Visões Arquiteturais

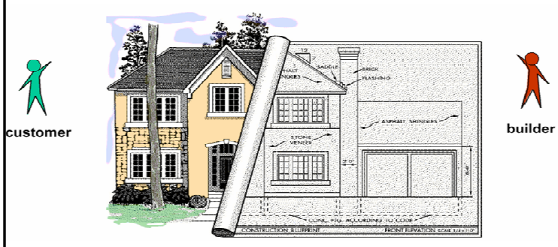
Visões

Existem várias formas de se observar o sistema em construção, cada pessoa envolvida resssalta propriedades que lhe interessa e omite as não relevantes.

–modo como as pessoas que desempenham papéis diferentes dentro do processo de desenvolvimento de software vêem o problema.

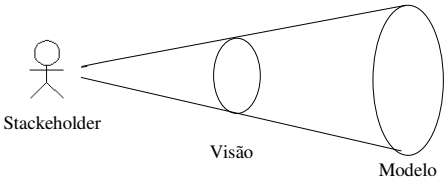
–modo como cada entidade (componente) da arquitetura de software pode ser observada (perspectivas diferentes).

Visões



O que são visões?

Uma projeção de um modelo sob determinada perspectiva.

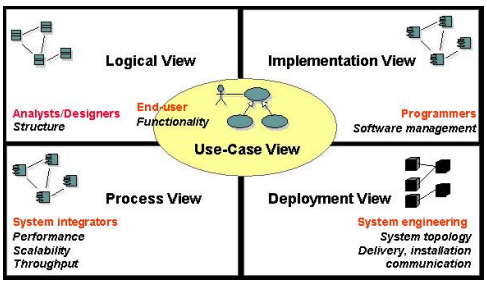


Esquemas de visões

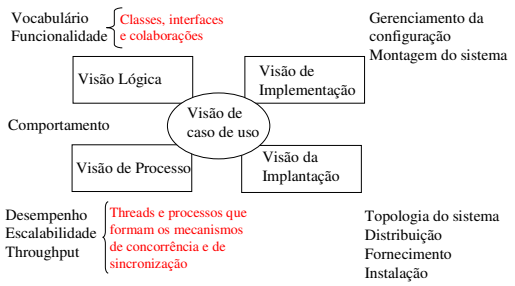
- São conjuntos de visões para agrupar formas diferentes de se observar a mesma coisa.
- Principais esquemas existentes:
  - Visões da OMT (Rumbaugh91)
  - Visões de Booch (Booch91
  - Visões do RM-ODP (Beitz97)
  - Visões de Zachman (Zachman97)
  - **Visões do RUP "4+1" (Krunchten00)**

    introduzido no artigo "The 4+1 View Model of Architecture", IEEE Software, 1995.

Esquema "4+1" (RUP)



Esquema "4+1" (RUP)



Visão Lógica (ou de Projeto)

- Analistas e desenvolvedores
- Ligada ao problema do negócio
- Independe de decisões de projeto
- Descreve e especifica a estrutura estática do sistema e as colaborações dinâmicas entre objetos via mensagens para realizarem as funções do sistema.
- Contém a coleção de pacotes, classes e relacionamentos.

Visão de Implementação (ou de Componente)

- Desenvolvedores
- Descrição da implementação dos módulos e suas dependências.
- Utilizada para saber como distribuir o trabalho de implementação e manutenção entre os membros da equipe considerando aspectos de reúso, subcontratação e aquisição de sw.

Visão de Processo (ou Concorrência)

- Trata a divisão do sistema em processos e processadores (propriedade não funcional)
- O sistema é dividido em linhas de execução de processos concorrentes (threads)
- Esta visão de concorrência deverá mostrar como se dá a comunicação e a concorrência destas threads.
- Considera questões de desempenho, confiabilidade, tolerância a falhas.

Visão de Implantação (ou Física, ou de Organização, Deployment View)

- Contém a parte física do sistema e a conexão entre suas sub-partes, interação hw-sw, com objetivo de colocar o sistema em operação.
- Visão de Organização: mostra a organização física do sistema, os computadores, os periféricos e como eles se conectam entre si.
- Esta visão será executada pelos desenvolvedores, integradores e testadores, e será representada pelo diagrama de implantação, pois considera o ambiente de desenvolvimento, teste e produção.

