Orientação a Objetos
Da Teoria à Prática em Java

CCUEC/Unicamp
outubro/99

Roteiro

Reutilização de Projeto e Software

- Design Patterns (Padrões de Projeto)
- Frameworks
- Componentes
- RMI (Remote Method Invocation)

Reutilização de Projeto

- **Design Patterns -** padrões de projeto e análise
- **Frameworks** orientados a objetos

Padrões de Projeto e Análise

Padrões de organização de hierarquias de classes, protocolos e distribuição de responsabilidades entre classes, que caracterizam construções elementares de projeto orientado a objetos.

Um padrão de projeto é um estrutura que aparece repetidamente nos projetos orientados a objetos para resolver um determinado problema de forma flexível e adaptável dinamicamente.

Padrões de Projeto e Análise

Descrição padrão de um “design pattern”

- ✔ objetivo
- ✔ motivação
- ✔ aplicabilidade
- ✔ estrutura
- ✔ participantes
- ✔ colaborações
- ✔ conseqüências
- ✔ implementação
- ✔ exemplo de codificação

Padrões de Projeto

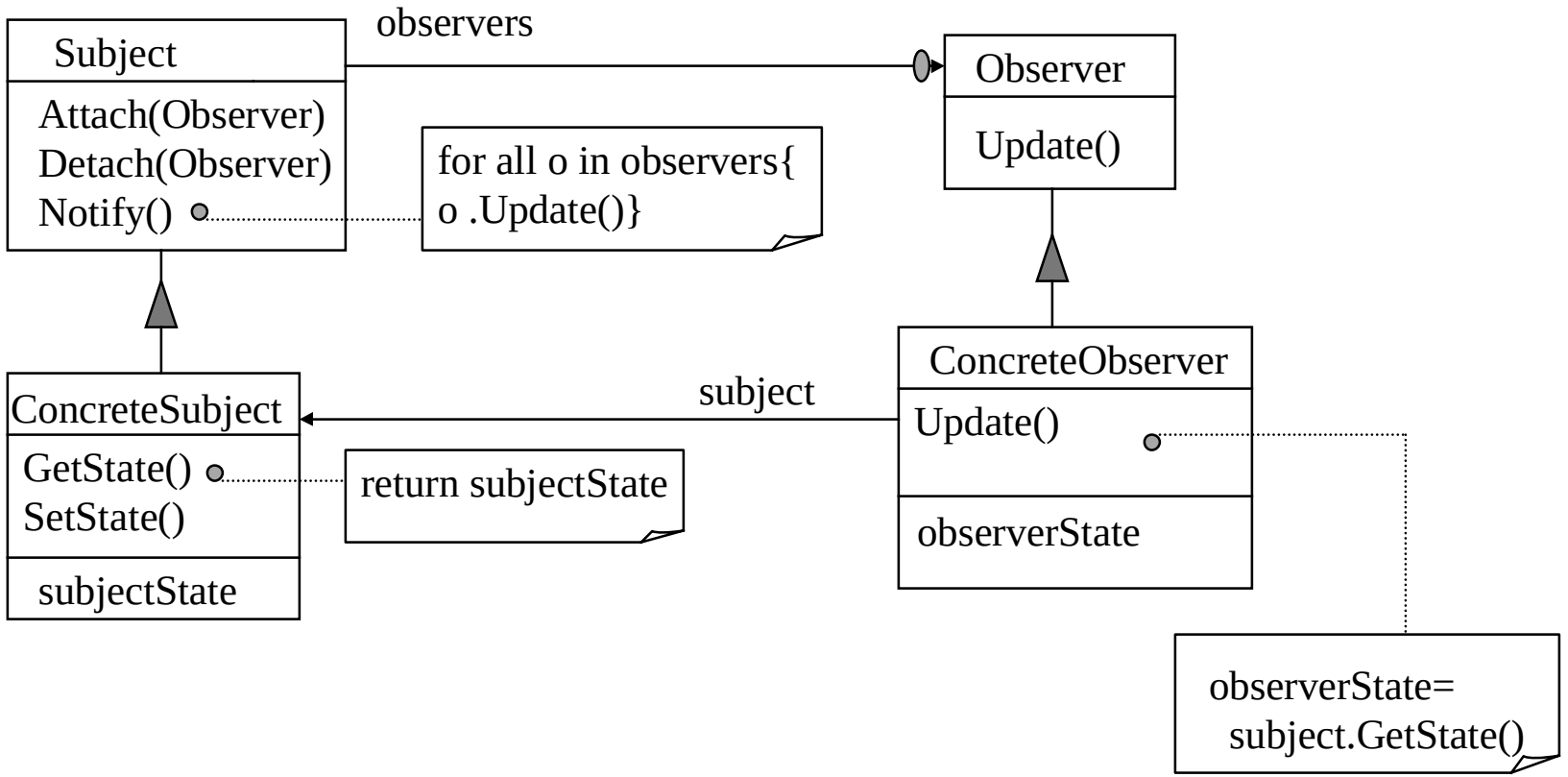
Observer

Suponhamos uma aplicação de Administração de Solicitação de Serviços (ASS) de uma empresa prestadora de serviços, onde serviços-objeto devem ser sempre comunicados a respeito da alteração do estado do funcionário-objeto responsável pela sua execução.

Como modelar a estrutura de dados e os métodos das classes Serviço e Funcionário para atender esse requisito?

Padrões de Projeto

Observer: define uma dependência de um para muitos entre objetos de tal forma que quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente



Padrões de Projeto

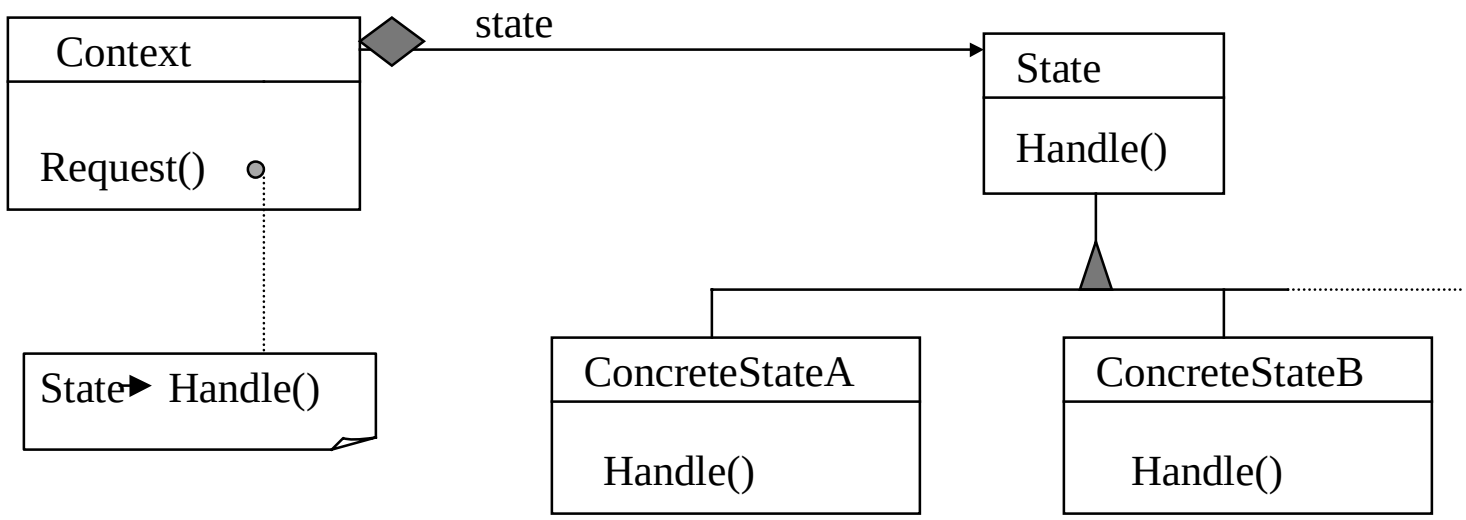
State

Suponhamos que na nossa aplicação ASS, serviços-objeto, dependendo do estado em que se encontram (espera, parado, em andamento, finalizado), tenham comportamentos diferentes, ou seja, reajem diferentemente à mesma mensagem.
Por exemplo: calculaValor()

Como modelar a estrutura de dados e os métodos da classe Serviço para atender a esse requisito?

Padrões de Projeto

State: permite um objeto alterar seu comportamento quando seu estado interno se altera. Parecerá que o objeto mudou de classe.



Padrões de Projeto

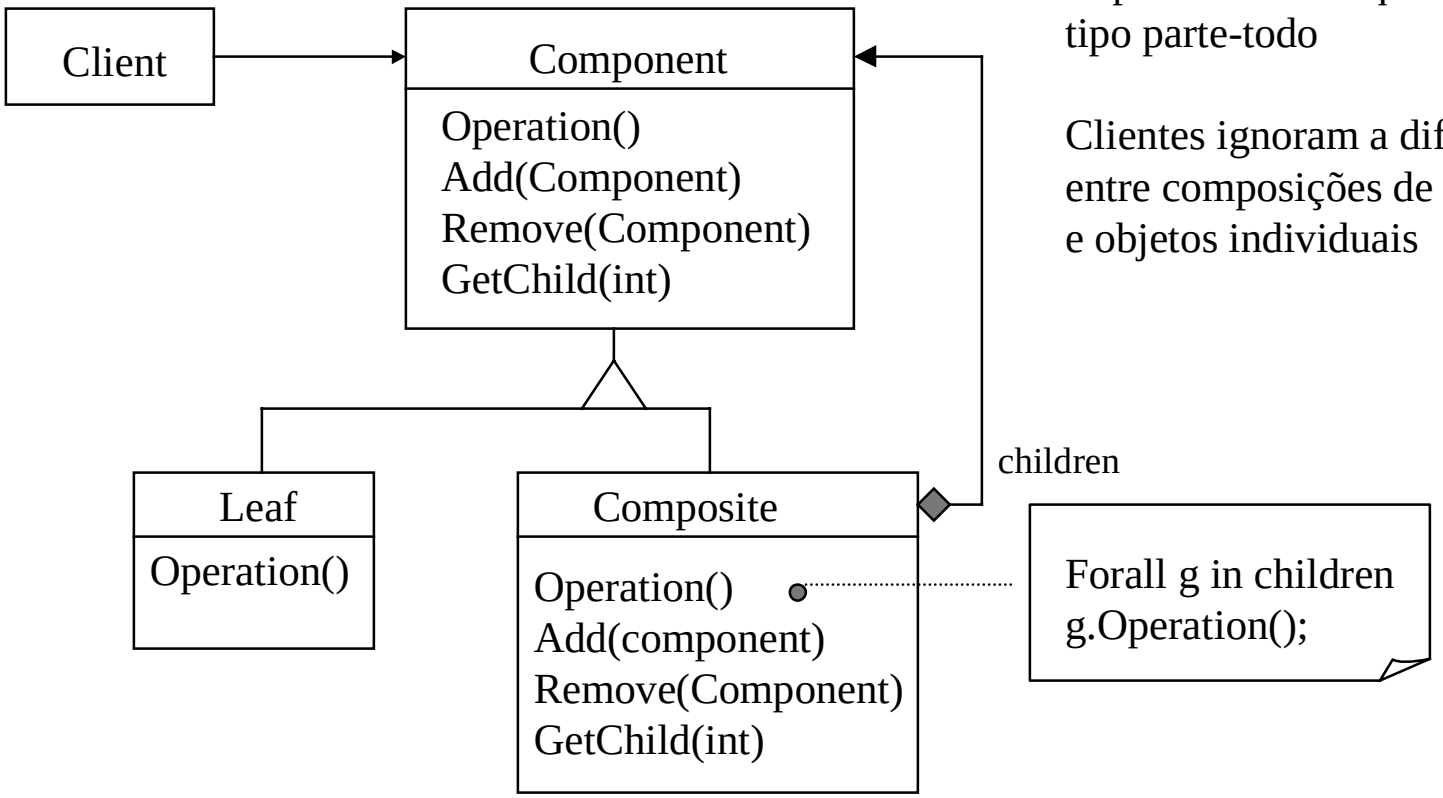
Composite

Na nossa aplicação ASS, um serviço pode ser um serviço de rede, de produção, de suporte ou de desenvolvimento de sistemas. Este último é na verdade uma composição de serviços que devem ser executados pela área de desenvolvimento e pelas outras especialidades mencionadas acima.
Quando um objeto-serviço receber a mensagem “descrever()”, não queremos nos preocupar com o tipo de serviço que está recebendo a mensagem.

