Model-View-Controller

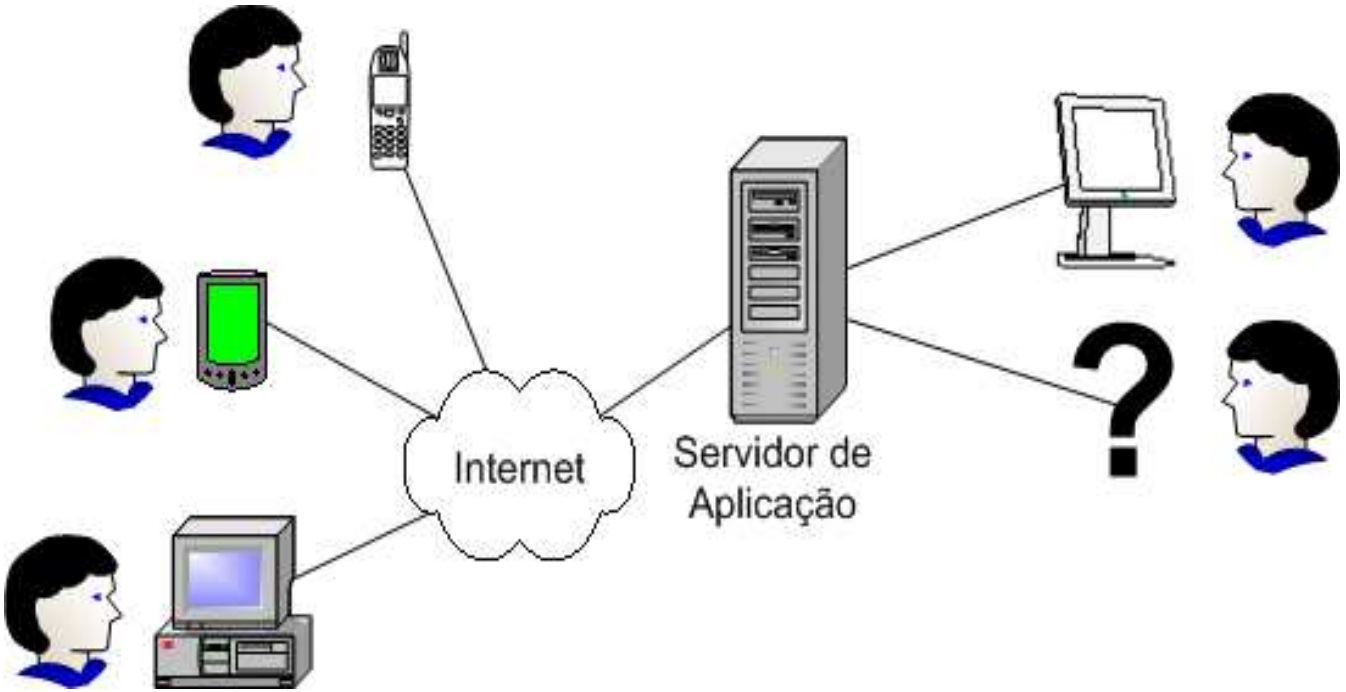
ALUNO:

RONI FABIO BANASZEWSKI

Objetivo

- Separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control)
- A idéia é permitir que uma mesma lógica de negócios possa ser acessada e visualizada através de várias interfaces.
- Na arquitetura MVC, a lógica de negócios (chamaremos de Modelo) não sabe de quantas nem quais interfaces com o usuário estão exibindo seu estado.
- Com as diversas possibilidades de interfaces que conhecemos hoje, a MVC é uma ferramenta indispensável para desenvolvermos sistemas

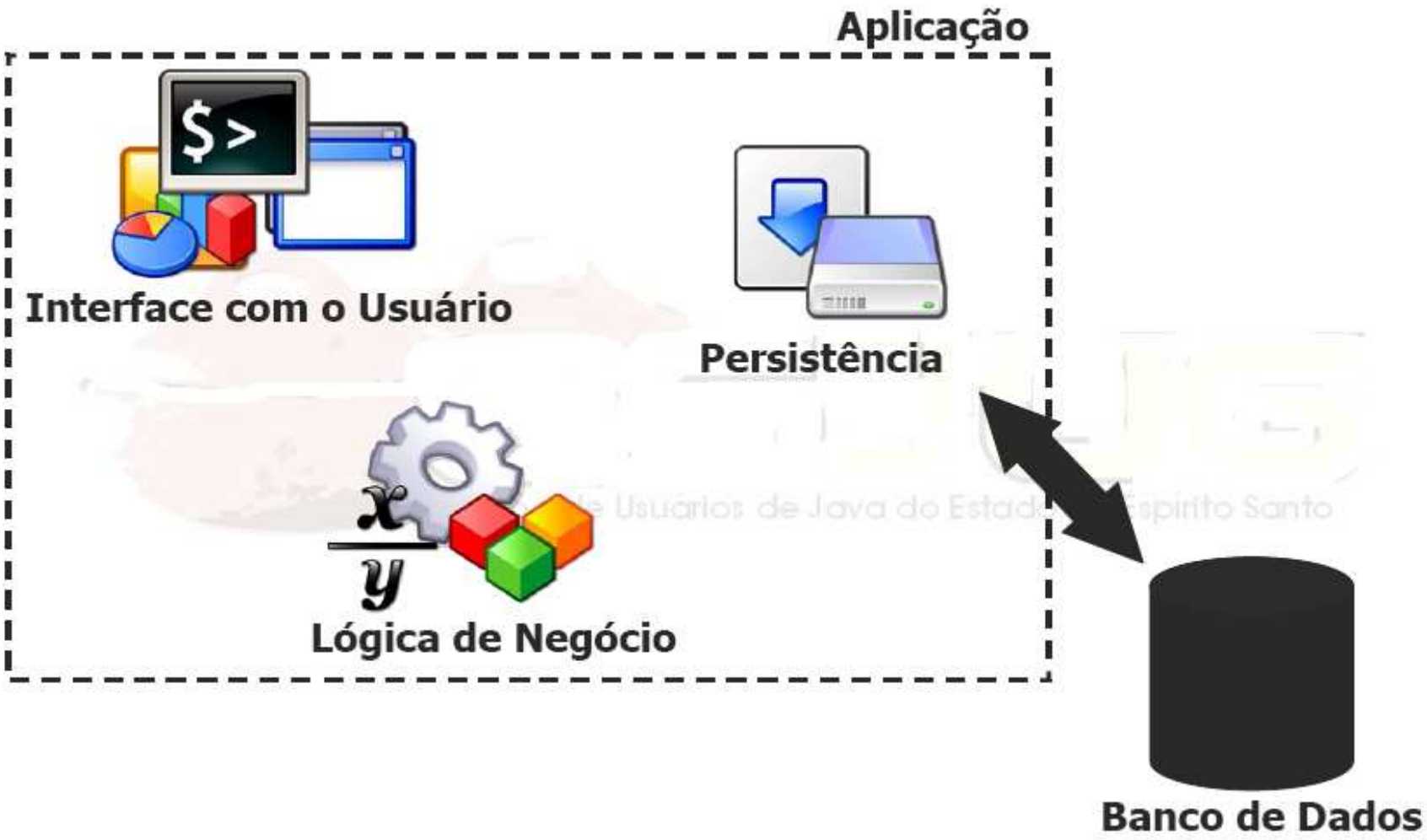
Diferentes interfaces de acesso



Histórico

- Desenvolvido pelo Xerox PARC para o Smalltalk, em 1978;
- Objetivo:
  - mapear entrada-processamento-saída em GUIs para OO: controle-modelo-visão;
- Usado para:
  - Criação de componentes GUI reutilizáveis (propósito inicial);
  - Estruturação da aplicações (pós-Web).

