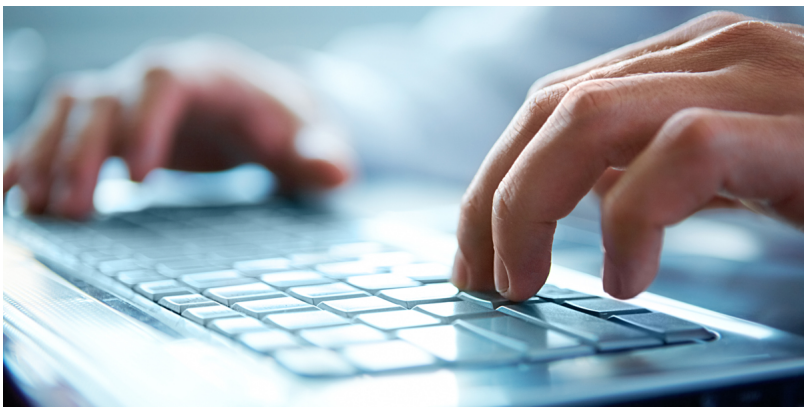
Arquitetura de Sistemas

Aula 10: Provisionamento e Construção – Parte II

INTRODUÇÃO



Nesta aula, você verá como garantir que as especificações dos componentes criados sejam tão independentes quanto possível das tecnologias a serem utilizadas.

Verá também que algumas técnicas de especificação não são suportadas por todas as tecnologias utilizadas, de modo que é necessária uma certa quantidade de interpretação.

OBJETIVOS

Reconhecer a importância do provisionamento e a construção de componentes para o melhor resultado na arquitetura de sistemas;

Analisar os elementos que compõe a etapa de construção de componentes como parte integrante da metodologia apresentada na disciplina;

Identificar a relação entre a construção de componentes e os outros processos da metodologia apresentada na disciplina.

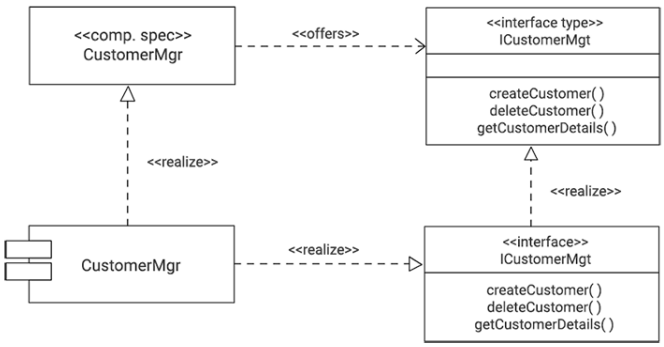
ESPECIFICAÇÃO DE COMPONENTES X CONSTRUÇÃO DE COMPONENTES

Em UML, um componente realiza uma ou mais interfaces. Como já discutimos antes, isso é suficiente para modelar alguns dos detalhes da tecnologia de implementação de componentes.

Para lidar com especificação, nós adicionamos alguns estereótipos UML, como especificação de componentes, as classes e suas interfaces.

Uma especificação de componente oferece um ou mais tipos de interfaces, por isso há uma correspondência bastante simples entre os elementos de especificação e os elementos de execução.

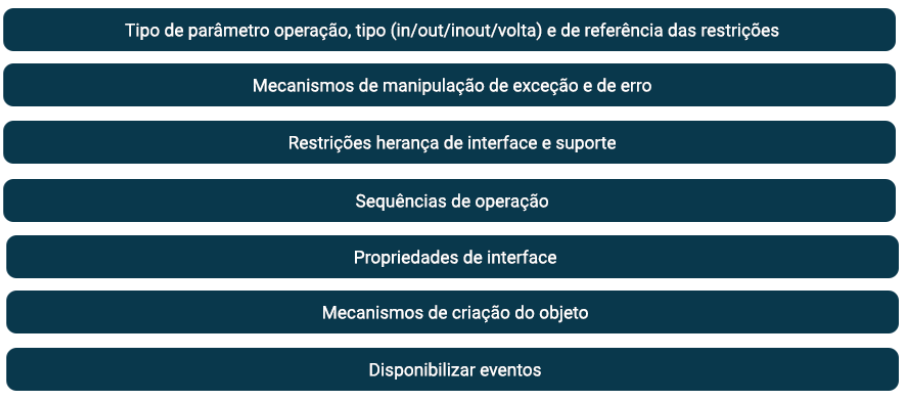
UML também define a relação entre o componente e uma interface através de relacionamentos.



Fonte: UML – conceitos de definição de interfaces.

MAPEAMENTO DE CONSTRUÇÃO E RESTRIÇÕES

Os principais problemas com mapeamento de construção e restrições com componentes de tecnologia surgem a partir das seguintes considerações:



HERANÇAS DE INTERFACE E SUPORTE DE INTERFACES

Há uma diferença enorme entre o COM + e EJB, quando se trata de herança de interface e suporte de interfaces.

COM +

COM + permite apenas herança única de interface.

