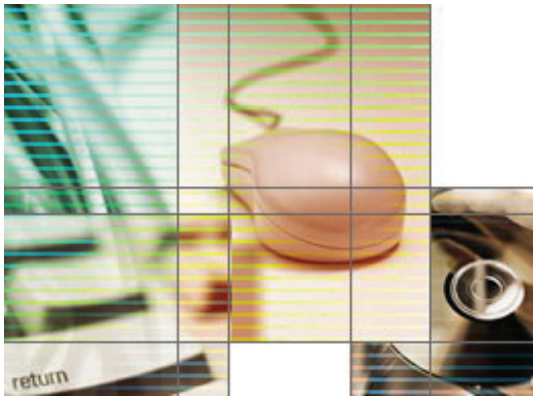


Rational Unified Process for Systems Engineering

Parte III: Análise e Design de Requisitos

por Murray Cantor
Principal Engineer
Rational Brand Services
IBM Software Group

Na publicação de agosto do The Rational Edge, começamos uma série de três partes para fornecer uma visão geral da evolução mais recente do Rational Unified Process for Systems Engineering ® ou RUP SE ®. O RUP SE é um aplicativo da estrutura do processo de engenharia de software Rational Unified Process, ou RUP®. Os usuários do RUP devem observar se o RUP Plug-In for SE disponível atualmente é o RUP SE v1 Plug-In, o qual foi disponibilizado em 2002.



A Parte I incluiu uma discussão de sistemas, os desafios que o desenvolvedor de sistemas modernos enfrentam e como o RUP SE os determina, as técnicas de especificação de requisitos e modelagem baseadas no RUP SE Unified Modeling Language (UML) e o uso da semântica de UML. A Parte II focalizará a arquitetura do sistema e apresentará a estrutura da arquitetura do RUP SE, que descreve as partes internas do sistema com base em vários pontos de vista. Agora, na Parte III, abordaremos a análise e o fluxo decrescente de requisitos e as especificações para elementos da estrutura do RUP SE. Isso incluirá uma descrição do Joint Realization Method, uma técnica nova para derivar juntamente a especificação de elementos de arquitetura em vários pontos de vista. Também incluiremos uma breve discussão do desenvolvimento de sistema com o RUP SE.

Nota do editor: O RUP SE v1 Plug-In se tornou disponível em 2002 e a v2 desse plug-in tornou-se disponível em junho de 2003. Embora as informações nesta série sejam consistentes com a v2, os artigos realmente abordam algumas

possíveis extensões para a estrutura do processo. Observe que o RUP SE Plug-In -- v1 e v2 -- é transferível por download do IBM Rational Developer Network (http://www.rational.net; autorização necessária).

RUP SE e Requisitos

Seguindo uma prática comum do sistema, o RUP SE determina dois tipos de requisitos:

1. Requisitos comportamentais -- O que o sistema faz para preencher sua função na empresa. No RUP SE, o comportamento de um sistema é capturado por seus casos de uso e suas análises em serviços. Os casos de uso e os serviços podem ter os requisitos de desempenho associados.
2. Requisitos suplementares -- Requisitos não funcionais, incluindo metas de design (por exemplo, confiabilidade ou custo de propriedade) e atributos de sistema (por exemplo, capacidade de dados ou peso total).

O RUP SE também oferece um padrão de processo para derivar requisitos para elementos de arquitetura:

1. Determine os requisitos ou as especificações de caixa preta para um determinado elemento de modelo.
2. Decomponha esse modelo em elementos de caixa branca, atribuindo funções e responsabilidades a esses elementos.
3. Estude como os elementos colaboram para atender juntamente os requisitos de caixa preta. Isso geralmente envolve algum formato do diagrama de colaboração.
4. Sintetize a análise da colaboração para determinar os requisitos de caixa preta para os elementos.

Este padrão de processo é bem conhecido[1] e é particularmente interessante que Friedenthal et al. o adotou em seu Object Oriented System Engineering Method (OOSEM).[2]

Portanto, especificação do sistema significa definir casos de uso, serviços do sistema e requisitos complementares que, se atendidos, resultarão em um sistema que atende a seu objetivo ou missão de negócios. O RUP SE distingue entre requisitos alocados e derivados. Um requisito será alocado se um requisito de caixa preta for designado a um elemento de caixa branca. Um requisito será derivado se ele for determinado estudando como o elemento de caixa branca colabora com outros para atender a um requisito de caixa preta. Os requisitos comportamentais e suplementares poderão ser derivados. Observe que os requisitos derivados levam em consideração a função do elemento de arquitetura no design do sistema.

Considere o seguinte exemplo. A maioria dos automóveis tem diferenciais, os

dispositivos que conectam o eixo de acionamento aos eixos que permitem que as rodas motrizes girem em diferentes velocidades sempre que o automóvel faz uma curva. Esta função é necessária porque a roda interna deve girar mais lentamente que a roda externa para que as duas rodas mantenham a tração. Se uma das rodas perder a tração, o automóvel poderá derrapar e, possivelmente, capotar. Apesar disso, não há requisito do sistema do automóvel para um diferencial. O requisito é que o automóvel mantenha a tração quando atravessar uma curso, o que pode ser realizado de várias maneiras que não incluam um diferencial. Por exemplo, qualquer uma dessas alternativas podem funcionar:

- Uma única roda motriz (como nos veículos de três rodas que às vezes aparecem em filmes de ficção científica).
- Dois motores, um por roda motriz, com algum tipo de solução de dirigir por ligação.

A convenção do diferencial prevalece, pois ela é superior do ponto de vista da engenharia (isto é, faz um melhor trabalho de atender a uma variedade de requisitos, além de controle da tração, como manter a estabilidade geral, otimizar o volume interior e gerenciar custo de manutenção de materiais) ou da restrição de design que o engenheiro deve tirar proveito de engenharia excelente anterior.

Agora, como o diferencial não é um elemento obrigatório do automóvel, não há mecanismo para atribuir requisitos do sistema ao diferencial. Em vez disso, o diferencial reproduz uma função em colaboração com outros elementos do automóvel (direção, freios, etc.), de forma que eles possam juntamente atender ao comportamento exibido do automóvel para fazer uma curva com segurança. O comportamento do diferencial, como "ajustar a velocidade das rodas", é derivado do requisito do sistema e da função que o diferencial reproduz. Esse comportamento é derivado, não alocado.