Visão de Caso de Uso (+1)

- Descreve o sistema como um conjunto de transações (funcionalidades) do ponto de vista dos atores externos (por eles desempenhadas)
- +1 porque mapeia o relacionamento das demais visões, mostrando como seus elementos interagem.

Plantas da Arquitetura

- É a representação gráfica de uma visão de arquitetura (utilizam-se diagramas UML)
  - **Visão lógica**. Diagramas de classe, diagrama de estado e diagramas de objetos,diagramas de pacotes, e de interação
  - **Visão de processos**. Diagramas de classes e diagramas de objetos (tarefa abrangente — processos e threads), estados
  - **Visão de implementação**. Diagrama de componentes
  - **Visão de implantação**. Diagramas de implantação
  - **Visão de casos de uso**. Os diagramas de casos de uso representam casos de uso, atores e classes. Os diagramas de seqüências representam objetos e suas colaborações.

Visões e Modelos UML

1. Visão Lógica

- Perfis usuários: arquitetos e designers
- Visão ESTRUTURAL
- Organização conceitual do SW em termos de:
  - Principais Camadas
  - Principais Componentes e Sub-sistemas
  - Principais Pacotes, frameworks, classes e interfaces
- As três categorias acima (Camadas, Componentes e Sub-sistemas e Pacotes) definem as sub-visões da Visão Lógica

Visão Lógica
Visão de Pacotes

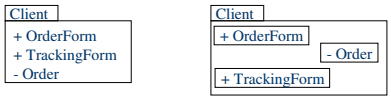
- Descreve como a solução será dividida em pacotes (ou namespaces).
- Uma boa visão de pacotes deve mapear a visão de componentes E a visão de camadas.
- Como documentar?
  - Seção normalmente composta de descrição textual das regras de criação, distribuição e nomenclatura de pacotes e elementos internos (sub-pacotes, classes, interfaces).
  - Diagramas de classes usando pacotes

Pacotes lógicos

- Um pacote lógico (ou módulo lógico) é um agrupamento lógico de classes e relações entre essas classes
  - divisão de um sistema em pacotes lógicos é uma divisão de responsabilidades
- Corresponde ao conceito de *package* em Java ou de *namespace* em C++ e C#
- Não confundir com empacotamento físico do software em arquivos de código fonte, executáveis, dll's, etc. (designados componentes em UML)
- Um pacote lógico pode atravessar vários arquvos
- Diagramas de pacotes lógicos utilizados para modelar a **arquitetura lógica** de um sistema de software (organização em módulos lógicos e especificação de interfaces e dependências entre módulos)

Conteúdo de um pacote

- Uma vez que representa um agrupamento, um pacote é em geral *dono* de diversos elementos: classes, interfaces, componentes, nós, colaborações, casos de uso, diagramas, e até outros pacotes
- Esses elementos podem ser indicados no interior do pacote, na forma de uma lista de nomes ou diagrama



Um pacote forma um *espaço de nomes*

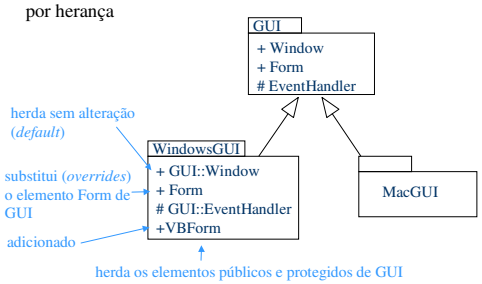
- classe *Order* do pacote *Client* é designada *Client::Order*

Dependências entre pacotes

- Dependência simples: uma alteração do pacote de destino afeta o pacote de origem (dependente) (informação útil para controlar alterações)
- Dependência com estereótipo **«access»**: o pacote de origem (dependente) acessa elementos exportados pelo pacote de destino (precisa de :: nos nomes)
- Dependência com estereótipo **«import»**: o pacote de origem (dependente) importa os elementos exportados pelo pacote de destino (não precisa de :: nos nomes)

