Engenharia de Software III

Aula 3
Padrões e Diagramas Arquiteturais

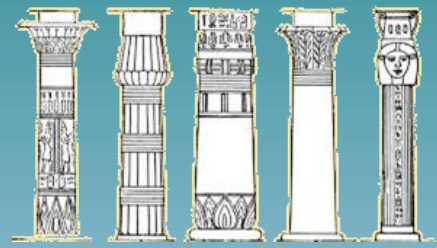
l.bertholdo@ifsp.edu.br

Conteúdo

- Padrões e Estilos Arquiteturais
  - Divisão em Camadas
  - Repositório
  - Cliente-Servidor
  - Dutos e Filtros (ou Fluxo de Dados)
- Diagramas Arquiteturais da UML
  - Diagrama de Comunicação
  - Diagrama de Pacotes
  - Diagrama de Componentes
  - Diagrama de Implantação

l.bertholdo@ifsp.edu.br

Padrões e Estilos Arquiteturais



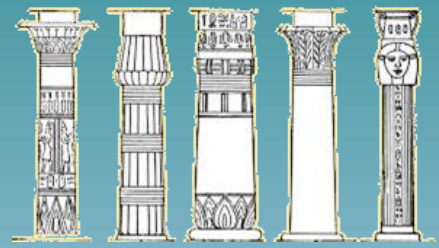
- Atualmente, a ideia de padrões como forma de apresentar, compartilhar e reusar o conhecimento sobre sistemas de software é amplamente usada. Hoje existem, por exemplo, padrões para estruturar, codificar e testar softwares, entre outros.

- Os **padrões de arquitetura** foram propostos na década de 1990 sob o nome de “estilos de arquitetura”. Na prática, hoje não há uma distinção significativa entre os dois termos (“padrão” e “estilo”).

*Um **padrão de arquitetura** é uma descrição abstrata e estilizada de boas práticas experimentadas e testadas em diferentes sistemas e ambientes. Em outras palavras, ele visa descrever uma organização de sistema que foi bem-sucedida em projetos anteriores.*

l.bertholdo@ifsp.edu.br

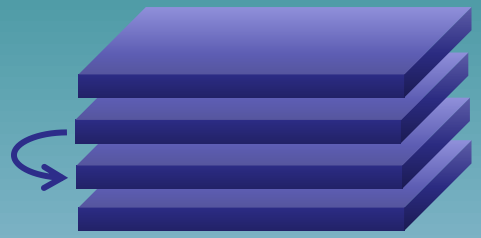
Padrões e Estilos Arquiteturais



- Entre outras questões, um padrão de arquitetura ajuda a definir uma **abordagem base** para se estruturar o sistema e uma **estratégia** para controlar o funcionamento dos seus componentes.
- A descrição de um padrão de arquitetura normalmente inclui:
  - Nome do padrão;
  - Breve descrição conceitual (textual e gráfica);
  - Exemplo do tipo de sistema em que o padrão pode ser usado;
  - Situações e contextos em que o padrão geralmente é usado;
  - Suas vantagens e desvantagens.

l.bertholdo@ifsp.edu.br

Divisão em Camadas



- As noções de **separação** e **independência** entre componentes de software são fundamentais para o projeto de arquitetura, porque permitem que alterações sejam mais autocontidas.

- Na **divisão em camadas**, a arquitetura do sistema é organizada em níveis separados, e cada nível só depende dos recursos e serviços oferecidos pelo nível imediatamente abaixo dele.

*Nessa abordagem são definidas várias camadas, cada uma realizando operações que progressivamente se tornam mais próximas do conjunto de instruções de máquina. Na **camada superior**, os componentes atendem às operações da interface do usuário. As **camadas intermediárias** fornecem as funcionalidades da aplicação. Na **camada inferior**, os componentes fazem a interface com o sistema operacional e outros sistemas.*

Divisão em Camadas

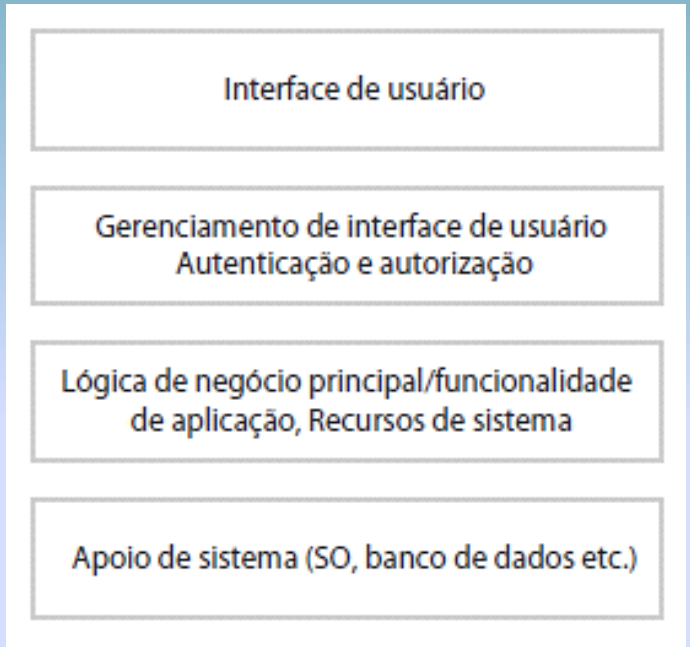
- Descrição do Padrão de Arquitetura em Camadas

Descrição	Organiza o sistema em camadas, onde cada camada tem uma função associada. Uma camada fornece serviços à camada acima dela, de modo que as camadas mais baixas representam os serviços que provavelmente serão usados em todo o sistema.
Exemplo	Sistema para compartilhamento de livros e documentos com direitos autorais entre diferentes bibliotecas universitárias.
Quando é usado	• Quando o desenvolvimento está distribuído por várias equipes, sendo cada equipe responsável por uma camada diferente; • Quando há um requisito de proteção multinível.
Vantagens	Permite a substituição de camadas inteiras, bastando adaptar as interfaces entre as camadas. Recursos redundantes, como autenticação por exemplo, podem ser fornecidos em cada camada para aumentar a confiabilidade do sistema.
Desvantagens	Na prática, pode ser difícil ter uma separação clara entre as camadas, e uma camada de alto nível pode ter de interagir diretamente com camadas de baixo nível, em vez de interagir com a camada imediatamente abaixo dela. O desempenho pode ser um problema devido aos vários níveis em que cada requisição de serviço é processada.