Uma única camada



Exemplo

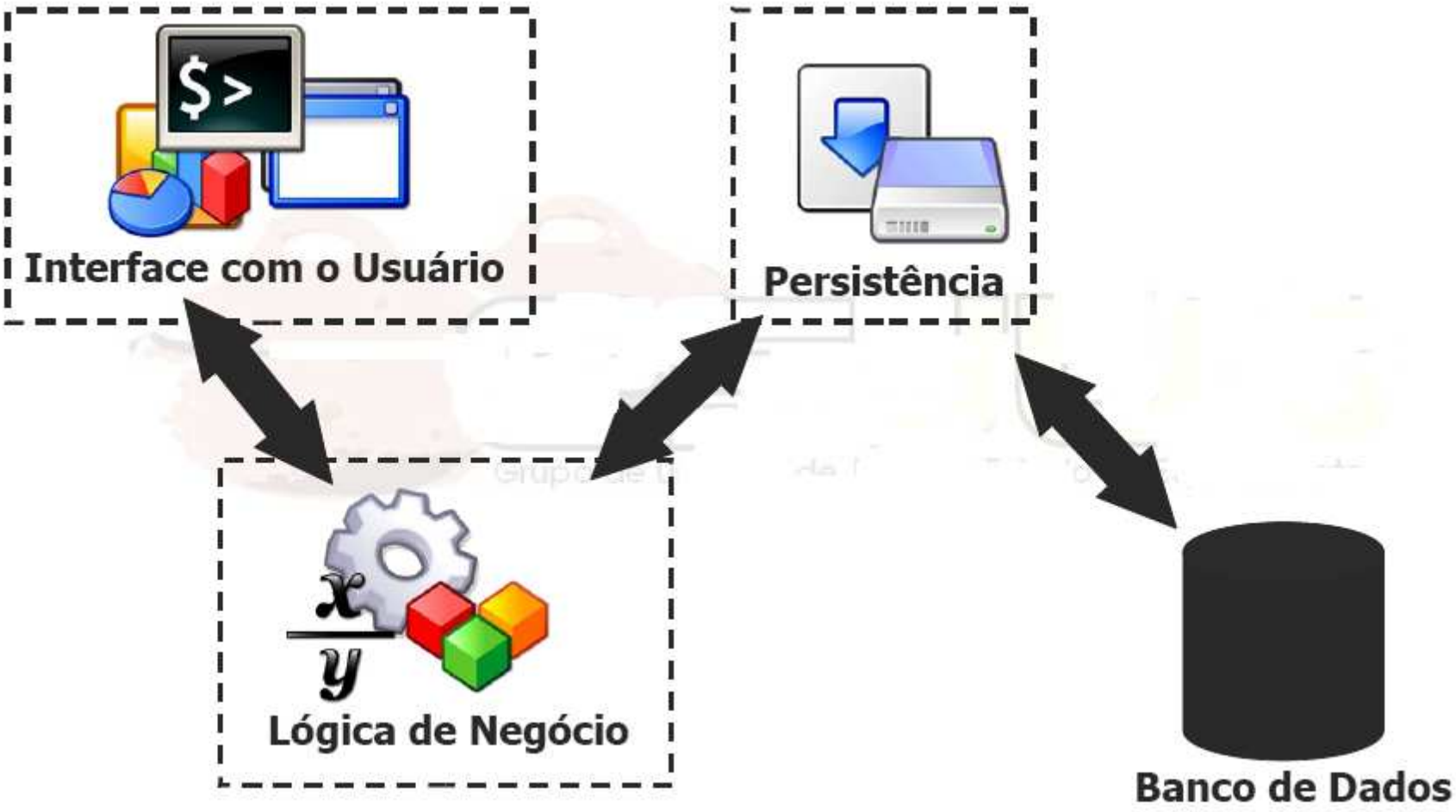
```
<html><body>

<%
sql = "SELECT * FROM ...";
rset = stmt.executeQuery(sql);

/* ... */

sql = "UPDATE Tabela SET ...";
stmt.executeUpdate(sql);
%>

</body></html>
```

Execução de queries ao banco de dados diretamente da página web.

Conseqüências: Uma camada

- Sem complicações para desenvolver;
- Difícil manutenção:
  - Código desorganizado;
  - Difícil depuração;
  - Dificulta o trabalho em equipe (Web Designer, desenvolvedor...).
- Difícil reuso.

Sistema em várias camadas



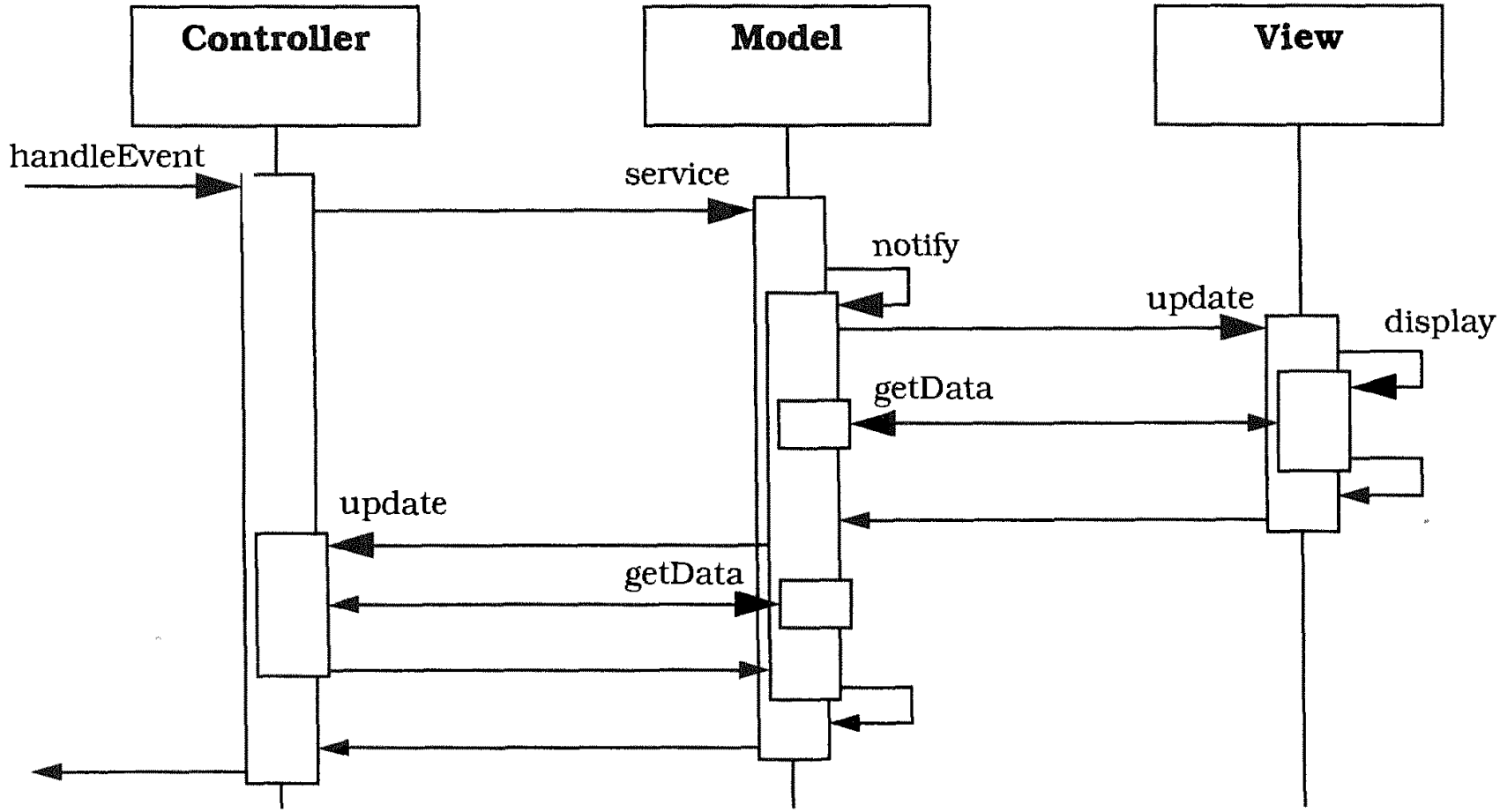
Conseqüências: Múltiplas camadas

- Maior complexidade no desenvolvimento;
- Manutenção mais simples:
  - Código organizado;
  - Depuração isolada de camadas;
  - Alterações numa camada não afetam outras.
- Facilita o reuso
- Simplifica a inclusão de um novo elemento de visão (ex.: cliente)
- Possibilita desenvolvimento das camadas em paralelo, se forem bem definidas.

