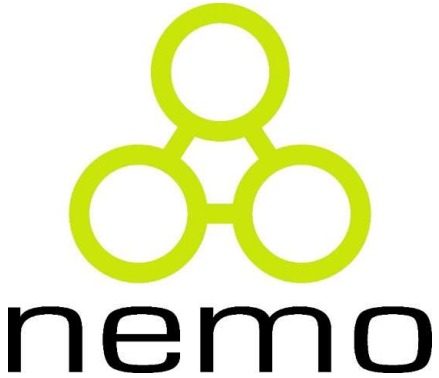


Projeto de Sistemas de Software

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Arquitetura de Software

- À medida que os sistemas computacionais crescem em tamanho e complexidade, as técnicas relacionadas ao projeto de estruturas de dados e algoritmos não são mais suficientes para lidar com os problemas envolvendo o projeto de software no nível de sistema.

- Passa a ser necessário considerar um nível de organização relativo à arquitetura do software.

- O projeto da arquitetura envolve, dentre outras, questões relativas à organização e estrutura geral do sistema, seleção de alternativas de projeto, atribuição de funcionalidades a elementos de projeto e atendimento a atributos de qualidade (requisitos não funcionais).

O que é Arquitetura de Software?

- A arquitetura de software de um sistema computacional refere-se à sua estrutura, consistindo de elementos de software, propriedades externamente visíveis desses elementos e os relacionamentos entre eles.

- A arquitetura define elementos de software (ou módulos) e envolve informações sobre como eles se relacionam uns com os outros.

- Uma arquitetura pode envolver mais de um tipo de estrutura, com diferentes tipos de elementos e de relacionamentos entre eles.

O que é Arquitetura de Software?

- Assim, uma arquitetura é antes de tudo uma abstração que suprime detalhes dos elementos que não afetam como eles são usados, como se relacionam, como interagem e como usam outros elementos.

- Em quase todos os sistemas modernos, elementos interagem com outros por meio de interfaces que dividem detalhes sobre um elemento em partes pública e privada. A arquitetura está preocupada com a parte pública dessa divisão.

Onde obtemos informações sobre a arquitetura do sistema?

O que é Arquitetura de Software?

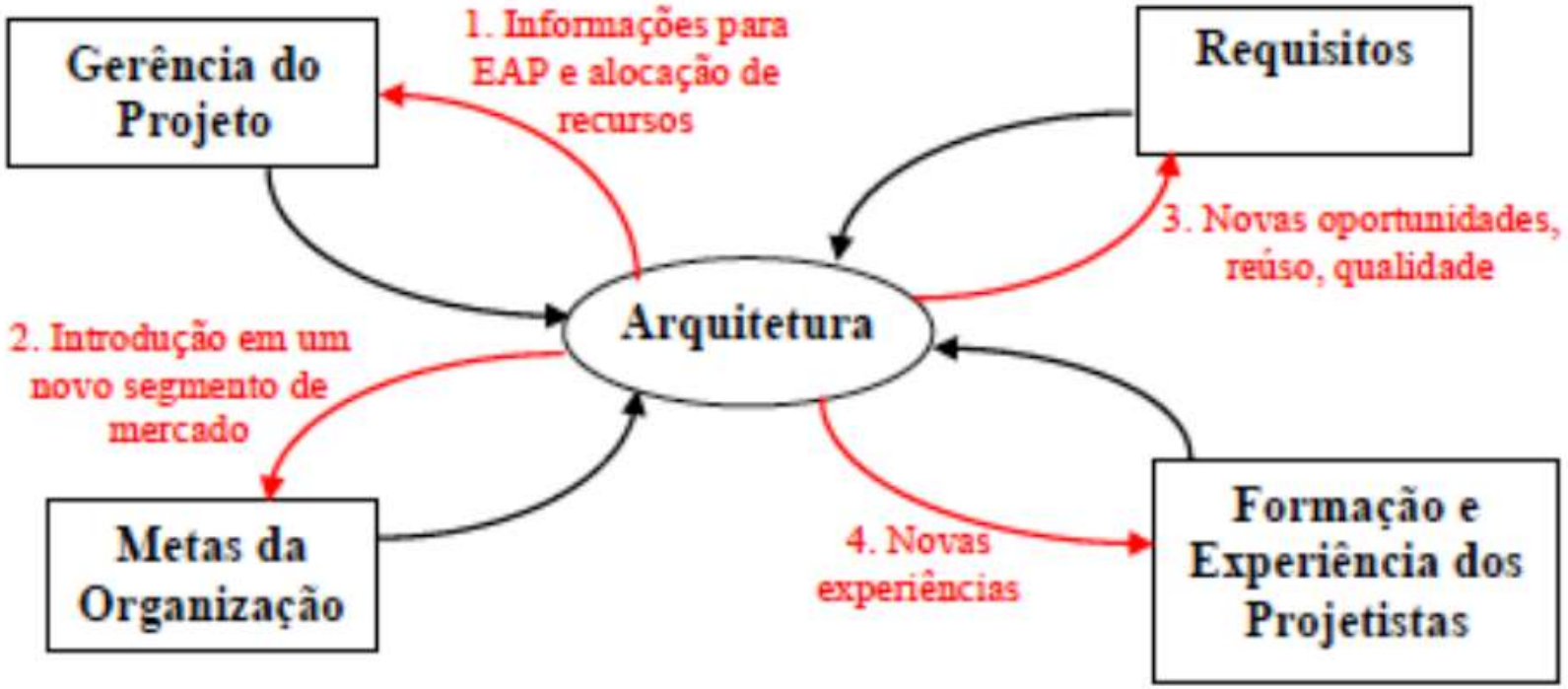
Onde obtemos informações sobre a arquitetura do sistema?

- Requisitos;
- Aspectos técnicos;
- Aspectos sociais;
- Aspectos de negócio da organização.

- No projeto da arquitetura de software, projetistas são influenciados por requisitos para o sistema, estrutura e metas da organização de desenvolvimento, ambiente técnico disponível e por sua própria experiência e formação.

O que é Arquitetura de Software?

- Além disso, os relacionamentos entre metas de negócio, requisitos de sistemas, experiência dos projetistas, arquiteturas e sistemas implantados geram diversos laços de realimentação que podem ser gerenciados pela organização.



nemo

O que é Arquitetura de Software?

- Muitas pessoas têm interesse na arquitetura de software, tais como clientes, usuários finais, desenvolvedores, gerentes de projeto e mantenedores.

- Alguns desses interesses são conflitantes e o projetista frequentemente te m de mediar conflitos até chegar à configuração que atenda de forma mais adequada a todos os interesses.

- Neste contexto, a arquitetura de software é importante principalmente porque:

- • **Representa uma abstração do sistema que pode ser usada para compreensão mútua, negociação, consenso e comunicação entre os interessados.**

O que é Arquitetura de Software?

- • **Manifesta as primeiras decisões de projeto.**

- Essas decisões definem restrições sobre a implementação e a estrutura do projeto.

- A implementação tem de ser feita considerando a divisão de elementos prescrita pela arquitetura. Os elementos têm de interagir conforme o prescrito e cada elemento tem de cumprir sua responsabilidade conforme definido pela arquitetura.

- Além disso, a extensão na qual o sistema vai ser capaz de satisfazer os atributos de qualidade (requisitos não funcionais de produto) requeridos é substancialmente determinada pela arquitetura.

O que é Arquitetura de Software?

- Particularmente a manutenibilidade é fortemente afetada pela arquitetura.

- A arquitetura divide possíveis alterações em três categorias: locais (confinadas em um único elemento), não locais (requerem a alteração de vários elementos, mas mantêm intacta a abordagem arquitetônica subjacente) e arquitetônicas (afetam a estrutura do sistema e podem requerer alterações ao longo de todo o sistema).

- Obviamente, alterações locais são as mais desejáveis e, portanto, uma arquitetura efetiva deve propiciar que as alterações mais prováveis sejam as mais fáceis de fazer.

O que é Arquitetura de Software?

- **• Constitui um modelo relativamente pequeno e intelectualmente compreensível de como o sistema é estruturado e como seus elementos trabalham em conjunto.**

- Além disso, esse modelo é transferível para outros sistemas, em especial para aqueles que exibem requisitos funcionais e não funcionais similares, promovendo reúso em larga escala.

- Um desenvolvimento baseado na arquitetura frequentemente enfoca a composição ou montagem de elementos que provavelmente foram desenvolvidos separadamente, ou até mesmo de forma independente.

O que é Arquitetura de Software?

- Essa composição é possível porque a arquitetura define os elementos que devem ser incorporados ao sistema.

- Além disso, a arquitetura restringe possíveis substituições de elementos, tomando por base a forma como eles interagem com o ambiente, como eles recebem e entregam o controle, que dados consomem e produzem, como acessam esses dados e quais protocolos usam para se comunicar e compartilhar recursos.

- É importante que o projetista seja capaz de reconhecer estruturas comuns utilizadas em sistemas já desenvolvidos, de modo a compreender as relações existentes e desenvolver novos sistemas como variações dos sistemas existentes.

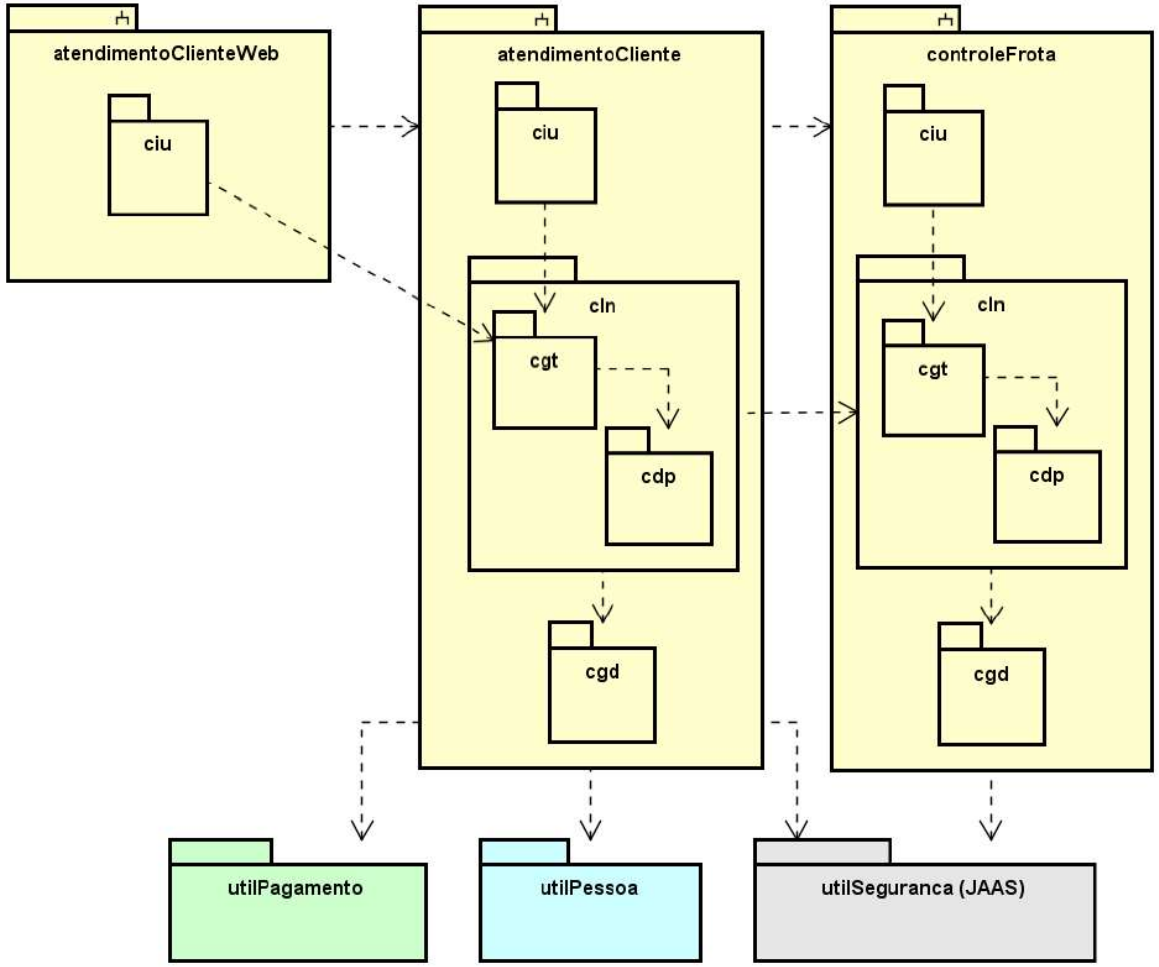
O que é Arquitetura de Software?