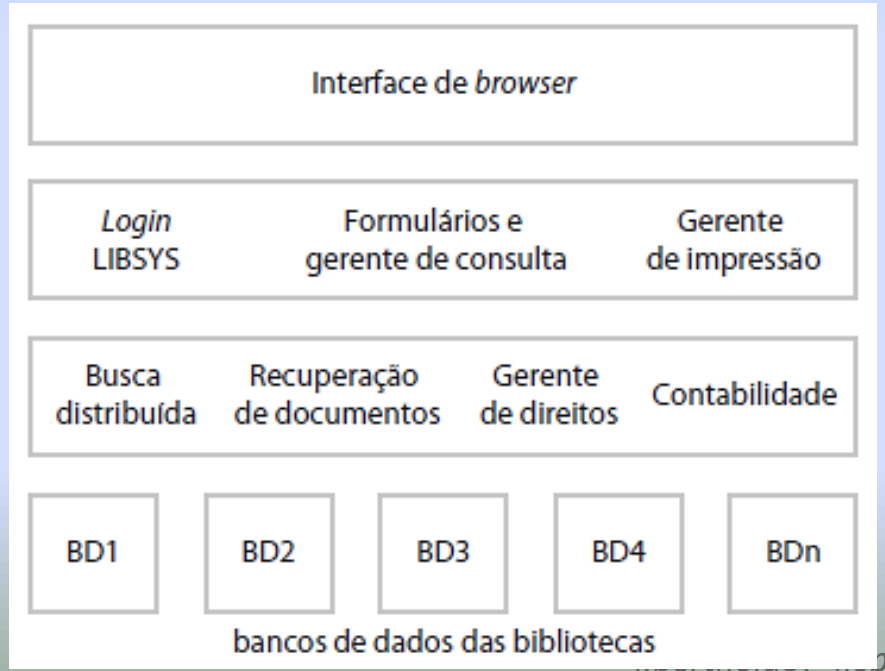
l.bertholdo@ifsp.edu.br

Divisão em Camadas

Visão Conceitual

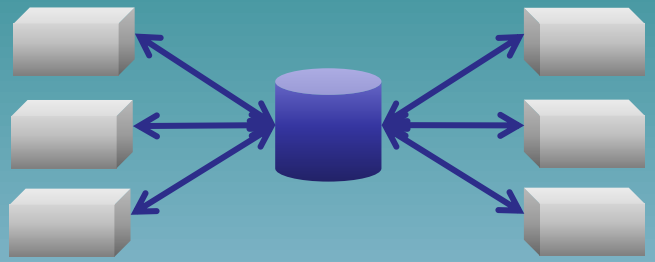


*A **camada inferior** inclui softwares de apoio ao sistema (sistema operacional e de banco de dados). A **camada de aplicação** inclui os componentes relacionados a funcionalidades e componentes utilitários usados pela aplicação. A **terceira camada** controla os eventos de interface de usuário e a autenticação de usuário. Por fim, a **camada superior** provê os recursos de interface de usuário.*



Exemplo: Sistema LIBSYS para compartilhar livros e documentos com direitos autorais entre bibliotecas

Repositório



- O **padrão repositório** descreve como um conjunto de componentes, que interagem entre si, podem compartilhar dados.

- Sistemas que usam grandes quantidades de dados são organizados **em torno de um banco de dados ou repositório compartilhado**. Esse modelo é adequado para aplicações nas quais os dados são gerados por um componente e usados por outro.

*A organização de features do sistema em torno de um repositório constitui uma maneira eficiente de **compartilhar dados**, pois elimina a necessidade de transmitir dados diretamente entre os componentes destas features. No entanto, os componentes, independentemente de suas particularidades, **devem ser compatíveis** com o modelo de repositório de dados estabelecido.*

l.bertholdo@ifsp.edu.br

Repositório

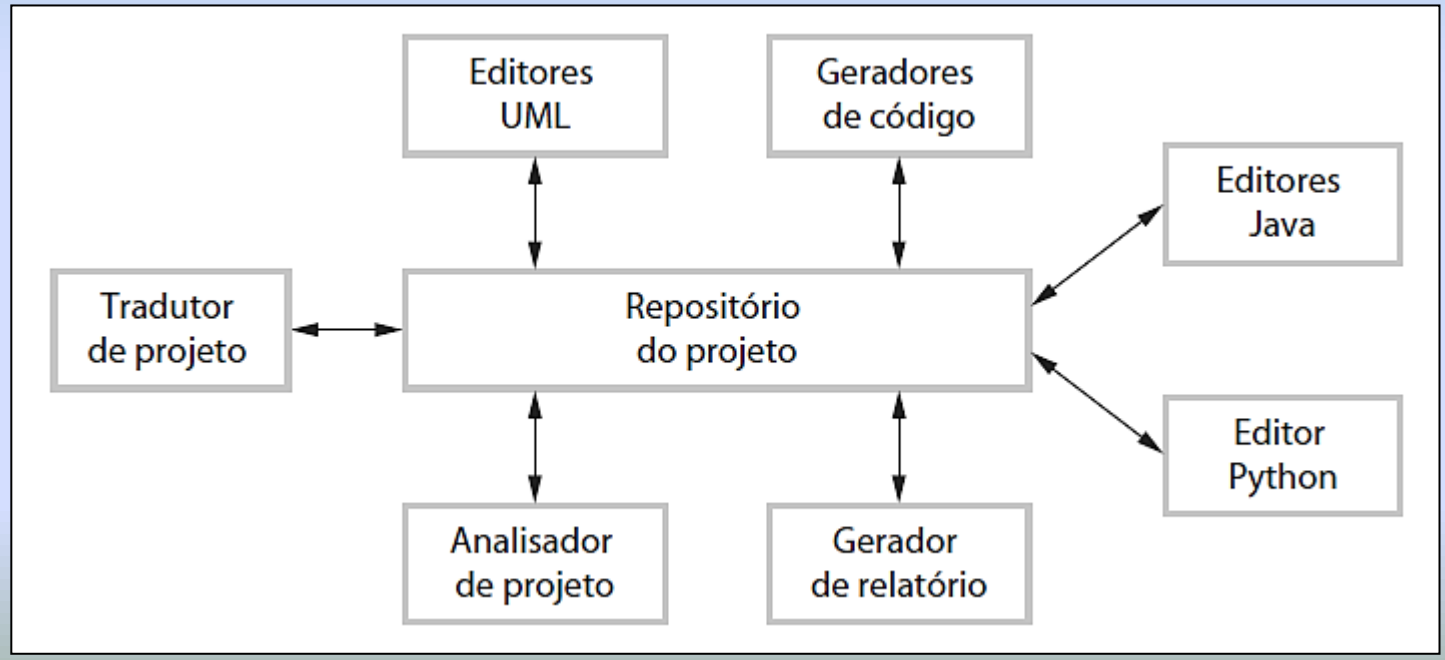
- Descrição do Padrão de Arquitetura de Repositório

