Engenharia de Software

Arquitectura de Software
Padrões de Arquitectura de Software

Luís Morgado

Instituto Superior de Engenharia de Lisboa
Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

Arquitectura de Software

- Métricas
- Princípios
- **Padrões**

Padrões de Arquitectura de Software

- No âmbito da Engenharia de Software
  - **Solução geral**, reproduzível, para um problema de ocorrência frequente
  - **Não é uma definição acabada** que possa ser directamente implementada
  - **É uma descrição (padrão) acerca de como resolver um problema** que pode ser utilizada em diferentes situações
- Utilização
  - Podem permitir **acelerar o processo de desenvolvimento** ao proporcionar soluções já anteriormente testadas
  - Podem permitir **identificar por antecipação questões não imediatamente aparentes** do problema a resolver
  - Disponibilizam **soluções de carácter geral** não comprometidas com um problema específico

Padrões de Arquitectura de Software

- Elementos base:
  - **Designação**
    - Para comunicação e documentação
  - **Problema**
    - Problema abordado
  - **Solução**
    - Partes e relações que constituem o padrão
  - **Consequências**
    - Custos
    - Benefícios
    - Alternativas

Padrões de Arquitectura de Software

- Classificação
  - **Nível de arquitectura**
    - **Subsistemas**
      - Agregados de mecanismos
      - Funcionalidade global
    - **Mecanismos**
      - Agregados de elementos
      - Funcionalidade local
  - **Domínios de problema**
    - Gerais
    - Interface
    - Criação e persistência de objectos
    - Análise sintáctica

Padrões GRASP

(*General Responsibility Assignment Software Patterns*)

- **Responsabilidade**
  - **Comportamental (fazer)**:
    - Realizar uma operação
    - Iniciar outros objectos
    - Interactuar com outros objectos
  - **Estrutural (conhecer)**:
    - Dados privados
    - Dados referentes a outros objectos
    - Dados derivados ou calculados
- **Padrões principais** [Larman, 2004]
  - *High Cohesion*
  - *Low Coupling*
  - *Creator*
  - *Controller*

Padrões de Organização de Subsistemas

Padrão Camadas (*Layers*)

Layers

Context

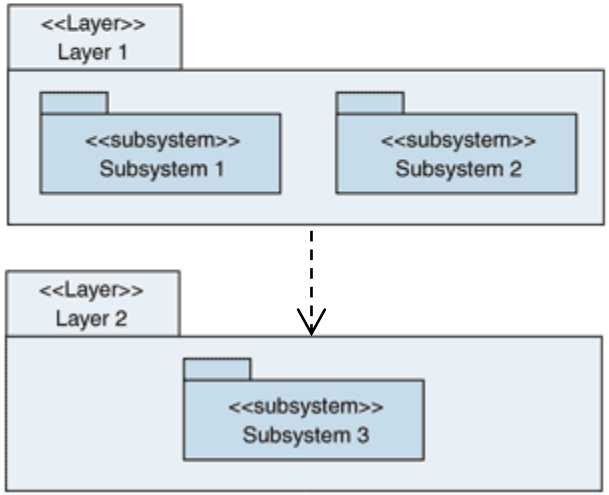
You are working with a large, complex system and you want to manage complexity by decomposition

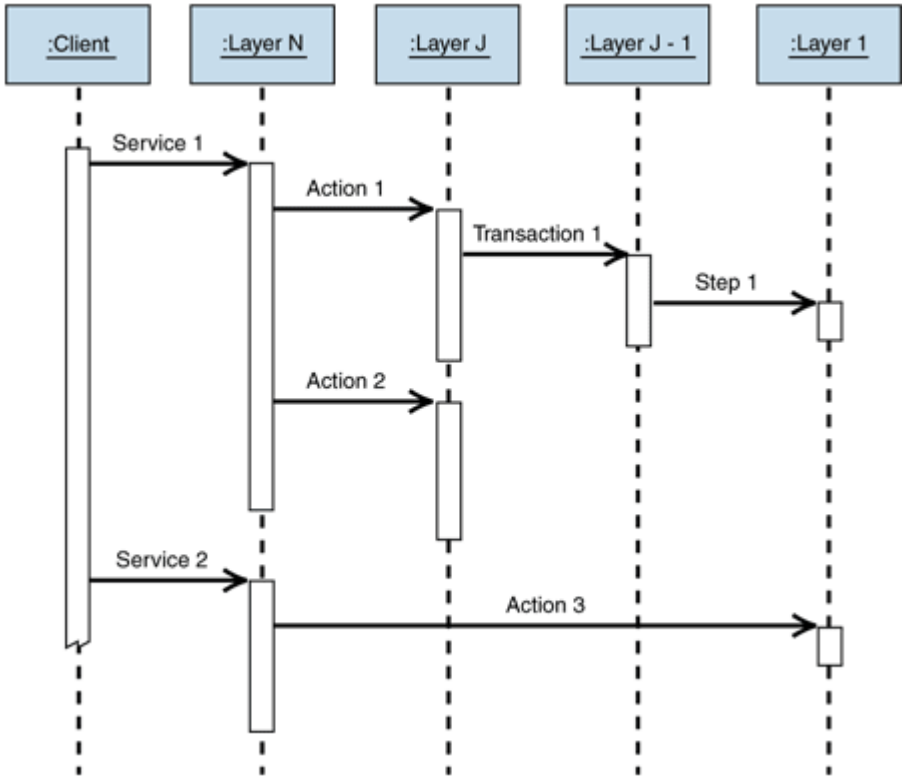
Problem

How do you structure an application to support such operational requirements as maintainability, reusability, extensibility, scalability, robustness, and security?

Solution

Compose the solution into a set of layers. Each layer should be cohesive and at roughly the same level of abstraction. Each layer should be loosely coupled to the layers underneath ....

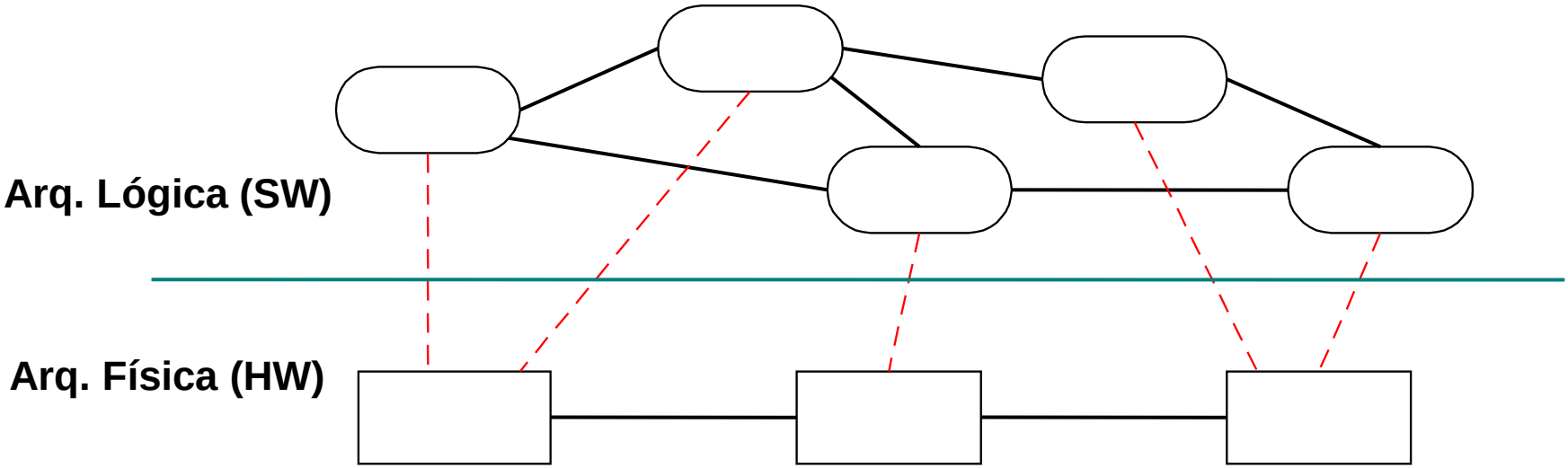




.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Padrões de Organização de Subsistemas