Generalização de pacotes

- Usada para especificar famílias de pacotes relacionados por herança

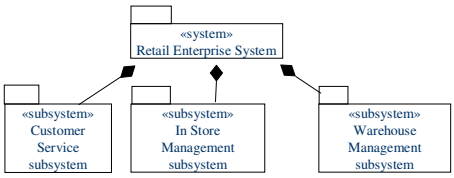


Estereótipos em pacotes

- «system» - pacote que representa o sistema completo que está a ser modelado (incluindo todos os modelos e elementos dos modelos)
- «subsystem» - pacote que representa uma parte independente de sistema completo que está a ser modelado; corresponde normalmente a um corte "vertical"
- «facade» (fachada) - pacote que constitui uma vista sobre outro pacote (não acrescenta funcionalidades, apenas apresenta de forma diferente)
- «framework» (infra-estrutura aplicacional) - pacote que representa um conjunto de classes abstractas e concretas concebido para ser estendido, implementando a funcionalidade típica de um determinado domínio de aplicação
- «stub» - pacote que serve como *proxy* para o conteúdo público de outro pacote
- «layer» - pacote que representa uma camada horizontal de um sistema

Composição de pacotes (1)

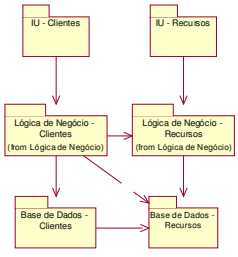
- Sub-pacotes podem ser indicados dentro do pacote-dono ou com relação de composição



Neste exemplo segue-se uma divisão vertical, por subsistemas!

Caso de estudo (biblioteca): divisão em camadas técnicas

Caso de estudo (biblioteca): divisão em camadas técnicas e áreas funcionais



Visão Lógica

Visão de Pacotes: Exemplo 1

- Camadas primeiro, componentes depois.
- Exemplo:
  - com.ibta.apresentacao.<componente>
  - com.ibta.negocio.<componente>
  - com.ibta.acessodados.<componente>
  - Onde <componente> pode ser:
    - pedido
    - produto
    - cliente
    - vendedor
    - seguranca

Essa estrutura é melhor quando for mais importante ter uma visão de camadas do que de componentes.

Visão Lógica

Visão de Pacotes: Exemplo 2

- Componentes primeiro, camadas depois.
- Exemplo:
  - com.ibta.pedido.<camada>
  - com.ibta.produto.<camada>
  - com.ibta.cliente.<camada>
  - com.ibta.vendedor.<camada>
  - com.ibta.seguranca.<camada>
  - Onde <camada> pode ser:
    - apresentacao
    - negocio
    - acesso a dados
    - entidades
    - businessdelegate
    - sessionfacade
    - ...

Essa estrutura é melhor quando for mais importante ter uma visão de componentes do que de camadas.

Visão Lógica

Visão de Camadas

- Esta seção descreve como o sistema está dividido em camadas (apresentação, negócio, etc), mostrando:
  - quais são as camadas
  - suas dependências
  - como se comunicam

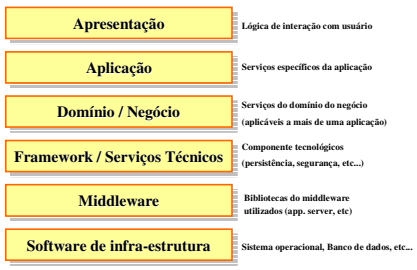
- Estrutura:
  - Incluir pelo menos um diagrama dos pacotes para cada camada e seus relacionamentos
  - Criar uma subseção para descrever cada camada separadamente, definindo suas responsabilidades e os principais serviços oferecidos
- Como modelar camadas: pacote com estereótipo <<layer>>

Visão Lógica

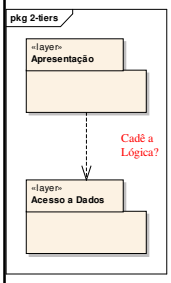
Visão de Camadas: Benefícios

- Problemas que as camadas visam eliminar:
  - Alto acoplamento: qualquer alteração em componentes do software tem alto impacto
  - Falta de clareza na atribuição de responsabilidades:
    - Lógica da aplicação "entrelaçada" com a lógica de apresentação: impossibilita reutilização de lógica de negócio entre clientes (desktop, browser, PDA, ...) distintos ou distribuição da lógica de negócio para outros nós
    - Serviços técnicos (pertencentes ao framework) "entrelaçados" com a lógica de negócio: dificuldade para reusar o framework técnico ou lógica de negócio genérica