Além disso, o diferencial deve atender aos requisitos suplementares derivados para suportar os requisitos suplementares definidos do sistema. Por exemplo, o diferencial terá uma provisão de peso e volume, além de uma medida de confiabilidade.

O uso de requisitos derivados para subsistemas que colaboram com a realização de casos de uso é denominado decomposição lógica. De maneira similar, a determinação de requisitos do subsistema por alocação é denominada decomposição funcional. Geralmente, a decomposição lógica é essencial para se obter sistemas de qualidade.[3]

Ele indica que os requisitos do sistema são derivados de uma compreensão dos serviços corporativos e da função que o sistema reproduz na empresa. No modelo de análise, os elementos da arquitetura do sistema são subsistemas, localidades e processos, conforme descrito na seção Arquitetura do Sistema, na Parte II desta série. É na disciplina de análise de requisitos que são determinados os requisitos para cada um desses tipos de elementos de arquitetura.

Por exemplo, com o modelo de negócio no lugar, o RUP SE sugere que você particione a empresa no sistema e em seus agentes para derivar requisitos do sistema. Em seguida, para determinar os requisitos do sistema, um analista pode estudar como o sistema e seus agentes colaboram para atender aos requisitos dos negócios.

As seções a seguir descrevem a abordagem do RUP SE para derivação de requisitos funcionais para sistemas e elementos do modelo de análise.

Derivando Requisitos Funcionais por Meio do Fluxo Decrescente de Caso de Uso

O fluxo decrescente de caso de uso é uma atividade para derivação de requisitos funcionais de sistemas e seus elementos. O fluxo decrescente pode ser aplicado para incluir detalhe em um nível de modelo ou para especificar elementos em um nível de modelo mais baixo. Por exemplo, fluxo decrescente pode ser utilizado para determinar serviços do sistema no nível de contexto. De maneira similar, ele pode ser utilizado no nível de análise para identificar serviços do subsistema e dividir os subsistema em subsistemas adicionais.

É importante observar que o fluxo decrescente pode ser aplicado recursivamente -- em outras palavras, os elementos da caixa branca tornam-se elementos da baixa preta no próximo aplicativo. Isso permite que a equipe discuta sobre grandes sistemas no nível apropriado de especificidade. O aplicativo repetido da atividade de fluxo decrescente permite que as equipes incluam detalhe enquanto gerenciam um nível consistente de abstração. Ele também permite design simultâneo; ou seja, cada entidade de caixa branca pode ser especificada suficientemente para ser tratada como uma entidade de caixa preta para que equipes distintas criem design adicional. Você pode utilizar essa abordagem não apenas para derivar requisitos para elementos do modelo de análise, mas também, com um pouco de modificação, determinar requisitos do sistema provenientes de requisitos de negócios.

Executar o fluxo decrescente da maneira hierárquica que descrevemos anteriormente resultará em um relacionamento interessante entre serviços e casos de uso: os serviços de caixa preta tornam-se casos de uso de caixa branca. Os casos de uso descrevem como uma entidade e os elementos em seu contexto colaboram para preencher algum propósito. Aqui, o objetivo do fluxo decrescente de caso de uso é suportar a entrega de um serviço do sistema. A realização do serviço consiste em cenários de caso de uso. Para cada subsistema UML, você pode criar um diagrama de contexto que mostre os agentes do sistema com os quais colaboradores do subsistema, bem como os subsistemas de mesmo nível com os quais ele compartilha um relacionamento de dependência (estes são os apelidos para a empresa e os agentes internos discutidos em Especificação do Sistema na Parte I). Do ponto de vista do subsistema, a realização do serviço é exatamente como ele colaborará com seus agentes para realizar suas funções. Este é exatamente um cenário de caso de uso. Observe que o fluxo decrescente não altera a heurística de valor comum da análise de caso de uso.

Os casos de uso no fluxo decrescente fornecem valor para a entidade de caixa preta e não necessariamente para algum dos agentes participantes.

Realização Simples

O fluxo decrescente de caso de uso é uma extensão da realização de caso de uso, uma prática elementar da análise de objeto. A realização de casos de uso consiste na localização de classes que participam da realização de um cenário de caso de uso e da descoberta de como os objetos das várias classes colaboram. A realização inclui especificar a ordem das mensagens dos objetos que são transmitidas durante a colaboração e ela é capturada em uma seqüência ou diagrama de colaboração. Na verdade, ao criar diagramas de seqüência, você pode descobrir as mensagens que uma operação de classe deve fornecer para que seus objetos possam participar na realização de seus casos de uso. No RUP SE, esta noção de realização é estendida de várias maneiras. Primeiro, a realização é aplicada a níveis de modelo superiores aos níveis de design. Por exemplo, os resultados do fluxo decrescente aplicado entre a empresa e o sistema retornam uma identificação de serviços do sistema. Se aplicado entre o sistema e seus elementos de modelo, o fluxo decrescente resulta em:

- Uma pesquisa de opinião de caso de uso para subsistemas.
- Identificação de serviços de subsistema e interfaces.
- Uma pesquisa de opinião de serviços de subsistema hospedado e/ou interfaces suportadas para localidades.

A idéia de estender a realização de casos de uso para subsistemas UML não é nova. Por exemplo, as realizações de subsistemas UML muitas vezes são referidas como Diagramas de Interação de Arquitetura.

Aqui são descritas as etapas do fluxo decrescente para criar o diagrama de contexto do sistema e identificar serviços do sistema:

1. Modelar uma caixa branca corporativa como um conjunto de sistemas de colaboração.
2. Modelar como os sistemas colaboram para realizar serviços corporativos, missão, e assim por diante.
3. Criar um diagrama de contexto para o sistema.
4. Determinar agentes (isto é, entidades que colaborem com o sistema).
5. Identificar entidades de E/S.
6. Agregar colaborações similares entre o sistema e seus agentes em casos de uso.
7. Incluir detalhe de caso de uso: desempenho, pré e pós-condições, e assim por diante.
8. Identificar serviços do sistema -- o que o sistema faz para dar suporte a seus casos de uso; agregar etapas similares de caixa branca.

