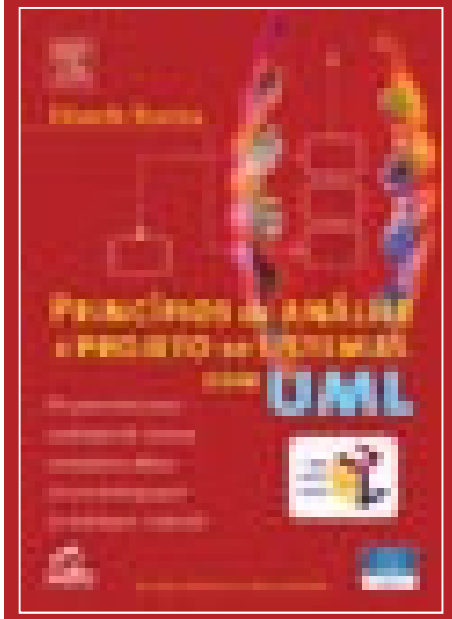
Princípios de Análise e Projeto de Sistemas com UML

2ª edição



Eduardo Bezerra

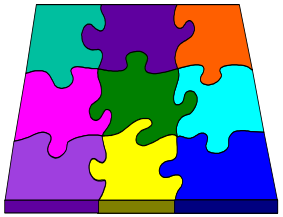
Editora Campus/Elsevier

Capítulo 11
Arquitetura do sistema

Nada que é visto, é visto de uma vez e por completo.

--EUCLIDES

Tópicos



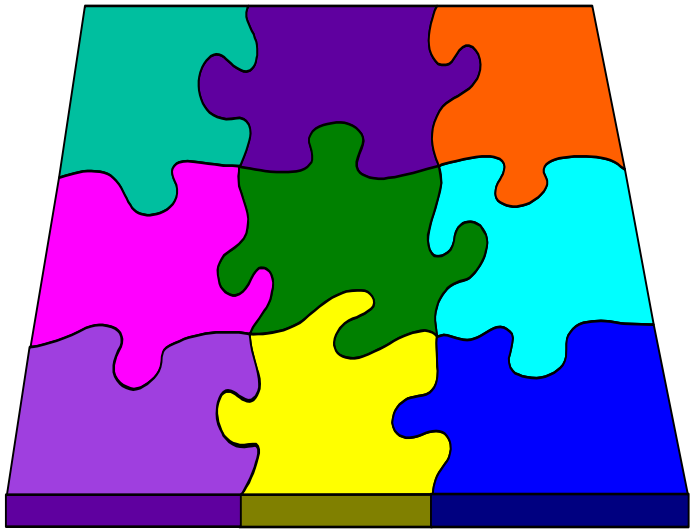
- Introdução
- Arquitetura lógica
  - Diz respeito a como um SSOO é decomposto em diversos subsistemas e como as suas classes são dispostas pelos diversos subsistemas.
- Alocação (arquitetura) física
  - Diz respeito a como os subsistemas devem ser dispostos fisicamente quando o sistema tiver de ser implantado.)
  - Dois aspectos importantes:
    - Alocação de camadas
    - Alocação de componentes
- Projeto da arquitetura no processo de desenvolvimento

Introdução

- Em um SSOO, os objetos interagem entre si através do envio de mensagens com o objetivo de executar suas tarefas.
  - Um SSOO também pode ser visto como um conjunto de ***subsistemas*** que o compõem.
- A definição dos subsistemas de um SSOO é feita no ***projeto da arquitetura*** ou ***projeto arquitetural***.
- Essa atividade define de que forma o sistema se divide em partes e quais são as interfaces entre essas partes.
- Vantagens de dividir um SSOO em subsistemas:
  - produzir unidades menores de desenvolvimento;
  - maximizar o reuso no nível de subsistemas componentes;
  - ajuda a gerenciar a complexidade no desenvolvimento.

Introdução

- Questões relacionadas à arquitetura de um sistema:
  - Como um sistema é decomposto em subsistemas, e como as suas classes são dispostas pelos diversos subsistemas?
  - Como subsistemas devem ser dispostos fisicamente quando o sistema tiver de ser implantado? (nós de processamento)
- De acordo com a especificação da UML: "Arquitetura de software é a estrutura organizacional do software. Uma arquitetura pode ser recursivamente decomposta em partes que interagem através de interfaces."
- As decisões tomadas para a definição da arquitetura de software influenciam diretamente na forma como um SSOO irá atender a seus ***requisitos não-funcionais***.



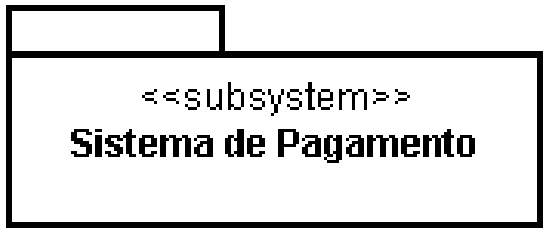
11.1 Arquitetura lógica

Arquitetura lógica

- Chamamos de arquitetura lógica à organização das classes de um SSOO em subsistemas, que correspondem a aglomerados de classes.
  - Um SSOO pode ser subdividido em diversos subsistemas
- Um subsistema provê serviços para outros através de sua interface.
- A interface de um subsistema corresponde ao conjunto de serviços que ele provê.
  - Cada subsistema provê ou utiliza serviços de outros subsistemas.