Descrição	Todos os dados em um sistema são gerenciados em um repositório central, acessível a todos os componentes do sistema. Os componentes não interagem diretamente, apenas por meio do repositório.
Exemplo	IDE em que os componentes usam um repositório com dados de projetos de software.
Quando é usado	• Quando o sistema gera grandes volumes de informações, as quais precisam ser armazenadas por longos períodos; • Quando partes do sistema são dirigidas a dados, nos quais a inclusão dos dados no repositório dispara uma ação ou aciona outra *feature*.
Vantagens	Os componentes podem ser independentes, ou seja, não precisam saber da existência uns dos outros. As alterações feitas no repositório para um componente podem propagar-se para todos os outros. Os dados podem ser gerenciados de forma mais consistente, pois tudo está em um só lugar.
Desvantagens	Problemas no repositório podem afetar todo o sistema. Pode não ser eficiente organizar a comunicação via repositório entre alguns componentes.

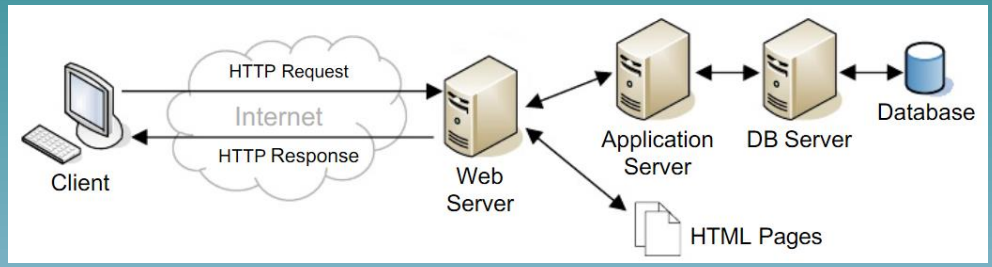
Repositório

A IDE inclui diversas features para apoiarem o desenvolvimento. O repositório, nesse caso, pode ser um ambiente de controle de versões que gerencia as alterações de software. Nessa abordagem, o repositório é passivo e o controle é de responsabilidade dos componentes que usam o repositório.

Exemplo: Arquitetura de repositório para uma IDE



Cliente-Servidor



- Modelo para sistemas distribuídos onde as funcionalidades são disponibilizadas por servidores como um conjunto de serviços, os quais são acessados e usados por computadores clientes.
- Este padrão é organizado com base em três componentes principais:
  - Um **conjunto de servidores** que oferecem serviços: servidores de páginas web, de aplicação, de banco de dados, de impressão etc;
  - Um **conjunto de clientes** que podem chamar os serviços oferecidos pelos servidores;
  - Uma **rede** que permite aos clientes acessar esses serviços, por meio de protocolos de comunicação.

*A arquitetura cliente-servidor também oferece o benefício da **separação** e **independência**, pois reduz o impacto em outras parte dos sistemas quando os serviços e servidores precisam ser modificados.*

l.bertholdo@ifsp.edu.br

Cliente-Servidor

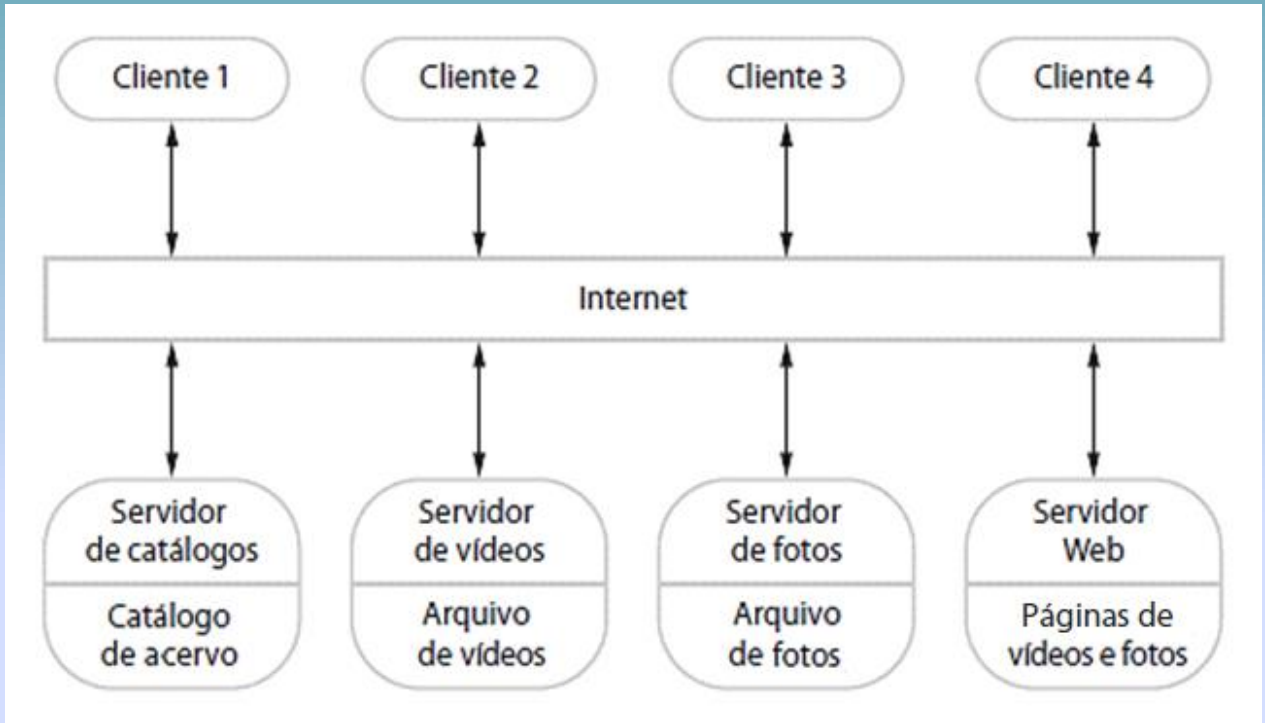
- Descrição do Padrão de Arquitetura Cliente-Servidor

