Fundamentos básicos de UML: O diagrama de classes

Uma introdução aos diagramas de estrutura em UML 2

Donald Bell (bellds@us.ibm.com)
IT Architect
IBM Corporation

19/Dez/2016

do The Rational Edge: como o exemplo mais importante do novo tipo de diagrama de estrutura em UML 2, o diagrama de classes pode ser usado por analistas, modeladores de negócios, desenvolvedores e testadores em todo o ciclo de vida de desenvolvimento de software. Este artigo oferece uma introdução abrangente.

Visualizar mais conteúdo nesta série

Esta é a próxima parte do artigo de uma série de artigos sobre os diagramas essenciais usados na Linguagem de Modelagem Unificada, ou UML. Em meu artigo anterior sobre diagramas de sequência, eu mudei o foco da especificação UML 1.4 para a especificação Adopted 2.0 Draft de UML da OMG (ou seja, UML 2). Neste artigo, vou discutir os Diagramas de estrutura, que são uma nova categoria que foi introduzida na UML 2. Como o propósito desta série é ensinar as pessoas sobre os elementos de notação e seus significados, este artigo concentra-se principalmente no diagrama de classes. O motivo logo ficará claro. Os próximos artigos abordarão outros diagramas incluídos na categoria de estrutura.

Eu também gostaria de lembrar os leitores que esta série é sobre os elementos da notação UML e que estes artigos não pretendem oferecer orientação sobre a melhor abordagem de modelagem

ou como determinar o que deve ser modelado em primeiro lugar. Na verdade, o propósito deste artigo e da série em geral é ajudar com o entendimento básico dos elementos de notação — suas sintaxes e seus significados. Com este conhecimento, você conseguirá ler diagramas e criar seus próprios diagramas usando os elementos de notação adequados.

Este artigo assume que você tenha pouco entendimento de design orientado a objetos. Se você precisar de um pouco de ajuda com os conceitos de OO, confira o tutorial de síntese da Sun Object-Oriented Programming Concepts. *Lendo as seções* "What Is a Class?" *e o* "What Is Inheritance?" *você terá um entendimento suficiente para que este artigo lhe seja útil. Além disso, o livro de David Taylor,* Object-Oriented Technologies: A Manager's Guide*, oferece uma explicação excelente, de alto nível, sobre design orientado a objetos sem precisar de um entendimento aprofundado de programação de computador.*

As vantagens e as desvantagens da UML 2

Em UML 2, existem duas categorias básicas de diagramas: diagramas de estrutura e diagramas de comportamento. Cada diagrama UML pertence a uma dessas duas categorias de diagrama. O propósito dos diagramas de estrutura é mostrar a estrutura estática do sistema que está sendo modelado. Eles incluem a classe, o componente e/ou os diagramas de objeto. Os diagramas comportamentais, por outro lado, mostram o comportamento dinâmico entre os objetos no sistema, incluindo itens como métodos, colaborações e atividades. Exemplos de diagramas de comportamento são atividades, casos de uso e diagramas de sequência.

Diagramas de estrutura em geral

Como já mencionei, os diagramas de estrutura mostram a estrutura estática do sistema que está sendo modelado, concentrando-se nos elementos de um sistema, sem restrição de tempo. A estrutura estática é transmitida mostrando os tipos e suas instâncias no sistema. Além de mostrar os tipos de sistema e suas instâncias, os diagramas de estrutura também mostram pelo menos alguns dos relacionamentos entre esses elementos e até mesmo podem mostrar suas estruturas internas.

Os diagramas de estrutura são úteis em todo o ciclo de vida de software para uma variedade de membros da equipe. Em geral, esses diagramas permitem a validação do design e a comunicação do design entre os indivíduos e as equipes. Por exemplo, os analistas de negócios podem usar diagramas de classe ou de objeto para modelar os ativos e os recursos atuais de um negócio, como o livro razão, os produtos ou a hierarquia geográfica de uma conta. Os arquitetos podem usar os diagramas de componente e de implementação para testar/verificar se o design está correto. Os desenvolvedores podem usar os diagramas de classes para projetar e documentar as classes codificadas pelo sistema (ou que serão codificadas em breve).

O diagrama de classes especificamente em

UML 2 considera os diagramas de estrutura como uma classificação; não existe um diagrama em si chamado "diagrama de estrutura". No entanto, o diagrama de classes oferece um ótimo exemplo do tipo de diagrama de estrutura e fornece um conjunto inicial de elementos de notação que todos os outros diagramas de estrutura usam. E como o diagrama de classes é tão fundamental, o restante deste artigo será focado no conjunto de notação do diagrama de classes. Ao final deste artigo, você terá um entendimento de como desenhar um diagrama de classe UML 2 e terá uma base sólida para entender outros diagramas de estrutura que serão abordados em artigos futuros.

Os fundamentos básicos

Como mencionado anteriormente, o propósito do diagrama de classes é mostrar os tipos que estão sendo modelados no sistema. Na maioria dos modelos UML, esses tipos incluem:

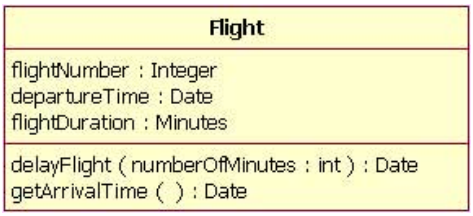
- uma classe
- uma interface
- um tipo de dado
- um componente.

A UML usa um nome especial para esses tipos: "classificadores". Em geral, é possível considerar um classificador como se fosse uma classe, mas tecnicamente, um classificador é um termo mais geral que também se refere aos outros três tipos acima.

Nome da classe

A representação UML de uma classe é um retângulo contendo três compartimentos empilhados verticalmente, como é mostrado na Figura 1. O compartimento superior mostra o nome da classe. O compartimento do meio lista os atributos da classe. O compartimento inferior lista as operações da classe. Ao projetar um elemento de classes em um diagrama de classes, deve-se usar o compartimento superior, e os dois compartimentos inferiores são opcionais. (Os dois inferiores seriam desnecessários em um diagrama que ilustrasse um nível mais alto de detalhes, no qual o propósito fosse mostrar somente o relacionamento entre os classificadores). A Figura 1 mostra o voo de uma companhia aérea modelado como uma classe UML. Como podemos ver, o nome é *Voo* e no compartimento do meio vemos que a classe Voo tem três atributos: flightNumber, departureTime e flightDuration. No compartimento inferior, vemos que a classe Voo tem duas operações: delayFlight e getArrivalTime.