Camadas: Exemplo Tradicional

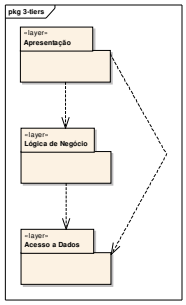


Visão de Camadas: 2 Camadas



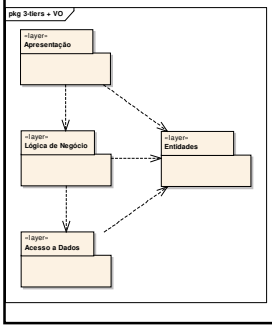
- Apresentação "Gorda"
  - Agrupamento de lógica de apresentação e de negócio (mais comum)
  - Típico de sistemas em plataformas do tipo Delphi ou VB
- Acesso a Dados "Gorda"
  - Agrupamento de lógica de negócio e acesso a dados.
  - Normalmente implementa a lógica em procedures na base de dados.
- Vantagens
  - Produtividade (para fazer a versão inicial da aplicação)
  - Performance
- Desvantagens
  - Baixa Escalabilidade
  - Dificuldade de Reúso
  - Dificuldade Manutenção

Visão de Camadas: 3 Camadas



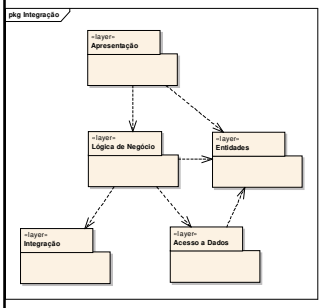
- Lógica de negócio separada tanto da camada de apresentação quanto da camada de acesso a dados.
- Evolução:
  - Isolamento da lógica de negócio.
    - Melhor escalabilidade
    - Melhor reúso da lógica de negócio
    - Maior facilidade de manutenção
- Desvantagem:
  - A dependência da camada de apresentação com Acesso a Dados permite (teoricamente) que a camada de apresentação acesse diretamente a base de dados (maior acoplamento, restrição arquitetural).

Visão de Camadas
3 Camadas + Entidades



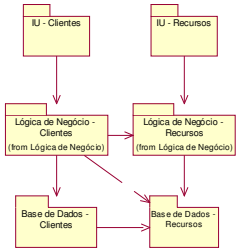
- Criação de uma nova "camada" horizontal de entidades do sistema (também chamados de Value ou Transfer Objects).
- Essas entidades carregam dados através de todas as camadas.
- Algumas arquiteturas não permitem que as entidades cheguem até a camada de apresentação. Utilizam, por exemplo, XML.
- Evolução:
  - Não existe dependência da camada de apresentação para a camada de acesso a dados.
  - Maior coesão na camada de acesso a dados.

Visão de Camadas
3 Camadas + Entidades + Integrações



- Criação de uma nova camada exclusiva para integração do sistema.
- Vantagens
  - Isolar todo o código de integração em uma única camada.
  - Ressaltar o código de integração
    - Impacto de mudanças
    - Suite de testes unitários
    - Divisão de equipe
    - Implementação de MOCKs (usar com parcimônia!)

Caso de estudo (biblioteca): divisão em camadas técnicas e áreas funcionais



Visão Lógica – Visão de Componentes

- Deve ilustrar todos os componentes relevantes construídos, comprados ou re-utilizados da solução.
- Componentes mais críticos podem ser descritos com maior nível de detalhe:
  - papel na solução final
  - motivação de criação do componente
- Estrutura
  - Deve-se criar um diagrama de componentes mostrando a relação de dependência entre eles. Os componentes ilustrados no diagrama devem ser descritos.

