Reuso de Software Aula 20

Desenvolvimento Dirigido por Modelos (MDD)

Eduardo Figueiredo
http://www.dcc.ufmg.br/~figueiredo
reuso.software@gmail.com

16 Maio 2011

Tópicos da Aula

- Introdução a UML
  - Visão geral de alguns diagramas
  - Diagrama de Classes
- Desenvolvimento Dirigido por Modelos
  - Modelagem de software
  - Conceitos e Definições
  - O Processo MDD

Reusar cada vez mais...

- Reuso tem progredido bastante junto com a elevação da abstração
  - No início, havia reuso de pequenos fragmentos de código *assembly* para economizar memória
  - O reuso aumentou nas linguagens estruturadas com o conceito de procedimentos e funções
  - ...

Reusar cada vez mais...

- Objetos reusáveis encapsulam dados e comportamentos
- Criou-se as bibliotecas de classes e objetos
- Frameworks, componentes, etc. são algumas apostas mais atuais em termos de reuso de código

- O que mais poderia vir a seguir?

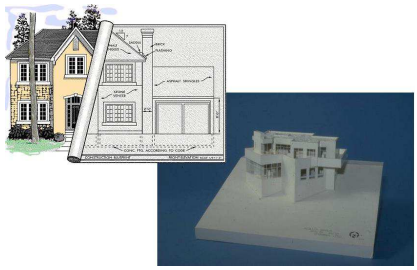
Reuso: Código x Modelos

- O reuso no nível de código é geralmente difícil
  - Envolve vários detalhes específicos da solução ou tecnologia adotada
- Solução seria elevar o nível de abstração
  - Passar ao reuso no nível de modelos
  - Modelos são mais abstratos e independentes de tecnologia

O que é modelagem?

- Atividade de construir modelos que expliquem as características ou comportamentos de um sistema
- Modelar software é tão essencial quanto ter uma planta de uma casa antes de construí-la

Modelar é Fundamental



A Linguagem UML

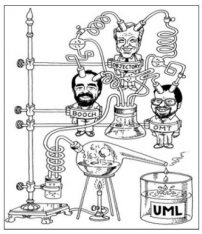
A Linguagem UML

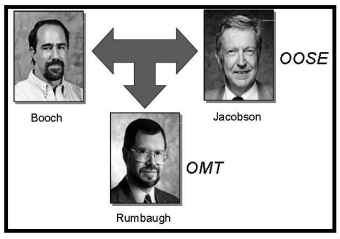
- UML (Linguagem de Modelagem Unificada)
- É uma notação gráfica (visual) para projetar sistemas OO
  - Não é uma linguagem de programação
- Define diagramas padronizados
- É extensível
- É complexa (muitos diagramas)
  - Mostrarei apenas um sub-conjunto da UML

De onde surgiu?

- Da união de três metodologias de modelagem
  - Método de Booch - Grady Booch
  - Método OMT - Ivar Jacobson
  - Método OOSE - James Rumbaugh
- Os três amigos

Fundadores da UML





História da UML

- 1994: Booch, Jacobson e Rumbaugh começaram a unificar suas notações
- 1996: Primeira versão (beta) da UML foi liberada
- 1996/97: Grandes empresas formaram a “UML Partners”
  - HP, IMB, Microsoft, Oracle, etc.
- 1997: UML foi adotada pela OMG (*Object Management Group*) como linguagem padrão de modelagem

Por que usar UML?

- Bons modelos são essenciais para a comunicação entre os *stakeholders*
- Padronização
  - Todo o time entende a modelagem, facilitando a manutenção
- Facilita a programação
  - Ferramentas para modelagem e geração de código (MDD)

Alguns Diagramas UML

- Diagramas Estruturais (Estáticos)
  - Diagrama de Casos de Uso
  - Diagrama de Classes
  - Diagramas de Objetos
  - Diagrama de Componentes, etc.
- Diagramas Dinâmicos
  - Diagrama de Sequência
  - Diagrama de Estados
  - Diagrama de Atividades
  - Diagrama de Colaboração, etc.