Conseqüências: Múltiplas camadas

- Requer análise mais aprofundada (mais tempo);
- Requer pessoal especializado;
- Não aconselhável para aplicações pequenas (custo x benefício não compensa).
- Se tivermos muitas visões e o modelo for atualizado com muita freqüência, a performance do sistema pode ser abalada;
- Se o padrão não for implementado com cuidado, podemos ter casos como o envio de atualizações para visões que estão minimizadas ou fora do campo de visão de usuário;
- Ineficiência: uma visão pode ter que fazer inúmeras chamadas ao modelo, dependendo de sua interface

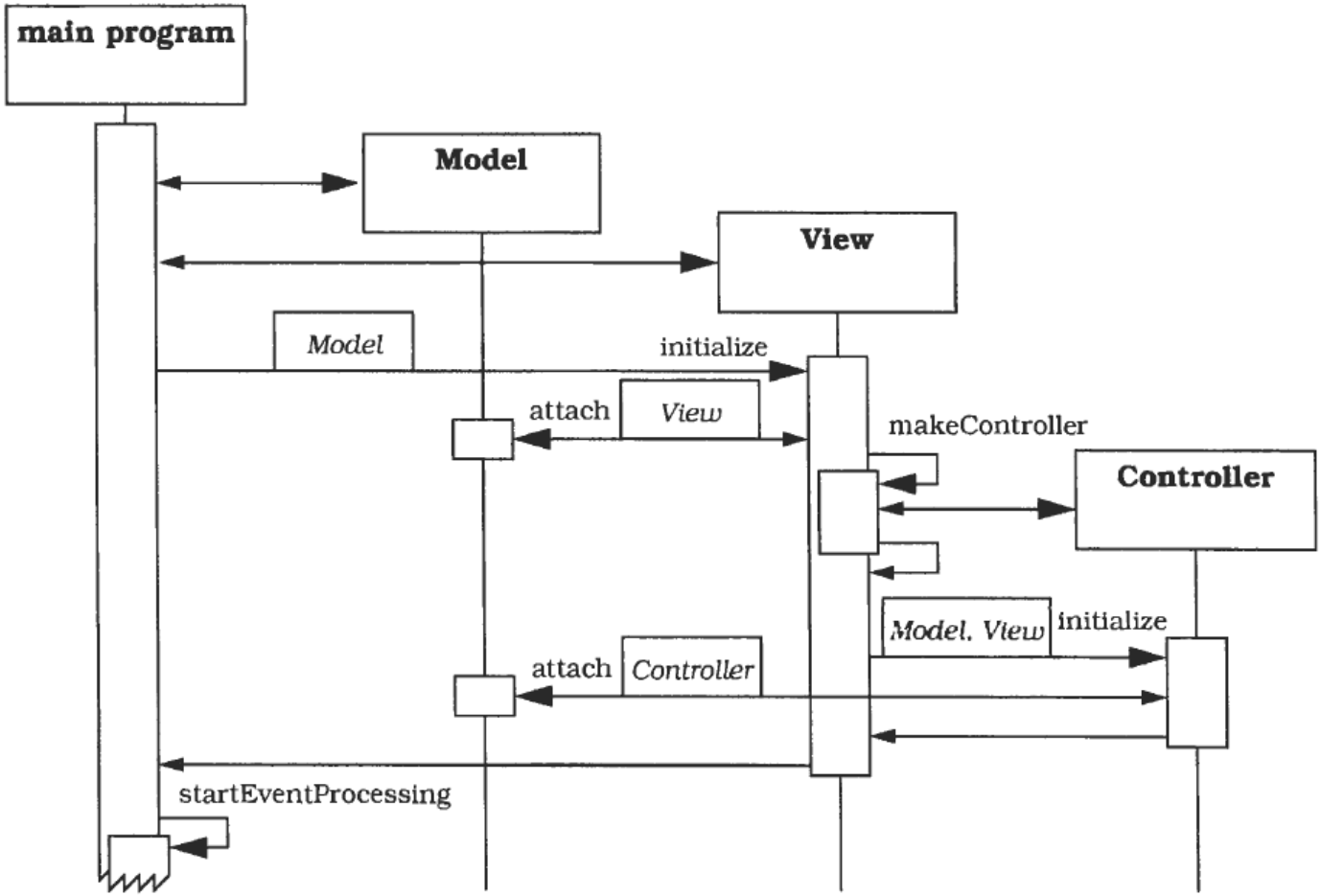
Colaboração



Descrição da Colaboração

- O controle recebe um evento da GUI e invoca um método do modelo
- O modelo computa e muda o estado de algum atributo
- O modelo notifica todos os controles e visões registradas
- Cada visão recupera os dados do modelo e se atualiza
- Cada controle também atualiza seu estado, como a habilitação de um menu...
- O controle recupera o controle da aplicação

