Estilos Arquiteturais

Silvia Regina Vergilio

Estilos e padrões arquiteturais

- Estilos arquiteturais
  - Definem meios de selecionar e apresentar blocos de construção de arquitetura
- Padrões arquiteturais
  - Projetos de alto nível, testados e validados, de blocos de construção de arquitetura

Shaw, M., Garlan, D. Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1966

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad e M. Stahl. Pattern-Oriented Software Architecture - A System of Patterns, NY: John Wiley and Sons, Inc. 1996

Categorias de Estilos de Arquiteturas

- **Estrutura** ("**From mud to structure**")- oferecem decomposição controlada das tarefas em sub-tarefas. Consideram requisitos estáveis e bem definidos.
- **Sistemas distribuídos** – aplicações distribuídas
- **Sistemas interativos** – interação HM.
- **Sistemas adaptáveis** – oferecem suporte para extensão e adaptação de aplicações devido a tecnologias e mudança de requisitos.

F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad e M. Stahl. *Pattern-Oriented Software Architecture - A System of Patterns,* NY: John Wiley and Sons, Inc. 1996

D. Garlan and Mary Shaw. An introduction to software architecture. Technical Report- CMU-CS-94166,Carnegie Mellon University, January 1994.

Estilos Arquiteturais: Taxonomia

Sistemas de Fluxo de Dados	*Sistemas de Chamada e Retorno*	*Componentes Independen-tes*	*Máquinas Virtuais*	*Sistemas Centrados em Dados*
Seqüenciais Batch	**Programa Principal e Sub-rotinas**	**Processos Comunican-tes**	**Interpreta-dor**	**Bancos de Dados**
Pipes & Filters	**Sistemas OO**	**Invocação Implícita (ou Sistemas Baseados em Eventos)**	**Sistemas Baseados em Regras**	**Sistemas de Hipertexto**
	Camadas			**Blackboards (Quadro-Negro)**

Extraído de (SHAW e GARLAN, 1996)

Estilos arquiteturais

Fluxo de dados (Data Flow)

1. Sequenciais Batch
2. Pipes & Filters (Dutos e Filtros)

Fluxo de Dados

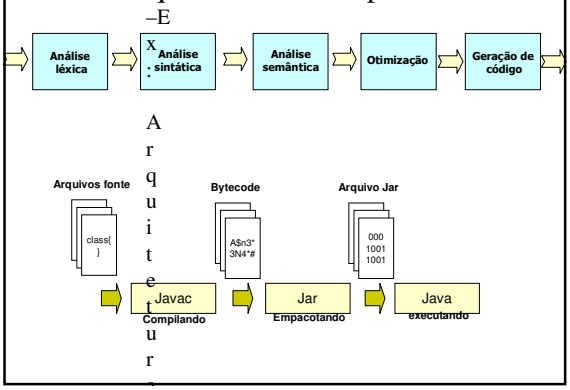
- Originário de sistemas operacionais UNIX e do projeto de compiladores

  Ex. Unix Pipes: condutores da saída de um programa para a entrada de um outro programa. **> Who | Sort**
- Transformações funcionais processam entradas para produzir saídas.
  - Componentes são chamados de filtros: recebem (tratam e refinam) entradas especificadas e transformam essas entradas em saídas
  - Conectores são dutos (*pipes)* - trasmitem saídas de um filtro para serem entradas de outro

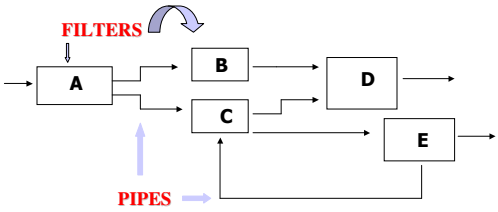
1. Seqüências (*Batch Sequential*)

- Especialização de pipes &filtros
- Programas independentes executados em seqüência (pipelines: seqüências linear de filtros)
  - Um após o outro
  - Dado transmitido por completo entre um programa e outro

Ex: Arquitetura de Compiladores

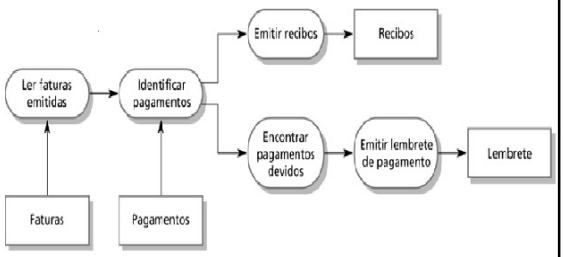


2. Pipes & Filters (Dutos e Filtros)



- Tipo mais geral – não precisa ser sequencial
- Pipes tipados – os tipos passados entre dois filtros tem um tipo bem definido.