Para permitir que objetos tenham múltiplas classificações, os componentes devem suportar múltiplas interfaces.

Por exemplo, um objeto documento **COM +**, suportando IDocument, IInvoice e IAccStmnt, pode ser:

Um documento;
Uma fatura;
Um extrato de conta.

Além disso, todas as interfaces **COM +** devem, finalmente, herdar de IUnknown-a, interface base **COM +.**

EJB

EJB permite herança múltipla de interfaces e permite que classes Java apoiem múltiplas interfaces, limitando apenas unicamente herança de classe.

Quando registramos uma classe Java como um **EJB** com um ambiente de componentes **EJB**, ficamos restritos à nomeação de uma interface (a chamada interface remota).

O container para o seu **EJB** fornece uma classe de suporte para essa interface, que são, então, delegados a sua classe Java.

Apesar de sua classe Java poder suportar muitas interfaces, o container **EJB** fica limitado a somente um deles.

Isso significa que, se quiser que seu componente suporte múltiplas interfaces, você vai precisar usar herança de interface múltipla para herdar toda a funcionalidade do componente de uma interface pai, que pode ser registrada no ambiente **EJB**.

Essa superinterface é utilizada efetivamente durante a especificação do componente.

Atenção

,
Devemos tomar cuidado ao usar a herança com tipos de interface. Sabendo-se que esses são artefatos de especificação, de definição e de herança, eles vão se comportar de formas diferentes., , Ao contrário de herança de classe de implementação (onde métodos de subclasse podem substituir os métodos da superclasse), as especificações de operação de subtipos só podem incrementar as definições por seus supertipos., , Mais especificamente, isso significa que uma especialização de operações pode enfraquecer uma condição prévia (torná-la menos rigorosa) e fortalecer a pós-condição (torná-la mais rigorosa)., , Isso assegura que qualquer cliente, dependente dos pressupostos e garantias do supertipo, não vai encontrar o subtipo se comportando de forma inesperada.

PROPRIEDADES DE INTERFACES

No COM +, uma propriedade de interface é a especificação abreviada para um get e um set, como um par de operações.

Ao usarmos UML para especificar interfaces, definimos que um atributo de interface existe puramente por apoiar a especificação de pré e pós-condições de operações, e você deve, explicitamente, definir quais operações deseja. Por exemplo, a interface base COM + IUnknown tem duas operações para o gerenciamento de contagens de referência:

A especificação dessas operações tem de se referir ao atributo de interface “contagem de referência”. Porém, no COM +, definir tal atributo significa, de forma indireta, definir a existência de operações de acesso. Ambos os processos estão diretamente relacionados, o que pode nos trazer problemas.

Saiba Mais

,
Assim, para mapeamento COM +, precisamos de uma maneira para determinar quais as operações devemos mapear para a propriedade COM + taquigrafia., , Uma maneira simples é usar uma convenção de nomenclatura., , Alternativamente, poderíamos fazer um trabalho mais elaborado e estender a especificação da operação com um valor pré-definido para o nome da propriedade, e interpretar, a partir da existência dentro dos parâmetros, para determinar qual foi o get e o conjunto de informações adquiridas.

CRIAÇÃO DE OBJETOS

Tanto no EJB como no COM + usamos uma abordagem de fábrica de objetos, onde um objeto componente é utilizado para criar instâncias de outro componente.

No EJB, a fábrica é o objeto inicial. No COM +, é objeto IClassFactory.

Dentro dos diagramas de arquitetura de componentes e interações de componentes, achamos mais conveniente para omitir esse nível de abstração, pois simplesmente mostram **<<criar>>** dependências diretamente para o próprio componente.

Quando mapeamos esses objetos para o ambiente de componentes, esses elementos precisam ser adicionados, para que seus diagramas de interação no nível de aplicação mostrem isso.

No COM +, há muita flexibilidade sobre o qual objeto é a fábrica. Isso é feito simplesmente designando um componente que disponibiliza o IClassFactory relevante para uma dada classe.

Esse mesmo elemento também pode fazer outras coisas, e, neste caso, você deve disponibilizar uma funcionalidade para o mesmo.

Atenção

,
Para beans de sessão EJB, você precisa apenas fornecer uma especificação da interface inicial, mas, para beans de entidade, você precisa fornecer a implementação também.

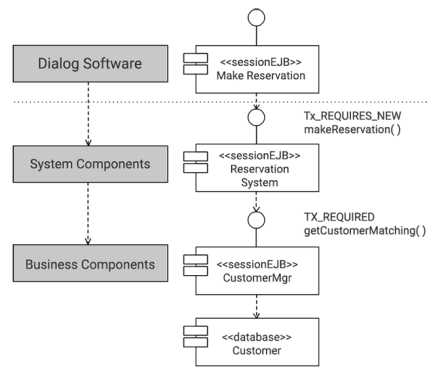
CORRESPONDÊNCIAS DA ARQUITETURA DA APLICAÇÃO

Apresentamos aqui uma série de camadas distintas na arquitetura de aplicação, onde cada camada contém diferentes responsabilidades.