Diagrama de subsistemas

- Uma visão gráfica dos diversos componentes de um SSOO pode ser representada por um ***diagrama de subsistemas***.
  - Um diagrama de susbistemas é um diagrama de pacotes, onde cada pacote representa um subsistema
  - Cada subsistema é rotulado com o estereótipo <<subsystem>>.
- Exemplo de susbsistema:



Alocação de classes a subsistemas

- Durante o desenvolvimento de um SSOO, seus subsistemas devem ser identificados, juntamente com as interfaces entre eles.
- Cada classe do sistema é, então, alocada aos subsistemas.
- Uma vez feito isso, esses subsistemas podem ser desenvolvidos quase que de forma independente uns dos outros.
- A seguir, são descritas algumas dicas que podem ser utilizadas para realizar a alocação de classes a subsistemas.

Alocação classes a subsistemas

- Modelo de classes de domínio: este é o ponto de partida para a identificação dos subsistemas de um SSOO.
  - Classes devem ser agrupadas segundo algum critério para formar subsistemas.
- Um critério de agrupamento possível: considerar as classes mais importantes do modelo de classes de domínio.
  - Para cada uma dessas classes, um subsistema é criado.
  - Outras classes menos importantes e relacionadas a uma classe considerada importante são posicionadas no subsistema desta última.
- Por outro lado, na fase de projeto:
  - Algumas das classes do modelo de domínio podem sofrer decomposições adicionais, o que resulta em novas classes.
  - Pode ser que uma classe de domínio seja ela própria um subsistema.
- Quando essas classes forem criadas, elas são adicionadas aos subsistemas pré-definidos.

Alocação classes a subsistemas

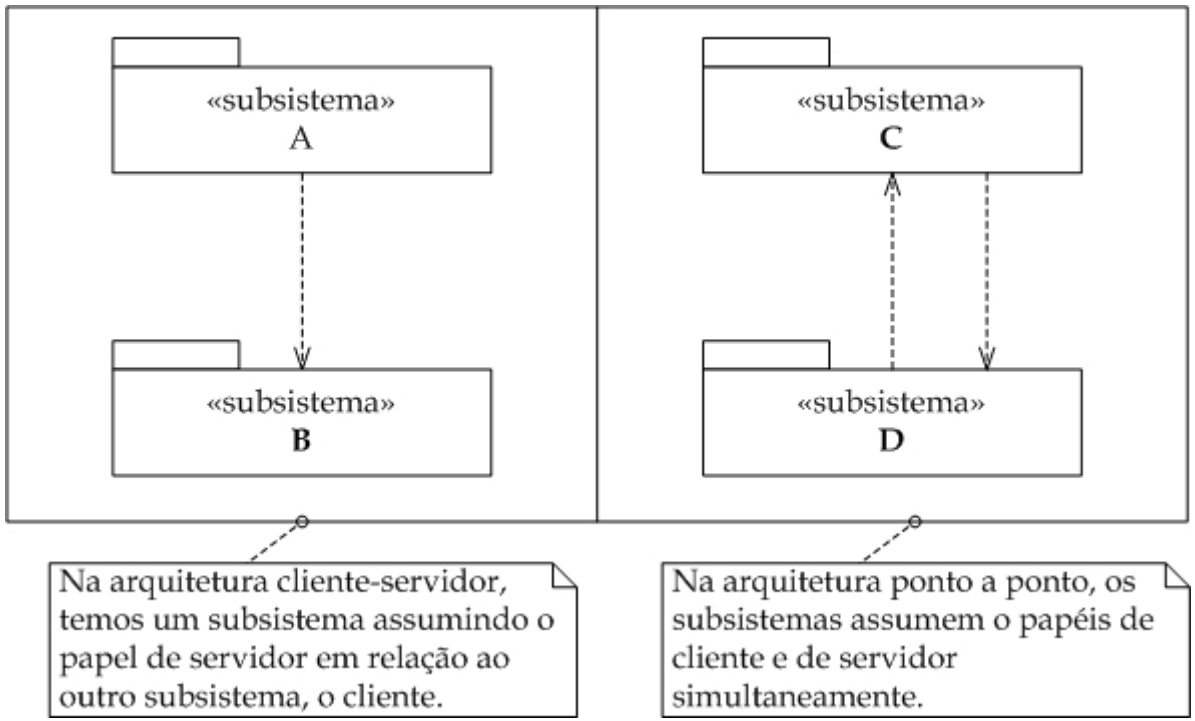
- Subsistemas devem ser minimamente acoplados.
- Subsistemas devem ser maximamente coesivos.
- Dependências cíclicas entre subsistemas devem ser evitadas.
  - A alternativa para eliminar ciclos é quebrar um subsistema pertencente ao ciclo em dois ou mais. Uma outra solução é combinar dois ou mais subsistemas do ciclo em um único.
- Uma classe deve ser definida em um único subsistema (embora possa ser utilizada em vários).
  - O subsistema que define a classe deve mostrar todas as propriedades da mesma.
  - Outros subsistemas que fazem referência a essa classe podem utilizar a notação simplificada da mesma.

