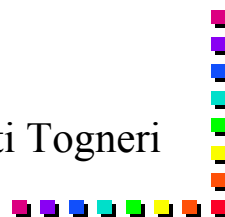


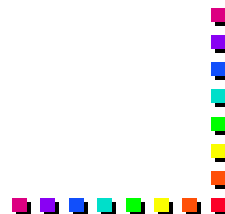


PROJETO ORIENTADO A OBJETOS

Professora: Denise Franzotti Togneri



INTRODUÇÃO



PROJETO ORIENTADO A OBJETOS ***INTRODUÇÃO***

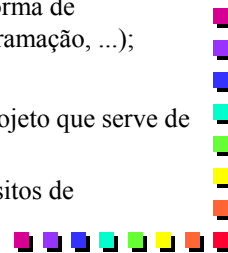
3

■ **ANÁLISE OO**

- Tecnologia “perfeita” disponível;
- Ênfase **no que** o sistema deve fazer;
- Identifica e define classes que refletem diretamente o domínio do problema e as responsabilidades do sistema dentro dele.

■ **PROJETO OO**

- Sabe-se que o sistema será construído em uma plataforma de implementação (HW, SO, SGBD, linguagem de programação, ...);
- Ênfase em **como** os requisitos serão implementados;
- Transforma o modelo de análise em um modelo de projeto que serve de base para a construção do software;
- Identifica e define classes adicionais, refletindo requisitos de implementação.

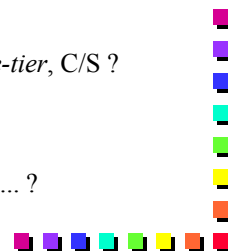


Denise F. Togneri

PROJETO ORIENTADO A OBJETOS - ***É DEPENDENTE DE:***

4

- Características da **linguagem de programação** a ser utilizada
 - Qual tipo de linguagem ? Orientada a objetos, a eventos, convencional, ?
 - Se for uma LPOO qual o nível de herança suportado (simples, múltipla) ?
 - Quais os mecanismos de acesso a atributos e operações ?
- Modelo de **Persistência de Objetos**
 - SGBDOO, SGBDR, arquivos, persistência na própria linguagem de programação ?
- Características da **Plataforma de Implementação**
 - A plataforma é multi-processada, a arquitetura é *three-tier*, C/S ?
 - O sistema é distribuído ?
- Características da **Interface com o usuário**
 - Qual tipo ? Interfaces gráficas, orientadas a caracter, ?



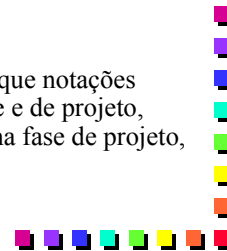
Denise F. Togneri

PROJETO ORIENTADO A OBJETOS

INTRODUÇÃO

5

- As ferramentas de modelagem utilizadas na fase de análise - Diagrama de Classes, Diagramas de Interação e Diagramas de Transição de Estados - são utilizadas também na fase de projeto, agora com o intuito de capturar os requisitos de implementação.
- Entretanto, a perspectiva de implementação existente no projeto demanda extensões à notação da análise para permitir representar
 - visibilidade
 - persistência
 - concorrência
 - exceções
 - restrições.
- Assim, ainda que a UML não especifique explicitamente que notações destes diagramas devem ser utilizadas nas fases de análise e de projeto, algumas de suas facilidades devem ser utilizadas apenas na fase de projeto, tais como
 - as notações de visibilidade de atributos e operações e
 - navegabilidade de relacionamentos.



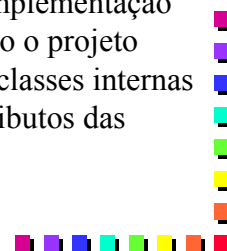
Denise F. Togneri

PROJETO ORIENTADO A OBJETOS

INTRODUÇÃO

6

- De modo geral, dois grandes passos do processo de projeto OO podem ser identificados (apesar dos diferentes métodos de projeto OO) (Pressman, 2002):
 - **Projeto da Arquitetura OO do Sistema:** descreve cada um dos subsistemas, de um modo passível de implementação, e as comunicações entre eles;
 - **Projeto de Objetos:** descreve aspectos de implementação de cada uma das classes do sistema, incluindo o projeto procedural de cada operação, a definição de classes internas e o projeto de estruturas de dados para os atributos das classes.

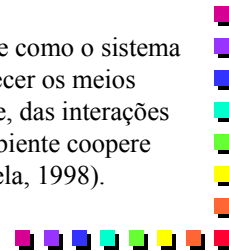


Denise F. Togneri

PROJETO ARQUITETURAL OO

7

- É a primeira tarefa a ser realizada em um projeto OO.
- Consiste em definir uma arquitetura para a aplicação. Será sobre esta arquitetura que o projetista poderá introduzir os aspectos de implementação em um modelo de análise.
- Em sistemas complexos, a definição da arquitetura do software deve ser iniciada durante a fase de análise para permitir uma melhor organização dos modelos de análise, evitando a complexidade.
- “A arquitetura de um software deverá fornecer resposta de como o sistema irá funcionar, em um ambiente operacional e deverá fornecer os meios necessários para a definição dos componentes do software, das interações entre eles e os padrões necessários para que todo esse ambiente coopere para produzir o software que está sendo projetado” (Magela, 1998).

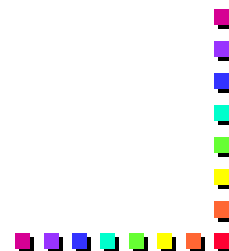


Denise F. Togneri

PROJETO ARQUITETURAL OO DOIS PRINCÍPIOS FUNDAMENTAIS

8

- Uma boa arquitetura de software pode ser obtida através da aplicação de **dois princípios fundamentais** (Magela, 1998):
 - **Produção de software em camadas com níveis de abstração definidos;**
 - **Separação entre interface e implementação.**
- Exemplos de softwares produzidos em camadas:
 - Banco de Dados: separa o físico do lógico;
 - Redes: Modelo ISO em camadas;
 - Sistema Operacional: Windows NT, OS/2;
 - UML: visões distintas para um mesmo projeto;
 - CORBA: separação entre interface e implementação;
 - Linguagem JAVA: independência de plataforma.



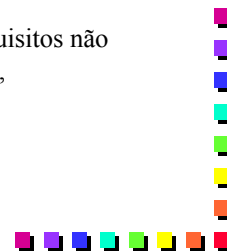
Denise F. Togneri

PROJETO ARQUITETURAL OO

1º PRINCÍPIO: PRODUÇÃO DO SW EM CAMADAS

9

- Uma boa arquitetura de software OO deve ser pensada em termos dos componentes que deverão compor o software.
- Não é mais possível entender **todo** o projeto do software, sendo necessário subdividi-lo em componentes gerenciáveis e com complexidade reduzida. Esta quebra deve ser refletida no projeto arquitetural do sistema.
- A visão arquitetural deve levar em conta, também, os requisitos não funcionais do sistema, tais como: segurança, desempenho, manutenibilidade,



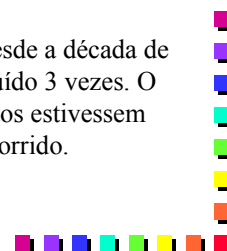
Denise F. Togneri

PROJETO ARQUITETURAL OO

2º PRINCÍPIO: SEPARAÇÃO ENTRE INTERFACE E IMPLEMENTAÇÃO

10

- **Deve-se separar/isolar, sempre que possível, o impacto da tecnologia para poder manter o modelo tão independente de plataforma quanto possível.**
- **Exemplo:** a troca do modelo de persistência não deve afetar tanto as demais partes do sistema.
- **Estudo de Caso:** O Sistema de Matrícula da Ufes que, desde a década de 80 não sofreu alterações em seus requisitos, já foi substituído 3 vezes. O sistema mudou porque a tecnologia mudou. Se os requisitos estivessem isolados do impacto da tecnologia, isto poderia não ter ocorrido.

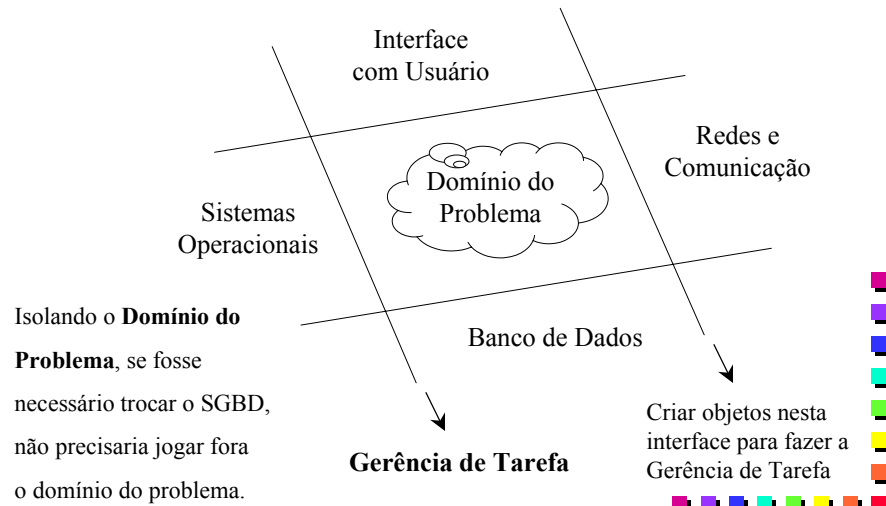


Denise F. Togneri

PROJETO ARQUITETURAL OO

2º PRINCÍPIO: SEPARAÇÃO ENTRE INTERFACE E IMPLEMENTAÇÃO

11



Denise F. Togneri

PROJETO ARQUITETURAL OO

12

■ PROJETO ESTRUTURADO

- Os DFDs (contendo processos) não são parecidos com o DEM (contendo módulos).
- Transformação dos modelos da Análise para modelos do Projeto

■ PROJETO OO

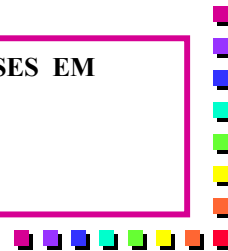
- Não ocorre transformação, mas sim:
- O modelo da análise é o primeiro grande modelo do projeto. Os ajustes são somente referentes à incorporação de requisitos em função da tecnologia não ser perfeita.
- No entanto, as classes, associações, ... continuam a existir.
- No Modelo de Classes já existente, serão acrescentados novas classes.

Denise F. Togneri

PROJETO ARQUITETURAL OO

13

- **PONTO DE PARTIDA PARA DEFINIÇÃO DA ARQUITETURA**
 - Os Casos de Uso
 - Organização de Classes em Pacotes
- **OBJETIVOS DA ORGANIZAÇÃO DE CLASSES EM PACOTES**
 - **Fornecer níveis de abstração para o modelo**, que podem ser organizados em camadas e, assim, tratados separadamente durante a fase de projeto.
 - **Garantir a produção de componentes para reuso.**
- **ALTERNATIVAS DA ORGANIZAÇÃO DE CLASSES EM PACOTES**
 - **Organização por Estereótipos**
 - **Organização por Domínio do Problema**

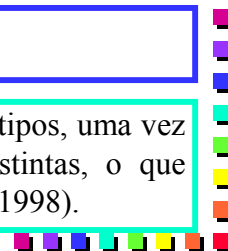


Denise F. Togneri

PROJETO ARQUITETURAL OO ORGANIZAÇÃO DE CLASSES POR ESTEREÓTIPOS

14

- Estereótipo é o tipo de função que uma classe exerce no sistema.
- Classes são agrupadas de acordo com seus estereótipos: **classes de negócio, de interface, de controle e de gerência de dados**
- Uma classe possui somente um estereótipo.
- Não podemos ter uma classe com dois estereótipos, uma vez que ela possuiria duas responsabilidades distintas, o que deveria levar a duas classes distintas (Magela, 1998).

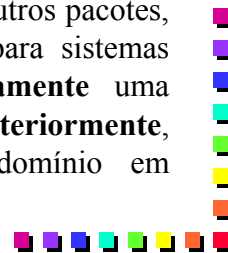


Denise F. Togneri

PROJETO ARQUITETURAL OO ORGANIZAÇÃO DE CLASSES POR DOMÍNIO DO PROBLEMA

15

- Conduz à construção de pacotes verticais, levando à produção de componentes de negócio (Magela, 1998).
- Esta abordagem isolada é, normalmente, insuficiente.
- Assim, uma vez que um pacote pode conter outros pacotes, uma abordagem mais eficiente, sobretudo para sistemas complexos, consiste em realizar **primeiramente** uma organização por domínio do problema e, **posteriormente**, fazer uma subdivisão dos pacotes do domínio em estereótipos.

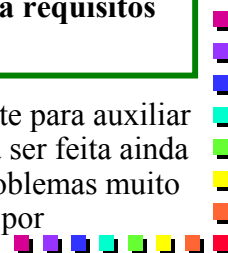


Denise F. Togneri

PROJETO ARQUITETURAL OO ORGANIZAÇÃO DE CLASSES POR DOMÍNIO DO PROBLEMA

16

- Quando esta abordagem baseada no domínio do problema for adotada, o primeiro passo a ser dado consiste em particionar o modelo de análise para definir coleções coesas de classes, relacionamentos e comportamento, empacotando-os em *pacotes* ou *subsistemas*.
- **Este passo é uma revisão da identificação de subsistemas feita na fase de análise, agora levando em conta requisitos de implementação.**
- Subsistemas podem ser particionados internamente para auxiliar a reduzir a complexidade. Esta subdivisão poderá ser feita ainda segundo o critério domínio do problema (para problemas muito complexos) ou usando o critério de agrupamento por estereótipos.



Denise F. Togneri

PROJETO ARQUITETURAL OO ORGANIZAÇÃO DE CLASSES POR DOMÍNIO DO PROBLEMA

17

- Subsistemas devem ser definidos e projetados em conformidade com os seguintes critérios (Pressman, 2002):
 - um subsistema deve possuir uma interface bem definida através da qual toda comunicação com o restante do sistema ocorre;
 - com exceção de um pequeno número de “classes de comunicação”, as classes dentro de um subsistema devem colaborar apenas com outras classes deste mesmo subsistema;
 - o número de subsistemas deve ser mantido pequeno.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

18

- Comunidade de Smalltalk: desenvolveu uma **metáfora** simples, para uma arquitetura de projeto, conhecida como **Modelo MCV** - “**Modelo-Visão-Controlador**” (*Model-View-Controller*)
- Essa **metáfora** sugere que uma arquitetura típica de projeto OO possui três componentes principais:
 - um grupo de classes que *modela* a aplicação em si;
 - um grupo de classes que provê uma *visão* da interface com os usuários;
 - um grupo de classes que *controla*, ou sincroniza, o comportamento dos demais.
- A arquitetura MCV desconsidera um importante componente: a **gerência de dados**, pois, em Smalltalk, todos os objetos são naturalmente persistentes.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

19

- Durante o projeto da arquitetura OO do sistema, um engenheiro de software deve considerar quatro (e não apenas três) componentes básicos (Coad e Yourdon, 1993):

- **Componente do Domínio do Problema:** corresponde aos subsistemas responsáveis por implementar diretamente os requisitos dos usuários; **o modelo de análise suporta este componente.**
- **Componente de Interação Humana:** corresponde aos subsistemas que implementam as interfaces com o usuário;
- **Componente de Gerência de Tarefa:** corresponde aos subsistemas responsáveis por controlar e coordenar tarefas;
- **Componente de Gerência de Dados:** corresponde aos subsistemas responsáveis pelo armazenamento e recuperação de objetos (persistência dos objetos).



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

20

Componente de Domínio do Problema (CDP)	Componente de Interação Humana (CIH)	Componente de Gerência de Tarefa (CGT)	Componente de Gerência de Dados (CGD)

Arquitetura Básica de Projeto Orientado a Objetos (Coad e Yourdon, 1993).

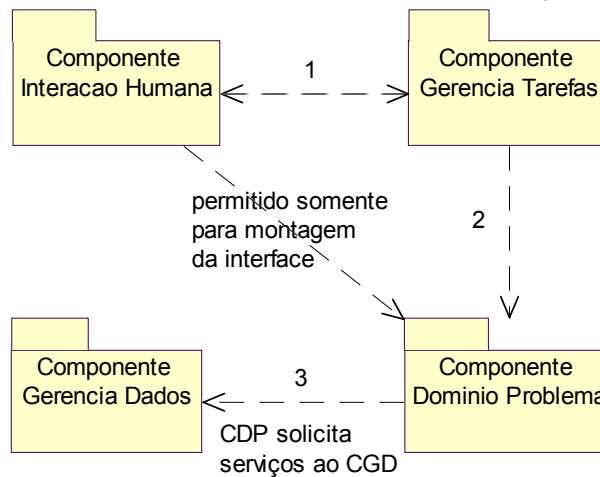


Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO RELAÇÃO ENTRE OS COMPONENTES

21

Entre a CIH e a CGT: deve-se decidir no projeto qual componente toma a iniciativa de solicitar os serviços !



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

22

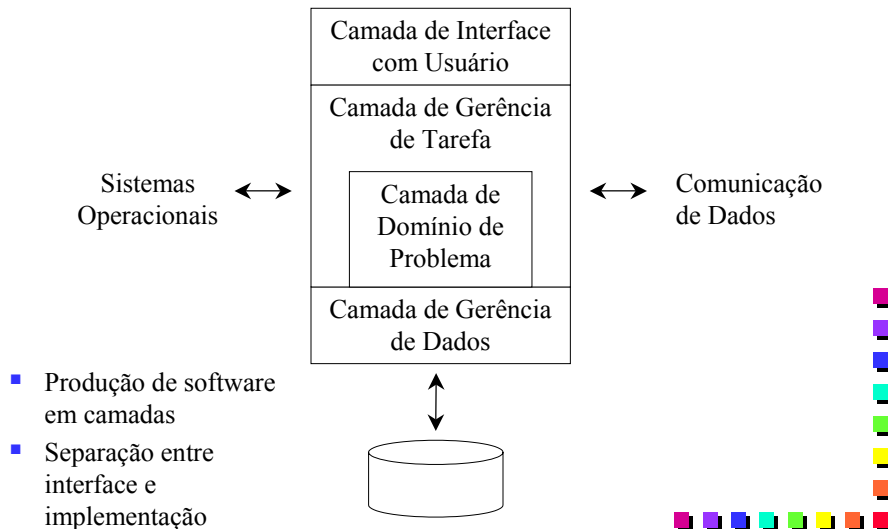
■ IDÉIA BÁSICA DA ARQUITETURA

- Buscar as mesmas classes que foram documentadas no modelo de análise e envolvê-las com classes adicionais para tratar aspectos relacionados à implementação de gerência de tarefa, gerência de dados e interação humana.
- Essa arquitetura não só preserva o modelo de análise, como também o utiliza como o cerne do modelo de projeto.

Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

23



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

24

- **FILOSOFIA DE PROJETO POR DETRÁS DESSA ARQUITETURA**
 - Sugere que as classes centrais, orientadas à aplicação na CDP, não devem estar cientes do “mundo exterior” e não têm de saber como interagir com tal mundo.
 - **Sem uma atenção consciente a esta filosofia, podemos chegar a uma arquitetura na qual cada classe:**
 - sabe como interagir com o usuário final
 - sabe como ler e escrever seus dados permanentes em arquivos de disco.
 - Uma abordagem assim poderia funcionar (quem sabe até de maneira mais rápida), mas seria muito suscetível a mudanças na interface com o usuário ou no modelo de persistência.
 - Além disto, tornaria a estrutura interna das classes mais complexa do que se tivessem de estar cientes apenas de seus detalhes essenciais, ligados ao domínio da aplicação.

Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTES DE PROJETO

25

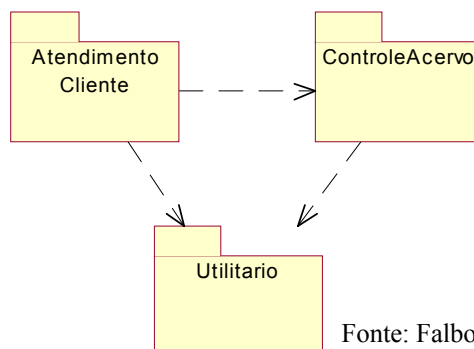
- **PRODUTO**
 - **Diagrama de Classes de Pacotes**
- Seguindo a arquitetura básica proposta por Coad e Yourdon (1993), temos quatro estereótipos:
 - Domínio do Problema
 - Interface com o Usuário
 - Gerência de Tarefas
 - Gerência de Dados
- A seguir, estudaremos cada um deles.



Denise F. Togneri

PROJETO ARQUITETURAL OO EXEMPLO DE ORGANIZ. DE CLASSES POR DOMÍNIO DO PROBLEMA E POR ESTEREÓTIPOS

Projeto de Arquitetura do Software Diagrama de Classes Principal de Projeto

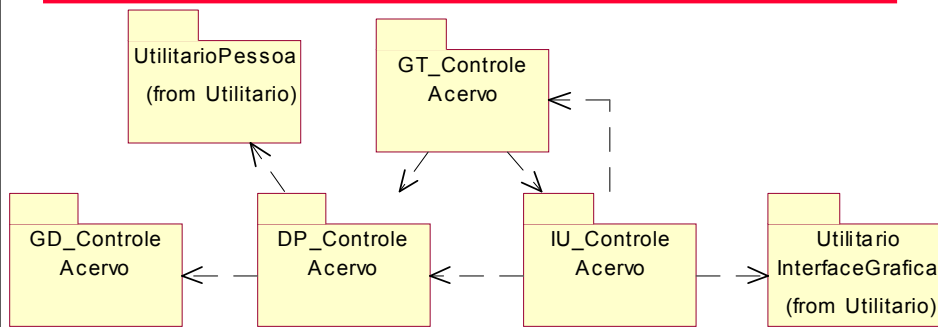


Fonte: Falbo, 2003.

Inicialmente, as classes foram agrupadas pelo **domínio do problema**, aproveitando os subsistemas definidos na Análise, sendo introduzido o pacote Utilitario que trata classes reutilizáveis em outros contextos.

PROJETO ARQUITETURAL OO
EXEMPLO DE ORGANIZ. DE CLASSES POR
DOMÍNIO DO PROBLEMA E POR ESTEREÓTIPOS

Projeto de Arquitetura do Software
Diagrama de Classes – Pacote ControleAcervo



Fonte: Falbo, 2003.

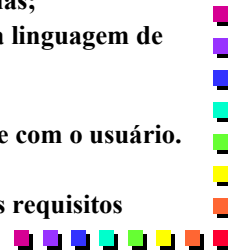
O pacote ControleAcervo foi decomposto no Diagrama de Classes de Pacotes, agora tomando por base os **estereótipos**.

COMPONENTE DE DOMÍNIO
DO PROBLEMA

PROJETO ARQUITETURAL OO **COMPONENTE DE DOMÍNIO DO PROBLEMA**

29

- No Projeto OO, os resultados da Análise OO fazem parte do Componente de Domínio do Problema (CDP).
- Algumas vezes, o modelo de análise desenvolvido pode ser transposto para dentro do CDP sem qualquer alteração adicional, outras não. Ele pode ser modificado ou estendido, de acordo com as necessidades do projeto.
- Levando em conta requisitos de implementação, as alterações neste componente podem advir da necessidade de:
 - reutilizar projetos anteriores e classes já programadas;
 - ajustar o modelo ao nível de herança suportado pela linguagem de programação;
 - ajustar o modelo para melhorar o desempenho;
 - ajustar o modelo para facilitar o projeto de interface com o usuário.
- O CDP pode ser alterado, ainda, para comportar outros requisitos tecnológicos, tais como segurança, confiabilidade, etc.



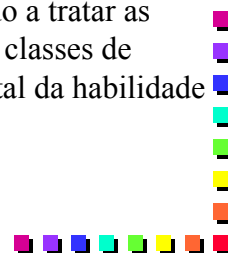
Denise F. Togneri

PROJETO ARQUITETURAL OO **ALTERAÇÕES NO CDP**

30

Reutilização de projetos anteriores e classes já programadas

- Levar em conta a existência de bibliotecas de classes passíveis de serem reusadas.
- Tipicamente, envolvem alterações nas hierarquias de generalização-especialização do modelo, de modo a tratar as classes apropriadas da OOA como subclasses de classes de biblioteca pré-existent, obtendo a vantagem total da habilidade de herdar atributos e métodos de tais classes.



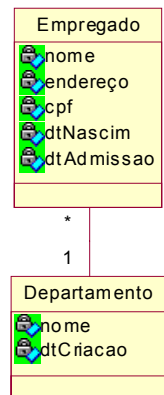
Denise F. Togneri

PROJETO ARQUITETURAL OO ALTERAÇÕES NO CDP

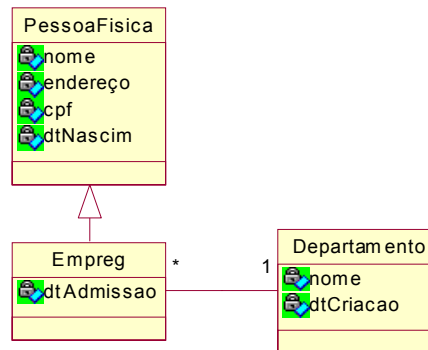
Reutilização de projetos anteriores e classes já programadas

- Exemplo: Como na empresa já existia uma classe **PessoaFisica**, deve-se fazer uma herança.

MODELO DE ANÁLISE OO



MODELO DE PROJETO OO

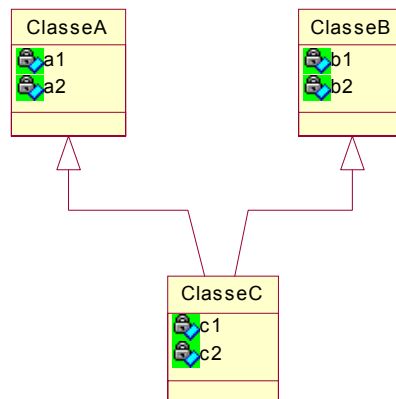


PROJETO ARQUITETURAL OO ALTERAÇÕES NO CDP

Ajuste do modelo ao nível de herança suportado pela linguagem de programação

- Exemplo: Herança múltipla não suportada pela linguagem de implementação.

MODELO DE ANÁLISE OO

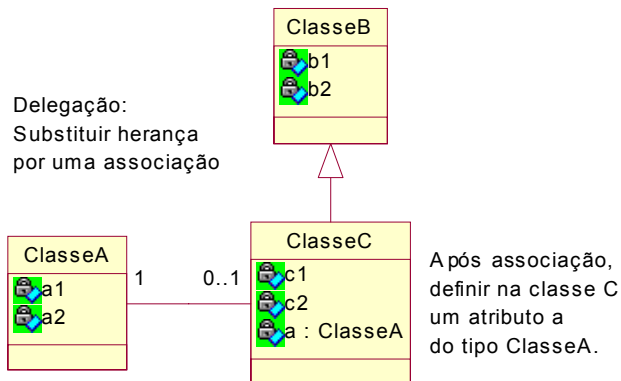


PROJETO ARQUITETURAL OO ALTERAÇÕES NO CDP

Ajuste do modelo ao nível de herança suportado pela linguagem de programação

- **Solução:** Verificar se a ClasseC é mais parecida (mais natural) com a ClasseA ou ClasseB. Supondo que ClasseC seja mais parecida com ClasseB:

MODELO DE PROJETO OO



PROJETO ARQUITETURAL OO ALTERAÇÕES NO CDP

34

Ajuste do modelo para melhorar o desempenho

- Pode ser uma preocupação se há um alto tráfego de mensagens entre objetos, se a linguagem de programação implementa herança ineficientemente, etc.
- Nestes casos, o projetista pode alterar o modelo de análise para melhor acomodar os ajustes necessários.
- **Exemplo:** inclusão de campos de saldo.

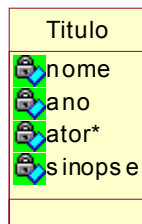


PROJETO ARQUITETURAL OO ALTERAÇÕES NO CDP

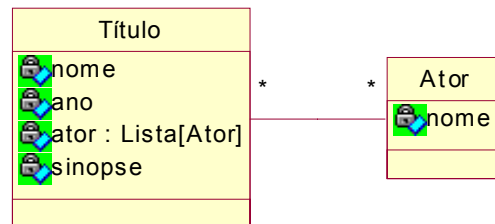
Ajuste do modelo para facilitar o projeto de interface com o usuário

- **Exemplo:** criação de classes em função da interface ==> atributo ator vira classe para apoiar a pesquisa por ator (apresentação de listas para seleção, atendendo ao critério de qualidade “facilidade de uso”).

MODELO DE ANÁLISE OO



MODELO DE PROJETO OO



A associação **pode ser** mapeada criando um atributo ator do tipo Lista[Ator] (lista de objetos do tipo Ator) e não do tipo String.

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ³⁶ TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE DAS ASSOCIAÇÕES

- A navegabilidade é uma característica da associação (da mesma forma que papel, cardinalidade, por exemplo), que é definida em tempo de projeto OO.
- A decisão sobre a navegabilidade da associação conduzirá à definição de novas variáveis nas classes, bem como do seu tipo e estrutura de dados.
- Representa a **navegação “direta”** de um objeto até outros objetos a que esteja vinculado, geralmente porque o objeto de origem armazena algumas referências dos objetos de destino.

Para representar explicitamente a direção da navegação, deve-se incluir uma seta na direção a ser seguida (Booch et al., 2000).



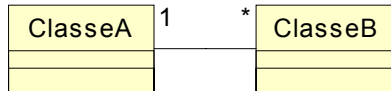
PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ³⁷

TRADUÇÃO DE RELACIONAMENTOS

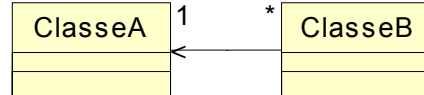
NAVEGABILIDADE UNIDIRECIONAL DAS ASSOCIAÇÕES 1:N

NOTAÇÃO

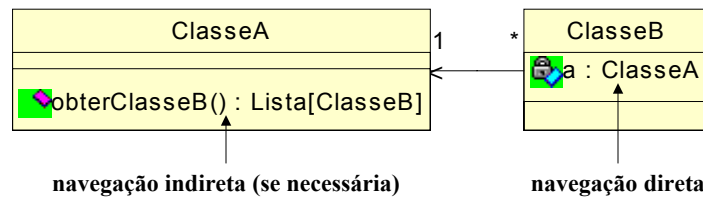
1 - DIAGRAMA DE CLASSES DE ANÁLISE



2 - DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE UNIDIRECIONAL



3 - SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE UNIDIRECIONAL



Denise F. Togneri

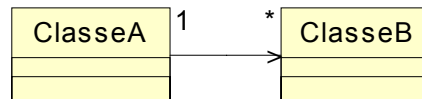
PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ³⁸

TRADUÇÃO DE RELACIONAMENTOS

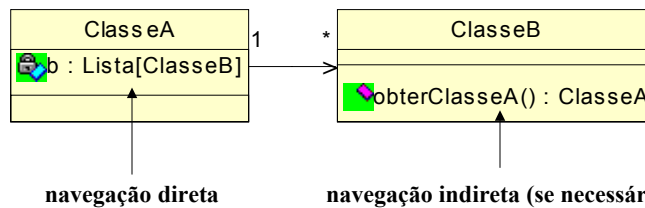
NAVEGABILIDADE UNIDIRECIONAL DAS ASSOCIAÇÕES 1:N

NOTAÇÃO

DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE UNIDIRECIONAL



SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE UNIDIRECIONAL



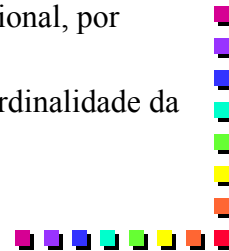
Denise F. Togneri

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP 39

TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE BIDIRECIONAL DAS ASSOCIAÇÕES NOTAÇÃO

- Se for tomada a decisão de construir uma navegação bidirecional, ou seja, os dois lados da associação possuírem navegabilidade direta, não é necessário desenhar as setas, pois, a menos que seja especificado o contrário, toda navegação em uma associação é bidirecional, por *default*.
- Essa observação é válida independente da cardinalidade da associação.



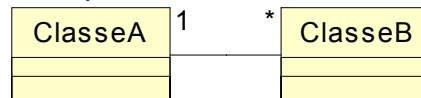
Denise F. Togneri

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP 40

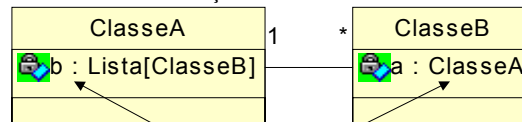
TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE BIDIRECIONAL DAS ASSOCIAÇÕES 1:N NOTAÇÃO

DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL

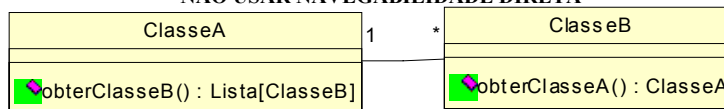


SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL



navegação direta bidirecional

CONFUSÃO POSSÍVEL DA NOTAÇÃO: REPRESENTAÇÃO DA OPÇÃO POR NÃO USAR NAVEGABILIDADE DIRETA

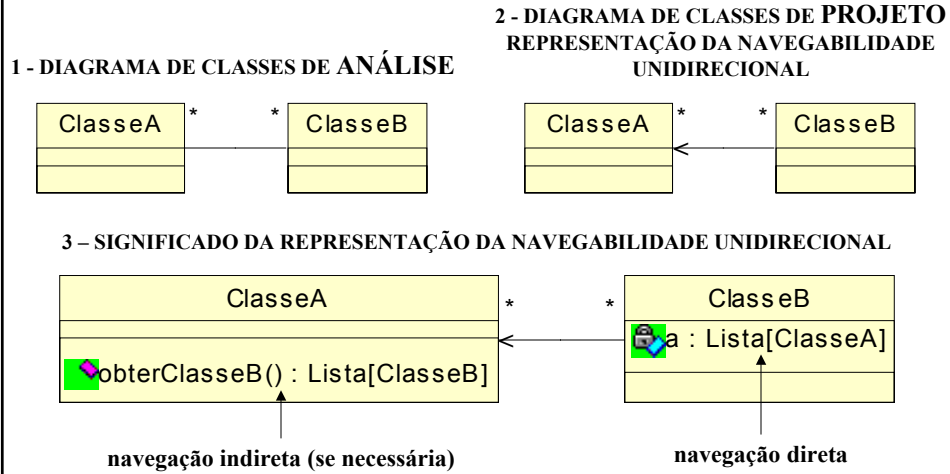


PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP 41

TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE UNIDIRECIONAL DAS ASSOCIAÇÕES N:N

NOTAÇÃO



Denise F. Togneri

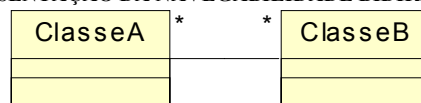
PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP 42

TRADUÇÃO DE RELACIONAMENTOS

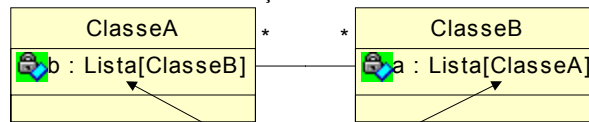
NAVEGABILIDADE BIDIRECIONAL DAS ASSOCIAÇÕES N:N

NOTAÇÃO

DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL

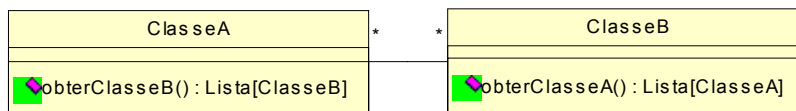


SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL



navegação direta bidirecional

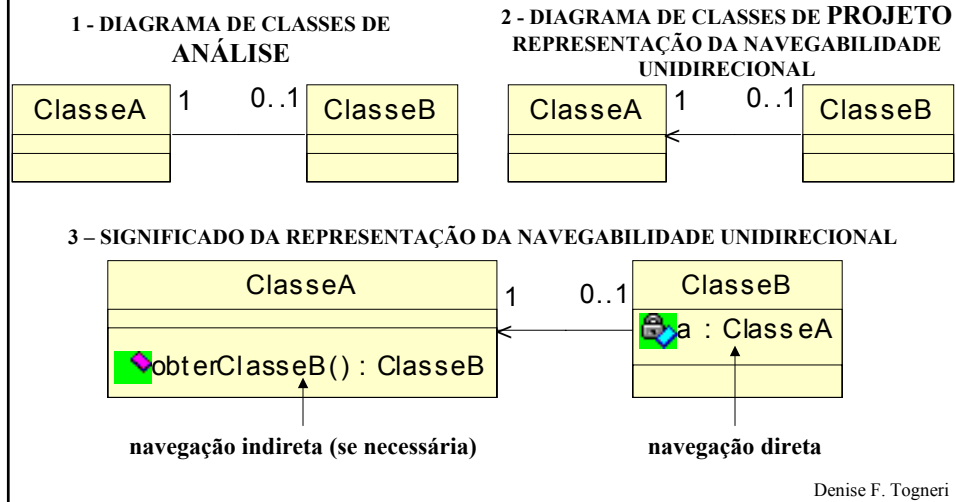
CONFUSÃO POSSÍVEL DA NOTAÇÃO: REPRESENTAÇÃO DA OPÇÃO POR NÃO USAR NAVEGABILIDADE



PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴³

TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE UNIDIRECIONAL DAS ASSOCIAÇÕES 1:1 NOTAÇÃO

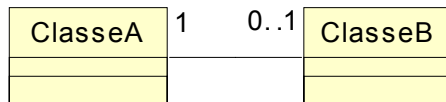


PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴⁴

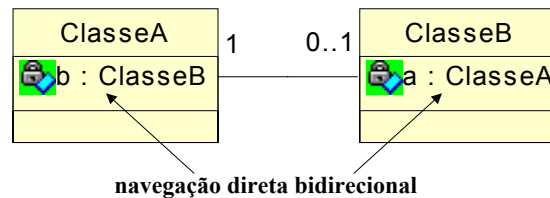
TRADUÇÃO DE RELACIONAMENTOS

NAVEGABILIDADE BIDIRECIONAL DAS ASSOCIAÇÕES 1:1 NOTAÇÃO

DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL



SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE BIDIRECIONAL

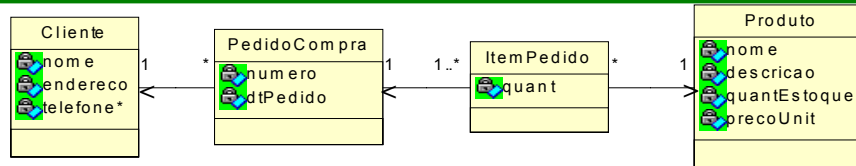


Denise F. Togneri

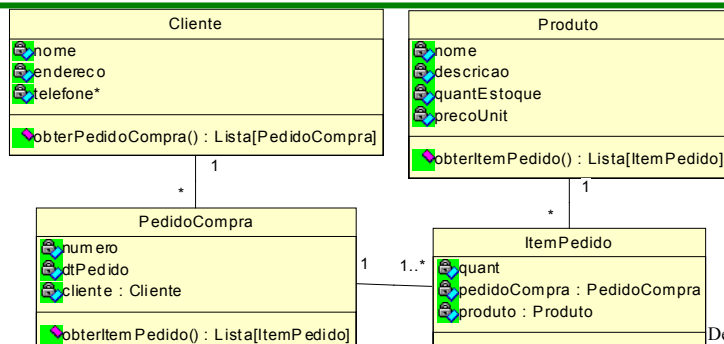
PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴⁵

TRADUÇÃO DE RELACIONAMENTOS

EXEMPLO DE DIAGRAMA DE CLASSES DE PROJETO REPRESENTAÇÃO DA NAVEGABILIDADE DAS ASSOCIAÇÕES



SIGNIFICADO DA REPRESENTAÇÃO DA NAVEGABILIDADE



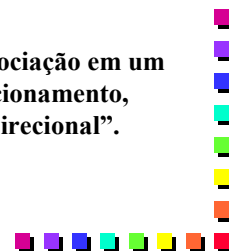
Denise F. Togneri

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴⁶

TRADUÇÃO DE RELACIONAMENTOS

GUIA PARA ESCOLHA DA NAVEGABILIDADE DAS ASSOCIAÇÕES

- Na **Análise OO**, as associações possuem, por definição, navegabilidade bidirecional. Contudo, a implementação de associações bidirecionais é muito cara ao sistema.
- Sendo assim, deve-se procurar um meio de tratar essas associações, tentando reduzir o bidirecionamento existente na associação.
- Magela (1998)** propõe a seguinte regra: “Para toda associação em um projeto, deverá ser analisada a navegabilidade do relacionamento, tornando-se, sempre que possível, uma associação unidirecional”.