Componentes

- Um componente é uma parte **física** (feita de bits e bytes) e **substituível** de um sistema, que proporciona a realização de um conjunto de interfaces
  - Podem-nos interessar diferentes ambientes: desenvolvimento, produção, testes, ...
- Exemplos: executáveis, bibliotecas, tabelas, ficheiros, documentos
- Um componente representa um empacotamento físico de elementos relacionados logicamente (normalmente classes)
- Notação: caixa com *tabs*

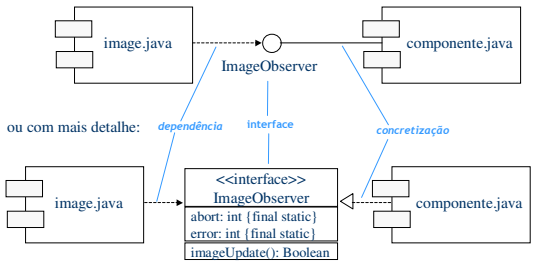
36

Interfaces

- Relação de dependência: um componente pode usar uma ou mais interfaces
  - Diz-se que essas interfaces são importadas
  - Um componente que usa outro componente através de uma interface bem definida, não deve depender da implementação (do componente em si), mas apenas da interface

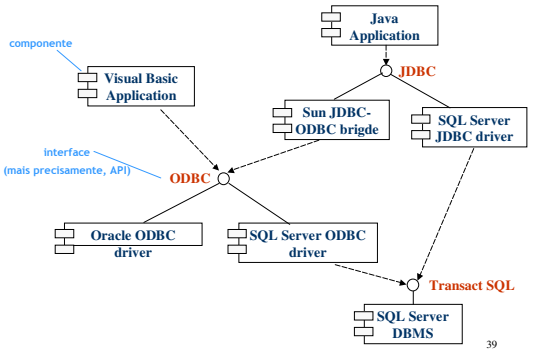
37

Interfaces – Exemplo 1



38

Interfaces – Exemplo 2



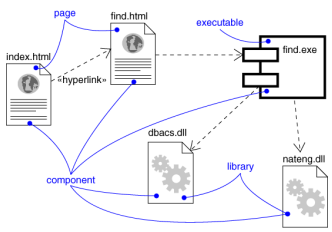
39

Tipos de dependências entre componentes

- Dependências simples:
  - entre arquivo com código fonte, para controle de alterações
  - entre executáveis e/ou bibliotecas, para gestão de configurações e dependências
  - entre executáveis ou bibliotecas e tabelas ou documentos de ajuda por eles usados
- Dependências estereotipadas:
  - estereótipo «hyperlink» - entre páginas html ou páginas html e executáveis
  - estereótipo «trace» - entre versões consecutivas do mesmo tipo de componente
- Não esquecer que é melhor depender das interfaces do que das implementações

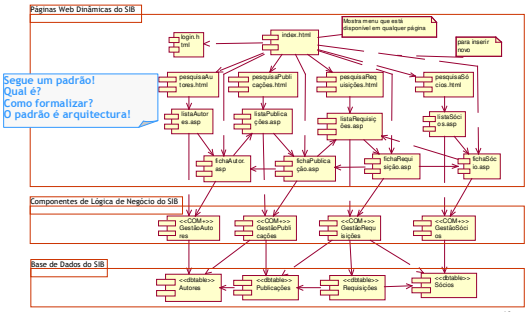
40

Dependências - Exemplo



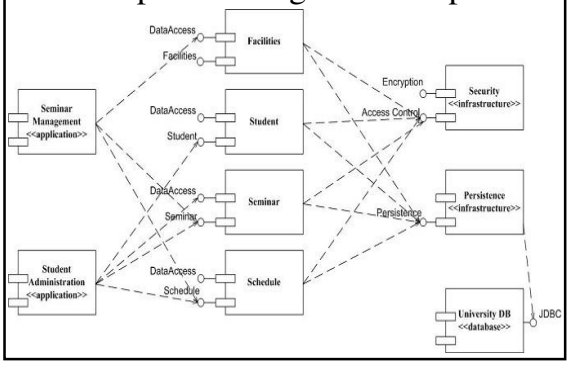
41

Caso de estudo (biblioteca)