Camadas de software

- Dizemos que dois subsistemas interagem quando um precisa dos serviços do outro.
- Há basicamente duas formas de interação entre subsistemas:
  - ***ponto a ponto***: na arquitetura ponto a ponto, a comunicação pode acontecer em duas vias.
  - ***cliente-servidor***: há a comunicação somente em uma via entre dois subsistemas, do cliente para o servidor. Nessa arquitetura, chamamos de ***camadas*** os subsistemas envolvidos.
- Essas formas de interação entre subsistemas influenciam a o modo pelo qual os subsistemas são distribuídos fisicamente pelos nós de processamento.

Camadas de software

- Arquiteturas cliente-servidor e ponto a ponto

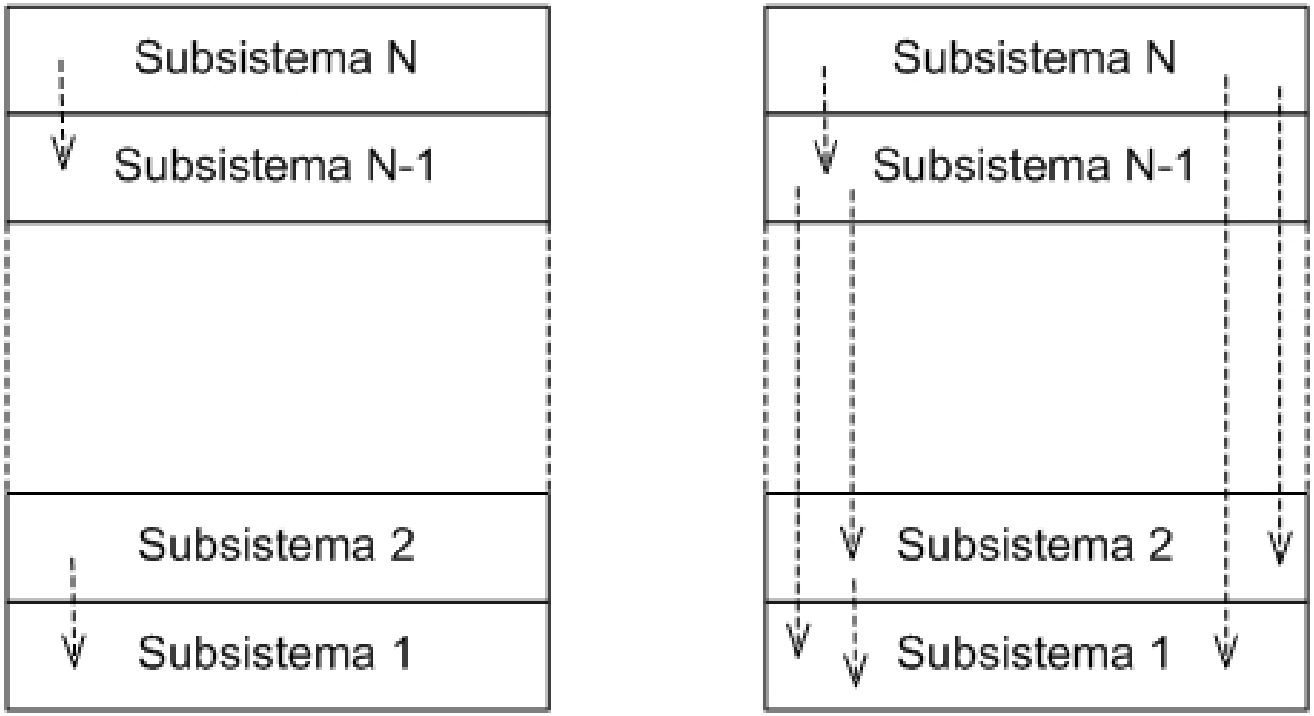


Camadas de software

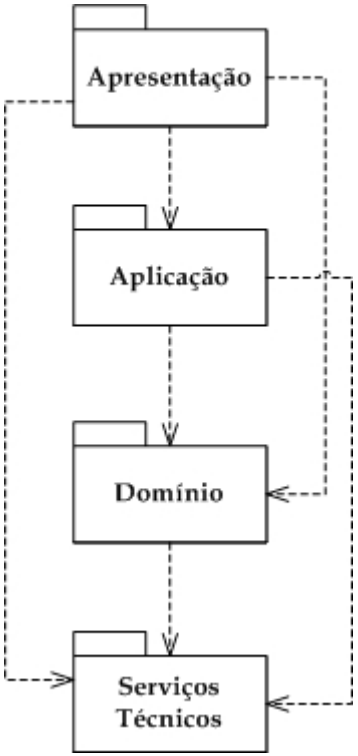
- Um SSOO projetado em camadas pode ter uma ***arquitetura aberta*** ou uma ***arquitetura fechada***.
  - Em uma arquitetura fechada, um componente de uma camada de certo nível somente pode utilizar os serviços de componentes da sua própria camada ou da imediatamente inferior.
  - Em uma arquitetura aberta, uma camada em certo nível pode utilizar os serviços de qualquer camada inferior.
- Na maioria dos casos práticos, encontramos sistemas construídos através do uso de uma arquitetura aberta.