Como estruturar a classe Serviço de forma a permitir essa facilidade?

Padrões de Projeto

Composite



Representa hierarquias do tipo parte-todo

Clientes ignoram a diferença entre composições de objetos e objetos individuais

Padrões de Projeto

Reconhecimento de um padrão de projeto

Padrões de Projeto

Como usar padrões de forma eficaz

✓ Estude os padrões aos quais você já tenha acesso
✓ Procure por conceitos básicos similares
✓ Aplique o padrão
✓ Não espere que tudo possa ser resolvido pelos padrões

Como descobrir novos padrões

✓ O padrão pode ser utilizado em outros lugares?
✓ Utilize o padrão várias vezes
✓ Espere fazer mudanças em seu diagrama de classes

Padrões de Projeto

Vantagens

✓ Aumentam a produtividade dos desenvolvedores
✓ Aumentam a consistência entre as aplicações
✓ São potencialmente melhores do que os códigos reutilizáveis
✓ Podem ser utilizados em combinação para resolverem problemas difíceis
✓ Existem novos padrões sendo desenvolvidos todos os dias

Desvantagens

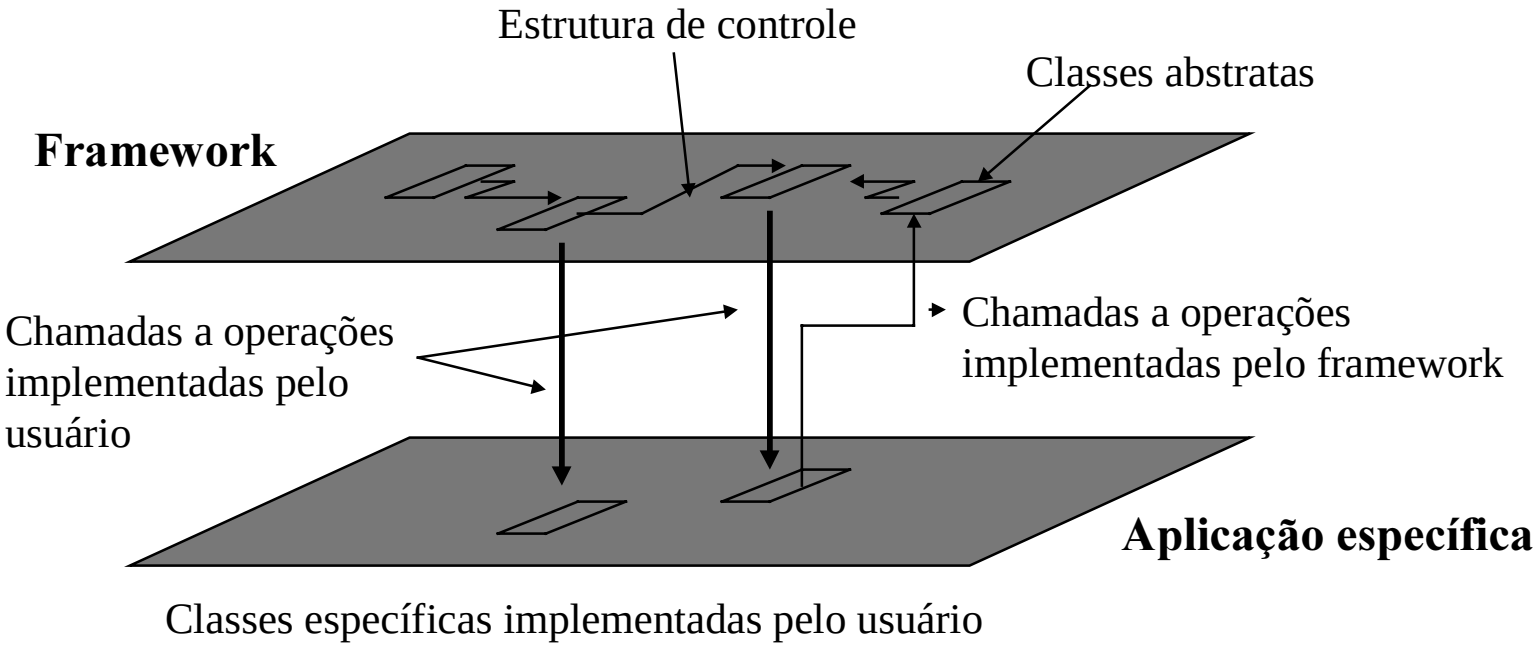
✓ Precisamos aprender um grande número de padrões
✓ Alguns desenvolvedores não estão abertos a aceitar o trabalho dos outros
✓ Os padrões não são códigos

Frameworks

Classes abstratas funcionam como um molde para as suas subclasses.
Da mesma forma, um projeto constituído por classes abstratas funciona como um molde para aplicações.
Um projeto constituído por classes abstratas é denominado **framework de aplicações orientado a objetos.**

Um framework pode ser considerado como uma infra-estrutura de classes que provêem o comportamento necessário para implementar aplicações dentro de um domínio através dos mecanismos de especialização e composição de objetos, típicos das linguagens orientadas a objetos

Frameworks - Visão Conceitual



Visão conceitual da estrutura de um framework

Frameworks - Exemplo

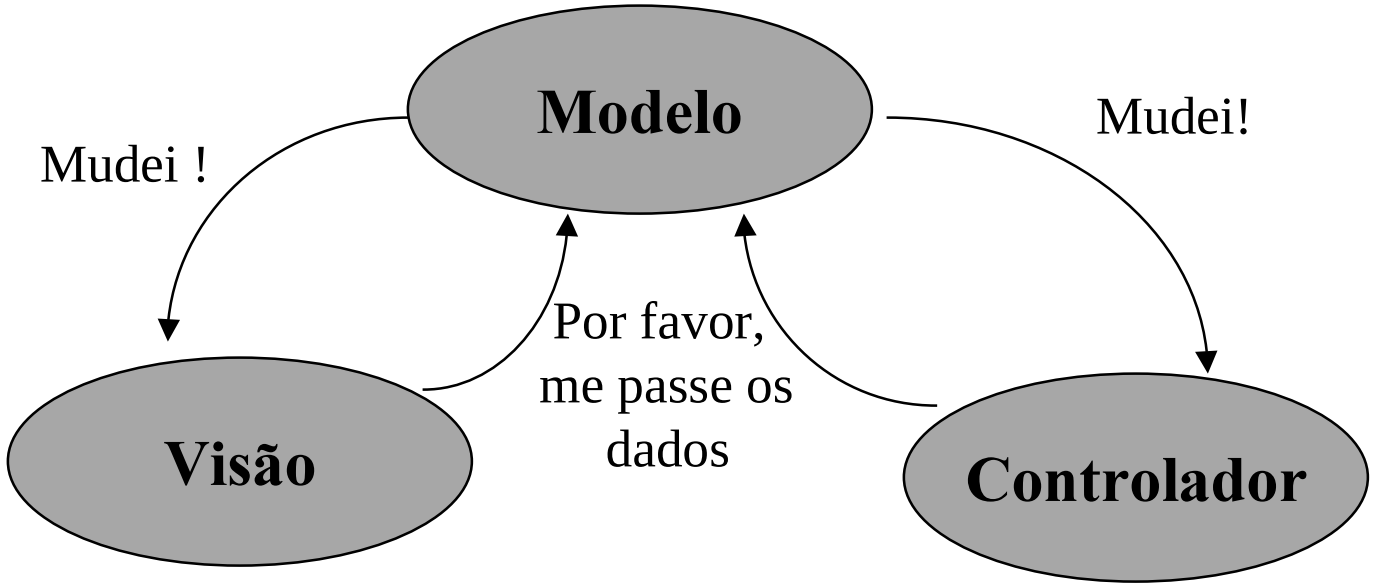
MVC - Model/View/Controller

Modelo: contém os dados da aplicação junto com a lógica dos negócios que define como alterar e acessar os dados; o modelo pode ser compartilhado entre vários objetos, visões e controladores.