9. Incluir atributos de sistema de sua análise de necessidades corporativas.

Quando uma realização consiste em um tipo de elemento de caixa branca, como classes ou subsistemas UML, chamamos isso de realização simples. Um exemplo é o fluxo decrescente da empresa para o sistema, como descrito a seguir.

Procedimento 1: Realização Coletiva

Em versões futuras do RUP SE, a realização simples descrita anteriormente será estendida à realização coletiva: analisando como os elementos de vários pontos de vista colaboram na realização de um serviço. Por exemplo, na realização coletiva, o fluxo decrescente pode consistir na determinação simultânea da colaboração de elementos informativos, físicos e lógicos.

A realização coletiva consiste nos seguintes procedimentos:

1. Escolha os pontos de vista participantes. O ponto de vista lógico é obrigatório.
2. Para cada etapa da caixa branca na realização de um serviço de caixa preta, você deve:
   - Especificar elemento lógico que a executa.
   - Modelar como os pontos de vista adicionais participam. Por exemplo, você pode incluir:

     -Ponto de vista físico -- Especifique a localidade de hosting; se houver duas localidades, decomponha em duas etapas.
     -Ponto de vista do processo -- Especifique o processo em execução; se houver dois processos, decomponha em duas etapas. -Ponto de vista de informações -- Especifique qual elemento do esquema de dados suporta a manipulação de qualquer informação utilizada.

     Em todo esse processo, aplique a seguinte regra de realização coletiva: Se uma etapa específica da caixa branca do elemento lógico exigir mais de um elemento dos outros pontos de vista, divida essa etapa em etapas adicionais, de forma que cada etapa exija exatamente um elemento de caixa branca de cada ponto de vista.

3. Criar diagramas de interação para cada ponto de vista:
   - Diagrama de interação de arquitetura
   - Diagramas de interação de localidade
   - Diagramas de interação de processo

4. Prover requisitos suplementares para desempenho, exatidão, e assim por diante, para cada etapa; avaliar/confirmar com diagramas de interação.

Procedimento 2: Especificar Recursos com Realização Coletiva

A realização coletiva tem uma variedade de aplicativos. Por exemplo, ela pode ser utilizada para fluxo decrescente que vai do sistema até a visualização lógica e de trabalhador para discutir sobre decisões de automação. Ou então, ela pode ser utilizada para fluxo decrescente que vai do sistema até elementos de processo, físicos e lógicos (esta aplicação é descrita em mais detalhes a seguir). Para não abordar especificações referentes aos recursos físicos do sistema, é necessário:

1. Desenvolver visualizações de nível de modelo de análise inicial (caixa branca do sistema). Para fazer isso:
   - Utilize métodos de análise orientados a objetos para a visualização lógica.
   - Aplique considerações físicas para a visualização da localidade.
2. Utilizar realização coletiva para modelar cada (arquiteturalmente significativa) especificação de serviço do sistema, incluindo:
   - Etapas de colaboração para subsistemas UML.
   - Localidades de hosting.
   - Processo em execução.
3. Capturar os requisitos de desempenho da caixa branca -- em outras palavras, a provisão dos requisitos de desempenho da caixa preta para as etapas da caixa branca. Para fazer isso:
   - Identifique os casos de uso do subsistema UML; em outras palavras, para cada subsistema, identifique os serviços do sistema que envolvem esse subsistema.
   - Para cada subsistema, identifique seus serviços de aplicar métodos de agregação em mensagens em colaboração.
   - Para cada localidade, crie uma pesquisa de opinião de serviços do subsistema host.
   - Para cada processo, crie uma pesquisa de opinião de serviços do subsistema executados.
4. Documentar a rastreabilidade entre casos de uso do sistema e do subsistema e/ou serviços do sistema e subsistema.

Procedimento 3: Fluxo Decrescente do Nível de Modelo de Contexto até de Análise

A designação de etapas de caixa branca para subsistemas, localidades e processos envolve um conjunto de decisões de design. Cada decisão inclui

detalhes para a função que cada elemento de análise reproduz no design do sistema geral. No processo de fazer designações, a equipe pode decidir refatorar o design, deslocar responsabilidades de um elemento para outro em uma determinada visualização. Além disso, observe que o fluxo decrescente oferece a oportunidade de incluir um nível adequado de detalhe e de refatorar as funções e as responsabilidades do subsistema, da localidade e do processo.

A Tabela 1 mostra um exemplo de fluxo decrescente de caixa branca para o serviço do sistema "Fechando venda com cartão de crédito", utilizando o subsistema (Figura 6) e o modelo de localidade 1 (Figura 8) para um sistema de venda no varejo pela Internet.

Tabela 1: Tabela de Realização Coletiva

System Actor Action	Black Box Budgeted Requirements	Worker Action	Subsystem Action	White Box Budgeted Requirements	Locality	Process
THE CUSTOMER PROVIDES A CREDIT CARD.	30 seconds	Clerk swipes credit card.	The Point of Sale terminal requests that Credit Card Services provide validation.	.5	Point of Sale Terminal	Terminal
			Credit card services queries bank credit card system.	28 seconds	Store Server	Order Processing
			If valid, Point of Sale prints receipt.	.5 seconds	Point of Sale Terminal	Terminal
		Clerks requests that customer sign receipt.				

A próxima etapa é determinar os casos de uso e o contexto do subsistema UML. Uma visualização de contexto do subsistema UML, como um contexto de sistema, é constituída do subsistema, de seus agentes e de quaisquer entidades relevantes de E/S. Para um subsistema, seus agentes podem consistir em seus subsistemas de mesmo nível e, possivelmente, agentes de sistema. A Figura 1 fornece um exemplo de diagrama de contexto do subsistema.