Figura 1: diagrama de classes da classe Voo



Lista de atributos da classe

A seção de atributo de uma classe (o compartimento do meio) lista cada um dos atributos da classe em uma linha separada. A seção de atributo é opcional, mas quando é usada, contém cada atributo da classe exibido em um formato de lista. A linha usa o seguinte formato:

```
nome : tipo de atributo
```

```
flightNumber : número inteiro
```

Continuando com nosso exemplo da classe Voo, podemos descrever os atributos de classe com as informações de tipo de atributo, como é mostrado na Tabela 1.

Tabela 1: os nomes de atributo da classe Voo com seus tipos associados

Nome do atributo	Tipo de atributo
flightNumber	Número inteiro
departureTime	Data
flightDuration	Minutos

Nos diagramas de classe de negócios, os tipos de atributo geralmente correspondem a unidades que fazem sentido aos prováveis leitores do diagrama (ou seja, minutos, dólares, etc).. No entanto, um diagrama de classes que será usado para gerar código precisa de classes cujos tipos de atributo sejam limitados aos tipos fornecidos pela linguagem de programação ou aos tipos incluídos no modelo que também serão implementados no sistema.

Às vezes, é interessante mostrar em um diagrama de classes que um atributo específico tem um valor padrão. (Por exemplo, em um aplicativo de conta bancária, uma nova conta bancária começaria com um saldo igual a zero). A especificação UML permite a identificação de valores padrão na seção de lista de atributos usando a notação a seguir:

```
nome : tipo de atributo = valor padrão
```

Por exemplo:

```
saldo : dólares = 0
```

É opcional mostrar um valor padrão para os atributos. A Figura 2 mostra uma classe Conta bancária com um atributo chamado *balance*, que tem um valor padrão igual a 0.

Figura 2: um diagrama de classes de Conta bancária mostrando o valor do atributo de saldo com o padrão de zero dólares

Lista de operações de classe

As operações de classes são documentadas no terceiro compartimento (inferior) do retângulo de diagrama de classes que, repetindo, é opcional. Como os atributos, as operações de uma classe são exibidas em um formato de lista, com cada operação em sua própria linha. As operações são documentadas usando a notação a seguir:

```
nome(lista de parâmetros) : tipo de valor retornado
```

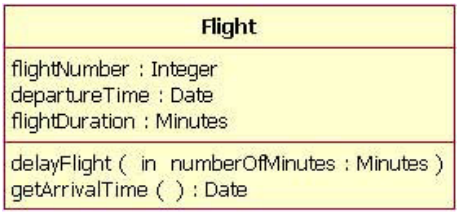
As operações da classe Voo são mapeadas na Tabela 2 abaixo.

Tabela 2: operações da classe Voo mapeadas da Figura 3

Nome da operação	Retorno de parâmetros	Tipo de valor
delayFlight	Nome: numberOfMinutes	N/A / Minutos
getArrivalTime	N/A	Data

A Figura 3 mostra que a operação delayFlight tem um parâmetro de entrada — numberOfMinutes — do tipo Minutos. No entanto, a operação delayFlight não tem um valor de retorno. [Observação: a delayFlight não tem um valor de retorno porque eu tomei a decisão de design de não ter. Alguém pode argumentar que a operação de atraso deveria retornar o novo horário de chegada e, se esse fosse o caso, a assinatura da operação apareceria como `delayFlight(numberOfMinutes : Minutos) : data.`] Quando uma operação tem parâmetros, eles são colocados dentro dos parênteses da operação. Cada parâmetro usa o formato "nome do parâmetro : tipo de parâmetro".

Figura 3: os parâmetros da operação da classe Voo incluem a marcação opcional "in"



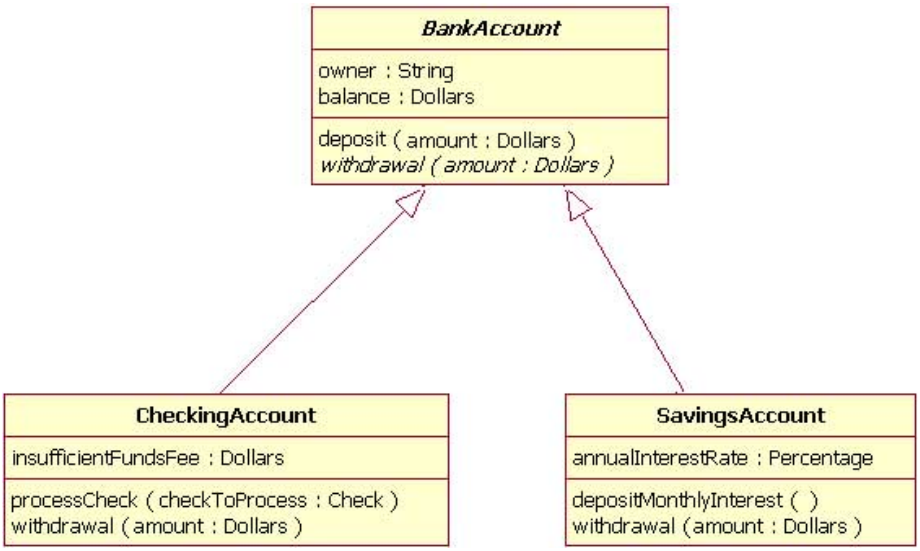
Ao documentar os parâmetros de uma operação, você pode usar um indicador opcional para mostrar se o parâmetro é ou não é de entrada ou de saída da operação. Esse indicador opcional aparece como "in" ou "out", como é mostrado no compartimento das operações na Figura

3. Geralmente, esses indicadores são desnecessários, a não ser que uma linguagem de programação mais antiga, como a Fortran, seja usada e, nesse caso, essas informações podem ser úteis. No entanto, em C++ e Java, todos os parâmetros são parâmetros "in", e como "in" é o tipo padrão de parâmetro de acordo com a especificação UML, a maioria das pessoas não usará os indicadores de entrada/saída.