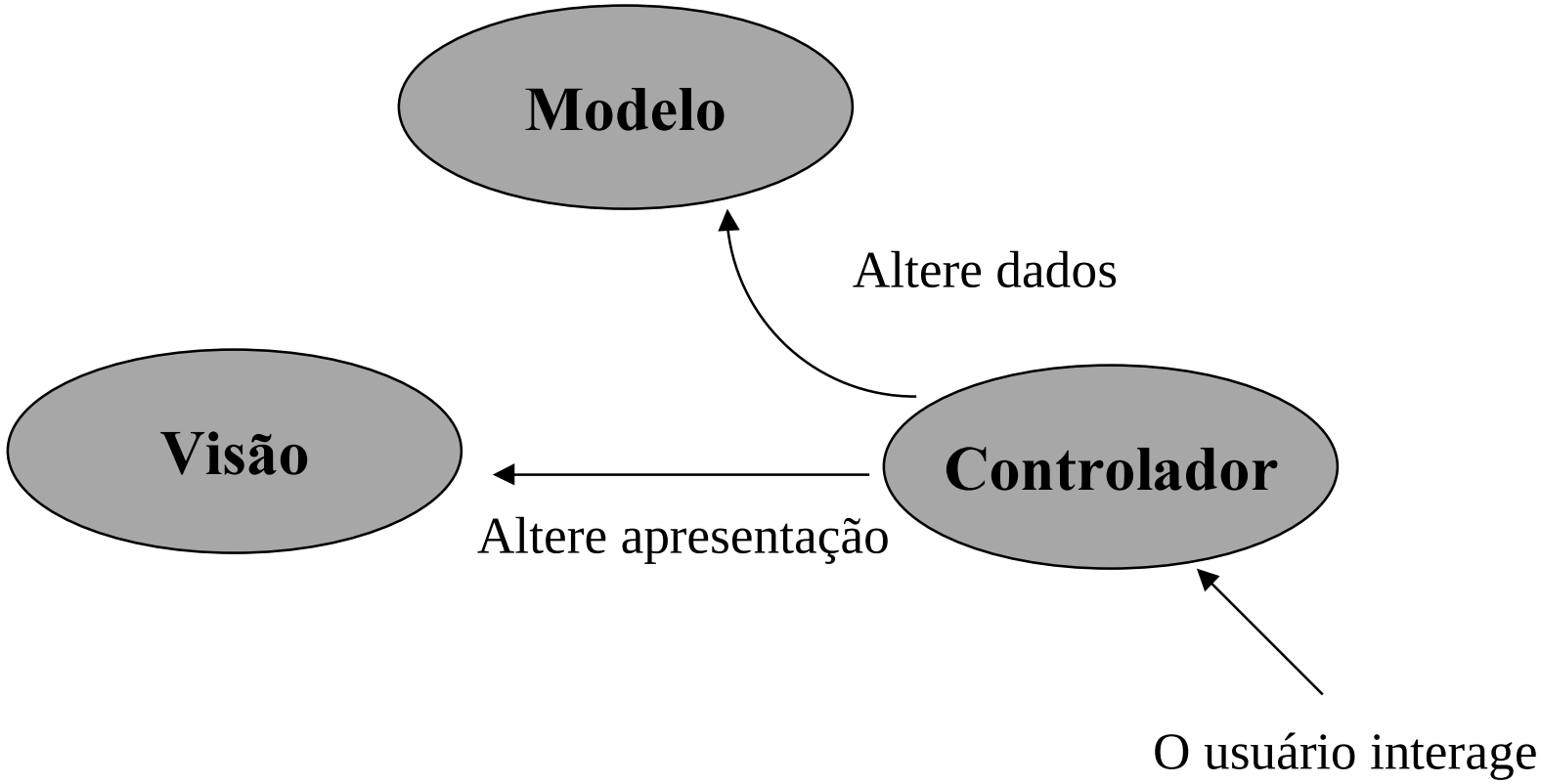
Visão: é a forma de apresentação dos dados do modelo para o mundo externo, pode ser na forma de GUI, fala, som, listagens ou mesmo em uma saída não orientada a usuários, como ligar um ar condicionado.

Controlador: é a forma de tratar a entrada do usuário ou outro meio e dar feedback para o modelo, normalmente alterando alguns de seus dados.

Framework MVC

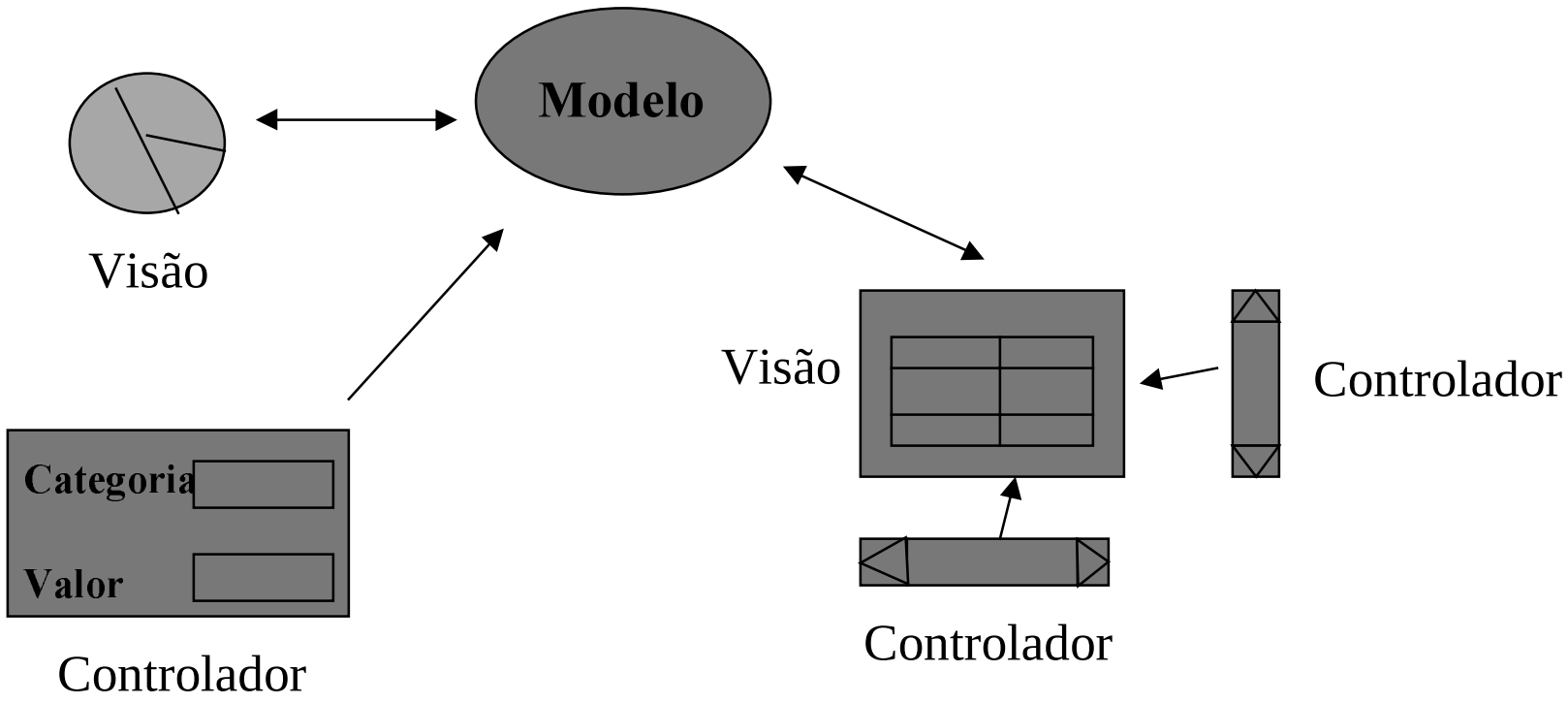


Framework MVC



MVC - Exemplo

MVC - Model/View/Controller



Reutilização de Software

- **Pacotes (Packages)**
- **Componentes**
- **Wrappers**

Packages

Packages são conjuntos de classes relacionadas entre si de forma que ofereçam facilidades uma às outras.

Java traz uma grande quantidade de classes agrupadas em pacotes que estão prontas para serem utilizadas pelo programador.

Packages

Alguns exemplos de Packages em Java

java.lang - base da linguagem Java
(Boolean, Character, Double, Float, Integer, Long, math, Object, String,..)

java.io - pacote que permite manipulação de streams lendo ou gravando em arquivos
(DataInputStream, FileInputStream, FileOutputStream, PrintScreen)

java.util - pacote que provê uma miscelânea de classes úteis incluindo estrutura de dados, time, date, geração de números randômicos, etc..

java.awt - pacote que provê um conjunto de manipulações de interface para o usuário tais como windows, caixas de diálogo, botões, cores, checkboxes..

java.applet - pacote que habilita a criação de applets através da classe Applet e também provê recursos de áudio

Packages - Exemplo

```
import java.applet.*;
import java.awt.*;
import java.util.Date;

public class DigitalClock extends java.applet.Applet implements Runnable {

        Font theFont = new Font("TimesRoman", Font.BOLD, 24);
        Date theDate;
        Thread runner;

        public void start ()
        { if ( runner == null ) { runner = new Thread(this); runner.start();} }

        public void stop ()
        { if (runner != null) { runner.stop(); runner = null;}  }
```

Packages - Exemplo

```
        public void run ()
        { while (true)
                { theDate = new Date();  repaint();
                  try { Thread.sleep(1000);}
                  catch (InterruptedException e) {}
                }
         }

        public void paint (Graphics g)
        { g.setFont(theFont);
          g.drawString(theDate.toString(), 10, 50);
        }
}
```

Componentes

Um componente é um pedaço de código encapsulado e acessível apenas a partir de sua interface

Um componente possui:

• seu **comportamento externo** (o que ele faz) que é definido na sua especificação
• suas **operações internas** (como ele faz o que se propõe a fazer) que está escondido do mundo externo e pode ser entendido apenas se examinarmos seu código fonte
• seu **executável** (runtime binário .exe .dll)

Componentes

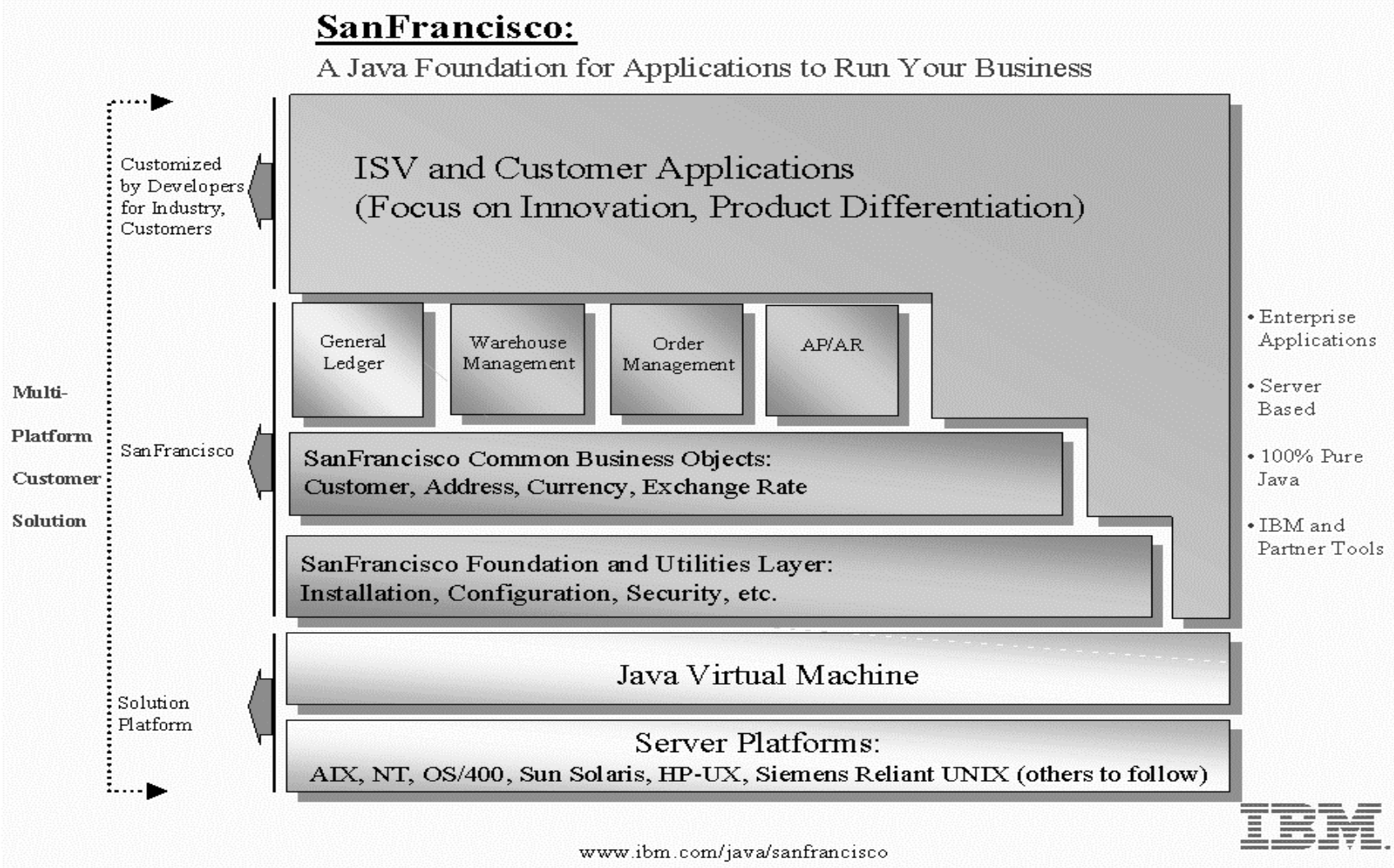
Componentes de negócio são essencialmente **objetos.**
Cada componente implementa a **lógica de negócio** e as propriedades relativas a uma entidade do mundo real.
O que os distingue dos objetos tradicionais é a capacidade de ser utilizados por aplicações produzidas em **diferentes linguagens e tecnologias**, rodando sobre diferentes sistemas operacionais.
A tecnologia de componentes altera radicalmente a forma como os sistemas de informação são desenvolvidos.
Os componentes podem ser considerados como blocos básicos de construção.
Para criar um novo sistema, os desenvolvedores apenas combinam componentes.