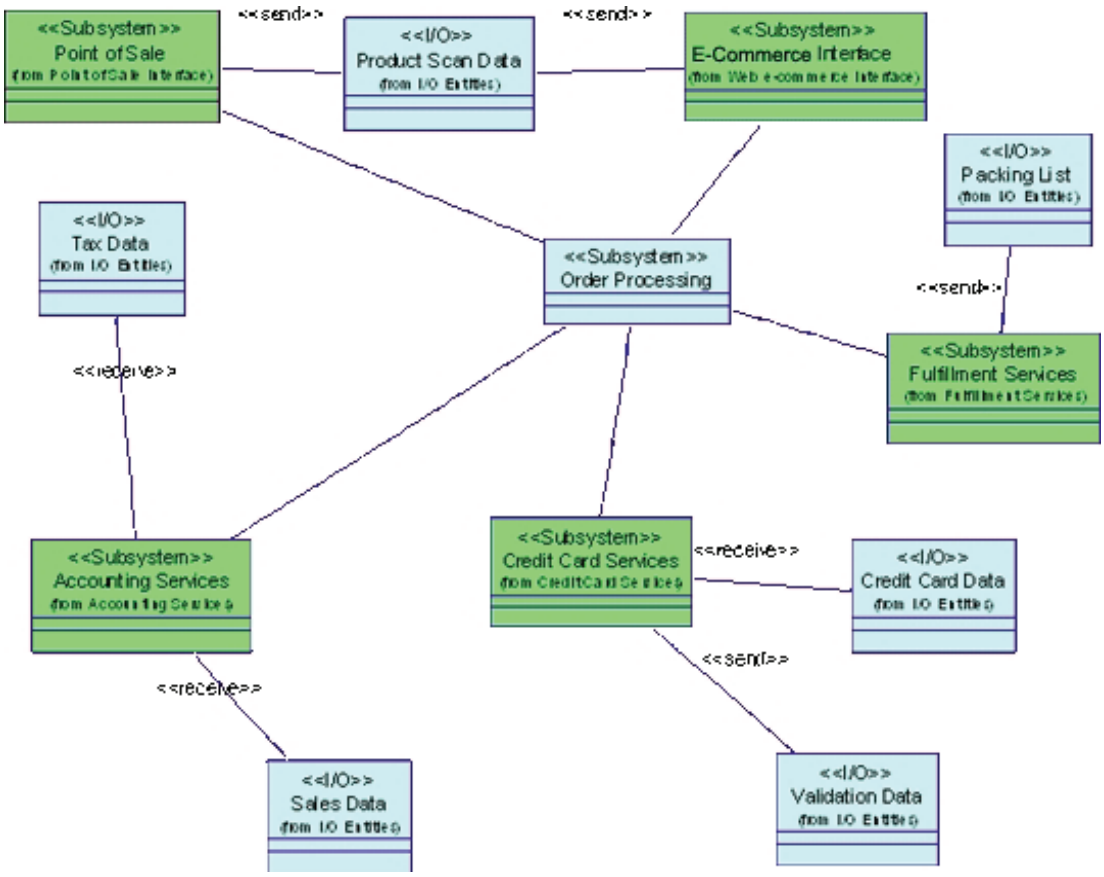


Figura 1: Diagrama de Contexto do Subsistema

Lembre-se que um caso de uso descreve como um sistema e seus agentes colaboram para oferecer um serviço de valor. Para um subsistema, o serviço de valor é o serviço do sistema em si. Ele mostra que para cada subsistema, seus casos de uso são exatamente os serviços do sistema em que ele colabora. Se você for particionar o esforço de desenvolvimento ao longo dos limites do subsistema ou como uma base para desenvolver etapas de teste, será útil manter uma pesquisa de opinião de caso de uso para subsistemas.

Você pode localizar serviços do subsistema classificando as etapas da caixa branca de serviço por subsistema. Para cada subsistema, classifique as etapas da caixa branca e agregue etapas similares, como mostrado na Tabela 2. Isso resulta na especificação de serviços fornecidos por subsistema.

Tabela 2: Exemplo de Pesquisa de Opinião dos Serviços Hospedados de Localidade

Nome da Localidade: serviços Processamento da Loja			
Responsabilidade de Localidade: Esta localidade hospeda transações e contabilidade das vendas da loja central. Ela fornece a interface para o processamento do escritório central e de cartão de crédito.			
Serviço do Subsistema	Subsistema	Serviço do Sistema	Texto da Caixa Branca do Serviço

Iniciar Venda com Cartão de Crédito	Processamento do Pedido	Digitar uma venda	Processamento de Pedido inicia uma lista de vendas
Incluir dados dados do produto	Processamento do pedido	Digitar uma venda	Os dados do scanner são enviados para processamento do pedido, que recupera nome, preço e status de impostos de uma lista de inventários e de atualizações
Calcular Total	Processamento do Pedido	Digitar uma venda	Processamento do Pedido soma o preço e calcula os impostos.

Depois de determinar os serviços do subsistema, você pode classificar o conjunto de serviços de subsistema por localidade ou por processo. A pesquisa de opinião dos serviços hospedados para cada localidade expressa qual cálculo ocorre na localidade, bem como os requisitos de desempenho associados. Essas informações fornecem a entrada para a especificação de componentes físicos que serão implementados na localidade. De maneira similar, a pesquisa de opinião de serviços executados para cada processo funciona como entrada para a especificação de componentes de software. Estas atividades incluem as seguintes etapas no processo de realização coletiva:

- Para cada localidade, crie uma pesquisa de opinião de serviços hospedados (como aqueles mostrados na Tabela 2).
- Para cada processo, crie uma pesquisa de opinião de serviços executados.

Descreveremos a especificação dos componentes de forma mais completa a seguir.

Uma abordagem alternativa para associar serviços do subsistema a localidades é definir uma interface do subsistema que constitui serviços que são hospedados na localidade e, em seguida, associar essa interface à localidade. Essa abordagem tem o benefício de manter a associação de serviço para localidade completamente no modelo UML.

A descrição textual no fluxo de eventos de caixa branca também pode ser expressada como um conjunto de diagramas de seqüência ou de colaboração. Esses diagramas conduzem o tráfego entre elementos de análise: Cada diagrama é um diagrama de seqüência cujos objetos são classes de proxy para os elementos de análise. As mensagens são chamadas dos serviços de subsistema. As Figuras 2 e 3 mostram os diagramas de interação de subsistema e de localidade para o fluxo de eventos descrito na Tabela 2.