Camadas de software

- Arquiteturas abertas e fechadas



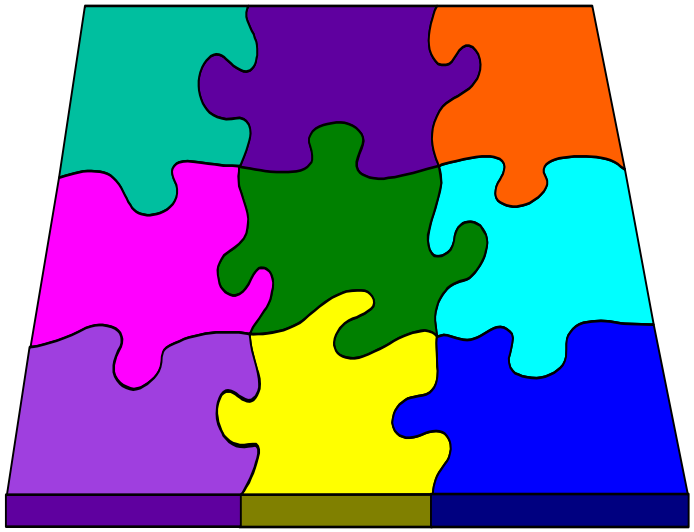
- Uma divisão tipicamente encontrada para as camadas lógicas de um SSOO é a que separa o sistema nas seguintes camadas:
  - ***apresentação***, ***aplicação***, ***domínio*** e ***serviços técnicos***.
- Da esquerda para a direita, temos camadas cada vez mais genéricas.
- Também da esquerda para a direita, temos a ordem de dependência entre as camadas;
  - por exemplo a camada da apresentação depende (requisita serviços) da camada de aplicação, mas não o contrário.



- Princípio básico: camadas mais altas devem depender das camadas mais baixas, e não o contrário.
  - Essa disposição ajuda a gerenciar a complexidade através da divisão do sistema em partes menos complexas que o todo.
  - Também incentiva o reuso, porque as camadas inferiores são projetadas para serem independentes das camadas superiores.
  - O acoplamento entre camadas é mantido no nível mínimo possível.
  - Uma mudança em uma camada mais baixa que não afete a sua interface não implicará em mudanças nas camadas mais altas.
  - E vice-versa, uma mudança em uma camada mais alta que não implica na criação de um novo serviço em uma camada mais baixa não irá afetar estas últimas.

- É importante notar que uma aplicação típica normalmente possui diversos subsistemas (ou pacotes) internamente a cada uma das camadas.
  - Por exemplo, em uma aplicação que forneça certo serviço que é acessível tanto por um usuário final, quando por outra aplicação (através de um serviço WEB), há duas camadas de aplicação, possivelmente fazendo acesso a mesma camada da aplicação.
- Uma certa camada pode ser dividida verticalmente no que costumamos chamar de ***partições***.
  - Por exemplo, em um sistema de vendas pela WEB, a camada de domínio pode ser decomposta nos seguintes subsistemas (partições): Clientes, Pedidos e Entregas.

- Durante a definição da arquitetura lógica de um SSOO, o uso de ***padrões de projeto*** é comum.
  - Para comunicação entre subsistemas, normalmente o padrão ***Façade*** é utilizado.
  - Para diminuir o acoplamento entre camadas (ou entre partições dentro de uma camada), o padrão ***Factory Method*** pode ser utilizado.
  - O padrão ***Observer*** também pode ser utilizado quando uma camada em certo nível precisa ser comunicar com uma camada de um nível superior.
    - O componente da camada inferior representa o sujeito, enquanto o componente da camada superior representa o observador. O sujeito não precisa ter conhecimento da classe específica do observador.



11.2 Implantação física

Arquitetura de implantação

- Representa a disposição física do sistema de software pelo hardware disponível.
- A divisão de um sistema em camadas é independente da sua disposição física.
  - As camadas de software podem estar fisicamente localizadas em uma única máquina, ou podem estar distribuídas por diversos processadores.
  - Alternativamente, essas camadas podem estar distribuídas fisicamente em vários processadores. (Por exemplo, quando a camada da lógica do negócio é dividida em duas ou mais máquinas.)
- O modelo que representa a arquitetura física é denominado ***modelo de implementação*** ou ***modelo da arquitetura física***.

Arquitetura de implantação

- A arquitetura de implantação diz respeito à disposição dos subsistemas de um SSOO pelos nós de processamento disponíveis.
- Para sistemas simples, a arquitetura de implantação não tem tanta importância.
- No entanto, na modelagem de sistemas complexos, é fundamental conhecer quais são os componentes físicos do sistema, quais são as interdependências entre eles e de que forma as camadas lógicas do sistema são dispostas por esses componentes.

Alocação de camadas

- Em um sistema construído segundo a arquitetura a cliente-servidor, é comum utilizar as definições das camadas lógicas como um guia para a alocação física dos subsistemas.
- Sendo assim, a cada nó de processamento são alocadas uma ou mais camadas lógicas.
- Note que o termo *camada* é normalmente utilizado com dois sentidos diferentes:
  - para significar uma camada lógica (*layer*)
  - e para significar uma camada física, esta última normalmente associada a um nó de processamento (*tier*).