- O entendimento de arquiteturas de software existentes permite que os projetistas avaliem alternativas de projeto.

- Neste contexto, uma representação da arquitetura é essencial para permitir descrever propriedades de um sistema complexo, bem como uma análise da arquitetura proposta.

- Muitas vezes, arquiteturas são representadas na forma de diagramas. Idealmente, a representação de uma arquitetura deve incorporar informações acerca dos tipos dos elementos e dos relacionamentos.

O que é Arquitetura de Software?



nemo

Classes de Sistemas

- Sistemas similares, muito provavelmente, terão arquiteturas similares.

- Sistemas podem ser classificados de muitas formas diferentes:

- Sistemas de Transformação (ou Processamento) em Lote;
- Sistemas de Transformação Contínua;
- Sistemas de Interface Interativa;
- Sistemas de Simulação Dinâmica;
- Sistemas de Tempo Real;
- Sistemas Gerenciadores de Transações;
- **Sistemas de Informação.**

Sistemas de Informação

- O principal objetivo dessa classe de sistemas é o gerenciamento de informações.

- SIs possuem as seguintes características:

- • SIs geralmente envolvem grandes quantidades de dados e a sua gerência é uma parte importante do sistema. Assim, bancos de dados são frequentemente utilizados.

- • Usuários tipicamente acessam os dados concorrentemente.

Classes de Sistemas

- • Há uma grande quantidade de interfaces com o usuário para tratar os dados. O perfil dos usuários varia de ocasional a regular e normalmente eles não dominam profundamente a tecnologia. Assim, os dados têm de ser apresentados em muitos diferentes meios para diferentes propósitos.

- • Um SI geralmente precisa estar integrado com outros SIs da organização. Os vários sistemas são construídos em diferentes momentos, por equipes distintas, usando diferentes tecnologias. Esta situação é agravada quando os SIs precisam estar integrados com sistemas de organizações parceiras.

- • Regras de negócio são impostas e é necessário lidar com diversas condições que, muitas vezes, estabelecem relações umas com as outras, de modos até surpreendentes.

nemo

Estilos Arquitetônicos

- Um estilo arquitetônico define um vocabulário de tipos de elementos e tipos de relacionamentos, e um conjunto de restrições sobre como eles podem ser combinados.

- Padrões arquitetônicos também estabelecem tipos de elementos e de relacionamentos entre eles, mas sua apresentação segue uma forma bem definida, indicando nome, contexto, problema, solução e consequências.

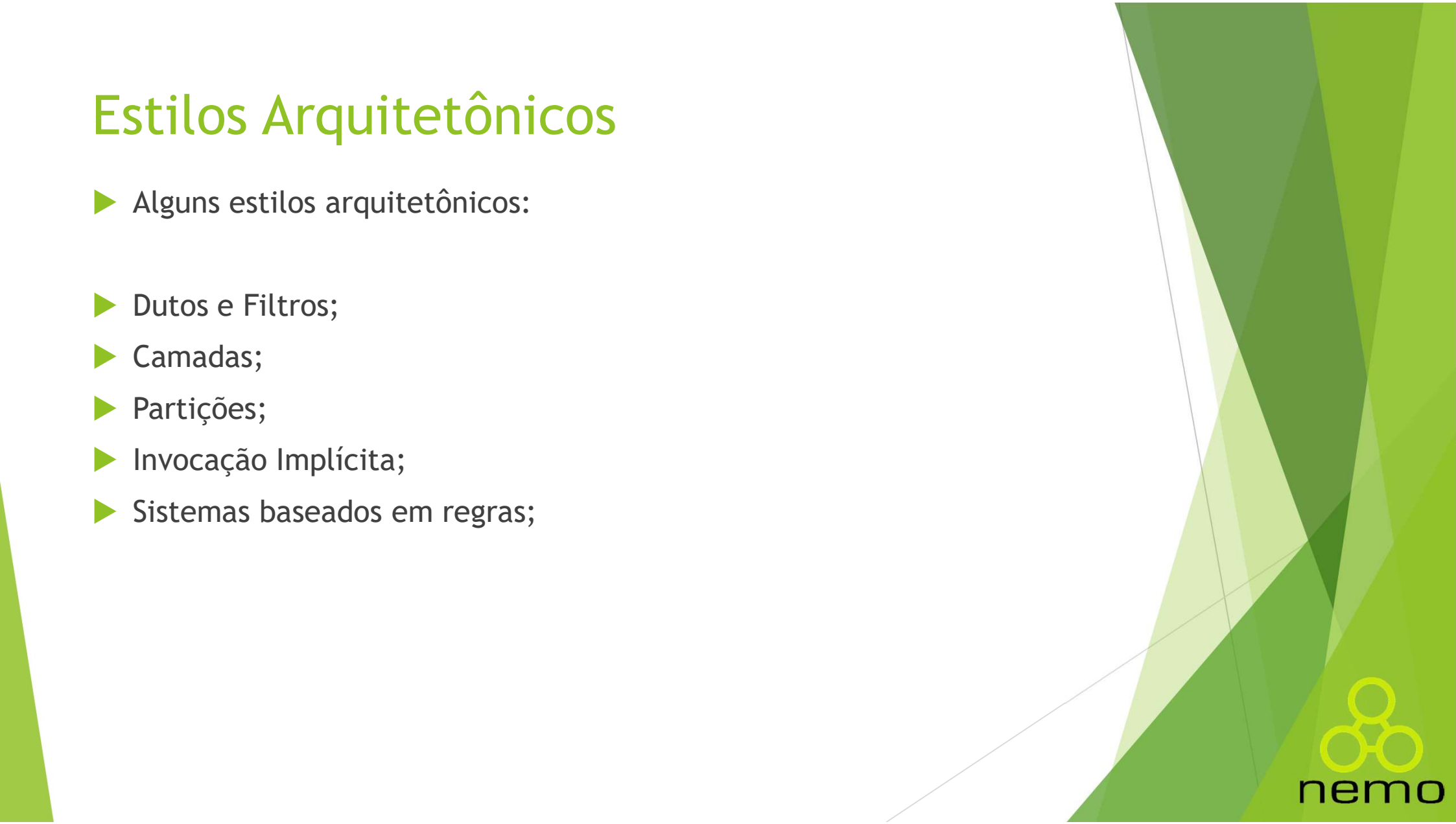
- Especialmente a solução define estratégias para tratar o problema, o que não acontece com os estilos arquitetônicos.

- Assim, normalmente, estilos têm uma apresentação mais livre e são descritos de maneira mais abstrata que padrões arquitetônicos.

Estilos Arquitetônicos

- Alguns estilos arquitetônicos:

- Dutos e Filtros;
- Camadas;
- Partições;
- Invocação Implícita;
- Sistemas baseados em regras;



Estilos Arquitetônicos

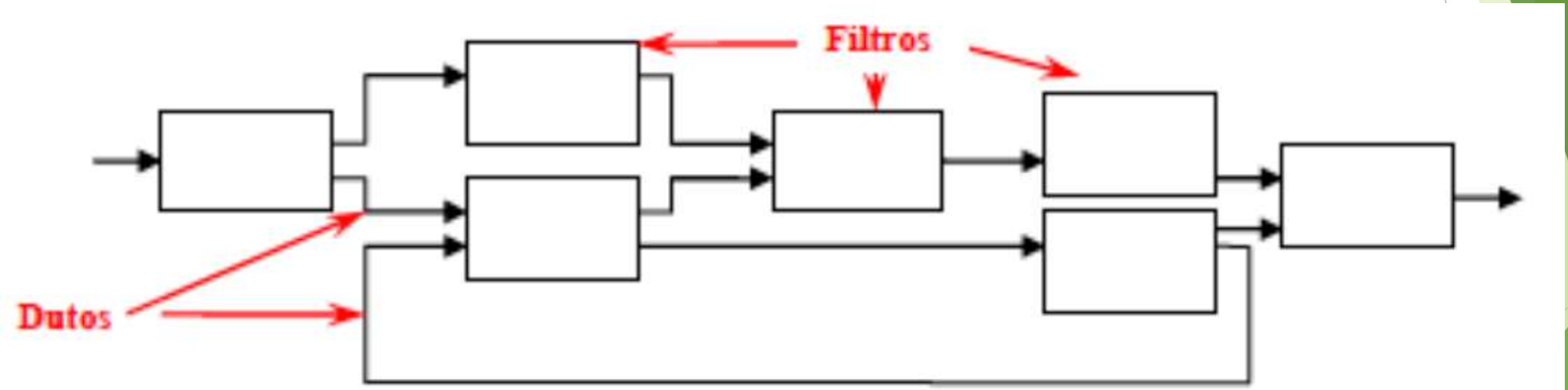
- **Dutos e Filtros**

- O estilo “dutos e filtros” (pipes and filters) considera a existência de uma rede pela qual dados fluem de uma extremidade (origem) até a outra (destino).

- O fluxo de dados se dá através dos dutos e os dados sofrem transformações nos filtros. Um duto provê uma forma unidirecional de fluxo de dados, atuando como um condutor entre dois componentes: do componente origem para o componente destino.

Estilos Arquitetônicos

- **Dutos e Filtros**



nemo

Estilos Arquitetônicos

- **Dutos e Filtros**

- Cada componente (filtro) possui um conjunto de entradas e um conjunto de saídas. Um componente lê dados de suas entradas e produz dados em suas saídas, realizando alguma transformação local. Os filtros têm de ser entidades independentes e não podem compartilhar estado (informações internas) com outros filtros.