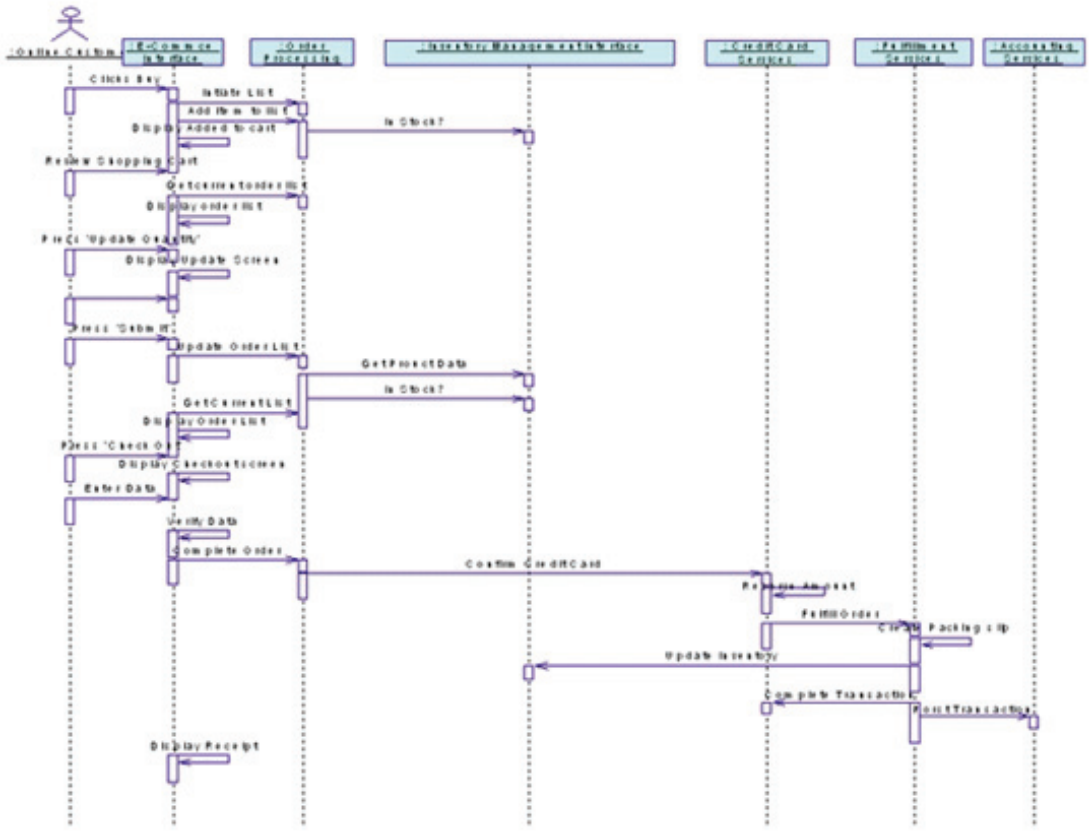


Figura 2: Exemplo de Diagrama de Interação de Subsistema
Clique para expandir

A Figura 2 fornece uma visão geral para o acoplamento e a coesão dos subsistemas. Essa visão geral pode ser utilizada para refatorar o design do subsistema; se houver tráfego intenso entre um par de subsistemas, por exemplo, poderá fazer sentido combiná-los. A Figura 3 é um exemplo de diagrama de interação de localidade.

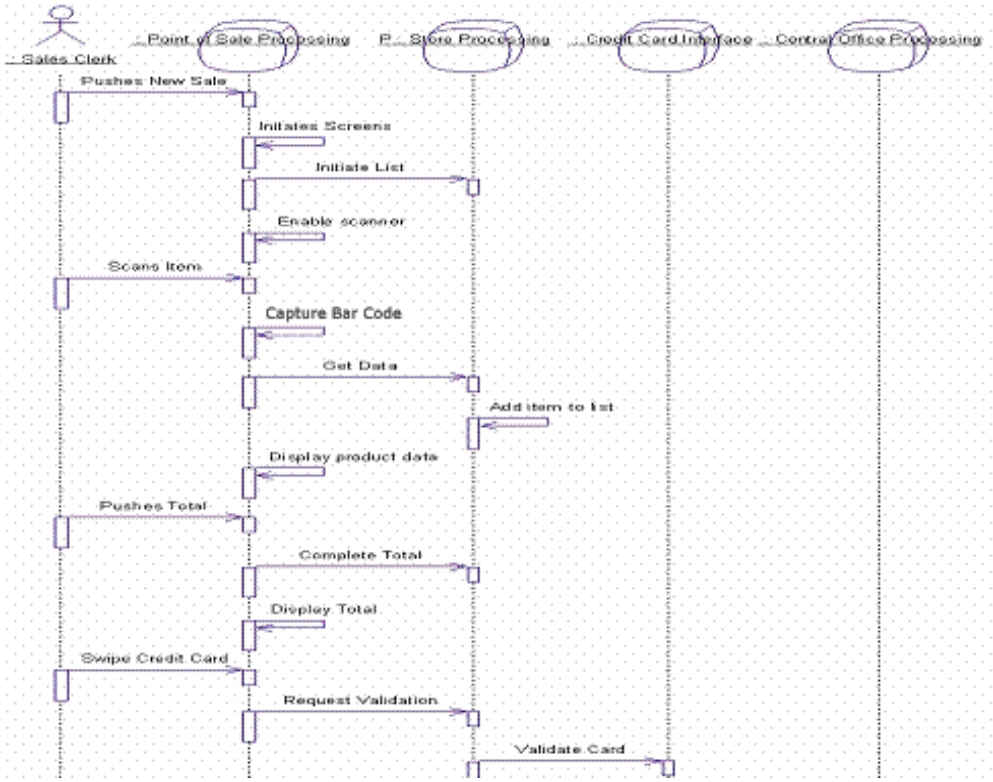


Figura 3: Exemplo de Diagrama de Interação de Localidade

O tráfego na Figura 3 mostra quais dados devem fluir entre as localidades. Essas informações são utilizadas para especificar associações entre localidades.

Flowdown de Requisitos Suplementares

Os requisitos suplementares são inicialmente capturados como atributos de classe do sistema ou valores com tag. Como parte do processo de análise, os arquitetos do sistema desenvolvem um diagrama de localidade inicial. A visualização de localidade é uma síntese das considerações não funcionais e fornece um contexto para determinar como requisitos não funcionais, como confiabilidade e capacidade, serão especificados.