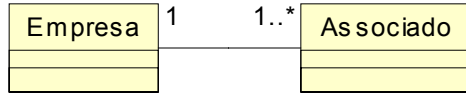
Denise F. Togneri

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴⁷

TRADUÇÃO DE RELACIONAMENTOS

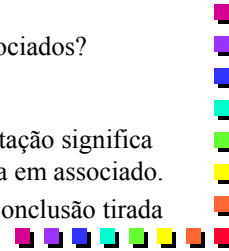
GUIA PARA ESCOLHA DA NAVEGABILIDADE DAS ASSOCIAÇÕES

Dado o Diagrama de Classes abaixo (Magela, 1998):



Para guiar a navegabilidade do relacionamento, inicialmente, duas perguntas básicas devem ser feitas em cada lado do relacionamento:

- 1 - Dado um associado, precisarei saber quem é sua empresa?
→ Navegação de associado para empresa.
 - 2 - Dado uma empresa, precisarei saber quem são todos os associados?
→ Navegação de empresa para associado.
- Se a Pergunta 1 tiver resposta **NÃO**, a nível de implementação significa que não será preciso ter um ponteiro (referência) de empresa em associado.
 - Se a Pergunta 2 tiver resposta **NÃO**, tiraremos a mesma conclusão tirada em 1.



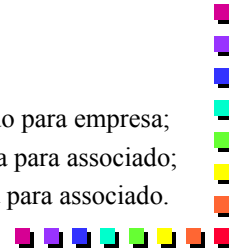
Denise F. Togneri

PROJ. ARQUIT. OO - ALTERAÇÕES NO CDP ⁴⁸

TRADUÇÃO DE RELACIONAMENTOS

GUIA PARA ESCOLHA DA NAVEGABILIDADE DAS ASSOCIAÇÕES

- Se a resposta for **SIM** nos dois lados do relacionamento, mais 3 perguntas devem ser feitas (Magela, 1998):
 - (1) Qual a frequência em que um associado precisará saber qual a sua empresa?
 - (2) Qual a frequência em que uma empresa precisará saber quem são seus associados?
 - (3) Existem realmente muitos objetos do tipo empresa (do lado muito do relacionamento)?
- A análise às respostas será (Magela, 1998):
 - Se (1) for baixa, desprezar a navegabilidade de associado para empresa;
 - Se (2) for baixa, desprezar a navegabilidade em empresa para associado;
 - Se (3) for baixa, desprezar a navegabilidade de empresa para associado.



Denise F. Togneri

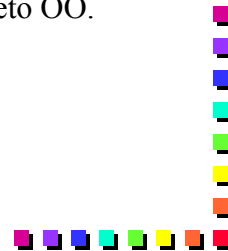
PROJETO ARQUITETURAL OO ***COMPONENTE DE DOMÍNIO DO PROBLEMA***

49

- **PRODUTOS**

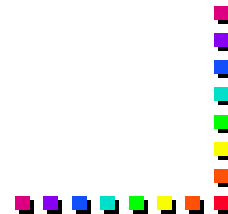
- Diagrama de Classes, Diagramas de Interação, de Estados e Atividades (em nível de Projeto)

- Nesta fase, os diagramas da Análise OO são alterados/transformados em diagramas de Projeto OO.



Denise F. Togneri

COMPONENTE DE ***INTERAÇÃO HUMANA***



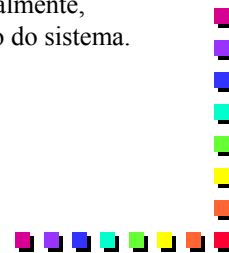
PROJETO ARQUITETURAL OO COMPONENTE DE INTERAÇÃO HUMANA

51

■ PREMISSA FUNDAMENTAL

- a porção do sistema que lida com a interface com o usuário deve ser mantida tão independente e separada do resto da arquitetura do software quanto possível.
- **Razão:** aspectos de interface com o usuário provavelmente serão alvo de alterações ao longo de toda a vida produtiva do sistema, e essas alterações devem ter um impacto mínimo (idealmente, *nenhum* impacto) nas partes específicas da aplicação do sistema.

- Utilização de prototipação.



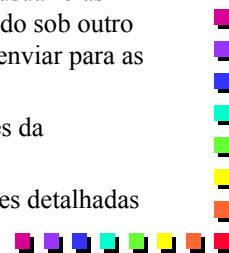
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE INTERAÇÃO HUMANA

52

■ PONTO DE PARTIDA PARA O PROJETO DA CIH

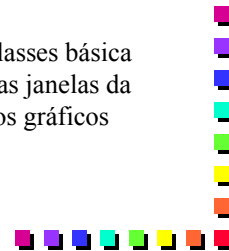
- casos de uso e seus cenários
- descrições de atores
- Com base nos casos de uso, devemos projetar uma **hierarquia de comandos**, definindo barras de menu, menus drop-down, ícones, etc., que levem a ações quando acionados pelo usuário.
- **Hierarquia de comandos:** é um meio de apresentar ao usuário as várias funcionalidades disponíveis no sistema, ou, olhando sob outro ponto de vista, as várias mensagens que o usuário pode enviar para as classes dentro do sistema.
- Todos os casos de uso devem ser implementados, através da navegação nesta hierarquia.
- Uma vez definida a hierarquia de comandos, as interações detalhadas entre o usuário e o sistema devem ser projetadas.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE INTERAÇÃO HUMANA

- Normalmente, não é necessário projetar as classes básicas de interfaces gráficas com o usuário.
- Existem vários ambientes de desenvolvimento de interfaces, oferecendo classes reutilizáveis (janelas, ícones, botões, ...) e, portanto, basta especializar as classes e instanciar os objetos que possuem as características apropriadas para o problema em questão.
- Em ambientes com interfaces gráficas, a hierarquia de classes básica para o CIH terá tipicamente uma superclasse “janela” e as janelas da aplicação serão construídas adicionando os outros objetos gráficos necessários, tais como botões, menus, ícones.

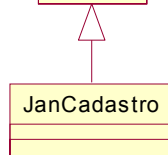
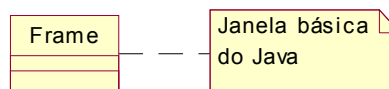


Denise F. Togneri

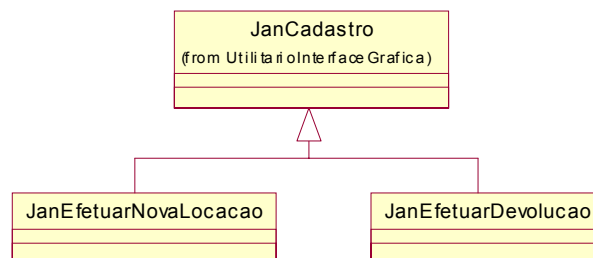
PROJETO ARQUITETURAL OO COMPONENTE DE INTERAÇÃO HUMANA

Exemplo:

Pacote **UtilitarioInterfaceGrafica**



Pacote **IU_AtendimentoCliente**



Denise F. Togneri



COMPONENTE DE GERÊNCIA DE TAREFAS



PROJETO ARQUITETURAL DO COMPONENTE DE GERÊNCIA DE TAREFAS

56

- Compreende a definição de tarefas e a comunicação e coordenação entre elas.
- **OBJETIVO BÁSICO**
 - definir e classificar as tarefas.
- **MOTIVAÇÃO**
 - Algumas funcionalidades não são facilmente distribuídas nas classes e objetos do CDP, principalmente aquelas que operam sobre vários objetos.
- **SOLUÇÃO POSSÍVEL MAS NÃO VIÁVEL**
 - pulverizar esse comportamento ao longo de vários objetos do CDP ou do CIH
- **PROBLEMA DESTA SOLUÇÃO**
 - **não é uma boa solução** segundo uma perspectiva de **alterabilidade**. Uma alteração em tal funcionalidade poderia afetar diversos objetos, e assim ser difícil de ser incorporada.

PROJETO ARQUITETURAL OO ***COMPONENTE DE GERÊNCIA DE TAREFAS***

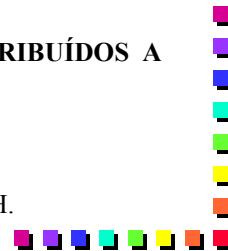
57

■ **ABORDAGEM POTENCIAL**

- modelar as classes do CGT como gerenciadores ou coordenadores de tarefas, responsáveis pela realização de tarefas sobre um determinado conjunto de objetos.
- Esses gerenciadores agem como aglutinadores, unindo outros objetos para dar forma a um caso de uso. Conseqüentemente, gerenciadores de tarefa são normalmente encontrados diretamente a partir dos casos de uso.

■ **TIPOS DE FUNCIONALIDADE TÍPICAMENTE ATRIBUÍDOS A GERENCIADORES DE TAREFA**

- comportamento relacionado a transações;
- seqüências de controle específicas a um caso de uso;
- funcionalidades que separam objetos do CDP e do CIH.

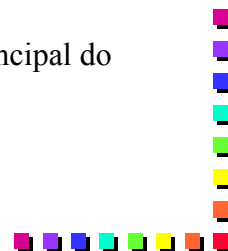


Denise F. Togneri

PROJETO ARQUITETURAL OO ***COMPONENTE DE GERÊNCIA DE TAREFAS***

58

- Em um projeto OO, uma **aplicação** qualquer é iniciada com uma chamada ao Componente de Gerência de Tarefa (CGT).
- O CGT mostra toda a seqüência de ações que deve ser executada para o processamento de algum caso de uso; portanto, diz-se que o CGT dá seqüência ao caso de uso.
- Normalmente, cada opção na barra do menu principal do sistema é um caso de uso.

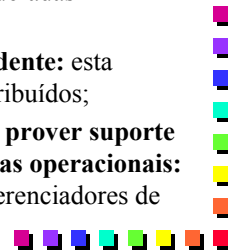


Denise F. Togneri

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

59

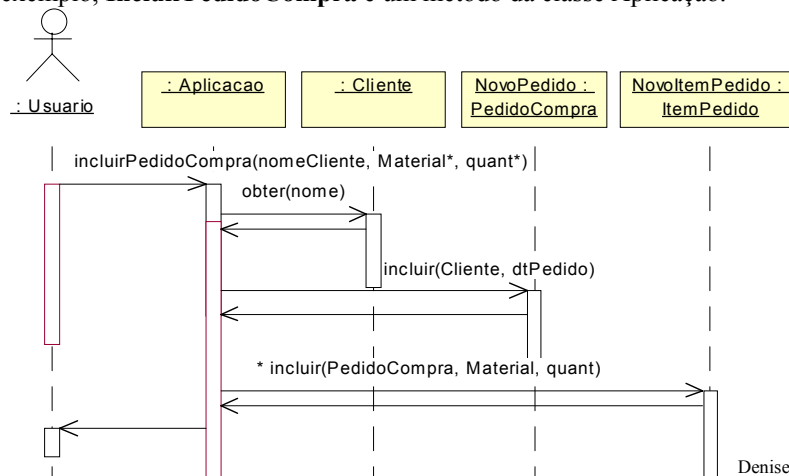
- É importante observar que os aspectos dinâmicos de um modelo de análise mostram a existência, ou não, de concorrência entre objetos (ou subsistemas).
- Se objetos (ou subsistemas) não têm de estar ativos em um mesmo momento, então não há necessidade de processamento concorrente e, portanto, o processamento do sistema pode ser atribuído a um único processador.
- Caso contrário, duas opções de alocação devem ser consideradas (Pressman, 2002):
 - **alocar cada subsistema a um processador independente:** esta abordagem requer sistemas multi-processados ou distribuídos;
 - **alocar os subsistemas para o mesmo processador e prover suporte a concorrência através de características de sistemas operacionais:** neste caso, serão necessárias novas classes do tipo “gerenciadores de tarefas”, responsáveis por este suporte.



Denise F. Togneri

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS** **PASSOS A SEREM SEGUIDOS**

- Na fase de Análise OO, cada diagrama de sequência é iniciado por um ator executando um método de uma classe genérica chamada Aplicação. No exemplo, **IncluirPedidoCompra** é um método da classe Aplicação.



Denise F. Togneri

PROJETO ARQUITETURAL OO

COMPONENTE DE GERÊNCIA DE TAREFAS

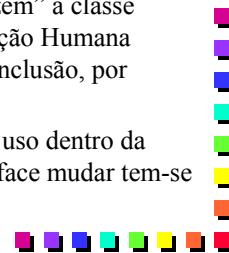
PASSOS A SEREM SEGUIDOS

61

- A classe Aplicação da Análise OO fará parte do CGT.
- Estudar, a seguir, se a classe Aplicação continuará sendo única ou se serão criadas mais de uma classe.
- Fazer o estudo referente a Modelo de Tarefas.

■ **ERRO COMUM**

- Muitos projetistas OO não criam o CGT. Eles “traduzem” a classe Aplicação da Análise OO pelo Componente de Interação Humana (CIH), ou seja, as validações/críticas e comandos de inclusão, por exemplo, são projetadas na própria window.
- Não se deve colocar o comportamento de um caso de uso dentro da interface; isto causa muitos problemas, pois se a interface mudar tem-se que reprogramar estas coisas.



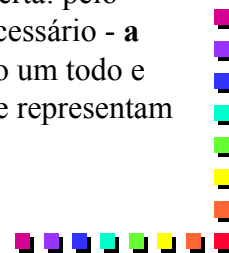
Denise F. Togneri

PROJETO ARQUITETURAL OO

COMPONENTE DE GERÊNCIA DE TAREFAS

62

- Para modelar o CGT, pelo menos três possíveis soluções existem:
 - atribuir um gerenciador de tarefa para cada caso de uso;
 - definir uma única classe de aplicação para todo o sistema;
 - uma solução intermediária entre as duas acima.
- Independente da solução adotada, uma coisa é certa: pelo menos um gerenciador de tarefa será sempre necessário - **a classe Aplicação**, representando o sistema como um todo e dando forma à aplicação. Os objetos desta classe representam as várias sessões (execuções) do sistema.



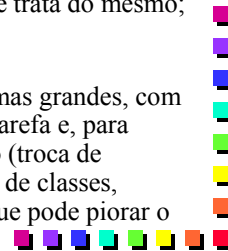
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

63

1ª SOLUÇÃO: ATRIBUIR UM GERENCIADOR DE TAREFA PARA CADA CASO DE USO

- Em um esboço preliminar, pode-se atribuir um gerenciador de tarefa para cada caso de uso (ex. `ControlarPedidoCompra`), sendo que seus casos de uso de nível mínimo (`IncluirPedidoCompra`, `AlterarPedidoCompra`, `CancelarPedidoCompra`, `ConsultarPedidoCompra`, etc) dão origem a operações da classe que representa o caso de uso).
- **Vantagem → Facilidade de alteração:** uma vez que, detectado um problema em um caso de uso, é fácil identificar a classe que trata do mesmo; esta solução é bem modular.
- **Desvantagem → Diminuição do desempenho:** para sistemas grandes, com muitos casos de uso, haverá muitas classes de gerência de tarefa e, para realizar uma tarefa, pode ser necessária muita comunicação (troca de mensagens) entre essas classes. Quanto maior a quantidade de classes, ocorrerá uma quantidade maior de troca de mensagens, o que pode piorar o desempenho.

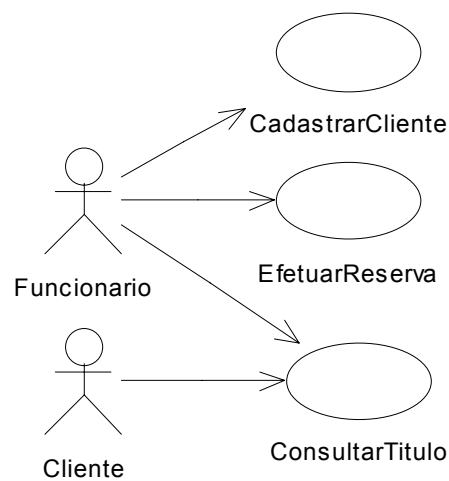


Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

1ª SOLUÇÃO: ATRIBUIR UM GERENCIADOR DE TAREFA PARA CADA CASO DE USO

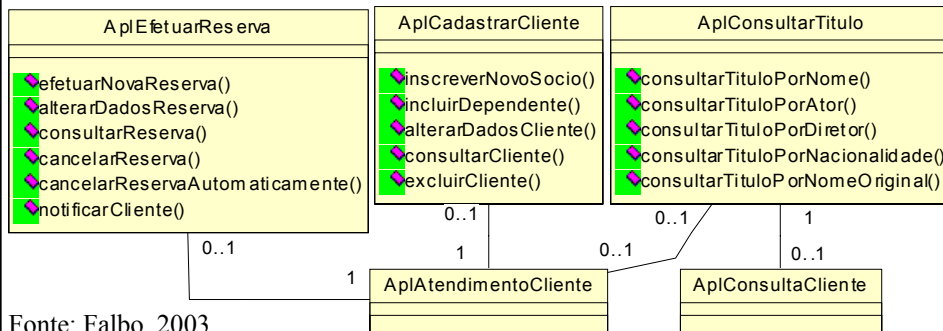
Diagrama de Caso de Uso: AtenderCliente - Parcial



Fonte: Falbo, 2003.