Herança

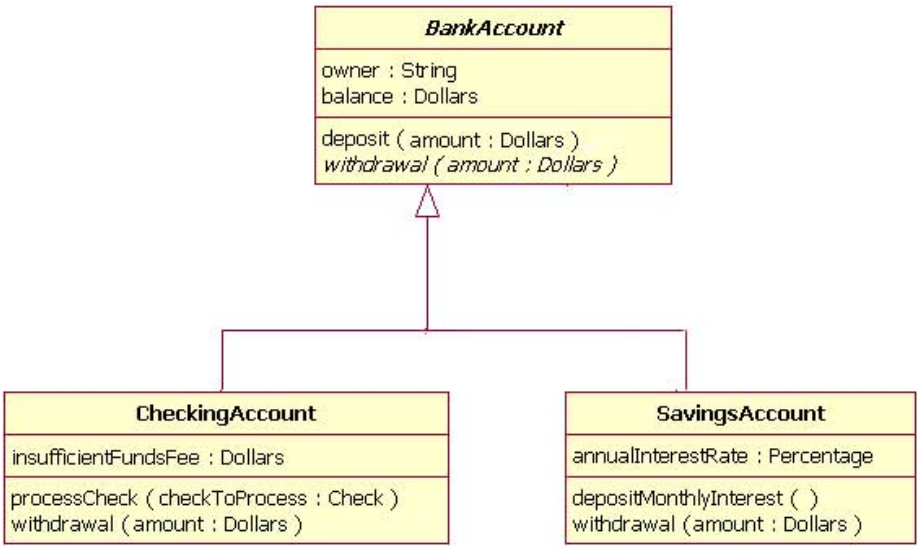
Um conceito muito importante em design orientado a objetos, *herança*, refere-se à capacidade de uma classe (classe-filha) de *herdar* a funcionalidade idêntica de outra classe (superclasse) e, em seguida, incluir sua nova funcionalidade própria. (Em um sentido que não é nada técnico, imagine se eu tivesse herdado as habilidades musicais gerais da minha mãe, mas na minha família eu fosse a única pessoa que tocasse guitarra elétrica). Para modelar a herança em um diagrama de classes, uma linha sólida é desenhada a partir da classe-filha (a classe que herda o comportamento) com uma ponta de seta (ou um triângulo) fechada, não preenchida, apontando para a superclasse. Considere os tipos de contas bancárias: a Figura 4 mostra como ambas as classes CheckingAccount e SavingsAccount herdam da classe BankAccount.

Figura 4: a herança é indicada por uma linha sólida com uma ponta da seta fechada, não preenchida, apontando para a superclasse



Na Figura 4, o relacionamento de herança é desenhado com linhas separadas para cada subclasse, que é o método usado no IBM Rational Rose e no IBM Rational XDE. No entanto, existe uma maneira alternativa de desenhar herança chamada *notação em árvore* . É possível usar a notação em árvore quando há duas ou mais classes-filhas, como na Figura 4, exceto que as linhas de herança se mesclam como uma ramificação da árvore. A Figura 5 é um novo desenho da mesma herança mostrada na Figura 4, mas desta vez usando a notação em árvore.

Figura 5: um exemplo de herança usando a notação em árvore



Classes e operações abstratas

O leitor observador perceberá que os diagramas nas Figuras 4 e 5 usam texto em itálico para o nome da classe BankAccount e a operação de retirada. Isso indica que a classe BankAccount é uma classe abstrata e o método de retirada é uma operação abstrata. Em outras palavras, a classe BankAccount fornece a assinatura da operação abstrata de retirada e as duas classes-filhas de CheckingAccount e SavingsAccount implementam cada uma sua própria versão dessa operação.

No entanto, as superclasses (classes-pai) não precisam ser classes abstratas. É normal que uma classe padrão seja uma superclasse.

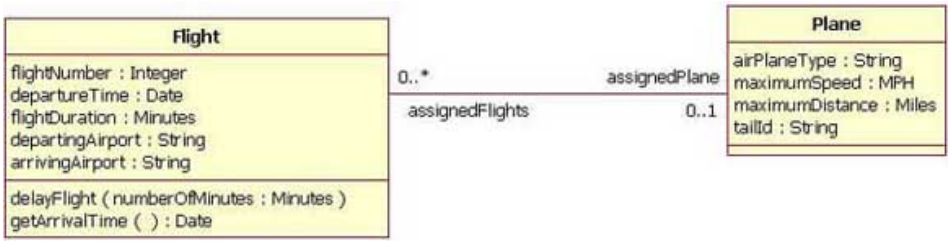
Associações

Ao modelar um sistema, certos objetos estarão relacionados entre si, e esses próprios relacionamentos precisam ser modelados para ficar mais claro. Existem cinco tipos de associações. Vou discutir duas delas — associações bidirecionais e unidirecionais— nesta seção, e os três tipos de associação restantes serão discutidos na seção *Além dos fundamentos básicos* . Observe que a discussão detalhada de quando usar cada tipo de associação não faz parte do escopo deste artigo. Neste artigo, vou focar o propósito de cada tipo de associação e mostrar como a associação é desenhada em um diagrama de classes.

Associação bidirecional (padrão)

Uma associação é uma ligação entre duas classes. As associações são sempre consideradas bidirecionais, isso significa que ambas as classes estão cientes de cada uma e do relacionamento que têm, a menos que você qualifique a associação como algum outro tipo. Voltando ao exemplo do Voo, a Figura 6 mostra um tipo padrão de associação entre a classe Voo e a classe Avião.

Figura 6: um exemplo de associação bidirecional entre uma classe Voo e uma classe Avião



Uma associação bidirecional é indicada por uma linha sólida entre as duas classes. Em ambas as extremidades da linha, você coloca um nome da função e um valor de multiplicidade. A Figura 6 mostra que o Voo está associado a um Avião específico e a classe Voo reconhece essa associação. A Avião assume a função de "assignedPlane" nesta associação porque o nome da função ao lado da classe Avião indica isso. O valor de multiplicidade ao lado da classe Avião de 0..1 significa que quando existe uma instância de um Voo, ela pode ter uma instância de um Avião associada a ela ou nenhum Avião associado a ela (ou sejam, talvez um avião ainda não tenha sido designado). A Figura 6 também mostra que um Avião reconhece sua associação com a classe do Voo. Nesta associação, o Voo assume a função de "assignedFlights". O diagrama na Figura 6 informa que a instância de Avião pode ser associada a nenhum voo (por exemplo, é um avião novo) ou a um número infinito de voos (por exemplo, o avião está em uso nos últimos cinco anos).