A prática de engenharia padrão leva em consideração o orçamento de capacidade, taxas de defeitos permitidos e outros. Esse esforço resulta em um conjunto de requisitos suplementares derivados para cada elemento de localidade. As características de localidade são eterminadas a partir desses requisitos.

Especificação de Componente

Mover-se do nível de análise para o de design de uma arquitetura requer a determinação do design de componente de hardware e software. Essa especificaçãode nível de design consiste nos componentes a serem implementados: hardware, software e trabalhadores.

Os componentes de hardware são determinados pela análise das localidades, juntamente com suas características derivadas e serviços do subsistema hospedado. Com essas informações, as realizações de nível do descritor das localidades podem ser selecionadas. Os diagramas do nó Descritor especificam os componentes, os servidores, as estações de trabalho, os trabalhadores e assim por diante, sem opções específicas de tecnologias. A Figura 4 é um exemplo do diagrama de nó do descritor que realiza o diagrama de localidade mostrado na Figura 8. A localidade de atendimento é realizada como quatro componentes: um gateway de armazém, um sistema de correio/postagem e dois trabalhadores.

Os nós do descritor herdam características de suas localidades através de um processo de alocação ou de orçamento.

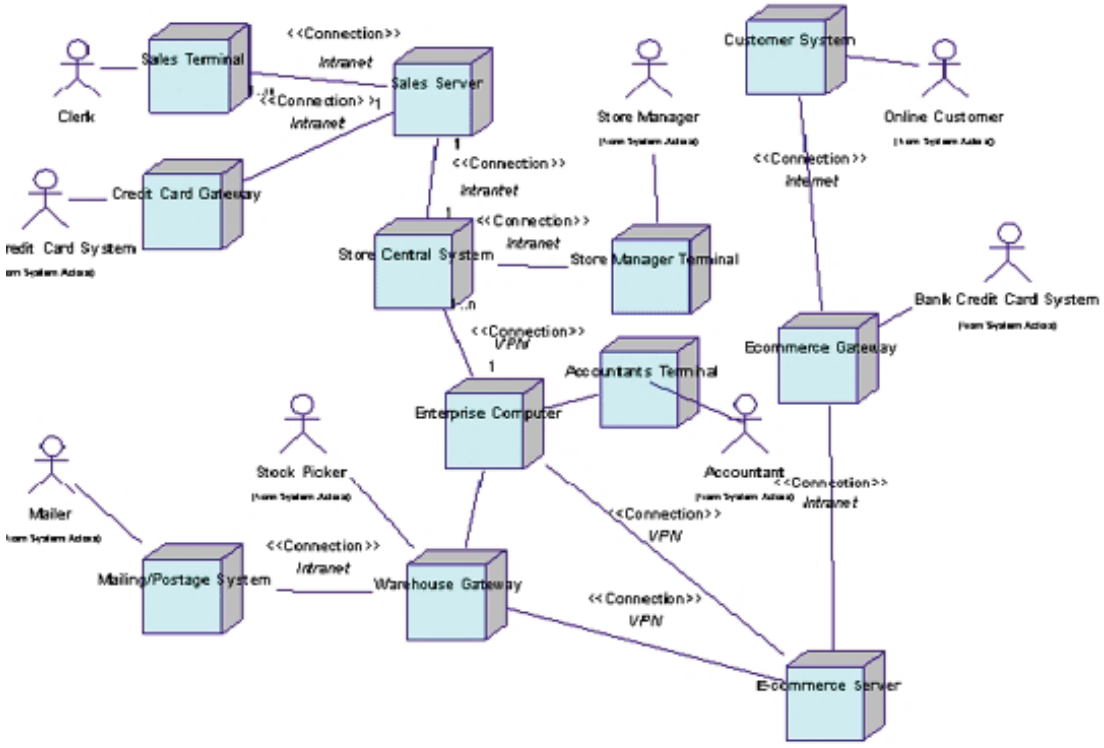


Figura 4: Exemplo de Diagrama de Nó do Descritor

Os componentes de hardware de implementação, em outras palavras, o conjunto de hardware implantado real, são determinados pela realização de negócios de custo/desempenho/capacidade a partir da visualização do descritor. Na realidade, um sistema pode ter mais de uma configuração de hardware, cada um atendendo pontos diferentes de preço/desempenho.

Os componentes são determinados pela especificação de um conjunto de classes de objeto e, em seguida, pela compilação e montagem do código associado àquelas classes nos arquivos executáveis. Um design de componente de software completamente considerado reflete uma variedade de preocupações:

- Localidade -- onde os componentes precisam ser executados.
- Hosting -- conjunto de instruções do processador e restrições de memória para o código em execução.
- Simultaneidade -- separação de processamento em hosts ou espaços de memória diferentes para endereçar preocupações de confiabilidade e relacionadas.

São apresentadas a seguir as informações necessárias para especificar os componentes que incluem as pesquisas de opinião de serviços de subsistemas hospedados para localidades e seus componentes de hardware realizados, pesquisas de serviços executados para processos e a visualização de classes participantes (VOPC) dos serviços do subsistema.

Para cada configuração de hardware, o método do RUP SE exige a criação de um componente das classes que participam de todos os serviços do subsistema hospedados em cada nó. Se esses serviços precisarem ser executados em mais de um processo, nós dividiremos os componentes posteriormente, designando as classes participantes dos serviços do subsistema executados por cada um dos processos. Observe que alguns serviços do subsistema podem ser executados por mais deum processo e, portanto, suas classes podem estar em mais de um componente. Concluímos o processo dividindo os componentes adicionais para explicar as restrições de memória (como comércios .exe e .dll), envio de limitações de mídia e assim por diante.

Essas atividades resultam em um conjunto de componentes de hardware e software específicos, que constituem o sistema.

Desenvolvimento do Sistema

Os projetos do RUP SE são gerenciados praticamente da mesma maneira que qualquer projeto RUP. No entanto, por causa do tamanho e das atividades adicionais requeridas para a maioria dos esforços de engenharia de sistema, há alguns diferenças, que abordaremos brevemente nesta seção.