Java Beans - Componentes Java

Arquitetura Java de **componentes reutilizáveis** independente de plataforma.

Um **JavaBean** é um componente de software reutilizável que pode ser manipulado visualmente em uma ferramenta de construção
(construtor de páginas Web, construtor visual de aplicações, construtor de GUIs)

Projeto SanFrancisco - IBM



Wrappers - legacy systems

Um **wrapper** é um componente que fornece serviços implementados por aplicações *legacy*.

Um **wrapper** pode ser utilizado para eliminar as dependências entre os sistemas atuais e fornecer a funcionalidade das aplicações *legacy* para novas soluções baseadas em componentes.

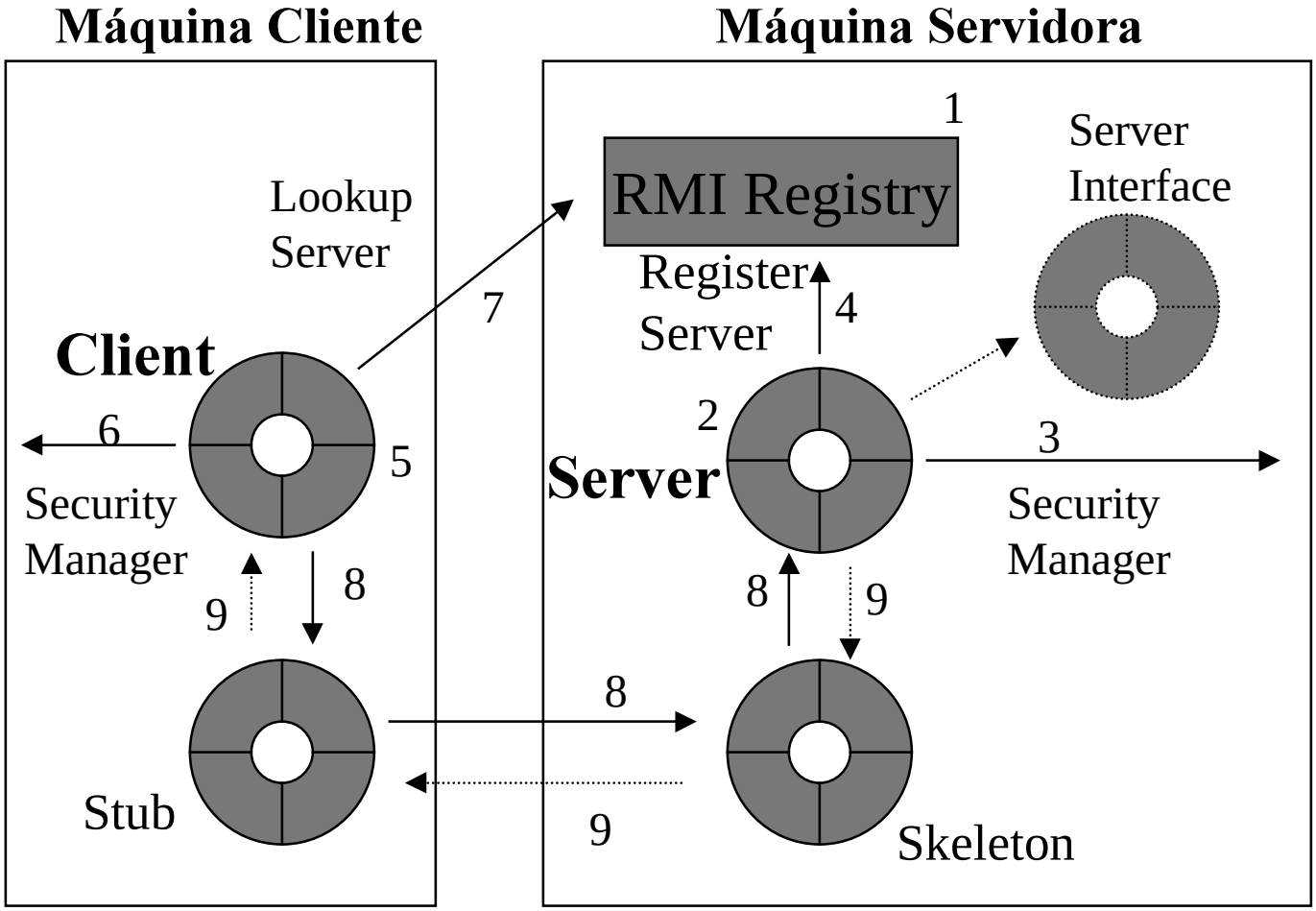
Objetos Distribuídos em Java

O sistema RMI (Remote Method Invocation) permite que um objeto rodando em uma máquina virtual Java (JVM) chame métodos de um objeto que esteja rodando em outra JVM.

RMI permite a comunicação remota entre programas escritos em Java.

RMI provê um mecanismo através do qual o servidor e o cliente se comunicam e passam informações. Estas aplicações são chamadas de aplicações de **objetos distribuídos**

RMI - Ambiente de Execução



Metodologia de Desenvolvimento

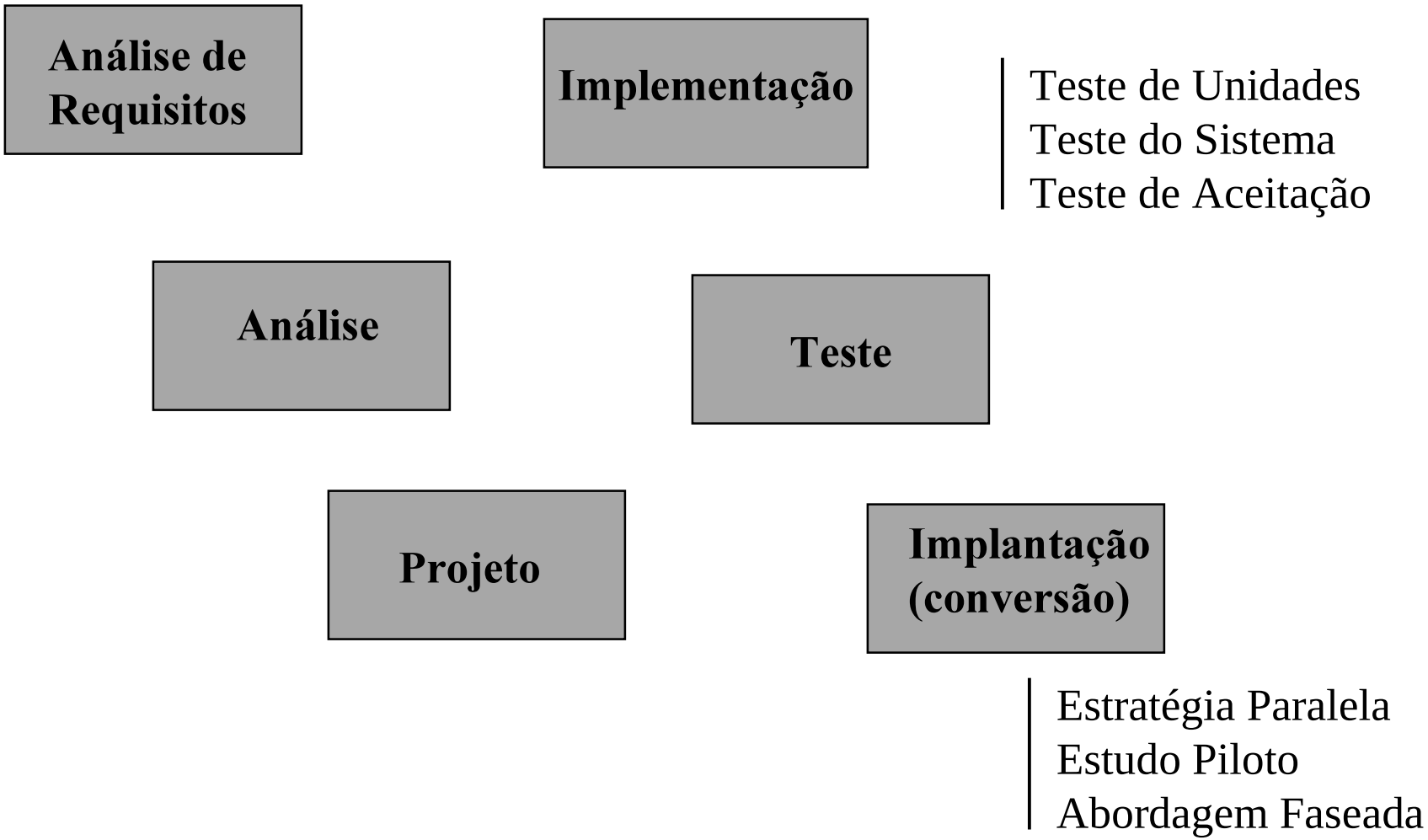
- Visão geral - metodologias e UML
- Técnicas de Análise
- Projeto: Objetos, Interface e Sistema
- Método Integrado
- Considerações Gerais
- Java Studio