Visão Geral de Alguns Diagramas UML

Diagrama de Caso de Uso

- Diagrama mais geral da UML
- Usado geralmente na fase de Especificação de Requisitos
- Mostra
  - Quais usuários realizam que funcionalidades do sistema
  - Alguns relacionamentos entre estas funcionalidades

Diagrama de Caso de Uso

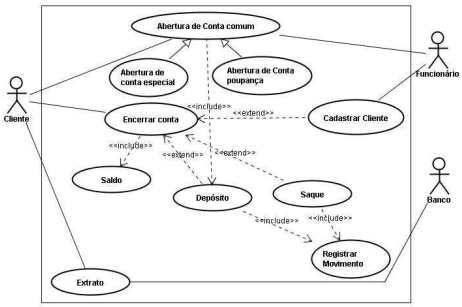


Diagrama de Sequência

- Preocupa-se com a ordem temporal em que as mensagens são trocadas
- Pode ser usado para detalhar um Caso de Uso
- Identifica
  - Os eventos associados a funcionalidade modelada
  - O ator responsável por este evento

Diagrama de Sequência

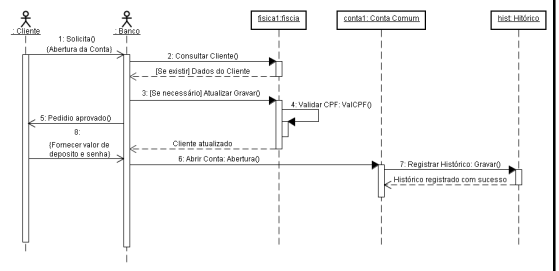


Diagrama de Classes

- Diagrama mais utilizado da UML
- Serve de apoio para a maioria dos outros diagramas
- Define a estrutura das classes do sistema
- Estabelece como as classes se relacionam

Diagrama de Classes

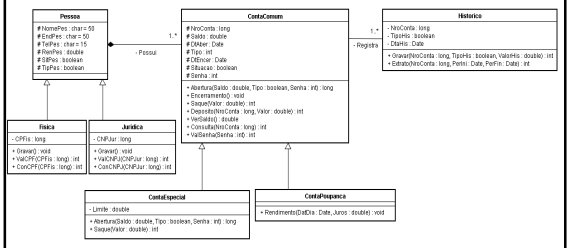


Diagrama de Objetos

- Complemento do Diagrama de Classes
- Exibe os valores armazenados pelos objetos de um Diagrama de Classes

Diagrama de Objetos

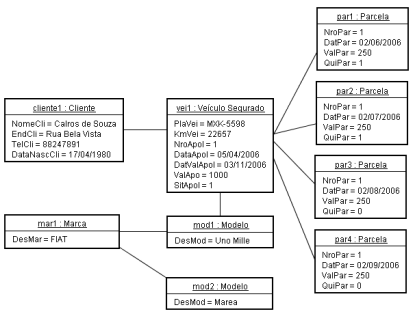
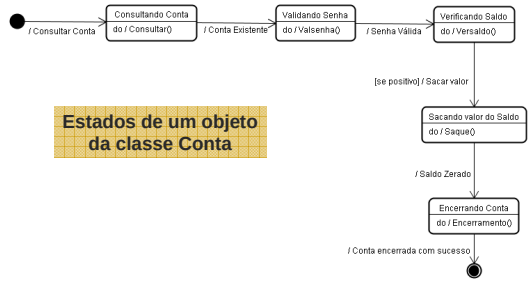


Diagrama de Estados

- Modela as mudanças sofridas por um objeto dentro de um determinado processo
- É utilizado para acompanhar os estados por que passa uma instância de uma classe

Diagrama de Estados



Estados de um objeto da classe Conta

Diagrama de Atividades

- Descreve as atividades a serem executadas para a conclusão de um processo
- Concentra-se na representação do fluxo de controle de um processo

Diagrama de Atividades

Processo: Validar Conta

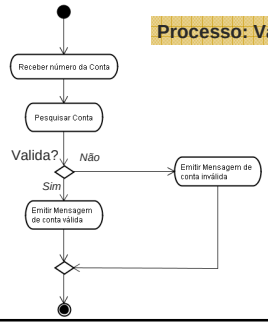


Diagrama de Classes

A estrutura do projeto

Diagrama de Classes

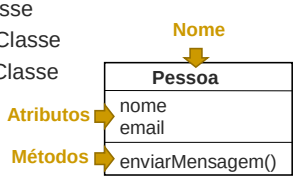
- Serve de apoio para a maioria dos outros diagramas
- Define a estrutura das classes do sistema
- Estabelece como as classes se relacionam

Diagrama de Classes

- Apresenta uma visão estática de como as classes estão organizadas
- Representa
  - Atributos e métodos de uma classe
  - Os relacionamento entre classes

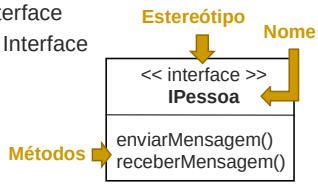
Representação de uma Classe

- Uma classe é representada por um retângulo com três divisões:
  - Nome da Classe
  - Atributos da Classe
  - Métodos da Classe