- Vantagens da alocação das camadas lógicas a diferentes nós de processamento:
  - A divisão dos objetos permite um maior grau de manutenção e reutilização, porque sistemas de software construídos em camadas podem ser mais facilmente estendidos.
  - Sistemas em camadas também são mais adaptáveis a uma quantidade maior de usuários.
    - Servidores mais potentes podem ser acrescentados para compensar um eventual crescimento no número de usuários do sistema.
- No entanto, a divisão do sistema em camadas apresenta a desvantagem de *potencialmente* diminuir o desempenho do mesmo.
  - a cada camada, as representações dos objetos sofrem modificações, e essas modificações levam tempo para serem realizadas.

- Um SSOO que divide a interação com o usuário e o acesso aos dados em dois subsistemas é denominado sistema cliente-servidor em duas camadas.
- A construção de sistemas em duas camadas é vantajosa quando o número de clientes não é tão grande (por exemplo, uma centena de clientes interagindo com o servidor através de uma rede local).
- No entanto, o surgimento da a Internet causou problemas em relação à estratégia cliente-servidor em duas camadas.
  - Isso porque a idéia básica da Internet é permitir o acesso a variados recursos através de um programa navegador (browser).

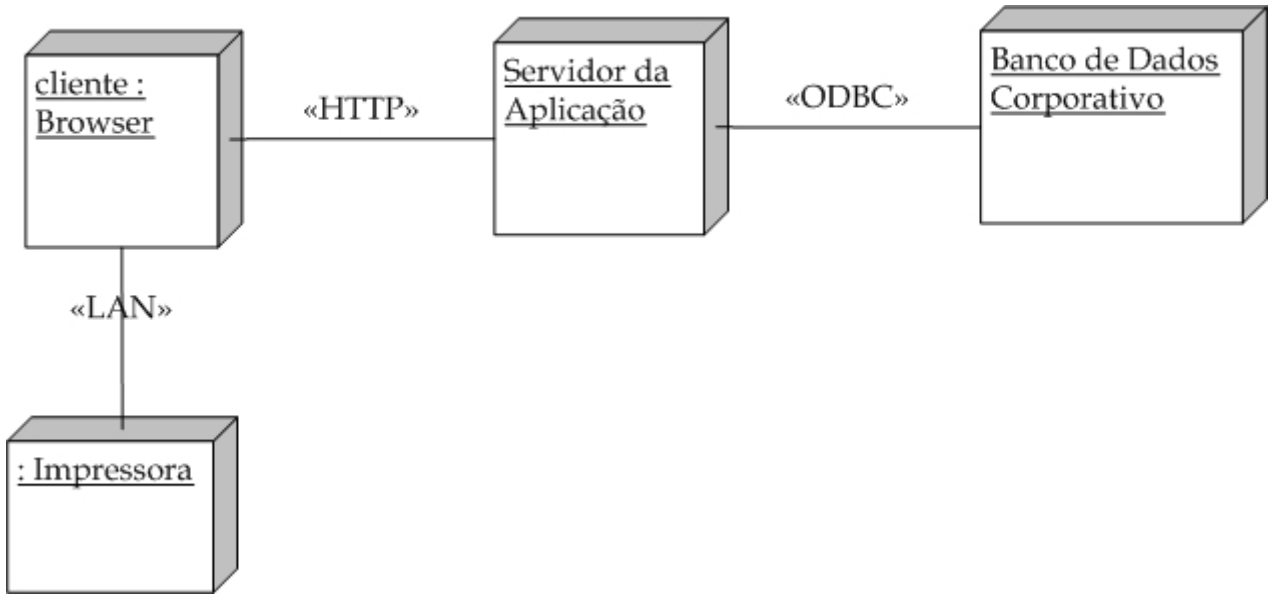
- A solução encontrada para o problema da arquitetura em duas camadas foi simplesmente dividir o sistemas em mais camadas de software.
  - Sistemas construídos segundo essa estratégia são denominados sistemas cliente-servidor em três camadas ou sistemas cliente-servidor em quatro camadas.
- Entretanto, a idéia básica original permanece: dividir o processamento do sistema em diversos nós.
  - Em particular, uma disposicão bastante popular, principalmente em aplicações para a WEB, é a arquitetura cliente-servidor em três camadas…

- Na arquitetura cliente-servidor em três camadas:
  - A camada lógica de apresentação fica em um nó de processamento (conhecido como *presentation tier*)
  - As camadas lógicas da aplicação e do domínio ficam juntas em outro nó (camada física denominada *middle tier*).
    - A camada física do meio corresponde ao ***servidor da aplicação***.
    - A camada de apresentação requisita serviços ao servidor da aplicação.
    - É também possível haver mais de um servidor de aplicação, com o objetivo de aumentar a disponibilidade e o desempenho da aplicação.
  - A camada física do meio faz acesso a outra camada física, onde normalmente se encontra um SGBD.
    - Esta última camada física é chamada de *camada de dados* (*data tier*).

- Uma vez definidas as alocações das camadas lógicas aos nós de processamento, podemos fazer a representação gráfica com suporte da UML, através do ***diagrama de implantação***.
- Os elementos desse diagrama são os ***nós*** e as ***conexões***.
- Um nó representa um recurso computacional e normalmente possui uma memória e alguma capacidade de processamento.
  - Exemplos: processadores, dispositivos, sensores, roteadores ou qualquer objeto físico de importância para o sistema de software.

- Os nós são ligados uns aos outros através de conexões.
  - As conexões representam mecanismos de comunicação: meios físicos (cabo coaxial, fibra ótica etc.) ou protocolos de comunicação (TCP/IP, HTTP etc.).