Ex: Sistema de processamento de faturas



Características: Dutos e Filtros

- Vantagens
  - Útil para aplicações de processamento de informação que interagem pouco com usuários
  - Rápida prototipação
  - Apóia reúso de transformações (filtros)
  - É fácil adicionar, recombinar, ou trocar, novas transformações (flexibilidade)
  - É relativamente simples implementar como sistema concorrente (vários filtros em paralelo) ou seqüencial
  - Eficiência em processamento

Características: Dutos e Filtros

- Desvantagens
  - Requer um formato comum para a transferência de dados ao longo do *pipeline*
  - Não é apropriado para aplicações interativas
  - Não existe compartilhamento de dados
  - Ausência de gerenciamento de erros.

Implementação

- Divida as tarefas do sistema em uma sequência de estágios de processamento
  - Cada estágio deve depender somente da saída do seu predecessor
  - Todos os estágios são conectados por um fluxo de dados
- Defina o formato de dados a ser passado ao longo de cada pipe
- Decida como implementar cada conexão com pipe
  - Se estes serão ativos ou passivos

Estilos arquiteturais Chamada (ou invocações) e Retorno

1. Programa Principal e Sub-Rotina
2. Invocação Remota de Procedimento (RPC)
3. Sistema Orientados a Objetos
4. Camadas (Layered)

Chamada e Retorno

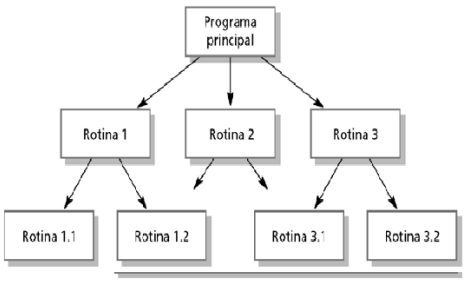
O controle de execução de um componente é realizado por outro componente através de uma invocação, que geralmente produz um retorno.

- Componentes: módulos, sub-rotinas, funções, objetos, e ou componentes complexos (grupos de componentes)
- Conectores: chamada de procedimentos, envios de mensagem, protocolos de comunicação, etc.

1. Programa principal e Sub-Rotinas (*Main Program/Subroutines*)

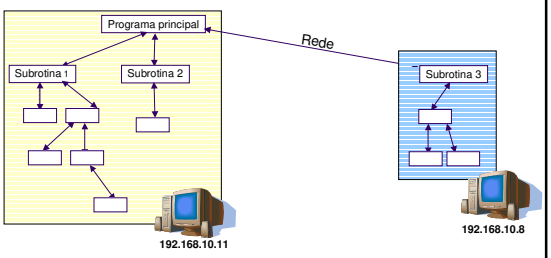
- Controle se inicia no topo de uma hierarquia de subrotinas e move-se para baixo na hierarquia.
  - Componentes – subrotinas
  - Conectores – chamadas de procedimento
- Vantagens: desenvolvimento pode ser independente; Gerenciamento de controle mais fácil de visualizar, na hierarquia de módulos.
- Desvantagens: o reúso, bem como alterações podem ser difíceis.

Exemplo:

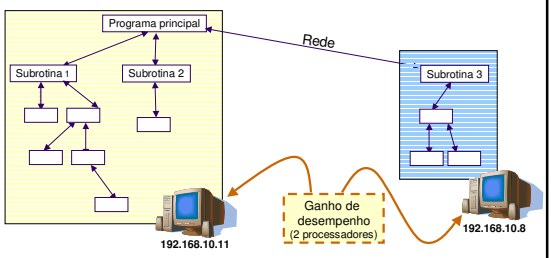


2. Invocação remota de procedimento (*RPC – remote procedure call*)

- especialização do programa principal e sub-rotinas



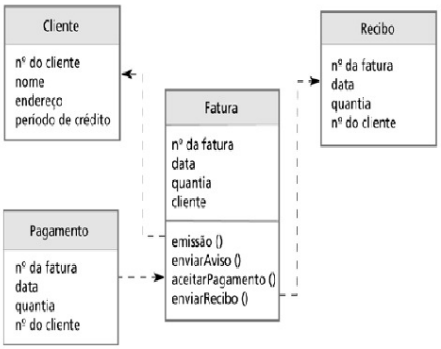
Vantagens



3. Sistema Orientados a Objetos

- Sistema como um conjunto de objetos fracamente acoplados e com interfaces bem definidas
  - Cada objeto oferece um conjunto de serviços
- Sistemas Orientados a Objetos (OO) podem ser vistos como uma rede ou grafo de objetos comunicantes.
  - Componentes: Objetos.
  - Conectores: envio de mensagem
  - Regras: objetos encapsulam seus dados; um objeto é responsável pela manutenção da integridade de sua representação.

