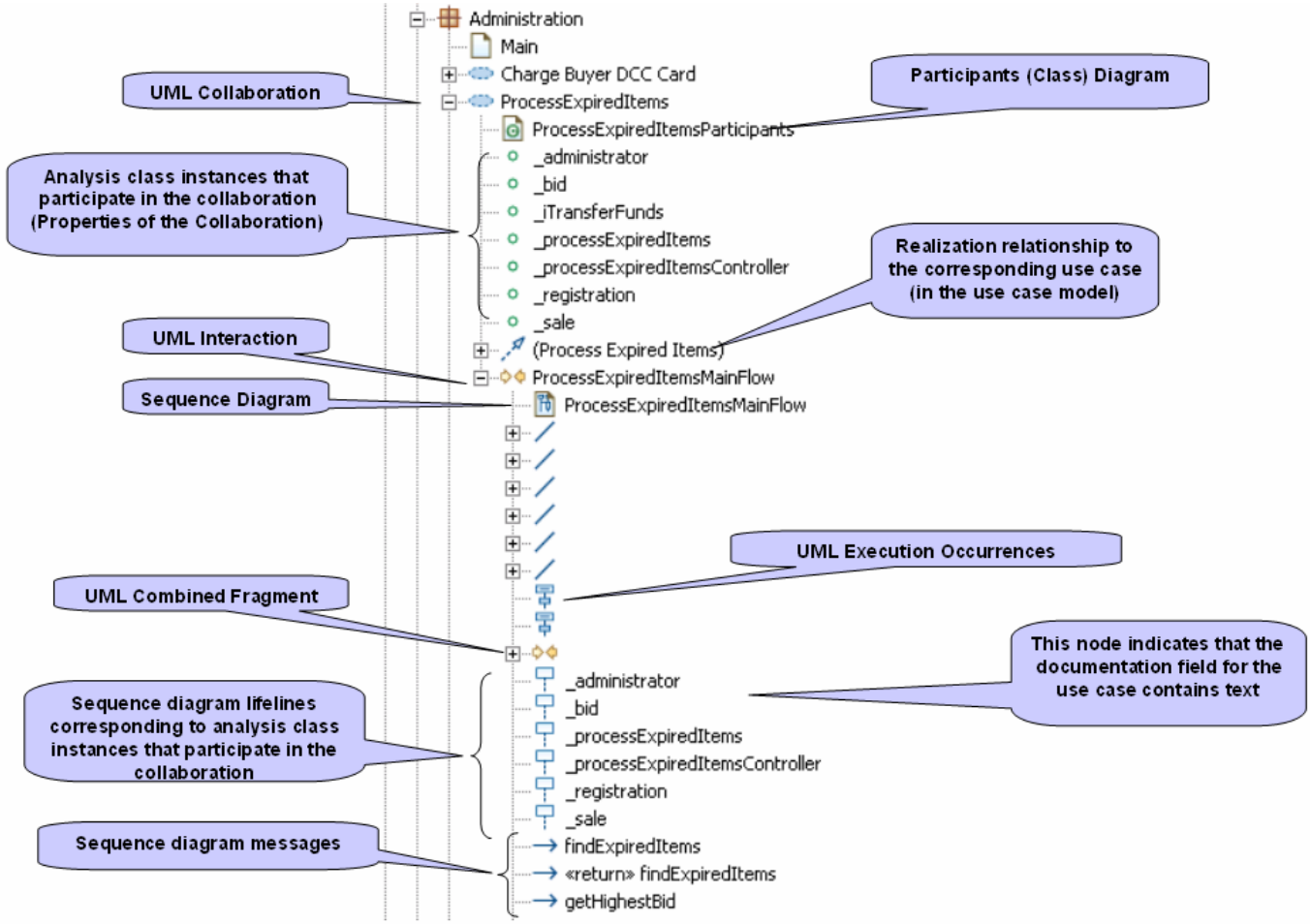
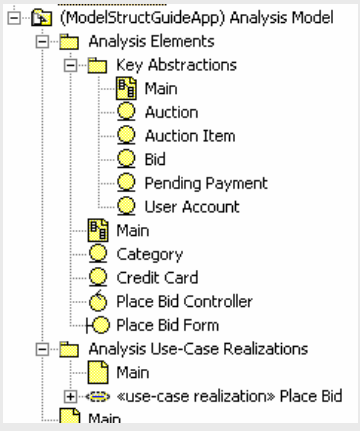


Figura 6-6

XDE/Rose

A estrutura tradicionalmente recomendada para **Modelo de Análise** conforme mostrado a seguir foi modificada no RSx para dar ênfase a uma organização de pacote orientada funcionalmente para as classes de análise. Observe também que a utilização de um pacote de Abstrações de Chaves (que poderia comprometer uma abordagem de empacotamento orientado funcionalmente de outra forma) é substituída pela utilização de um diagrama (ou diagramas) de Abstrações de Chaves em um pacote de «perspectiva».



7. Diretrizes de Organização Interna do Modelo de Design

Organização de Alto Nível do Modelo de Design

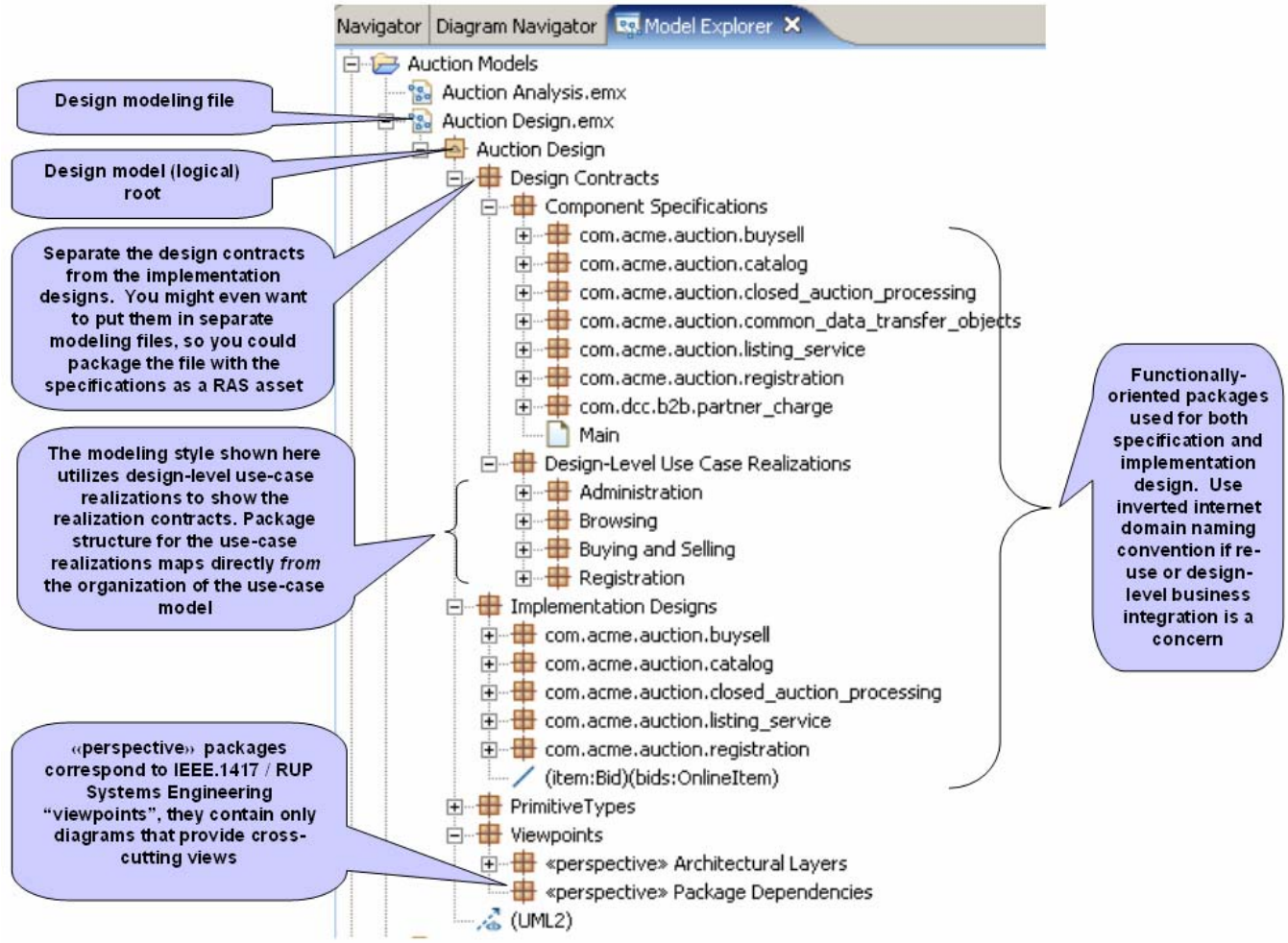


Figura 7-1

Figura **7-1** anterior ilustra as seguintes diretrizes de estruturação dos modelos de design:

1. Separar especificações a partir de designs de implementação. A ilustração mostra a utilização de pacotes de "Contratos de Design" e "Designs de Implementação" de nível superior para esta realização.