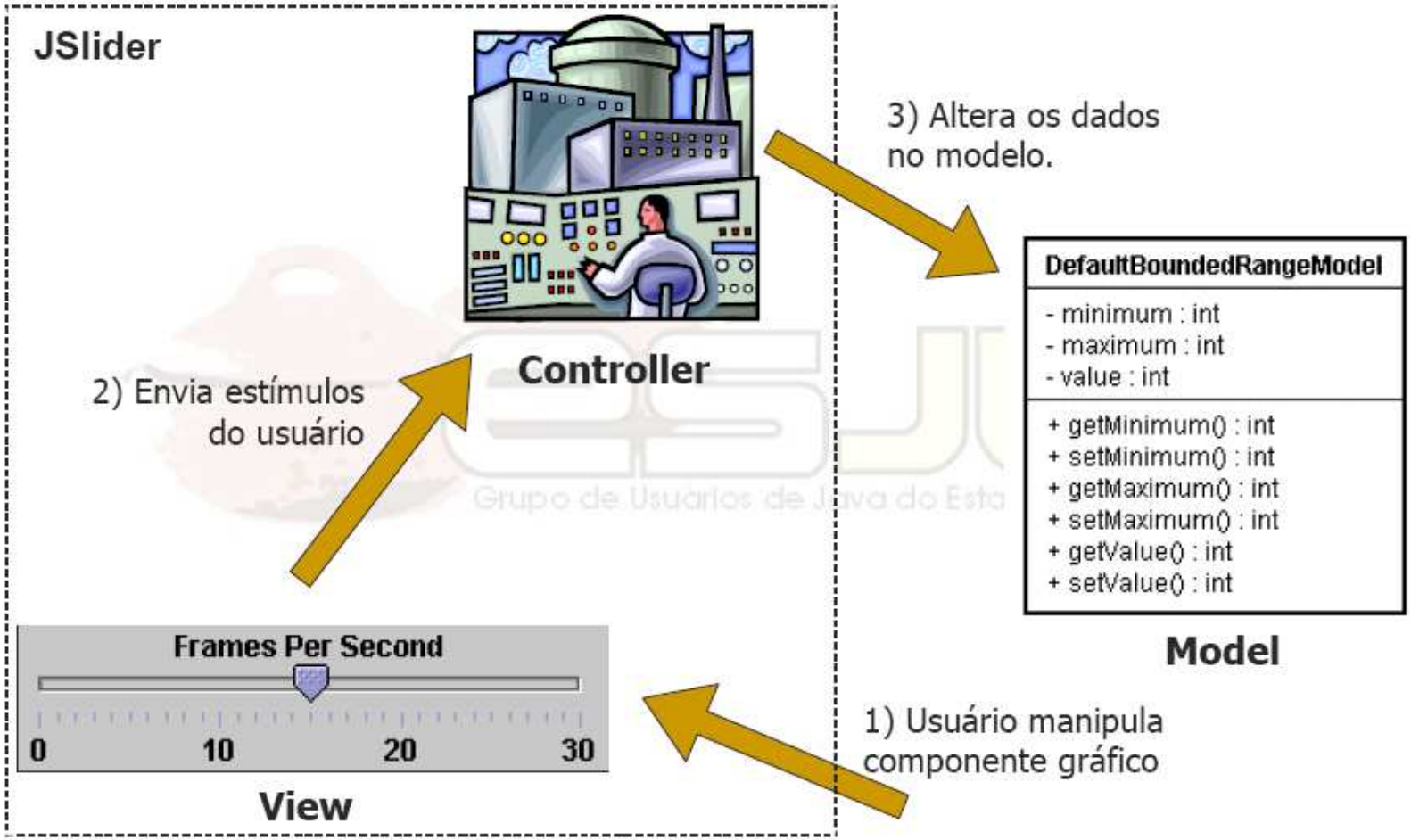
Colaboração



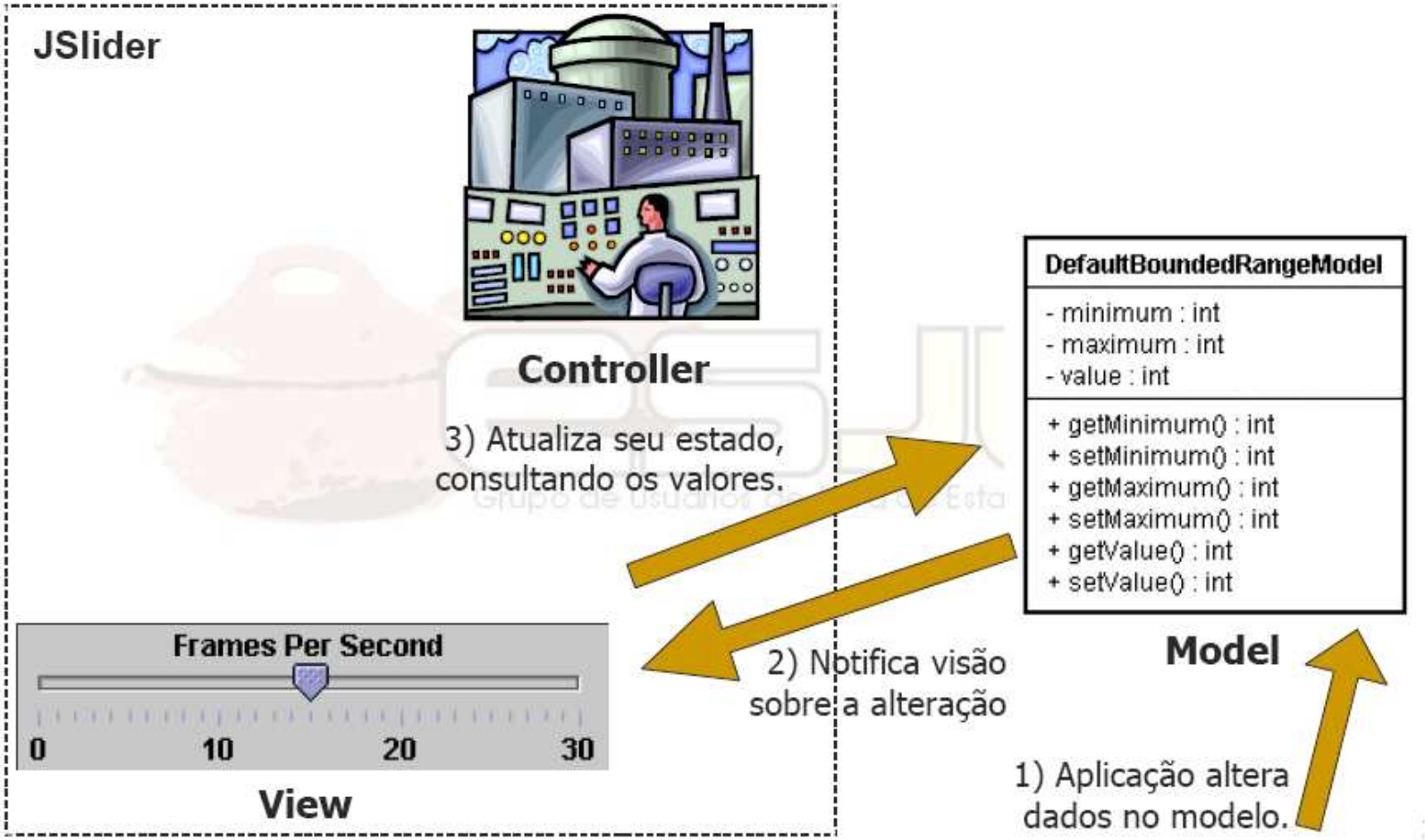
Descrição da colaboração

- Criação dos participantes:
  - Uma instancia do modelo é criada
  - As visões são criadas e recebem uma referência para o modelo
  - Cada visão se registra no mecanismo de notificação do modelo
  - A visão cria um controlador, e passa sua própria referência e também do modelo
  - O controlador também se registra no mecanismo de notificação do modelo
  - A aplicação começa a processar eventos

MVC em GUIs


JSlider
Controller
3) Altera os dados
no modelo.
DefaultBoundedRangeModel
- minimum : int
- maximum : int
- value : int
+ getMinimum() : int
+ setMinimum() : int
+ getMaximum() : int
+ setMaximum() : int
+ getValue() : int
+ setValue() : int
Model
2) Envia estímulos
do usuário
Frames Per Second
0
10
20
30
View
1) Usuário manipula
componente gráfico