- Exemplo de diagrama de implantação

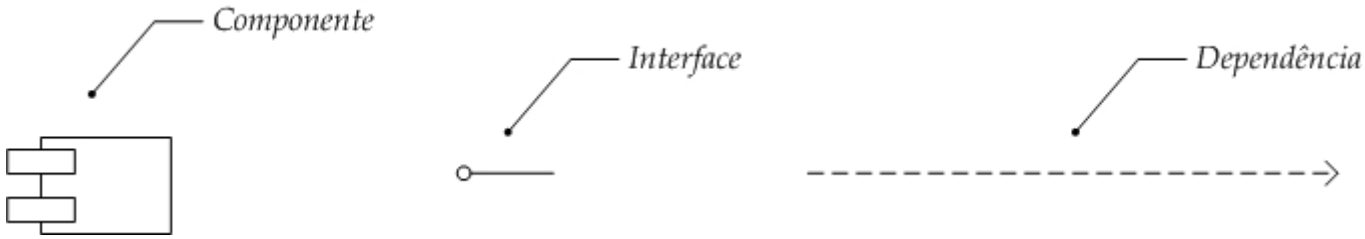


Alocação de Componentes

- Na arquitetura (alocação) física, devemos também definir quais os ***componentes de software*** de cada camada.
- Um componente de software é uma unidade que existe a tempo de execução, que pode ser utilizada na construção de vários sistemas e que pode ser substituída por outra unidade que tenha a mesma funcionalidade.
- As tecnologias COM (Microsoft), CORBA (OMG) e EJB (Sun) são exemplos de tecnologias baseadas em componentes.
- Um componente provê acesso aos seus serviços através de uma ***interface***.
  - Quando construído segundo o paradigma OO, um componente é composto de diversos objetos.
  - Nesse caso, a interface do componente é constituída de um ou mais serviços que as classes dos referidos objetos implementam.

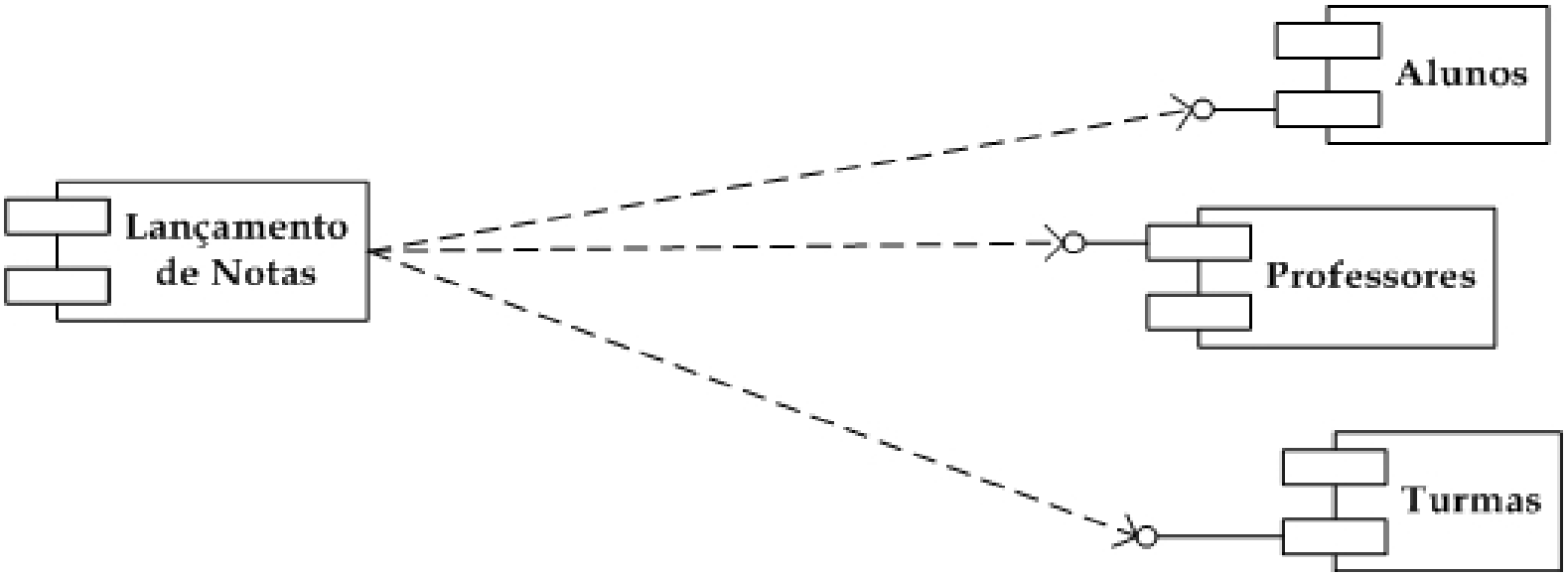
Alocação de Componentes

- A UML define uma forma gráfica para representar componentes visualmente, o diagrama de componentes.
- Esse diagrama mostra os vários componentes de software e suas dependências.
- Os elementos gráficos desse diagrama são ilustrados na figura abaixo.