Organização do Projeto

A transição de um processo serializado tradicional (processo "em cascata") para um iterativo tem implicações profundas em como um projeto deve ser organizado. Em um processo serializado, os membros da equipe normalmente são designados a um projeto até que seus artefatos estejam concluídos. Por exemplo, a equipe de engenharia conclui as especificações, as distribui para a equipe de desenvolvimento e vai para o próximo projeto. Em qualquer projeto com base no RUP, tal distribuição não ocorre. Em vez disso, os artefatos evoluem iterativamente durante todo o processo de desenvolvimento. Isso requer que os membros da equipe responsáveis pelos artefatos do projeto, como o banco de dados de requisitos e a arquitetura UML, deve ser designada ao projeto de desenvolvimento durante todo o seu ciclo de vida.

A Figura 5 mostra a organização de um típico projeto do RUP SE. Ela consiste em uma coleção de equipes de desenvolvimento, cada uma com um gerente de projeto e um líder técnico. Também existem equipes que lidam com a arquitetura geral do sistema e o gerenciamento do projeto.

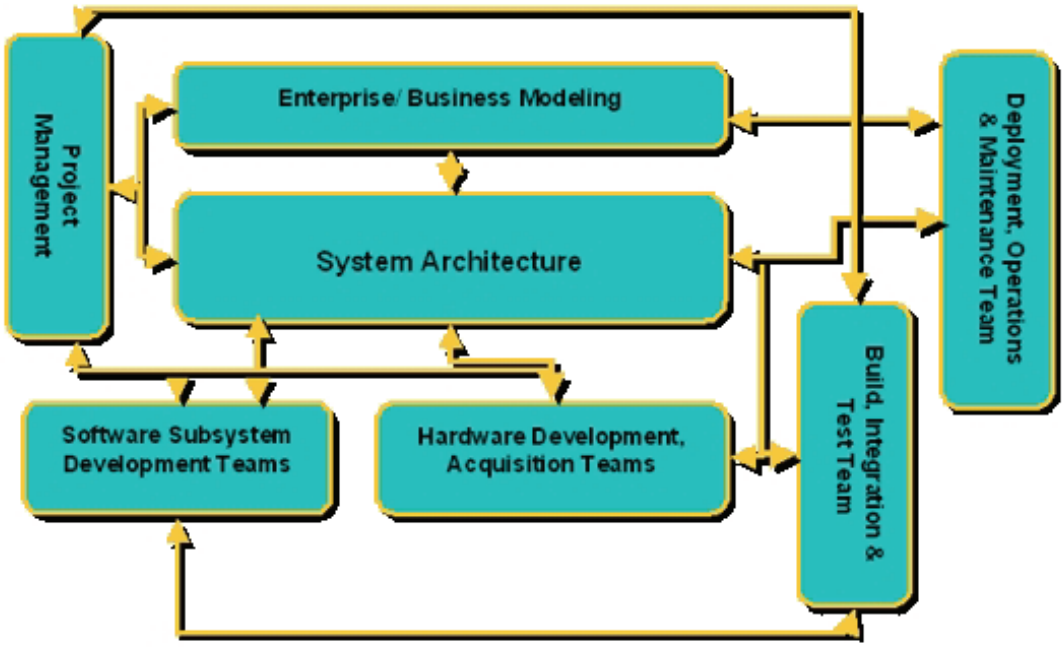


Figura 5: Um Gráfico da Organização do RUP SE

As equipes nesta figura têm as seguintes funções:

- A Equipe de Modelagem Corporativa analisa a necessidade do negócio e gera modelos de negócios e/ou artefatos relacionados, como conceito de documentos de operações.
- A Equipe de Arquitetura do Sistema trabalha com a Equipe de Modelagem Corporativa para criar os requisitos de derivação do sistema e de contexto do sistema. Essa equipe desenvolve as visualizações de localidade e de subsistema, bem como seus requisitos derivados. Em todo o processo de desenvolvimento, a Equipe de Arquitetura do Sistema funciona como um ponto de escalação técnica para resolver problemas de engenharia e de arquitetura. Essa equipe também trabalha com as equipes de desenvolvimento para especificar a arquitetura do componente de software. Os membros da equipe incluem os líderes técnicos das equipes de desenvolvimento.
- A Equipe de Gerenciamento de Projeto cuida dos problemas padrão do projeto, como revisões do projeto, planejamento de recursos, controle de orçamento, valor atribuído e variações e planejamento de iteração coordenado.
- Para cada iteração, a Equipe de Integração e Teste recebe o código e os componentes de hardware das equipes de desenvolvimento, compila os

componentes de software e instala os componentes de hardware e software em uma configuração de laboratório. A equipe também planeja, executa e relata sobre os testes do sistema para cada iteração.

- As Equipes de Desenvolvimento de Subsistema são responsáveis pelo design e implementação da realização de software de um ou mais subsistemas. As equipes baseiam o trabalho nos casos de uso derivados descobertos durante a atividade de fluxo decrescente. Dependendo do tamanho e da complexidade do sistema, os casos de uso do subsistema podem ser realizados como design de classe e módulos de código associados, ou os subsistemas podem ser posteriormente decompostos em subsistemas. No último caso, uma equipe de subsistema pode ser decomposta posteriormente em equipes de sub-subsistemas e uma equipe de arquitetura de subsistema pode ser criada. Esse processo permite a escalabilidade da abordagem do RUP SE.

- As Equipes de Aquisição e Desenvolvimento de Hardware sãoresponsáveis pelo design, especificação e entrega dos componentes de hardware.

- A Equipe de Operações de Implementação e Manutenção lida com problemas operacionais e serve como uma ligação com os usuários. Essa equipe pode instalar e manter o sistema no campo. Em outros casos, essa equipe pode manipular o relatório de defeitos do usuário e fornecer correções para o campo.

Design e Implementação Simultâneos

Um recurso atraente da abordagem de organização de RUP SE é que ela escala para programas muito grandes. Depois que você decompuser o sistema em subsistemas e localidades com seus requisitos derivados, cada um desses elementos de modelo de análise é adequado para o design e o desenvolvimento simultâneos. Conforme observamos, você pode designar subsistemas UML para separar equipes de desenvolvimento e designar localidades para equipes de desenvolvimento e aquisição de hardware. Cada equipe trabalha a partir de sua pesquisa de opinião derivada de serviços hospedados ou interfaces designadas para desenvolver sua parte do modelo de design e dos modelos de implementação. Dessa forma, o design e a implementação dos elementos de design podem continuar em paralelo.