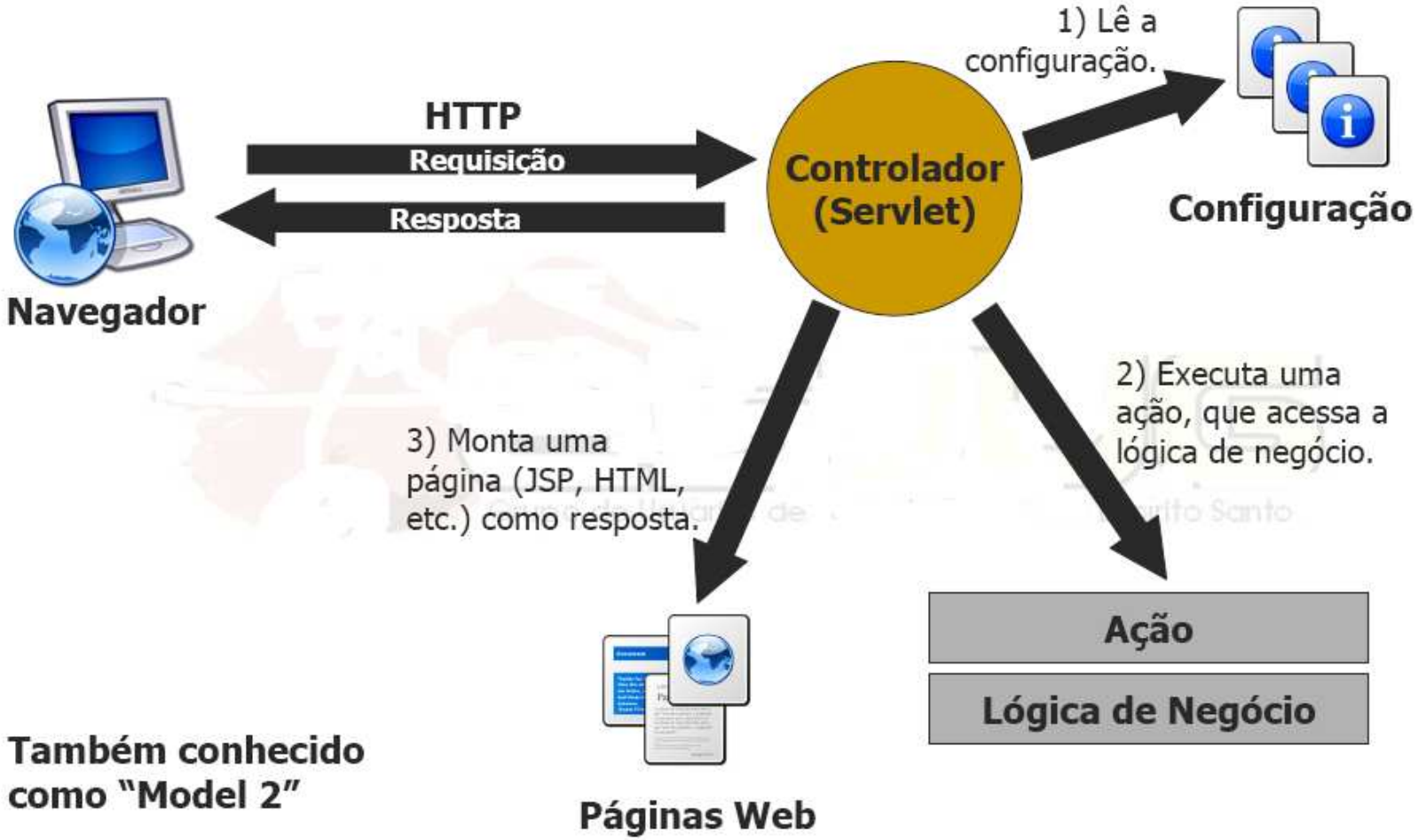
MVC em GUIs


JSlider
Controller
3) Atualiza seu estado, consultando os valores.
DefaultBoundedRangeModel
- minimum : int
- maximum : int
- value : int
+ getMinimum() : int
+ setMinimum() : int
+ getMaximum() : int
+ setMaximum() : int
+ getValue() : int
+ setValue() : int
Model
2) Notifica visão sobre a alteração
1) Aplicação altera dados no modelo.
Frames Per Second
0
10
20
30
View

MVC no Swing

- Model:
  - classe que encapsula um valor que o
  - componente gráfico representa.
- View:
  - classe que“renderiza” o componente gráfico no Look & Feel adequado.
- Controller:
  - classe que recebe estímulos do usuário no componente gráfico exibido e altera o modelo.

MVC na Web


1) Lê a configuração.
HTTP
Requisição
Resposta
Controlador (Servlet)
Configuração
Navegador
2) Executa uma ação, que acessa a lógica de negócio.
3) Monta uma página (JSP, HTML, etc.) como resposta.
Ação
Lógica de Negócio
Também conhecido como "Model 2"
Páginas Web

MVC na Web

- Model:
  - classes de lógica de negócio.
- View:
  - páginas HTML, JSP e similares.
- Controller:
  - Servlet que recebe as requisições HTTP, chama algum método de negócio e, dependendo do retorno, escolhe uma view e redireciona.

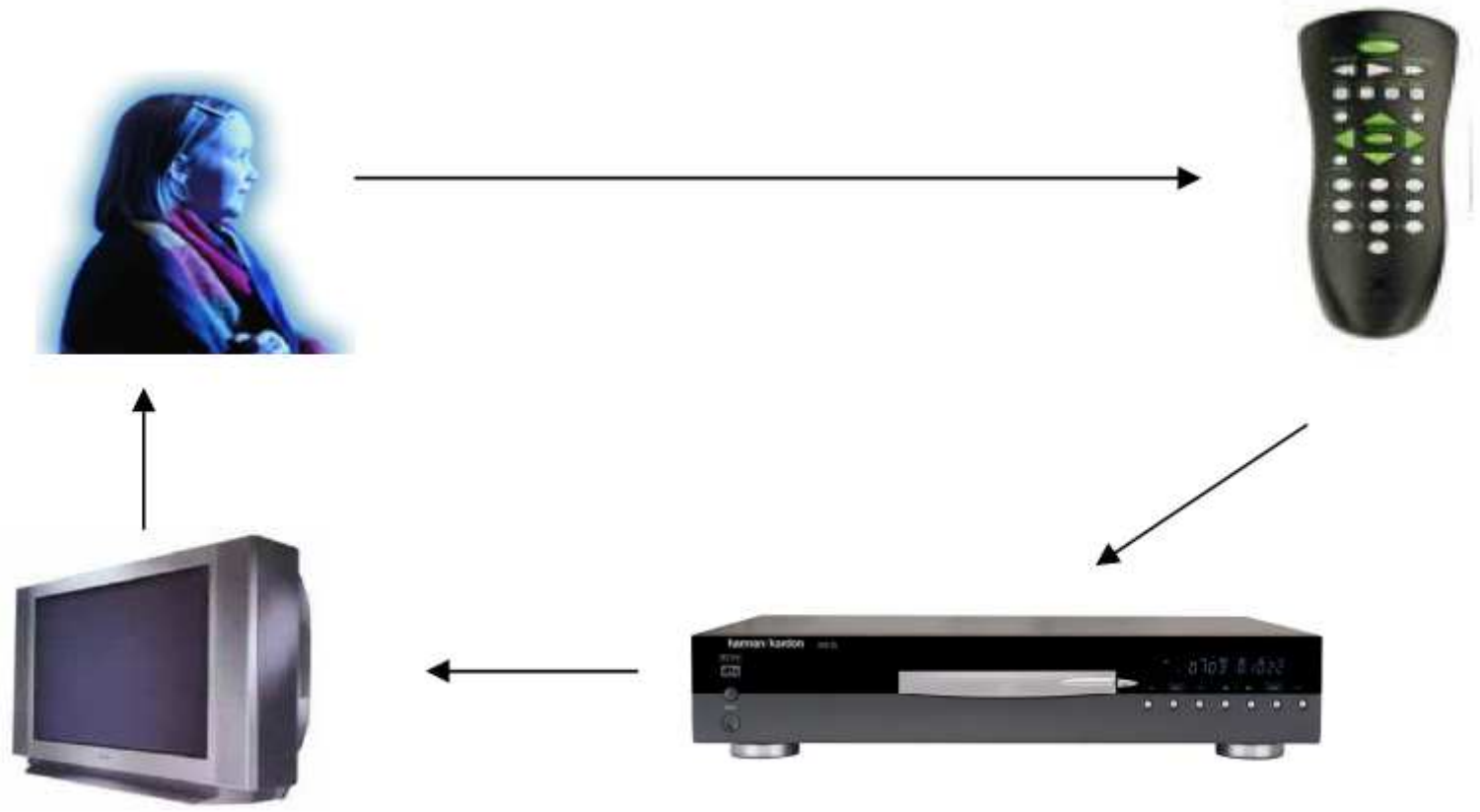
Problemas

- Interfaces com o usuário são sensíveis a mudanças: O usuário está sempre querendo mudar funcionalidades e a interface das aplicações. Se o sistema não suporta estas mudanças, temos um grande problema!
- A aplicação pode ter que ser implementada em outra plataforma. A mesma aplicação possui diferentes requisitos dependendo do usuário:
  - um digitador prefere uma interface onde tudo pode ser feito através do teclado e visualizado como texto.
  - um gerente prefere uma interface através do mouse e de menus com visualização gráfica
- Neste contexto, se o código para a interface gráfica é muito acoplado ao código da aplicação, o desenvolvimento pode se tornar muito caro e difícil.