- Vantagens:

- • Permite que o projetista compreenda o comportamento geral de um sistema como uma composição simples dos comportamentos dos filtros individuais.

Estilos Arquitetônicos

- **Dutos e Filtros**

- • Facilita o reúso. Quaisquer dois filtros podem ser conectados, desde que eles estejam de acordo em relação aos dados sendo transmitidos entre eles.

- • Facilita a manutenção e o crescimento. Novos filtros podem ser adicionados a um sistema existente, bem como filtros podem ser substituídos por outros melhores ou atualizados.

- • Suporta execução concorrente. Cada filtro pode ser implementado como uma tarefa separada e potencialmente executada em paralelo com outros filtros.

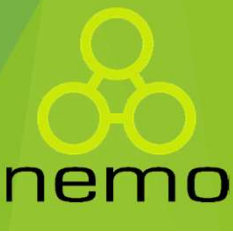
Estilos Arquitetônicos

- **Dutos e Filtros**

- Desvantagens:

- • Apesar dos filtros poderem processar dados de forma incremental, eles são inerentemente independentes e, portanto, o projetista deve pensar cada filtro como provendo uma transformação completa dos dados de entrada para dados de saída.

- • Devido a seu caráter transformacional, este estilo não cai bem para tratar aplicações interativas.



Estilos Arquitetônicos

- **Camadas**

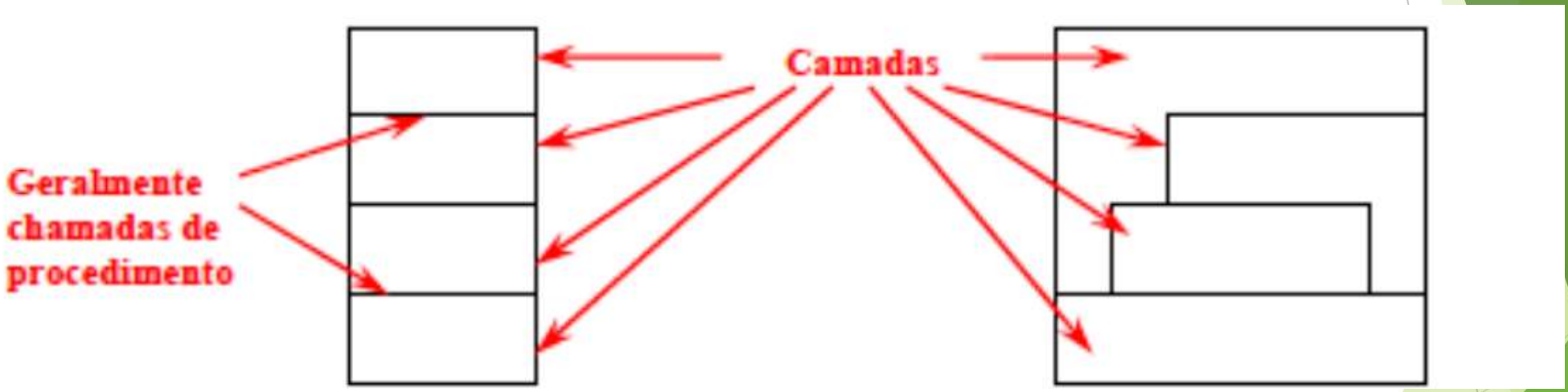
- No estilo em camadas, um sistema é organizado hierarquicamente, como um conjunto ordenado de camadas, cada uma delas construída em função das camadas inferiores e fornecendo serviços para as camadas superiores.

- O conhecimento é unidirecional, i.e., uma camada conhece as camadas inferiores, mas não tem qualquer conhecimento sobre as camadas superiores.

- Há uma relação cliente-servidor entre uma camada usuária de serviços e suas camadas inferiores, provedoras de serviços. Cada camada acrescenta um nível de abstração sobre a camada inferior.

Estilos Arquitetônicos

- Camadas



Arquitetura fechada

Arquitetura aberta

nemo

Estilos Arquitetônicos

- **Camadas**

- Vantagens:

- • Apoiam o projeto baseado em níveis crescentes de abstração, permitindo dividir um problema complexo em camadas, tratando de diferentes preocupações.

- • Tratam bem a manutenibilidade e a possibilidade de crescimento, especialmente as arquiteturas fechadas, pois alterações em uma camada afetam apenas a camada adjacente superior.

- • Apoiam o reúso e diferentes implementações da mesma camada podem ser usadas alternativamente, desde que provejam as mesmas interfaces para as camadas superiores.

Estilos Arquitetônicos

- **Camadas**

- Desvantagens:

- • Nem todos os sistemas são facilmente estruturados em camadas, sendo muitas vezes difícil encontrar os níveis de abstração corretos. Mesmo quando isso é possível, considerações de desempenho podem colocar obstáculos, sobretudo para o uso de uma arquitetura fechada.

- • Pode não ser fácil definir o número adequado de camadas. Esse número dependerá não só da funcionalidade a ser provida pelo sistema, mas também dos requisitos não funcionais desejados.

Estilos Arquitetônicos

- **Partições**

- Partições dividem um sistema verticalmente em subsistemas fracamente acoplados, cada um fornecendo, normalmente, um tipo de serviço.

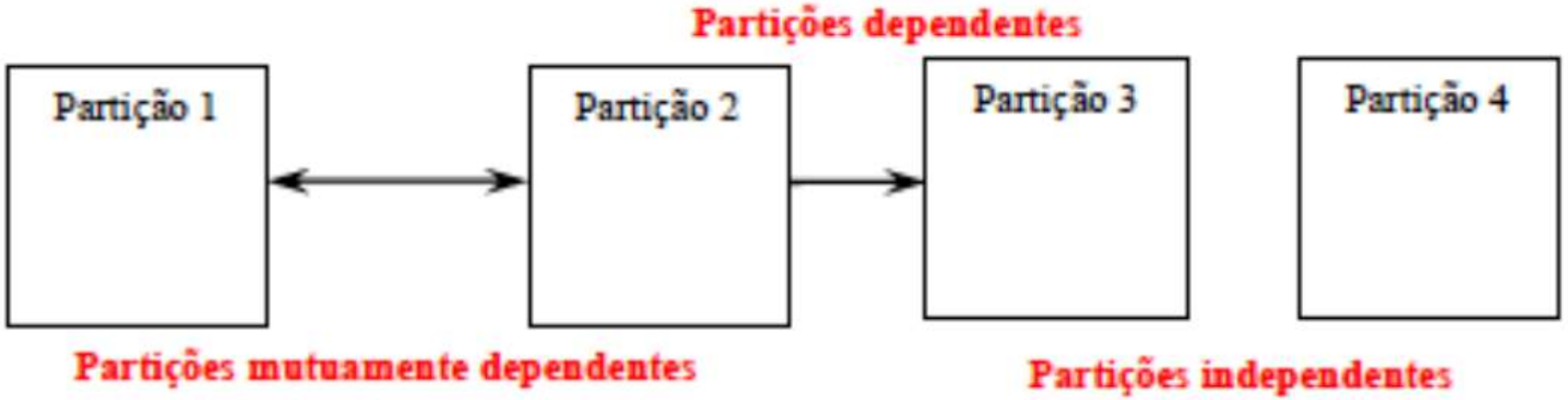
- As partições podem ter algum conhecimento acerca das outras, mas esse conhecimento não é profundo e evita maiores dependências de projeto.

- Ao contrário das camadas, que variam em seu nível de abstração, as partições simplesmente dividem um sistema em partes, todas tendo um nível de abstração semelhante.

- Outra diferença entre camadas e partições é que as camadas dependem umas das outras, enquanto as partições podem ser tanto independentes como dependentes e eventualmente mutuamente dependentes.

Estilos Arquitetônicos

- Partições



nemo

Estilos Arquitetônicos

- **Invocação Implícita**

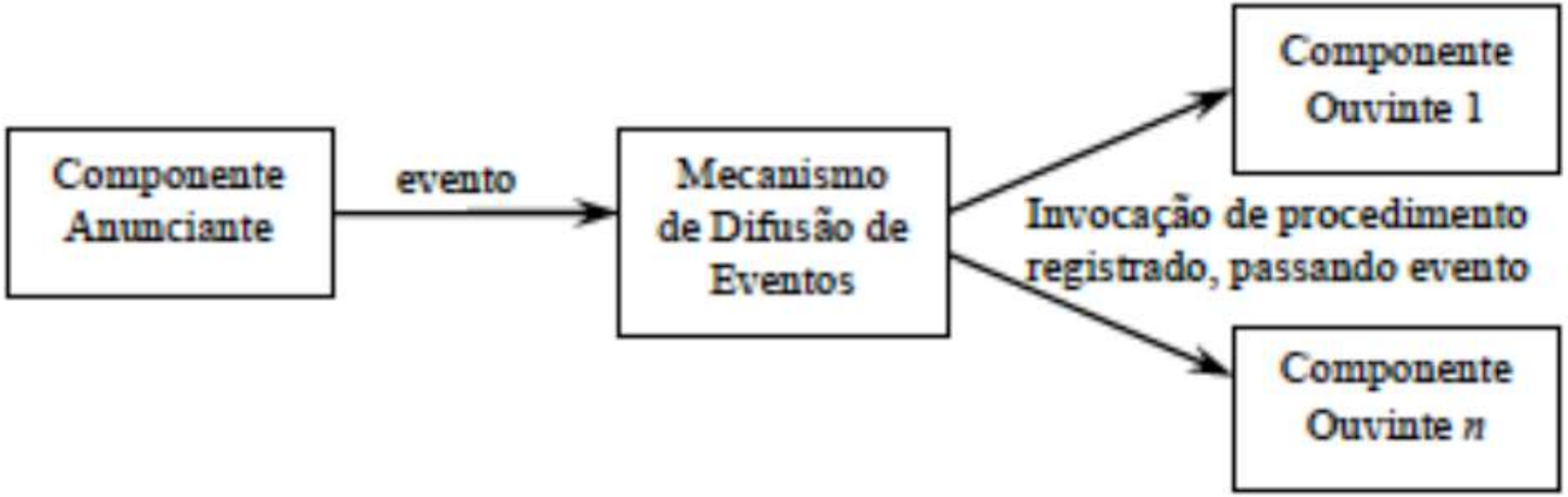
- O estilo arquitetônico invocação implícita (ou baseado em eventos) requer que os componentes interessados em um evento registrem-se a fim de recebê-lo.
- Componentes podem tanto registrar interesse em receber eventos quanto em divulgar eventos.

- A ideia por trás da invocação implícita é a de que um componente pode anunciar um ou mais eventos, enquanto outros componentes podem registrar interesse por um evento, associando um procedimento a ele.

- Quando um evento é anunciado, o próprio sistema invoca todos os procedimentos registrados para o evento. Assim, o anúncio de um evento implicitamente provoca a invocação de procedimentos em outros componentes.