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

1ª SOLUÇÃO: ATRIBUIR UM GERENCIADOR DE TAREFA PARA CADA CASO DE USO



Fonte: Falbo, 2003.

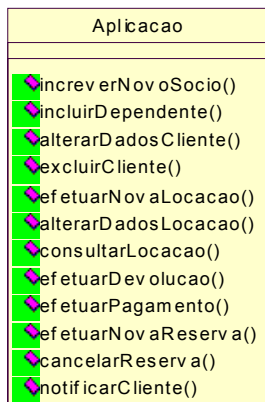
As classes **AplEfetuarReserva**, **AplCadastrarCliente** e **AplConsultarTitulo** lidam com seus respectivos casos de uso. A **aplicação de consulta a título** pode existir isoladamente da aplicação de atendimento a cliente e por isso foi isolada por uma **cardinalidade 0..1** e foi associada à classe **AplConsultaCliente**. As classes **AplAtendimentoCliente** e **AplConsultaCliente** dão forma a duas aplicações executáveis, a primeira utilizada pelo ator **Funcionario** e a segunda utilizada pelo ator **Cliente**.

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

66

2ª SOLUÇÃO: DEFINIR UMA ÚNICA CLASSE DE APLICAÇÃO PARA TODO O SISTEMA

- Neste caso, deve ser criado um método (uma operação) para cada caso de uso de nível mínimo. Cada um destes métodos vai conter toda a sequência de execução do caso de uso. **Exemplo:**



Fica evidente que, exceto para sistemas muito pequenos, a classe de aplicação será extremamente complexa e, portanto, esta abordagem não seria prática.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

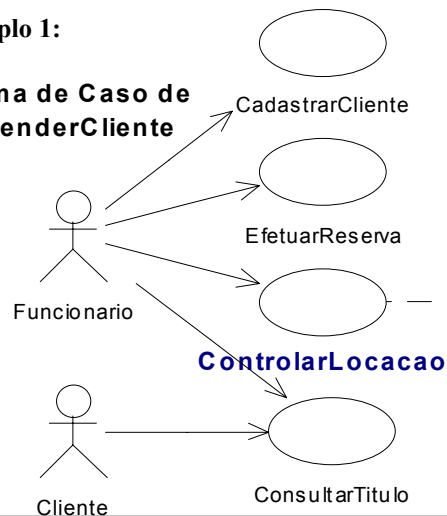
- Conduz a melhores resultados.
- Nessa abordagem, casos de uso complexos são designados a classes de gerência de tarefas específicas.
- Casos de uso mais simples e de alguma forma relacionados são tratados por uma mesma classe de aplicação.

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

Exemplo 1:

**Diagrama de Caso de
Uso: AtenderCliente**



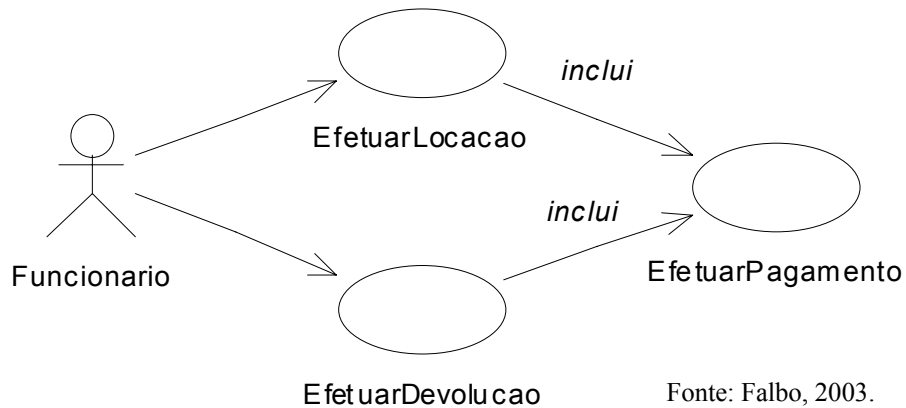
Engloba os casos de uso diretamente envolvidos com a Locação e é, portanto, decomposto nos casos de uso: EfetuarLocacao, EfetuarDevolucao e Efetuar Pagamento

Fonte: Falbo, 2003.

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

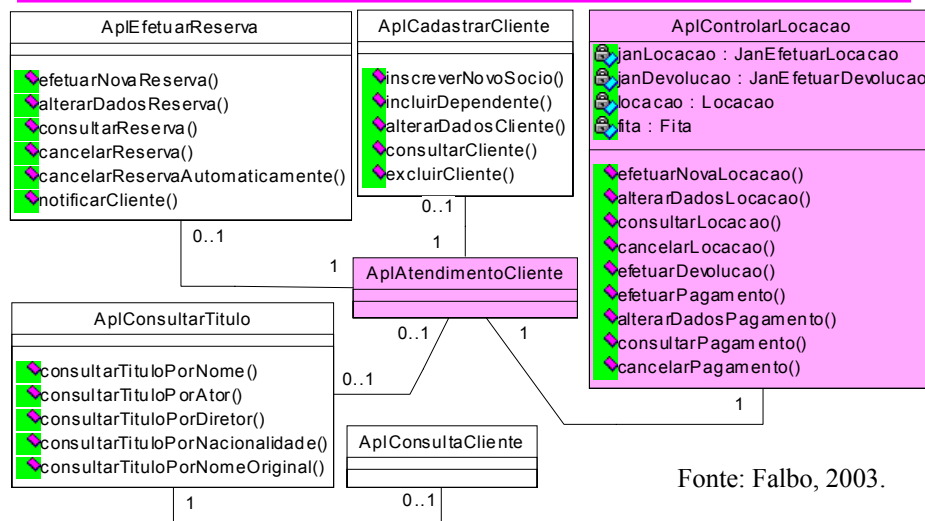
3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

Diagrama de Caso de Uso: ControlarLocacao



PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE TAREFAS

3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

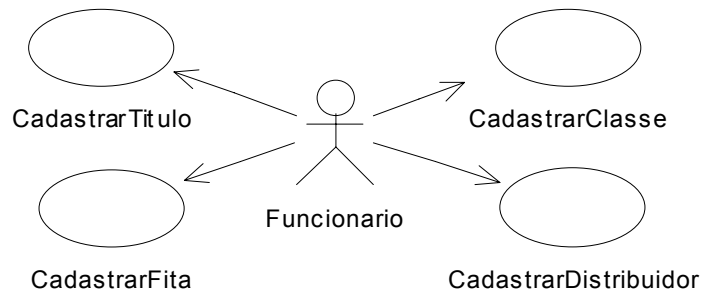


PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

Exemplo 2:

Diagrama de Caso de Uso: ControlarAcervo



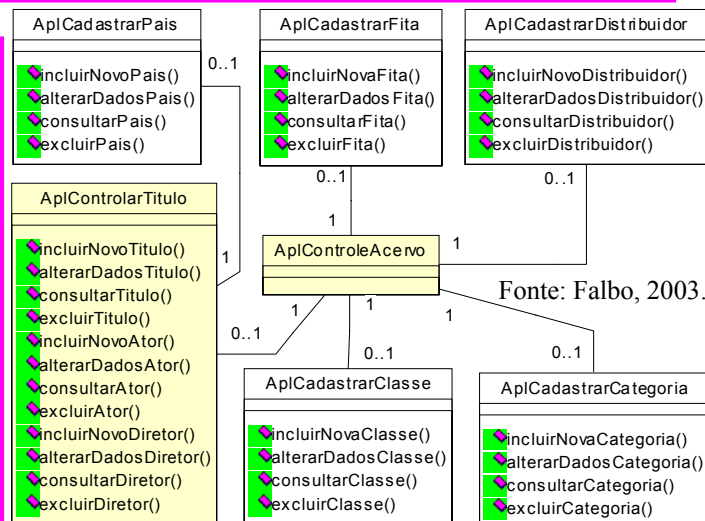
Fonte: Falbo, 2003.

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

3ª SOLUÇÃO: UMA SOLUÇÃO INTERMEDIÁRIA ENTRE AS DUAS ANTERIORES

Exemplo 2:

A classe **AplControlarTitulo** trata do caso de uso **CadastrarTitulo** e dos casos de uso fortemente relacionados a ele: **CadastrarAtor** e **CadastrarDiretor**.
A classe **AplControleAcervo** dá forma à aplicação Controle de Acervo como um executável.
Fonte: Falbo, 2003.

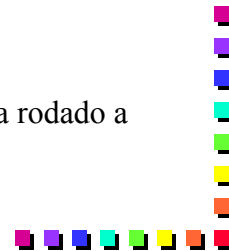


Fonte: Falbo, 2003.

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

73

- É necessário levar em conta, ainda, quantos “executáveis” devem ser gerados para o sistema.
- Se mais do que um for necessário, cada “executável” terá de dar origem a uma classe de aplicação.
- Outros fatores que afetam esta decisão são
 - aspectos de distribuição geográfica e
 - se o sistema será um aplicativo ou um sistema rodado a partir de um navegador.



Denise F. Togneri

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

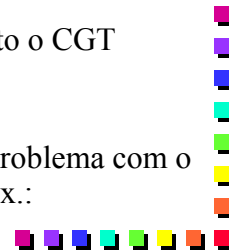
- O levantamento das tarefas do sistema também é uma atividade que pode ser iniciada na fase de análise. Utilizar Modelo de Tarefas.
- **Conceito de Tarefa:** Conjunto de atividades essenciais e tecnológicas agrupadas em uma unidade de execução. Ex: um arquivo executável (.EXE) em um ambiente DOS seria uma tarefa. Em um ambiente multitarefa, uma tarefa pode ser disparada de várias estações/terminais (Xavier e Portilho, 1995).

QUADRO DE ALOCAÇÃO EM TAREFA (QAT)			
Código:	Nome da Tarefa:	Processador:	
Nº do Processo ou Evento	Atividade(s)	Frequência	Estimulador(es)
(nº processo no DFD)	(nome do processo no DFD)	(frequência média de utilização da atividade em relação ao tempo) Ex: 3/dia, 2/mês, ...	

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

75

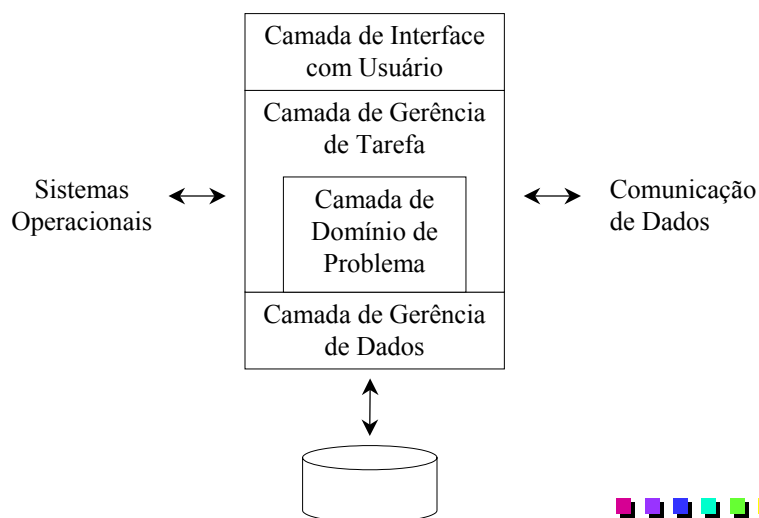
- Os projetos dos Componentes de Interação Humana e de Gerência de Tarefa estão bastante relacionados, já que, muitas vezes, são as tarefas que determinam a necessidade de elementos de interface com o usuário para sua execução.
- Assim, os projetos dos CIH e CGT devem ser realizados em conjunto.
- De fato, tanto o CIH (através do protótipo) quanto o CGT podem ser iniciados na fase de análise.
- Ainda existem outras interfaces do domínio do problema com o Componente do Domínio do Problema (CDP). Ex.: comunicação de dados, S.O.



Denise F. Togneri

PROJETO ARQUITETURAL OO **COMPONENTE DE GERÊNCIA DE TAREFAS**

76

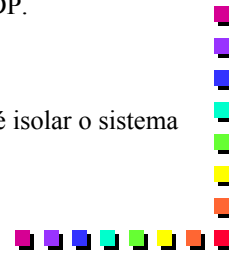


Denise F. Togneri

PROJETO ARQUITETURAL OO ***COMPONENTE DE GERÊNCIA DE TAREFAS***

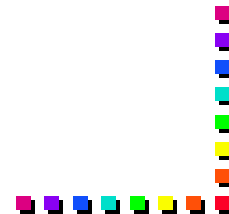
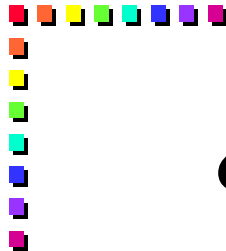
77

- Toda vez que for necessário estabelecer mecanismos de comunicação (com outro software, linguagem de programação, com o sistema operacional) deve-se fazer isto via CGT.
- O CGT deve envolver o CDP para não poluir-lo com coisas técnicas.
- Se, por exemplo, mudar o sistema operacional, basta trocar a classe do CGT que faz a comunicação com o S.O . E não do CDP.
- **ASPECTO IMPORTANTE**
 - A idéia de se trabalhar com software em camadas é isolar o sistema dos impactos tecnológicos.



Denise F. Togneri

COMPONENTE DE ***GERÊNCIA DE DADOS***



PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS

79

- Provê a infra-estrutura básica para o armazenamento e a recuperação de objetos no sistema. Sua finalidade é isolar os impactos da tecnologia de gerenciamento de dados sobre a arquitetura do software (Coad e Yourdon, 1993).

■ QUESTÃO PRIMORDIAL

- Como tornar persistentes os objetos do sistema?

■ SUPORTE À PERSISTÊNCIA DE OBJETOS

- uso de arquivos
- persistência de objetos fornecida pela própria linguagem de programação
- bancos de dados de objetos
- bancos de dados relacionais



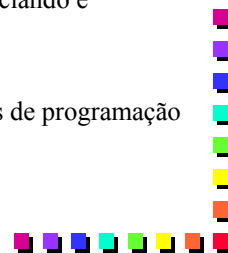
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

80

■ USO DE ARQUIVOS

- Neste caso, é necessário estabelecer uma estratégia para escrita e leitura de uma série de objetos em um arquivo simples.
- Um layout simples para um arquivo pode ser pensado em termos de um objeto por linha, com os atributos do objeto iniciando e terminando em posições específicas.
- As facilidades oferecidas pelas próprias linguagens de programação para manipular arquivos devem ser utilizadas.



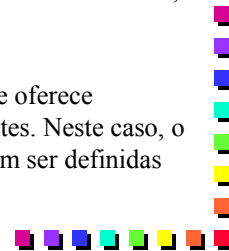
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

81

■ PERSISTÊNCIA DE OBJETOS FORNECIDA PELA PRÓPRIA LINGUAGEM DE PROGRAMAÇÃO

- É o caso de Smalltalk (arquivo IMAGE) e Eiffel (classe STORABLE).
- Em Smalltalk, todos os objetos são persistentes e, ao encerrar uma sessão, o estado de todo ambiente é salvo em um arquivo. Neste caso, não há projeto do CGD.
- Eiffel, por sua vez, oferece a classe STORABLE, que oferece mecanismos para salvar e recuperar objetos persistentes. Neste caso, o projeto do CGD consiste em definir que classes devem ser definidas como sub-classes de STORABLE.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

82

■ BANCO DE DADOS DE OBJETOS = SOLUÇÃO IDEAL

- Em um ambiente orientado a objetos, a solução ideal seria usar um Sistema Gerenciador de Banco de Dados Orientado a Objetos (SGBDOO), onde cada uma das classes persistentes na arquitetura de software corresponderia a exatamente uma base de dados gerenciada pelo SGBDOO.
- Mesmo neste caso o CGD continuaria a existir, para que futuras alterações (ex. troca de SGBD) não afetassem o CDP.



Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

83

ALGUNS EXEMPLOS DE SGBDs ORIENTADOS A OBJETOS

- O2 (O2 Technology)
- ObjectStore
- Objectivity
- ONTOS (Ontologic)
- Versant
- Jasmine (Computer Associates)
- POET (Poet Software)
- GemStone
- OpenOBD (da HP)



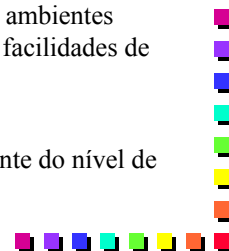
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

84

BANCOS DE DADOS RELACIONAIS

- Ainda que haja diferenças entre a abordagem relacional e o paradigma de objetos, os bancos de dados relacionais têm sido utilizados pela maioria dos desenvolvedores OO para armazenar objetos (Ambler, 1998).
- Alguns ambientes de programação, tais como certos ambientes C++ e o ambiente Delphi (Object Pascal), oferecem facilidades de interface com alguns bancos de dados relacionais.
- Neste caso, o projeto do CGD é fortemente dependente do nível de suporte oferecido.



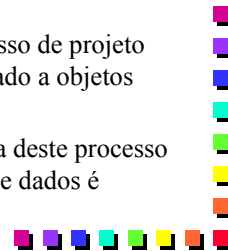
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS SUPORTE À PERSISTÊNCIA DE OBJETOS

85

■ BANCOS DE DADOS RELACIONAIS

- Caso o ambiente encapsule totalmente o banco de dados relacional, o projeto pode ser muito semelhante ao projeto usando um banco de dados OO;
- Caso contrário, o projetista deve efetuar um projeto de bases de dados relacionais, definindo suas tabelas, para só então poder utilizá-las no projeto OO do CGD.
- Utilizando SGBDs relacionais, geralmente, o processo de projeto começa pela tradução das classes no modelo orientado a objetos para a **terceira forma normal** padrão.
- Para cada tabela em terceira forma normal, derivada deste processo de “normalização de objetos”, uma tabela na base de dados é definida.



Denise F. Togneri

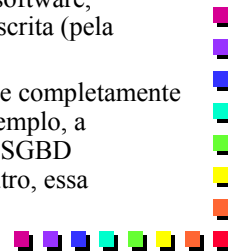
PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

86

A despeito da opção de persistência adotada, outra questão deve ser considerada: Que classes devem suportar a persistência dos objetos?

1ª ALTERNATIVA – NÃO RECOMENDADA

- Tornar cada classe, ao longo de toda a arquitetura do software, responsável por suas próprias atividades de leitura e escrita (pela própria persistência).
- **Desvantagem:** nesta abordagem, a arquitetura torna-se completamente dependente da tecnologia de persistência e, se, por exemplo, a organização migra de um sistema de arquivo para um SGBD relacional, ou mesmo de um SGBD relacional para outro, essa migração impactaria todas as classes ao longo do sistema.