2. Utilize pacotes de nível inferior para estabelecer agrupamentos orientados funcionalmente. Você pode, por exemplo, começar com a organização utilizada durante a análise e deixá-la evoluir à medida que toma decisões de como as classes de análise serão mapeadas para classes de design, componentes e serviços reais. (Todo esquema organizacional inicial provavelmente evoluirá durante o design -- consulte discussão adicional a seguir).

   Uma palavra sobre subsistemas pode ser adequada nesse ponto. Nas versões do UML anteriores à

2, um subsistema é um tipo especializado de pacote. No UML 2, um subsistema é um tipo especializado de componente, e um Componente pode conter pacotes. Assim, no UML 2, Componentes de «subsistema» são alternativas de organização/espaço de nomes viáveis aos pacotes; o UML2 ainda é indefinido quanto ao uso apropriado de subsistema vs. pacote. Sugestão: utilize Pacotes em níveis de granularidade, como os subsistemas de design de um determinado aplicativo, e reserve os Subsistemas para representar aplicativos inteiros (por exemplo, CRM ou SCM) em visualizações de arquitetura gerais da empresa.

> ***XDE/Rose***
> *No momento da elaboração deste documento, esperava-se que as ferramentas de importação dos modelos Rose e XDE oferecessem a opção de mapear os subsistemas UML 1.x para Subsistemas UML2 ou para pacotes com a* palavra-chave «subsistema» *aplicada*

3. É provável que a organização de elementos de design evoluirá do modo como os casos de uso do sistema são organizados (no modelo de caso de uso e talvez no modelo de análise, se um modelo de análise separado estiver sendo mantido). Utilize pacotes para subdividir ainda mais os contratos de design nas especificações de elemento de design (os contratos de uso), e as realizações de caso de uso em nível de design (os contratos de realização), e manter uma subestrutura de pacotes para as realizações de caso de uso que continuam a refletir a organização dos próprios casos de uso.

4. *Considere* o uso de camadas de arquitetura como a base para o esquema organizacional de segundo nível para os elementos que compõem as especificações e os designs de implementação das áreas funcionais (e consulte discussões adicionais a seguir)

5. Em componentes e pacotes UML que agrupam elementos de modelo semântico, coloque os diagramas que fornecem visualizações específicas desse agrupamento. Essa diretriz refere-se ao fato de esse agrupamento ser baseado em subconjuntos do domínio de negócios orientados funcionalmente, em uma camada de arquitetura ou no que você tiver. Faça com que o diagrama 'padrão' tenha o mesmo nome do próprio pacote ou componente e componha-o para mostrar uma visão geral do conteúdo do pacote. Isso mantém alguns diagramas próximos do que eles representam, tornando mais fácil navegar e entender o modelo.

6. Talvez você queira introduzir a utilização de um espaço de nomes de domínio da Internet invertido no modelo de design. Análise racional:
   - Basicamente, os mesmos motivos de que fazer isso é importante com relação a implementações específicas do idioma:
     a. cenários envolvendo trabalho de integração onde haja vários aplicativos orientados ao modelo envolvidos (especialmente com empresas parceiras)
     b. reutilização de cenários
   - Isso provavelmente simplificará a configuração subseqüente de transformações em implementação (mapeamento de local e nome da origem para o destino).

7. *Considere* o uso de nomes de pacote que serão válidos na(s) plataforma(s) de implementação de destino para evitar a carga e uma possível confusão de mapeamento de espaço de nomes. (Em sua maior parte, isso significa simplesmente "não utilizar espaços ou pontuações que não sejam sublinhados nos nomes".)

8. Utilize minúsculas em nomes de pacote para facilitar a distinção de nomes de classe em um pacote.

9. *Considere* o uso de nomes diferentes para Interfaces e os Componentes ou as Classes que os realizam. Utilize ILoan e Loan ou Loan e LoanImpl para os nomes de interfaces e implementações. Isso realmente não é necessário no modelo, mas muitas vezes é uma boa idéia em código gerado, de modo que essa é outra área em que você pode poupar a si mesmo de um trabalho de configuração de transformação subseqüente.

10. No cenário a seguir, todo conteúdo em nível de análise do qual o código não se destina a ser gerado deverá ser separado em pacotes estereotipados como «análise»[8].
    - A) você optou por ignorar o uso de um modelo de análise separado e preencher o modelo de design com conteúdo de nível de análise e manter esse conteúdo no nível de abstração de análise, criando também ao mesmo tempo o conteúdo de nível de design no mesmo modelo, e
    - B) você conduzirá transformações de modelo em código no Modelo de Design EIT

11. Utilize diagramas em pacotes de «perspectiva» para capturar visualizações de recorte cruzado e alto nível dos elementos de design. Análise racional: forneça visualizações de recorte cruzado, visualizações de conteúdo 'arquiteturalmente significativo' e visualizações que atraiam diferentes tipos de investidores, mantendo ao mesmo tempo os elementos semânticos do modelo organizados em agrupamentos orientados funcionalmente.

É importante reconhecer que as estruturas de pacote dos modelos de design evoluirão com o tempo. Basicamente, a organização deve corresponder ao modo como você estrutura sua arquitetura em componentes e serviços. Essa abordagem à organização da *atividade final* do design possibilitará então o melhor potencial para compactar recursos reutilizáveis e o mapeamento mais direto do design para o conjunto de projetos e pastas que conterão os artefatos de implementação (código, metadados, documentação) gerados do design.

Entretanto, a organização *inicial* deve corresponder mais ou menos diretamente à abordagem organizacional utilizada no modelo de Caso de Uso e depois revisada durante a análise[9]. De fato (conforme descrito na seção anterior "Diretrizes de Organização Interna do Modelo de Análise"), você pode optar por deixar que seu modelo de análise evolua para o design adequado. Em outras palavras, a organização inicial do design tenderá a agrupar conjuntos de interesses de negócios coesos e livremente ligados e a isolar elementos reutilizáveis ou de recorte cruzado. Essa abordagem à organização inicial prova ser eficaz porque

- Se você espera utilizar transformações para gerar conteúdo de modelo de design a partir de conteúdo de modelo de análise ou de caso de uso, os mapeamentos de pacote de origem para pacote de destino serão simples e diretos
- Uma abordagem organizacional inicial com base na coesão funcional e na união livre dos pacotes indica claramente a melhor oportunidade de mapeamento para a organização final orientada a componente, significando que deve reduzir a quantidade de refatoração a ser feita como parte do processo de design
- A união livre de pacotes tem o potencial de aprimorar os fluxos de trabalho de equipe e facilitar a reutilização nos casos em que o design é fatorado em vários arquivos de modelagem

As abordagens alternativas são possíveis, naturalmente, e em alguns casos, aconselháveis como uma organização de *atividade final*:

- Se você estiver direcionando aplicativos da Web baseados em J2EE que incluem EJBs, a organização do design poderá prever as convenções do RSA e do Rational Application

[8] Tais pacotes serão ignorados pelas transformações.

[9] O pacote das classes de análise muitas vezes é refatorado significativamente, conforme é descoberto, para melhor suportar a reutilização e os requisitos funcionais não previstos.

Developer com relação a projetos J2EE.[10] Especificamente, você poderá optar por definir pacotes de design de nível superior que correspondam a camadas de arquitetura (apresentação e negócios, com os negócios divididos em subcamadas na sessão e no domínio). Obviamente, essa não é uma abordagem de plataforma neutra; assim, só será aconselhável se você souber que a solução que está projetando não será implementada em uma plataforma que não a J2EE.