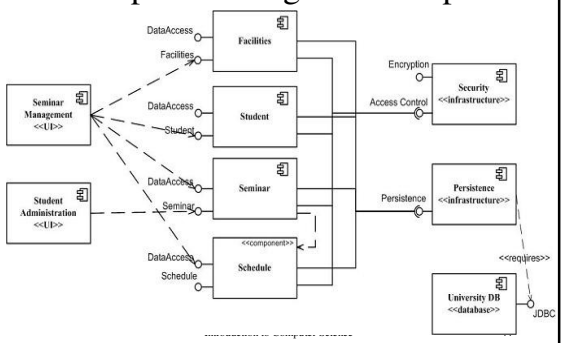


42

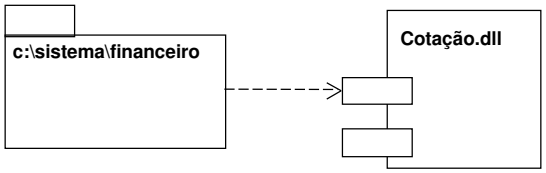
Component Diagram Example



Component Diagram Example

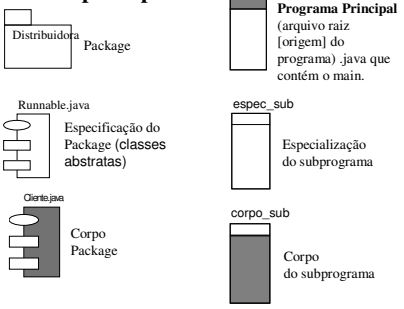


Dependência entre pacotes e componentes

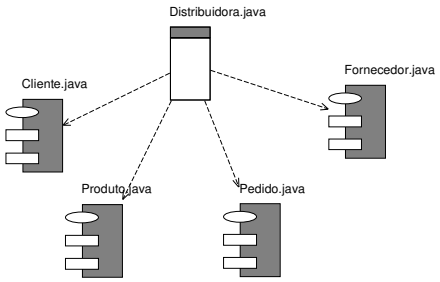


Estereótipos

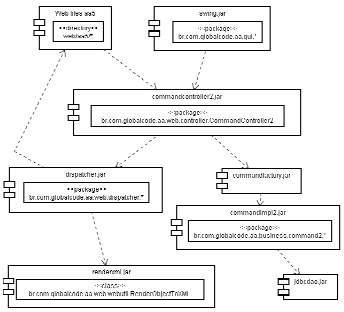
➲**É composto por:**



Estereótipos - Exemplo



Visão Lógica
Visão de Componentes: Exemplo



Exercício

- Dado o modelo desenvolvido para a loja de informática
- Crie a visão de componentes (diagrama de componrentes)
- Crie visão de camadas (Não existe diagrama de camadas, usar um diagrama de classes com pacotes para camadas)
- Crie um diagrama de classes com os principais pacotes da aplicacao

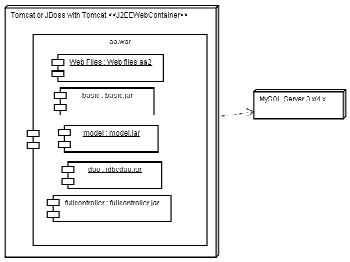
2. Visão de Implementação

- Perfis usuários: implementadores
- Representa o mapeamento da visão lógica para o código-fonte e diretórios da aplicação.
- Descrever um resumo da estrutura de empacotamento do código, na plataforma alvo.
  - Por exemplo: divisão em DLL's, JAR's, EAR's, WAR's, arquivos de configuração, etc...
- Estrutura:
  - Diagrama de implantação com os componentes que ilustram o mapeamento lógico-físico da aplicação.

Visão de Implementação Exemplo Texto

- MyApp.ear - *Ear da Aplicação*
- Presentation.war - *Módulo Web*
- \WEB-INF
- spring.xml - *Arquivo de config. do Spring*
- \Componentes - *Pasta de componentes*
- Component1.jar - *Componente 1*
- Component1.jar - *Componente 2*
-

Visão de Implementação usando um diagrama de implantação



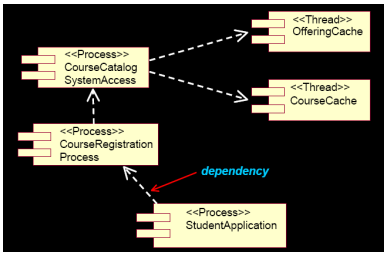
3. Visão de Processos

- Perfis usuários: arquitetos e implantadores
- Visão DINÂMICA/COMPORTAMENTAL
- Descreve os principais processos e threads da solução:
  - Suas responsabilidades e relacionamentos
  - Alocação para os elementos estruturais (descritos na visão lógica)
- Pode ser utilizada para documentar os processos batch da aplicação.
- Foco em:
  - Análise de concorrência, performance, throughtput e escalabilidade