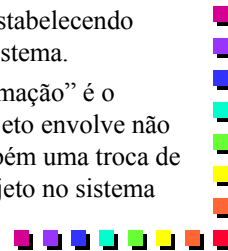
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

87

2ª ALTERNATIVA: MAIS ELEGANTE

- Fazer com que apenas uma parte da arquitetura de software fique ciente da tecnologia de persistência adotada, no caso o CGD.
- Essa parte, o CGD, serve como uma camada intermediária para as classes de objetos persistentes, tipicamente do CDP, estabelecendo um protocolo para a persistência dos objetos.
- Via conexões de mensagem, o CGD lê e escreve dados, estabelecendo uma comunicação entre a base de dados e os objetos do sistema.
- O preço a ser pago por este tipo de “ocultamento de informação” é o **desempenho**: cada requisição para ler ou escrever um objeto envolve não apenas os comandos físicos de leitura/gravação, mas também uma troca de dados (via parâmetros de mensagem) entre o CGD e o objeto no sistema (Yourdon, 1994).



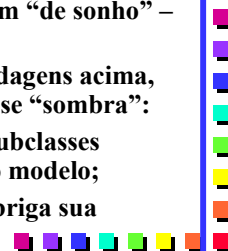
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

88

2ª ALTERNATIVA: MAIS ELEGANTE

- Nesta abordagem, as operações de armazenamento e recuperação de objetos não são colocadas nas classes do CDP, mas sim em uma ou mais classes “salvadoras de objetos” dentro do CGD.
- Existem pelo menos 3 abordagens nesta alternativa:
 - Prover uma classe “sombra” no CGD para cada classe persistente do modelo;
 - Prover classes genéricas de persistência: abordagem “de sonho” – mais elegante e complexa;
 - Prover alguma solução híbrida entre as duas abordagens acima, como por exemplo criando no CGD uma superclasse “sombra”:
 - cujos atributos e métodos são herdados pelas subclasses “sombra” relativas a cada classe persistente do modelo;
 - que podem possuir métodos abstratos, o que obriga sua implementação pelas subclasses “sombra”.



Denise F. Togneri

PROJETO ARQUITETURAL OO

COMPONENTE DE GERÊNCIA DE DADOS

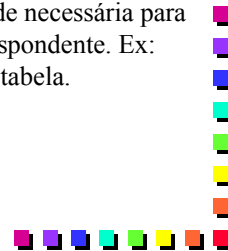
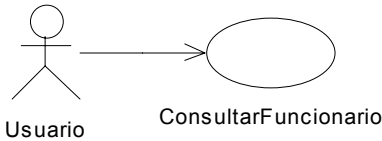
CLASSES PARA SUPORTE À PERSISTÊNCIA

89

2ª ALTERNATIVA: MAIS ELEGANTE

2.1 - Prover uma classe “sombra” no CGD para cada classe persistente do modelo

- Criação da classe “sombra” no CGD
- A abordagem mais direta para a camada de persistência consiste em prover uma classe “sombra” no CGD para cada classe persistente nos demais componentes da arquitetura.
- Tal classe salvadora de objetos encapsula a funcionalidade necessária para se implementar a persistência dos objetos da classe correspondente. Ex: contém um comando SQL de inclusão de linhas em uma tabela.
- Exemplo: Diagrama de Caso de Uso

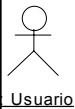


Denise F. Togneri

PROJETO ARQUITETURAL OO

COMPONENTE DE GERÊNCIA DE DADOS

CLASSES PARA SUPORTE À PERSISTÊNCIA



Usuario

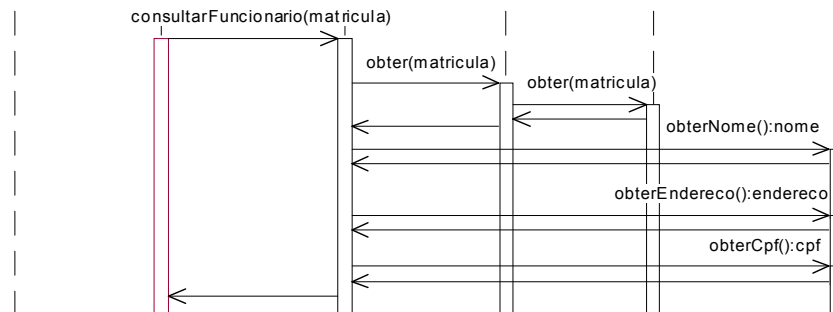
JanConsultarFunc :
JanConsultarFunc

AplControlarFunc :
AplControlarFunc

: Funcionario

: FuncionarioP

FuncionarioConsultado
: Funcionario



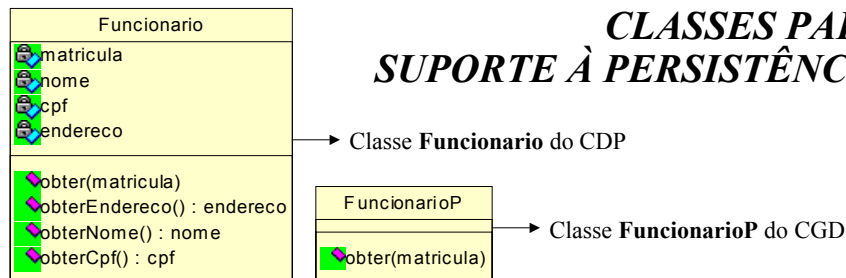
O método **obter(matricula)** da classe **FuncionarioP** possui o comando SQL select que consulta a tabela FUNC para obter a linha/objeto desejado, como no exemplo abaixo:

```

public void obter(int matricula) throws ClassNotFoundException, SQLException {
    if (validateConn()) {PreparedStatement stt = _conn.prepareStatement(
        "select nm_func, ds_end, no_cpf from FUNC where no_func_matr = ?");
        stt.setInt(1,this.matricula);
        ResultSet rs = stt.executeQuery();
    }
  }
  
```

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

91



- Os métodos da **classe Funcionario do CDP** possuem somente validações, críticas, verificações de integridade, regras de negócio,
- Os métodos da **classe FuncionarioP do CGD** não possuem validações; apenas manipulam objetos no SGBD;
- A **classe Funcionario** envia mensagem para a **classe FuncionarioP** solicitando a I/A/E/C de objetos no SGBD;
- **NUNCA fazer herança entre as classes de objeto e objeto persistente respectiva;**
- Entre as classes de objetos persistentes pode-se usar hierarquia (estrutura generalização/especialização);
- Neste caso, o CDP está “livre” das operações de Banco de Dados.

Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

92

2ª ALTERNATIVA: MAIS ELEGANTE

2.2 - Prover classes genéricas de persistência: abordagem “de sonho”

- Uma abordagem mais elegante e complexa consiste em prover classes genéricas que estabelecem protocolos para comunicação com os meios de armazenamento secundários e utilizá-las para a persistência dos objetos das classes correspondentes.
- Essa abordagem pode ser alcançada com o desenvolvimento ou uso de pacotes prontos que fornecem esse suporte.
 - Ex: HIBERNATE (www.hibernate.org)

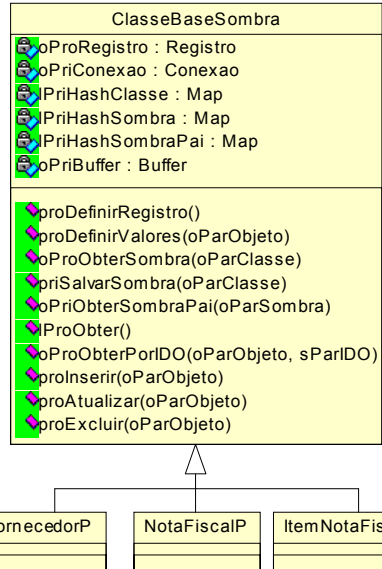
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS CLASSES PARA SUPORTE À PERSISTÊNCIA

2ª ALTERNATIVA: MAIS ELEGANTE

2.3 - Prover alguma solução híbrida entre as duas abordagens

- Uma solução intermediária é criar no CGD uma superclasse “sombra” cujos atributos e métodos são herdados pelas classes “sombra” relativas a cada classe persistente do modelo;



PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS

PERSISTÊNCIA EM B.D. RELACIONAL



PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

95

Uma vez que os bancos de dados relacionais são os dispositivos de armazenamento mais confiáveis e utilizados atualmente (Magela, 1998), a seguir, será detalhado o projeto do CGD, pressupondo o uso desse dispositivo de armazenamento.



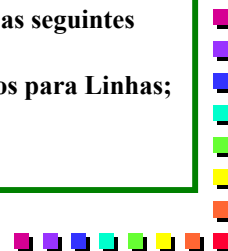
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

96

- Claramente, há uma diferença semântica significativa entre o modelo de classes de um projeto OO e o modelo relacional.
- Para que a persistência de objetos seja feita em um banco de dados relacional, é necessário proceder um mapeamento entre esses dois mundos.
- Este mapeamento só deve ser visível na camada de persistência, isto é, no CGD, isolando o CDP do impacto da tecnologia de bancos de dados.

- No mapeamento dos mundos de objetos e relacional, as seguintes questões devem ser abordadas:
 - Mapeamento de Classes para Tabelas e de Objetos para Linhas;
 - Mapeamento de Herança;
 - Mapeamento de Relacionamentos entre Objetos.



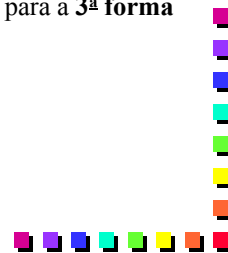
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

97

MAPEANDO CLASSES EM TABELAS E OBJETOS EM LINHAS

- Quando não há herança, cada classe deve ser mapeada em uma tabela e cada instância da classe (objeto) em uma linha desta tabela.
- O modelo de classes deve ser normalizado previamente para a **3ª forma normal**, eliminando-se atributos multivalorados.

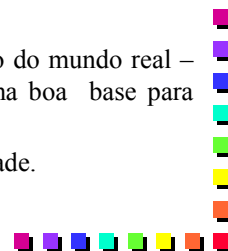


Denise F. Togneri

PROJ. ARQ. OO - CGD - PERSISTÊNCIA EM SGBDR NORMALIZAÇÃO - 1ª, 2ª E 3ª FORMAS NORMAIS

NORMALIZAÇÃO

- Para derivar um Modelo Relacional, além de derivar os relacionamentos e as estruturas de agregação e generalização-especialização, as estruturas de dados também devem ser normalizadas.
- Os **objetivos gerais** do processo de normalização são (Date, 2000):
 - Eliminar certas espécies de redundâncias;
 - Evitar certas anomalias de atualização;
 - Produzir um projeto que seja uma “boa” representação do mundo real – isto é, que seja intuitivamente fácil de entender e uma boa base para crescimento futuro.
 - Simplificar a imposição de certas restrições de integridade.



Denise F. Togneri

PROJ. ARQ. OO - CGD - PERSISTÊNCIA EM SGBDR

NORMALIZAÇÃO - 1ª, 2ª E 3ª FORMAS NORMAIS

PRIMEIRA FORMA NORMAL (1FN)

Um modelo está na primeira forma normal se (Cougo, 1997):

1. **Está integrado por tabelas:** modelo relacional definido em forma de tabelas contendo linhas e colunas unicamente.
2. **As linhas das tabelas são unívocas:** significa definir as chaves primárias da tabela de forma a proibir a existência de 2 linhas iguais.
3. **As linhas não contém itens repetitivos:** ou seja, evitar uso de estruturas de dado do tipo vetor, uma vez que não é padrão SQL ANSI.
4. **Os atributos são atômicos:** ou seja, não se pode usar itens de grupo.
5. **Os atributos não contém valores nulos:** é uma definição formal no modelo relacional. No entanto, do ponto de vista prático atual, não há restrição quanto à existência de valores nulos em colunas de uma tabela para que ela esteja na primeira forma normal (1FN).

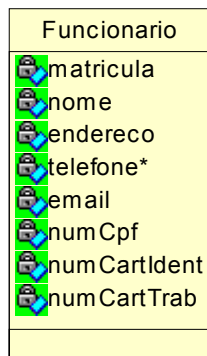
Processo para Obtenção da Primeira Forma Normal

1. Definir as chaves candidatas e escolher a chave primária da tabela.
2. Transformar atributos compostos em atômicos.
3. Em cada tabela, eliminar grupos repetitivos gerando novas linhas, uma para cada ocorrência de item repetitivo, mantendo os valores dos demais itens. Denise F. Togneri

PROJ. ARQ. OO - CGD - PERSISTÊNCIA EM SGBDR

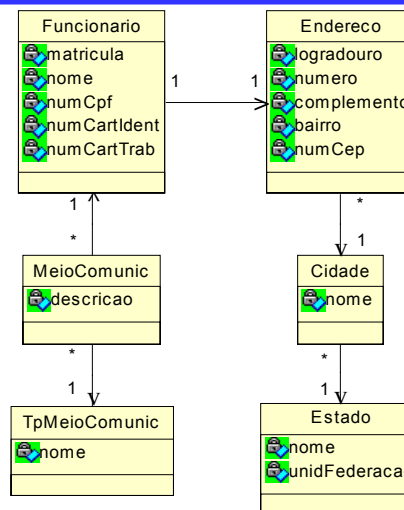
EXEMPLO DE NORMALIZAÇÃO DA PRIMEIRA FORMA NORMAL (1FN)

Diagrama de Classes
de Análise OO



os atributos **endereco** (item de grupo) e **telefone** (item repetitivo) não estão na 1FN.

Diagrama de Classes de Projeto OO
Componente de Domínio do Problema



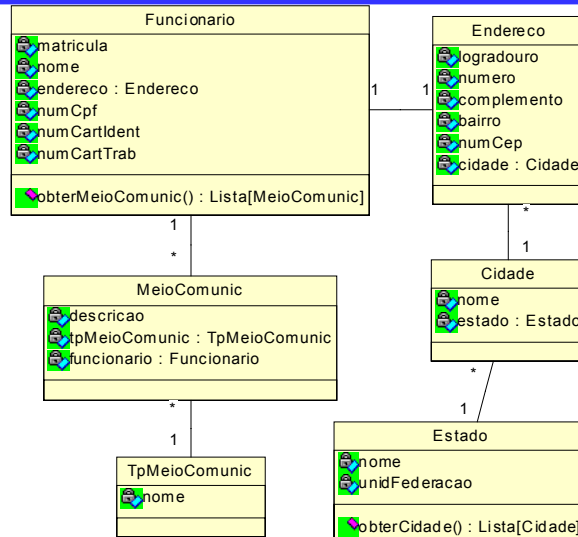
Denise F. Togneri

PROJ. ARQ. OO - CGD - PERSISTÊNCIA EM SGBDR

EXEMPLO DE NORMALIZAÇÃO DA PRIMEIRA FORMA NORMAL (1FN)

Diagrama de Classes de Projeto OO - Componente de Domínio do Problema

Significado das decisões de navegabilidade



Denise F. Togneri

PROJ. ARQ. OO - CGD - PERSISTÊNCIA EM SGBDR NORMALIZAÇÃO - 1ª, 2ª E 3ª FORMAS NORMAIS

SEGUNDA FORMA NORMAL (2FN)

- Uma tabela está na segunda forma normal se está na 1FN e cada uma das colunas não pertencentes à chave primária não for dependente parcialmente dessa chave (Cougo, 1997).
- Ao analisar uma tabela, deveremos excluir de nossa análise as colunas formadoras da chave primária dessa tabela. Além disto, a dependência parcial de uma chave só será possível se a chave primária for definida por mais de uma coluna. Caso tenhamos só uma coluna definindo a chave primária, essa tabela já estará, automaticamente, na segunda forma normal.
- O que significa dependência parcial da chave? Dizemos que uma coluna depende parcialmente da chave se, para que seu valor seja determinado não necessitarmos conhecer a chave como um todo, mas sim somente um ou alguns de seus valores.

Processo para Obtenção da Segunda Forma Normal

1. Identificar as colunas que não participam da chave primária da tabela
2. Para cada uma das colunas identificadas, analisar se seu valor é determinado por parte, ou pela totalidade da chave.
3. Para as colunas dependentes parcialmente da chave, (a) criar novas tabelas onde a chave primária será(ão) a(s) coluna(s) da chave primária original que determinou(aram) o valor da coluna analisada e (b) excluir da tabela original as colunas dependentes parcialmente da chave.

Denise F. Togneri

PROJ. ARO. OO - CGD - PERSISTÊNCIA EM SGBDR NORMALIZAÇÃO - 1ª, 2ª E 3ª FORMAS NORMAIS

TERCEIRA FORMA NORMAL (3FN)

- **Uma tabela está na terceira forma normal se está na 2FN, e se nenhuma coluna não pertencente à chave fica determinada transitivamente por esta.**
- Este enunciado deixa claro que nessa atividade só deverão ser analisadas as colunas não pertencentes à chave primária, independente dela ser composta ou não.
- A dependência transitiva de uma chave só será possível se a tabela tiver ao menos duas colunas não pertencentes à chave. Caso tenhamos somente uma coluna externa à chave, essa tabela já estará automaticamente na terceira forma normal.
- **O que significa uma dependência transitiva da chave?** Dizemos que uma coluna depende transitivamente da chave se seu valor é determinado pelo conteúdo de uma coluna não chave que, por sua vez, também já é determinada pela chave primária da tabela.

Processo para Obtenção da Terceira Forma Normal

1. Identificar as colunas que não participam da chave primária da tabela.
2. Para cada uma das colunas identificadas, analisar se seu valor é determinado por alguma outra coluna não pertencente à chave
3. Para as colunas dependentes transitivamente de outra coluna, (a) criar novas tabelas onde a chave primária será(ão) a(s) coluna(s) que determinou(aram) o valor da coluna analisada e (b) excluir da tabela de origem as colunas dependentes transitivamente, mantendo, porém, a coluna determinante da transitividade na tabela.

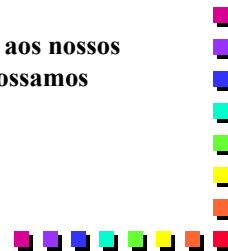
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

104

MAPEANDO CLASSES EM TABELAS E OBJETOS EM LINHAS

- Uma questão importante:
 - No modelo relacional, toda tabela tem que ter uma chave primária;
 - Objetos, por sua vez, têm identidade própria, independentemente dos valores de seus atributos.
 - Assim, **que identificador único devemos designar aos nossos objetos no banco de dados relacional, para que possamos distingui-los?**



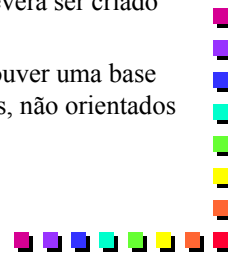
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

105

MAPEANDO CLASSES EM TABELAS E OBJETOS EM LINHAS

- **1ª Solução – não recomendável para abordagem OO**
 - Consiste em observar se há um atributo na classe com esta propriedade de identificação única e utilizá-lo, então, como chave primária.
 - Caso não haja um atributo com tal característica, deverá ser criado um.
 - Esta abordagem deve ser utilizada sempre que já houver uma base de dados legada, sendo utilizada por outros sistemas, não orientados a objetos.