Níveis de Arquitectura

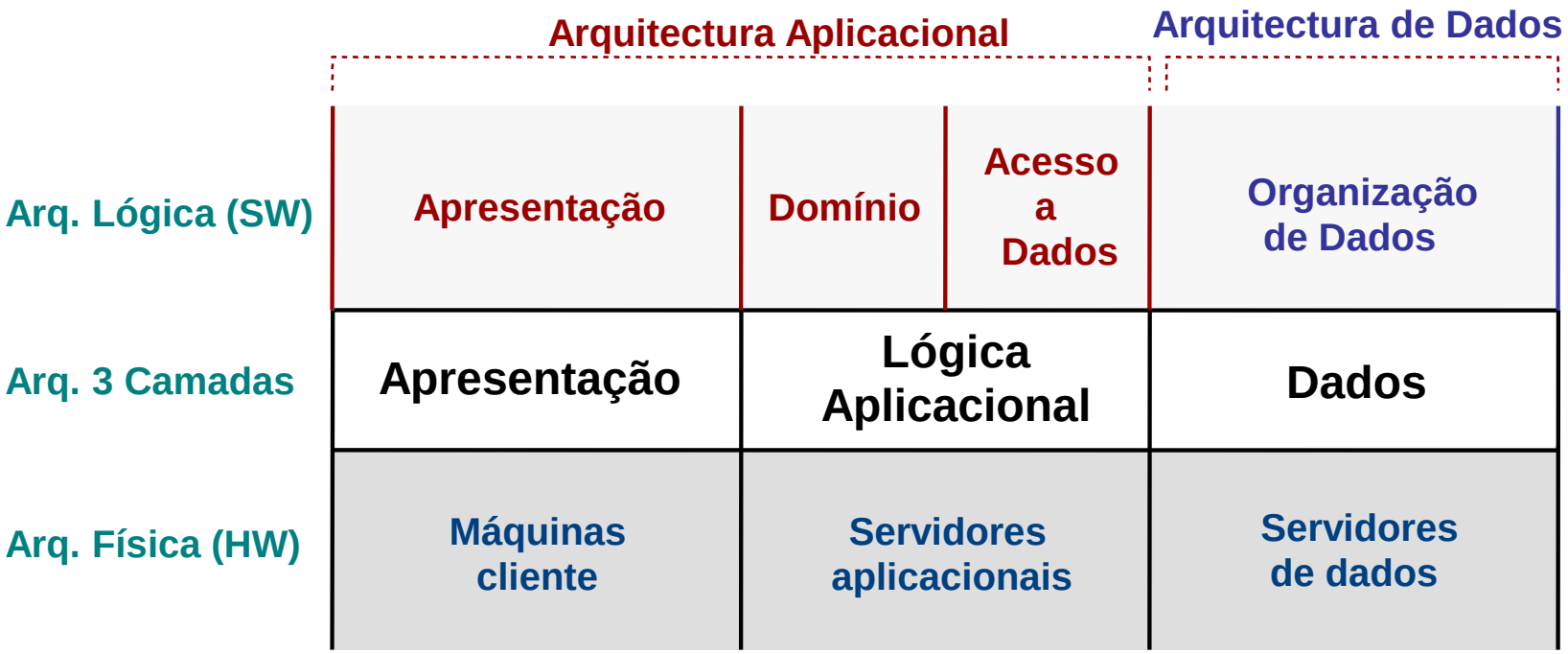


Redução de acoplamento entre níveis lógico e físico

- Facilidade de distribuição
- Facilidade de evolução

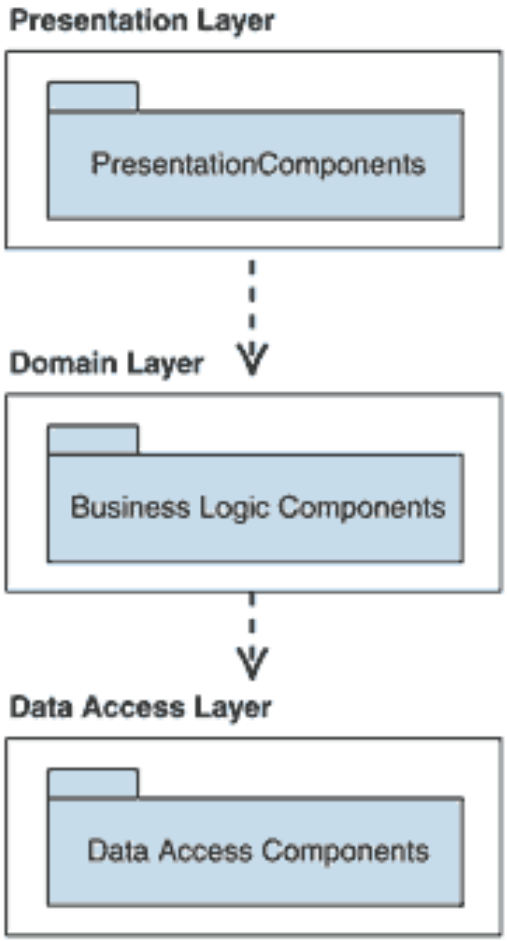
Arquitectura Física e Lógica

Níveis Físicos e Camadas Lógicas



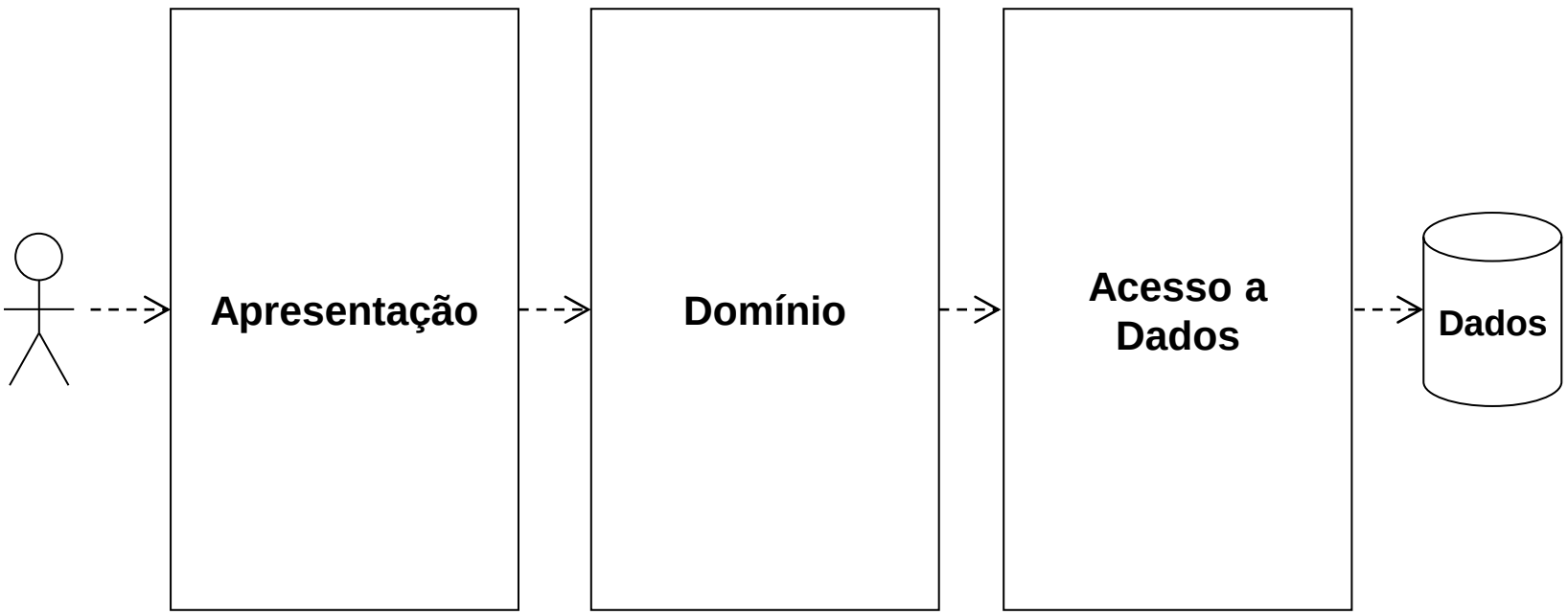
Padrões de Organização de Subsistemas

Arquitectura Lógica de 3 Camadas

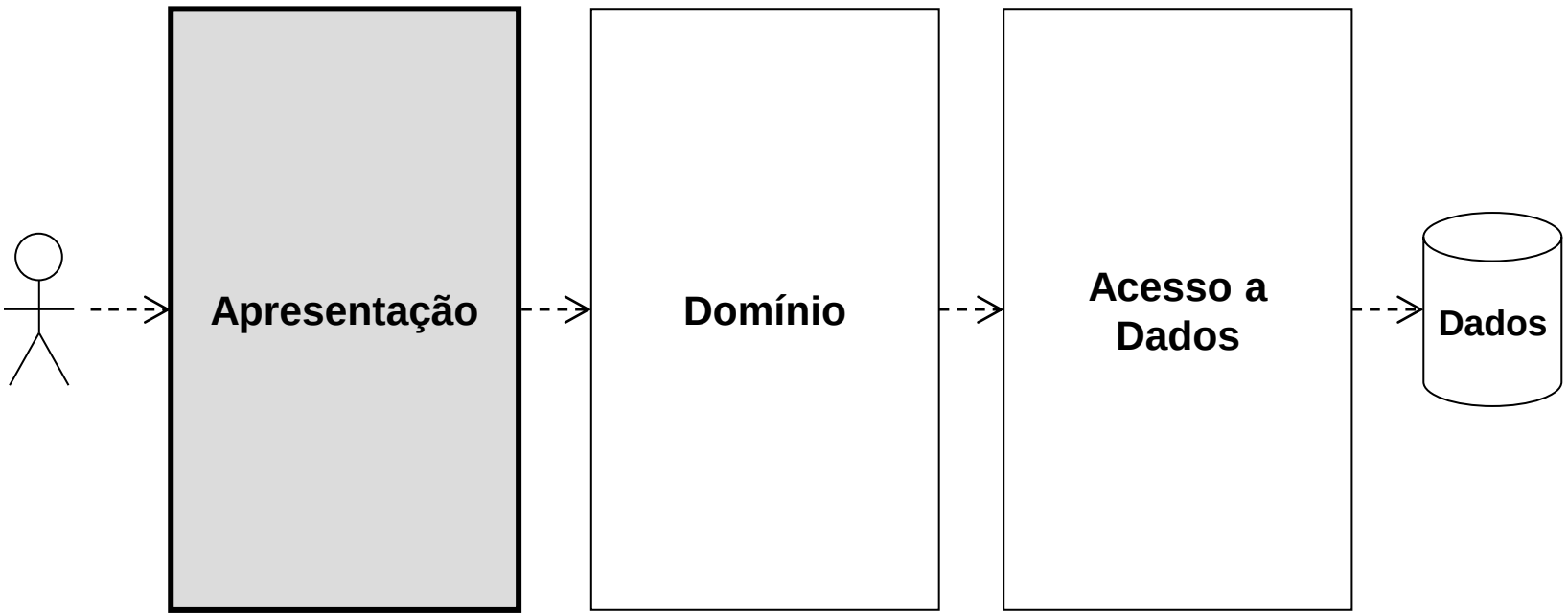


.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Arquitectura Lógica de 3 Camadas



Arquitectura de 3 Camadas



Padrão *Model-View-Controller* (MVC)

Contexto

As aplicações informáticas implicam tipicamente quer a interacção com o utilizador, quer o processamento de dados internos em função dessa interacção.

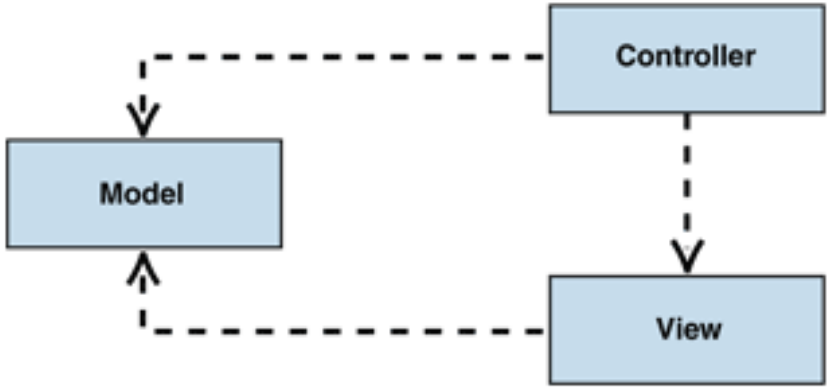
Problema

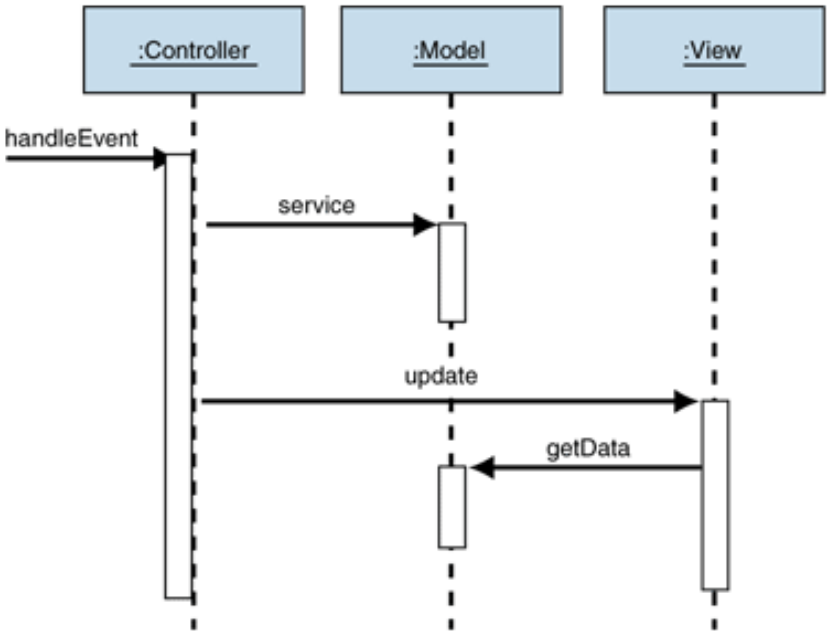
Como organizar a relação entre as classes responsáveis pela interacção com o utilizador e as classes responsáveis pelo processamento interno?