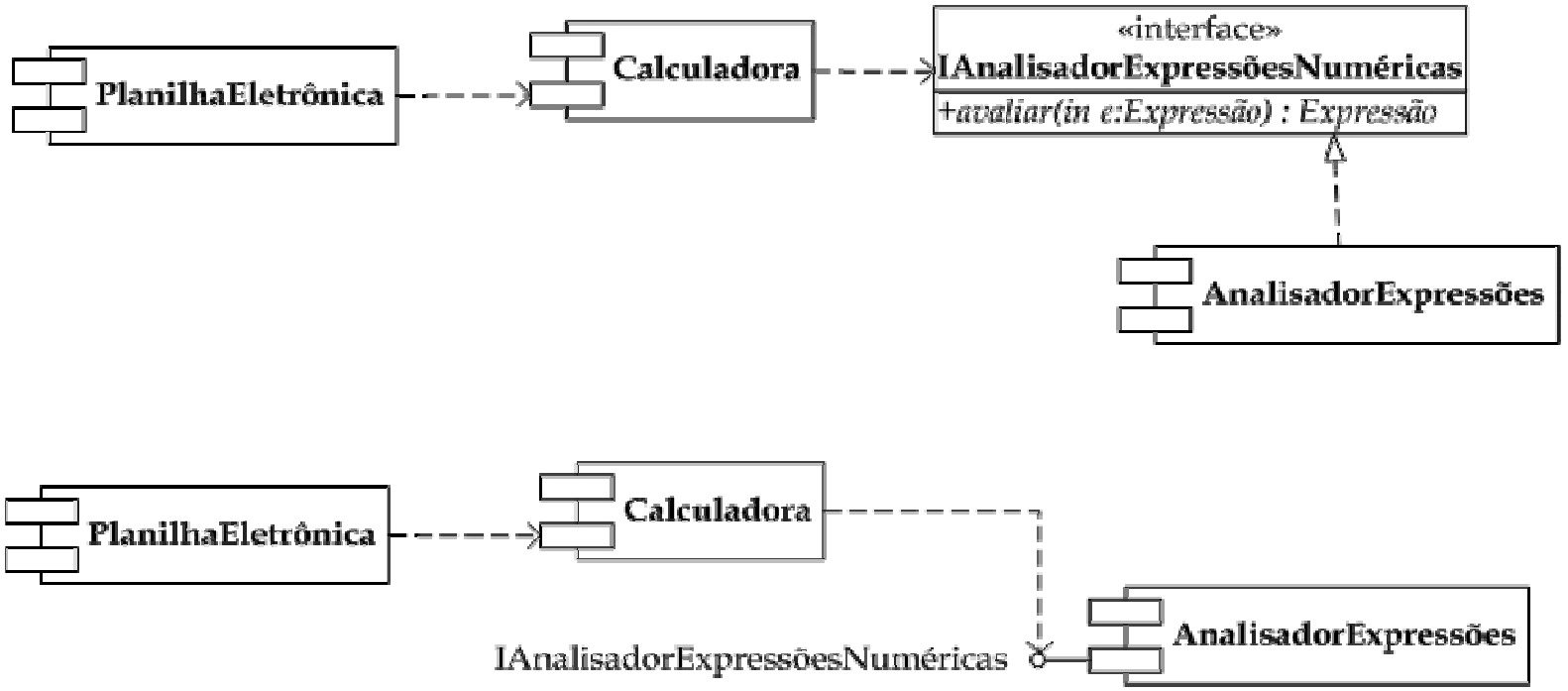
Alocação de Componentes

- Exemplo de diagrama de componentes.