Metodologia

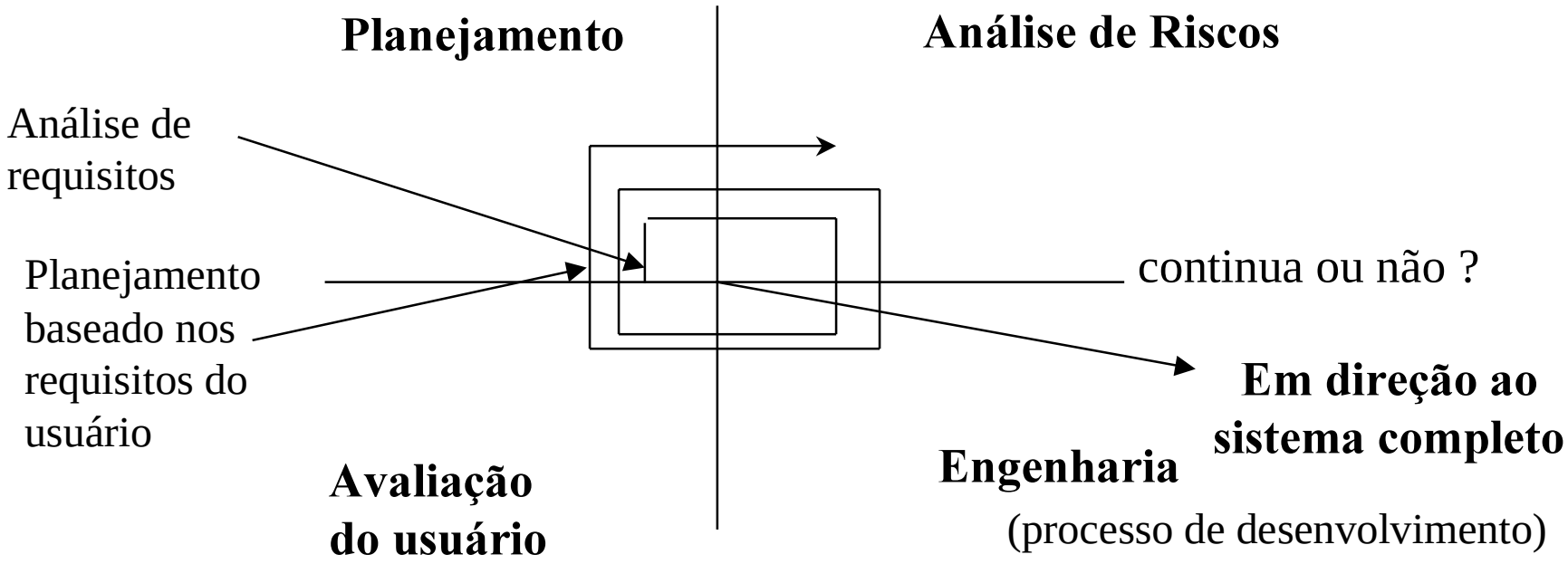
Coleção de **técnicas e diretrizes** para construção, manutenção e melhoria em produtos de software.

Fornece uma base de comunicação, um conjunto de técnicas e uma base para engenharia de software.

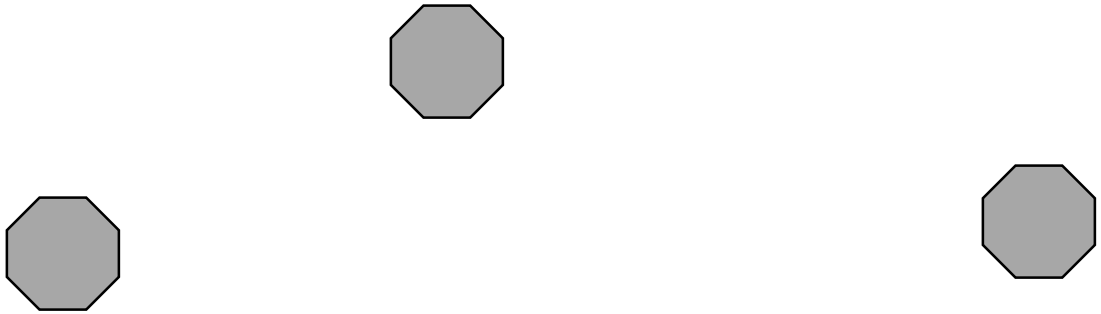
Fases Clássicas



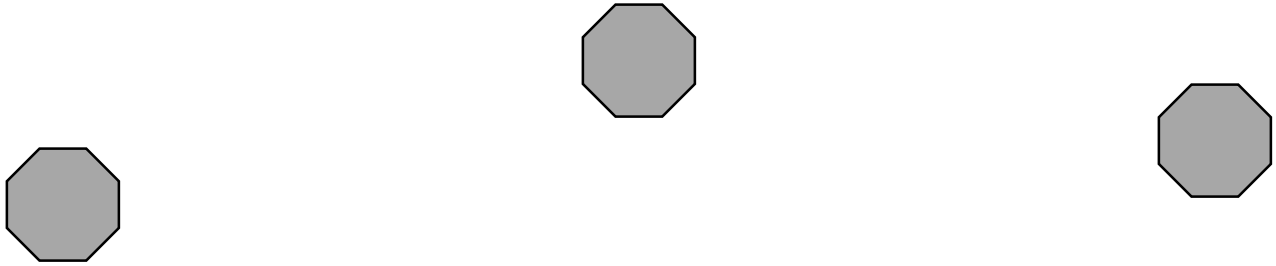
Modelo Espiral



Metodologia OO

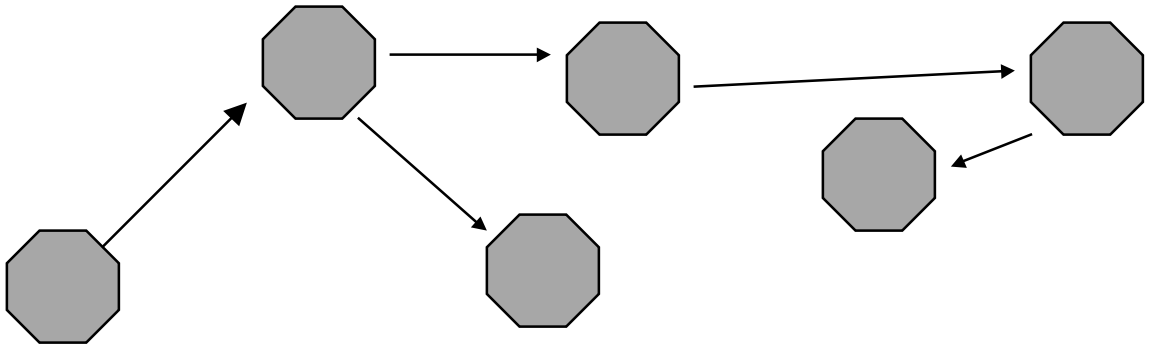


O que faz com que uma metodologia de desenvolvimento de sistemas seja considerada Orientada a Objetos?



Metodologia OO

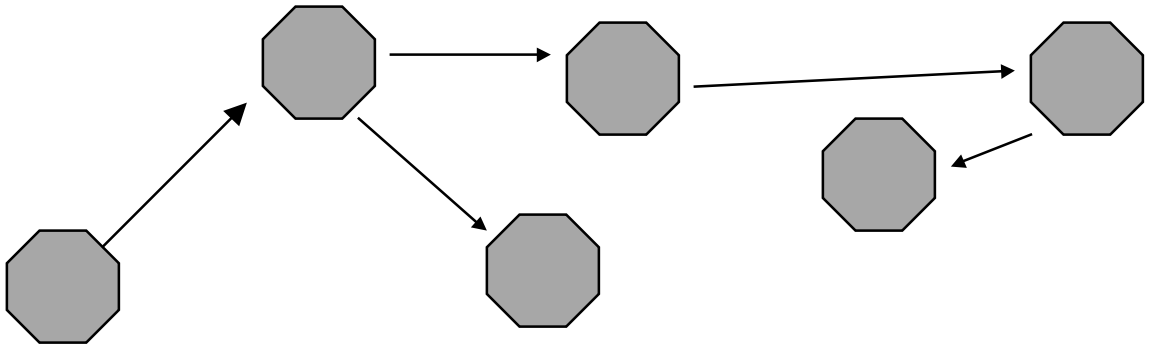
Uma metodologia de desenvolvimento de sistemas é considerada Orientada a Objetos se ela orienta a construção de sistemas a partir do entendimento do mundo real como um conjunto de objetos que comunicam-se entre si de forma coordenada



Metodologia OO

Quais são as principais atividades ?

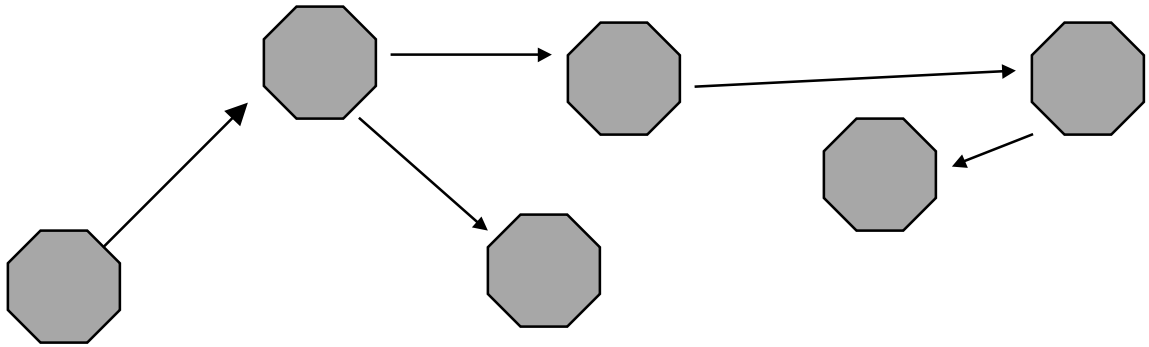
- Entender quais são os **objetos** envolvidos no domínio do problema
- Entender como se **comunicam** no mundo real
- **Projetar** a forma como devem ser **implementados**



Metodologia OO

Quais as principais técnicas utilizadas?

- Entendimento do mundo real - Revisão de processos, Use cases
- Objetos e seus relacionamentos - Modelo de Objetos, CRC, DTE, DI
- Projeto - Padrões de projeto, frameworks, componentes
- Implementação - ambientes de desenvolvimento, middleware (RMI), banco de dados



Métodos de Desenvolvimento OO

Booch - Object-Oriented Design with Applications

Wirfs-Brock - Designing Object-Oriented Software (**CRC**)

Rumbaugh - Object-Oriented Modeling and Design (**OMT**)