Descrição	Em uma arquitetura cliente-servidor, o sistema está organizado em serviços. Cada serviço é prestado por um servidor, e os clientes acessam os servidores para fazer uso destes serviços.
Exemplo	Acervo de fotos e vídeos, organizado como um sistema cliente-servidor.
Quando é usado	• Quando os dados em um banco de dados compartilhado precisam ser acessados a partir de uma série de locais; • Quando a carga de processamento do sistema varia muito, uma vez que os servidores podem ser redundantes.
Vantagens	Os servidores centralizam serviços. Uma funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada em todos os servidores.
Desvantagens	Cada serviço está localizado em um servidor, tornando-se vulnerável a ataques ou falhas que ocorrerem no servidor. O desempenho e o funcionamento do sistema podem ser imprevisíveis, pois dependem fortemente da rede.

Cliente-Servidor

A aplicação cliente é, simplesmente, uma interface de usuário, acessada via browser para usar os serviços.

Exemplo: Sistema multiusuário baseado na Internet para fornecer filmes e fotos

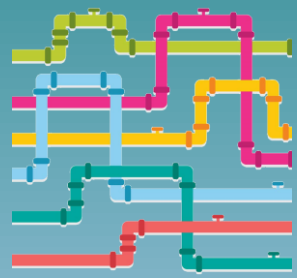


Nesse sistema, os vídeos devem ser comprimidos, pois precisam ser transmitidos de forma rápida e em sincronia, mas em resolução relativamente baixa. Já as fotos devem ter resolução alta e não são comprimidas. Por isso, é conveniente manter vídeos e fotos em servidores separados.

Os servidores web e de catálogos devem ser capazes de lidar com uma variedade de consultas e fornecer aos usuários dados sobre os vídeos e fotos, além de oferecer recursos para venda destes.

l.bertholdo@ifsp.edu.br

Dutos e Filtros (ou Fluxo de Dados)



- O **padrão de dutos e filtros** considera um conjunto de componentes, denominado **filtros**, conectados por **dutos** que transmitem dados de um componente para o seguinte.
- Um filtro é projetado para esperar uma **entrada de dados** em um formato específico, **transformar** estes dados e produzir uma **saída de dados** apropriada para o filtro seguinte.
- Cada filtro trabalha de modo independente e não precisa conhecer o funcionamento interno dos filtros que se encontram antes e depois dele.

Os componentes são chamados de "filtros" porque a transformação que eles realizam "filtra" ou "prepara" os dados de entrada para o componente seguinte.

l.bertholdo@ifsp.edu.br

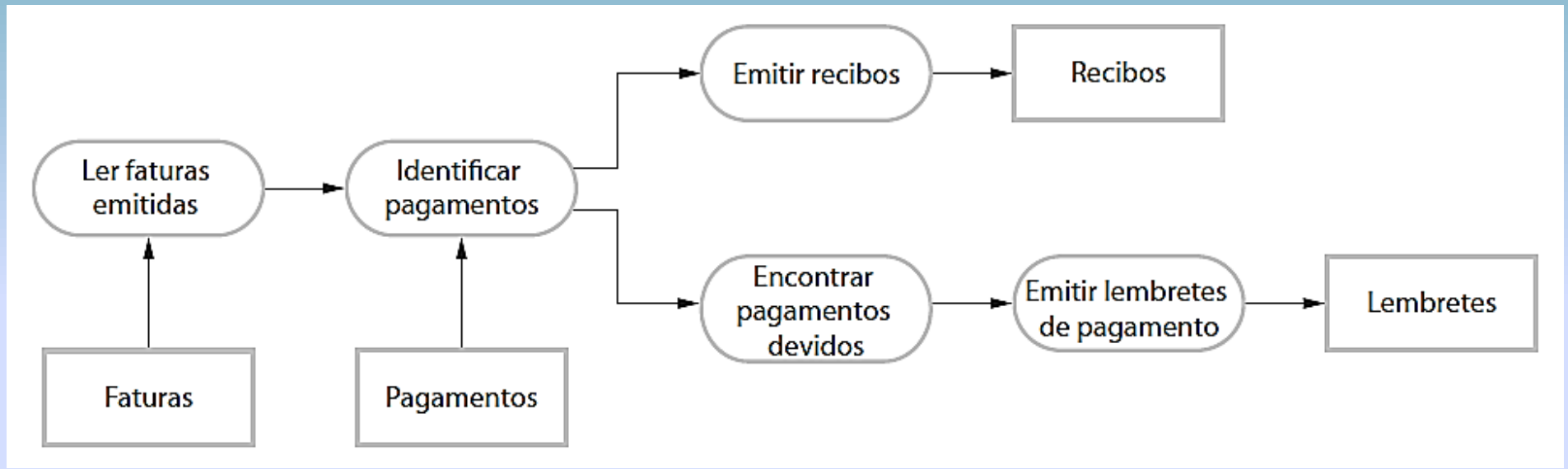
Dutos e Filtros (ou Fluxo de Dados)