- Em geral, muitas vezes é no caso em que aplicativos com n camadas estão sendo construídos que o conhecimento do desenvolvedor e a divisão do trabalho correspondem às camadas de apresentação e de negócios, de modo que novamente você pode optar por utilizar pacotes de nível superior correspondentes a essas camadas de arquitetura. Mas tenha cuidado ao organizar classes que se destinam a suportar determinadas *funções de negócios* a fim de suportar uma *arquitetura* específica. É ainda mais difícil alterar.
- Se você tiver motivos para utilizar uma abordagem organizacional orientada a não-componente/serviço/subsistema, ainda assim poderá mapear a organização do design para um conjunto de projetos e pastas de destino, investindo pouco trabalho adicional na configuração das transformações de geração de código. Um tipo especial de modelo parceiro referido como 'modelo de mapeamento' pode ser utilizado para definir mapeamentos particularmente complexos.

Conteúdo do Modelo de Design

Não há regras rigorosas para o que deve residir no modelo de design, mas as seguintes sugestões podem ser úteis.

[10] Em termos gerais: Um Projeto Corporativo por sistema ou aplicativo ou subsistema grande e para cada Projeto Corporativo, um projeto da Web para a camada de apresentação e vários projetos EJB nos quais os projetos EJB correspondem geralmente aos componentes ou subsistemas secundários e nos quais normalmente projetos EJB separados são utilizados para a camada de sessão (EJBs de sessão) e de domínio (EJBs de entidade) por componente ou subsistema. Consulte a seção 9 deste white paper para obter informações adicionais.

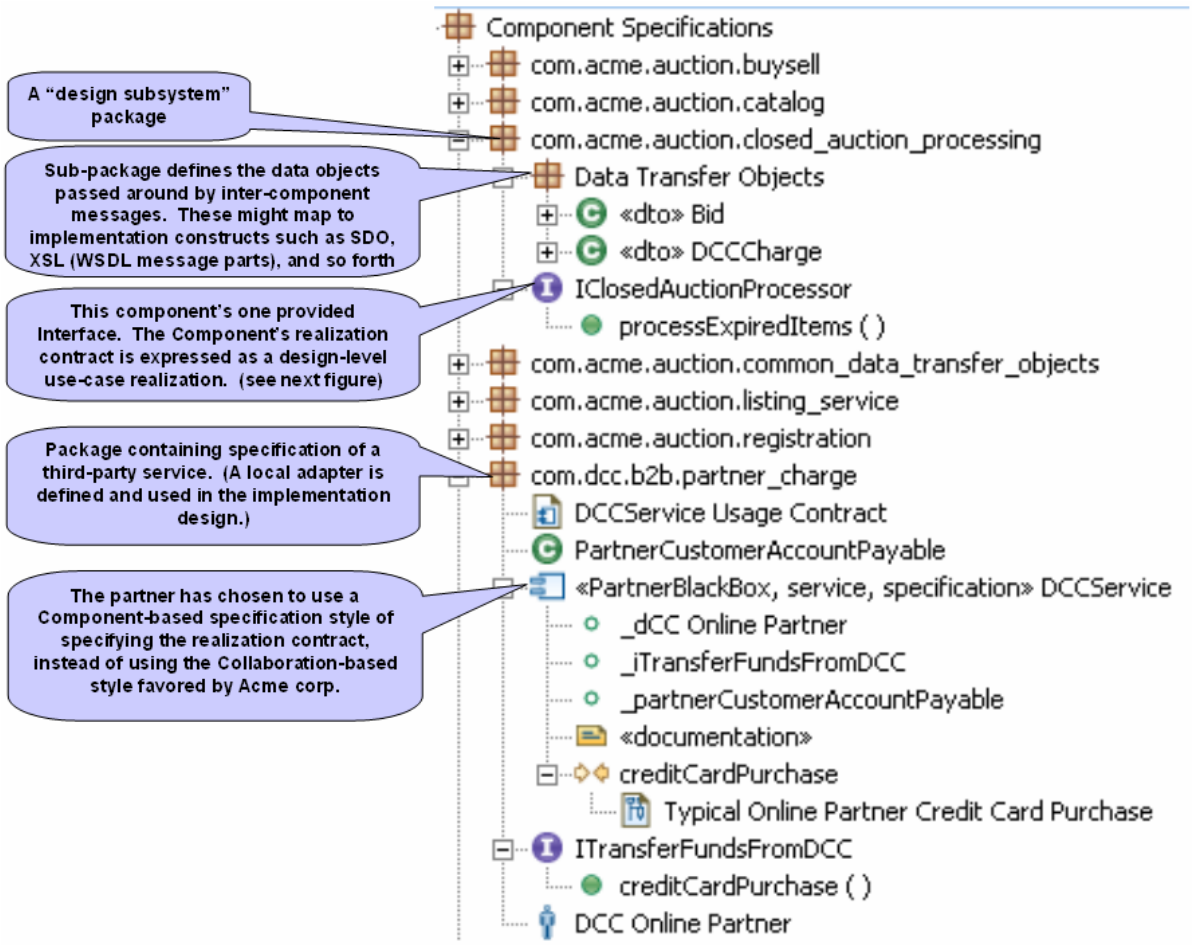

A "design subsystem" package
Sub-package defines the data objects passed around by inter-component messages. These might map to implementation constructs such as SDO, XSL (WSDL message parts), and so forth
This component's one provided Interface. The Component's realization contract is expressed as a 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

Figura 7-2

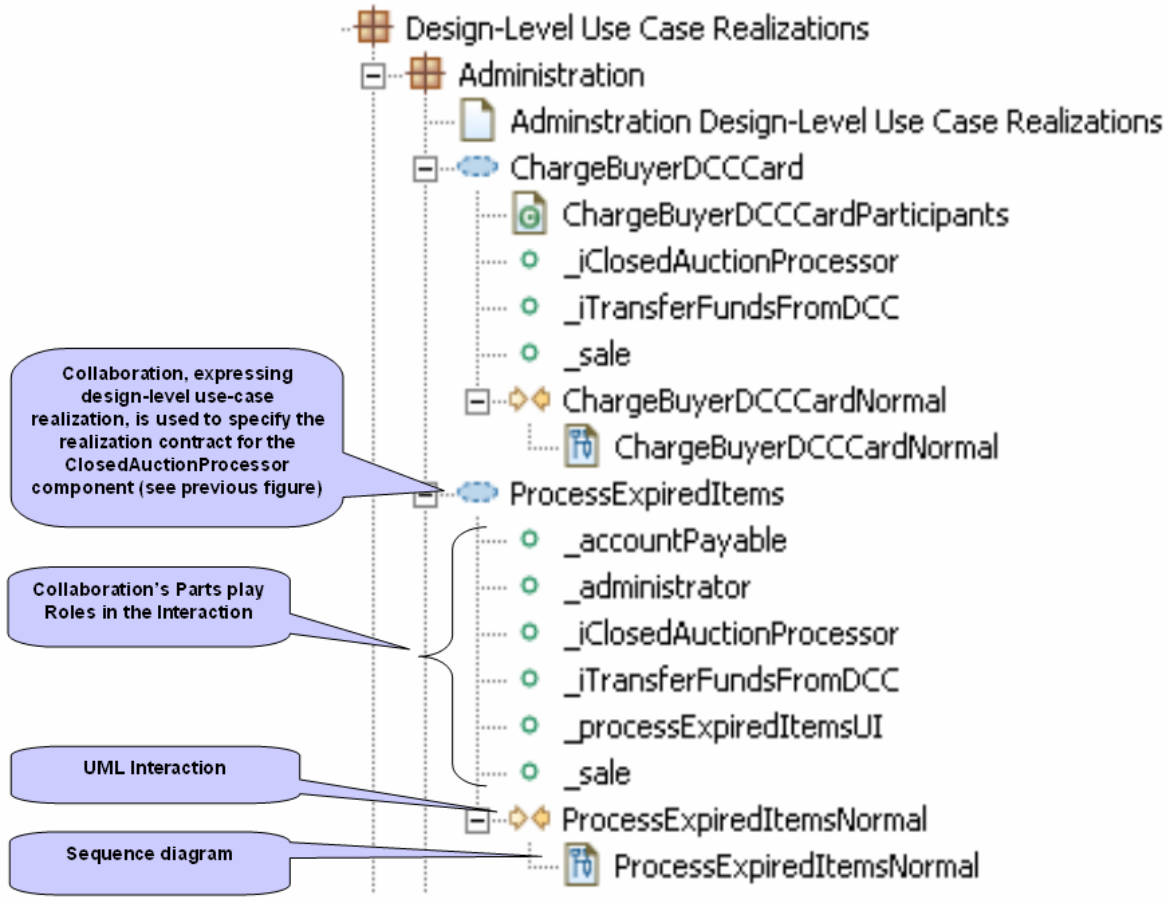


Figura 7-3

Figura 7-2 e a **Figura 7-3** seguem a estrutura organizacional representada na **Figura 7-1** e descrevem como os contratos de design podem ser especificados.