Estilos Arquitetônicos

- **Invocação Implícita**



nemo

Estilos Arquitetônicos

- **Sistemas Baseados em Regras**
- Um sistema baseado em regras utiliza regras explícitas para expressar o conhecimento do domínio de um problema e permite, através da confrontação do conhecimento existente em fatos conhecidos sobre um determinado problema, inferir novos fatos a partir dos existentes.

- A principal diferença entre um sistema baseado em regras e um sistema desenvolvido usando uma abordagem tradicional está na maneira como o conhecimento sobre o domínio do problema é codificado.

- Em aplicações tradicionais, o conhecimento sobre o domínio do problema é codificado tanto nas instruções propriamente ditas quanto nas estruturas de dados.
- Já na abordagem de regras, todo o conhecimento relativo ao domínio do problema é codificado exclusivamente nas estruturas de dados. Nenhum conhecimento é armazenado nas instruções ou nos programas propriamente ditos.

Estilos Arquitetônicos

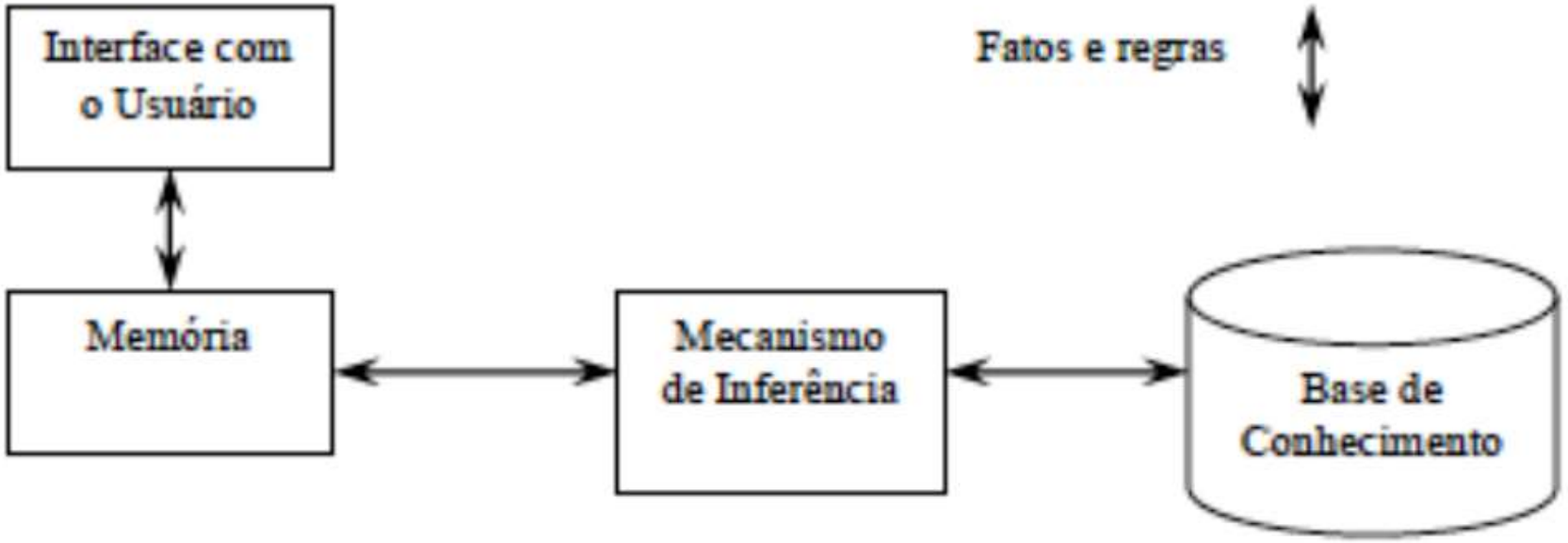
- **Sistemas Baseados em Regras**

- Um sistema baseado em regras utiliza regras explícitas para expressar o conhecimento do domínio de um problema e permite, através da confrontação do conhecimento existente em fatos conhecidos sobre um determinado problema, inferir novos fatos a partir dos existentes.

- A arquitetura geral de um sistema baseado em regras compreende dois componentes principais: um conjunto de declarações totalmente dependentes do domínio do problema, chamado de base de conhecimento, e um programa independente do domínio do problema (apesar de altamente dependente das estruturas de dados), chamado máquina de inferência.

Estilos Arquitetônicos

- **Sistemas Baseados em Regras**



nemo

Padrões Arquitetônicos

- Padrões são um ponto de partida e não um fim em si.
- De início, não é necessário saber detalhes sobre os diversos padrões. É importante ter uma visão geral dos mesmos para saber que problemas eles resolvem e como eles resolvem, de modo a selecioná-los quando forem aplicáveis.
- Uma vez reconhecida a necessidade de se aplicar o padrão, tem-se de descobrir como aplicá-lo ao problema específico que se tem em mãos.
- Muito dificilmente será possível aplicar um padrão cegamente. Padrões são artefatos semiprontos, o que implica na necessidade de completar a solução no contexto específico do projeto. Padrões são relativamente independentes, mas não são isolados uns dos outros, sobretudo os padrões arquitetônicos.

Padrões Arquitetônicos

- Um estilo subjacente aos principais padrões para SIs é o estilo em camadas.

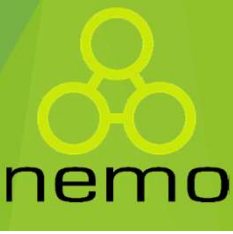
- No que se refere à organização de SIs em camadas, componentes podem ser agrupados em três camadas lógicas principais:

- • **Camada de Apresentação ou de Interface com o Usuário**: sua função é tratar a interação entre o usuário e o sistema. As responsabilidades principais dessa camada são exibir informações para os usuários e interpretar comandos do usuário em ações da lógica de negócio e da persistência de dados.

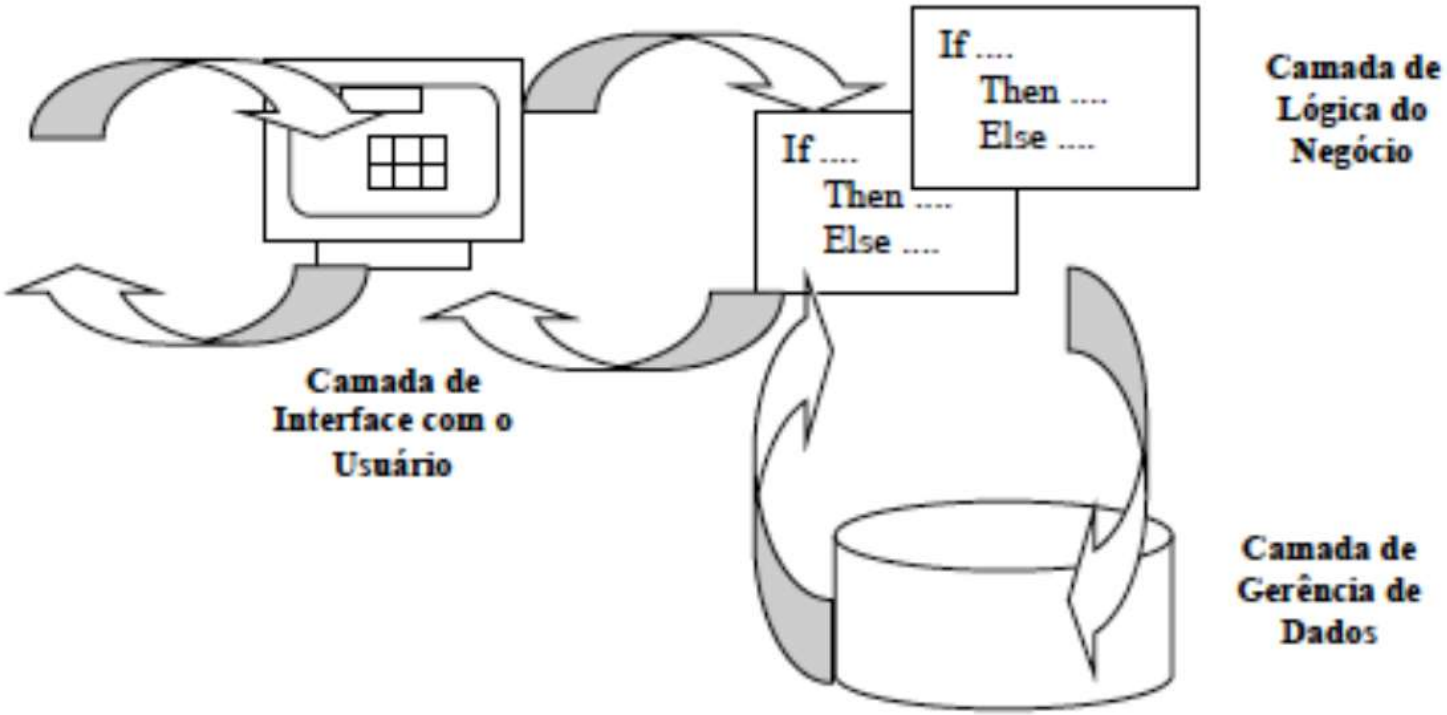
Padrões Arquitetônicos

- • **Camada de Lógica de Negócio**: contém as funcionalidades que apoiam os processos de negócio. Conceitos do domínio, regras de negócio, processamentos e cálculos são encontrados nesta camada.

- • **Camada de Persistência ou de Gerência de Dados**: provê acesso a dados corporativos. É sua responsabilidade gerenciar requisições concorrentes de acesso às bases de dados, assim como a sincronização de elementos de dados distribuídos.



Padrões Arquitetônicos



Padrões Arquitetônicos

- Algumas vezes as camadas são dispostas de modo que a camada de lógica de negócio oculta completamente a camada de gerência de dados da interface com o usuário (arquitetura fechada). Mais frequentemente, contudo, a camada de apresentação acessa a gerência de dados diretamente (arquitetura aberta).

- Geralmente um sistema específico pode possuir vários pacotes de cada um desses três grandes tipos. Uma aplicação pode ser projetada para ser manipulada pelos usuários tanto por meio de uma interface com o usuário rica, quanto por uma interface de linha de comando, tendo, portanto, diferentes pacotes de interface com o usuário para cada um dos propósitos.

Padrões Arquitetônicos

- Via de regra, a seguinte regra sobre dependências deve ser respeitada: as camadas de negócio e de gerência de dados não devem ser dependentes da camada de apresentação.

- Os padrões apresentados em (FOWLER, 2003) são organizados de acordo com essas camadas. Assim, há grupos de padrões relativos à:

- • Camada da Lógica de Negócio: padrões nesse grupo tratam da organização das funcionalidades na camada de lógica de negócio. São eles: Script de Transação (Transaction Script), Modelo de Domínio (Domain Model), Módulo Tabela (Table Module) e Camada de Serviço (Service Layer).