Ex: Sistema de processamento de faturas



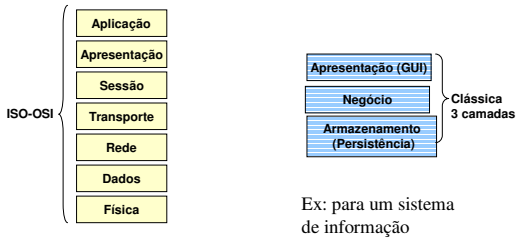
Características: Sistemas Orientados aObjetos

- Vantagens
  - Objetos são fracamente acoplados devido ao uso de interfaces
  - Linguagens de implementação orientada a objeto são amplamente usadas.
- Desvantagens
  - Mudanças de interface têm alto impacto
  - Não envolve restrições topológicas, o que pode dificultar a manutenção
    - Dependências entre objetos não são limitadas

4. Camadas (Layered)

- Sistema organizado hierarquicamente.
- Cada camada oferece o serviço para a camada superior (externa) e utiliza um serviço da camada inferior (interna).
  - Componentes: são camadas, grupo de tarefas em um mesmo nível de abstração
  - Conectores: protocolos que indicam como as camadas irão interagir e limitam as comunicações a camadas adjacentes; permitem comunicação em máquinas diferentes
- Cada camada está associada a um conjunto de entidades (entidade complexa), constituindo-se de diferentes componentes (por ex.: conjunto de objetos, funções, etc).

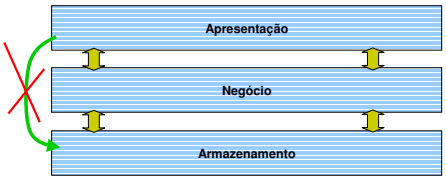
Exemplos



Ex: para um sistema de informação

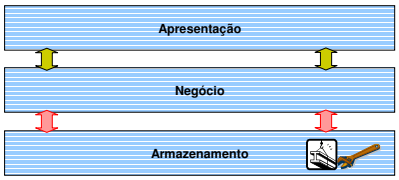
Características

- Camadas se comunicam apenas com outras adjacentes

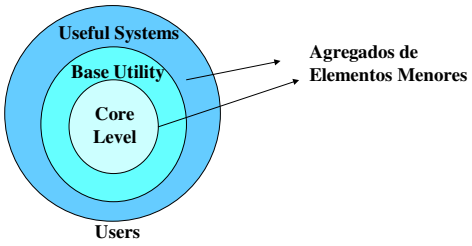


Características

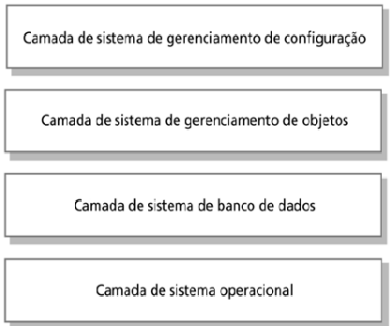
- Alterações locais não são propagadas



Outra maneira de representar



Ex: Sistema de gerenciamento de versões



Implementação

- Defina o critério de abstração para agrupar tarefas em camadas
  - Exemplo: a distância do hardware pode formar os níveis mais baixos e a complexidade conceitual os níveis mais altos
- Determine o número de níveis de abstração de acordo com seu critério de abstração
- Nomeie as camadas e determine as tarefas de cada uma delas
  - A tarefa da camada mais alta é a percebida pelo cliente
  - As tarefas das demais camadas visam ajudar a realização da tarefa da camada mais alta

Implementação

- Especifique os serviços
- Especifique uma interface para cada camada
- Refine cada camada:
  - Estruturação de cada camada individualmente
  - Quando uma camada é complexa ela deve ser separada em componentes individuais e cada componente pode seguir um padrão ou estilo diferente

Características: Estilo em Camadas

Vantagens:

- Facilidade de compreensão: utiliza níveis crescentes de abstração, particionando problemas complexos em sequência de passos incrementais.
- Suporte a padronização
- Desenvolvimento independente
- Facilidade de manutenção, reúso das camadas e suporte à evolução dos sistemas, oferencendo flexibilidade e boa manutenibilidade
- As dependências tendem a ser locais (dentro da camada)- restrição de comunicação entre camadas adjacentes, mudanças afetam no máximo duas camadas
- Se interface bem definida, permite uso de diferentes implementações da mesma camada

Características: Estilo em Camadas

Desvantagens

- Às vezes é difícil estruturar um sistema através de camadas. Conseqüências:
  - É comum que a estruturação seja violada
  - Camadas relaxadas são necessárias – todas as camadas podem se comunicar entre si.
  - Mudanças em serviços de uma cada inferior, podem requerer propagação de mudanças até as superiores
  - Pode haver necessidade de duplicação de funcionalidade
- *Overhead* (problemas) de implementação, comunicação, e desempenho
- Complexidade na Implementação e Testes do Sistema

Estilos arquiteturais Componentes Independentes

1. Processos comunicantes
   Cliente-servidor, Peer to peer

2. Baseado em eventos
   Invocação implícita

Componentes Independentes

- Processos são independentes
- Envio de dados entre processos, normalmente sem controlar a execução de cada um deles.
- O controle geralmente é feito com envio de mensagens ou baseado em eventos.
- Alto grau de modificabilidade através do desacoplamento de várias porções da computação