Para sistemas muito grandes, uma abordagem de sistemas de sistemas pode ser adotada. Nesse caso, cada subsistema UML tem seu próprio modo de ocalidade e você precisa apenas determinar preocupações lógicas. Isso permite que o aplicativo da estrutura de organização mostrado na Figura 5 no nível do subsistema, fornecendo ainda mais escalabilidade.

Desenvolvimento, Integração e Teste Iterativos

Um recurso central da abordagem do RUP é que o sistema é desenvolvido em uma série de iterações, cada uma das quais produz um protótipo de trabalho com uma funcionalidade incrementalmente nova. O sistema é integrado e testado em cada iteração e o teste da iteração é um subconjunto dos testes do sistema.

Conseqüentemente, a iteração final resulta em um sistema totalmente testado pronto para transição para a configuração operacional.

O tempo e o conteúdo das iterações são capturados no plano de iteração no início do projeto. Entretanto, como qualquer artefato do RUP, o plano de iteração é atualizado continuamente para refletir o entendimento emergente do sistema à medida que ele evolui.

O conteúdo de uma iteração, capturado em um plano de iteração do sistema, é especificado por quais casos de uso e requisitos suplementares são realizados pelos componentes desenvolvidos na iteração. Cada iteração é testada pelo subconjunto dos casos de teste do sistema aplicáveis.

Lembre-se de que os subsistemas têm serviços derivados rastreados a partir de serviços do sistema. Esse rastreio fornece uma base de planos de iteração derivados para os subsistemas e as localidades. Isto é, o conteúdo de cada iteração do sistema é rastreável para a funcionalidade que precisa ser fornecida pelos subsistemas e localidades para suportar a iteração. Na prática, as equipes de desenvolvimento negociarão o conteúdo da iteração para refletir os aspectos práticos de seu desenvolvimento. Por exemplo, uma iteração de sistema inicial não pode requerer a funcionalidade completa de um subsistema. Compromissos devem ser feitos.

Um bom plano de iteração do sistema fornece a oportunidade de identificar e resolver antecipadamente os riscos técnicos do sistema, antes do pânico típico da fase de integração e teste baseada em cascata. Os riscos técnicos podem envolver requisitos funcionais e não-funcionais. Por exemplo, a integração antecipada pode revelar problemas de inicialização e failover que não podem ser totalmente entendidos com especificações detalhadas de design e interface. Na prática, as iterações iniciais devem validar se a arquitetura é suficiente para atender aos requisitos não-funcionais.

O desenvolvimento interativo de sistemas pode parecer mais caro porque requer mais teste, além de ambientes de hardware simulados ou montados para suportar as iterações iniciais. A coordenação do conteúdo para cada iteração em equipes de desenvolvimento também exige mais esforço de gerenciamento do projeto. Entretanto, esses custos aparentes são compensados pelas economias na identificação antecipada e mitigação dos riscos associados à arquitetura do sistema. É um princípio padrão de engenharia que a remoção de defeitos de arquitetura tardiamente em um projeto é muito mais cara do que a remoção dos mesmos antecipadamente. Além das despesas adicionadas, a remoção tardia de defeitos no processo também inclui incerteza, assim como os riscos de planejamento e orçamento tardios em um projeto.

A função da organização de teste em um projeto iterativo é diferente da função de teste dentro de um projeto serializado e baseado em cascata. Em vez de alocar uma grande quantidade de tempo para integração geral do sistema após o desenvolvimento, uma organização de teste com base iterativa gasta tempo integrando, testando e relatando defeitos em todo o ciclo de vida do projeto.

Em Resumo

O RUP SE, fornecido como um Rational Unified Process® (RUP) Plug-In, é um aplicativo da estrutura RUP e suporta o desenvolvimento de sistemas em grande escala que são compostos por componentes de software, hardware, trabalhadores e informações. O RUP SE inclui uma estrutura do modelo de arquitetura que permite a consideração de diferentes perspectivas formais (lógicas, físicas, informação, etc.) para fornecer uma solução que determina as preocupações dos vários envolvidos no desenvolvimento. Uma característica distinta do RUP SE é que os requisitos desses componentes do sistema são juntamente derivados da crescente especificidade dos requisitos gerais do sistema.

O RUP SE é idealmente adequado para projetos que:

- Sejam grandes o suficiente para exigir várias equipes com desenvolvimento simultâneo.
- Tenham desenvolvimento simultâneo de hardware e software.
- Tenham problemas de implementação arquiteturalmente significativos.
- Incluam um novo projeto da infra-estrutura subjacente da tecnologia da informação para suportar processos de negócios em desenvolvimento.

O RUP SE fornece à equipe de desenvolvimento do sistema as vantagens das práticas recomendadas do RUP, ao mesmo tempo que fornecem uma estrutura para determinar os problemas gerais do sistema. Alguns dos benefícios do RUP SE são:

- Equipe de suporte do sistema -- Oferece colaboração contínua entre analistas de negócios, arquitetos, engenheiros de sistema, desenvolvedores de software, desenvolvedores de hardware e testadores.
- Qualidade do sistema -- Fornece visualizações que permitem que as equipes tratem dos problemas de qualidade do sistema em um processo direcionado por arquitetura.
- Modelagem visual do sistema -- Fornece suporte de UML para arquitetura de sistemas.
- Escalabilidade --Sobe a escala para sistemas muito grandes.
- Desenvolvimento de componente -- Fornece fluxos de trabalho para determinar componentes de hardware e software.
- Design e desenvolvimento iterativo do sistema -- Suporta o design simultâneo e o desenvolvimento iterativo de componentes de hardware e software.

Notas

[1] Consulte o relatório do IBM Rational de Maria Ericsson, "Developing Large Scale Systems Using the Rational Unified Process" em http://www.rational.com/products/whitepapers/sis.jsp.

[2] Sanford Friedenthal et al., "Adapting UML for an Object-Oriented Systems Engineering Method." Proceedings of the 2000 INCOSE Symposium.

[3] Murray Cantor. "Thoughts on Functional Decomposition," The Rational Edge, April, 2003.

Para obter mais informações sobre os produtos ou serviços discutidos neste artigo, clique aqui e siga as instruções fornecidas.

Obrigado!