Padrões Arquitetônicos

- • Camada de Apresentação: padrões nesse grupo enfocam a separação dos objetos de interfaces gráficas (visão) do controle da interação com o usuário.

- Exemplos são os padrões Modelo Visão Controlador (Model View Controller) e Controlador de Aplicação (Application Controller). Além disso, há padrões mais focados na visão (p.ex., Template View) e outros mais voltados para o projeto de controladores (p.ex., Controlador Frontal ou, em inglês, Front Controller).

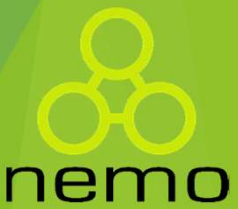
Padrões Arquitetônicos

- • Camada de Gerência de Dados: padrões nesse grupo se referem ao mapeamento objeto relacional, já que a tecnologia dominante de banco de dados para Sistemas de Informação é a dos bancos de dados relacionais.

- Há, dentre outros, padrões relativos a como a lógica de negócio conversa com os bancos de dados, bem como padrões relacionados a como mapear estruturas do mundo de objetos (p.ex., associações e herança) para bancos de dados relacionais.

Projeto de Sistemas de Informação Distribuídos

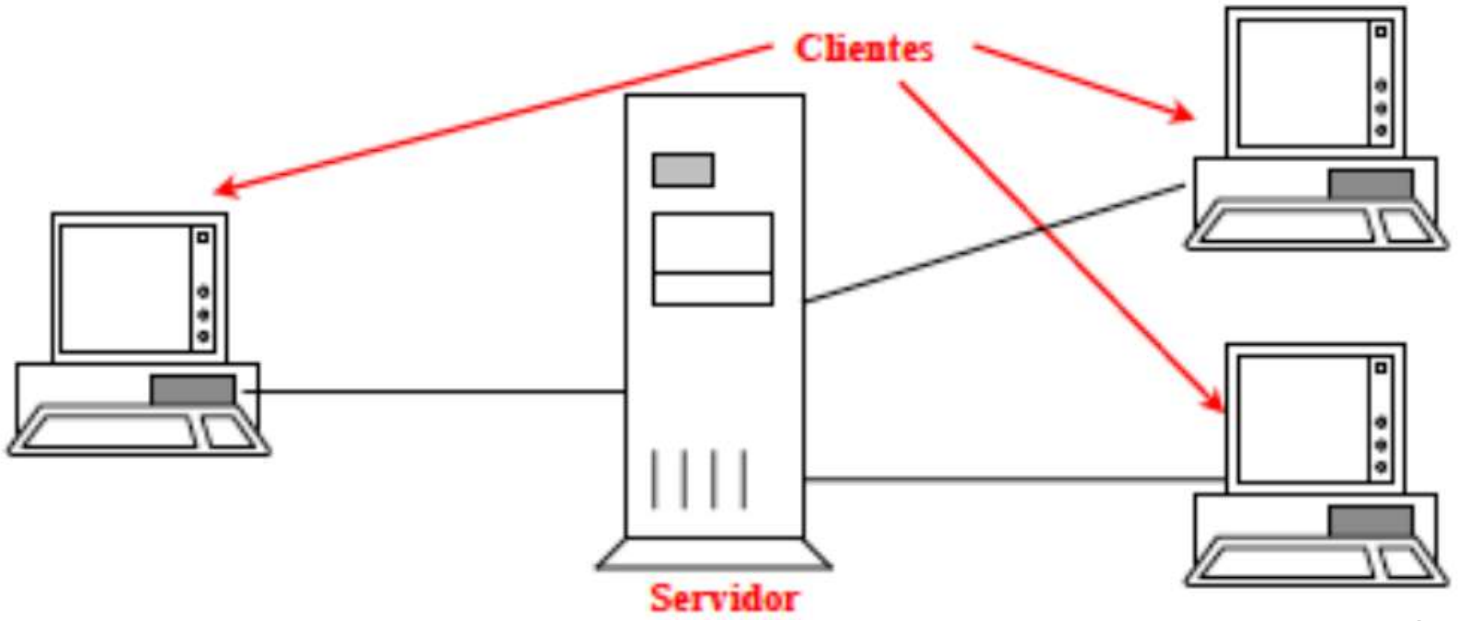
- Tipicamente, SIs, sobretudo de médio a grande porte, são aplicações distribuídas e utilizam arquiteturas ditas cliente-servidor.
- Nessa arquitetura, componentes assumem os papéis de clientes e servidores de serviços.
- Um servidor é um componente que fica em estado de espera, aguardando a solicitação de um serviço por um ou mais clientes. O servidor pode trabalhar de forma síncrona ou assíncrona.
- Os clientes, por sua vez, podem ser vistos como processos independentes, i.e., a execução de seu processo não interfere em outros.



Projeto de Sistemas de Informação Distribuídos

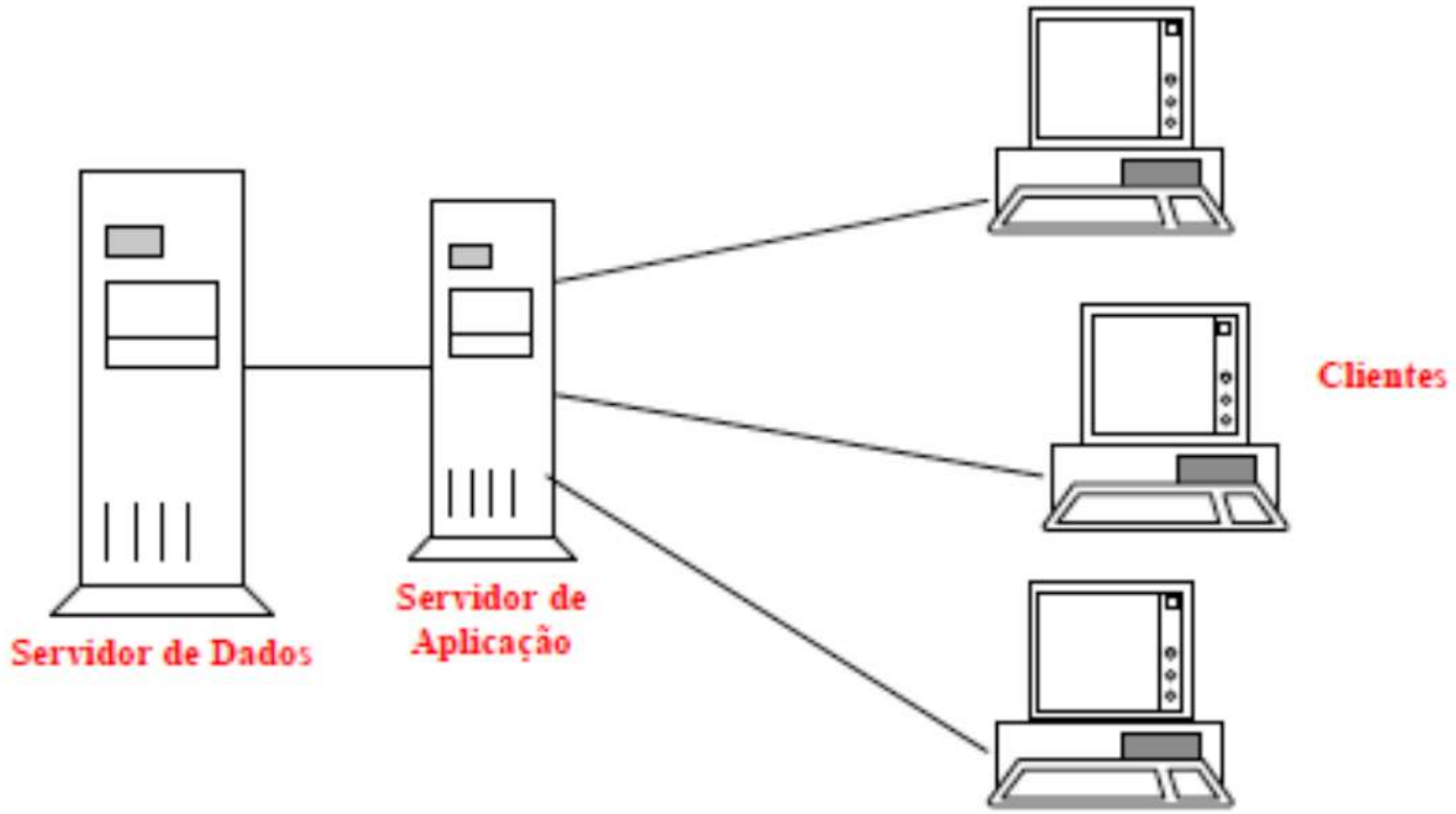
- O estilo em camadas é a base para as arquiteturas cliente-servidor.
- Geralmente, clientes tratam as solicitações dos usuários, transformando-as em algum protocolo e as transmitindo para um servidor. O servidor processa as solicitações e retorna respostas para o cliente. Por fim, o cliente envia as respostas para o usuário.
- Em sua forma mais simples, a arquitetura cliente-servidor envolve múltiplos clientes fazendo requisições para um único servidor, o que caracteriza uma arquitetura em duas camadas (two-tier).

Projeto de Sistemas de Informação Distribuídos



nemo

Projeto de Sistemas de Informação Distribuídos



nemo

Projeto de Sistemas de Informação Distribuídos

- Hoje, reconhece-se a necessidade de separar a camada de lógica do negócio das demais camadas.

- Essa separação traz várias vantagens, dentre elas:

- • Reusabilidade: Classes de negócio tendem a ser complexas e podem desempenhar diferentes papéis dentro dos sistemas corporativos. A meta é criar classes que garantam as regras de negócio para uma particular classe de negócio e reutilizá-las em todos os contextos que lidem com a mesma.

Projeto de Sistemas de Informação Distribuídos

- • Portabilidade: ao se mover a lógica de negócio para uma camada separada, permite-se tirar proveito de diferentes plataformas de hardware e software, sem ter que reescrever grandes porções do código.

- • Manutenibilidade: o fato da lógica de negócio estar concentrada em uma camada única da arquitetura de software facilita a localização das alterações e evoluções a serem feitas no sistema. O mesmo pode se dizer em relação à interface com o usuário e à gerência de dados.

Projeto de Sistemas de Informação Distribuídos

- É preciso, portanto, escolher onde cada uma das camadas lógicas vai rodar (camadas físicas).
- De maneira geral, a gerência de dados ficará no servidor.
- No que se refere à camada de apresentação, a decisão depende principalmente do tipo desejado de interface com o usuário.
- A opção mais simples é rodar tudo no servidor, cabendo ao cliente apenas a apresentação de uma interface HTML (front end) rodando em um navegador Web. Essa solução facilita a manutenção e a implantação de novas versões, mas compromete a rapidez com que o sistema reconhece uma solicitação do usuário (responsiveness), uma vez que a solicitação tem de trafegar, ida e volta, para o servidor.

Projeto de Sistemas de Informação Distribuídos

- Por fim, em relação à camada de domínio, pode-se rodar tudo no cliente (cliente pesado), tudo no servidor (servidor pesado) ou dividi-la.