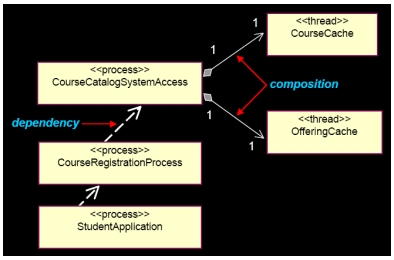
Visão de Processos

- Estrutura:
  - Diagrama de classes e interação UML ilustrando os principais processos, threads e suas interações.
  - Importante ilustrar:
    - Pontos de disparo e sincronização de processamento
    - Protocolos de comunicação

Modelando Processos...



Modelando Processos (II)



Visão de Processos

- Para processamento online, nas plataformas distribuídas atuais (J2EE e .Net), parte do que se pretende mapear e dar visibilidade a partir desta visão é contemplado pela plataforma/infraestrutura.
  - Por exemplo: a criação de threads e a distribuição de processamento através de objetos distribuídos (invocação de processamento remoto) são gerenciados pelos servidores de aplicação.
- No entanto, em casos de processamento distribuído entre processos distintos (processamento online e batch interagindo, processamento em servidor de aplicação J2EE invocando outro processo – C++, por exemplo - via sockets), vale ilustrar esta interação através desta visão.

4. Visão de Implantação (Deployment)

- Perfis usuários: arquitetos, implantadores e equipe de infraestrutura
- Visão ESTRUTURAL
- Descreve:
  - a implantação física dos processos e componentes para os nós de processamento/infra-estrutura
  - as ligações entre os nós de infra-estrutura
- Estrutura:
  - Diagrama de implantação (deployment diagrams)
  - Diagramas de infra-estrutura
  - Descrição dos nós de infra-estrutura (formato tabular) e rede

Visão de Implantacão (Deployment)

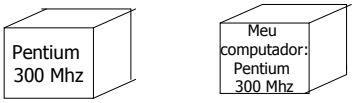
Função	x	Fabricante	Características	Software instalado
Web Server	1	Taiwan International	Pentium IV 1.0GHz, 256MB RAM, Storage interno 10GB	IIS 5.0
Base Dados – Web	1	Taiwan International	Pentium IV 1.0 GHz, 1.0GB RAM, Storage interno 1000GB	SQL Server 2000, Aplicativos VB 6 (batches de consolidação Web – Corporativo)
Banco Dados – Corp.	1	Moon Microsystems	Moon Thunder Station III, 10 GB RAM, Storage interno 3000GB	Oracle 9i
Estações	20	Xing Ling Systems	Pentium III 500MHz, 256MB RAM	Aplicativos VB 6

Diagramas de Implantação

- Mostram a organização do hw e a ligação do sw com os dispositivos físicos (computadores e periféricos)
- Trata-se de um gráfico de nós conectados por associações de comunicação.
- Cada nó pode conter instâncias de componentes.

Nós

- Representa um recurso computacional com pelo menos memória e capacidade de processamento.
- Pode-se ter uma instância desse tipo usando *Nomedonó: Tipodenó*



Nós - Notação

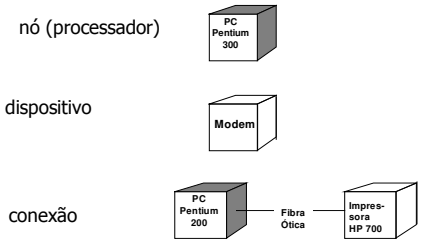
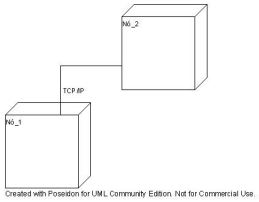
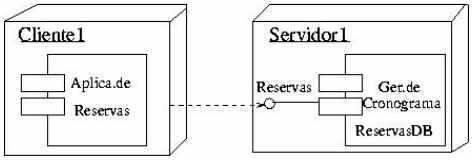


Diagrama de Implantação

- Como usar
  - Contém nós (software e hardware) e conexões (meio de comunicação, TCP/IP, JNI, Http, por exemplo).