Para quem está imaginando quais são os possíveis valores de multiplicidade das extremidades das associações, a Tabela 3 abaixo lista alguns valores de multiplicidade de exemplo, juntamente com seus significados.

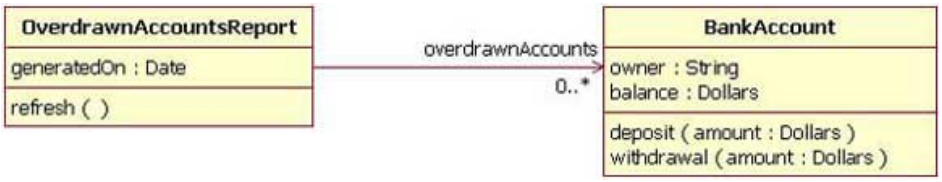
Tabela 3: valores de multiplicidade e seus indicadores

Possíveis valores de multiplicidade	
Indicador	**Significado**
0..1	Zero ou um
1	Somente um
0..*	Zero ou mais
*	Zero ou mais
1..*	Um ou mais
3	Somente três
0..5	Zero a cinco
5..15	Cinco a quinze

Associação unidirecional

Em uma associação unidirecional, duas classes são relacionadas, mas somente uma classe reconhece que o relacionamento existe. A Figura 7 mostra um exemplo de um relatório de contas com saldo negativo com uma associação unidirecional.

Figura 7: um exemplo de uma associação unidirecional: a classe OverdrawnAccountsReport reconhece a classe BankAccount, mas a classe BankAccount não reconhece a associação



Uma associação unidirecional é desenhada como uma linha sólida com uma ponta da seta aberta (não a ponta de seta fechada, ou triângulo, usada para indicar a herança) apontando para a classe reconhecida. Como as associações padrão, a associação unidirecional inclui um nome da função e um valor de multiplicidade, mas diferentemente da associação bidirecional padrão, a associação unidirecional contém somente o nome da função e o valor de multiplicidade para a classe reconhecida. Em nosso exemplo na Figura 7, OverdrawnAccountsReport reconhece a classe BankAccount e a classe BankAccount exerce a função de "overdrawnAccounts". No entanto, diferentemente da associação padrão, a classe BankAccount não reconhece que está associada à OverdrawnAccountsReport. [Observação: pode parecer estranho que a classe BankAccount não reconheça a classe OverdrawnAccountsReport. Essa modelagem permite que as classes de relatório reconheçam a classe de negócios que relatam, mas as classes de negócios não reconhecem que estão sendo relatadas. Isso enfraquece o acoplamento dos objetos, deixando o sistema mais adaptável a mudanças].

Pacotes

Inevitavelmente, se você estiver modelando um grande sistema ou uma grande área de um negócio, haverá muitos classificadores diferentes no modelo. Gerenciar todas as classes pode ser uma tarefa desencorajadora, portanto, a UML fornece um elemento de organização chamado *pacote*. Os pacotes permitem que os modeladores organizem os classificadores de modelo em namespaces, semelhante às pastas em um sistema de arquivamento. Dividir um sistema em vários pacotes facilita o entendimento do sistema, principalmente se cada pacote representar uma parte específica do sistema. [Observação: os pacotes são ótimos para organizar as classes do modelo, mas é importante lembrar que os diagramas de classes devem comunicar facilmente as informações sobre o sistema que está sendo modelado. Em casos em que os pacotes têm muitas classes, é melhor usar vários diagramas de classes específicos do tópico em vez de apenas produzir um único diagrama de classes grande].

Existem duas maneiras de desenhar pacotes em diagramas. Não existe regra para determinar qual notação usar, exceto usar seus próprios critérios para avaliar qual é a mais fácil de ser lida no diagrama de classes que está desenhando. Ambas as maneiras começam com um retângulo grande com um retângulo menor (guia) acima de seu canto superior esquerdo, como é visto na Figura 8. Mas o modelador deve decidir como a associação ao pacote deve ser mostrada, da seguinte forma:

- Se o modelador decidir mostrar a associação ao pacote dentro do retângulo grande, todos esses membros deverão ser colocados dentro do retângulo. [Observação: é importante

entender que quando eu digo "todos esses membros", isso significa somente as classes que o diagrama atual mostrará. Um diagrama que mostra um pacote com conteúdos não precisa mostrar todo o seu conteúdo; ele pode mostrar um subconjunto dos elementos contidos de acordo com algum critério, que não é necessariamente todos os classificadores do pacote]. Além disso, o nome do pacote precisa ser colocado no retângulo menor do pacote (como é mostrado na Figura 8).

- Se o modelador decidir mostrar a associação ao pacote fora do retângulo grande, todos os membros que serão mostrados no diagrama deverão ser colocados fora do retângulo. Para mostrar quais classificadores pertencem ao pacote, uma linha é desenhada de cada classificador para um círculo que tem um sinal de mais dentro do círculo conectado ao pacote (Figura 9).