Vamos agora indicar brevemente como certas características de especificação dessas camadas podem ser mapeadas para as tecnologias-alvo.

É evidente que as considerações detalhadas de uma arquitetura de sistemas por componentes são muito extensas. Por isso, nos limitamos a algumas observações gerais sobre o uso de sessão e de entidade **EJBs** (essa distinção não existe no **Com +**) e quais as propriedades de transação devem ser utilizadas.

A camada de software de interação é responsável pela gestão da interface com o usuário. Nesse caso, não precisaríamos dividir em componentes, bastaria modelar a relação do usuário com o sistema.



Fonte: UML – elementos de camadas.

SUBCOMPONENTES

Fonte da Imagem:

Uma especificação é um componente que encapsula a sua funcionalidade (sob a forma de uma ou mais interfaces) que correspondem a uma única unidade de execução.

Isso define a unidade de fornecimento e substituição na arquitetura de aplicativos, geralmente provisionando e substituindo requisitos de granularidade para uma aplicação de negócios de grande escala.

No entanto, às vezes, queremos apresentar um modelo de objetos mais rico e detalhado ao cliente, sem a introdução de um enorme conjunto de componentes separados, cada um agindo de forma isolada.

Como podemos, então, equilibrar a necessidade de definir um componente de forma mais grosseira para efeitos de provisionamento e de forma mais detalhada para a especificação e uso?

Atenção

Os componentes CustomerMgr e clientes devem ser substituídos como uma unidade única.

Outro ponto importante para utilizarmos uma abordagem de subcomponente é quando temos componentes em sua arquitetura, para os quais existiam a intenção de desenvolver, substituir e gerenciar separadamente, mas a interação entre eles é tal que decidimos pelas vantagens de uma única implementação de desempenho.

A abordagem subcomponente permite-lhe manter a especificação conforme sua estruturação de negócios e detalhamento para implementação.

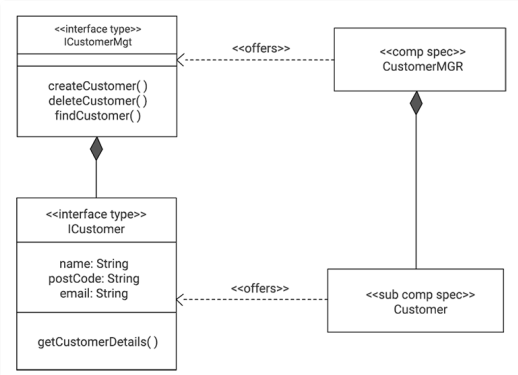
Instâncias são criadas por operações específicas

Subcomponentes não têm objetos de fábrica explícitas como os componentes. Em vez disso, suas instâncias são criadas por operações específicas do componente pai ou irmão.

Por exemplo, a operação ICustomerMgt :: createCustomer () iria criar uma instância do subcomponente do cliente.

Subcomponente objeto estado share com o seu componente, passando a existir somente em tempo de execução.

Um componente e todos os seus subcomponentes devem ser embalados em conjunto para o mesmo módulo (.exe, .dll .jar).



Fonte: UML – subcomponentes.

INTERAÇÃO COM SISTEMAS EXISTENTES

Fonte da Imagem:

Em alguns momentos de nossa disciplina, tratamos do desenvolvimento somente utilizando o conceito de novos componentes. Porém, no mundo real, quando utilizamos a arquitetura de sistemas por componentes, a maioria dos componentes já estão prontos em sistemas existentes ou em aplicativos de prateleira.

Para os casos de utilização de componentes prontos, devemos fazê-lo através de APIs, disponibilizadas por esses componentes.

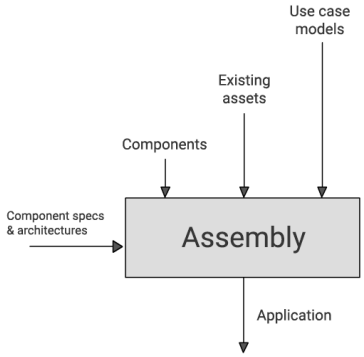
Todos os elementos de interação com os componentes prontos devem ser feitos através de interfaces, que precisam ser bem especificadas e definidas para não deixar dúvidas de seus usos e de seus resultados.

CONSTRUÇÃO

A construção é o processo de reunir componentes de software existentes ativos em um ambiente de TI e projetar uma interface de usuário para esse sistema, com base nos modelos para formar uma aplicação.

A construção compartilha várias das características presentes na configuração padrão do ambiente, principalmente no que se refere a práticas de gerenciamento.

Cada componente individual disponível pode ser visto como um item de configuração separado a ser utilizado, mantendo-se um controle de versão.



Fonte: UML – workflow de componentes.

Sobre “projetar componentes”, assinale a alternativa cujo o projeto NÃO faz parte deste passo:

☐
A) Projeto do modelo de classes.

☐
B) Projeto do modelo de casos de uso.

☐
C) Projeto do modelo de dados.

☐
D) Projeto do modelo de componentes.

○
○
○
○

Justificativa

Glossário