Representação de uma interface

- Uma interface é semelhante a uma classe, mas não tem atributos
- Uma interface possui
  - Nome da Interface
  - Métodos da Interface
- Estereótipo
  - interface

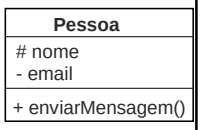


Tipos de visibilidade

- Pública (+)
  - O atributo ou método pode ser utilizado por qualquer classe
- Protegida (#)
  - Somente a classe ou sub-classes terão acesso
- Privada (-)
  - Somente a classe terá acesso

Tipos de visibilidade

- Pública (+)
  - O atributo ou método pode ser utilizado por qualquer classe
- Protegida (#)
  - Somente a classe ou sub-classes terão acesso
- Privada (-)
  - Somente a classe terá acesso



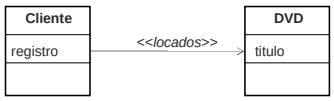
Relacionamento

- Classes possuem relacionamentos entre elas (para comunicação)
  - Compartilham informações
  - Colaboram umas com as outras
- Principais tipos de relacionamentos
  - Associação
  - Agregação / Composição
  - Herança
  - Dependência

Associações

- Descreve um vínculo entre duas classes
  - Chamado **Associação Binária**
- Determina que as instâncias de uma classe estão de alguma forma ligadas às instâncias da outra classe

Representação de Associação



Agregação

- Tipo especial de associação
- Demonstra que as informações e um objeto precisam ser complementadas por um objeto de outra classe
- Associação Todo-Parte
  - objeto-todo
  - objeto-parte

Representação de Agregação

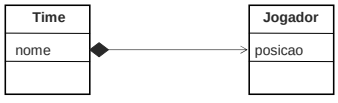
- Um losango na extremidade da classe que contém os *objetos-todo*

Composição

- Uma variação do tipo agregação
- Representa um vínculo mais forte entre objetos-todo e objetos-parte
- Objetos-parte **têm** que pertencer ao objeto-todo
  - O todo não existe (ou não faz sentido) sem a parte

Representação da Composição

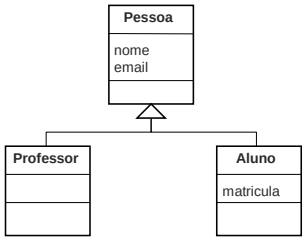
- Um losango preenchido, e da mesma forma que na Agregação, deve ficar ao lado do objeto-todo



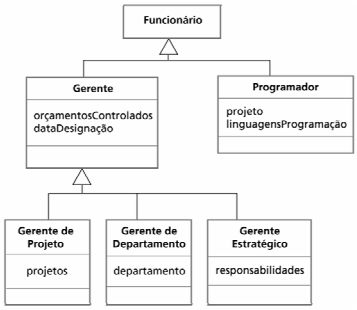
Especialização / Generalização

- Identificar super-classe (geral) e sub-classes (especializadas)
  - Semântica “é um”
  - Tudo que a classe geral pode fazer, as classes específicas também podem
- Atributos e métodos definidos na classe-mãe são **herdados** pelas classes-filhas

Especialização / Generalização

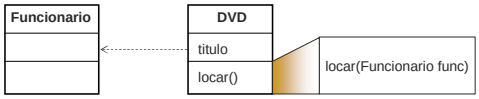


Hierarquia de Generalização



Dependência

- Identifica um baixo grau de dependência de uma classe em relação a outra
  - Representado por uma reta tracejada



Desenvolvimento Dirigido por Modelos (MDD)

Motivação para MDD

- Software é caro
  - Os EUA sozinho investem mais de $250 bilhões em software
  - Nos EUA, aproximadamente 175 mil projetos envolvem milhões de pessoas
  - Mais de 30% dos projetos são cancelados
  - Aproximadamente 30% dos projetos custam o dobro do que estimado

Demanda Crescente

- Apesar dos problemas, a demanda por software continua crescendo
- Nos últimos 50 anos, duas importantes iniciativas foram feitas para tentar atender a demanda
  - Linguagens mais expressivas e de mais alto nível foram propostas
  - Aumento do reuso de software

Nível de Abstração das Linguagens

- Histórico de desenvolvimento de software
  - Cartões perfurados (dados) e válvulas (processamento)
  - Linguagens assembly e pouco depois FORTRAN
  - COBOL e C trouxeram portabilidade de hardware
  - C++ e Java surgiram para implementar o conceito de Orientação a Objetos
    - Java trouxe portabilidade de SO

Aumento de Reuso

- Reuso tem progredido bastante junto com a elevação da abstração
  - Reuso de pequenos fragmentos de código assembly (economia de memória)
  - O reuso aumentou nas linguagens estruturadas com o conceito de procedimentos e funções
  - Surgiram bibliotecas de funções em programação orientada a objetos

Aumento de Reuso

  - Reuso de objetos que encapsulam dados e comportamentos
  - Criou-se as bibliotecas de classes e objetos
  - Frameworks, componentes, etc. são algumas apostas mais atuais em termos de reuso de código

- O que poderia vir a seguir?