Diagramas de Implantação



Diagramas de Implantação

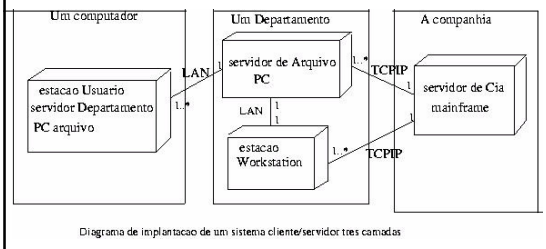
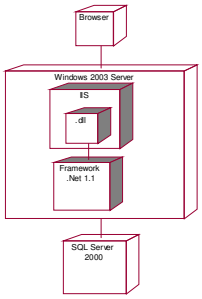
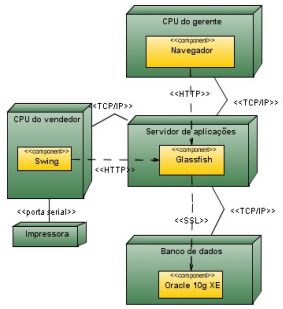


Diagrama de implantacao de um sistema cliente/servidor tres camadas

Visão de Implantação: Exemplo Diagrama de Implantação



Visão de Implantação: Exemplo Diagrama de Implantação



Exercício

- Dado o modelo desenvolvido para a loja de informática
- Crie a visão de implantacão do sistemas supondo que os sistema seja web ou cliente servidor
- Utilize um diagrama de deployment UML

(+1) Visão de Casos de Uso – o +1

- Perfil usuário: arquitetos e gerentes de projetos
- Objetivo:
  - Descrição dos casos de uso arquiteturalmente mais relevantes e seus requisitos não-funcionais
  - Em outras palavras, os UC's que, quando implementados exercitam o maior número de elementos arquiteturais possíveis

Visão de Caso de Uso

- Por que é o +1 de 4+1 ?
  - Por não se tratar de uma visão com foco arquitetural, mas sim, uma visão com foco gerencial (minimização de riscos, amadurecimento da arquitetura)
  - Além disso, é uma visão cuja implementação permeia todas as visões
    - Ou seja, para implementar e implantar os UC's da Visão de UCs, será necessário conhecimento e que já estejam pré-definidas todas as outras visões.

Visão de UC - Estrutura

- Uma tabela contendo o nome do UC e os aspectos arquiteturais validados.
- Estes podem ser: Requisitos não-funcionais, Mecanismos Arquiteturais, Produtividade

Nome do UC	Motivo da Escolha
Login	Valida mecanismo de autenticação. Prepara alicerce de construção para a próxima fase do projeto.
Enviar notas fiscais para o SAP	Valida integração com o SAP. Permite validação de requisito não-funcional de performance na integração com SAP.
Capturar arquivo	Valida robustez do sistema. Permite validar tamanho máximo de arquivo suportado pelo sistema.
Relatório de vendas	Exercita mecanismo de relatórios. Gera implementação padrão para os implementadores → produtividade.

Armadilhas da Escolha

- Não ter como foco os seguintes pontos:
  - minimização dos riscos técnicos, arquiteturais e de produtividade
  - cobertura e teste da arquitetura proposta
- Escolher os UCs para ter "alguma coisa para mostrar" no final da elaboração (efeito "mini-construção")
- Escolher o caso de uso mais complicado do sistema e isso esticar demais a construção
  - apesar de fazer sentido, vai ser difícil justificar para gerentes e sponsors
  - possíveis soluções: fazer uma parte relevante do UC apenas ou deixá-lo para início de construção (antecipando ao máximo)
- Escolher casos de uso "por escolher". O propósito é bem claro → validar a arquitetura. Não precisa atingir um número mínimo de UCs para validação.

Seleção de visões

- Quais são as visões arquiteturais relevantes para o sistema sendo desenvolvido?