- Descrição do Padrão de Arquitetura de Dutos e Filtros

Descrição	O processamento dos dados é organizado de modo que cada componente (filtro) seja autocontido e realize um determinado tipo de transformação de dados, os quais fluem (como em um duto) de um componente para outro.
Exemplo	Sistema para processamento de faturas.
Quando é usado	Quando a aplicação realiza processamentos em que os dados de entrada devem ser transformados em etapas separadas e sequenciais.
Vantagens	Fácil de entender, pois o estilo de *workflow* corresponde à estrutura de muitos processos de negócios. Permite o reúso dos processamentos dos filtros. Pode ser implementado tanto como um sistema sequencial quanto concorrente.
Desvantagens	Cada processamento deve analisar o formato de suas entradas e gerar as saídas para um formato acordado, tarefas adicionais que podem sobrecarregar o sistema.

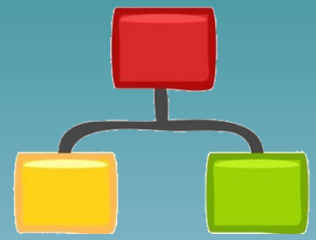
Dutos e Filtros (ou Fluxo de Dados)

Exemplo: Sistema para processamento de faturas



*O **primeiro filtro** lê as faturas emitidas para os clientes. Uma vez por semana, o **filtro seguinte** compara os pagamentos realizados com as faturas emitidas. Um **terceiro filtro** emite um recibo para cada fatura paga. Em paralelo, **outro filtro** detecta as faturas que não foram pagas dentro do prazo e, na sequência, um **filtro** emite um lembrete para essas faturas.*

Diagramas Arquiteturais da UML



- A UML (*Unified Modeling Language*) oferece diagramas com as mais diferentes finalidades. Entre eles, existem alguns comumente usados para representar as visões de arquitetura de um projeto. São eles:
  - **Diagrama de Comunicação:** Usado para representar a **visão de processo** da arquitetura.
  - **Diagrama de Pacotes:** Usado para representar a **visão lógica** da arquitetura.
  - **Diagrama de Componentes:** Usado para representar a **visão de desenvolvimento (ou implementação)** da arquitetura.
  - **Diagrama de Implantação:** Usado para representar a **visão física** da arquitetura.

l.bertholdo@ifsp.edu.br

Diagrama de Comunicação

- Este diagrama salienta os **vínculos de dados** entre os **participantes de uma interação**. Na UML 1.X, era chamado de **diagrama de colaboração**.
- Um outro tipo de diagrama de interação, o **diagrama de sequência**, desenha cada participante como uma linha de vida e mostra a sequência de mensagens na direção vertical.
- Em vez disso, o **diagrama de comunicação** permite posicionar os participantes livremente, desenhar vínculos para mostrar como eles se conectam e usar números para mostrar a sequência de mensagens.

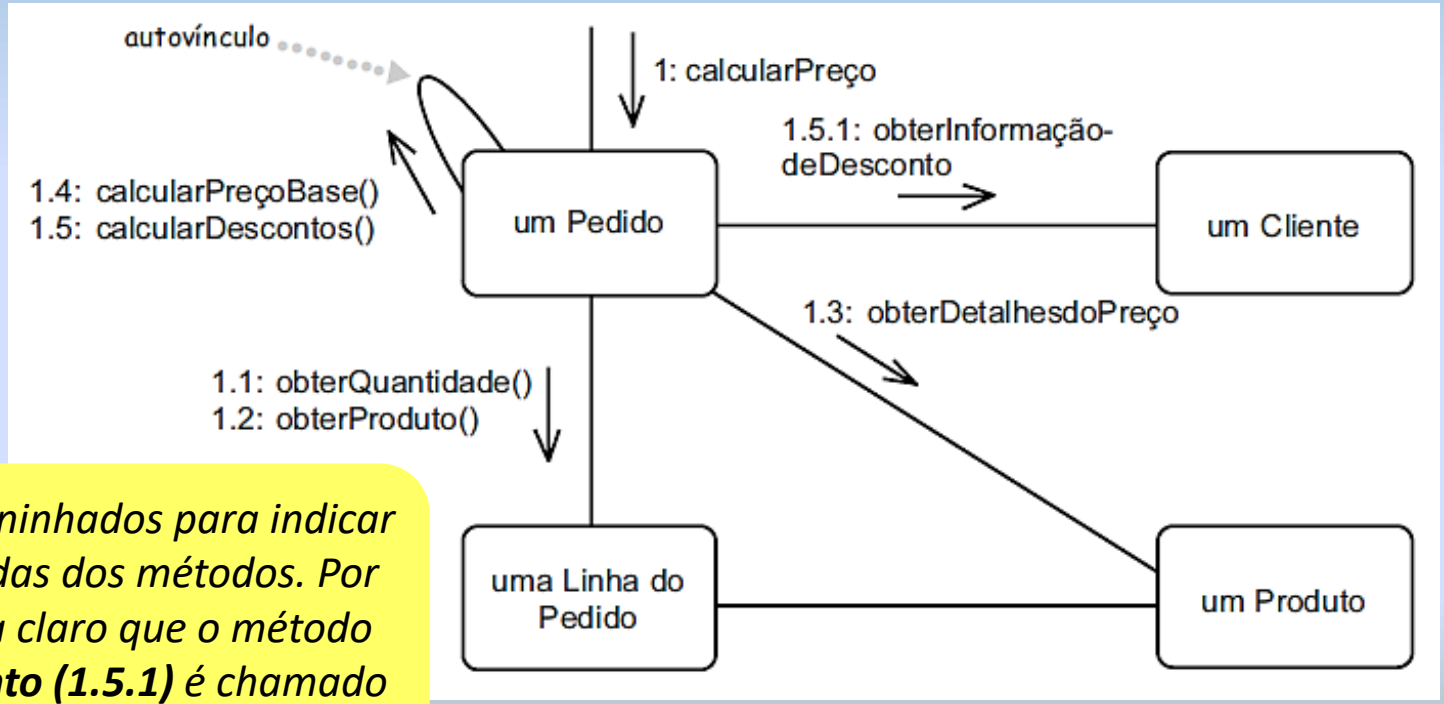
*Embora seja um tipo de **diagrama de interação**, e não um tipo **estrutural**, este diagrama é útil para a arquitetura, pois apresenta a forma como as entidades do sistema se comunicam em tempo de execução.*

l.bertholdo@ifsp.edu.br

Diagrama de Comunicação

*As linhas do diagrama são os **vínculos** entre os **participantes**, representados pelas caixas.*