- O contrato de uso para um componente "ClosedAuctionProcessor" é expresso como uma única Interface[11] (**Figura 7-2**). O correspondente contrato de realização é especificado por uma única realização de caso de uso de nível de design expresso como Colaboração[12] (**Figura 7-3**) Observe que, enquanto realizações de caso de uso em nível de análise mostram colaborações entre classes de análise, as realizações em nível de design mostram colaborações entre elementos de design menos abstratos[13]. Se for desejado que o subconjunto de especificação de um modelo de design seja compactado independentemente do subconjunto de design de implementação, é importante que as realizações de caso de uso em nível de design utilizem apenas elementos de especificação de análise ou design –nunca elementos de design de implementação -- em suas funções.

[11] Os componentes naturalmente podem ter várias interfaces fornecidas; o que ocorre é que este exemplo tem apenas uma

[12] Outros componentes podem participar de vários casos de uso do sistema, de modo que seus contratos de realização poderão residir em várias realizações de caso de uso. Nesses casos, você também poderia incluir no mesmo pacote com a Interface do componente um diagrama chamado "{nome do componente} Where-Used" no qual colocará links para os vários diagramas que compõe as realizações de caso de uso desses casos de uso.

[13] Outra provável diferença: alguns diagramas "participantes" nas realizações em nível de design podem ser Diagramas de Componente que representam ligação de componentes, em vez de (ou além de) Diagramas de Classe participantes, conforme sugerido para realizações de caso de uso em nível de análise.

- Os contratos de uso e realização para o “DCCService” de terceiros estão juntos em um pacote[14]. Mais uma vez, vemos que o contrato de uso consiste em uma única Interface, mas nesse caso o contrato de realização é expresso utilizando um Componente de «especificação» (**Figura 7-2**). (Caso contrário, a especificação do contrato de realização simplesmente não faria diferença, expressa com o uso de comportamentos – nesse caso, uma Interação chamada “creditCardPurchase”). Outro exemplo que utiliza Componentes, em vez de Colaborações, é mostrado na **Figura 7-4**

- Operações são definidas em interfaces, que podem ser realizadas por componentes de «especificação» (se utilizados) ou por classificadores no Design de Implementação que implementa as interfaces.

- A especificação de objetos de transferência de dados (que serviria como os tipos de parâmetros das operações fornecidas, e poderiam ser mapeados para construções de implementação, como esquema XML ou SDOs) também pode ser incluída como parte do contrato de uso. Para componentes que não são projetados para serem distributivos, você pode ou não optar por especificar objetos de transferência de dados como as especificações dos tipos utilizados como parâmetros de operação. Para serviços distributivos (por exemplo, Serviços da Web), é obrigatório que as operações do serviço não façam referência a objetos em um espaço de endereçamento local, de modo que deve-se utilizar DTOs.

> ***XDE/Rose***
> Em versões anteriores da UML, a orientação para as realizações de caso de uso era utilizar uma Instância de Colaboração por caso de uso e uma interação e um diagrama de seqüência para cada um dos fluxos significativos da realização.
>
> No RSx, muitas vezes, você poderá utilizar apenas uma interação e um diagrama, porque os diagramas de seqüência UML2 agora suportam notações para caminhos de execução alternativos.
>
> Além disso, na UML 2 não há mais uma ‘Instância de Colaboração’. Em seu lugar, existe o ‘Uso de Colaboração’, que requer uma Colaboração como seu tipo. Por isso, no RSx, utilize Colaborações para representar realizações de caso de uso.

- [14] Observe que a situação hipotética aqui é que a empresa DCC forneceu à empresa Acme a especificação UML, com a Acme então incorporada em seu modelo de design. Esse é o tipo de cenário no qual o uso de espaços de domínio invertidos da Internet poderia ser conveniente.
-

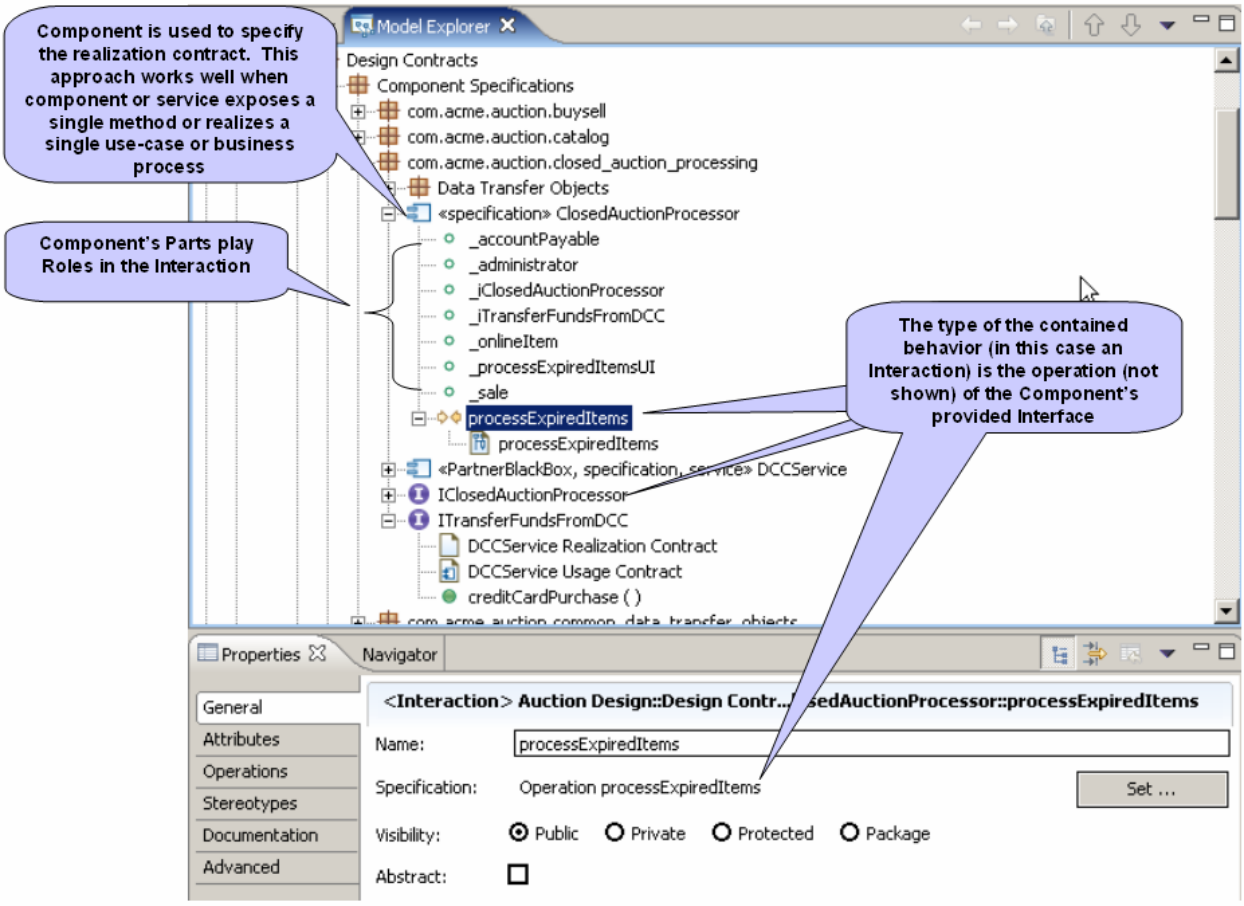


Figura 7-4

- Uma possível abordagem para especificar designs de implementação é mostrada na **Figura 7-5** A estrutura de implementação é definida utilizando classes simples que contêm operações. Essa abordagem é bem típica de modelos de design criados utilizando o UML 1.x. Uma segunda abordagem possível que pode estar mais voltada aos objetivos do UML2 é mostrada na **Figura 7-6**. Aqui, em vez de Classes, vemos que são utilizados Componentes e que os Componentes não possuem Operações e sim comportamentos próprios (nesse caso, uma Interação).