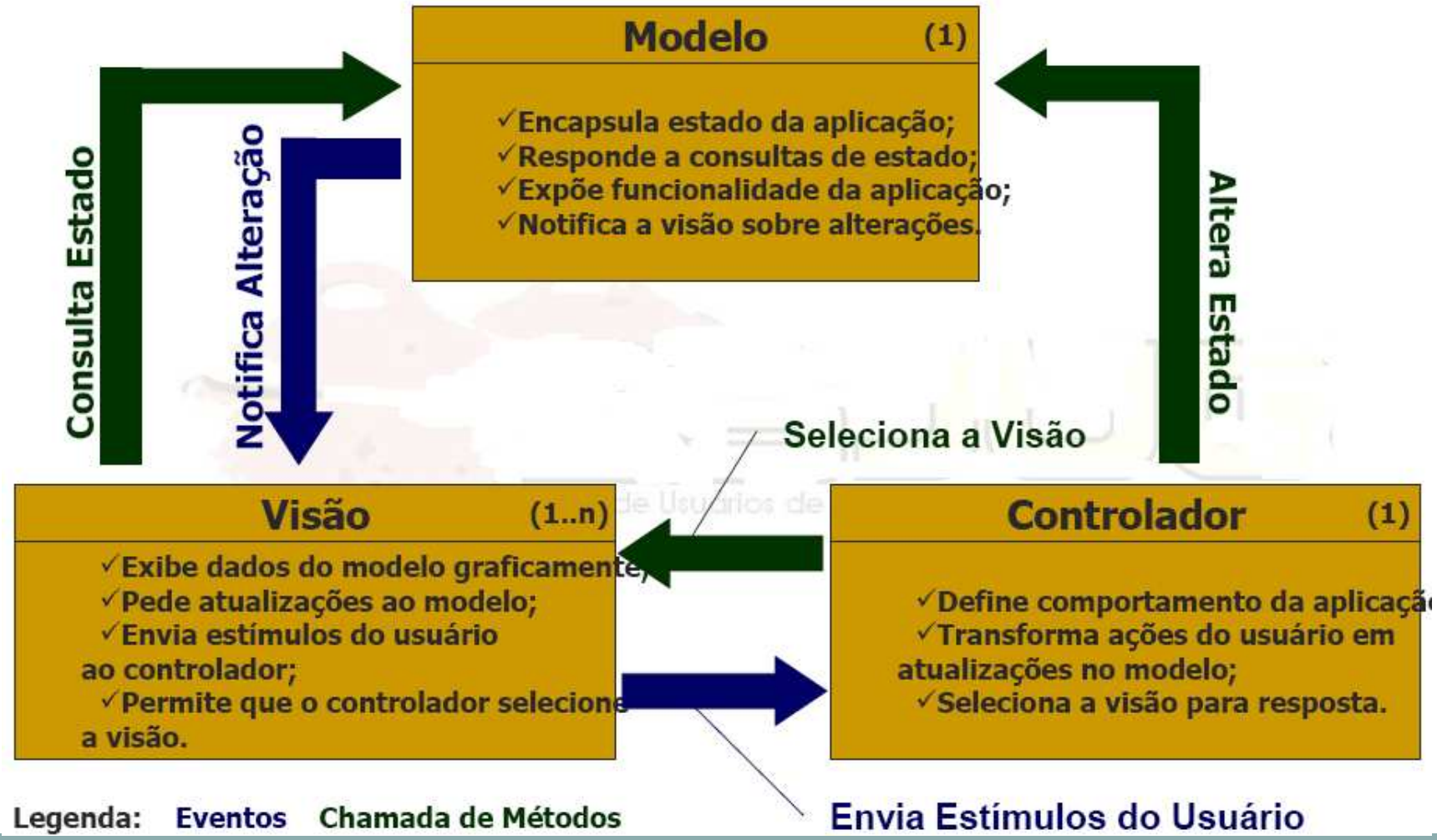
Exemplo

- Queremos implementar um sistema de votação, fazer uma enquete. A enquete deve permitir o voto dos usuários.
- Os votos são contabilizados e exibidos de duas formas:
  - Tela com votos absolutos, que mostra os totais de votos para cada opção;
  - Tela com percentual de votos.
- Patterns\mvc\enquete.jar

Analogia



Camadas MVC


Modelo (1)
✓Encapsula estado da aplicação;
✓Responde a consultas de estado;
✓Expõe funcionalidade da aplicação;
✓Notifica a visão sobre alterações.
Consulta Estado
Notifica Alteração
Altera Estado
Seleciona a Visão
Visão (1..n)
✓Exibe dados do modelo graficamente;
✓Pede atualizações ao modelo;
✓Envia estímulos do usuário ao controlador;
✓Permite que o controlador selecione a visão.
Controlador (1)
✓Define comportamento da aplicação
✓Transforma ações do usuário em atualizações no modelo;
✓Seleciona a visão para resposta.
Legenda: Eventos Chamada de Métodos
Envia Estímulos do Usuário

Frameworks Java para Web

- Diversos frameworks open-source implementam o MVC para a Web:
- JSF (futuro padrão JCP);
- Struts (Apache);
- Tapestry (Apache);
- WebWork (OpenSymphony);
- Dentre outros...