Solução

Separar o modelo de domínio, o controlo de interacção e a apresentação em três classes distintas [Burbeck,1992].

Padrão *Model-View-Controller* (MVC)



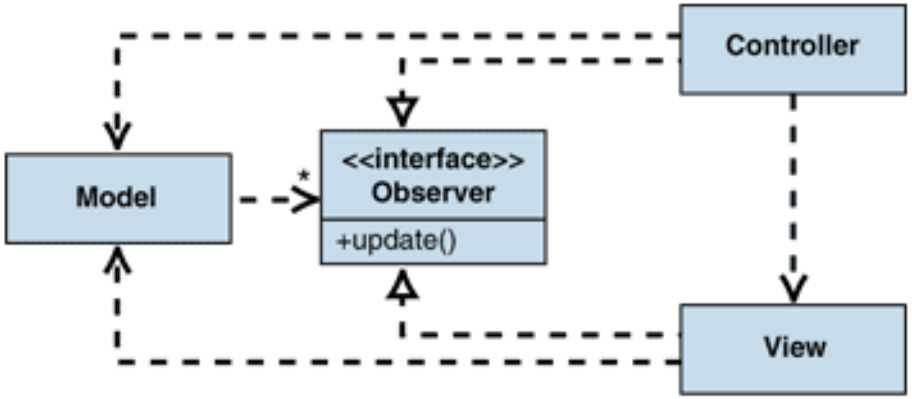


The *Model-View-Controller (MVC)* pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes [Burbeck92]:

- **Model**: the model **manages** the **behavior** and **data** of the **application domain**, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- **View**: The view **manages the display of information**.
- **Controller**: The controller **interprets the mouse and keyboard inputs from the user**, informing the model and/or the view to change as appropriate.

Padrão *Model-View-Controller* (MVC)

Modelo Activo: O *Modelo* muda de estado independentemente do controlador.



Integração com o padrão ***Observer***:
Evita que o *Modelo* dependa das *Vistas*.

Padrão Observador (*Observer*)

Contexto

Num sistema os diferentes objectos devem colaborar no sentido de atingir os objectivos globais desse sistema. Essa colaboração implica frequentemente que alguns objectos necessitem de informar outros objectos acerca de alterações no seu estado interno.

Problema

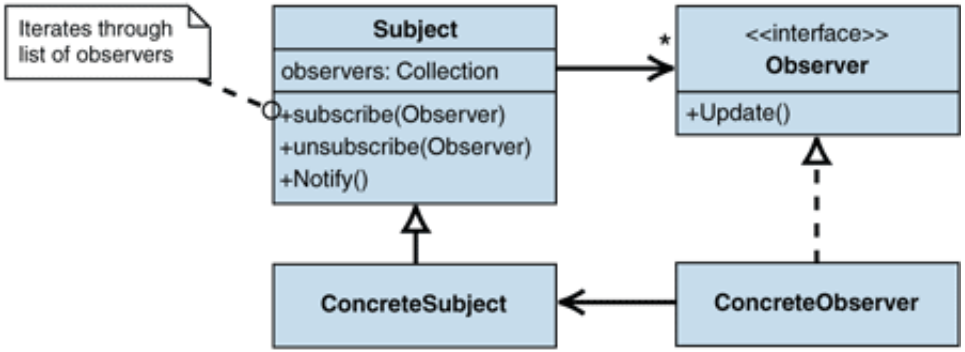
Como pode um objecto notificar alterações do seu estado interno a outros objectos sem ter uma dependência desses objectos definida *a priori* ?

Solução

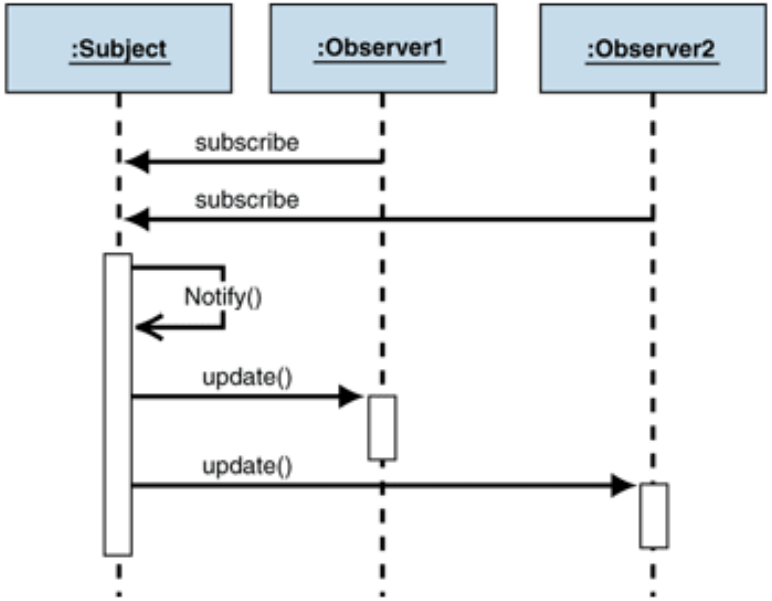
Manter informação actualizada dinamicamente, acerca dos objectos que possam estar interessados em ser notificados das alterações de estado.

Padrão Observador (*Observer*)

Definição de Estrutura

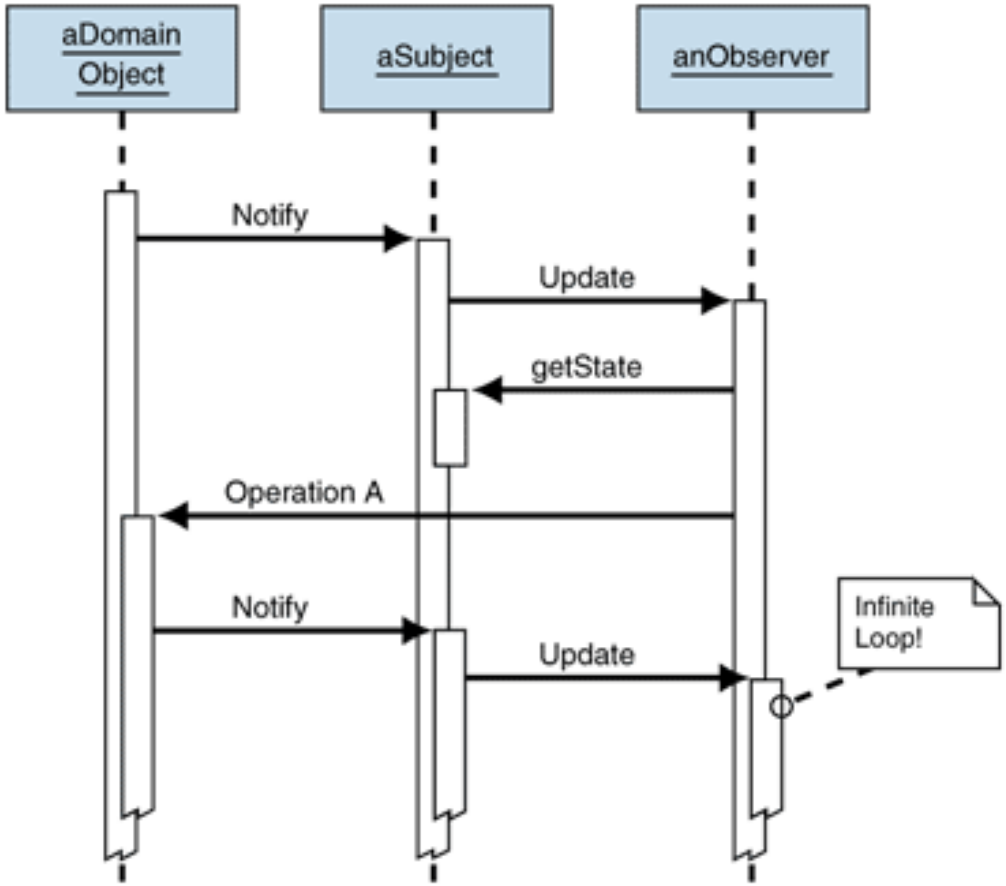


Definição de comportamento



.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

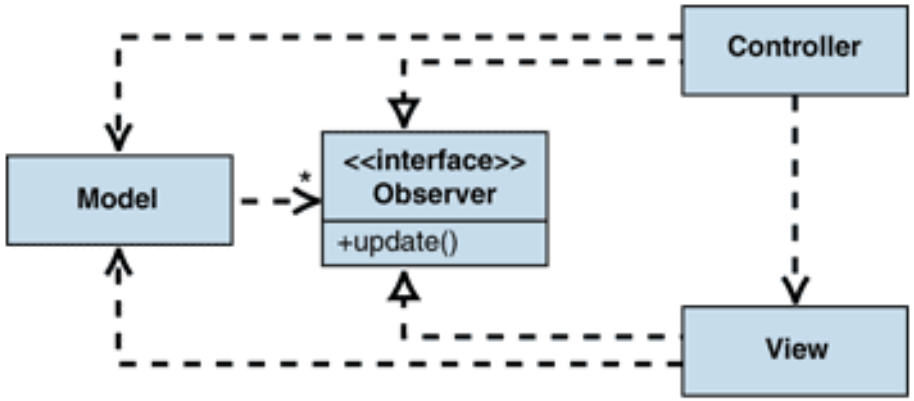
Padrão Observador (*Observer*)