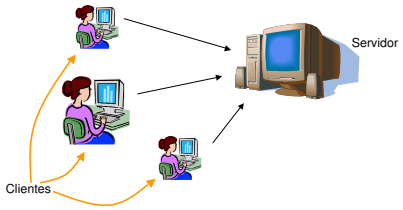
1. Processos comunicantes

- Baseado na comunicação via troca de mensagens entre processos
  - Em geral, via rede
- Cliente - Servidor
- Ponto a ponto (*Peer to Peer* – P2P)

Estilo Cliente-Servidor

- Mostra como dados e processamento são distribuídos por uma variedade de componentes
  - Servidores independentes que fornecem serviços tais como impressão, transferência de arquivos, gerenciamento de dados, etc.
  - Clientes utilizam esses serviços
- Clientes e servidores normalmente se comunicam através de uma rede
  - Diversas tecnologias de comunicação são possíveis

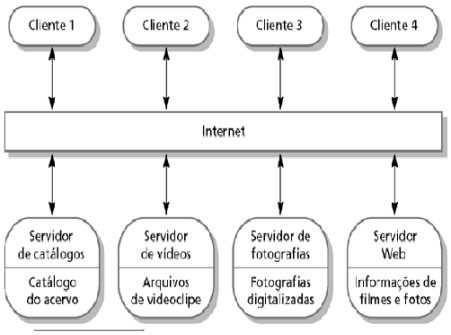
Ex: Aplicação: Internet



Ex: Sistema– Cliente-Servidor

Cliente	Middleware		Servidor
Web browser	ODBC	TCP/IP	Web
GUI	Mail	Diretorios	DBMS
DSM/OS	httpp	RPC	Groupware

Ex: Biblioteca de filmes e fotografias



Características: Cliente-Servidor

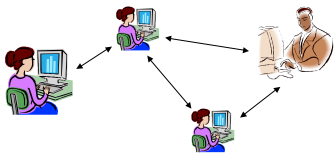
- Vantagens
  - Separação de interesses
  - Inerentemente distribuído: pode haver balanceamento de carga, tolerância a falhas
  - É fácil adicionar novos servidores ou atualizar servidores existentes.
  - Utilização dos recursos do servidor
  - Escalabilidade: aumentando a capacidade computacional do servidor

Características: Cliente-Servidor

- Desvantagens
  - Gerenciamento redundante em cada servidor;
  - Nenhum registro central de nomes e serviços – pode ser difícil descobrir quais servidores e serviços estão disponíveis
  - Requisições e respostas casadas
  - Introduz complexidade
  - Custos de comunicação
  - Falhas no servidor

Ponto a Ponto (P2P)

- Não há distinção entre nós
- Cada nó mantém seus próprios dados e endereços conhecidos
- Cada nó é "cliente e servidor ao mesmo tempo"
- Vantagem: reduz problemas de falhas
- Desvantagem: aumenta o tempo de consulta

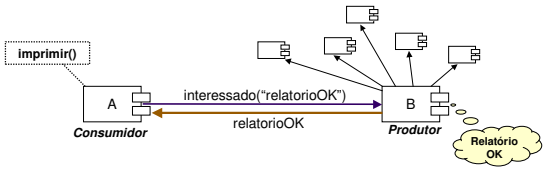


2. Baseado em eventos

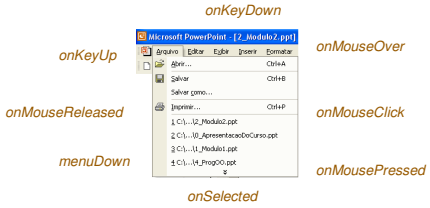
- Desacoplamento entre consumidores e produtores de eventos
- Escalabilidade: adição de novos observadores para eventos que já são produzidos
- Invocação implícita: O produtor de eventos não controla quem será notificado ou quando ele será notificado

Exemplo: relatório de impressão

- Produtores e consumidores são independentes
- Execução via procedimentos disparados via mudança de estados
- Escalabilidade no número de interessados
- Consumidores se registram nos Produtores
- Produtores notificam consumidores registrados



Ex: Interface Gráfica



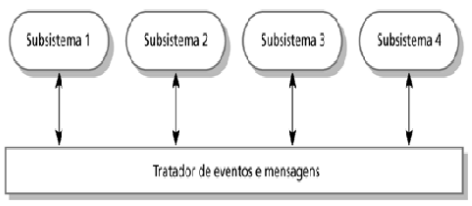
Sistemas orientados a eventos

- Dirigidos por eventos gerados externamente
  - O *timing* dos eventos está fora do controle dos componentes que os processam
- Estilo *Publisher/Subscriber*
  - Eventos são transmitidos a todos os componentes.
  - Qualquer componente interessado pode respondê-los
- Estilo Orientado a Interrupções
  - Usado em sistemas de tempo real
  - Interrupções são detectadas por tratadores e passadas por outro componente para processamento.