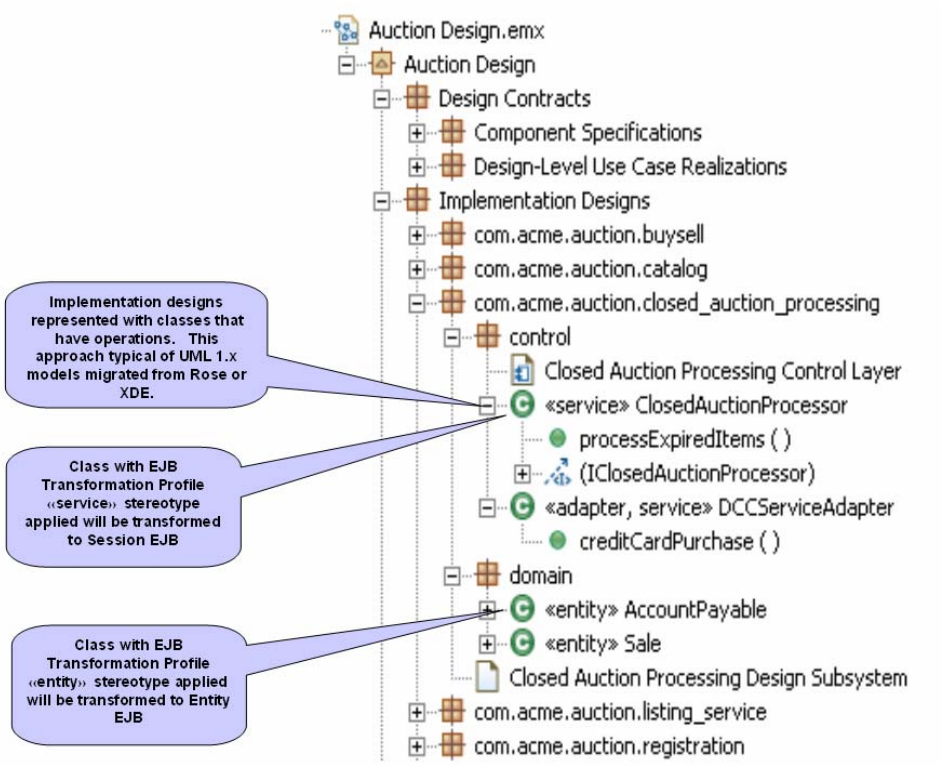


Figura 7-5

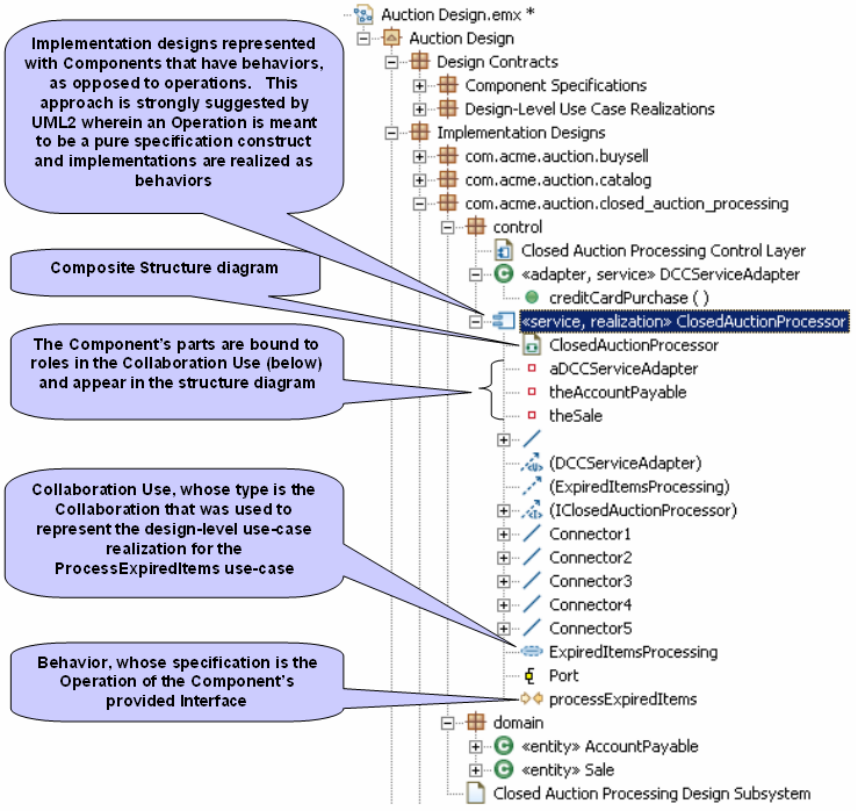


Figura 7-6

8. Diretrizes de Organização Interna do Modelo de Visão Geral da Implementação

> ***XDE/Rose***
> Nas Diretrizes de Estrutura de Modelo XDE, um modelo de Visão Geral da Implementação era recomendado como dispositivo para fornecer uma visão geral da implementação em nível de subsistemas. Os detalhes de cada subsistema eram então especificados no modelo de código do projeto que implementava o subsistema.

Mais exatamente, não deverá ser necessário utilizar um modelo de Visão Geral da Implementação no RSx. Se as diretrizes organizacionais do modelo de design forem seguidas, a organização (atividade final) do modelo de design deverá naturalmente tomar forma em torno dos componentes (incluindo o «subsistema» de maior importância e as variedades de «serviço» mais distributivas). Dessa forma, por meio de transformações, os pacotes do design podem ser mapeados para projetos. Por exemplo, no caso de uma implementação J2EE, eles serão mapeados para os diversos projetos Java, EJB, Web, J2EE Application e outros, nos quais a implementação será desenvolvida. (E esses projetos de fato representam o modelo de implementação da solução, conforme observado na seção Conceitos Básicos e Terminologia deste documento.)

Se você pensar nisso em termos ascendentes, em outras palavras, deverá pensar em como a implementação será organizada em termos de projetos e pastas e fatorar isso na organização do modelo de design de modo que os mapeamentos de transformação acabem sendo diretos. Ou, em termos descendentes, significa que a organização do modelo de design, conforme evoluída no decurso do trabalho de design, deverá determinar o conjunto de modelos de implementação (projetos) que serão necessários. Em qualquer das duas maneiras, a organização de atividade final do modelo de design naturalmente deverá tratar da necessidade de uma visão geral dos projetos e subprojetos, isto é, um diagrama que representa os pacotes no modelo de design deverá ser equivalente a uma visão geral dos projetos e suas subpastas.

Entretanto, você ainda poderá preferir rascunhar a estrutura de seu projeto em um estágio antecipado, ou simplesmente ver uma representação das estruturas do projeto mais explícita visualmente – por exemplo, uma em que os artefatos que representam projetos e pastas tenham explicitamente a palavra-chave «project» e «folder», ou ainda «EJB Project» e «Web Project». Outra consideração é que representar artefatos de implementação detalhados (JARs, por exemplo) seria inapropriado para o modelo de design (que, na teoria de operação do Rational Software Architect destina-se a plataforma neutra). Mas esses artefatos são perfeitamente aceitáveis para inclusão em um modelo de Visão Geral da Implementação. Por isso, há algumas razões para você talvez querer utilizar um modelo de Visão Geral da Implementação. **Figura 8-1-1** a seguir representa uma amostra do modelo de Visão Geral da Implementação

Uma idéia final é que um modelo de Visão Geral da Implementação pode ser um local adequado para capturar diagramas informais de vários aspectos da solução. **Figura 8-2-2** a seguir mostra um diagrama informal de alto conceito do sistema de leilão no qual a maioria das amostras deste documento se baseia.