Coad-Yourdon - Object-Oriented Analysis

Jacobson - OO Software Engineering - A **Use Case** Driven Approach

Shlaer-Mellor - Object **Lifecycles**-Modeling the World in States

Coleman et al: Fusion - OO Development: The **Fusion** Method

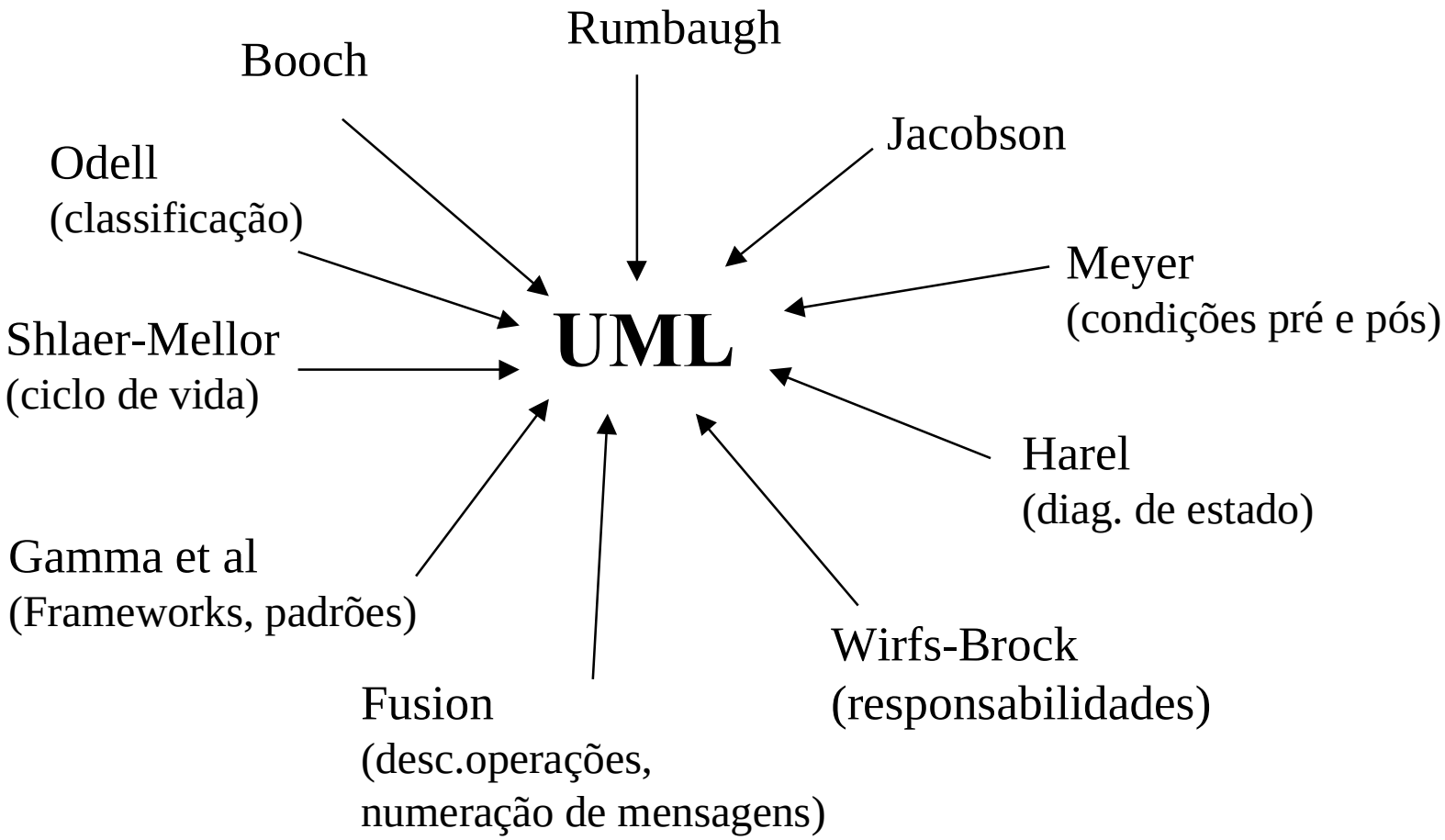
Métodos de Desenvolvimento OO

Estratégia

Escolher um Método como sendo o método **principal** para ser seguido e ater-se à sua notação para todo o ciclo de vida.

Usar **técnicas de outros métodos** para dar suporte aos esforços de modelagem e desenvolvimento.

UML - Unified Modeling Language



Análise Orientada a Objetos

Análise Dinâmica vs Análise Estática

A **Análise Estática** descreve a estruturas e os relacionamentos entre os objetos do domínio do problema
(Modelo de Classes de Objetos)

A **Análise Dinâmica** descreve o comportamento dos objetos em termos de suas mudanças ao longo do tempo
(DTE, DI)

Análise - Use Case

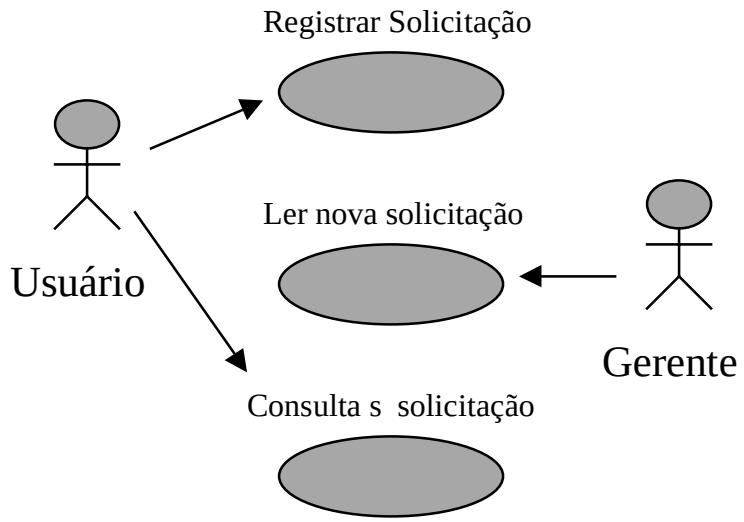


Diagrama do Use Case

Ator: Usuário
Atividade: Registrar solicitação no sistema
Evento: necessidade de execução de um serviço

Curso básico de ação:
1) cliente informa nome, e-mail, telefone,....
2) cliente informa área destino dentro do CPD
3) cliente escolhe produto na lista de produtos existentes
4) o sistema gera um número de solicitação
5) apresentar opção de vincular solicitação a outra já existente
6) avisar área destino que existe uma nova solicitação para ela

Descrição do Use Case

Análise - Modelo de Classes

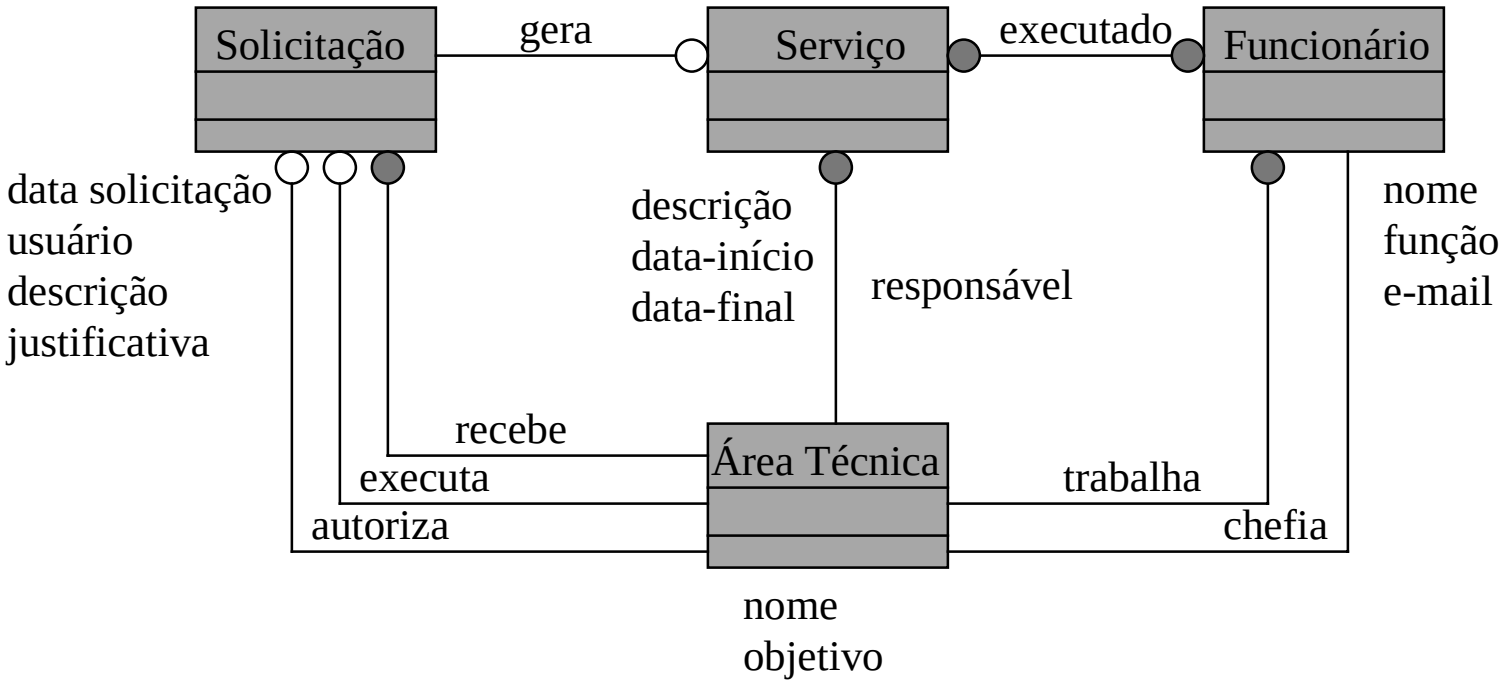


Diagrama de classes extraído a partir da análise dos *Use Cases*

Análise - CRC

Classes/responsabilidades/colaboradores

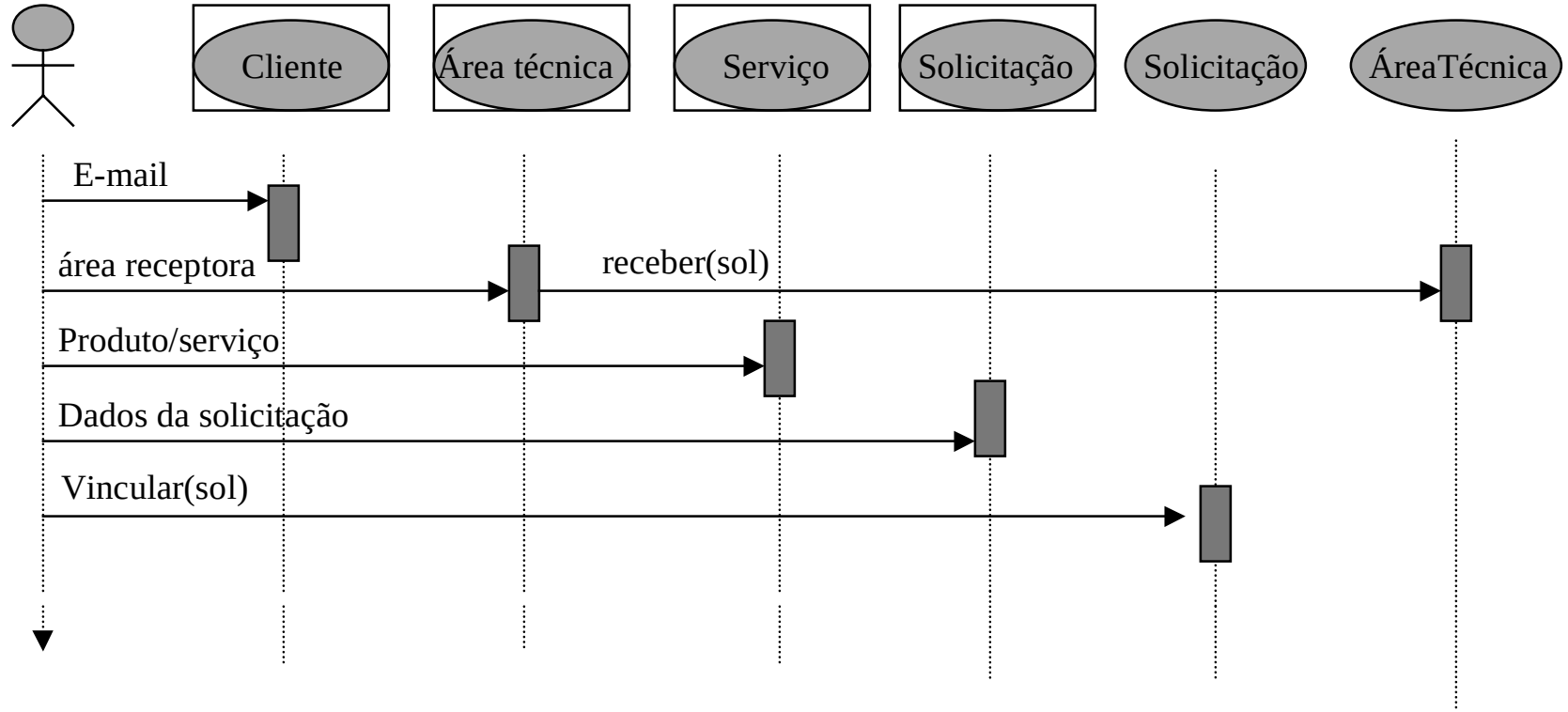
Classe: Serviço
Superclasse:

Responsabilidades	**Colaboradores**
$incluir	
$obter o no. de serviços ativos	
$obter no. solic por serviço	Solicitação
obter identificação	
obter data previsão inicio	
....	
obter usuários por produto	Solicitação
obter área responsável	Área técnica

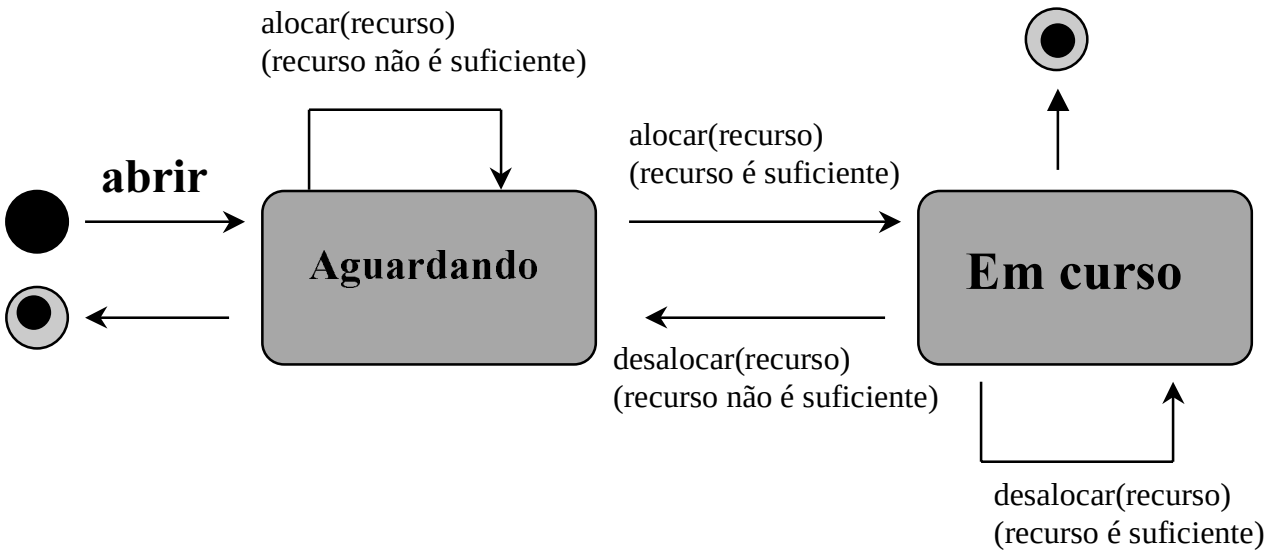
Feed-back para validar os métodos da classe

Análise - Diagrama de Interação (DI)

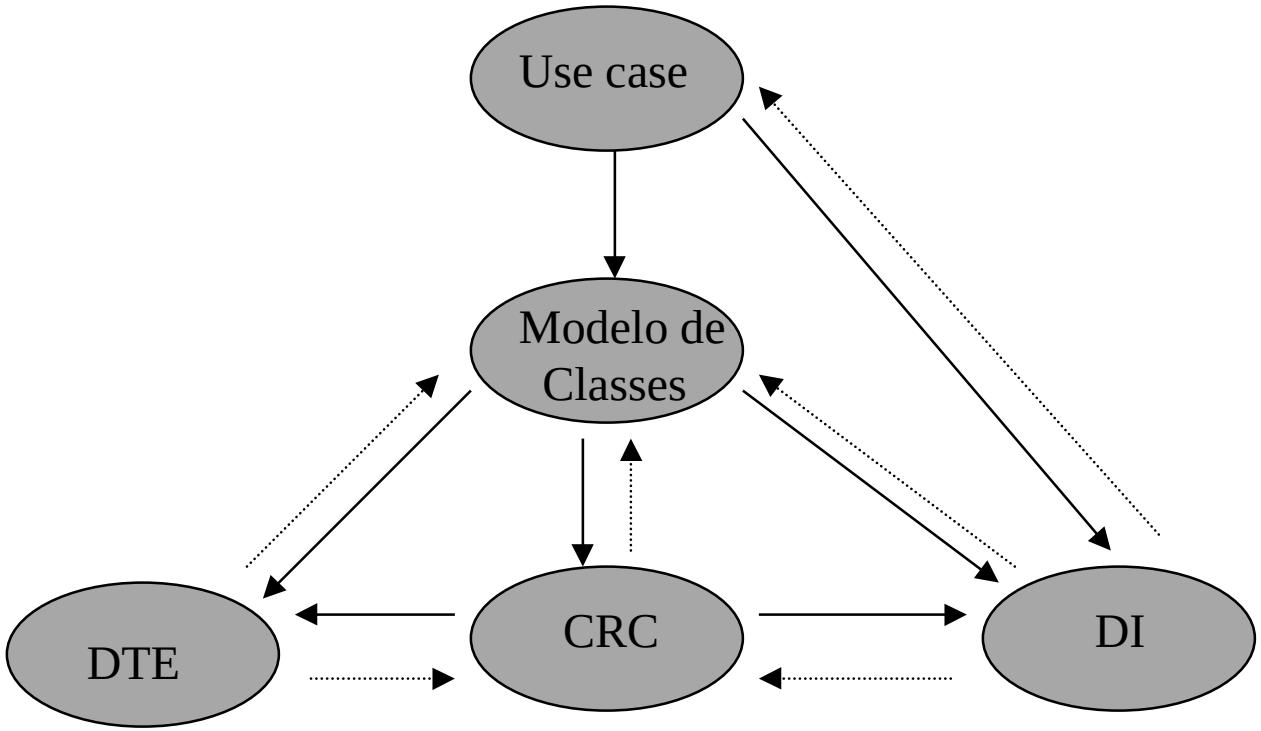
Registrar Solicitação no sistema (pelo usuário)



Análise - Diagrama Transição de Estado

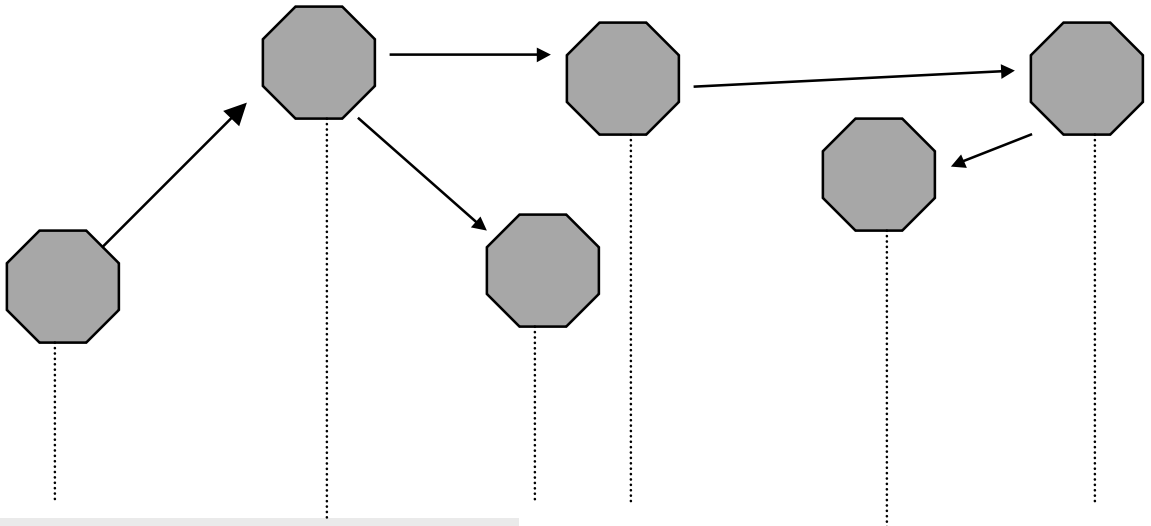


Relacionamento entre as Técnicas de Análise



Análise

No final da análise, já sabemos quais os objetos do “negócio” envolvidos e como eles devem interagir de forma geral, através do modelo de classes e do diagrama de interação



Projeto Orientado a Objetos

Projeto

Será que as classes de negócio são suficientes para a implementação do sistema ?

- O modelo de objetos pode ser otimizado ?
- Como o usuário vai interagir com o sistema ?
- Quem irá controlar o fluxo de mensagens ?
- Quem vai interagir com o banco de dados ?

Projeto

Tanto o domínio do problema, como o domínio da solução devem ser entendidos como um conjunto de classes de objetos

- **Classes de interface -** interação com o usuário
- **Classes de controle -** seqüência das mensagens
- **Classes de banco de dados -** persistência dos dados
- **Classes de negócio -** relacionadas no modelo de análise

Projeto

Principais atividades de projeto

Projeto de Objetos

- Otimização e refinamento do modelo de classes da análise
- Conversão para o modelo relacional de banco de dados

Projeto de Interface

Define a forma de interação dos usuários com a aplicação sendo construída

Projeto de Sistema

Produz uma arquitetura de aplicação a qual inclui decisões sobre a organização do sistema e a alocação de módulos em componentes de hardware e software

Projeto

Tecnologias de apoio

Projeto de Objetos

- Design Patterns (padrões de projeto)
- Packages

Projeto de Interface

Utilização segundo um Guia de Estilo que oriente a utilização correta de recursos gráficos (GUIs), formulários, frames e outros, considerando o ambiente de implementação (por exemplo, Web)

Projeto de Sistema

Arquiteturas de aplicação , frameworks, componentes

Projeto de Interface

Etapas principais

- Análise de usuários e tarefas
- Projeto inicial da interface
- Avaliação da interface sem o usuário
- Teste da interface com os usuários
- Construção da interface
- Acompanhar a utilização da interface pelos usuários
- Adaptar o projeto de interface

Projeto de Interface

Princípios básicos

- Transparência
- Robustez
- Orientação
- Produtividade
- Integridade

Projeto de Interface

Princípios básicos (cont)

- Controle
- Clusterização
- Visibilidade
- Consistência Inteligente
- Utilização de cores como complemento