Modelo Publisher/Subscriber

- É efetivo na integração de componentes em computadores diferentes em uma rede
  - Desacoplamento espacial e temporal
  - Componentes não sabem se um evento será tratado e nem quando será.
- Alguns componentes (*publishers*) publicam eventos
- Componentes (*subscribers*) registram interesse em eventos específicos e podem tratá-los
- A política de controle não é embutida no tratador de eventos e mensagens

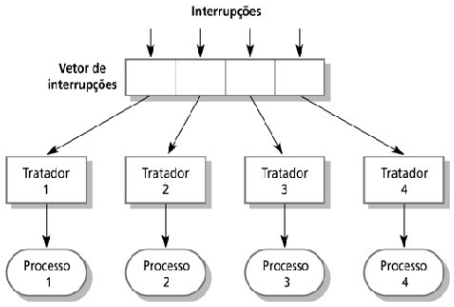
Publisher/Subscriber



Estilo Orientado a Interrupções

- Usado em sistemas de tempo real onde a resposta rápida para um evento é essencial
- Existem tipos de interrupções conhecidos
  - Um tratador definido para cada tipo
- Cada tipo é associado a uma localização da memória
  - Uma chave de hardware causa a transferência de controle para um tratador.
- Permite respostas rápidas, mas é complexo para programar e difícil de validar.

Controle dirigido a interrupções



Estilos arquiteturais
Máquina Virtual
(Virtual Machine)

1. Interpretador

2. Baseado em Regras

Máquina Virtual

Uma máquina virtual é produzida no software, geralmente para uma determinada linguagem.

- Simular funcionalidade não nativa para obter portabilidade

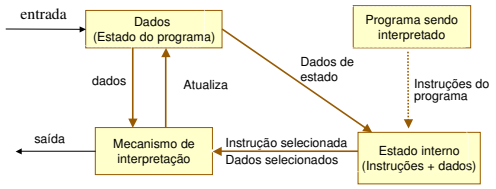
Ex: Simuladores de Software (linguagens - Java VM)

1. Interpretador (Interpreter)

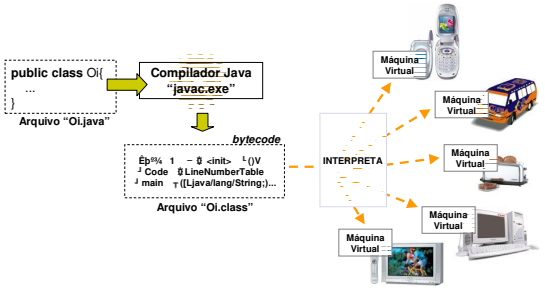
- Esta arquitetura inclui geralmente 4 componentes:
  - mecanismo de interpretação
  - uma memória que contém o pseudo-código a ser interpretado
  - representação do estado do mecanismo de interpretação
  - representação do estado atual do programa sendo simulado

A máquina tenta preencher a lacuna que existe entre o que o programa precisa e o que o hardware disponibiliza

Componentes de um Interpretador



Ex: Java VM



Características - Interpretador

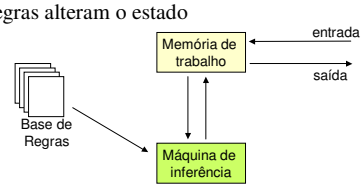
- Desvantagens: Desempenho
- Vantagens: Portabilidade

Algumas pesquisas apontam que algumas das linguagens interpretadas já conseguem ser mais rápidas que C

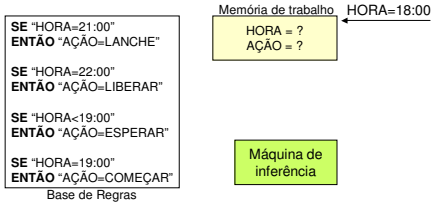
- Java, por exemplo

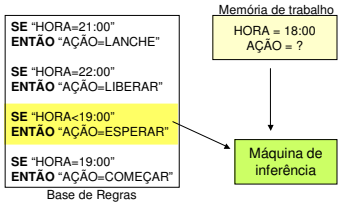
2. Baseado em Regras (Rule-Based)

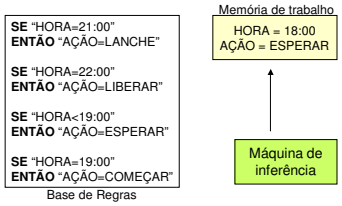
- Conjunto de regras sobre um estado
- Definição do estado atual com base em dados de entrada
- Regras alteram o estado