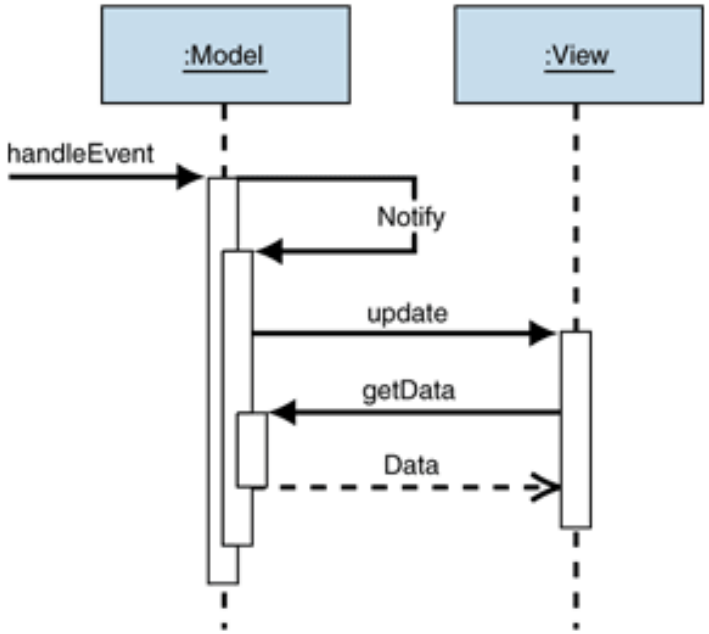


Risco: Possibilidade de ciclos de actualização encadeados!!!

Padrão *Model-View-Controller* (MVC)

Modelo Activo: O *Modelo* muda de estado independentemente do controlador.





Integração com o padrão ***Observer***:
Evita que o *Modelo* dependa das *Vistas*.

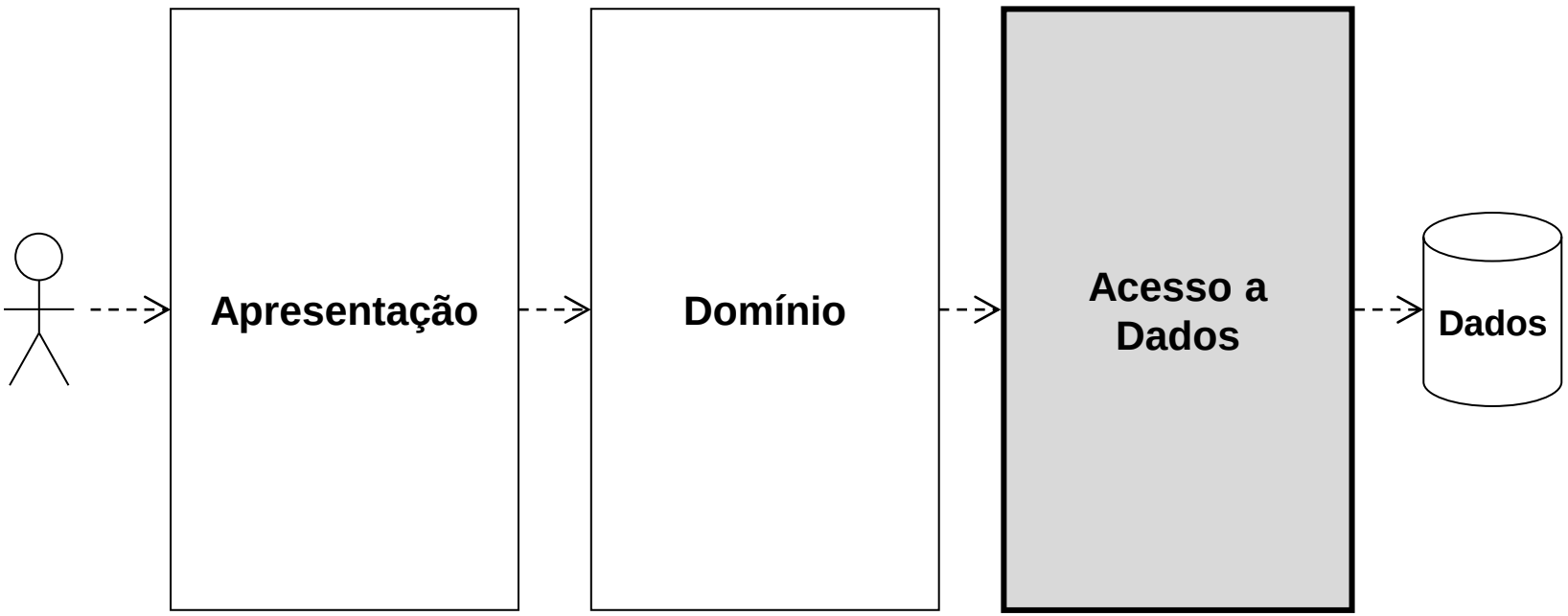
.NET Architecture Center (MSDN)
Enterprise Solution Patterns Using Microsoft .NET

Padrões de Apresentação

- **Variantes do padrão MVC**
  - **Padrão Model-View-Presenter (MVP)**
    - Apresentador (*Presenter*)
      - Intermediador
    - Vista passiva
    - Controlador de supervisão
  - **Padrão Model-View-ViewModel (MVVM)**
    - Adaptador de apresentação (*ViewModel*)
    - Actualização automática das vistas
      - *Data binding*

Organização do Acesso a Dados

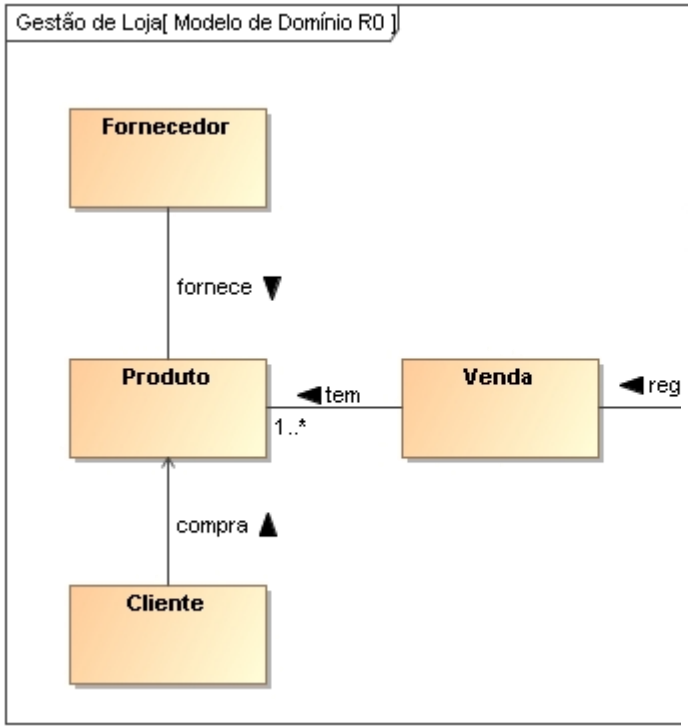
Arquitectura Lógica de 3 Camadas



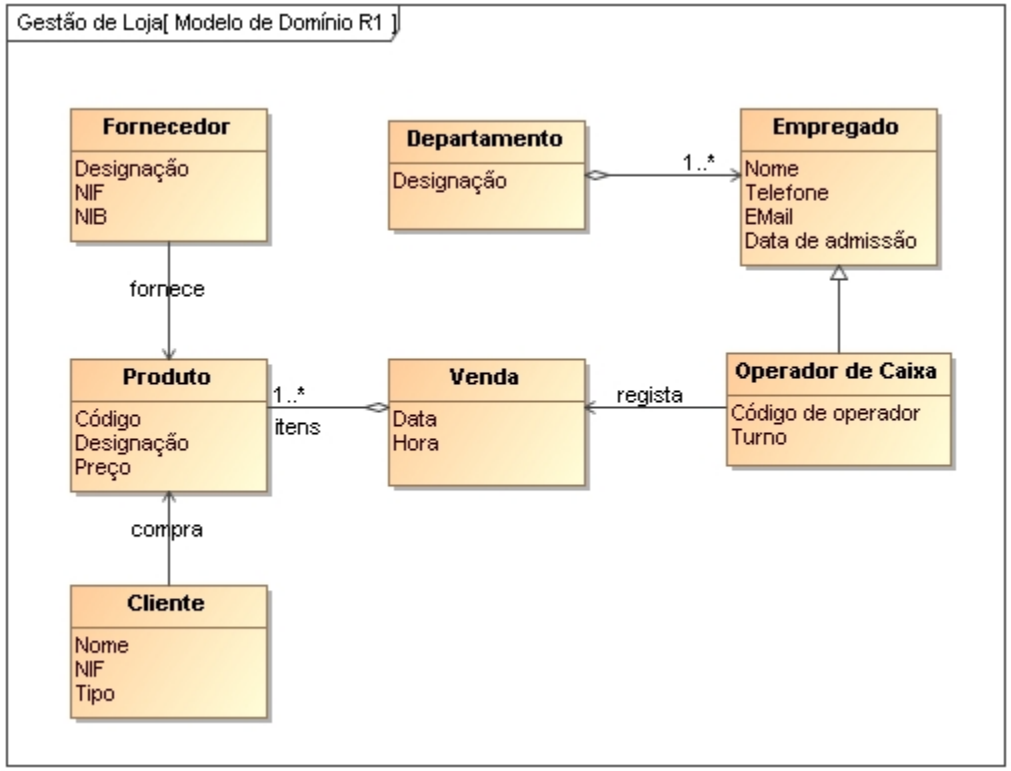
Arquitectura de Dados: Exemplo

Modelo Conceptual (Modelo de Domínio)

Início



Refinamento



Arquitectura de Dados: Exemplo

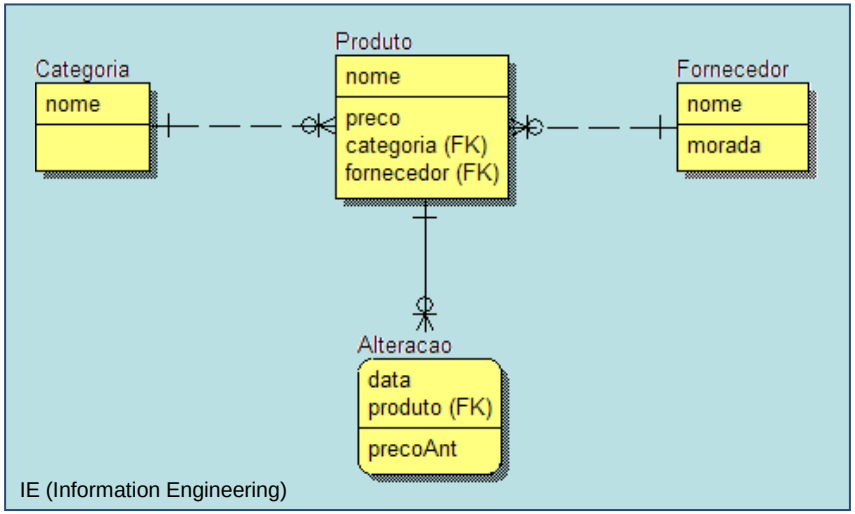
Modelo Lógico (e.g. Modelo Relacional)

