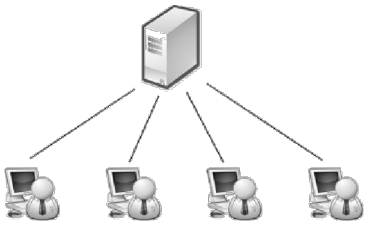
Exemplos de Arquitetura de Software

Marcos Monteiro, MBA, ITIL V3

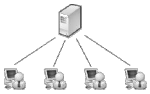
http://www.marcosmonteiro.com.br
contato@marcosmonteiro.com.br

Cliente - Servidor



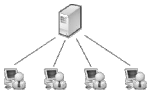
Separa clientes e servidores, sendo interligados entre si geralmente utilizando-se uma rede de computadores.

Cliente - Servidor



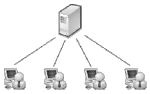
- Cada instância de um cliente pode enviar requisições de dado para algum dos servidores conectados e esperar pela resposta.
- Por sua vez, algum dos servidores disponíveis pode aceitar tais requisições, processá-las e retornar o resultado para o cliente.

Cliente - Servidor



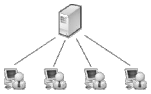
- Muitas vezes os clientes e servidores se comunicam através de uma rede de computador com hardwares separados, mas o cliente e servidor podem residir no mesmo sistema. A máquina servidor é um host que está executando um ou mais programas de servidor que partilham os seus recursos com os clientes.
- Um cliente não compartilha de seus recursos, mas solicita o conteúdo de um servidor ou função de serviço.
- Os clientes, portanto, iniciam sessões de comunicação com os servidores que esperam as solicitações de entrada.

Descrição



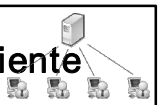
- Funções como a troca de e-mail, acesso à internet e acessar banco de dados, são construídos com base no modelo cliente-servidor.
  - Por exemplo, um navegador da web é um programa cliente em execução no computador de um usuário que pode acessar informações armazenadas em um servidor web na Internet.
  - Usuários de serviços bancários acessando do seu computador usam um cliente navegador da Web para enviar uma solicitação para um servidor web em um banco. Esse programa pode, por sua vez encaminhar o pedido para o seu próprio programa de banco de dados do cliente que envia uma solicitação para um servidor de banco de dados em outro computador do banco para recuperar as informações da conta. O saldo é devolvido ao cliente de banco de dados do banco, que por sua vez, serve-lhe de volta ao cliente navegador exibindo os resultados para o usuário.

Descrição



- Cada instância de software do cliente pode enviar requisições de dados a um ou mais servidores ligados.
- Por sua vez, os servidores podem aceitar esses pedidos, processá-los e retornar as informações solicitadas para o cliente.
- Embora este conceito possa ser aplicado para uma variedade de razões para diversos tipos de aplicações, a arquitetura permanece fundamentalmente a mesma.

Características do Cliente

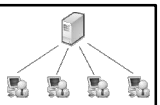


- Sempre inicia pedidos de servidores;
- Espera por respostas;
- Recebe respostas;
- Normalmente, se conecta a um pequeno número de servidores de uma só vez;
- Normalmente, interage diretamente com os usuários finais através de qualquer interface com o usuário , como interface gráfica do usuário.

Características do Servidor

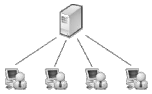
- Sempre esperar por um pedido de um dos clientes;
- Serve os clientes pedidos, em seguida, responde com os dados solicitados aos clientes;
- Um servidor pode se comunicar com outros servidores, a fim de atender uma solicitação do cliente.

☺ Vantagens



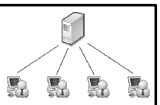
- Na maioria dos casos, a arquitetura cliente-servidor permite que os papéis e responsabilidades de um sistema de computação possa ser distribuído entre vários computadores independentes que são conhecidos por si só através de uma rede.
  - Isso cria uma vantagem adicional para essa arquitetura:
    - **Maior facilidade de manutenção**. Por exemplo, é possível substituir, reparar, atualizar ou mesmo realocar um servidor de seus clientes, enquanto continuam a ser a consciência e não afetado por essa mudança;