Figura 8: um elemento do pacote de exemplo que mostra seus membros dentro dos limites do retângulo do pacote

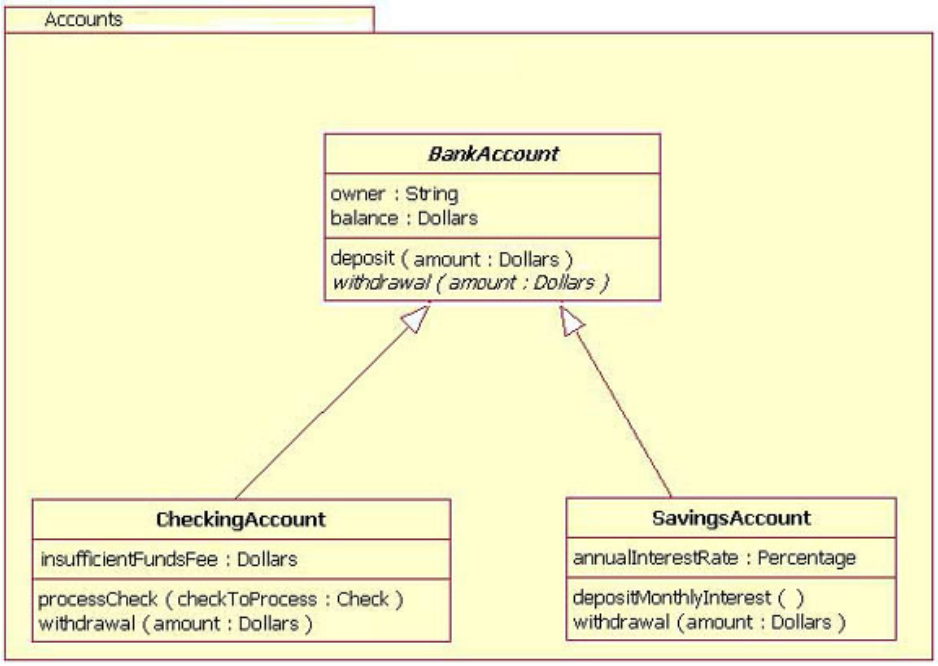
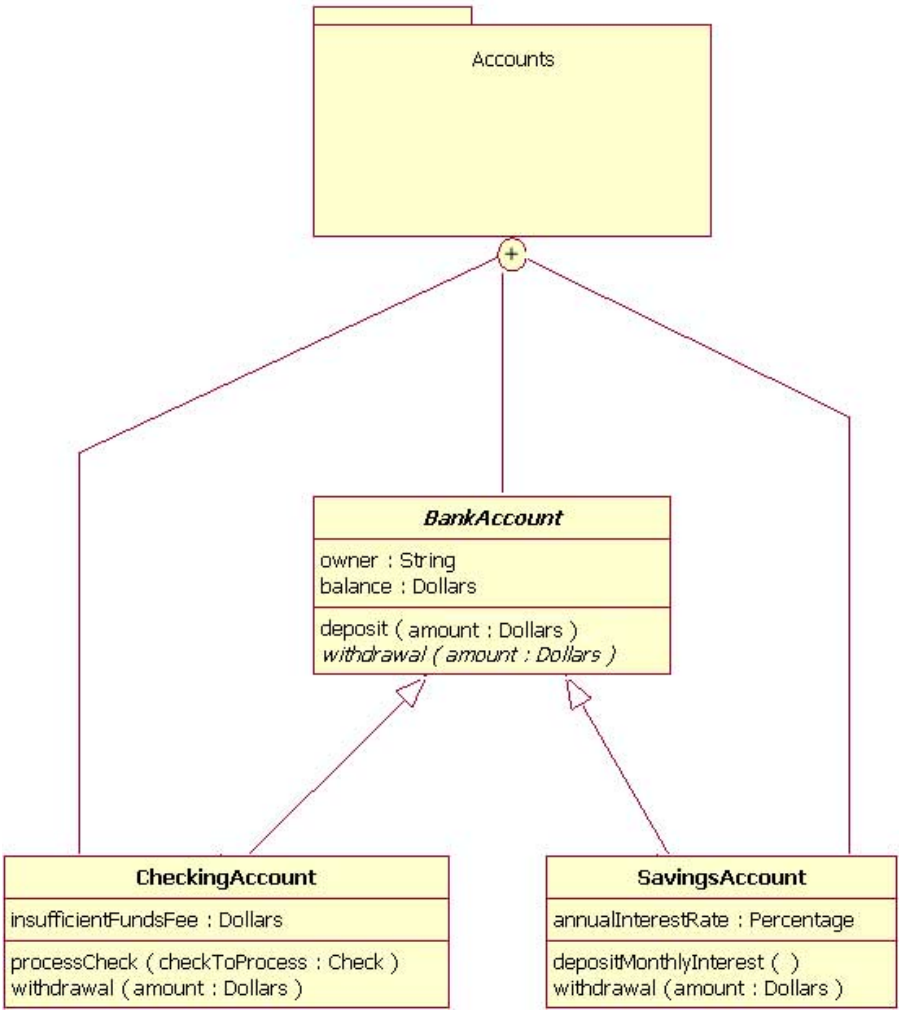


Figura 9: um elemento do pacote de exemplo mostrando sua associação por meio de linhas conectadas



A importância de entender os fundamentos básicos

É mais importante que nunca em UML 2 entender os fundamentos básicos do diagrama de classes. O motivo é que o diagrama de classes fornece os blocos de construção básicos para todos os outros diagramas de estrutura, como os diagramas de componente ou de objetos (para citar alguns).

Além dos fundamentos básicos

Neste ponto, eu já abordei os fundamentos básicos do diagrama de classes, mas não pare de ler! Nas próximas seções, vou abordar os aspectos mais importantes do diagrama de classes, que podem ser muito úteis. Esses aspectos incluem interfaces, os três tipos restantes de associações, visibilidade e outras adições na especificação UML 2.

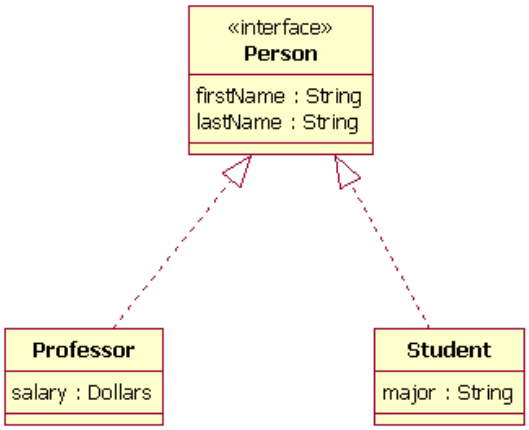
Interfaces

Anteriormente neste artigo, eu sugeri considerar os *classificadores* simplesmente como classes. Na verdade, um classificador é um conceito mais geral que inclui tipos de dados e interfaces.