Produto(nome, preco, categoria [FK], fornecedor [FK])

Categoria(nome)

Fornecedor(nome, morada)

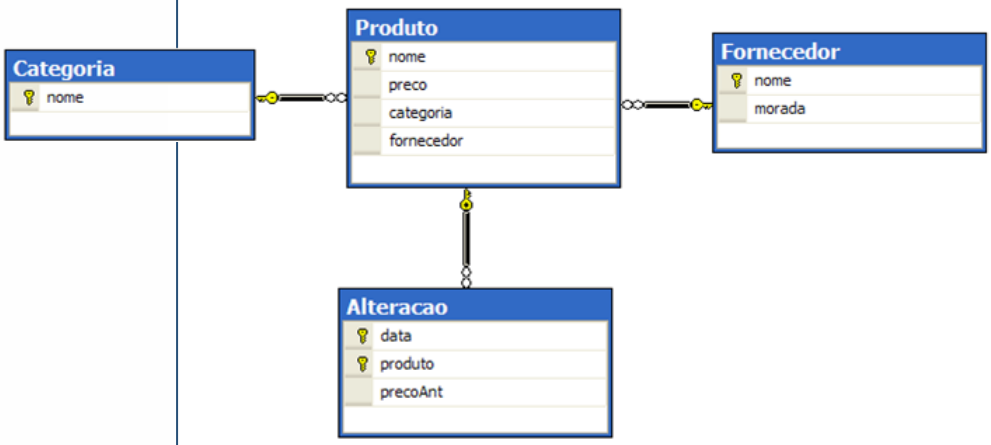
Alteração(produto [FK], data, preco)



Arquitectura de Dados: Exemplo

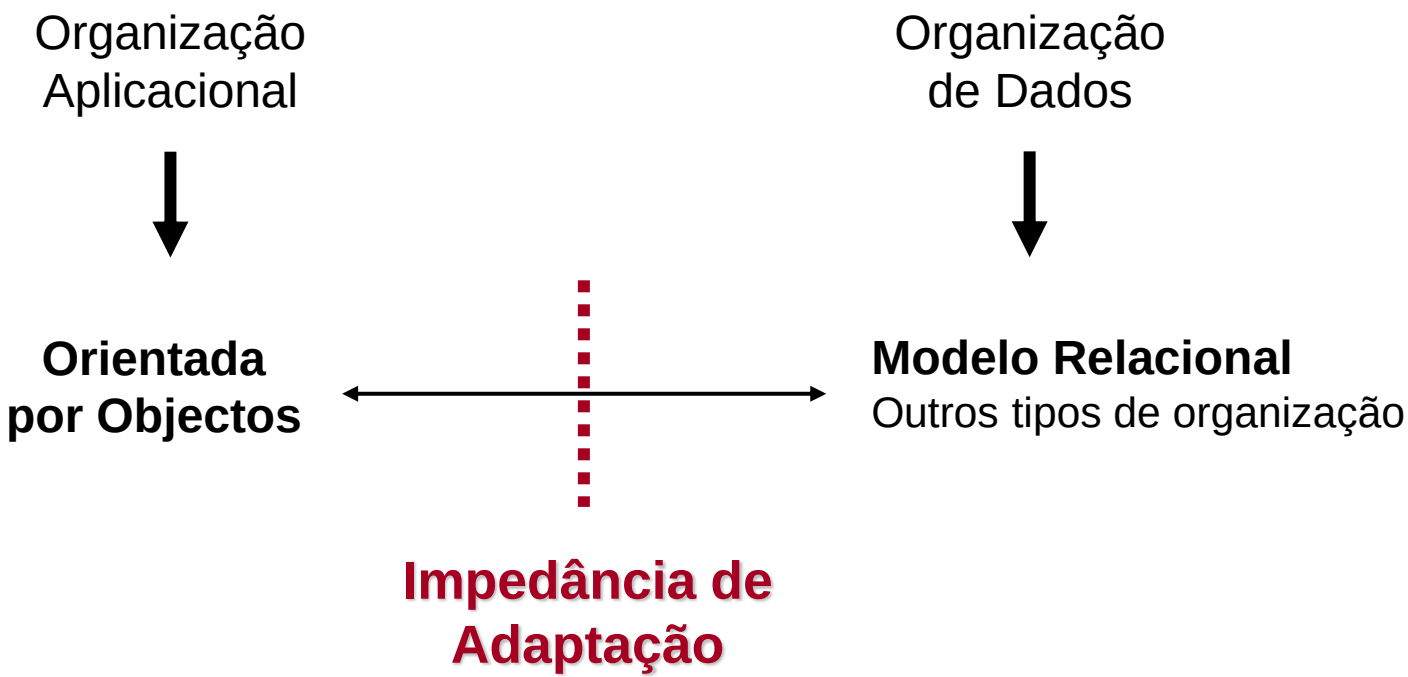
Modelo Detalhado (Modelo Físico)

```
CREATE TABLE Categoria (
       nome varchar(20) NOT NULL,
       PRIMARY KEY (nome)
)

CREATE TABLE Fornecedor (
       nome varchar(20) NOT NULL,
       morada varchar(20) NULL,
       PRIMARY KEY (nome)
)

CREATE TABLE Produto (
       nome varchar(20) NOT NULL,
       preco int NOT NULL,
       categoria varchar(20) NOT NULL,
       fornecedor varchar(20) NOT NULL,
       PRIMARY KEY (nome),
       FOREIGN KEY (categoria) REFERENCES Categoria,
       FOREIGN KEY (fornecedor) REFERENCES Fornecedor
)

CREATE TABLE Alteracao (
       data datetime NOT NULL,
       produto varchar(20) NOT NULL,
       precoAnt int NOT NULL,
       PRIMARY KEY (data, produto),
       FOREIGN KEY (produto) REFERENCES Produto
)
```