- Rodar tudo no servidor é a melhor opção do ponto de vista da manutenibilidade.

- Dividir a lógica de negócio entre o servidor e o cliente traz uma dificuldade adicional na manutenibilidade do sistema: saber onde certa porção da aplicação está rodando.
- Neste caso, devem-se isolar as porções que vão rodar no cliente em módulos autocontidos que sejam independentes das outras partes do sistema.

Projeto de Sistemas de Informação Distribuídos

- Levando-se em consideração os aspectos discutidos anteriormente, o projeto de sistemas de informação distribuídos envolve questões adicionais, dentre elas:

- • a definição de requisitos de distribuição geográfica do sistema,

- • a definição das máquinas clientes, servidoras e da configuração e número de camadas de hardware cliente-servidor,

- • a localização dos processos,
- • a localização dos dados físicos e as estratégias de sincronização para dados distribuídos geograficamente.

Projeto de Sistemas de Informação Distribuídos

- Para responder a essas (e outras) questões, é importante levantar algumas informações a cerca de:

- • volumes de dados, frequência de disparo de casos de uso e expectativas de tempos de resposta para os mesmos;

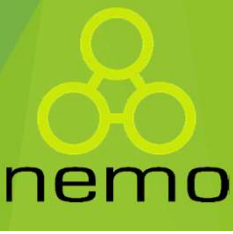
- • topologia geográfica do negócio e necessidades de distribuição geográfica da computação, incluindo a necessidade de distribuição de dados e de processos nos diferentes locais.

Efetuando Estimativas

- Determinar a arquitetura mais apropriada para um sistema de informação distribuído envolve a quantificação da capacidade de computação necessária para o problema.

- O modelo de análise provê a base necessária para a realização de estimativas dos requisitos de computação do sistema final.

- Assim, uma atividade do projeto de SIs consiste em completar as informações de análise, fazendo estimativas importantes acerca do modelo estrutural e do modelo de casos de uso.



Efetuando Estimativas – Modelo Conceitual

- A realização de estimativas sobre o modelo conceitual estrutural (diagrama de classes) é importante para a definição da arquitetura, sobretudo no que concerne à escolha de táticas a serem aplicadas.

- Uma importante estimativa é o tamanho da base de dados do sistema.

- Essa informação é útil para apoiar a definição de quais estratégias adotar em relação ao armazenamento e a comunicação de dados.

Efetuando Estimativas

- Para se estimar o tamanho da base de dados de um sistema, duas informações são essenciais: *o tamanho de uma instância de uma classe* (para cada classe) e o *número estimado de instâncias que poderão ser acumuladas ao longo do tempo*.

- Para determinar o tamanho de um objeto, devem-se observar os tipos de dados de atributo da classe.

- Já a estimativa do número de instâncias esperadas não é tão simples. Algumas classes, tais como Pedido e Itens de Pedido, têm potencial de crescimento ilimitado.

Efetuando Estimativas

- Para essas classes, o modelo de casos de uso pode dizer quais casos de uso criam instâncias das mesmas.

- Para estimar o tamanho das bases de dados relativas a essas classes, é necessário saber quão frequentemente o caso de uso ocorre e o período de retenção da informação na base de dados.

Efetuando Estimativas

- Em suma, as seguintes informações para cada classe do modelo estrutural devem ser coletadas:

- • tamanho estimado de uma instância, calculado pelo somatório dos tipos de dados de cada atributo,

- • taxa de ocorrência dos casos de uso que criam novas instâncias da classe,

- • período de retenção.

Efetuando Estimativas

- Uma vez que é importante saber que tipo de operação um caso de uso pode realizar em uma classe, é útil produzir uma matriz CRUD (*Create*, *Retrieve*, *Update*, *Delete* - Criar, Recuperar, Atualizar, Eliminar) *Caso de Uso x Classe*, mostrando se, por ocasião da ocorrência do caso de uso, o sistema vai criar, atualizar, recuperar ou eliminar instâncias da classe.

	Classes		
Casos de Uso	Cliente	Pedido	Item de Pedido
Efetuar pedido	R	C	C
Cancelar pedido	R	RD	RD
Alterar pedido	R	RU	CRUD

nemo

Efetuando Estimativas – Casos de Uso

- Um modelo de casos de uso captura as funcionalidades de um sistema segundo uma perspectiva externa, isto é, dos usuários.

- Em um modelo de casos de uso, alguns casos de uso são mais críticos do que outros em termos do negócio da organização. Além disso, cada caso de uso pode ter requisitos não funcionais específicos.

- No que se refere a desempenho, pode ser importante levantar informações acerca da frequência de ocorrência (ou disparo) e o tempo de resposta desejado para um caso de uso.

Efetuando Estimativas – Casos de Uso

- O objetivo é identificar os casos de uso críticos, já que eles serão objeto de atenção especial. Para esses, devem ser levantadas informações sobre a frequência média de ocorrência, a taxa de pico e o período de tempo do pico.

- Se a ocorrência dos eventos for uniforme ao longo do tempo, a frequência média de disparo aponta para o nível normal de operação do sistema.

- Para eventos irregulares, contudo, é necessário conhecer as taxas de pico e o período em que esses picos ocorrem.

Efetuando Estimativas – Casos de Uso

- Além disso, uma importante questão se coloca: **o sistema tem de ser dimensionado para tratar picos, de modo a sempre atender satisfatoriamente às necessidades dos usuários?**
- Se o sistema for dimensionado pela capacidade de pico, provavelmente ficará ocioso durante grande parte do tempo.
- Por outro lado, se for dimensionado pela média, nos momentos de pico não atenderá totalmente às necessidades dos usuários.
- Esta não é uma decisão fácil e não deve ser tomada pelo projetista. Ao contrário, essa decisão cabe, principalmente, ao cliente.

Determinando Requisitos de Distribuição Geográfica

- Não é necessário analisar todos os casos de uso de um sistema. É suficiente concentrar a atenção nos principais casos de uso do negócio, aqueles com os maiores impactos sobre o cliente, com os maiores volumes de dados e com as localizações mais remotas geograficamente.

- Outro passo importante do projeto arquitetônico consiste em examinar a distribuição geográfica dos casos de uso, o que conduz naturalmente à distribuição necessária dos dados.

Determinando Requisitos de Distribuição Geográfica

- Juntos, a frequência de disparo dos casos de uso, volume de dados, restrições de tempo de resposta e a distribuição geográfica do negócio formarão a base para se determinar uma arquitetura aceitável para o sistema em questão.

- Algumas matrizes podem ser usadas para mapear o modelo de análise na topologia de localizações do negócio.

	Localização		
Casos de Uso	Internet	Matriz	Filiais
Efetuar pedido		X	X
Cancelar pedido	X	X	
Alterar pedido	X	X	

nemo

Determinando Requisitos de Distribuição Geográfica

- Uma vez levantadas as necessidades de computação de cada localização do negócio, pode-se avaliar qual a melhor estratégia para a distribuição de dados.

- Algumas opções são manter todos os dados em uma única base de dados centralizada, descentralizar completamente os dados, ou usar uma abordagem de fragmentação.

Determinando Requisitos de Distribuição Geográfica

- Quando a estratégia de uma única base de dados centralizada é adotada, os dados são mantidos em um servidor de dados central e todas as aplicações que necessitem acessá-los devem fazer suas consultas e atualizações no servidor central.

- Os benefícios são numerosos, tais como:

- (i) os dados estão sempre atualizados e não há redundância;

- (ii) o projeto é mais simples, tendo em vista que a segurança pode ser mantida centralmente e nenhuma rotina de sincronização é requerida; e

- (iii) é fácil fazer cópias de segurança dos dados.

Determinando Requisitos de Distribuição Geográfica

- Entretanto, há também desvantagens, tais como:

- (i) não permite operação desconectada e, portanto, a disponibilidade do sistema é fortemente afetada pela comunicação de dados e pela disponibilidade do servidor de dados;

- (ii) o desempenho do sistema também é muito dependente da velocidade de comunicação de dados.

Determinando Requisitos de Distribuição Geográfica

- Uma solução diametralmente oposta consiste em ter os dados totalmente descentralizados e replicados. Neste caso a base de dados é completamente replicada em todos os locais que dela necessitem.

- Atualizações em um local podem ser irradiadas (broadcast) para outros locais em tempo real.

- São benefícios dessa estratégia:
- (i) o tempo de resposta não é onerado pelo tráfego em rede; e

- (ii) a disponibilidade também tem sua dependência diminuída em relação à comunicação de dados.

nemo

Determinando Requisitos de Distribuição Geográfica

- Contudo, há diversos problemas, dentre eles:
- (i) o tráfego global na rede aumenta devido à replicação de dados em todos os locais;
- (ii) rotinas complexas de sincronização são necessárias para manter as várias cópias da base de dados atualizadas;
- (iii) podem surgir problemas se o mesmo registro for atualizado em dois locais;

Determinando Requisitos de Distribuição Geográfica

- (iv) se um dos servidores cair ou o software de replicação falhar, pode ser difícil reconstruir o conjunto dos dados e reaplicar atualizações na ordem correta;
- (v) procedimentos de backup tornam-se mais complexos, uma vez que não se tem uma visão clara de qual é a base de dados principal;
- e (vi) dados completamente replicados conduzem a uma redundância desnecessária de dados.

Determinando Requisitos de Distribuição Geográfica

- Uma abordagem intermediária entre a centralização e a replicação total é a fragmentação de dados, na qual apenas os dados necessários para cada localidade são mantidos localmente.

- A fragmentação pode ser: vertical, horizontal, híbrida.

Determinando Requisitos de Distribuição Geográfica

- • **Vertical**: ocorre quando apenas certos elementos (classes / atributos das classes) são fisicamente distribuídos a locais remotos.
- Cada localidade possui apenas aqueles elementos que são requeridos pelos casos de uso que lá ocorrem.
- Isto reduz tráfego em rede, pois apenas os elementos de dados necessários precisam ser sincronizados com outros locais.
- Entretanto, essa estratégia pode ser bastante complexa para gerenciar. Os procedimentos de replicação devem ser capazes de sincronizar atualizações atributo-a-atributo em diferentes locais.

Determinando Requisitos de Distribuição Geográfica

- • **Horizontal**: ocorre quando apenas algumas instâncias de uma classe são fisicamente distribuídas a locais remotos.

- Esta estratégia é empregada tipicamente quando localidades têm seus próprios dados que não são manipulados em outras localidades.
- Cada localidade tem sua própria cópia do banco de dados. Os esquemas são idênticos, mas os dados que povoam as bases de dados são diferentes.

- Geralmente, há uma base de dados principal que contém todos os registros.
- Assim como a fragmentação vertical, a horizontal diminui o tráfego na rede, eliminando transferências de dados desnecessárias.
- Contudo, o processo de sincronização é também de alguma forma complicado, principalmente quando diferentes locais compartilham registros.