Uma discussão completa de quando e como usar tipos de dados e interfaces efetivamente nos diagramas de estrutura de um sistema não faz parte do escopo deste artigo. Então por que eu menciono tipos de dados e interfaces aqui? Em algum momento, é possível que você deseje modelar esses tipos de classificadores em um diagrama de estrutura e é importante usar a notação adequada ao fazer isso, ou pelo menos estar ciente sobre esses tipos de classificadores. Desenhar esses classificadores incorretamente provavelmente confundirá os leitores do diagrama de estrutura, e o sistema resultante provavelmente não atenderá aos requisitos.

Uma classe e uma interface são diferentes: uma classe pode ter uma instância real de seu tipo, enquanto uma interface deve ter pelo menos uma classe para implementá-la. Em UML 2, uma interface é considerada uma especialização de um elemento de modelagem de classe. Portanto, uma interface é desenhada como uma classe, mas o compartimento superior do retângulo também contém o texto "«interface»", como é mostrado na Figura 10. [Observação: ao desenhar um diagrama de classes, está completamente dentro da especificação UML colocar «classe» no compartimento superior do retângulo, como você faria com «interface»; no entanto, a especificação UML diz que colocar o texto "classe" neste compartimento é opcional, e isso será assumido caso «classe» não seja exibido].

Figura 10: exemplo de um diagrama de classes no qual as classes Professor e Aluno implementam a interface Pessoa



No diagrama mostrado na Figura 10, ambas as classes Professor e Aluno implementam a interface Pessoa e não herdam dela. Podemos concluir isso por dois motivos: 1) O objeto Pessoa é definido como uma interface — ele tem o texto "«interface»" na área de nome do objeto e vemos que os objetos Professor e Aluno são objetos de *classe* porque estão rotulados de acordo com as regras para desenhar um objeto de classe (não existe um texto de classificação adicional na área de nome deles). 2) Sabemos que a herança não está sendo mostrada aqui porque a linha com a seta é pontilhada e não sólida. Como é mostrado na Figura 10, uma linha *pontilhada* com uma seta fechada e não preenchida significa realização (ou implementação). Como vimos na Figura 4, uma linha de seta *sólida* com uma seta fechada e não preenchida significa herança.

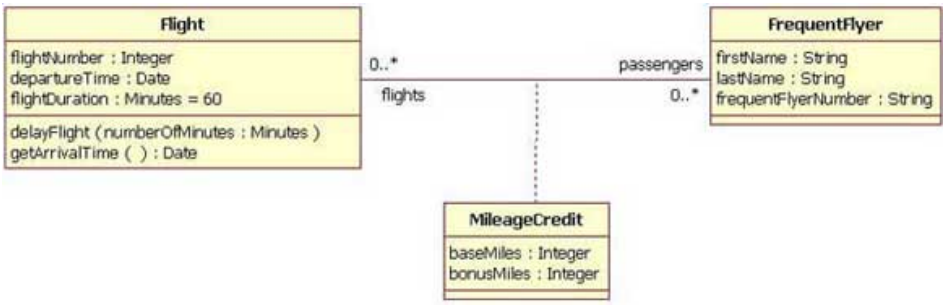
Mais associações

Acima, eu discuti as associações bidirecionais e unidirecionais. Agora, vou abordar os três tipos restantes de associações.

Classe de associação

Na modelagem de associações, algumas vezes é necessário incluir outra classe porque ela inclui informações valiosas sobre o relacionamento. Para isso, seria necessário usar uma *classe de associação* a ser vinculada à associação primária. Uma classe de associação é representada como uma classe normal. A diferença é que a linha de associação entre as classes primárias faz intersecção com uma linha pontilhada conectada à classe de associação. A Figura 11 mostra uma classe de associação para o nosso exemplo do setor de companhia aérea.

Figura 11: incluindo a classe de associação MileageCredit



No diagrama de classes mostrado na Figura 11, a associação entre a classe Voo e a classe FrequentFlyer resulta em uma classe de associação chamada MileageCredit. Isso significa que quando uma instância de uma classe Voo for associada a uma instância de uma classe FrequentFlyer, também haverá uma instância de uma classe MileageCredit.

Agregação

A agregação é um tipo especial de associação usado para modelar um relacionamento do "todo com as suas partes". Nos relacionamentos básicos de agregação, o ciclo de vida de uma classe *parte* é independente do ciclo de vida da classe *todo* .

Por exemplo, podemos considerar *Carro* como uma entidade inteira e *Roda do carro* como uma parte do Carro geral. A roda pode ser criada semanas antes e pode permanecer em um armazém até ser colocada no carro durante a montagem. Nesse exemplo, a instância da classe Roda tem uma vida claramente independente da instância da classe Carro. No entanto, há momentos em que o ciclo de vida da classe *partenão é* independente do ciclo de vida da classe *todo*— isso é chamado agregação de composição. Considere, por exemplo, o relacionamento de uma Empresa com seus Departamentos. A *Empresa e os Departamentos* são modelados como classes, e um Departamento não pode existir antes que uma Empresa exista. Aqui, a instância da classe Departamento depende da existência da instância da classe Empresa.

Vamos explorar mais a agregação básica e a agregação de composição.

Agregação básica

Uma associação com um relacionamento de agregação indica que uma classe faz parte de outra classe. Em um relacionamento de agregação, a instância da classe-filha pode ter uma vida maior do que sua classe-pai. Para representar um relacionamento de agregação, desenhe uma linha sólida a partir da classe-pai para a classe da parte e desenhe um losango não preenchido na

extremidade da associação classe-pai. A Figura 12 mostra um exemplo de relacionamento de agregação entre um Carro e uma Roda.

Figura 12: exemplo de associação de agregação

Agregação de composição

O relacionamento de agregação de composição é apenas outra forma de relacionamento de composição, mas o ciclo de vida da instância da classe-filha depende do ciclo de vida da instância da classe-pai. Na Figura 13, que mostra um relacionamento de composição entre uma classe Empresa e uma classe Departamento, observe que o relacionamento de composição é desenhado de forma semelhante ao relacionamento de agregação, mas desta vez o losango está preenchido.