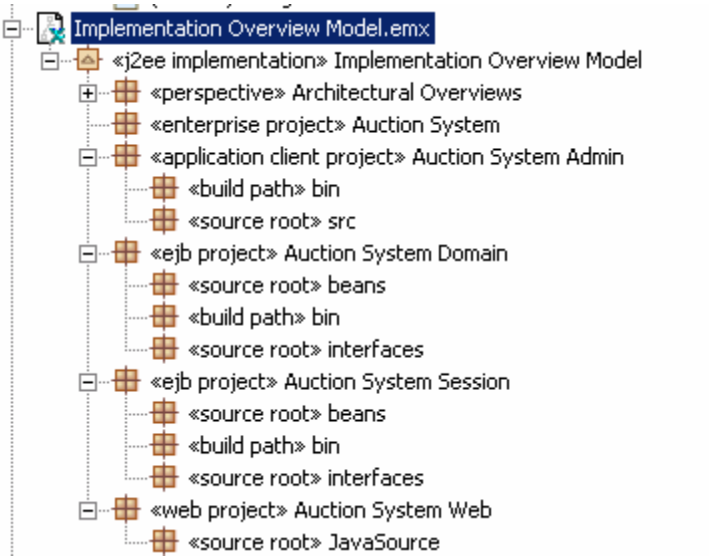


Figura 8-1

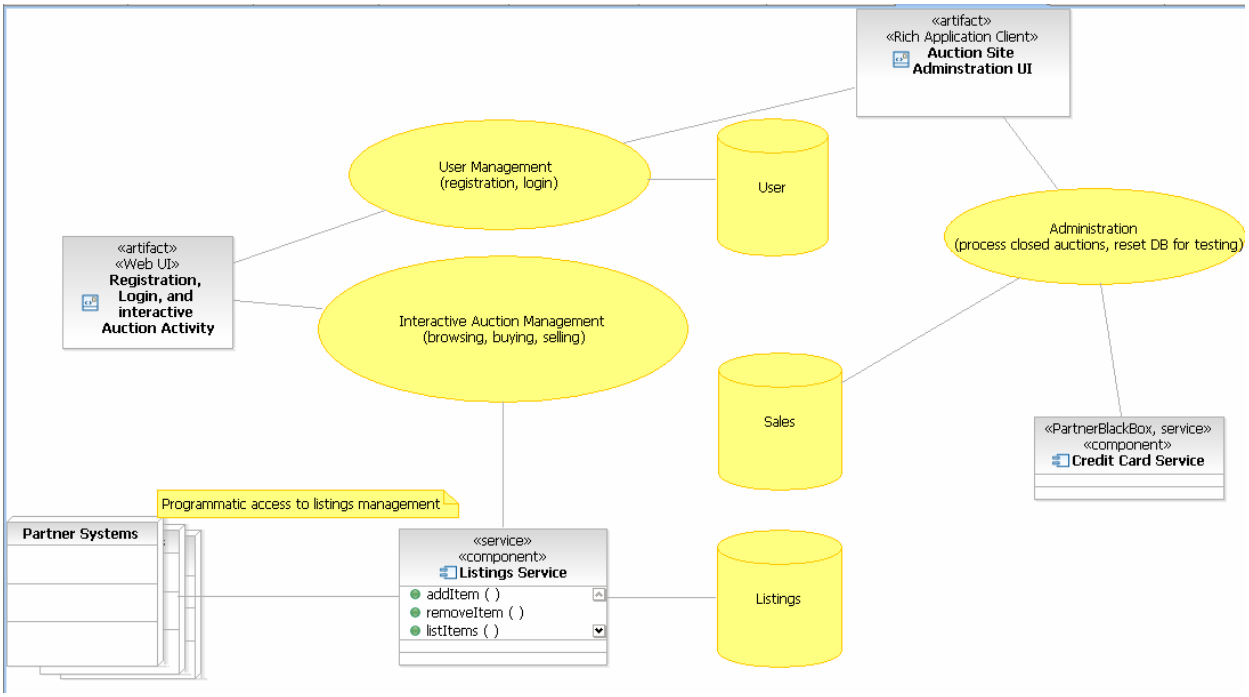


Figura 8-2

Diretrizes de Organização Interna do Modelo de Implantação

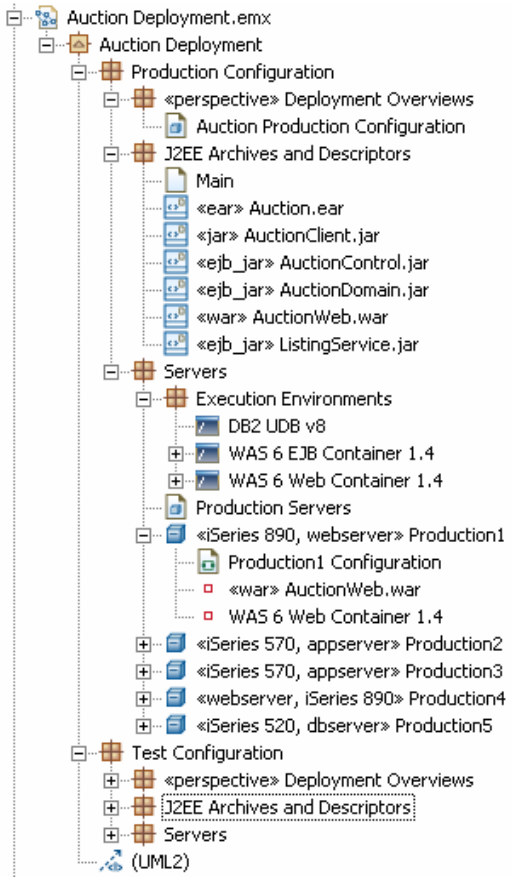


Figura 0-1

É provável que menos se precisa ser dito sobre o modelo de implantação do que sobre qualquer outro modelo tratado neste documento. Normalmente, deve haver muito poucas implicações no recebimento de dados de sua organização de modelagem de implantação e opções de conteúdo; assim, faça apenas o que fizer sentido. Mesmo assim – só para mantê-lo pensando – uma possível estratégia e um pouco de conteúdo representativo são representados acima na **Figura 0-1**. Apenas algumas observações neste exemplo:

Especificações de configurações de produção foram separadas das configurações de teste

Visões gerais (por exemplo, clusters, centros de dados ou empresas) são mantidas em pacotes de «perspectivas»

Uma abordagem mais simples foi tomada com respeito à especialização e à classificação de nós e artefatos: uma combinação de pacote e uso de palavras-chave. Uma abordagem mais sofisticada seria desenvolver um perfil especializado de UML que definisse estereótipos especializados e propriedades apropriadas para descrever e documentar os tipos de recursos utilizados em seu próprio ambiente.

Utilizando um Arquivo de Modelagem para Representar o Documento de Arquitetura de Software

Dadas suas ferramentas para organizar modelos, como links de diagrama e suporte para vários arquivos de modelo com referências de modelo cruzado, torna-se um assunto quase trivial criar um modelo que, efetivamente, represente o Documento de Arquitetura de Software do RUP e as “4+1 Visualizações da Arquitetura”.

Na forma mais simples, seria possível fazer alguma coisa ao longo das linhas da **Figura 0-1**. Criar um arquivo de modelagem e preenchê-lo com um conjunto simples de pacotes correspondentes às 4+1 Visualizações. (O exemplo é mostrado sem um pacote para Visualização do Processo, visto que o sistema neste exemplo pouco apresenta em matéria de simultaneidade.)

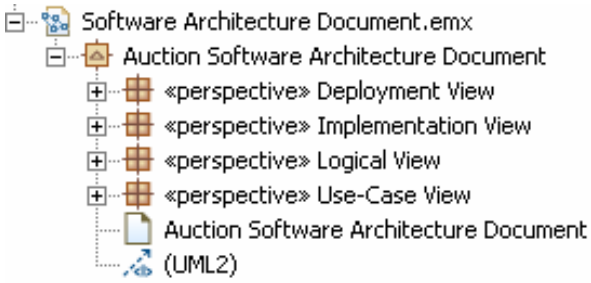


Figura 0-1

Depois, talvez, compor o diagrama padrão ao longo das linhas sugeridas na **Figura 0-2**. Você também pode incluir texto ou notas adicionais neste diagrama

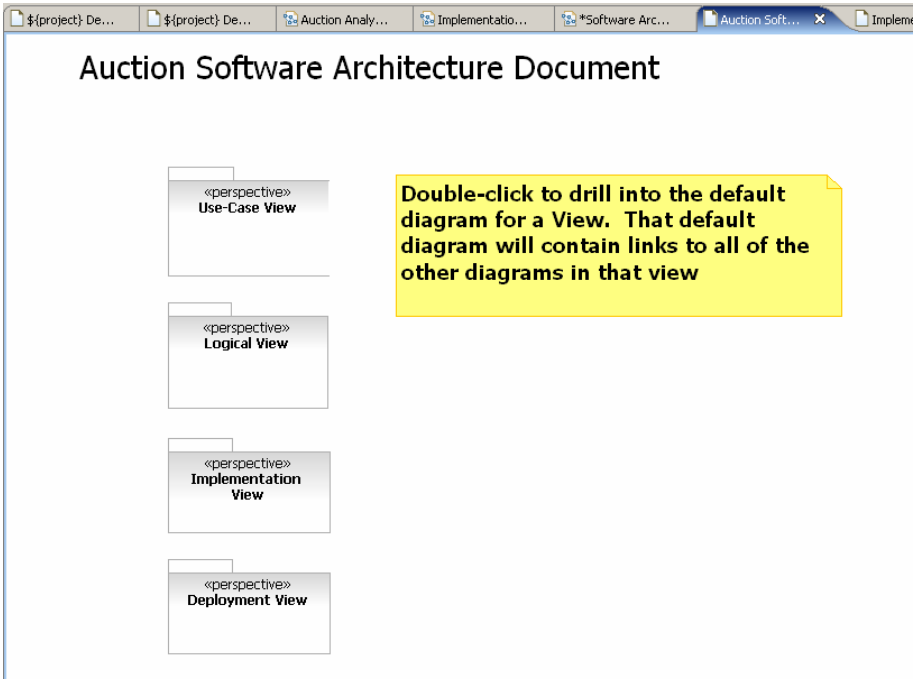


Figura 0-2

Em seguida, apenas criar diagramas no arquivo de modelagem do Documento de Arquitetura de Software, utilizando estas abordagens:

Crie diagramas que sejam compostos pelos elementos de semântica de UML de outros arquivos de modelagem e que representem novas visualizações que não foram encontradas nesses outros arquivos de modelagem, mas que são necessários como parte do documento de arquitetura

Crie diagramas que sejam compostos por formas geométricas e/ou elementos de UML “ad-hoc” que residam no arquivo de modelagem do Documento de Arquitetura de Software. (Esses elementos UML devem servir apenas para finalidades de documentação ou de esclarecimento e não devem ter significado semântico para a implementação real da solução que está sendo descrita)

Crie diagramas que apenas contenham links para os diagramas existentes em outros arquivos de modelagem. (Essa técnica funcionará bem se o arquivo de modelagem do documento de arquitetura tiver que ser distribuído com outros arquivos de modelagem para consumo dos leitores. Se, em vez disso, o documento de arquitetura for publicado na Web, siga uma das outras abordagens indicadas)

Considerações de Desenvolvimento de Equipe e Gerenciamento de Modelo

Esta seção apresentará algumas considerações de quando e por que você poderá optar por particionar um modelo em vários arquivos de modelagem. Um tratamento mais abrangente desses problemas é encontrado na ajuda on-line do RSx. Presume-se aqui que o leitor esteja relativamente familiarizado com os conceitos de desenvolvimento paralelo e com a noção de mesclagem das alterações que foram feitas paralelamente às múltiplas cópias de um artefato.

Em primeiro lugar, uma revisão rápida: na seção “Conceitos Básicos e Terminologia”, discutimos os diversos tipos de modelo, como Caso de Uso, Análise e Design, conforme reconhecidos pelo RUP. Foram dados exemplos para ilustrar que no RSx…

se você estiver construindo vários aplicativos, poderá ter vários modelos de cada tipo (por exemplo, vários modelos de caso de uso, vários modelos de análise e assim por diante)

um modelo (no sentido lógico) pode permanecer como um ou mais arquivos de modelagem, por exemplo, “o modelo de design para aplicativo ‘X’”) pode permanecer como um único arquivo de modelagem ou como uma coleção de vários arquivos de modelagem.

Modelos de Particionamento

As técnicas para particionar modelos em vários arquivos de modelagem são abordadas na Ajuda on-line do RSx e não são tratadas aqui. Neste documento, estamos interessados em quando e por que você particionaria. Há duas circunstâncias sob as quais você poderá optar por manter um determinado modelo como diversos arquivos de modelagem:

O modelo chegou a um tamanho que não pode ser gerenciado, ou sua estrutura de pacotes atingiu uma profundidade que não pode ser gerenciada[15]

Você começa a ter um número muito grande de entregas simultâneas de alterações em um arquivo de modelagem, resultando na necessidade de executar mesclagens com mais de 2 contribuidores de alteração[16]. (Se você achar essa instrução pouco clara, continue lendo.)

Modelagem nas Equipes

Quando a sua política de gerenciamento de configuração permitir o desenvolvimento paralelo de modelos[17], alterações descoordenadas serão feitas no arquivo (e de fato no modelo lógico ou no subconjunto de modelo que o arquivo representa). Em algum ponto, as alterações precisarão ser

[15] O particionamento é exigido principalmente quando os arquivos se tornam muito grandes para as máquinas que estão em uso na comunidade de usuários. Por exemplo, um modelo que atinge 30 MB no disco se torna muito difícil de trabalhar no dia a dia em uma máquina com 1 GB de RAM. Em tal máquina, seria desejável particionar o modelo com o objetivo de tentar manter o tempo todo na RAM de 5 a 10 MB equivalente em modelos. Uma alternativa é fazer upgrade da RAM (uma solução de custo relativamente baixo, mas que funciona muito bem -- uma máquina de 2 GB sem arquivo de troca executa muito mais rapidamente quase todas as operações do Eclipse, desse modo, fazendo com que modelos muito grandes também sejam executados muito bem.)

[16] A ferramenta de mesclagem de modelo RSx suporta mesclagens com no máximo 3 contribuidores: dois de ‘alteração’ e um de ‘base’ (aka o ascendente comum). Quando houver mais de 2 contribuidores de ‘alteração’ com o qual lidar, as mesclagens começarão a cair em forma de cascata e, a partir desse ponto, um dos dois contribuidores de ‘alteração’ será o conjunto de resultados das mesclagens concluídas anteriormente na sessão.

[17] Exemplos de políticas de desenvolvimento “paralelo”:

um arquivo de modelagem pode ter registro de saída de acesso não exclusivo

um arquivo de modelagem é trabalhado em paralelo nos fluxos de desenvolvimento de vários profissionais)

mescladas. Se for confirmado que algumas alterações entram em conflito com outra, a mesclagem será considerada "**não trivial**" porque alguém deverá tomar decisões sobre quais alterações em conflito devem "vencer". (Uma mesclagem 'trivial' é aquela em que as alterações descoordenadas não estão em conflito e o mecanismo de mesclagem de modelo pode executar as mesclagens sem intervenção humana.)
Mesclagens não triviais podem ser difíceis de empreender. Como elas podem ser minimizadas?

Você tem duas armas básicas para evitar conflitos e as mesclagens não triviais resultantes. Uma delas é a "arquitetura sólida". A outra, a "propriedade sólida". Elas andam juntas: arquitetura sólida *permite* propriedade sólida.

Arma nº 1: Arquitetura Sólida

"Arquitetura sólida" neste contexto se refere principalmente à decomposição. Os *princípios* de decomposição arquitetural que se aplicam aqui são os mesmos que conduzem o Desenvolvimento Orientado a Objetos, o Design Baseado em Componentes e as Arquiteturas Orientadas a Serviços:

Tente ao máximo separar as funções de negócios

Agrupe as coisas que devem permanecer rigorosamente ligadas. Isole esses agrupamentos uns dos outros

Se a decomposição resultante tiver um grande número de 'grãos', dependendo do modelo de equipe (lembre-se de que arquitetura sólida e propriedade sólida andam juntas), você talvez queira iniciar o agrupamento desses grãos em agregações de alta afinidade (o que, em termos de modelagem, significa Pacotes UML)

Haverá coisas que *sempre* deverão ser obtidas por muitas (e, em alguns casos, todas) unidades de decomposição. Agrupe essas coisas em um pacote 'comum' e planeje cada iteração de desenvolvimento para que inclua um pouco de "cascata" no início da iteração cujo foco esteja na estabilização da matéria "comum".

Existe também um elemento de tempo. À medida que você muda da compreensão mais abstrata para mais concreta de uma solução, seu sentido da melhor organização da arquitetura (e modelo) evoluirá. Planeje uma atividade de refatoração (reorganização) do modelo na transição de cada fase para a seguinte (análise de negócios, requisitos, análise de aplicativo, design de alto nível, ...).

Se você examinar sua solução e tudo parecer altamente interdependente e rigidamente ligado, ou sua arquitetura precisa funcionar ou há algo a respeito da natureza de seu domínio de problema que significa que você na verdade não pode decompor o problema. Em qualquer dos casos, você tem algumas opções:

Decida por designar o projeto a uma equipe bem pequena que compartilha um espaço físico e se comunica muito ativamente com outra com relação a todas as alterações que elas fazem que possam afetar outros artefatos.

Esteja preparado para executar muitas mesclagens não triviais

XDE/Rose

Se você for um usuário Rose ou XDE, talvez tenha tentado fazer mesclagens não triviais de modelos. A boa notícia é que mesclagens não triviais no RSx são bem menos difíceis. Uma diferença principal é que o RS*x* opera em uma *malha* de arquivos de modelagem relacionados, contrariamente a uma *hierarquia* de arquivos ou subunidades em pétala. Essa diferença crítica significa que suportamos decomposição lógica de alto nível muito melhor no nível físico no RS*x*.