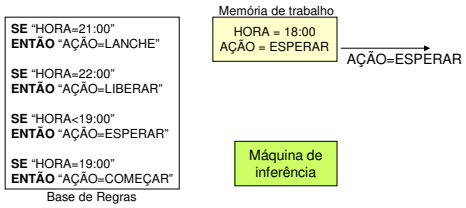


Ex: Prolog, Sistemas Especialistas









Estilos arquiteturais Centrado em Dados

1. Repositório
2. Blackboard

Centrado em Dados *(Data-centered)*

- A meta é a integração de dados
- Sistemas cujas partes precisam trocar dados com frequência
- Descreve o acesso e atualização de repositórios de dados amplamente acessíveis
- Existe um grande depósito de dados centralizado, manipulado por computações independentes

1. Repositório (*Repository*)

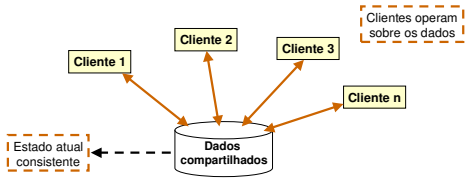
– Dados compartilhados podem ser mantidos em um banco de dados central e acessados por todos os subsistemas

– Cada subsistema mantém seu próprio banco de dados e passa dados para outros subsistemas

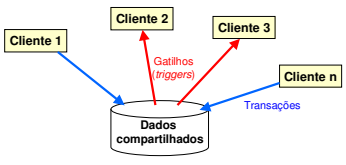
- Podem usar uma abstração de repositório centralizado
- Implementação distribuída

Repositório (*Repository*)

- Integridade, escalabilidade (novos clientes, novos dados)



Ex: Banco de dados tradicional



Ex: Arquitetura de uma Ferramenta Case

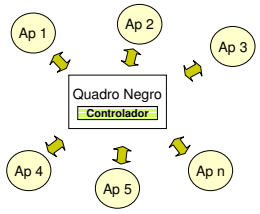
Características: Repositório

- Vantagens
  - É uma maneira eficiente de compartilhar grandes quantidades de dados
  - Dados aderem a uma representação comum
  - Simplifica a projeto de aplicações fortemente baseadas em dados
    - Tanto para troca de informações quanto para armazenamento
- Desvantagens
  - Os subsistemas devem estar de acordo com um modelo de dados padronizado
  - A evolução de dados é difícil e dispendiosa
  - Dificuldade para distribuir de forma eficiente

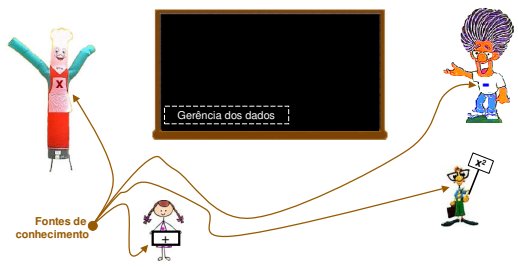
2. Quadro negro (*Blackboard*)

- O sistema é dividido em
  - blackboard: aramazena dados – o vocabulário
  - base de conhecimento: subsistemas independentes, cada qual resolvendo aspectos específicos do problema
  - componente de controle: monitora mudanças no blackboard e decide as ações

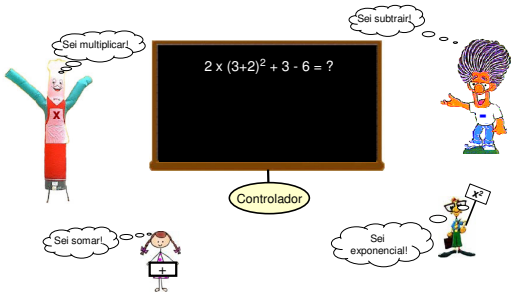
Quadro negro (*Blackboard*)



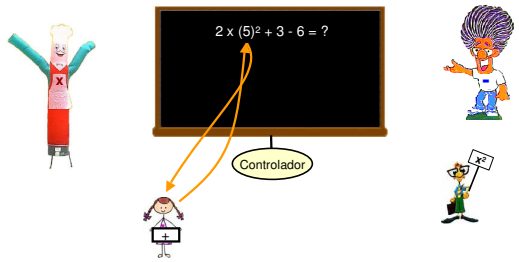
Quadro negro (*Blackboard*)



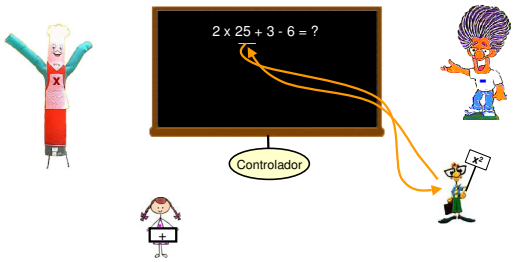
Quadro negro (*Blackboard*)



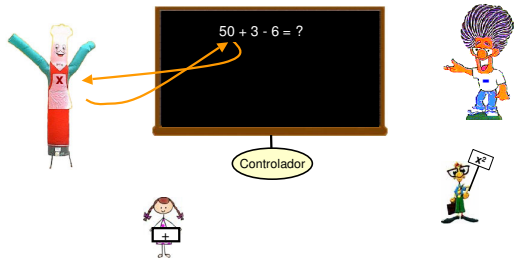
Quadro negro (*Blackboard*)