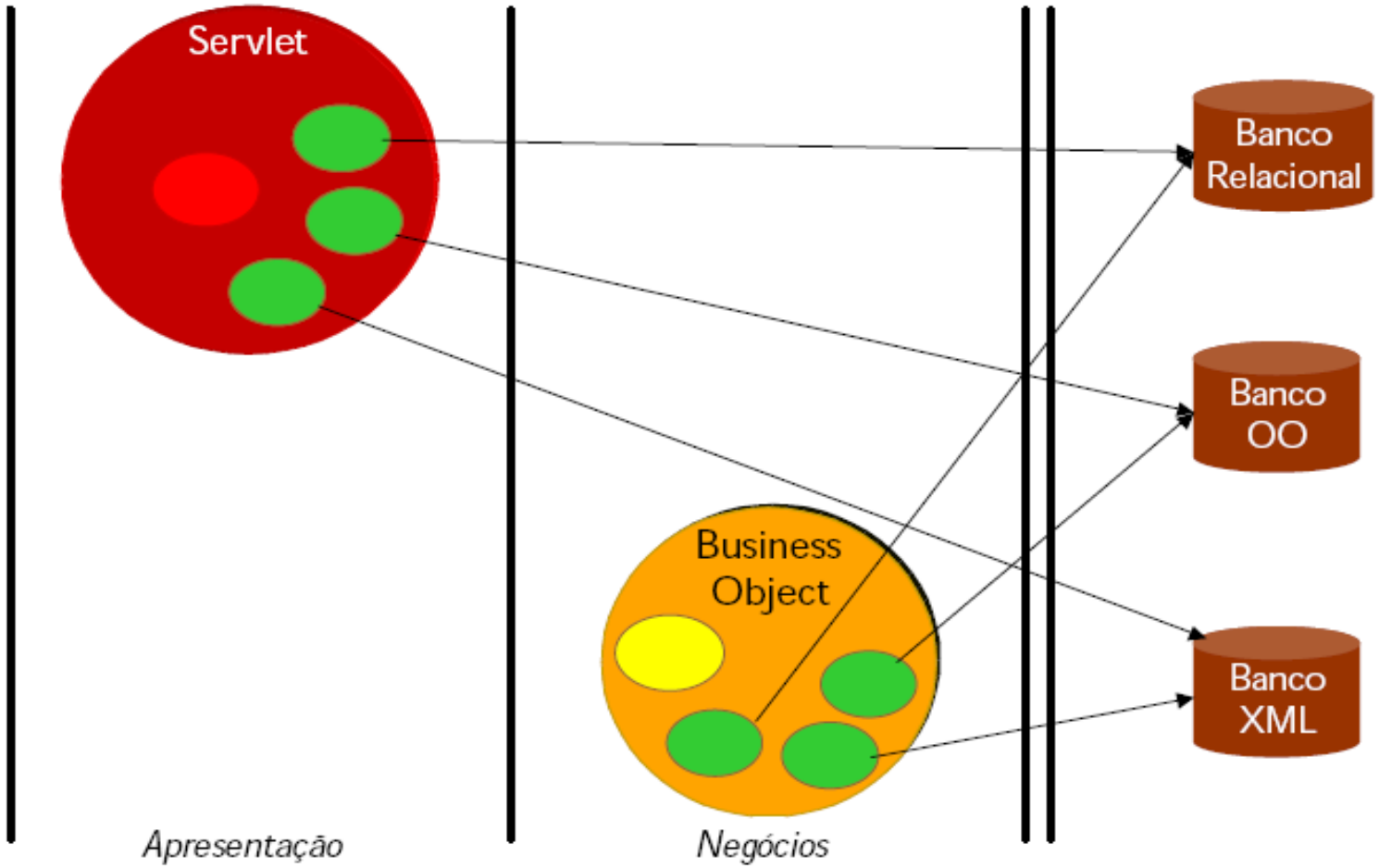
Importância do MVC

- É importante conhecer o MVC para entender melhor a API Swing;
- Ao utilizar MVC para estruturar suas aplicações elas produzirá componentes reutilizáveis e de manutenção mais simples;
- MVC já se tornou um padrão para a camada Web, e é importante conhecê-lo para utilizar bem os frameworks.

DAO – Data Access Object

ABSTRAIR E ENCAPSULAR TODO O ACESSO A UMA FONTE DE DADOS. O DAO GERENCIA A CONEXÃO COM A FONTE DE DADOS PARA OBTER E ARMAZENAR OS DADOS.

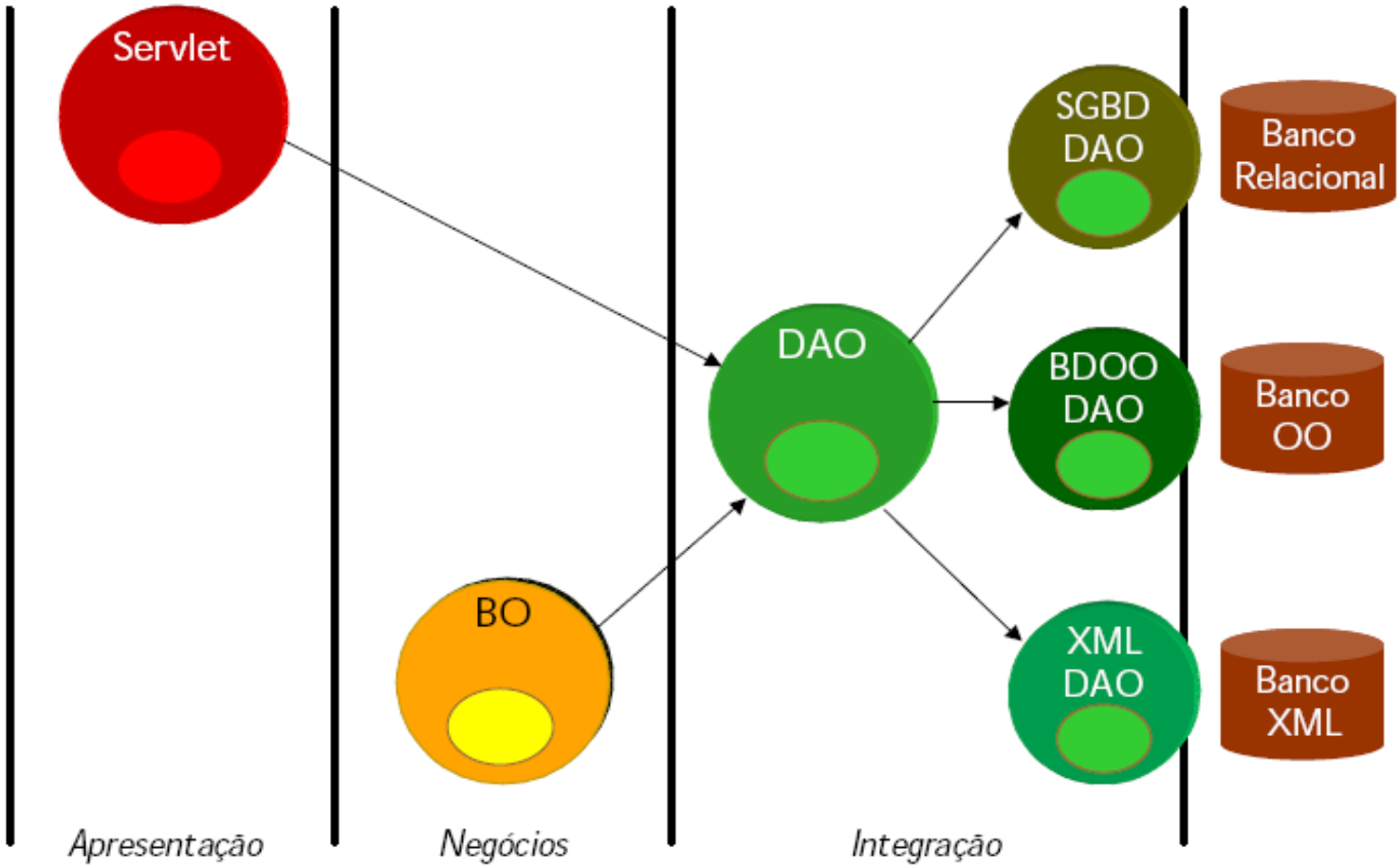
Problema


Servlet
Banco Relacional
Banco OO
Business Object
Banco XML
Apresentação
Negócios

Problema

- *Forma de acesso aos dados varia consideravelmente dependendo da fonte de dados usado*
  - *Banco de dados relacional*
    - *Diferem na sintaxe SQL...*
  - *Arquivos (XML, CSV, texto, formatos proprietários)*
- *Persistência de objetos depende de integração com fonte de dados*
  - *Colocar código de persistência (ex: JDBC) diretamente no código do objeto que o utiliza ou do cliente amarra o código desnecessariamente à forma de implementação*
  - *Ex: difícil passar a persistir objetos em XML, LDAP, etc.*