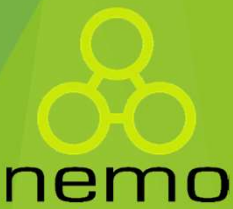
nemo

Determinando Requisitos de Distribuição Geográfica

- • **Híbrida**: bases de dados distribuídas compartilham os mesmos tipos de entidades lógicas, mas possuem diferentes atributos e instâncias.

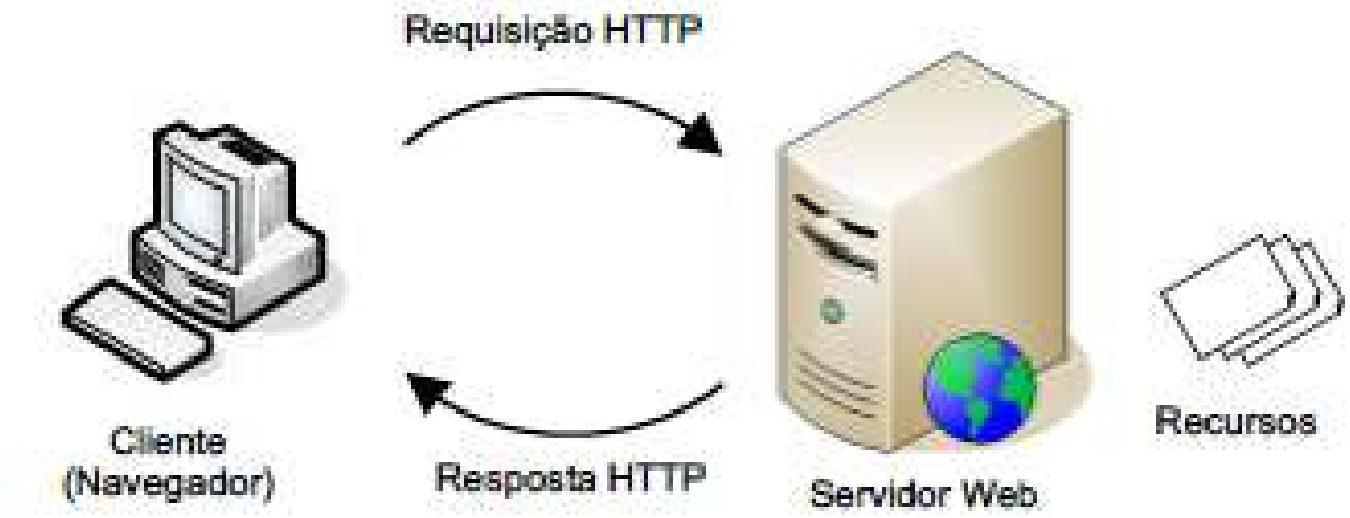
- Cada local possui apenas os atributos e instâncias que são realmente necessários para os eventos que ali ocorrem.

- É fácil perceber que tal estratégia pode ser muito difícil de gerenciar.



Aplicações Web

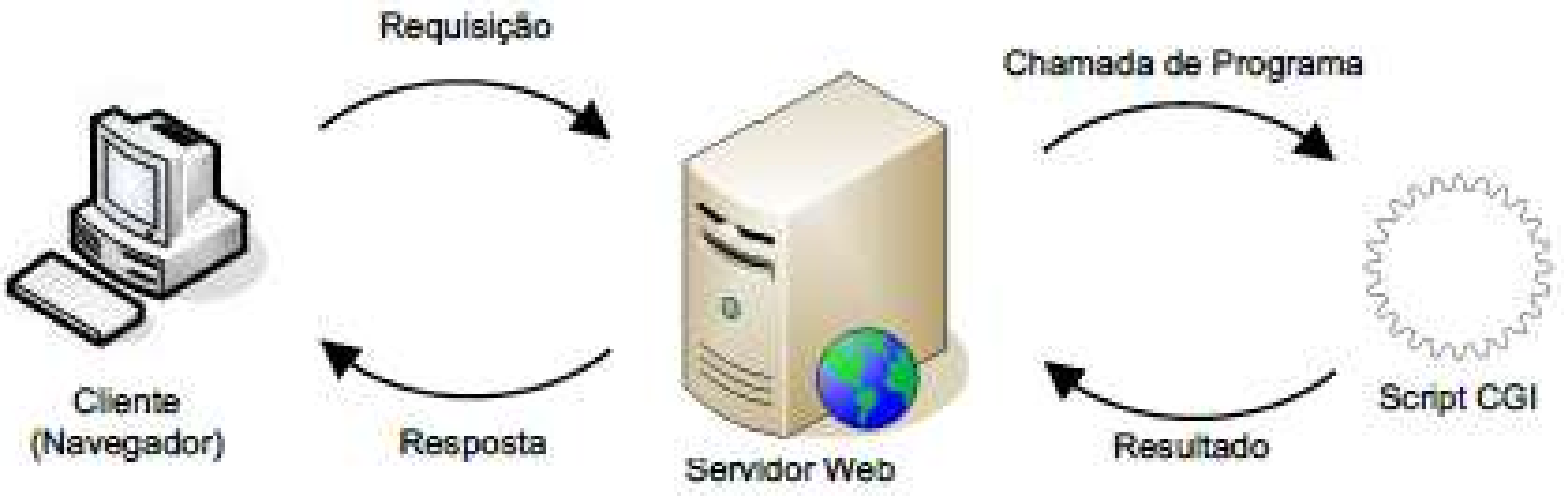
- De maneira simplista, uma aplicação Web consiste em um conjunto de páginas Web para interação com o usuário, armazenando, processando e provendo informações.



nemo

Aplicações Web

- O estabelecimento de tecnologias XML e algumas extensões às arquiteturas cliente-servidor tradicionais pavimentaram o caminho para o desenvolvimento de páginas Web dinâmicas, compostas em tempo de execução a partir de conteúdo extraído dinamicamente de uma fonte de dados, o qual é usado para produzir a apresentação final da página.



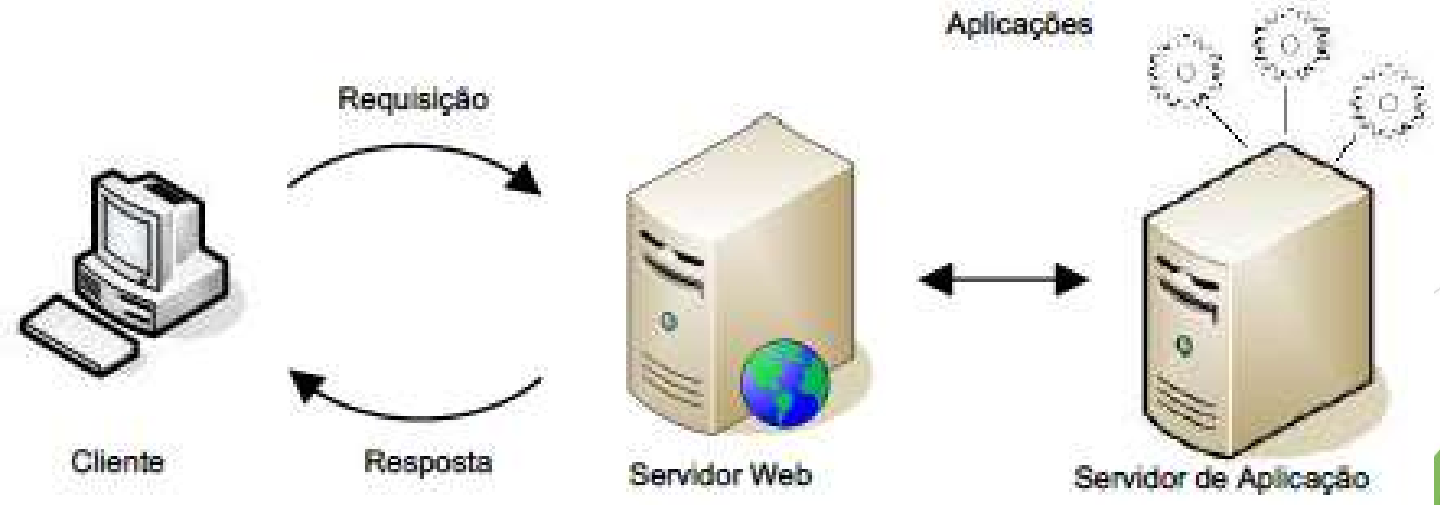
nemo

Aplicações Web

- Entretanto, a arquitetura CGI tem algumas desvantagens significativas, que a tornam impraticável na maioria das situações.
- O principal problema é o desempenho: para cada requisição HTTP para um script CGI, o servidor Web inicia um novo processo, o qual é terminado no fim da execução do script.
- A criação e a finalização de processos são atividades muito custosas, as quais se tornam rapidamente um gargalo.
- Além disso, o fato de terminar o processo após a execução do script CGI, após cada requisição, impede que as informações sobre a interação com o usuário sejam mantidas entre requisições de usuário consecutivas (a menos que elas sejam armazenadas em uma base de dados), o que tem impacto novamente no desempenho.

Aplicações Web

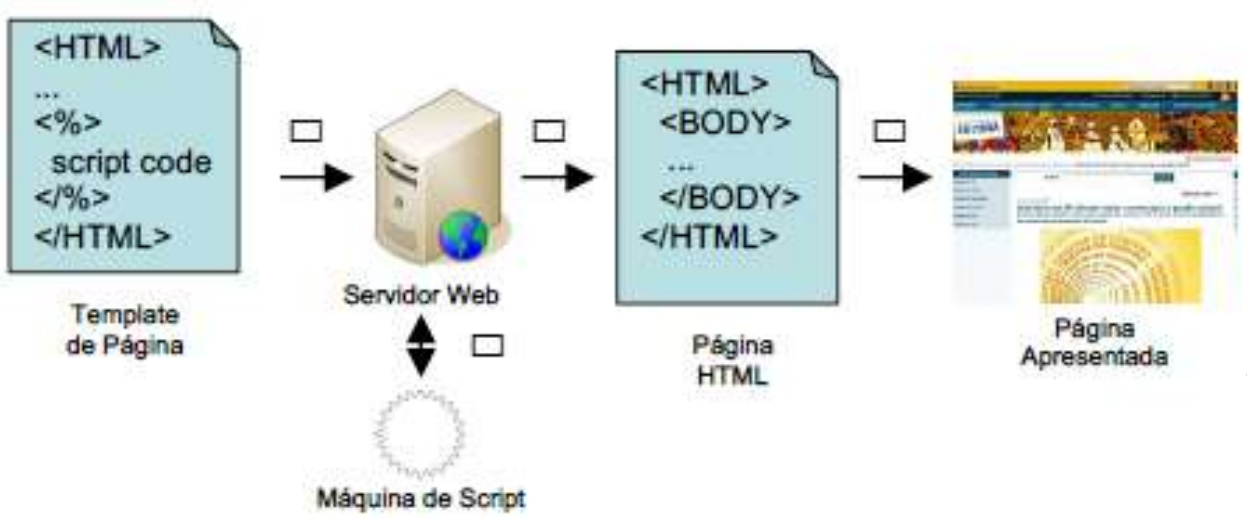
- As limitações da arquitetura CGI podem ser superadas pela extensão das capacidades do servidor Web com uma máquina capaz de executar aplicações, na qual os programas para computar as respostas HTTP podem ser executados de maneira eficiente, sem serem terminados após cada requisição, e os recursos compartilhados podem ser associados a uma ou mais aplicações e concorrentemente acessados por múltiplos usuários.



nemo

Aplicações Web

- (i) o servidor Web encaminha o template para a máquina de script;
- (ii) a máquina de script processa as instruções embutidas, determina as partes dinâmicas da página e as inclui no resultado, uma página HTML plana;
- (iii) a página gerada é retornada para o servidor Web;
- (iv) este, por sua vez, encaminha a página para o cliente (navegador), onde é apresentada (renderizada).



nemo

Arquiteturas Multicamadas de Aplicações Web

- Uma aplicação Web pode ser considerada um tipo de sistema distribuído, com uma arquitetura cliente-servidor com as seguintes características:

- • Grande número de usuários distribuídos;
- • Ambiente de execução heterogêneo composto de vários tipos de hardware, conexões de rede, sistemas operacionais, servidores e navegadores;
- • Natureza extremamente heterogênea, que depende da grande variedade de componentes que a constituem.
- • Habilidade de gerar componentes em tempo de execução, de acordo com a entrada do usuário e status do servidor.