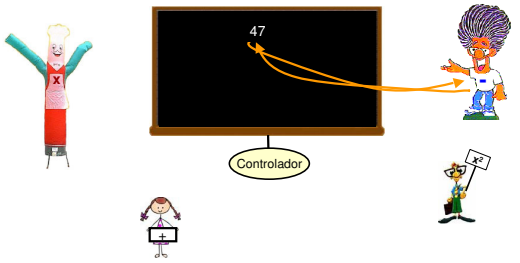
Quadro negro (*Blackboard*)



Quadro negro (*Blackboard*)



Quadro negro (*Blackboard*)



Características: *Blackboard*

Usado em:

- Sistemas que não possuem estratégias de soluções determinísticas conhecidas e são baseados em soluções aproximadas ou parciais.
- Problemas que podem ser decompostos em sub-problemas e abrangem muitos domínios de conhecimento
  - Sistemas complexos - Resolução Distribuída de Problemas - RDP
  - Paradigma de agentes

Características: *Blackboard*

- Vantagens:
  - Ajuda a resolver problemas de experimentação
  - Suporte a mudanças e manutenção
  - Escalabilidade – aplicações independentes
  - Reúso de conhecimentos
  - Suporte: tolerância a falhas e robustez

Características: *Blackboard*

- Desvantagens
  - Dificuldades para testar por não usar algoritmos determinísticos
  - Nenhuma boa solução é garantida
  - Dificuldade em estabelecer uma boa estratégia de controle.
  - Baixa eficiência e alto esforço de desenvolvimento
  - Não suporta paralelismo

Seleção de Estilos

O que considerar?
Passos a seguir

Fluxo de Dados

- As interfaces entre os componentes são simples
- O sistema produz resultados simples e bem identificáveis que derivam diretamente da transformação seqüencial de uma entrada facilmente identificável
- A relação entre entrada e saídas é temporalmente independente

Fluxo de Dados

- Sequencial: transformações são seqüenciais
  - Existe uma única saída, resultante de uma única entrada de dados
- Fitros e Pipes:
  - A computação envolve transformações sobre uma cadeia de dados contínua
  - As transformações são incrementais. Uma transformação pode executar antes do término do passo anterior

Chamada/Retorno

- A ordem da computação é fixa
- Componentes não podem fazer progresso enquanto aguardando o resultado das chamadas

Chamada/Retorno

- Orientação a objetos:
  - Ocultamento da Informação: produz módulos similares, que no decorrer do desenvolvimento e teste se beneficiam do uso de herança

Chamada/Retorno

- Camadas:
  - As tarefas do sistema podem ser particionadas entre: específicas da aplicação, e genéricas a muitas aplicações, mas específicas à plataforma subjacente
  - Portabilidade é importante
  - Pode-se usar uma infra-estrutura de computação pré-existente

Componentes Independentes

- O sistema executa em uma plataforma multi-processada (ou que pode vir a ser no futuro)
- O sistema pode ser estruturado como um conjunto de componentes fracamente acoplados, de modo que um componente pode fazer progressos de forma independente dos outros.
- Ajuste de desempenho é importante, seja através da re-alocação de tarefas a processos,seja através da re-alocação de processos a processadores

Componentes Independentes

- Processos Comunicantes:
  - Objetos Distribuídos: OO + Componentes Independentes
  - Redes de filtros: Data-Flow + Componentes Independentes
  - Cliente-Servidor: As tarefas podem ser divididas entre geradores de pedidos (ou consumidores de dados) executores de pedidos (ou produtores de dados)

Componentes Independentes

- Baseada em Evento
  - Quando é necessário desacoplar consumidores e produtores de eventos
  - Quando é necessário escalabilidade, e permitir a adição de novos processos ao sistema, os quais serão integrados a eventos já sinalizados no sistema

Centrada em Dados

- As questões importantes são o armazenamento, representação, gerenciamento e recuperação de uma grande quantidade de dados persistentes

Máquina Virtual

- Não existe uma máquina que execute um modelo computacional que foi projetado

Seleção de estilos –Passos a seguir

1. Identificar os principais elementos da arquitetura
2. Identificar o estilo arquitetural dominante
3. Considerar responsabilidades adicionais associadas com a escolha do estilo
4. Modificar o estilo para atingir objetivos adicionais

1. Identificar os principais elementos da arquitetura

- Cada elemento arquitetural tem um estilo arquitetural dominante que reflete as qualidades importantes que devem ser alcançadas no contexto daquele elemento
- A escolha do estilo arquitetural dominante é baseada nos principais elementos arquiteturais
- Os atributos de qualidade sobre cada elemento arquitetural podem acarretar a utilização ou não de um estilo