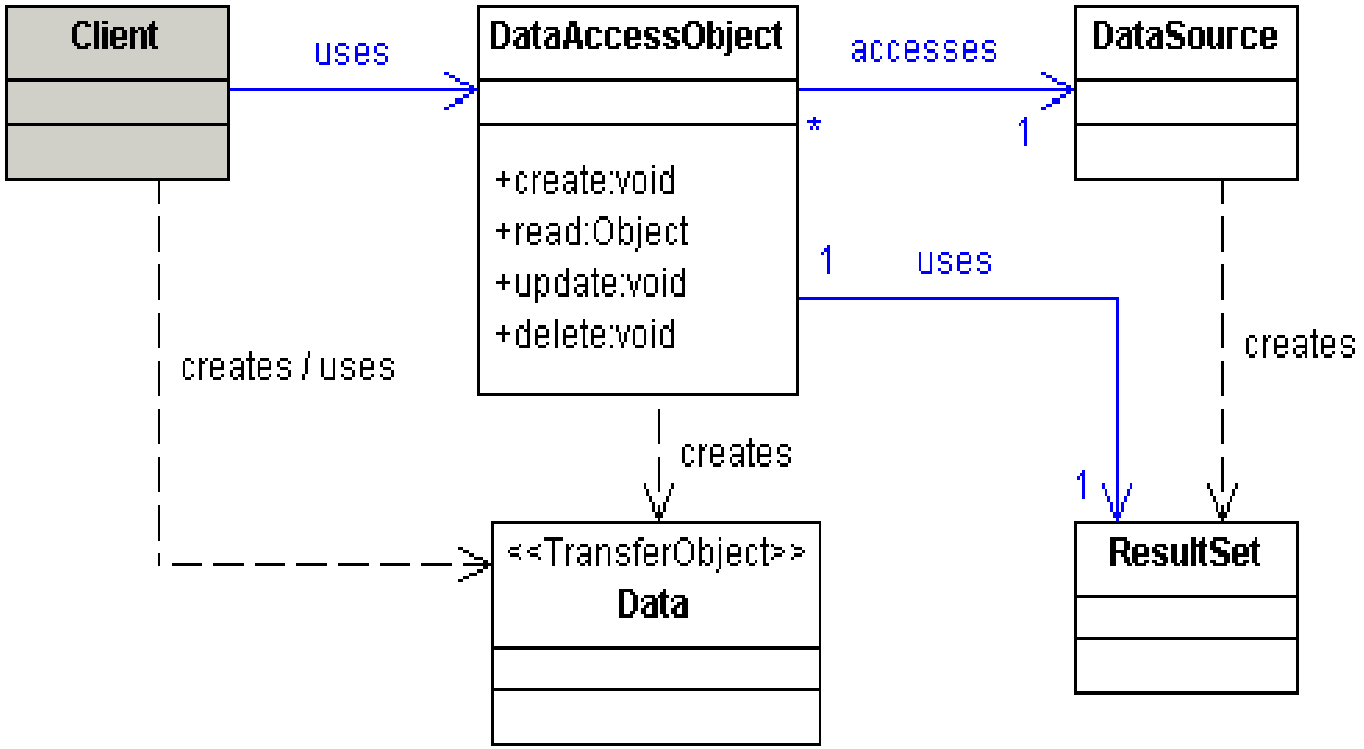
Solução: DAO


Servlet
BO
DAO
SGBD DAO
BDOO DAO
XML DAO
Banco Relacional
Banco OO
Banco XML
Apresentação
Negócios
Integração

DAO

- *Data Access Object (DAO) oferece uma interface comum de acesso a dados e esconde as características de uma implementação específica*
  - *Uma API: métodos genéricos para ler e gravar informação*
  - *Métodos genéricos para concentrar operações mais comuns (simplificar a interface de acesso)*
- *DAO define uma interface que pode ser implementada para cada nova fonte de dados usada, viabilizando a substituição de uma implementação por outra*
- *DAOs não mantêm estado nem cache de dados*

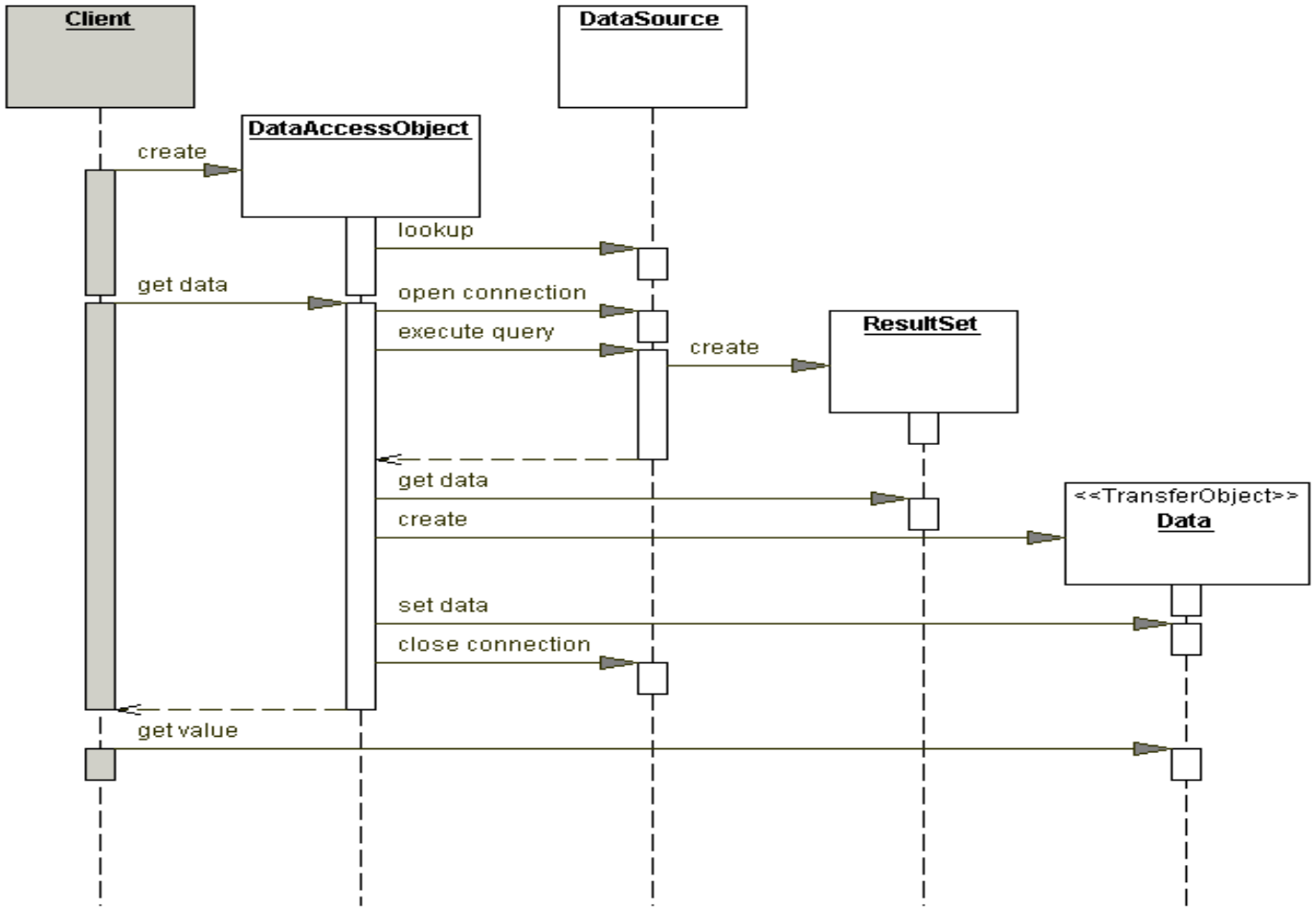
Estrutura


Client
uses
DataAccessObject
accesses
DataSource
*
1
+create:void
+read:Object
+update:void
+delete:void
1
uses
creates / uses
creates
creates
1
<<TransferObject>>
Data
ResultSet

Participantes

- *Client: objeto que requer acesso a dados*
- *DataAccessObject: esconde detalhes da fonte de dados*
- *DataSource: implementação da fonte de dados*
- *Data: objeto de transferência usado para retornar dados ao cliente. Poderia também ser usado para receber dados.*
- *ResultSet: resultados de uma pesquisa no banco*

Colaboração



Conseqüências

- *Transparência quanto à fonte de dados*
  - *Facilita migração para outras implementações*
  - *Basta implementar um DAO com mesma interface*
- *Centraliza todo acesso aos dados em camada separada*
  - *Qualquer componente pode usar os dados (servlets, EJBs)*
- *Camada adicional*
  - *Pode ter pequeno impacto na performance*
- *Requer design de hierarquia de classes (Factory)*

CONTATO:

RONIFABIO@GMAIL.COM