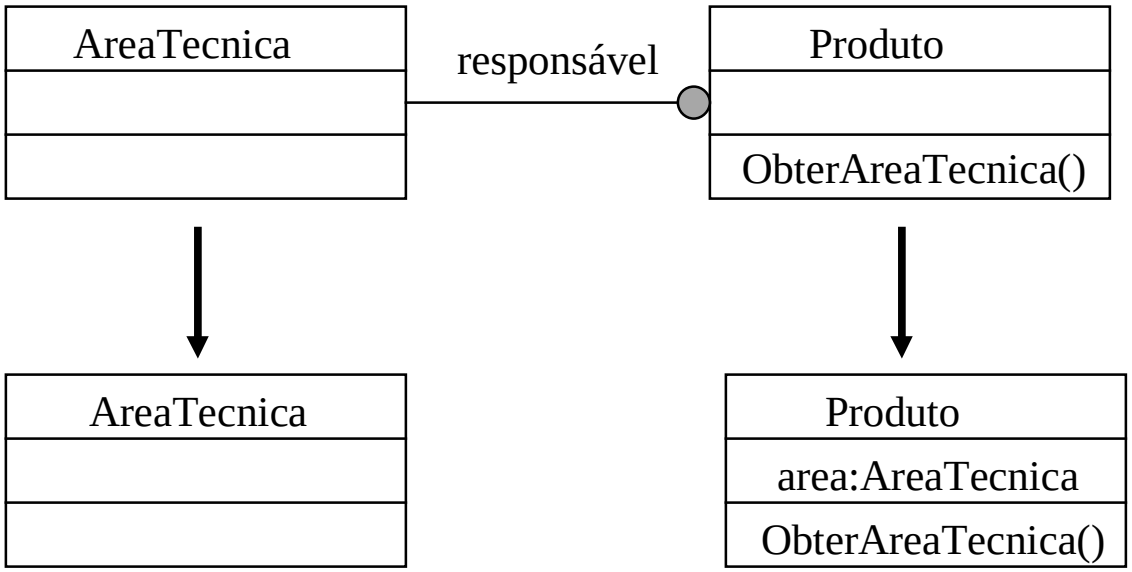
Projeto de Objetos

Etapas básicas

- Projetar algoritmos para implementar os métodos
- Otimizar caminhos de acesso aos dados
- Implementar controle para interações externas
- Ajustar a estrutura de classes para aumentar a herança
- Projetar associações
- Determinar a representação de objetos
- Empacotar classes e associações em módulos

Projeto de Objetos

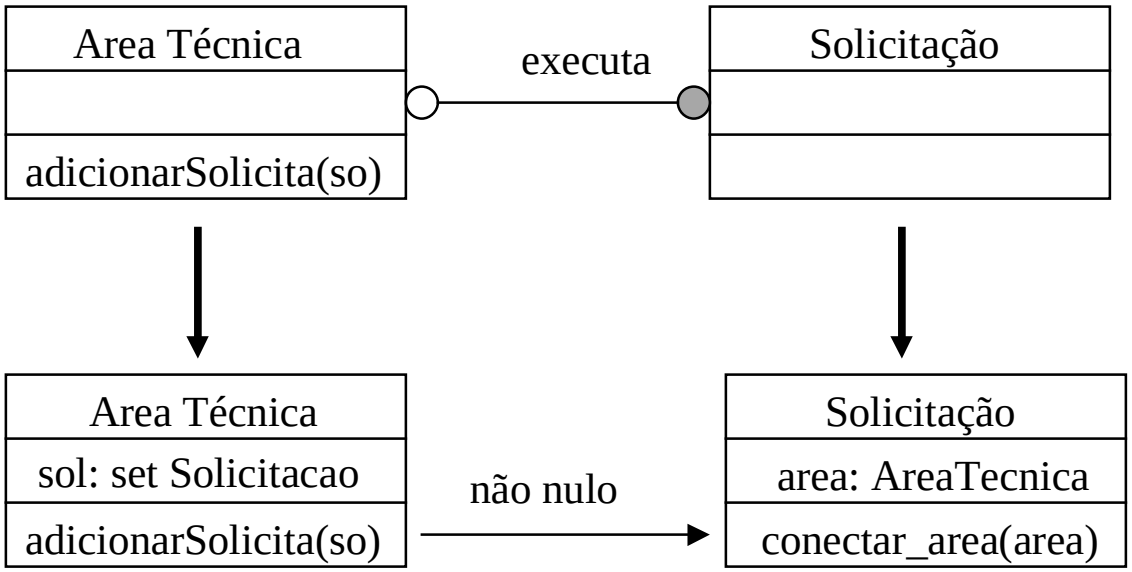
Exemplo de refinamento do modelo de classes da análise



Associações de mão única são atravessadas apenas em uma direção
Regra Geral: implementada como atributos de ponteiro

Projeto de Objetos

Exemplo de refinamento do modelo de classes da análise



Associação bidirecional: execução bem-sucedida de *new Solicitação* requer execução bem-sucedida de *conectar_area* de maneira a ter um vínculo bidirecional consistente

Projeto de Sistemas

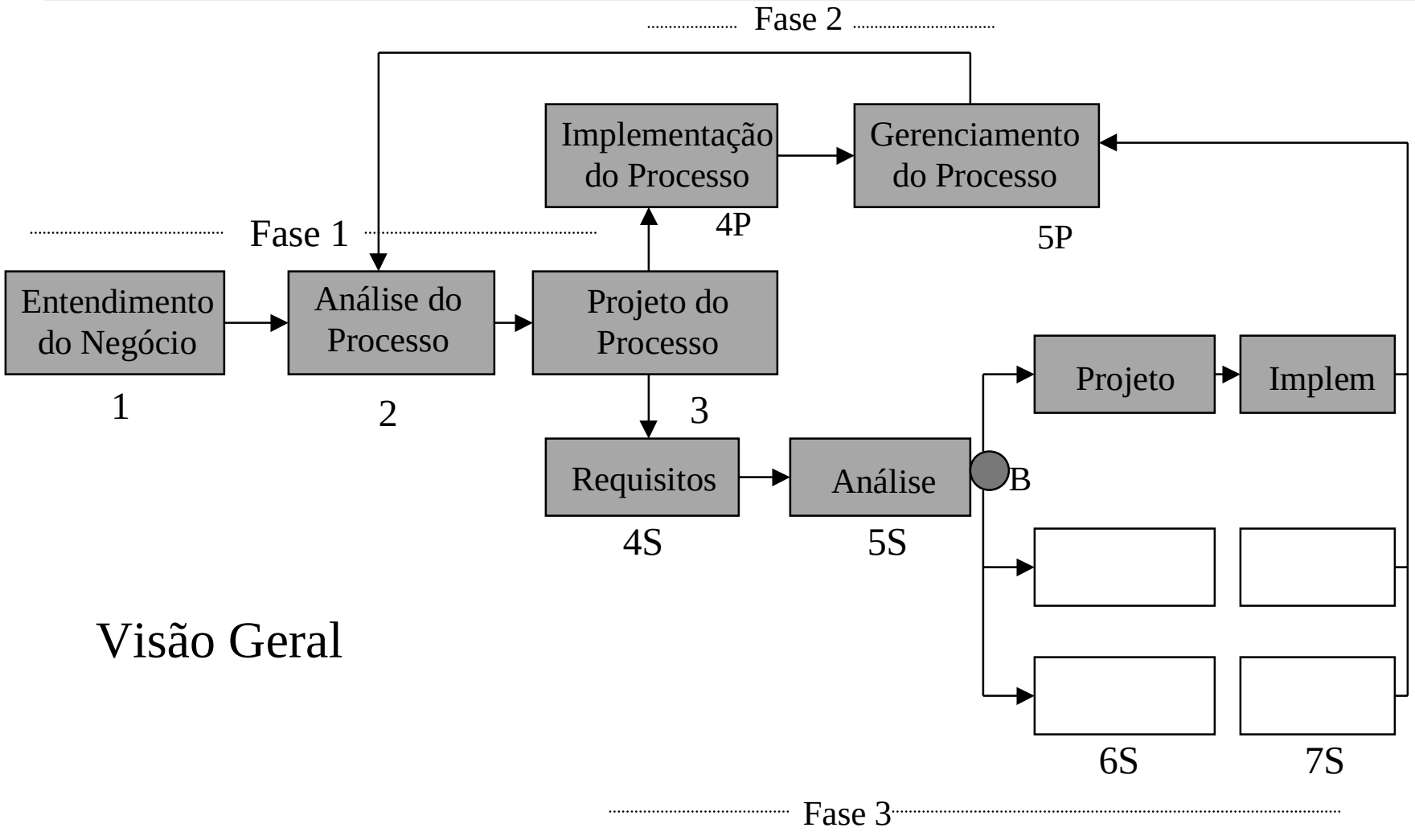
Questões básicas para a Arquitetura da Aplicação

- Organização geral e estrutura de controle global
- Atribuição de funcionalidade aos módulos
- Composição dos módulos
- Distribuição física dos módulos
- Protocolos para comunicação
- Sincronização e acesso aos dados
- Expansão e desempenho

Tecnologias de Implementação

- Linguagens Orientadas a Objetos
  - Java, C++, Smalltalk
- Ambientes de Desenvolvimento
  - VisualAge for Java(IBM)
  - Visual Café (Symantec)
  - Jbuilder 3 (Borland)
- Banco de Dados Orientado a Objetos
  - O2 (O2 Technology)
  - Objectstore (Object Design Inc.)
  - Gemstone (Servio Logic)
  - Jasmine (Computer Associates)

Método Integrado



Visão Geral

Aprendizado

Implementação de uma metodologia OO

- Mudança de paradigma - treinamento intensivo
- Protótipos sem compromissos
- Primeiros sistemas devem ser livres...
- Grupo formal de metodologia - proposta e treinamento
- Acerto do ambiente de desenvolvimento - suporte, padrões
- Administração de classes/objetos - Biblioteca de Classes
- Ferramenta CASE

Orientação a Objetos

Benefícios

- Reaproveitamento
- Estabilidade
- Projetista pensa em termos de comportamento dos objetos e não em detalhes de baixo nível
- Construção de objetos cada vez mais complexos
- Confiabilidade
- Verificação de precisão
- Novos mercados de software
- Desenvolvimento acelerado
- Integridade

Orientação a Objetos

Benefícios

- Programação facilitada
- Manutenção facilitada
- Ciclo de vida dinâmico
- Refinamento durante a construção
- Modelagem mais realista
- Melhor comunicação entre profissionais de sistema e clientes
- Interoperabilidade
- Processamento cliente-servidor
- Processamento distribuído e em paralelo
- Bibliotecas de classes industrializadas e corporativas

Java Studio

Um ambiente de desenvolvimento Orientado a Objetos

Referências

(1) Análise e Projeto Orientados a Objetos
Scott W. Ambler (IBPI Press)

(2) Modelagem e Projetos Baseados em Objetos
J. Rumbaugh et. al (Campus)

(3) Princípios de Análise e Projeto Baseados em Objetos
James Martin (Campus)

(4) Design Patterns - Elements os Reusable Object-Oriented Software
Erich Gamma et. al (Addison-Wesley)

(5) Manual do Visual Age for Java

(6) Software Engineering - A Practitioner´s Approach
Roger S. Pressman