Suportamos também mesclagem de comparação com ferramenta melhor – de fato, com uma experiência de mesclagem **demasiadamente** melhor.

- O mecanismo de mesclagem de backend no RS*x* é 10.000 (sim, dez mil) vezes mais rápido do que o backend XDE.
- Implementa também várias tecnologias, como “deltas de composição” (um mecanismo de agrupamento que classifica as alterações por diagrama e agrupa grandes ações de diagramação para permitir operações de grupo e/ou atômicas), “proteção de integridade de modelo”, “atomicidade” e “processamento delta em cascata”. Isso reduz a probabilidade de corromper um arquivo por mesclagem, quase no mesmo nível da simples edição do arquivo.
- Agora existe uma verdadeira GUI de mesclagem de diagrama visual para resolver conflitos de layout/cosmético

Arma nº 2: Propriedade Sólida

Tendo estabelecido a decomposição de arquitetura sólida, ela provará ser razoavelmente direta (dispensa conhecimento especializado) para mapear a propriedade “sólida” dos componentes de arquitetura para profissionais individuais ou equipes pequenas. Quando cada pacote lógico (ou ramificação) em um modelo pode ser trabalhado exclusivamente por um profissional, as mesclagens efetuadas com esse modelo serão em sua maioria triviais (independentemente de o modelo estar armazenado como um único arquivo de modelagem ou como vários). O mesmo pode ser dito se cada ramificação puder ser trabalhada exclusivamente por uma equipe pequena cujos membros estejam dispostos a se comunicar bastante e de modo informal sobre suas atividades.

A mesclagem não trivial pode ser evitada com a divisão dos modelos em vários arquivos de modelagem? Em uma palavra: “não”. **Interdependências de arquitetura são um fenômeno lógico, não físico**. Quando você particiona um modelo em vários arquivos de modelagem, as representações das interdependências de elementos simplesmente se tornam referências de arquivo cruzado, em vez de referências em arquivo. Você não facilita em nada a resolução de conflitos (na verdade, torna-a mais difícil). E ao introduzir referências de arquivo cruzado, você introduz possíveis pontos de interrupção (consulte barra lateral).

Barra Lateral: Referências de Arquivo Cruzado de Modelagem

Sempre que dois elementos de modelagem residirem em arquivos de modelagem diferentes e você criar um relacionamento entre eles, crie uma "referência de arquivo cruzado de modelagem". Como os arquivos de modelagem (arquivos .emx) são expostos no sistema de arquivo host do OS e podem ser movidos, renomeados ou modificados de alguma forma fora do ambiente Eclipse, essas referências representam possíveis pontos de interrupção. Entretanto, desde que os arquivos de modelagem sejam sempre modificados e tenham suas alterações gerenciadas por meio do ambiente Eclipse, e você respeite as seguintes diretrizes, não deverá ter interrupções.

Sempre que você trabalhar com (editar) um 'fechamento' de arquivos de modelagem (isto é, uma coleção de arquivos de modelagem que façam referência uns aos outros) tenha presente no espaço de trabalho todos os arquivos de modelagem desse fechamento. Isso não implica rigorosamente em que todos os arquivos de modelagem de um fechamento devem residir no mesmo projeto, mas o uso de um único projeto geralmente garante que todos os modelos estarão presentes, já que, em fluxos de trabalho CM típicos, todos os arquivos de modelo andam juntos no mesmo projeto.

Vamos recapitular então:

Se você precisar de arquitetura sólida, ou se a tiver mas faltar propriedade sólida, as mesclagens não triviais serão tão freqüentes que nenhuma quantidade de particionamento de modelos poderá ajudar.

Se você tiver arquitetura sólida e propriedade sólida, reduzirá grandemente (mas não eliminará) a freqüência de mesclagens não triviais. Você não a eliminará porque sempre haverá interdependências de componentes. Os elementos 'comuns' anteriormente mencionados são um exemplo, embora dificilmente seja o único.

Particionar modelos em vários arquivos não é nem de perto tão importante quanto estruturar modelos logicamente para permitir que vários profissionais trabalhem em um arquivo de modelagem em paralelo sem introduzir conflito de alterações.

A boa notícia é que o RS*x* controla a mesclagem de modelos de modo bem mais rápido e mais eficaz do que qualquer outra ferramenta de modelagem disponível.

E quando eu deverei particionar os modelos? Tente evitar isso, exceto nos casos em que o tamanho absoluto de um modelo tiver começado a testar os limites do hardware e os arquivos de modelagem criados por você normalmente puderem ser trabalhados *exclusivamente* (isto é, somente um membro da equipe tem a saída do arquivo registrada a qualquer momento) **e** *isoladamente* (isto é, a maioria das alterações pode ser feita no arquivo sem exigir também o acesso a outros arquivos que contêm elementos de modelo relacionados).

Posfácio: Alterações Entre as Versões 6.x e 7.x do RSx

Quando o RSx foi liberado pela primeira vez (versões 6.x), ele não suportava o conceito de ‘subunidade’ como suportado pelo Rose e pelo XDE (subunidade é um arquivo de modelagem que contém o subconjunto de um modelo e é ‘transparente’ no sentido de não aparecer na visualização lógica do modelo que você vê na visualização do explorador de modelos). Em vez disso, o particionamento de modelos era limitado à definição de vários modelos de nível superior (“nível superior” no sentido de que cada arquivo de modelagem apareceria como uma entrada de nível superior e separada, na visualização do explorador de modelos).

O RSx fornece também um recurso que permite selecionar um Pacote UML de um modelo existente e o recurso “Criar um Modelo” com base nesse Pacote. Você pode pensar nisso como “cortar” o Pacote para criar um novo arquivo de modelagem. O resultado é que, na visualização do explorador de modelos, o Pacote parece se tornar um novo modelo lógico de nível superior. A visibilidade da estrutura de retenção hierárquica original é perdida, mas sempre que você abrir um arquivo de modelagem no qual os Pacotes foram “cortados” dessa maneira, todos os modelos “cortados” também serão abertos. Isso pode resultar em registros de saída desnecessários, bem como congestionamento na visualização do explorador de modelos e aparecimento de várias guias na área de janela do editor que, juntos, dificultam a navegação.

A inclusão de suporte de subunidade nas versões 7.0 do RSx permite que você particione modelos em vários arquivos, sem a perda da visibilidade na estrutura hierárquica e, junto com a eliminação das guias do “editor de modelo” na área de janela do editor, controla os outros problemas de navegação.

Entretanto, houve uma lição na estruturação de modelos a ser aprendida com a experiência da versão 6.x anterior que ainda se aplica atualmente. Conforme observado anteriormente, a abertura de um arquivo de modelagem no qual Pacotes foram cortados resultaria na abertura de todos os modelos cortados. Isso poderia ser evitado com o planejamento antecipado de uma estratégia de particionamento, em vez do uso do recurso “Criar um Modelo”.

Aqueles que seguem o recurso “Criar um Modelo” em geral começariam com um único arquivo de modelagem, e criariam Pacotes para representar a organização da arquitetura de solução. Por exemplo, poderá haver um Pacote para cada componente ou subsistema principal da solução (em que ‘componente’ e ‘subsistema’ refletem o uso informal daqueles termos que rastreiam os dias anteriores de cálculo, não suas definições semânticas UML formais). Pode haver também Pacotes para componentes “comuns”, de “utilitário” ou de “estrutura” . Por isso, conforme tal modelo crescia, os Pacotes que correspondiam aos componentes específicos do aplicativo podiam ser “cortados” como arquivos de modelagem separados, de modo que as equipes que criavam esses componentes podiam (teoricamente) trabalhar isoladamente com esses arquivos de modelo. Mas, na prática, a abertura de qualquer um desses arquivos de modelo específicos do componente resulta na abertura do modelo ‘mestre’ original (no qual as partes ‘comuns’ residem), e isso por sua vez abre todos os outros modelos específicos do componente de aplicativo. Os resultados são os registros de saída desnecessários, mencionados anteriormente, e a dificuldade de navegação.

Uma abordagem melhor seria planejar antes para definir um modelo separado para cada componente específico do aplicativo, junto com um modelo para os componentes ‘comuns’ (ou talvez, vários modelos que definem camadas sucessivas de componentes comuns/reutilizáveis, isto é, refletindo as camadas de abstração da arquitetura de implementação). Dessa forma, todo modelo de componente específico do aplicativo poderia ser aberto pela equipe que o possui, e somente os modelos ‘comuns’ dos quais esse modelo específico do aplicativo depende precisariam ser abertos também.