*As setas indicam a **direção** dos vínculos. Por exemplo, um **pedido** obtém os dados de um produto para uma **linha (item) do pedido**. Embora exista o vínculo entre uma **linha do pedido** e um **produto**, não há chamadas de métodos entre os dois.*



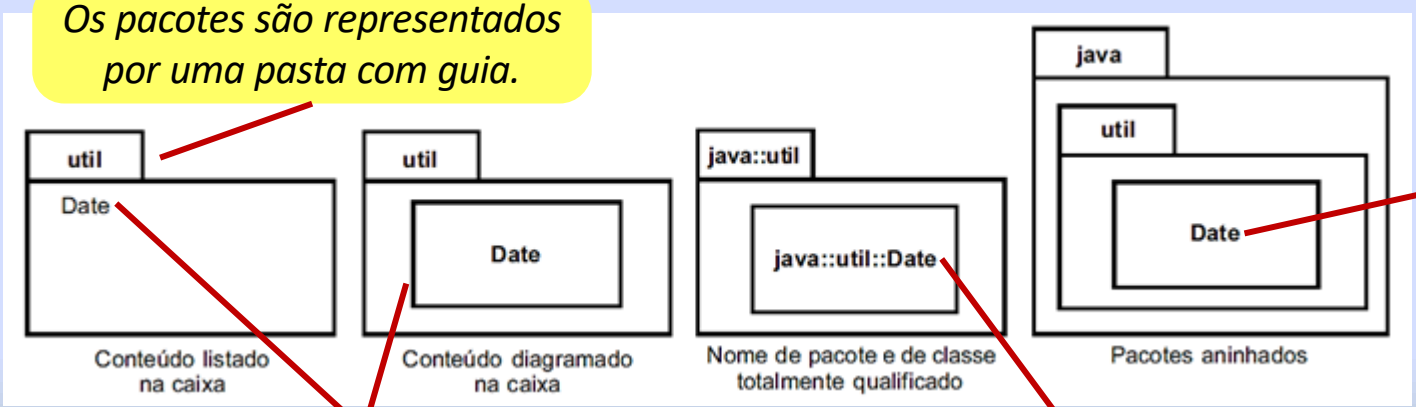
*Os números decimais são aninhados para indicar como são feitas as chamadas dos métodos. Por exemplo, no diagrama fica claro que o método **obterInformaçãodeDesconto (1.5.1)** é chamado dentro do método **calcularDesconto (1.5)**.*

l.bertholdo@ifsp.edu.br

Diagrama de Pacotes

- Sistemas grandes podem ter centenas de classes. O **diagrama de pacotes** é usado principalmente para ajudar a estruturar estes sistemas, por meio do **agrupamento de classes em pacotes**.
- Nesse diagrama, cada classe deve ter um nome exclusivo dentro do pacote a que pertence. Os pacotes podem ser membros de outros pacotes, de modo que os pacotes são divididos em **subpacotes** que possuem seus próprios subpacotes e assim por diante.

Os pacotes são representados por uma pasta com guia.



É possível mostrar todos os detalhes de uma classe, incluindo até um diagrama de classes dentro do pacote. Listar simplesmente os nomes faz sentido quando se deseja apenas indicar quais classes estão em quais pacotes.

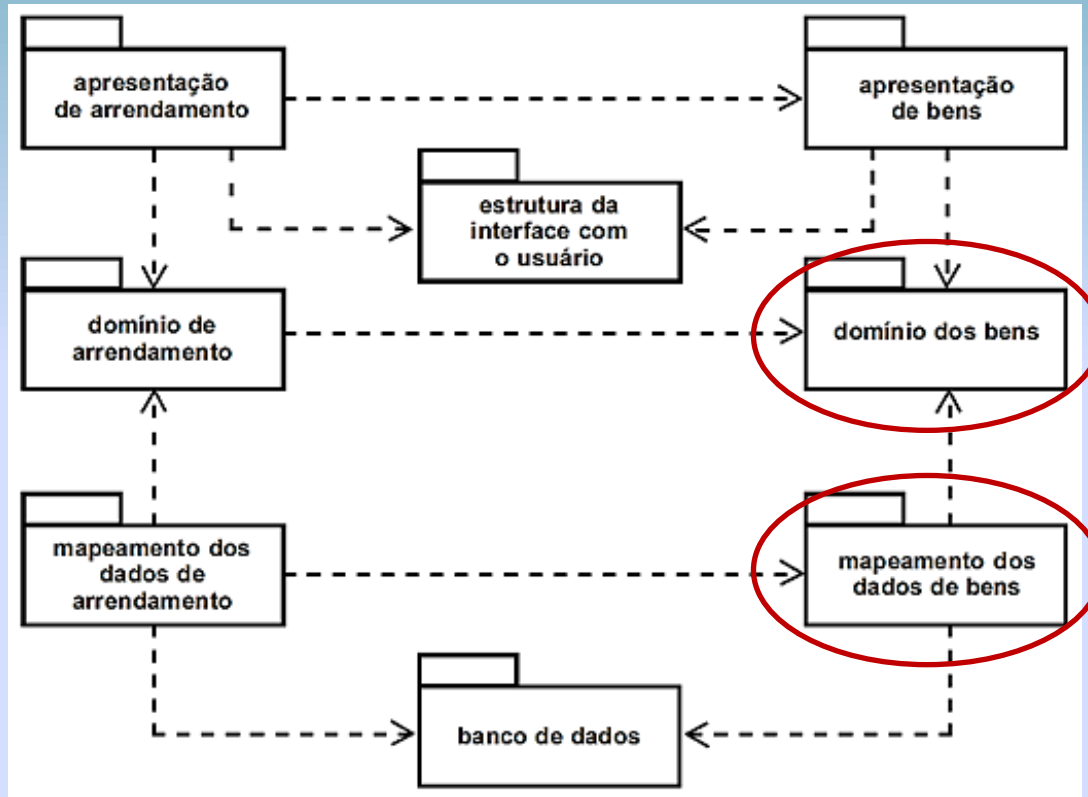
Pode-se mostrar simplesmente o nome do pacote ou mostrar também suas classes.