2. Identificar o estilo arquitetural dominante

- O estilo dominante pode ser modificado para alcançar objetivos particulares
- Se nenhum estilo conhecido parece ser apropriado, o arquiteto deve projetar e documentar um novo estilo
- As decisões sobre escolhas baseadas em atributos de qualidade dentro de um estilo devem ser documentadas

3. Considerar responsabilidades adicionais associadas com a escolha do estilo

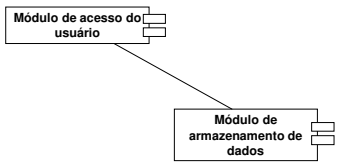
- A escolha de um estilo arquitetural introduzirá responsabilidades adicionais
- Por exemplo:
  - Se o estilo é "Quadro negro", então devem-se gerenciar os mecanismos para o controle do quadro negro
  - Se o estilo é "cliente-servidor", devem-se gerenciar os protocolos de interação
- Responsabilidades adicionais devem ser atribuídas a elementos arquiteturais existentes ou a novos elementos criados para este fim.

4. Modificar o estilo para atingir objetivos adicionais

- Pode-se alterar o estilo arquitetural caso este necessite ser adaptado devido a atributos de qualidade ou até mesmo funcionalidade

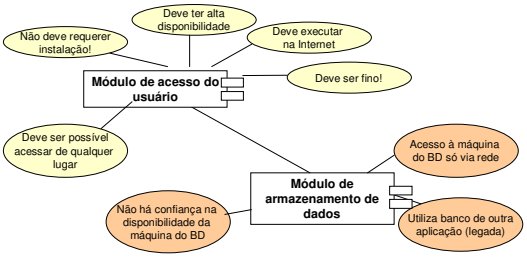
Exemplo: Sistema de matrícula

- 1. Identificar os principais elementos da arquitetura



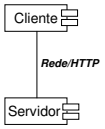
Exemplo: Sistema de matrícula

- 2. Identificar o estilo arquitetural dominante



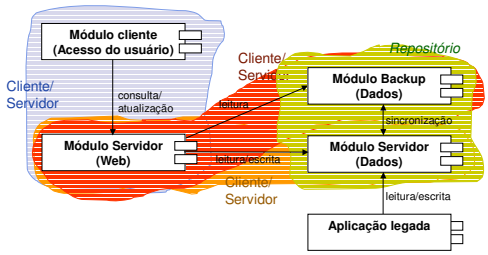
Exemplo: Sistema de matrícula

- 3. Considerar responsabilidades adicionais associadas com a escolha do estilo
  - Estilo dominante escolhido: Cliente-servidor
  - Estilo secundário: Repositório
  - Responsabilidades: considerar protocolo de comunicação



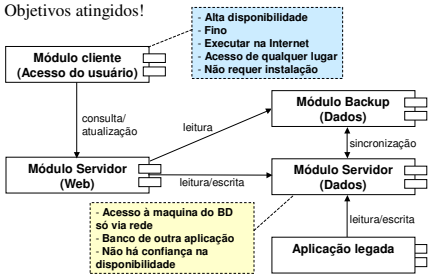
Exemplo: Sistema de matrícula

- 4. Modificar o estilo para atingir objetivos adicionais
  - Cliente servidor de 2 camadas e repositório com *backup*



Exemplo: Sistema de matrícula

- 4. Modificar o estilo para atingir objetivos adicionais
  - Objetivos atingidos!



Estilos arquiteturais
Estudo de Caso- *Garlan e Shaw (1994)*

Objetivo

– Ilustrar como princípios de arquitetura podem aumentar o entendimento de sistemas de software.

– Avaliar diferentes soluções e seus impactos para um mesmo problema

1. Palavra-chave em contexto: objetivo

- Analisar diferentes arquiteturas com respeito às seguintes considerações de projeto:
  - Mudança no algoritmo de processamento (ex: shifts nas linhas podem ser feitos em cada linha lida, em todas as linhas após terem sido lidas, ou por demanda);
  - Mudança na representação dos dados;
  - Mudança na funcionalidade (extensibilidade);
  - Performance (espaço e tempo);
  - Grau de reuso.

Problema exemplo: palavra-chave em contexto

Key Word In Context

Problem Description:

"The KWIC index system accepts an ordered set of lines, each line is an ordered set of words, and each word is an ordered set of characters.

Any line may be 'circularly shifted' by repeatedly removing the first word and appending it at the end of the line.

The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order."

On the Criteria for Decomposing Systems into Modules. David Parnas. *CACM*, 1972

Palavra-chave em contexto: entrada e saída

KWIC: Key Word In Context

- Inputs: *Sequence of lines*

```
Pipes and Filters
Architectures for Software Systems
```

- Outputs: *Sequence of lines, circularly shifted and alphabetized*

```
and Filters Pipes
Architectures for Software Systems
Filters Pipes and
for Software Systems Architectures
Pipes and Filters
Software Systems Architectures for
Systems Architectures for Software
```


Selecione um estilo para o sistema KWIC, e represente os elementos arquiteturais do sistema no estilo escolhido.