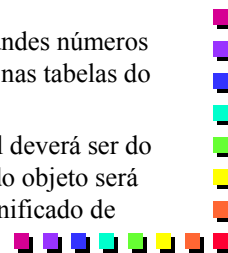
Denise F. Togneri

PROJETO ARQUITETURAL OO COMPONENTE DE GERÊNCIA DE DADOS PERSISTÊNCIA EM B.D. RELACIONAL

106

MAPEANDO CLASSES EM TABELAS E OBJETOS EM LINHAS

- **2ª Solução**
 - Consiste em dar a cada objeto um atributo chamado de identificador de objeto (IDO).
 - É uma maneira mais eficaz, sobretudo para permitir a construção de componentes mais genéricos de persistência.
 - Os IDOs são tipicamente implementados como grandes números inteiros, que são utilizados como chaves primárias nas tabelas do banco de dados relacional.
 - Essa coluna na tabela do banco de dados relacional deverá ser do tipo auto-incremento, garantindo que a unicidade do objeto será mapeada na tabela, e não deve possuir nenhum significado de negócio (Ambler, 1998) (Magela, 1998).



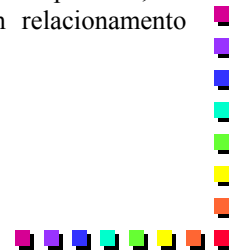
Denise F. Togneri

107

PROJ. ARQUIT. OO
PERSISTÊNCIA EM B.D. RELACIONAL
TRADUÇÃO DE RELACIONAMENTOS

MAPEAMENTO DE RELACIONAMENTOS

- É necessário transpor chaves entre tabelas para mapear relacionamentos.
- As regras válidas para o modelo relacional têm de ser aplicadas, tal como criar uma tabela adicional para mapear um relacionamento muitos-para-muitos.



Denise F. Togneri

108

DIAGRAMA RELACIONAL
NOTAÇÃO DAS ASSOCIAÇÕES

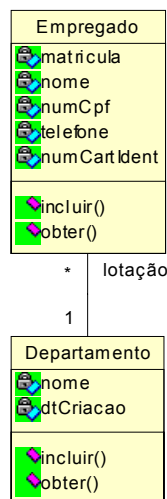
No Diagrama Relacional, as ligações que derivam das associações, são representadas por linhas contínuas, associadas aos símbolos abaixo.

Cardinalidade no Diagrama de Classes	Ligação no D. Relacional
0..1	
1	
*	
1..*	

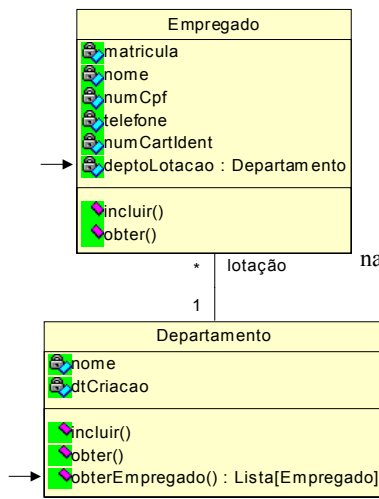
Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 109
TRADUÇÃO DE RELACIONAMENTO 1:N

**Diagrama de Classes
de Análise OO**

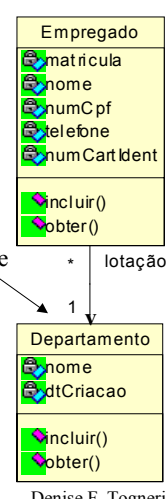


**Diagrama de Classes de Projeto OO
Componente de Domínio do Problema**



OU

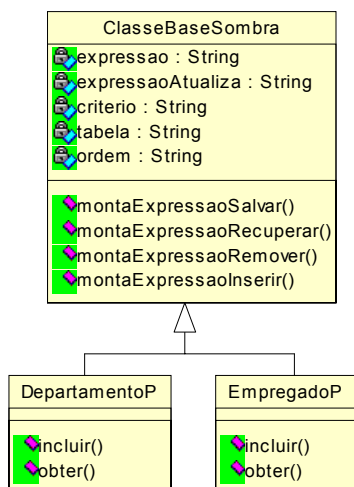
indicação
navegabilidade



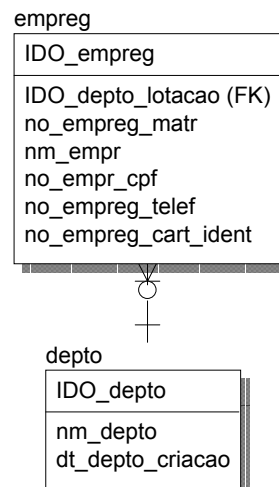
Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 110
TRADUÇÃO DE RELACIONAMENTO 1:N

**Diagrama de Classes - Projeto OO
Componente de Gerência de Dados**



**Diagrama Relacional
SGBDR**



Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 111

TRADUÇÃO DE RELACIONAMENTO N:N

Diagrama de Classes
de Análise OO

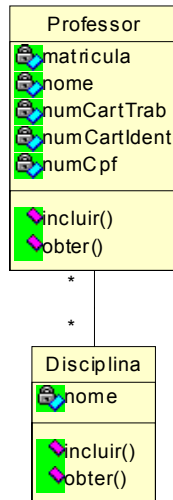
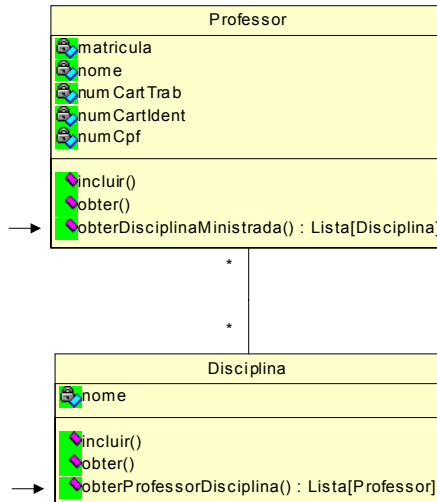


Diagrama de Classes de Projeto OO
Componente de Domínio do Problema



A solução adotada foi implementar uma **navegação indireta** (via métodos).

Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 112

TRADUÇÃO DE RELACIONAMENTO N:N

Diagrama de Classes - Projeto OO
Componente de Gerência de Dados

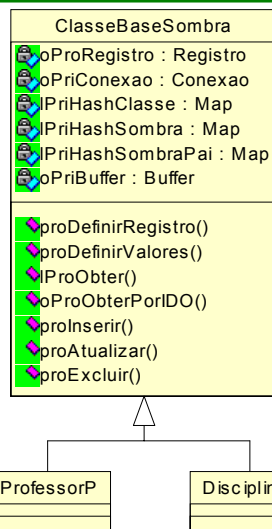
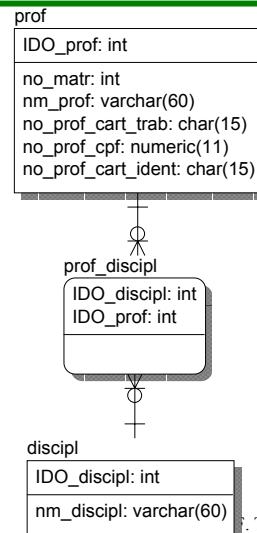


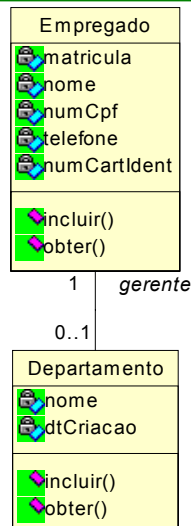
Diagrama Relacional
SGBDR



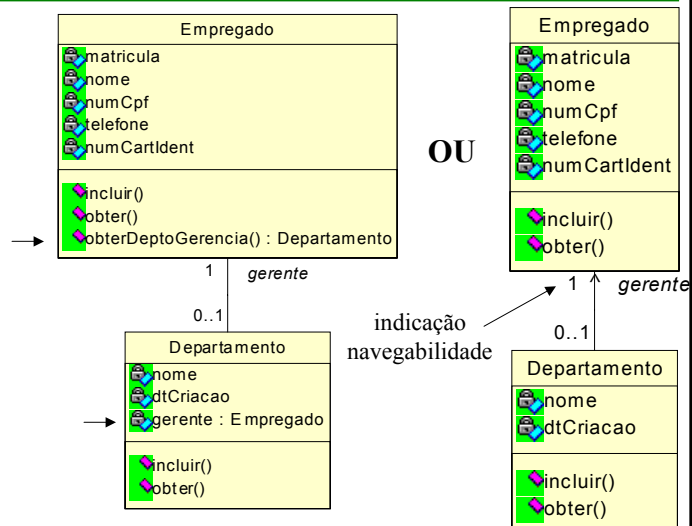
Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 113
TRADUÇÃO DE RELACIONAMENTO 1:1

**Diagrama de Classes
de Análise OO**

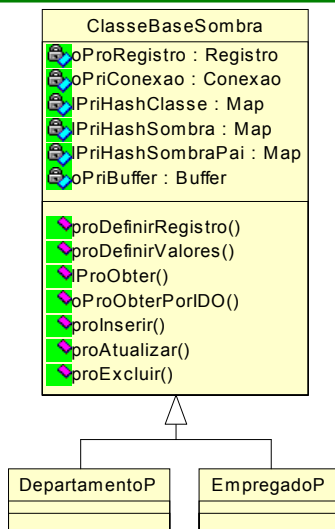


**Diagrama de Classes de Projeto OO
Componente de Domínio do Problema**

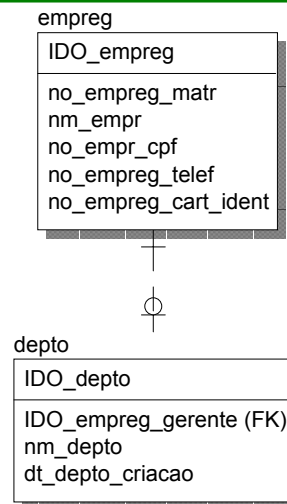


PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 114
TRADUÇÃO DE RELACIONAMENTO 1:1

**Diagrama de Classes - Projeto OO
Componente de Gerência de Dados**



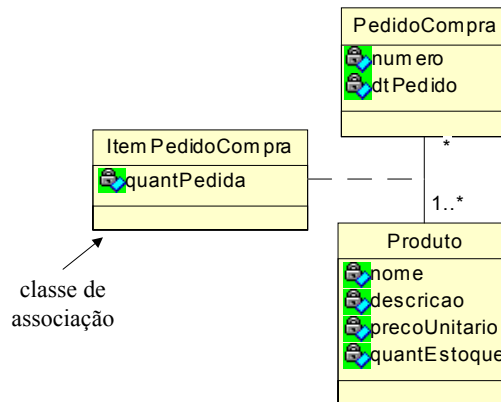
**Diagrama Relacional
SGBDR**



Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 115
TRADUÇÃO DE CLASSE DE ASSOCIAÇÃO

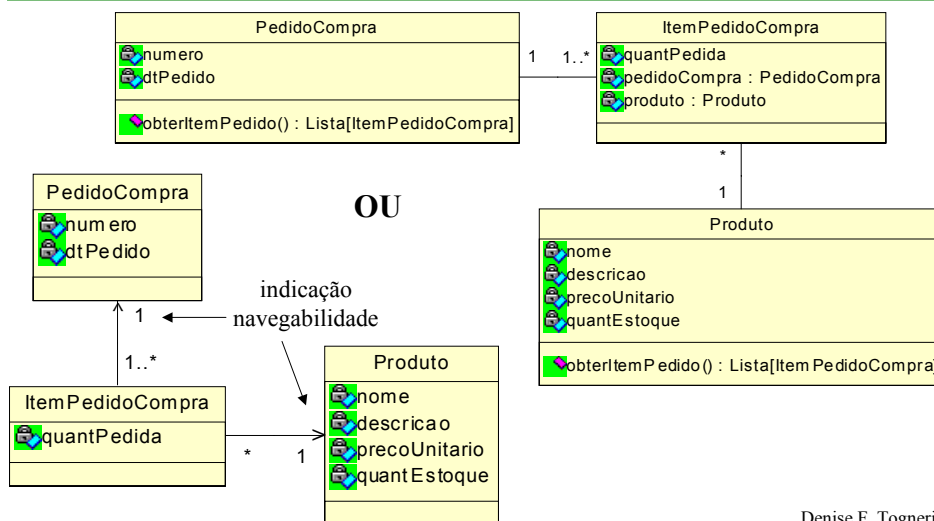
**Diagrama de Classes
de Análise OO**



Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 116
TRADUÇÃO DE CLASSE DE ASSOCIAÇÃO

**Diagrama de Classes de Projeto OO
Componente de Domínio do Problema**

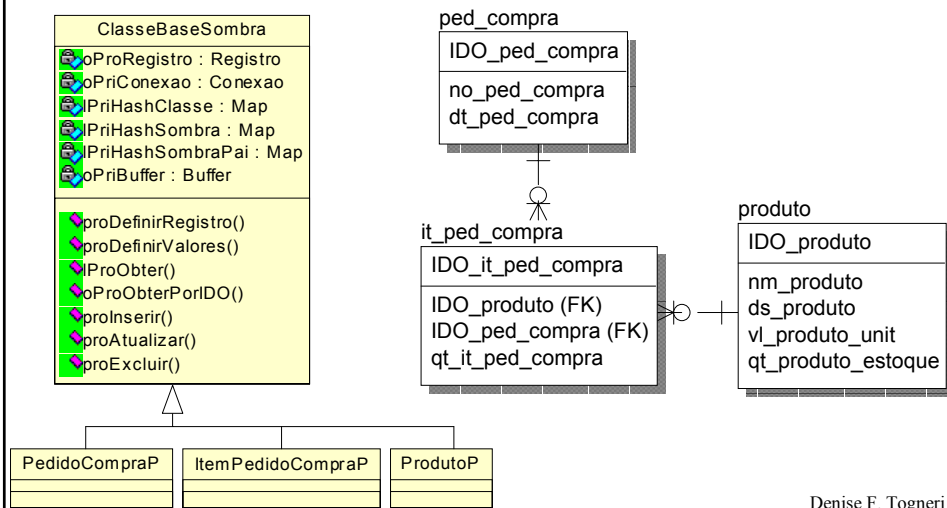


Denise F. Togneri

PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 117 **TRADUÇÃO DE CLASSE DE ASSOCIAÇÃO**

Diagrama de Classes - Projeto OO
Componente de Gerência de Dados

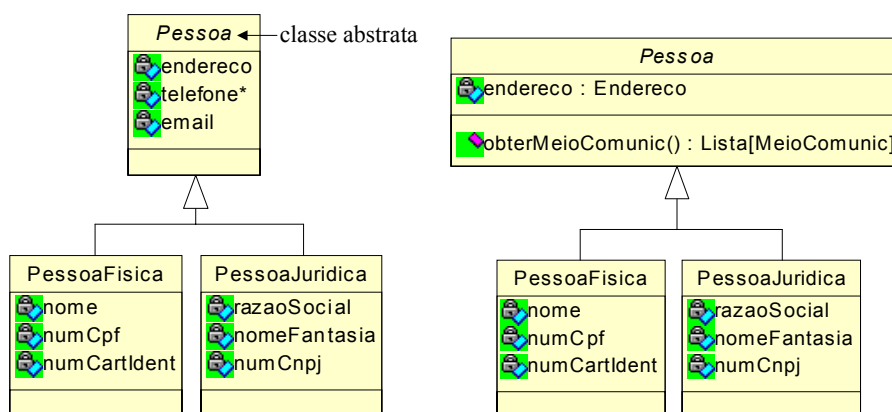
Diagrama Relacional
SGBDR



PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 118 **TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.**

Diagrama de Classes
de Análise OO

Diagrama de Classes de Projeto OO
Componente de Domínio do Problema

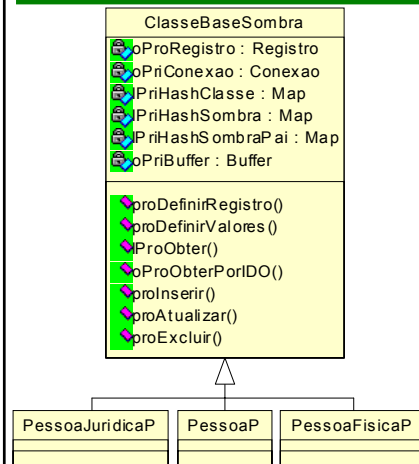


PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 119 **TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.**

PRIMEIRA SOLUÇÃO

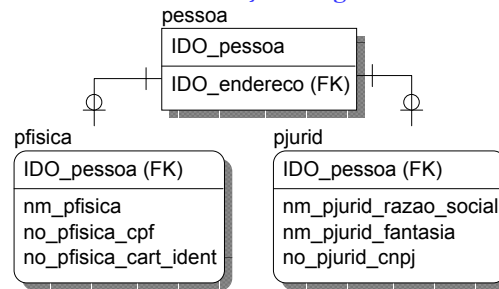
Diagrama de Classes - Projeto OO - CGD

Diagrama Relacional - SGBDR



PessoaP existe para suportar métodos que acessam tabela pessoa no SGBDR.

1a Solução - mais comum! Não é a única!
Ver observação a seguir



No mapeamento da hierarquia de herança, optou-se por criar uma tabela para cada classe na hierarquia. Uma justificativa para essa decisão ocorre quando houver muito processamento sobre a superclasse, o que inviabilizaria uma estratégia de tabelas somente para classes concretas.

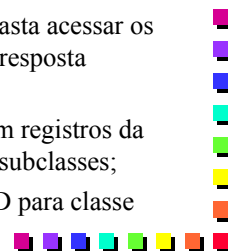
PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 120 **TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.**

PRIMEIRA SOLUÇÃO

COMPONENTE DE GERÊNCIA DE DADOS

OBSERVAÇÃO SOBRE CLASSE “SOMBRA” DO CGD PARA CLASSE ABSTRATA DO CDP

- Supondo que, para uma superclasse seja criada uma tabela no SGBDR e que para cada subclasse também seja criada uma tabela:
 - muitas vezes, para implementar alguns casos de uso, basta acessar os registros da tabela referente à superclasse para obter a resposta necessária;
 - Portanto, nestes casos, não existir métodos que acessam registros da superclasse sem precisar acessar registros referente às subclasses;
 - Sendo assim, vale a pena criar a classe sombra do CGD para classe abstrata do CDP.

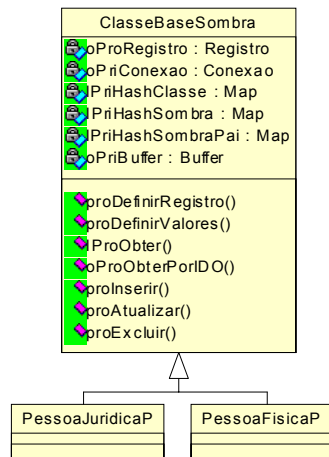


PROJ. ARQUIT. OO - PERSISTÊNCIA EM SGBDR 121 **TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.**

SEGUNDA SOLUÇÃO

Diagrama de Classes - Projeto OO - CGD

Diagrama Relacional - SGBDR



pfisica
IDO_pfisica
IDO_endereco (FK)
nm_pfisica
no_pfisica_cpf
no_pfisica_cart_ident

pjurid
IDO_pjurid
IDO_endereco (FK)
nm_pjurid_razao_social
nm_pjurid_fantasia
no_pjurid_cnpj

No mapeamento da hierarquia de herança, optou-se por criar apenas tabelas para classes concretas, no caso PessoaFisica e PessoaJuridica. Sendo assim, todas as relações e os atributos da superclasse Pessoa foram mapeados na tabelas correspondentes às classes concretas. Uma justificativa para essa decisão ocorre quando não houver processamento sobre a superclasse, mas apenas nas subclasses concretas.