Pode-se usar nomes totalmente qualificados (com dois pontos duplos) ou apenas nomes simples.

Diagrama de Pacotes

O diagrama também mostra as dependências entre uma ou mais classes dos pacotes. Por exemplo, o pacote ***apresentação de bens*** *(cliente), que agrupa classes de interface de usuário, é dependente do pacote* ***domínio de bens*** *(fornecedor), que agrupa classes que representam atributos e métodos de objetos do tipo "bem".*

Quanto mais dependentes um pacote possui, mais estável a ***interface*** *dele precisa ser, pois qualquer alteração nessa interface será propagada para todos os seus pacotes dependentes. Assim, o pacote do* ***domínio dos bens*** *precisa de uma interface mais estável do que o pacote de* ***mapeamento de dados de bens****. É por isso que pacotes mais estáveis geralmente demandam mais interfaces e classes abstratas.*



Se uma classe no pacote do ***domínio de bens*** *muda, talvez tenhamos que alterar as classes dentro do pacote do* ***domínio de arrendamento****. Mas essa alteração não se propaga necessariamente para a* ***apresentação de arrendamento****, exceto se o domínio de arrendamento também precisar mudar sua interface.*

Diagrama de Componentes

- Este diagrama é usado para mostrar os **relacionamentos dos componentes** por meio de suas **interfaces**.
- Um **componente** é uma unidade de software independente que encapsula dentro de si sua implementação, e oferece serviços para o meio externo por meio de interfaces bem definidas.
- A **interface** de um componente define como o meio externo pode acessar e usar os serviços do componente.

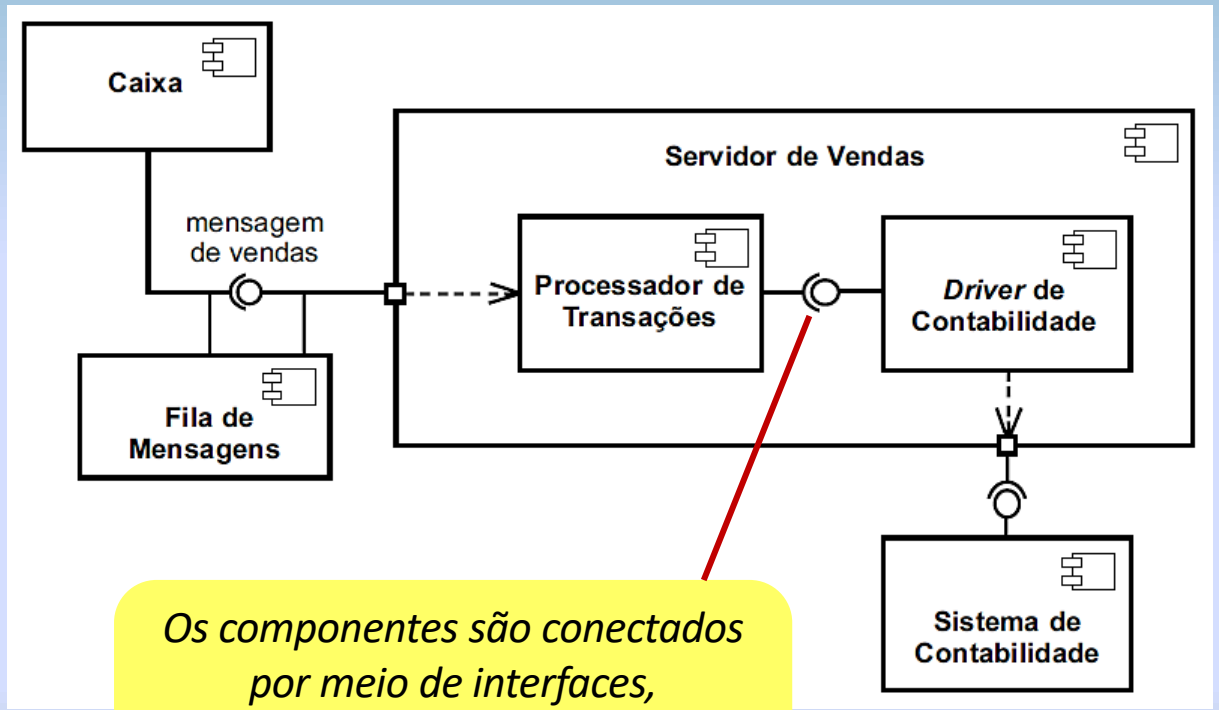
Um componente de software normalmente:

- *Executa uma atividade bem definida no sistema;*
- *É produzido e implantado independentemente de outros componentes;*
- *Pode ser combinado com outros componentes para compor aplicações;*
- *Pode ser configurado de acordo com as necessidades dos usuários;*
- *Pode ser facilmente atualizado e reutilizado.*

l.bertholdo@ifsp.edu.br

Diagrama de Componentes

Os componentes são representados por uma caixa com um ícone. Alternativamente, pode-se incluir a palavra-chave ***«component»****.*



Os componentes são conectados por meio de interfaces, frequentemente representadas pela notação de bola-e-soquete.