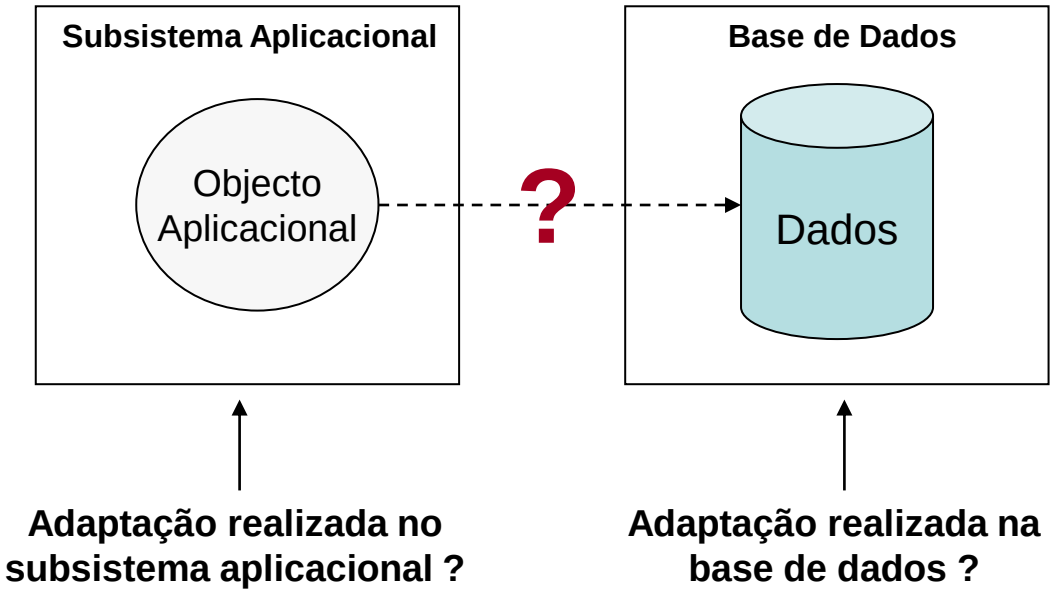
Arquitectura de Dados

Relação entre *arquitectura aplicacional* e *arquitectura de dados*

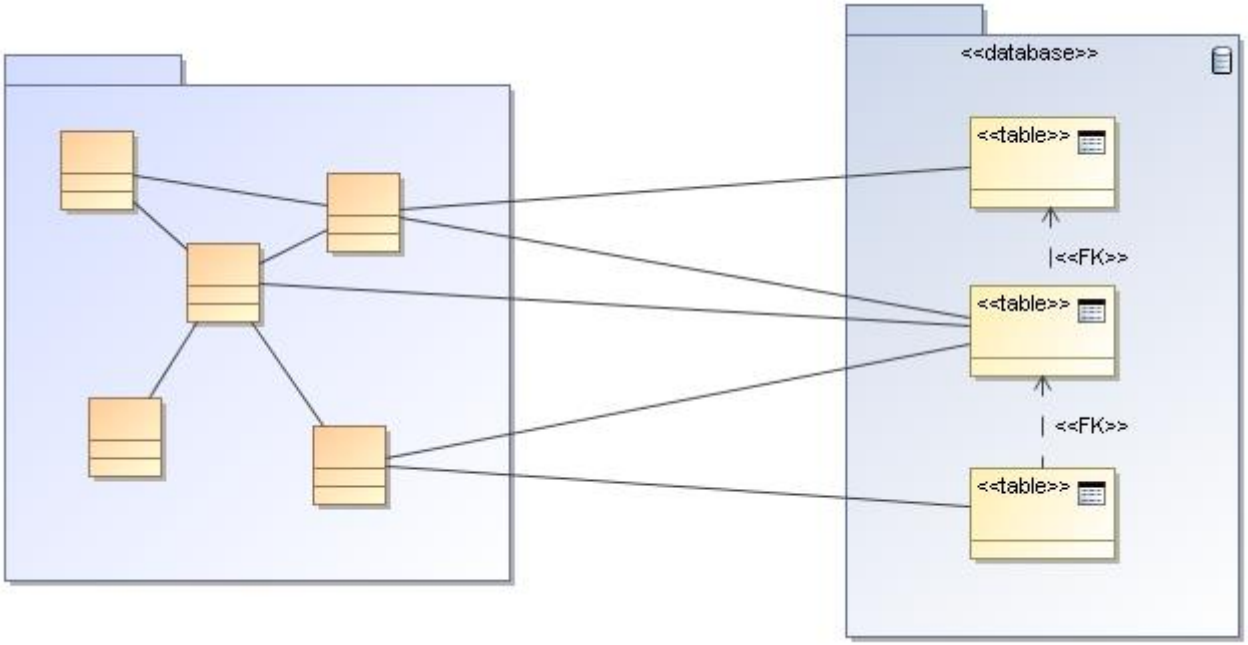


Arquitectura de Dados

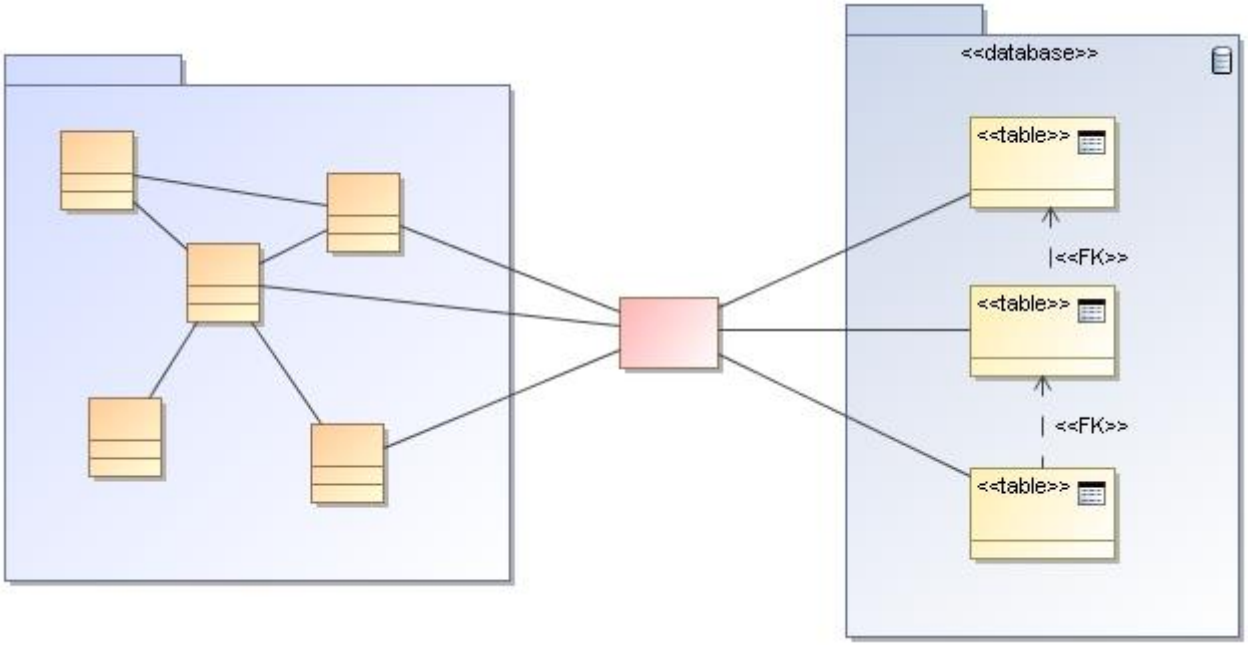
Relação entre *arquitectura aplicacional* e *arquitectura de dados*