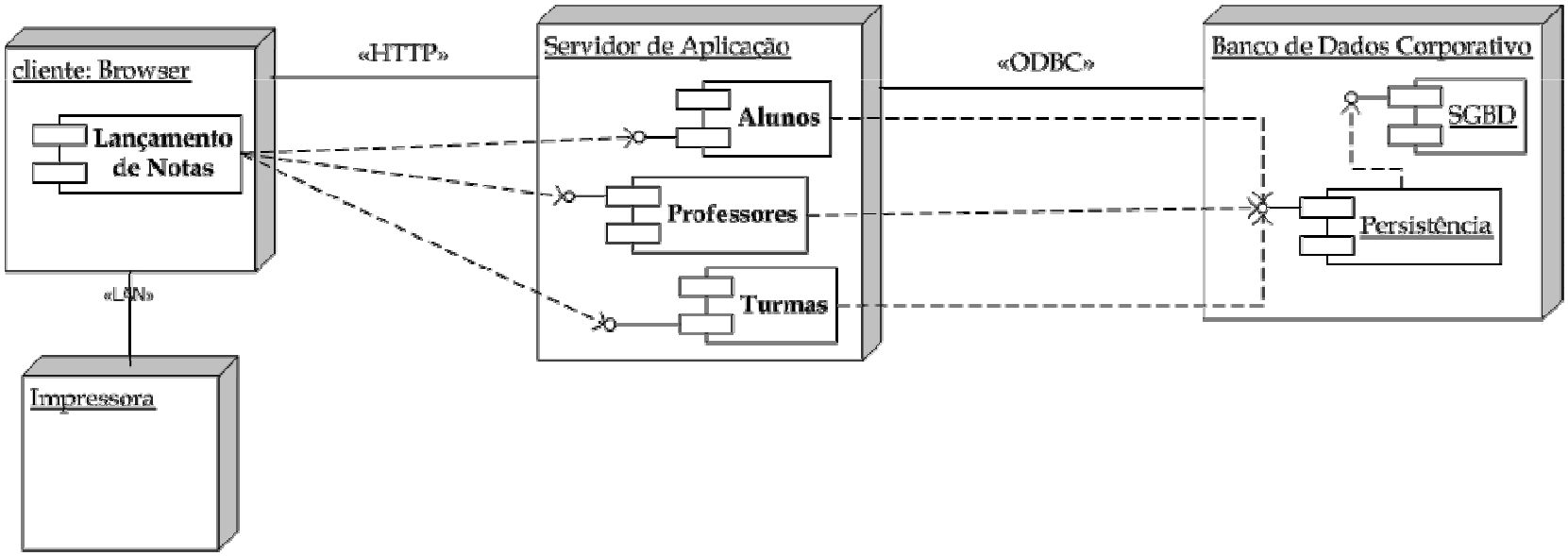
Alocação de Componentes

- Alternativas para representar interfaces de componentes.



Alocação de Componentes

- Exemplo de diagrama de componentes embutido em um diagrama de implantação.



Alocação de Componentes

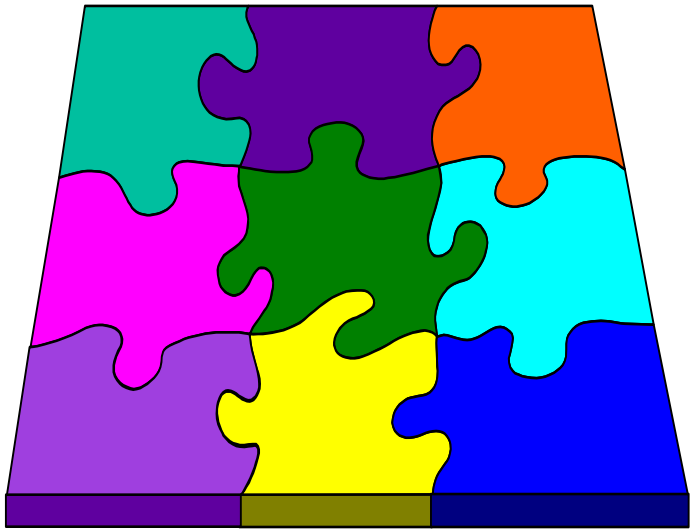
- A atividade de alocação de componentes aos nós físicos só tem sentido para sistemas distribuídos.
  - Para sistemas que utilizam um único processador, não há necessidade desta atividade.
- Um dos principais objetivos: distribuir a carga de processamento do sistema para aumentar o desempenho.
  - No entanto, nem sempre isso aumenta o desempenho.
  - Isso porque a sobrecarga de comunicação entre os nós pode anular os ganhos obtidos com a distribuição do processamento.

Alocação de Componentes

- Envio de mensagem versus “distância”
  - dentro de um processo executando em um nó.
  - entre processos no mesmo nó.
  - ultrapassa as fronteiras de um nó para ser executada em outra máquina.
- Portanto, durante a alocação de componentes, o arquiteto de software deve considerar diversos fatores.
  - muitos desses fatores se sobrepõem ou são incompatíveis.

Alocação de Componentes

- Fatores relacionados ao desempenho:
  - Utilização de dispositivos
  - Carga computacional
  - Capacidade de processamento dos nós
  - Realização de tarefas
  - Tempo de resposta
- Outros fatores:
  - Outros requisitos não funcionais do sistema
  - Segurança
  - Diferenças de plataformas
  - Características dos usuários do sistema
  - Necessidade ou benefícios da distribuição das camadas lógicas do sistema
  - Redundância



11.3 Projeto da arquitetura no processo de desenvolvimento

Projeto da arquitetura no processo de desenvolvimento

- A construção dos diagramas de componentes é iniciada na fase de elaboração (projeto da arquitetura) e refinada na fase de construção (projeto detalhado).
- A construção de diagramas de componentes se justifica para componentes de execução.
  - permite visualizar as dependências entre os componentes e a utilização de interfaces a tempo de execução do sistema.
  - sistema distribuído: representar seus componentes utilizando diagramas de implantação.

Projeto da arquitetura no processo de desenvolvimento

- Não é recomendável usar diagramas de componentes para representar dependências de compilação entre os elementos do código fonte do sistema.
  - A maioria dos ambientes de desenvolvimento tem capacidade de manter as dependências entre códigos fonte, códigos objeto, códigos executáveis e páginas de script.
  - Só adiciona mais diagramas que terão que ser mantidos e que não terão uma real utilidade.
  - Há também vários sistemas de gerenciamento de configurações e de versões de código (e.g, CVS).

Projeto da arquitetura no processo de desenvolvimento

- Em relação ao diagrama de implantação, sua construção tem início na fase de elaboração.
- Na fase de construção, os componentes são adicionados aos diversos nós.
  - Cada versão do sistema corresponde a uma versão do DI, que exibe os componentes utilizados na construção daquela versão.
  - O DI deve fazer parte dos manuais para instalação e operacionalização do sistema.
- Nem todo sistema necessita de diagramas de componentes ou diagramas de implantação.
  - Sistemas simples versus sistemas complexos.