A ***caixa registradora*** *se conecta com um* ***servidor de vendas****, usando uma* ***interface de mensagens de vendas****. Como a rede não é confiável, existe uma* ***fila de mensagens****. Quando a rede está ativa, a* ***caixa*** *se comunica diretamente com o* ***servidor****. Quando não está, se comunica com a* ***fila****, que, por sua vez, se comunica com o* ***servidor*** *(após a rede ser reativada).*

A ***fila*** *fornece a* ***interface de mensagens*** *para se comunicar com a* ***caixa****, cujo uso é exigido para quem deseja se comunicar com o* ***servidor****.*

O ***servidor de vendas*** *é dividido em dois componentes: o* ***processador de transações****, que implementa a* ***interface de mensagens****, e o* ***driver de contabilidade****, que se comunica com o* ***sistema de contabilidade****.*

Diagrama de Implantação

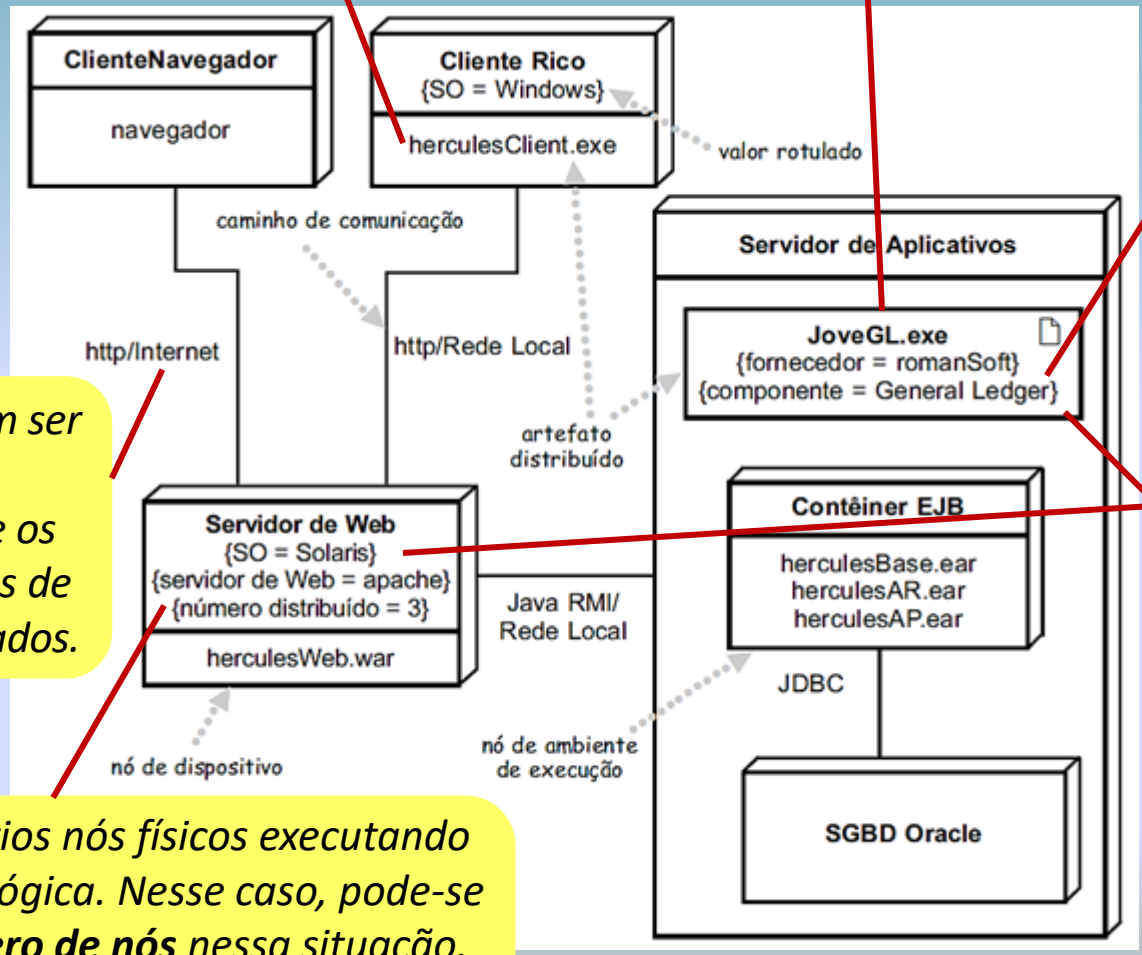
- O **diagrama de implantação** mostra o layout físico do sistema, exibindo quais partes do software são executadas em quais partes do hardware.
- Os principais itens do diagrama são os **nós** conectados. Eles podem representar tanto um **hardware**, como um computador ou outro dispositivo, quanto um **software**, como o sistema operacional ou um ambiente *container* para isolar processos de aplicações.
- Os nós contêm os **artefatos** gerados pelos softwares, na maioria das vezes arquivos. Esses arquivos podem ser executáveis (binários, EXEs, DLLs, JARs etc), arquivos de dados, arquivos de configuração, documentos HTML etc.
- A listagem de um artefato dentro de um nó mostra que ele está localizado ou instalado nesse nó do sistema.

l.bertholdo@ifsp.edu.br

Diagrama de Implantação

*Os **artefatos** podem ser mostrados como retângulos ou como simples nomes dentro de um nó.*

*Os artefatos podem ser a **implementação de um componente**. Para mostrar isso, pode-se usar um valor indicado na caixa do artefato.*



*Os **caminhos** podem ser rotulados com informações sobre os protocolos e formas de comunicação utilizados.*

*Os nós e os artefatos podem ser **rotulados** por meio de chaves e valores, para associar informações úteis.*

*É comum ter vários nós físicos executando a mesma tarefa lógica. Nesse caso, pode-se declarar o **número de nós** nessa situação. Aqui, o rótulo indica três servidores Web.*

l.bertholdo@ifsp.edu.br

Referências

- FOWLER, Martin. **UML essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2007.

- PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software:** uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

- SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.