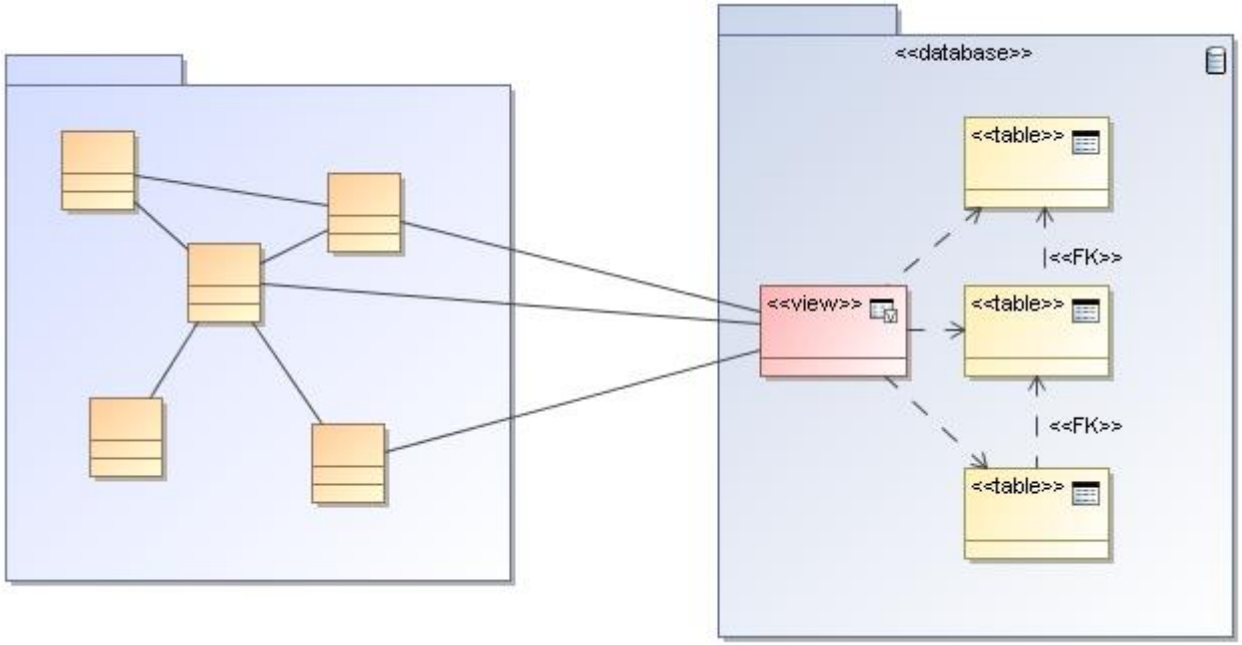
Organização do Acesso a Dados



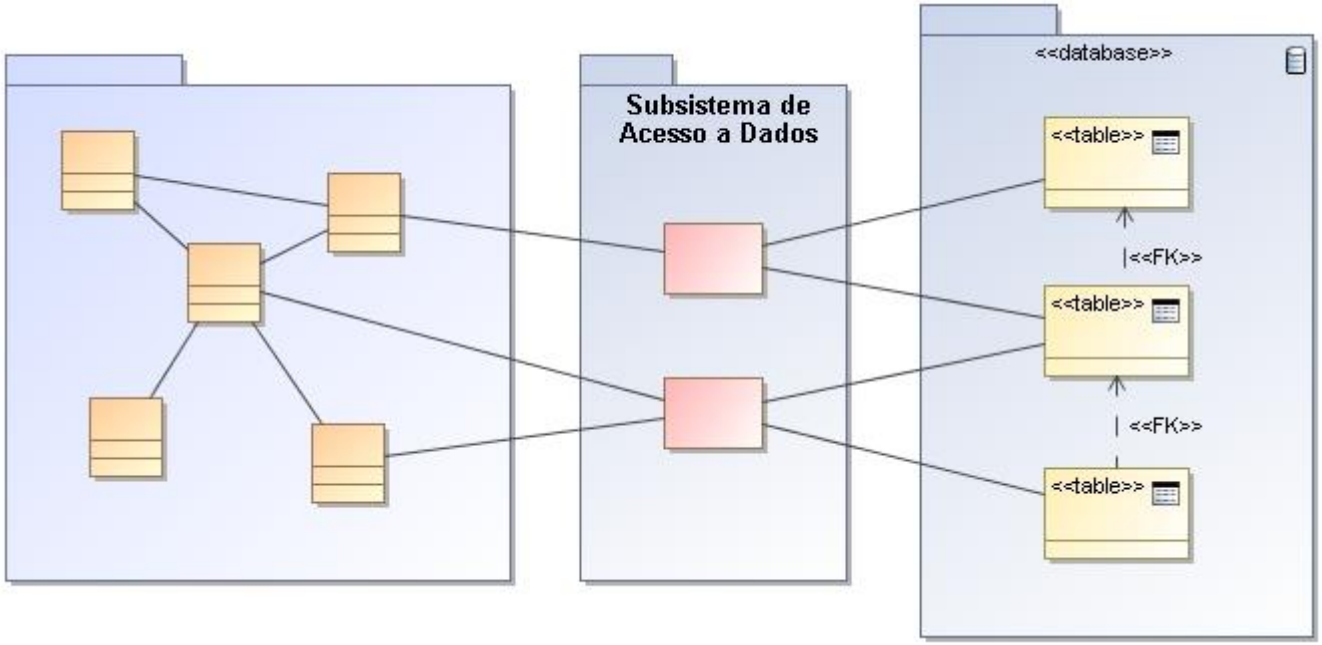
Organização do Acesso a Dados



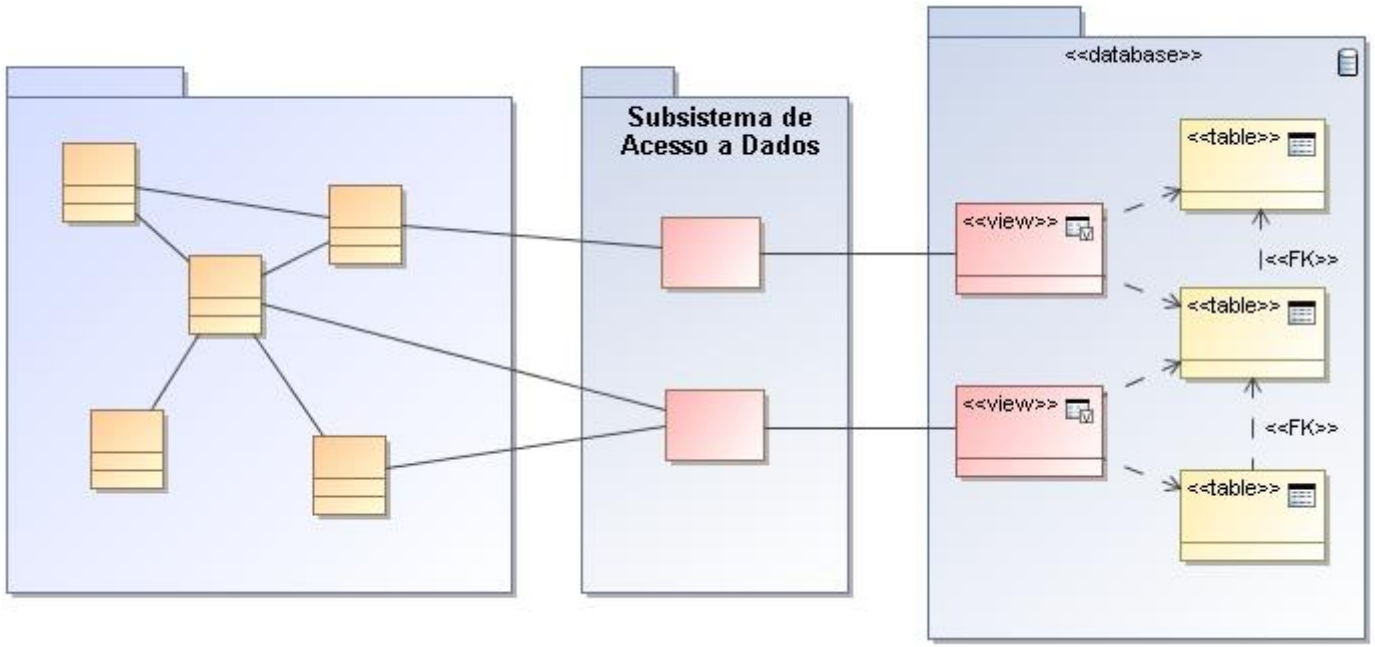
Organização do Acesso a Dados



Organização do Acesso a Dados

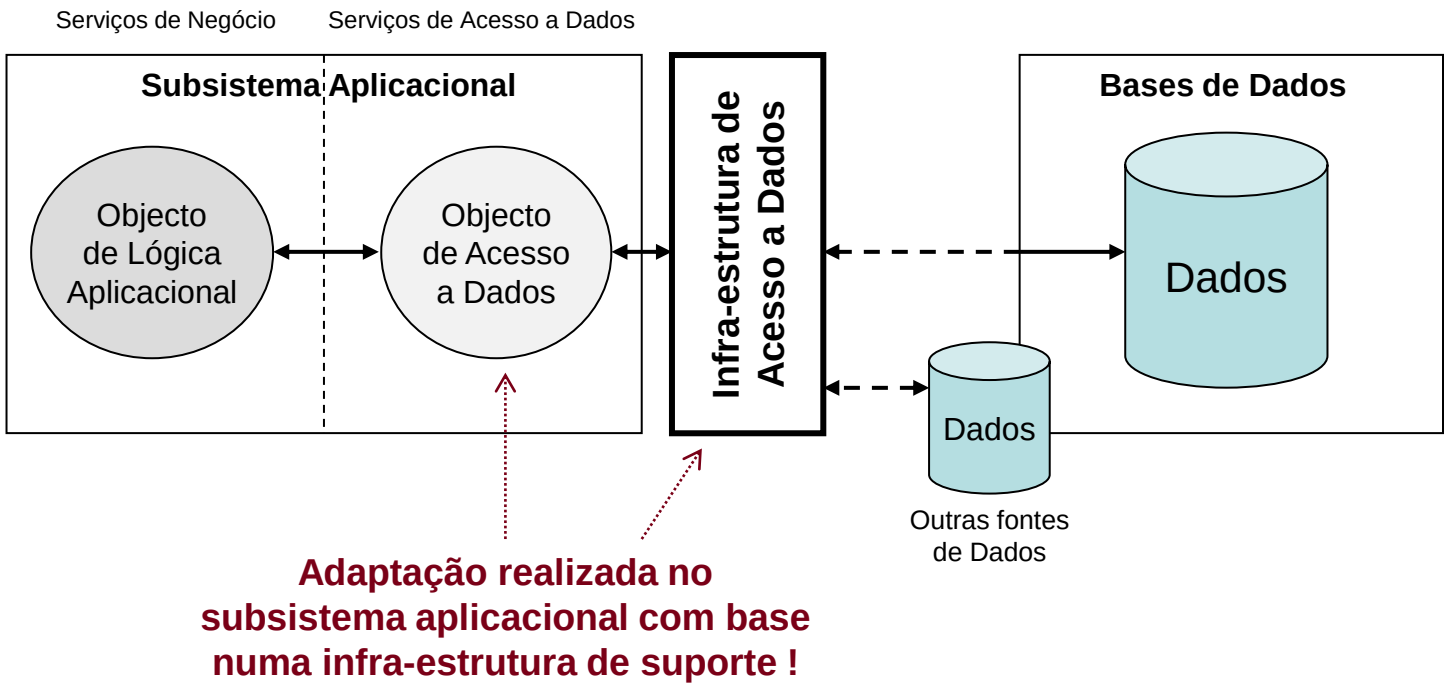


Organização do Acesso a Dados



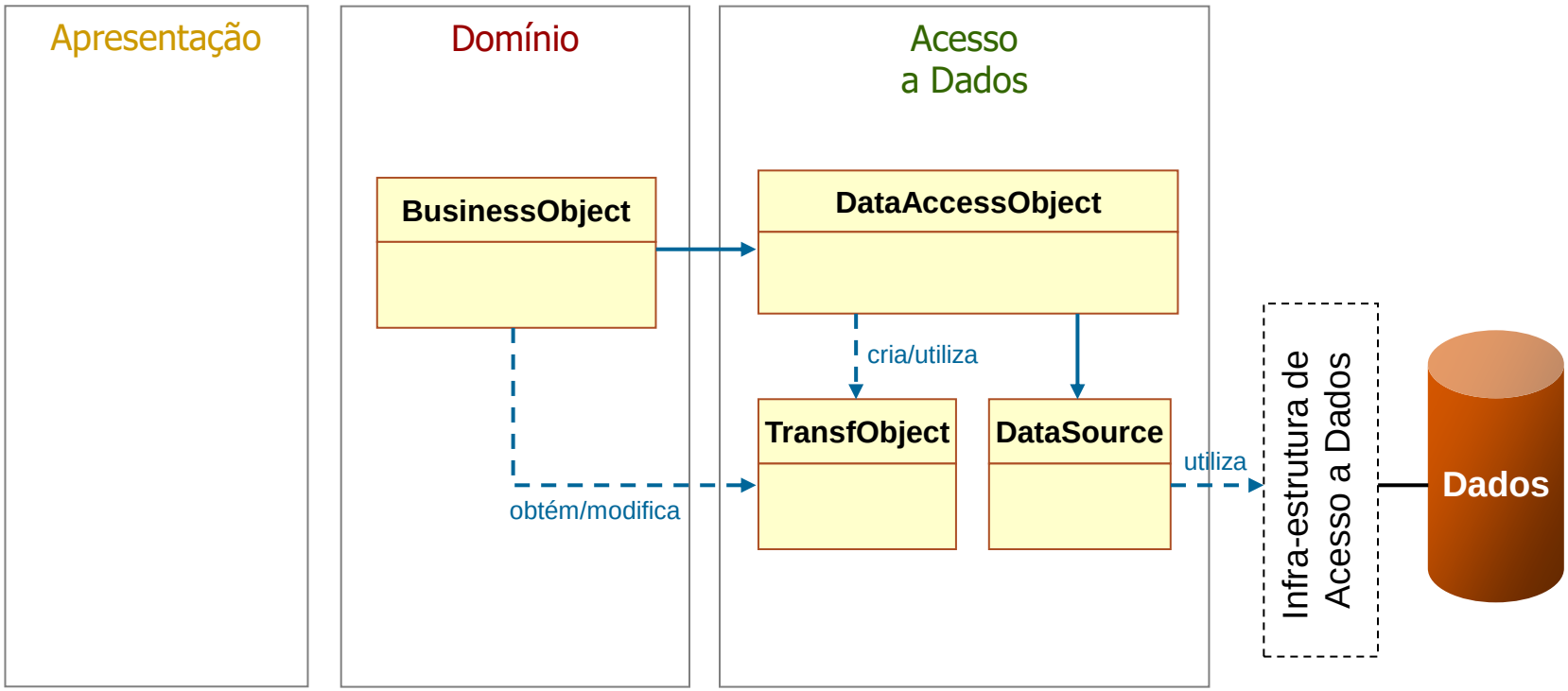
Arquitectura de Acesso a Dados

Relação entre arquitectura aplicacional e arquitectura de dados



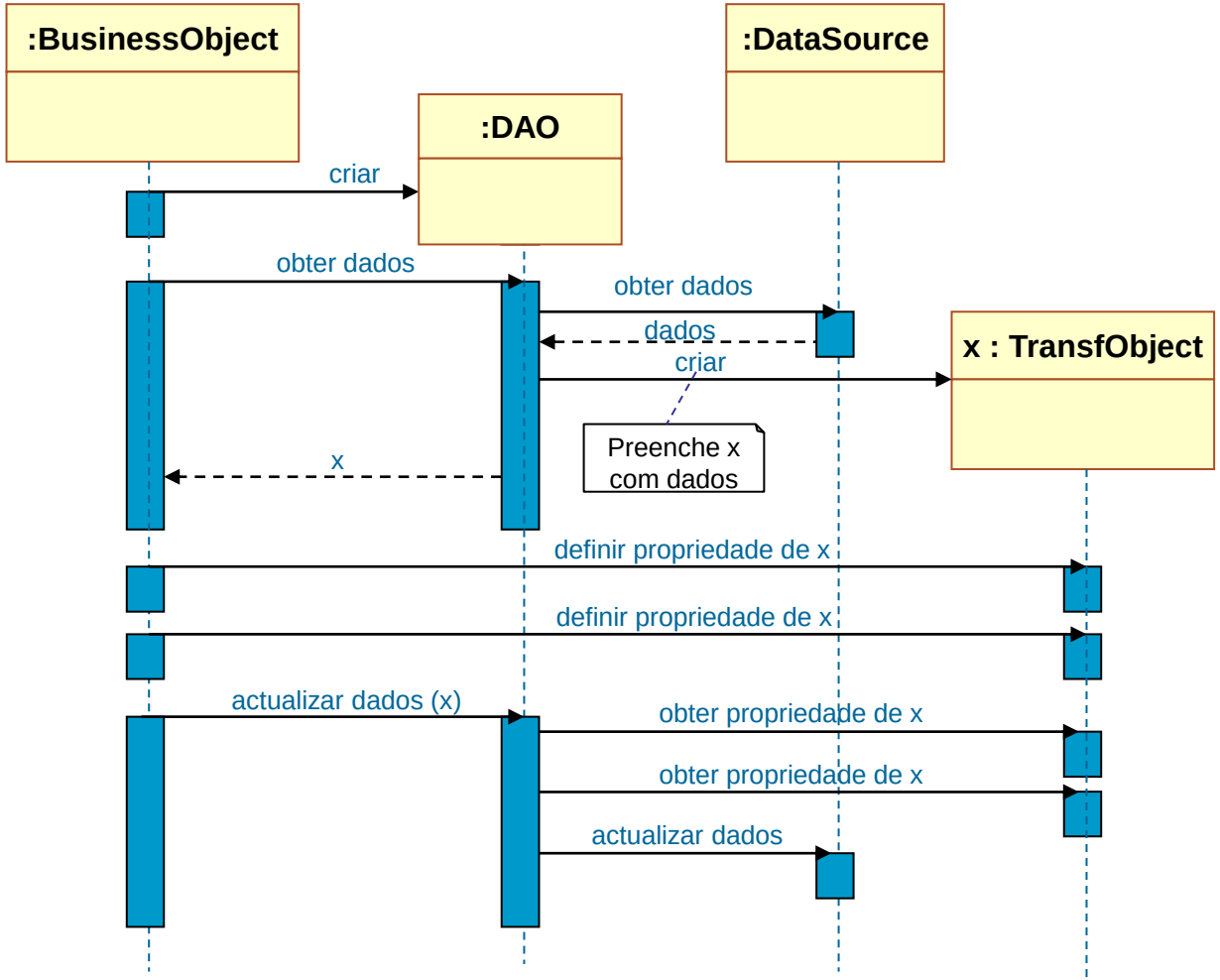
Padrões de Acesso a Dados

- Padrão DAO (*Data Access Objects*)



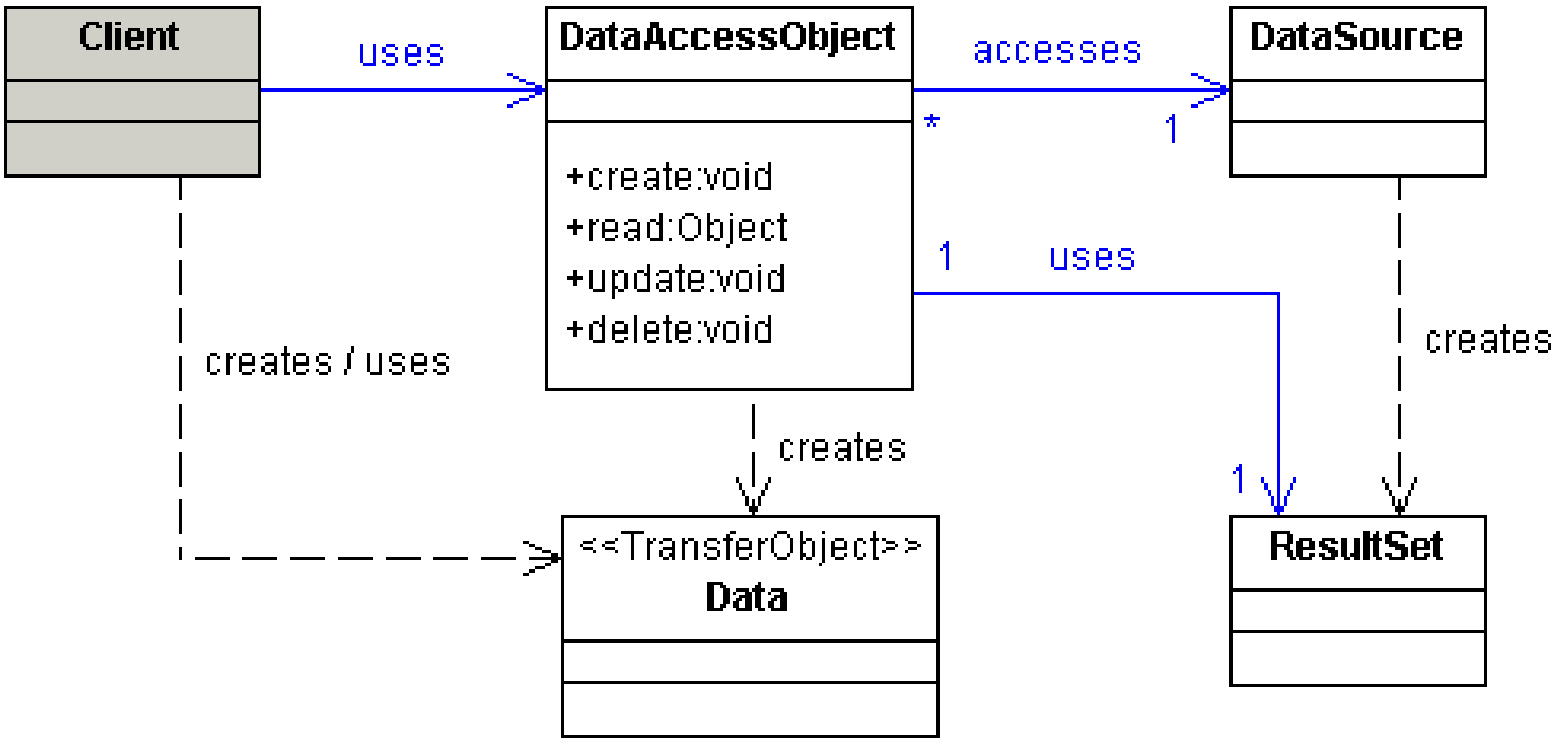
Padrões de Acesso a Dados

- Padrão DAO (*Data Access Objects*)

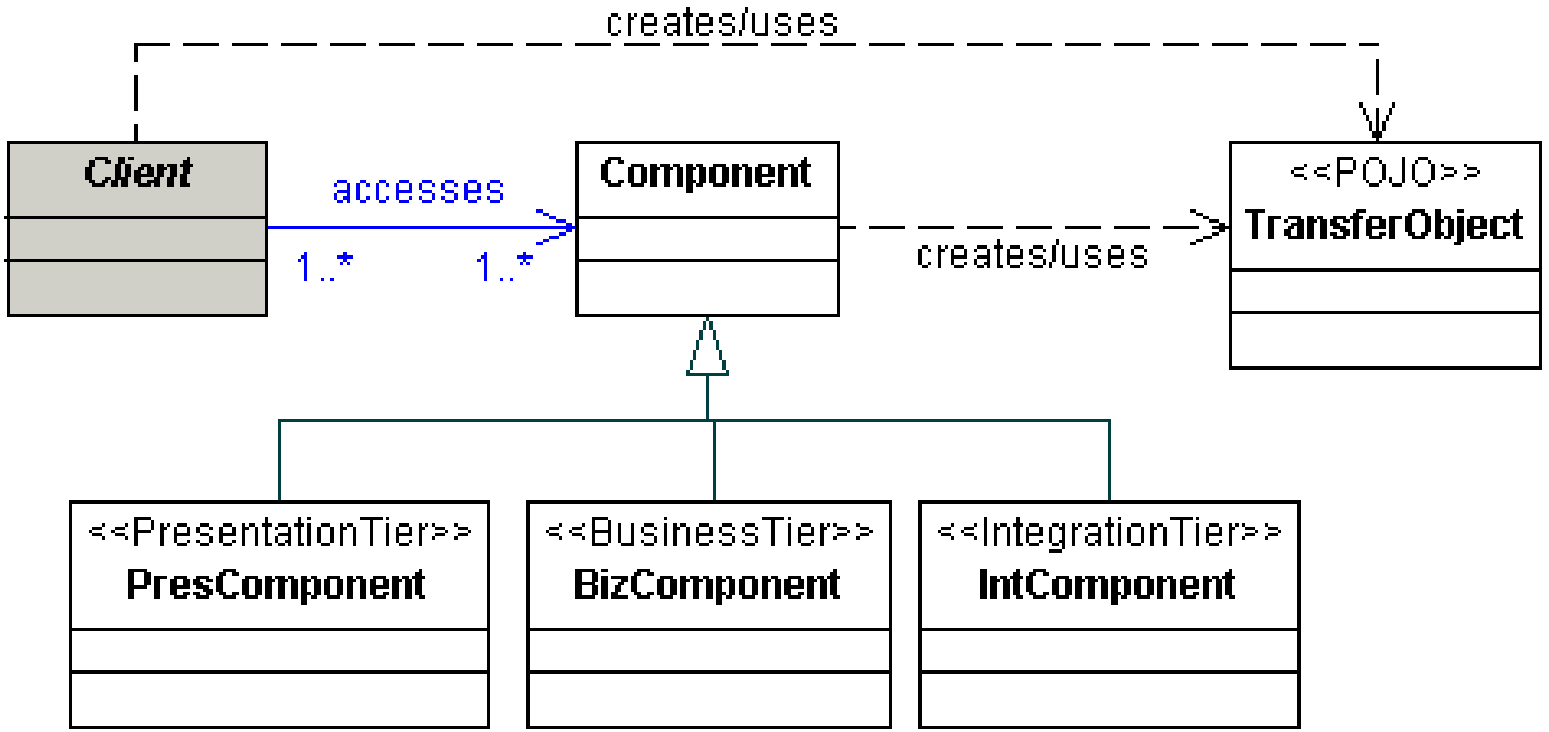


Padrão *Data Access Object*

Modo de operação *CRUD*



Padrão *Transfer Object*



Data Transfer Object

- ***Data Transfer Object* (DTO)**
  - Objecto que transporta dados entre subsistemas
  - Forma de reduzir o custo de comunicação entre subsistemas reduzindo o número de interacções, através da utilização de um DTO que agrega os dados de modo a ser necessário apenas uma interacção
- ***Data Transfer Objects* vs. *Business Objects***
  - Um DTO não tem qualquer comportamento excepto para acesso aos seus próprios dados
    - Tipicamente imutável
  - Um DTO não deve conter qualquer lógica de domínio

Bibliografia

[Pressman, 2003]

R. Pressman, *Software Engineering: a Practitioner's Approach*, McGraw-Hill, 2003.

[Gamma et al., 1995]

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[Shaw & Garlan, 1996]

M. Shaw, D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[Larman, 2004]

C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, Prentice Hall, 2004.

[Parnas, 1972]

D. Parnas, *On the Criteria to Be Used in Decomposing Systems into Modules*, Communications of the ACM 15-12, 1968.

[Kruchten, 1995]

F. Kruchten, *Architectural Blueprints - The "4+1" View Model of Software Architecture*, IEEE Software, 12-6, 1995.

[Burbeck, 1992]

S. Burbeck; *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*, http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html,1992

[Booch, 2004]

G. Booch, *Software Architecture*, IBM, 2004.