Apenas as classes “sombra” PessoaFisicaP e PessoaJuridicaP foram criadas.

Denise F. Togneri

PROJ. ARQUIT. OO 122 **PERSISTÊNCIA EM B.D. RELACIONAL** **TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.**

MAPEAMENTO DE HERANÇA EM UM SGBD RELACIONAL

- 2 soluções possíveis
 - Utilizar uma tabela por classe concreta na hierarquia
 - Utilizar uma tabela por classe na hierarquia.

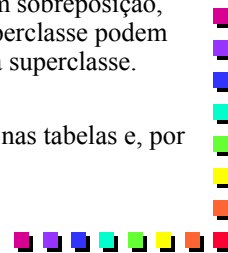


Denise F. Togneri

***PERSISTÊNCIA EM B.D. RELACIONAL
TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.***

MAPEAMENTO DE HERANÇA EM UM SGBD RELACIONAL

- **Utilizar uma tabela por classe concreta na hierarquia**
 - Cada tabela derivada para as classes concretas inclui tanto os atributos da classe quanto os de suas superclasses.
 - Esta solução não é viável quando há generalização com sobreposição, isto é, quando subclasses derivadas de uma mesma superclasse podem ocorrer simultaneamente para uma mesma instância da superclasse.
 - Neste caso, poderemos ter redundância de informação nas tabelas e, por conseguinte, inconsistência.

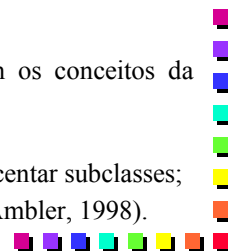


Denise F. Togneri

***PERSISTÊNCIA EM B.D. RELACIONAL
TRADUÇÃO DA ESTR. DE GEN. /ESPECIAL.***

MAPEAMENTO DE HERANÇA EM UM SGBD RELACIONAL

- **Utilizar uma tabela por classe na hierarquia**
 - Solução mais genérica e não apresenta o problema anterior;
 - Deve haver uma tabela para cada classe e visões para cada uma das classes derivadas (subclasses).
 - Esta abordagem é a que está mais de acordo com os conceitos da orientação a objetos.
 - É muito mais fácil modificar uma superclasse e acrescentar subclasses;
 - Além disso, ela suporta o polimorfismo muito bem (Ambler, 1998).



Denise F. Togneri

DIAGRAMA DE SEQÜÊNCIA DE PROJETO OO

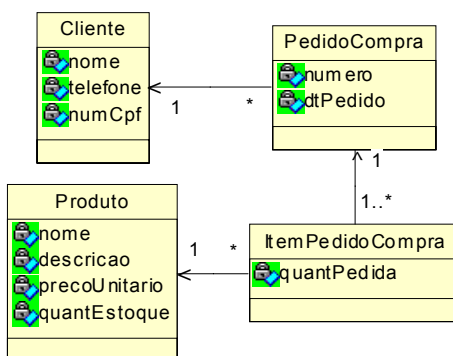
DIAGRAMA DE SEQÜÊNCIA DE PROJETO OO

126

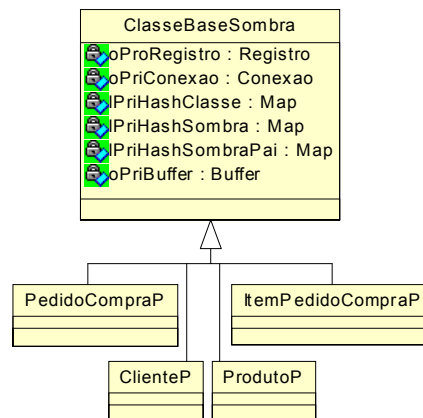
Os Diagramas de Seqüência de Análise OO devem ser alterados para contemplar todas as decisões tomadas na atividade de Projeto.

- Exemplo: Supondo os Diagramas de Classes de Projeto abaixo:

Componente de Domínio de Problema



Componente de Gerência de Dados



Denise F. Togneri

DIAGRAMA DE SEQÜÊNCIA DE PROJETO OO 127

O Diagrama de Seqüência de Projeto do caso de uso **IncluirPedidoCompra** poderia ser:

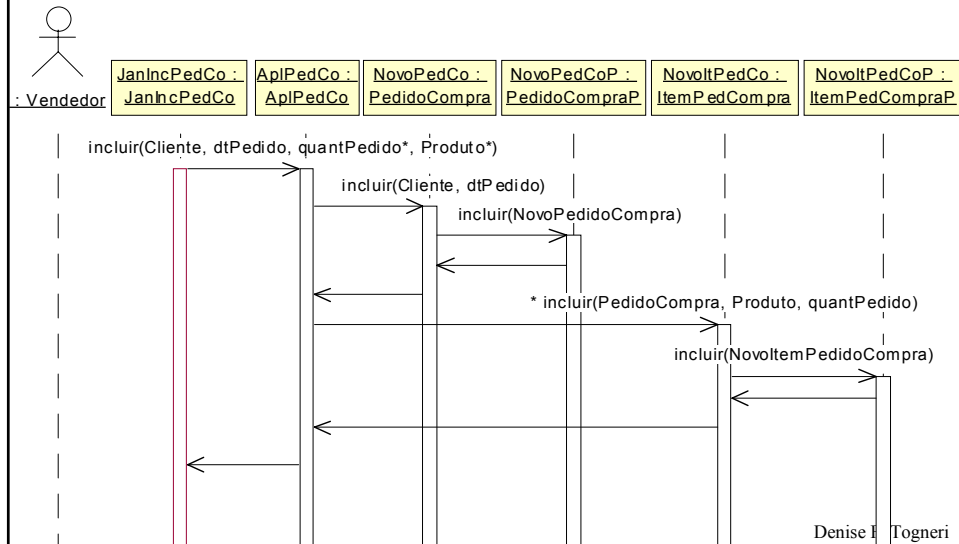
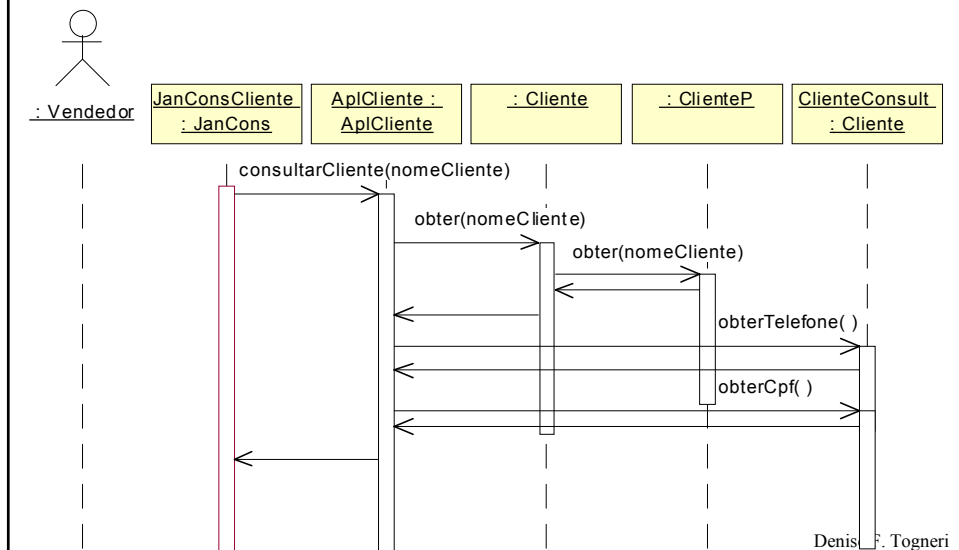


DIAGRAMA DE SEQÜÊNCIA DE PROJETO OO 128

O Diagrama de Seqüência de Projeto do caso de uso **ConsultarCliente** poderia ser:





MODELAGEM DA ARQUITETURA FÍSICA DO SISTEMA OO (Booch et al., 2000)



MODELAGEM DA ARQUITETURA FÍSICA DO SISTEMA ORIENTADO A OBJETOS

130

- Podem ser utilizados:
 - **Diagrama de Componentes**
 - Mostra um conjunto de componentes e seus relacionamentos.
 - São empregados para a modelagem da visão estática de **implementação** de um sistema.
 - Isto envolve a modelagem de itens físicos que residem em um nó, tais como executáveis, bibliotecas, tabelas, arquivos e documentos;
 - **Diagrama de Implantação**
 - Mostra a configuração de nós de processamento em tempo de execução e os componentes que neles existem.
 - São empregados para a modelagem da visão estática de **funcionamento** de um sistema.
 - Esta visão direciona primariamente a distribuição, entrega e instalação das partes que formam o sistema físico.
- Graficamente, tanto o diagrama de componentes quanto o de implantação podem ser vistos como **uma coleção de vértices e arcos**.

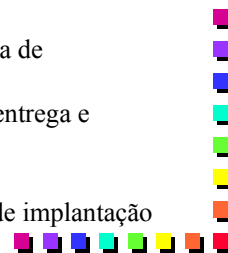
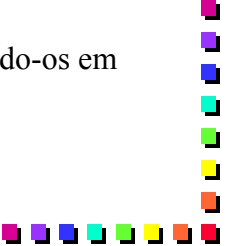




DIAGRAMA DE COMPONENTES

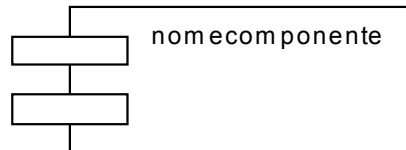


DIAGRAMA DE COMPONENTES 132

- Os Diagramas de Componentes costumam conter (Booch *et al.*, 2000, p. 389):
 - Componentes
 - Interfaces
 - Relacionamentos de dependência, generalização, associação e realização
 - Pacotes (ou subsistemas), notas e restrições.
 - Os componentes podem ser organizados, agrupando-os em pacotes da mesma maneira que as classes.
- 

Um componente é a parte física e substituível de um sistema ao qual se adapta e fornece a realização de um conjunto de interfaces. Graficamente, é representado como um retângulo com abas (Booch *et al.*, 2000, p. 343).

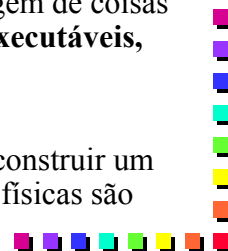
NOTAÇÃO GRÁFICA UML - COMPONENTE



- Essa notação permite visualizar um componente independente de qualquer sistema operacional ou linguagem de programação.
- Todo componente deve ter um nome único, definido a partir do vocabulário de implementação e, dependendo do sistema operacional destino, incluem extensões tais como **.dll** ou **.exe** ou **.jar**
- **OBS:** Uma *interface* é uma coleção de operações utilizadas para especificar um serviço de uma classe ou componente.

Denise F. Togneri

- **Os componentes vivem no mundo material dos bits, ou seja, efetivamente residem nos nós físicos e podem ser executados diretamente ou, de alguma maneira indireta, participar de um sistema em execução.**
- Portanto, são um importante bloco de construção para a modelagem de aspectos físicos de um sistema.
- Os componentes são empregados para a modelagem de coisas físicas que podem residir em um nó, tais como **executáveis, bibliotecas, tabelas, arquivos e documentos.**
- A **modelagem física** de um sistema é feita para construir um sistema executável. Na UML, todas essas coisas físicas são modeladas como componentes.



Denise F. Togneri

COMPONENTES E CLASSES 135

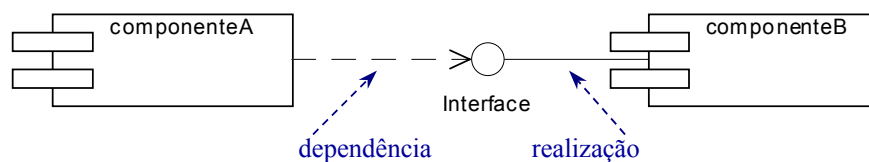
COMPONENTES	CLASSES
COISAS EM COMUM	
<ul style="list-style-type: none"> ■ Têm nomes. ■ Podem realizar um conjunto de interfaces. ■ Podem participar de um relacionamento de dependência, generalização e associação. ■ Admitem instâncias. ■ Podem ser participantes de interações. 	<ul style="list-style-type: none"> ■ Têm nomes. ■ Podem realizar um conjunto de interfaces. ■ Podem participar de um relacionamento de dependência, generalização e associação. ■ Admitem instâncias. ■ Podem ser participantes de interações.
DIFERENÇAS	
<ul style="list-style-type: none"> ■ Representam coisas físicas que vivem no mundo dos bits. ■ Podem viver em nós. ■ Em geral, somente têm operações que são alcançadas por meio de suas interfaces. ■ Representam o pacote físico de componentes lógicos. 	<ul style="list-style-type: none"> ■ Representam abstrações lógicas. ■ Não podem viver em nós. ■ Podem ter atributos e operações diretamente.

Denise F. Togneri

COMPONENTES E INTERFACES 136

Uma **interface** é uma coleção de operações utilizadas para especificar um serviço de uma classe ou componente. Na UML, é representada graficamente como um círculo.

NOTAÇÃO GRÁFICA UML – INTERFACE (FORMA ICÔNICA)

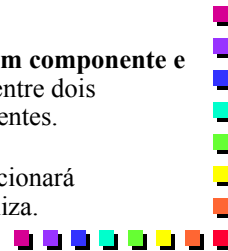


- As interfaces ultrapassam as fronteiras lógica e física. A mesma interface utilizada ou realizada por um componente será encontrada utilizada ou realizada pelas classes que o componente implementa.

Denise F. Togneri

COMPONENTES E INTERFACES 137

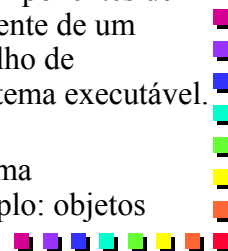
- **Interface de exportação:**
 - é uma **interface realizada** por um componente, significando uma interface em que o componente fornece um serviço para outros componentes. Um componente pode fornecer muitas interfaces de exportação.
- **Interface de importação:**
 - é uma **interface utilizada** por um componente, significando uma interface a qual o componente se adapta e a partir da qual é construído. Um componente poderá estar em conformidade com muitas interfaces de importação. Além disso, um componente pode tanto importar quanto exportar interfaces.
- **Uma determinada interface poderá ser exportada por um componente e importada por outro.** O fato dessa interface se encontrar entre dois componentes quebra a dependência direta entre os componentes.
- Um componente que utiliza uma determinada interface funcionará adequadamente, qualquer que seja o componente que a realiza.



Denise F. Togneri

TIPOS DE COMPONENTES 138

- **Componentes de Implantação.** São os componentes necessários e suficientes para formar um sistema executável, tais como as bibliotecas de ligações dinâmicas (DLLs) e os executáveis (EXEs). A definição UML para componentes é suficientemente ampla para incluir modelos clássicos de objetos, tais como COM+, CORBA e *Enterprise Java Beans*.
- **Componentes do produto do trabalho.** São o resíduo do processo de desenvolvimento, formados por arquivos de código-fonte e arquivos de dados a partir dos quais os componentes de implantação são criados. Não participam diretamente de um sistema executável, mas são os produtos do trabalho de desenvolvimento, utilizados para a criação do sistema executável.
- **Componentes de execução.** São criados como uma consequência de um sistema em execução. Exemplo: objetos COM+ instanciados a partir de uma DLL.



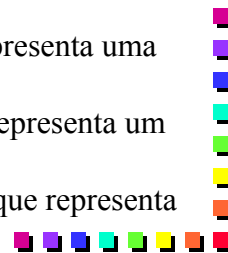
Denise F. Togneri

ESTEREÓTIPOS-PADRÃO PARA COMPONENTES ¹³⁹

O **estereótipo** é um mecanismo de extensibilidade da UML que permite ampliar o vocabulário da UML, através da criação de novos tipos de blocos de construção, derivados de outros já existentes, mas específicos a determinado problema.

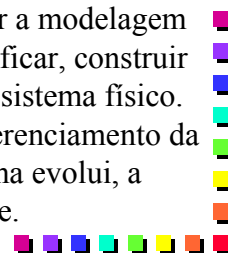
- Existem 5 **estereótipos-padrão para componentes** na UML:
 - **EXECUTÁVEL**: especifica um componente que poderá ser executado em um nó;
 - **BIBLIOTECA**: especifica uma biblioteca de objetos estática ou dinâmica;
 - **TABELA**: especifica um componente que representa uma tabela de banco de dados;
 - **ARQUIVO**: especifica um componente que representa um documento contendo código-fonte ou dados;
 - **DOCUMENTO**: especifica um componente que representa um documento.

- OBS: a UML não especifica ícones definidos para esses estereótipos. Denise F. Togneri



MODELAGEM DE EXECUTÁVEIS E DE BIBLIOTECAS ¹⁴⁰

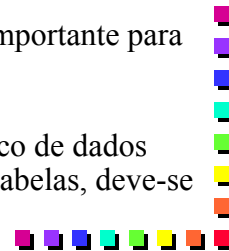
- O propósito mais comum para o qual os componentes serão utilizados é a modelagem de componentes de implantação que compõem a sua implementação.
- Se a **implementação do sistema consiste em exatamente um único executável**, não é necessário modelar componentes.
- Se o sistema for composto por vários executáveis e está associado a bibliotecas de objetos, deve-se fazer a modelagem de componentes que auxiliará a visualizar, especificar, construir e documentar as decisões tomadas em relação ao sistema físico. Quando for necessário controlar as versões e o gerenciamento da configuração dessas partes, à medida que o sistema evolui, a modelagem de componentes também é importante.



Denise F. Togneri

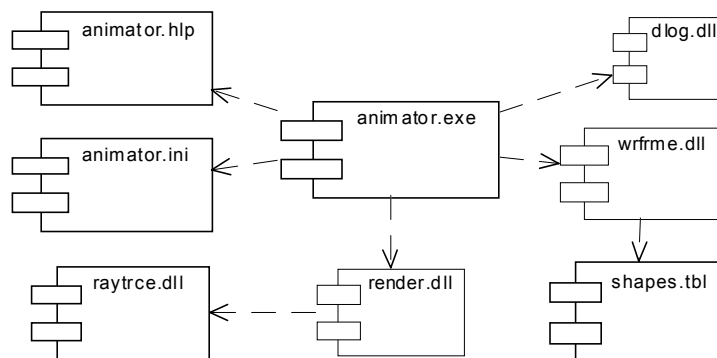
MODELAGEM DE TABELAS, ARQUIVOS E DOCUMENTOS 141

- Algumas vezes, será necessário modelar alguns **componentes de implantação auxiliares** que não são executáveis, nem bibliotecas, e ainda assim, são críticos para a entrega física do sistema.
- **Exemplos:** arquivos de dados, documentos de ajuda, scripts, arquivos de registro, arquivos de iniciação e arquivos de instalação/remoção.
- A modelagem desses componentes é uma parte importante para controlar a configuração do sistema.
- No entanto, em geral, como a modelagem de banco de dados pode se tornar complicada em função das várias tabelas, deve-se fazê-la em um modelo próprio.



Denise F. Togneri

EXEMPLO DE DIAGRAMA DE COMPONENTES 142



- A figura acima apresenta um **executável** (animator.exe) e
 - quatro **bibliotecas** (dlog.dll, wrfme.dll, render.dll e raytrce.dll),
 - um **documento** (animator.hlp),
 - um **arquivo simples** (animator.ini) e
 - uma **tabela de banco de dados** (shapes.tbl)
 que fazem parte do sistema entregue ao redor do executável.