Reuso de Modelos

- O desenvolvimento dirigido por modelos (MDD) seria um passo adicional para continuar o avanço
  - Elevar ainda mais o nível de abstração no desenvolvimento de software
  - Reusar partes maiores e mais abstratas
- Objetivo é não precisar implementar, mas reusar/gerar todo o sistema

Trabalho em Tempo de Projeto

- Mesmo com todas as abordagens para reuso de código, ainda há poucos casos de reuso de sistemas completos
  - É comum vermos sistemas sendo completamente re-implementados em uma tecnologia diferente
- MDD propõe que o desenvolvimento, manutenção, evolução e reuso sejam feitos no nível de projeto (modelos)

Por que?

- O reuso no nível de código é geralmente difícil
  - Envolve vários detalhes específicos da solução ou tecnologia adotada
- Solução: elevar o nível de abstração
  - Passar ao reuso no nível de modelos
  - Modelos são mais abstratos e independentes de tecnologia

Problemas

- Imaturidade do desenvolvimento dirigido por modelos
- Falta suporte de ferramentas e ambientes de desenvolvimento
- Modelos são vistos como extras
  - Código seria o principal
- Desenvolvedores são resistentes
  - Não gostam de “brincar” com figuras
  - Temem por seus empregos como programadores

Termos e Definições

O que são Modelos?

- Modelos são um conjunto de elementos que descrevem uma realidade de forma abstrata ou hipotética
  - São úteis para comunicação
  - São geralmente mais baratos de serem construídos que o objeto real
  - Ajudam o equipe a raciocinar sobre o problema

Abstração e Classificação

- Abstração significa ignorar informações que não são importantes em um determinado contexto
- Classificação significa agrupar elementos com base me propriedades comuns
  - Os elementos agrupados são diferentes

Modelos em MDD

- Modelos podem ser criados em diferentes níveis de abstração
  - Alguns modelos são independentes de plataforma (outros não)
- Modelos em MDD são uma combinação de
  - Diagramas inter-relacionados
  - Texto que conectam os diagramas

Metamodelos

- Um metamodelo é um modelo de uma linguagem de modelagem
  - Define a estrutura, semântica e limitações dos modelos
- Exemplo
  - Modelos UML são definidos pelo Metamodelo UML

Plataforma

- Plataforma é a especificação de um ambiente de execução para um conjunto de modelos
- MDD propõe a separação
  - Modelos independentes de plataforma (*platform independet models*) **PIM**
  - Modelos específicos de plataforma (*platform specific models*) **PSM**

Relacionamentos entre Modelos

- MDD permite desenvolvimento iterativo e incremental
  - Modelos podem ser representados em diferentes níveis de abstração
- Modelos possuem relacionamentos semânticos entre eles
  - Os relacionamentos permitem automatizar (parte da) geração de outros modelos

Criação de Modelos

- Em MDD, obviamente, os modelos devem ser criados
- Algumas regras na criação de modelos
  - Não crie um modelo se você não precisar
  - Não crie modelos intermediários que não ajudam a resolver o problema
  - Evite redundâncias entre os modelos

Modelos Executáveis

- Modelos executáveis definem tudo que é necessário para uma funcionalidade
  - E ainda permitem uma fácil comunicação com o cliente (e entre desenvolvedores)
- Agem exatamente como o código
  - Requer um compilador de modelos

Bibliografia da Aula (MDD)

- S. J. Mellor, K. Scott, A. Uhl, D. Weise. **MDA Distilled: Principles of Model-Driven Architecture**. Addison-Wesley, 2004.
  - Capítulos. 1 e 2
- Ian Sommerville. **Engenharia de Software**, 9ª Edição. Pearson Education, 2011.
  - 5.5 Engenharia Dirigida por Modelos

Bibliografia da Aula (UML)

- BOOCH, G., RUMBAUGH, J., JACOBSON, I. **UML, Guia do Usuário**. Rio de Janeiro: Campus, 2000.
- M. Fowler. **UML Essencial**, 3a Edição. Bookmann, 2004.