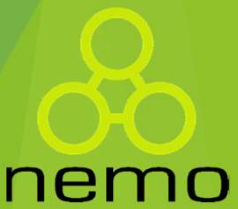
Arquiteturas Multicamadas de Aplicações Web

- Arquiteturas de software de aplicações Web dessa natureza são normalmente organizadas em três camadas de software:

- • **Camada de Apresentação**: responsável por processar as requisições vindas do cliente e construir as páginas HTML.

- É desenvolvida usando extensões de servidores Web, as quais são capazes de construir dinamicamente as páginas HTML a serem enviadas como resposta para o cliente.

- Essas páginas são geradas tomando por base os dados produzidos pela execução de componentes de negócio, que estão na camada abaixo.



Arquiteturas Multicamadas de Aplicações Web

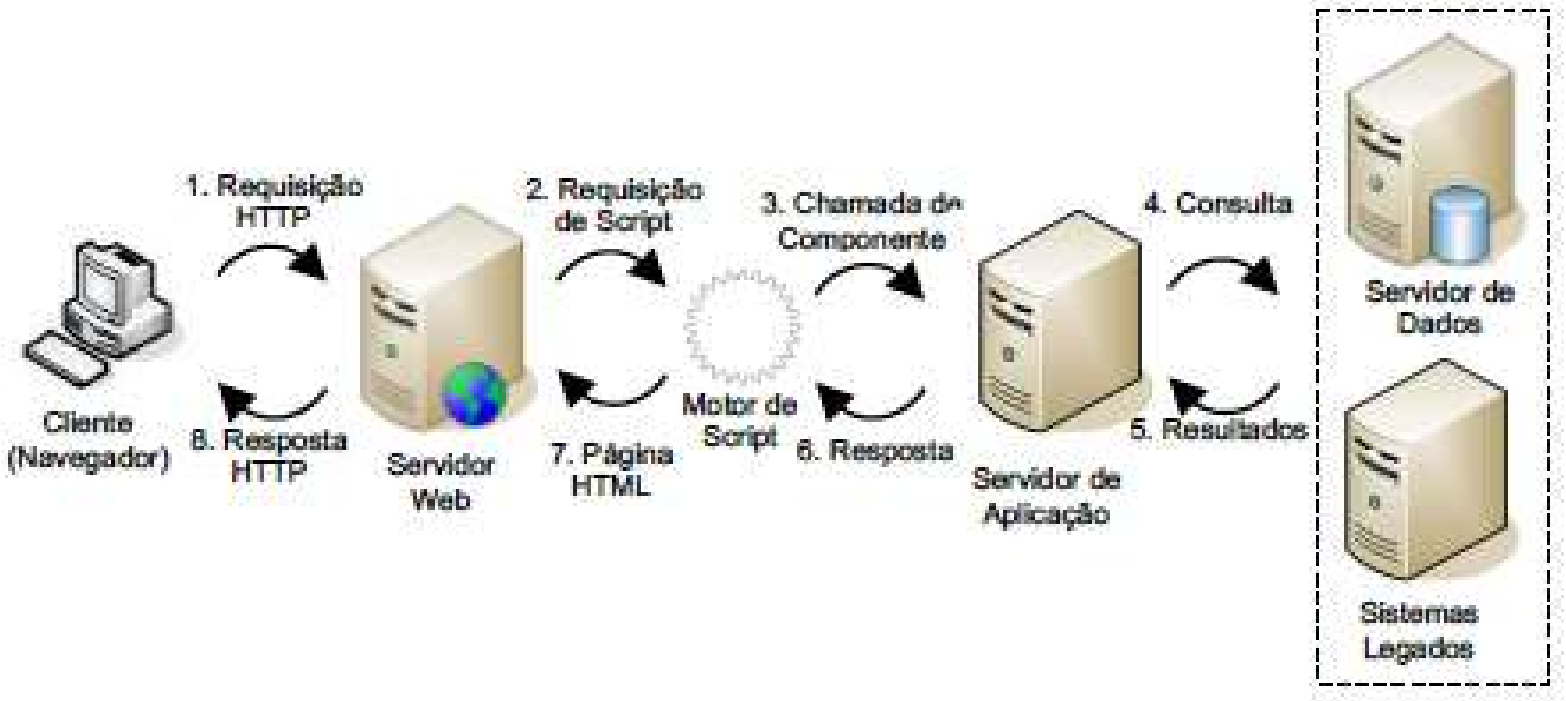
- • **Camada de Lógica de Negócio**: responsável por executar componentes que realizam a lógica de negócio da aplicação.

- Para tal, comunica-se com a camada de gerência de recursos para acessar bases de dados persistentes, sistemas legados ou para invocar serviços externos.

- • **Camada de Gerência de Recursos**: representa o conjunto de serviços oferecidos por diferentes sistemas, tais como aqueles suportando o acesso a bases de dados, a outros sistemas ou, de maneira geral, a serviços Web externos.

Arquiteturas Multicamadas de Aplicações Web

- Para tornar real essa arquitetura de software de três camadas, é necessário um ambiente de execução suportando comunicação em camadas:



nemo

Aplicações Ricas para Internet - RIAs

- O termo RIA designa uma classe de soluções caracterizada pelo propósito comum de adicionar novas características às aplicações Web tradicionais.

- Essas características incluem:
- • Interfaces com o usuário sofisticadas, incluindo facilidades do tipo arrastar e soltar (drag & drop), animações e sincronização multimídia, buscando dar às aplicações Web uma interatividade semelhante à das aplicações desktop.
- • Mecanismos para minimizar a transferência de dados entre cliente e servidor, movendo a gerência das camadas de apresentação e interação do servidor para o cliente.
- • Armazenamento e processamento de dados tanto no lado do cliente quanto no lado do servidor, buscando melhor utilizar a capacidade de computação do cliente.

Aplicações Ricas para Internet - RIAs

- Do ponto de vista da distribuição de dados, nas RIAs, os dados da aplicação podem ser armazenados tanto no cliente quanto no servidor.
- No caso dos dados serem armazenados no cliente, estes são enviados ao servidor ao final de cada operação.
- Do ponto de vista da distribuição da lógica de negócio, nas RIAs tanto o cliente quanto o servidor podem realizar operações complexas.
- Do ponto de vista da comunicação cliente-servidor, as RIAs permitem tanto a comunicação síncrona quanto a comunicação assíncrona.

Aplicações Ricas para Internet - RIAs

- Finalmente, do ponto de vista de aprimoramento da interface com o usuário, as RIAs possibilitam apresentação e interações com o usuário avançadas e menor tempo de resposta.

- A aplicação pode operar como aplicação de página única, evitando atualizações da página inteira e permitindo carregamento progressivo de elementos da página. Contudo, pode gerar problemas de desempenho e de compatibilidade com navegadores.

- As aplicações Web passaram a poder ter partes da lógica de aplicação no lado do cliente, mudando o paradigma cliente-servidor estrito das primeiras aplicações Web. A lógica de aplicação do lado do cliente não é mais apenas usada para enriquecer a camada de apresentação com características de interfaces com o usuário dinâmicas; ao contrário, ela é parte da lógica de negócio global da aplicação.

O Processo de Projeto de Software

- Projetar a arquitetura de um software requer o levantamento de informações relativas à plataforma de implementação do sistema, as quais se somarão ao conhecimento acerca dos requisitos funcionais e não funcionais, para embasar as decisões relativas à arquitetura do sistema que está sendo projetado.

- De maneira geral, o processo de projetar envolve os seguintes passos:

- 1. Levantar informações acerca da plataforma de implementação do sistema, incluindo linguagem de programação a ser adotada, mecanismo de persistência e necessidades de distribuição geográfica.

O Processo de Projeto de Software

- 2. Com base nos requisitos, iniciar a decomposição do sistema em subsistemas, considerando preferencialmente uma decomposição pelo domínio do problema.

- Se na fase de análise já tiver sido estabelecida uma decomposição inicial em subsistemas, esta deverá ser utilizada.

- Neste momento, deve-se escolher um estilo arquitetônico (ou uma combinação adequada de estilos arquitetônicos) para organizar a estrutura geral do sistema. Um bom ponto de partida para essa escolha pode ser a classe de sistemas à qual pertence o sistema que está sendo projetado.

O Processo de Projeto de Software

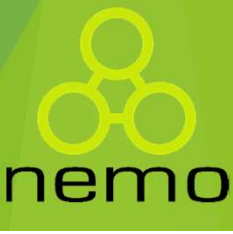
- 3. Priorizar os atributos de qualidade e definir quais deles guiarão o projeto da arquitetura, ditos condutores da arquitetura.

- 4. Para cada atributo de qualidade, identificar táticas a serem aplicadas. Uma vez que essas táticas podem levar a conflitos, procurar balanceá-las, respeitando as prioridades estabelecidas no passo anterior.

O Processo de Projeto de Software

- 5. Estabelecer uma arquitetura base, identificando tipos de módulos e tipos de relacionamentos entre eles, os quais são dados, essencialmente, pela combinação dos estilos arquitetônicos escolhidos.

- 6. Alocar requisitos funcionais (casos de uso) e não funcionais aos componentes da arquitetura.



O Processo de Projeto de Software

- 7. Avaliar a arquitetura, procurando identificar se ela acomoda tanto os requisitos funcionais, quanto os atributos de qualidade, dando atenção especial aos atributos considerados "condutores da arquitetura", ou seja, aqueles de maior prioridade.

- 8. Uma vez definida a arquitetura em seu nível mais alto, passar ao projeto de seus elementos. Padrões arquitetônicos são instrumentos muito valiosos para o projeto dos componentes da arquitetura.

That’s all Folks!