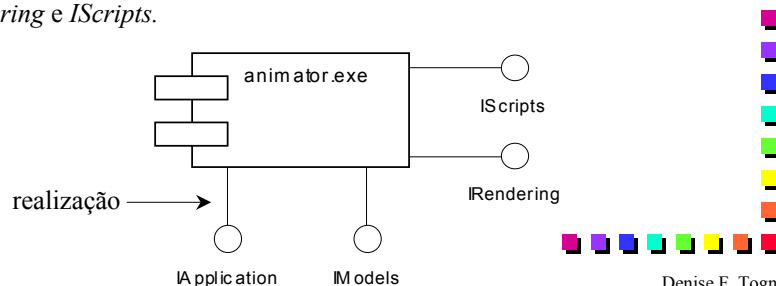


Denise F. Togneri

MODELAGEM DE UMA API – INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES

143

- Se um sistema for desenvolvido/montado a partir de partes de componentes, freqüentemente é desejável ver as **interfaces de programação de aplicações (APIs)**, que podem ser utilizadas para a montagem dessas partes.
- As APIs representam as costuras programáticas de um sistema, que podem ser modeladas pela utilização de interfaces e componentes. Uma API é essencialmente uma **interface** realizada por um ou mais componentes.
- A figura abaixo expõe as APIs do executável animator.exe. Existem quatro interfaces que formam a API do executável: *IApplication*, *IModels*, *IScripts* e *IRendering*.



Denise F. Togneri

MODELAGEM DE CÓDIGO-FONTE

144

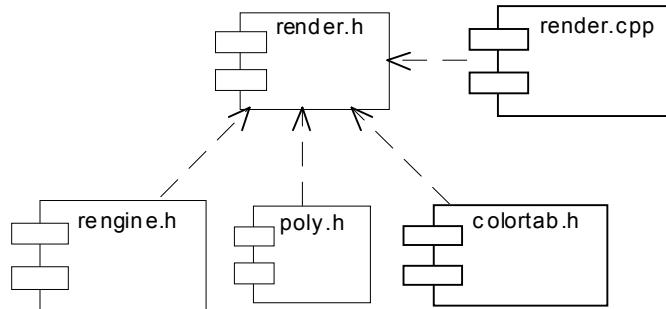
- O **objetivo mais comum** para o qual os componentes são empregados é a modelagem das partes físicas que compõem sua implementação, incluindo tabelas, arquivos, documentos e APIs.
- O **segundo objetivo mais comum** para o uso dos componentes é a **modelagem da configuração de todos os arquivos de código-fonte que as ferramentas de desenvolvimento usam para criar esses componentes. Esses representam os componentes do produto do trabalho do processo de desenvolvimento.**
- A modelagem gráfica do código-fonte é particularmente útil para a visualização das dependências de compilação entre seus arquivos de código-fonte e para o gerenciamento da separação e reunião de grupos desses arquivos ao criar bifurcações e uniões de caminhos de desenvolvimento.
- Dessa maneira, os componentes da UML podem ser interfaces gráficas para as ferramentas de gerenciamento da configuração e de controle de versão.



Denise F. Togneri

EXEMPLO DE DIAGRAMA DE COMPONENTES MODELAGEM DE CÓDIGO-FONTE EM C++

145



- O exemplo acima mostra alguns arquivos de código-fonte que são utilizados para construir a **biblioteca render.dll**.
- Essa figura inclui quatro **arquivos de cabeçalhos em C++ (header files)**, chamados *render.h*, *engine.h*, *poly.h* e *colortab.h*, representando o código-fonte para a especificação de certas classes. Também existe um **arquivo de implementação ou de corpo em C++ (body file)**, chamado *render.cpp*, representando a implementação de um desses cabeçalhos.

Denise F. Togneri

DICAS E SUGESTÕES

146

- Ao fazer a **modelagem de componentes na UML**, deve-se lembrar que está se modelando a dimensão física.
- Um componente bem estruturado
 - fornece uma abstração clara de algo definido a partir do aspecto físico do sistema;
 - fornece a realização de um conjunto de interfaces, pequeno e bem-definido.
 - implementa diretamente um conjunto de classes que trabalham juntas para a execução da semântica dessas interfaces com economia e elegância;
 - está fracamente acoplado a outros componentes; muitas vezes, será feita a modelagem de componentes somente em conexão com relacionamentos de dependência e de realização.

Denise F. Togneri



DIAGRAMA DE IMPLANTAÇÃO



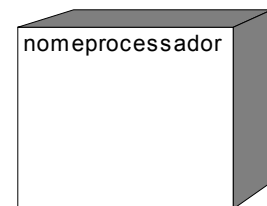
DIAGRAMA DE IMPLANTAÇÃO

148

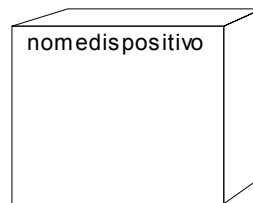
- Os Diagramas de Implantação costumam conter (Booch *et al.*, 2000, p. 404):
 - Nós
 - Relacionamentos de dependência e associação entre os nós
 - Componentes, cada um dos quais deve residir em algum nó
 - Pacotes (ou subsistemas), utilizados para agrupar elementos do modelo em conjuntos maiores
 - Notas e restrições.
- São empregados para a modelagem da visão estática de **funcionamento** de um sistema. Esta visão direciona primariamente a distribuição, entrega e instalação das partes que formam o sistema físico.

Um nó é um elemento físico que existe em tempo de execução e representa um recurso computacional, geralmente tendo pelo menos alguma memória e, com frequência, capacidade de processamento. Gráficamente, é representado por um cubo (Booch *et al.*, 2000, p. 357).

NOTAÇÃO GRÁFICA UML - NÓ



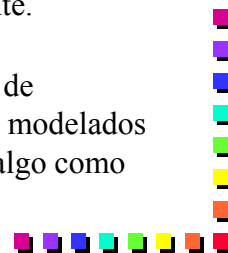
PROCESSADOR



DISPOSITIVO

Denise F. Togneri

- Os nós são empregados para a modelagem da topologia do hardware em que o sistema é executado.
- Representam tipicamente um **processador** ou um **dispositivo** em que os componentes poderão ser instalados e executados. Todo nó deve ter um nome único.
- Um **processador** é um nó que tem capacidade de processamento, significando que ele pode executar um componente.
- Um **dispositivo** é um nó que não tem capacidade de processamento (pelo menos, nenhum dos que são modelados nesse nível de abstração) e, em geral, representa algo como interfaces para o mundo real.



Denise F. Togneri

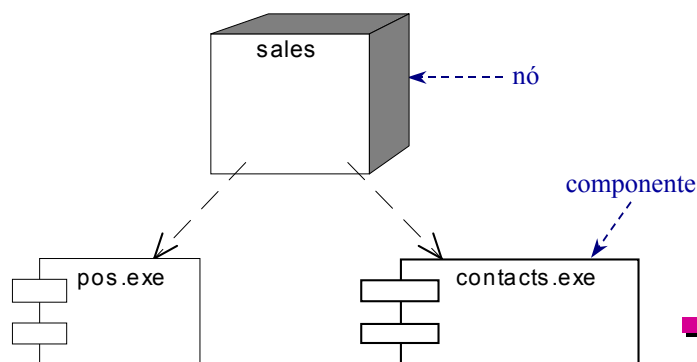
NÓS E COMPONENTES 151

COMPONENTES	NÓS
COISAS EM COMUM	
<ul style="list-style-type: none"> ■ Têm nomes. ■ Podem participar de relacionamentos de dependência, generalização e associação. ■ Admitem instâncias. ■ Podem ser participantes de interações. 	<ul style="list-style-type: none"> ■ Têm nomes. ■ Podem participar de relacionamentos de dependência, generalização e associação. ■ Admitem instâncias. ■ Podem ser participantes de interações.
DIFERENÇAS	
<ul style="list-style-type: none"> ■ São itens que participam da execução de um sistema. São executados pelos nós. ■ Representam os pacotes físicos de elementos lógicos. 	<ul style="list-style-type: none"> ■ São itens que executam os componentes. ■ Representam o funcionamento físico dos componentes. Localização em que os componentes são instalados.

Denise F. Togneri

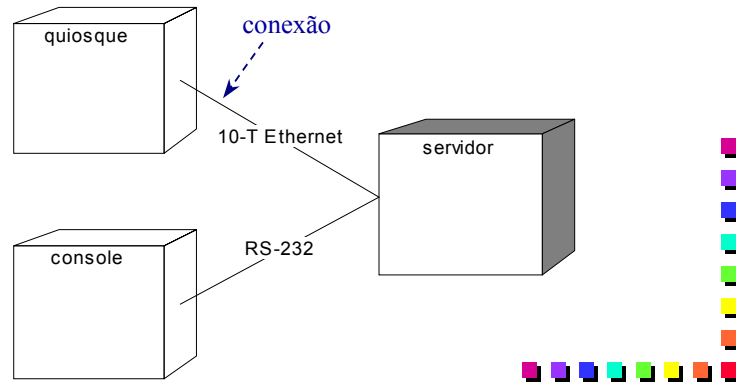
RELACIONAMENTO ENTRE NÓS E COMPONENTES

- O **relacionamento entre um nó e os componentes instalados** pode ser mostrado explicitamente pela utilização de um **relacionamento de dependência** mas, na maioria das vezes, não é necessário modelá-lo, e sim mantê-los como parte da especificação do nó.
- Um conjunto de objetos ou componentes que são alocados a um nó como um grupo é chamado de **unidade de distribuição**.



Denise F. Togneri

- O tipo de relacionamento mais comum que será utilizado entre os nós é uma associação.
- Nesse contexto, uma associação representa uma conexão física entre os nós, assim como uma conexão de Ethernet, uma linha serial ou um barramento compartilhado.



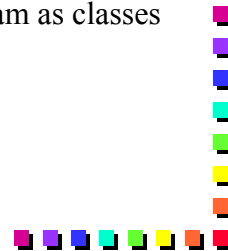
Denise F. Togneri

PROJETO DE OBJETOS

PROJETO DE OBJETOS

155

- Devemos desenvolver:
 - um projeto detalhado dos atributos e das operações que compõem cada classe, e
 - uma especificação das mensagens que conectam as classes com seus colaboradores.

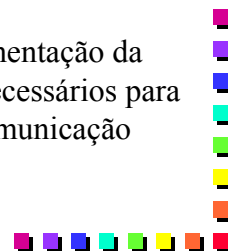


Denise F. Togneri

PROJETO DE OBJETOS

156

- Inicialmente, uma descrição de protocolo para cada classe deve ser provida, estabelecendo o conjunto de mensagens da classe (sua interface) e uma descrição da operação a ser executada quando um objeto da classe receber uma dessas mensagens.
- Neste momento, deve-se definir, portanto, que operações e atributos devem ser públicos ou privados à classe.
- A seguir, deve-se fazer uma descrição da implementação da classe, provendo detalhes internos (“ocultos”) necessários para a implementação, mas não necessários para a comunicação entre objetos.



Denise F. Togneri

- **No que tange aos atributos**, esta descrição deve conter:
 - uma especificação das estruturas de dados privadas da classe, com indicações de itens de dados e tipos para os atributos.
 - Deve-se definir, também, a navegabilidade das associações. Esta decisão conduzirá à definição de novas variáveis na classe, bem como do seu tipo e estrutura de dados.
- **Para as operações**, deve-se definir:
 - os tipos e estruturas de dados para as interfaces, bem como uma especificação procedural de cada operação (projeto algorítmico).
 - No caso de operações complexas, é uma boa opção modularizá-las, criando sub-operações, estas privadas à classe.
 - O projeto algorítmico de uma operação pode revelar a necessidade de variáveis locais aos métodos ou de variáveis globais à classe para tratar detalhes internos.

Denise F. Togneri

PROJETO DE OBJETOS 158
VISIBILIDADE DE ATRIBUTOS E OPERAÇÕES

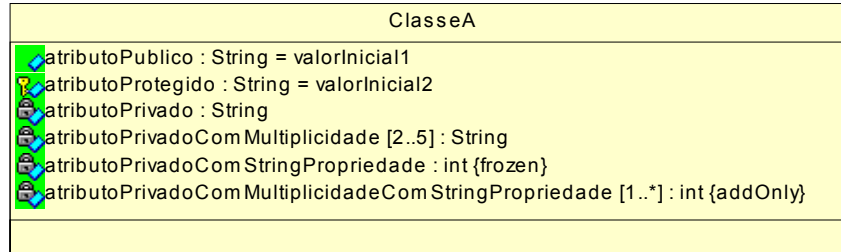
A visibilidade de um atributo ou operação de uma classe especifica se ela pode ser utilizada por outras classes.

- Existem **3 níveis de visibilidade**:
 - **Público**
 - Qualquer **classe** pode usar a característica
 - Especificado por ser antecedido pelo **símbolo +**
 - **Protegido**
 - qualquer **descendente da classe** é capaz de usar a característica
 - Especificado por ser antecedido pelo **símbolo #**
 - **Privado**
 - **Somente a própria classe** é capaz de usar a característica
 - Especificado por ser antecedido pelo **símbolo -**

Denise F. Togneri

SINTAXE DE UM ATRIBUTO NA UML

[visibilidade] nome [multiplicidade] [: tipo] [= valor-inicial] [{string-propriedade}]

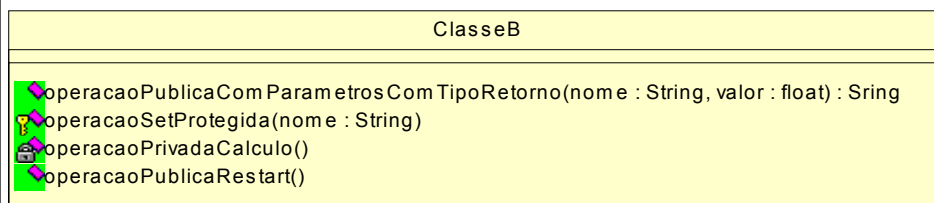


- Existem 3 **propriedades** definidas que podem ser utilizadas com os atributos
 - changeable**: (default) não há restrições para se modificar o valor do atributo;
 - addOnly**: no caso de atributos com multiplicidade maior do que um, valores adicionais poderão ser incluídos, mas uma vez criado, o valor não poderá ser removido ou alterado;
 - frozen**: o valor do atributo não poderá ser modificado depois que o objeto for iniciado. No C++ por exemplo, essa propriedade é mapeada para const

Denise F. Togneri

SINTAXE DE UMA OPERAÇÃO NA UML

[visibilidade] nome [(lista-de-parâmetros)] [: tipo-de-retorno]
[{string-propriedade}]



- Existem várias **propriedades** que podem ser utilizadas para as operações tais como *leaf*, *isQuery*, *sequential*, *guarded* e *concurrent* (Booch et al., 2000, p. 129) .

Denise F. Togneri



CRITÉRIOS DE QUALIDADE DE PROJETOS



CRITÉRIOS DE QUALIDADE DE PROJETOS

162

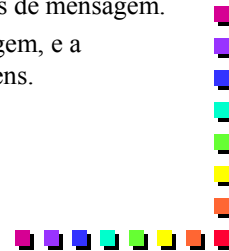
- Um bom projeto equilibra custo e benefício de modo a minimizar o custo total do sistema ao longo de seu tempo de vida total (e não somente durante a construção).
- Coad e Yourdon (1993) propõem alguns critérios baseados na observação e estudos de casos reais, entre eles:
 - Acoplamento;
 - Coesão;
 - Reutilização;
 - Clareza do Projeto;
 - Efetivo Uso da Herança;
 - Protocolo de Mensagens Simples;
 - Operações Simples;
 - Habilidade de “avaliar por cenário”.

CRITÉRIOS DE QUALIDADE DE PROJETOS

163

■ *Acoplamento:*

- diz respeito ao grau de interdependência entre componentes de software.
- O objetivo é minimizar o acoplamento, isto é, tornar os componentes tão independentes quanto possível.
- No OOD, estamos preocupados principalmente com o acoplamento entre classes e entre subsistemas, gerado por conexões de mensagem.
- A meta é minimizar o número de conexões de mensagem, e a complexidade e o volume de informação nas mensagens.



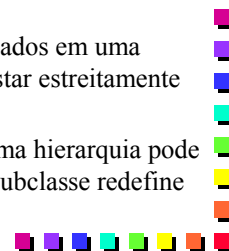
Denise F. Togneri

CRITÉRIOS DE QUALIDADE DE PROJETOS

164

■ *Coesão:*

- define como as atividades de diferentes componentes de software estão relacionadas umas com as outras.
- Vale a pena ressaltar que coesão e acoplamento são interdependentes e, portanto, uma boa coesão geralmente conduz a um pequeno acoplamento.
- No OOD, três níveis de coesão devem ser verificados:
 - coesão de métodos individuais: um método deve executar uma e somente uma função, passível de ser descrita por uma sentença contendo um único verbo e um único objeto;
 - coesão de uma classe: atributos e serviços encapsulados em uma classe devem ser altamente coesos, isto é, devem estar estreitamente relacionados; e
 - coesão de uma hierarquia de classes: a coesão de uma hierarquia pode ser avaliada examinando-se até que extensão uma subclasse redefine ou cancela atributos e métodos herdados da superclasse.

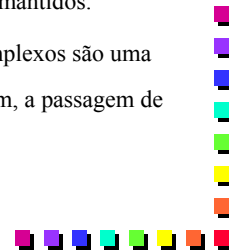


Denise F. Togneri

CRITÉRIOS DE QUALIDADE DE PROJETOS

165

- *Reutilização*: bons projetos devem ser fáceis de serem reutilizados.
- *Clareza do Projeto*: um projeto deve ser passível de entendimento por outros projetistas.
- *Efetivo Uso da Herança*: para sistemas médios, com aproximadamente 100 classes, as hierarquias devem ter de 2 a 7 níveis de generalização-especialização. Projetos com uso intensivo de herança múltipla devem ser evitados, pois são mais difíceis de serem entendidos e, conseqüentemente, de serem reutilizados e mantidos.
- *Protocolo de Mensagens Simples*: protocolos de mensagem complexos são uma indicação comum de acoplamento excessivo entre classes. Assim, a passagem de muitos parâmetros deve ser evitada.

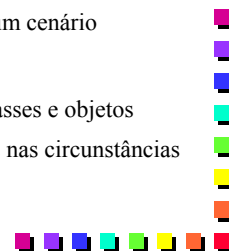


Denise F. Togneri

CRITÉRIOS DE QUALIDADE DE PROJETOS

166

- *Operações Simples*:
 - os métodos que implementam as operações de uma classe devem ser bastante pequenos.
 - Se um método envolve muito código, é uma forte indicação de que as operações da classe foram pobremente fatoradas.
- *Habilidade de “avaliar por cenário”*:
 - é importante que um projeto possa ser avaliado a partir de um cenário particular escolhido.
 - Revisores devem poder representar o comportamento de classes e objetos individuais, e assim, verificar o comportamento dos objetos nas circunstâncias desejadas.



Denise F. Togneri



PADRÕES DE PROJETO (DESIGN PATTERNS)

Referência:

GAMMA, Erich. **Padrões de Projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