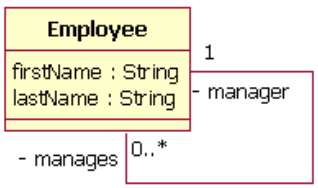
Figura 13: exemplo de um relacionamento de composição

No relacionamento modelado na Figura 13, uma instância da classe Empresa sempre terá pelo menos uma instância da classe Departamento. Como o relacionamento é um relacionamento de composição, quando a instância de Empresa é removida/destruída, a instância de Departamento também é automaticamente removida/destruída. Outro recurso importante da agregação de composição é que a classe da parte somente pode estar relacionada a uma instância da classe-pai (como a classe Empresa em nosso exemplo).

Associações reflexivas

Agora, nós já discutimos todos os tipos de associação. Como você deve ter observado, todos os nossos exemplos mostraram um relacionamento entre duas classes diferentes. No entanto, uma classe também pode ser associada a si mesma, usando uma associação reflexiva. Isso pode não fazer sentido inicialmente, mas lembre-se de que as classes são abstrações. A Figura 14 mostra como uma classe Funcionário poderia estar relacionada a si mesma por meio da função gerente/gerencia. Quando uma classe é associada a si mesma, isso não significa que uma instância da classe está relacionada a si mesma, mas que uma instância da classe está relacionada a outra instância da classe.

Figura 14: exemplo de um relacionamento de associação reflexiva



O relacionamento desenhado na Figura 14 significa que uma instância de Funcionário pode ser o gerente de outra instância Funcionário. No entanto, como a função de relacionamento "gerencia" tem uma multiplicidade de 0..*; um Funcionário pode não ter nenhum outro Funcionário para gerenciar.

Visibilidade

Em design orientado a objetos, existe uma notação de visibilidade para atributos e operações. A UML identifica quatro tipos de visibilidade: pública, protegida, privada e pacote.

A especificação UML não requer que a visibilidade de atributos e operações seja exibida no diagrama de classes, mas requer que ela seja definida para cada atributo e operação. Para exibir a visibilidade no diagrama de classes, coloque a marca de visibilidade na frente do nome do atributo ou da operação. Embora a UML especifique quatro tipos de visibilidade, uma linguagem de programação real pode incluir visibilidades adicionais ou pode não suportar as visibilidades definidas pela UML. A Tabela 4 exibe as diferentes marcas para os tipos de visibilidade suportados pela UML.

Tabela 4: marcas para os tipos de visibilidade suportados pela UML

Marca	Tipo de visibilidade
+	Público
#	Protegido
-	Privado
~	Pacote

Agora, vamos observar uma classe que mostra os tipos de visibilidade indicados para seus atributos e operações. Na Figura 15, todos os atributos e operações são públicos, com exceção da operação updateBalance. A operação updateBalance é protegida.

Figura 15: uma classe BankAccount que mostra a visibilidade de seus atributos e operações

Adições da UML 2

Agora que cobrimos os tópicos básicos e os tópicos avançados, vamos cobrir algumas das notações incluídas no diagrama de classes da UML 1.x.

Instâncias

Ao modelar a estrutura de um sistema, pode ser útil mostrar instâncias de exemplo das classes. Para modelar isso, a UML 2 fornece o elemento *especificação de instância* , que mostra informações interessantes usando instâncias de exemplo (ou reais) no sistema.

A notação de uma instância é a mesma que a de uma classe, mas em vez do compartimento superior ter simplesmente o nome da classe, o nome é uma concatenação sublinhada de:

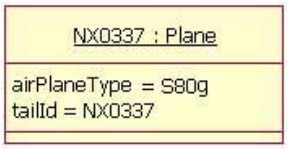
```
Nome da instância : nome da classe
```

Por exemplo:

```
Donald : pessoa
```

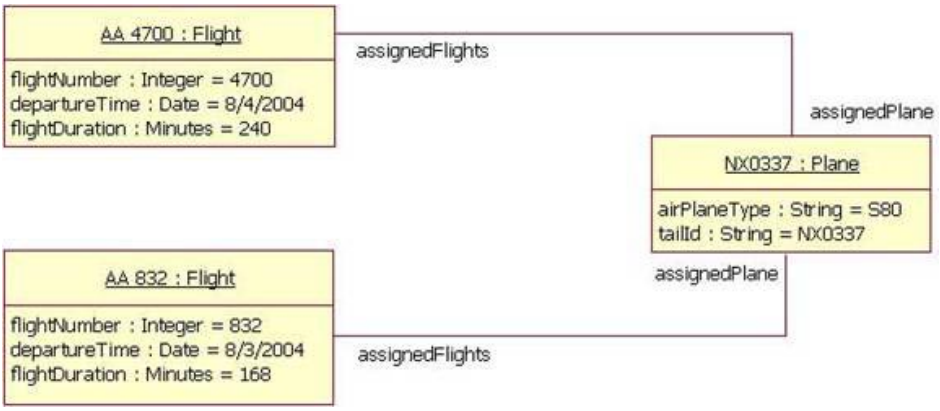
Como o propósito de mostrar instâncias é mostrar informações relevantes ou interessantes, não é necessário incluir no modelo todos os atributos e as operações da instância. É totalmente apropriado mostrar somente os atributos e seus valores que sejam interessantes, como ilustra a Figura 16.

Figura 16: uma instância de exemplo de uma classe de Voo (somente os valores de atributo interessantes são mostrados)



No entanto, simplesmente mostrar algumas instâncias sem seus relacionamentos não é muito útil, portanto, a UML 2 também permite a modelagem dos relacionamentos/associações no nível da instância. As regras para desenhar associações são as mesmas para os relacionamentos normais de classe, embora haja um requisito adicional ao modelar as associações. A restrição adicional é que os relacionamentos de associação devem corresponder aos relacionamentos do diagrama de classes e, portanto, os nomes da função de associação também devem corresponder ao diagrama de classes. Um exemplo disso é mostrado na Figura 17. Nesse exemplo, as instâncias são instâncias de exemplos do diagrama de classes encontrado na Figura 6.

Figure 17: um exemplo da Figura 6 usando instâncias em vez de classes



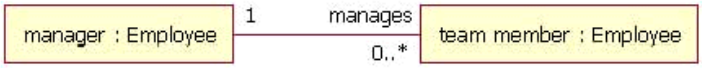
A Figura 17 tem duas instâncias da classe Voo porque o diagrama de classes indicou que o relacionamento entre a classe Avião e a classe Voo é de *zero para muitos*. Portanto, nosso exemplo mostra as duas instâncias de Voo às quais a instância do Avião NX0337 está relacionada.

Funções

A modelagem das instâncias de classes é, às vezes, mais detalhada do que o desejável. Às vezes, você pode simplesmente desejar modelar um relacionamento de classe em um nível mais

genérico. Nesse caso, use a notação de *função* . A notação de função é muito semelhante à notação de instâncias. Para modelar uma função de classe, desenhe uma caixa e coloque o nome da função de classe e o nome da classe dentro, como com a notação de instâncias mas, nesse caso, não sublinhe as palavras. A Figura 18 mostra um exemplo das funções representadas pela classe Funcionário descrita pelo diagrama na Figura 14. Na Figura 18, embora a classe Funcionário esteja relacionada a si mesma, podemos afirmar que o relacionamento é realmente entre um Funcionário exercendo a função de gerente e um Funcionário exercendo a função de membro da equipe.

Figura 18: um diagrama de classes mostrando a classe na Figura 14 em suas funções diferentes



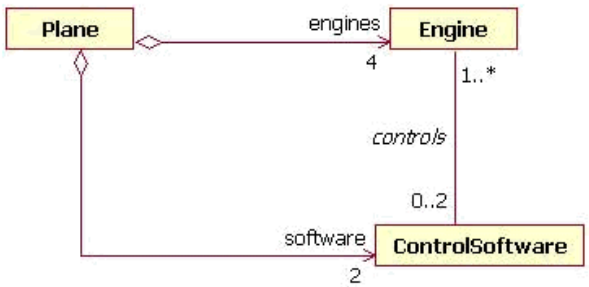
Observe que não é possível modelar uma função de classe em um diagrama de classes simples, embora a Figura 18 faça parecer que seja. Para usar a notação de função, será necessário usar a notação de Estrutura interna, discutida a seguir.

Estruturas internas

Um dos recursos mais úteis dos diagramas de estrutura UML 2 é a nova notação de estrutura interna. Ela permite mostrar como uma classe ou outro classificador é composto internamente. Isso não era possível na UML 1.x, porque o conjunto de notação o limitava a mostrar somente os relacionamentos de agregação que uma classe tinha. Agora, na UML 2, a notação de estrutura interna permite mostrar com mais clareza como essas partes da classe se relacionam entre si.

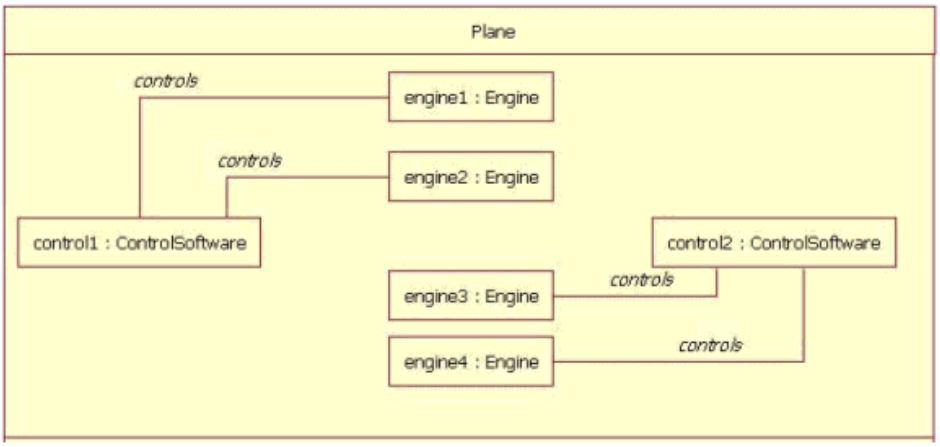
Vamos observar um exemplo. Na Figura 18, temos um diagrama de classes mostrando como uma classe Avião é composta de quatro mecanismos e dois objetos de software de controle. O que está faltando nesse diagrama é alguma informação de como as partes do avião são montadas. Observando o diagrama na Figura 18 não é possível afirmar se os objetos de software de controle controlam dois mecanismos cada um ou se um objeto de software de controle controla três mecanismos e os outros controlam um mecanismo.

Figura 19: um diagrama de classes que somente mostra os relacionamentos entre os objetos



Desenhar uma estrutura interna da classe melhorará esta situação. Comece desenhando uma caixa com dois compartimentos. O compartimento superior contém o nome da classe, e o compartimento inferior contém a estrutura interna da classe, mostrando as classes da parte da classe-pai em suas respectivas funções, além de como cada classe específica está relacionada às outras nessa função.. A Figura 19 mostra a estrutura interna da classe Avião. Observe como a estrutura interna elimina a confusão.

Figura 20: uma estrutura interna de exemplo de uma classe Avião



Na Figura 20 o Avião tem dois objetos ControlSoftware e cada um controla dois mecanismos. O ControlSoftware no lado esquerdo do diagrama (control1) controla os mecanismos 1 e 2. O ControlSoftware no lado esquerdo do diagrama (control2) controla os mecanismos 3 e 4.

Conclusão

Existem pelo menos dois motivos importantes para entender o diagrama de classes. O primeiro é que ele mostra a estrutura estática dos classificadores em um sistema. O segundo é que o diagrama fornece a notação básica para outros diagramas de estrutura prescritos pela UML. Os desenvolvedores podem pensar que o diagrama de classes foi criado especificamente para eles, mas outros membros da equipe também podem achá-los úteis. Os analistas de negócios podem usar diagramas de classes para modelar sistemas a partir da perspectiva do negócio. Como veremos em outros artigos nesta série sobre os fundamentos básicos da UML, outros diagramas — incluindo os diagramas de atividade, sequência e statechart — referem-se às classes modeladas e documentadas no diagrama de classes.

A seguir nesta série sobre os fundamentos básicos da UML: O diagrama de componente.

Temas relacionados: Usando os novos recursos do modelador de UML no IBM Rational Software Architect Versão 7.5 Visão geral do Rational Software Architect for WebSphere Software Versão 7.5 IBM Rational Software Delivery Platform

Sobre o autor

Donald Bell

Donald Bell é um IT Architect do software IBM Rational e trabalha com os clientes para definir e adotar práticas e ferramentas efetivas para entrega de software.