☺ Vantagens



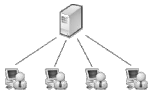
- Todos os dados são armazenados nos servidores, que geralmente possuem controles de segurança muito maior do que a maioria dos clientes.
- Servidores podem controlar melhor o acesso e recursos, para garantir que apenas os clientes com as permissões adequadas podem acessar e alterar dados;

☺ Vantagens



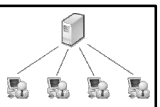
- Desde o armazenamento de dados é centralizada, as atualizações dos dados são muito mais fáceis de administrar, em comparação com o paradigma P2P, onde uma arquitetura P2P, atualizações de dados podem precisar ser distribuída e aplicada a cada ponto na rede, que é o time-consuming é passível de erro, como pode haver milhares ou mesmo milhões de pares;

☺ Vantagens



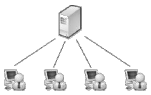
- Muitas tecnologias avançadas de cliente-servidor já estão disponíveis, que foram projetadas para garantir a segurança, facilidade de interface do usuário e facilidade de uso;

☺ Vantagens



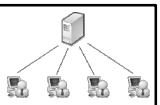
- Funciona com vários clientes diferentes de capacidades diferentes.

☹ Desvantagens



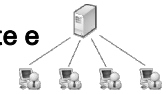
- Redes de tráfego de bloqueio é um dos problemas relacionados com o modelo cliente-servidor.
  - Como o número de solicitações simultâneas de cliente para um determinado servidor, o servidor pode ficar sobrecarregado;

☹ Desvantagens



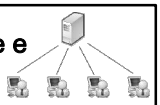
- O paradigma cliente-servidor não tem a robustez de uma rede P2P.
  - Sob cliente-servidor, se um servidor crítico falhar, os pedidos dos clientes não podem ser cumpridos.
  - Em redes P2P, os recursos são normalmente distribuídos entre vários nós. Mesmo se um ou mais nós partem e abandonam baixar um arquivo, por exemplo, os nós restantes ainda deve ter os dados necessários para completar o download.

Protocolos de transporte e aplicações de rede



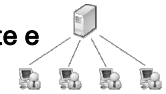
- Os protocolos do nível de transporte fornecem serviços que garantem uma transferência confiável de dados e aplicativos entre computadores (ou outros equipamentos) remotos. Os programas na camada de aplicação usam os protocolos de transporte para contactar outras aplicações. Para isso, a aplicação interage com o software do protocolo antes de ser feito o contacto. A aplicação que aguarda a conexão informa ao software do protocolo local que está pronta a aceitar mensagem. A aplicação que estabelece a conexão usa os protocolos de transporte e rede para contactar o sistema que aguarda. As mensagens entre as duas aplicações são trocadas através da conexão resultante.

Protocolos de transporte e aplicações de rede



- Existem duas formas para que se estabeleça uma ligação cliente-servidor: enquanto uma delas é orientada à conexão, a outra não é.
  - O TCP, por exemplo, é um protocolo de transporte orientado à conexão em que o cliente estabelece uma conexão com o servidor e ambos trocam múltiplas mensagens de tamanhos variados, sendo a aplicação do cliente quem termina a sessão.
  - Já o protocolo UDP não é orientado à conexão, nele o cliente constrói uma mensagem e a envia num pacote UDP para o servidor, que responde sem estabelecer uma conexão permanente com o cliente.
    - Ver arquivo services:
      - Windows : \WINDOWS\system32\drivers\etc\services
      - Linux : /etc/services

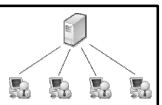
Protocolos de transporte e aplicações de rede



- Comandos do Prompt de comando que auxiliaram em teste de conexão:
  - ping
    - Testar se o há rede com o servidor
      - ping <IP ou host>
        - » Ex: ping 192.168.0.1
        - » Espera-se por resposta, envia-se 4 pacotes e se recebe 4 pacotes.
  - telnet
    - Um otima forma de testar camada de transporte
      - telnet <IP> <PORTA>
        - » Ex: telnet 192.168.0.1 8080
        - » Testando um servidor tomcat que esteja operando na porta 8080, qualquer mensagem ou ausência desta é indicador de conexão, o problema se dará caso apareça “falha de conexão”.

  OBS: Um firewall do servidor por bloquear resposta ao ping e ao telnet.

Referências



- MENDES, Antonio. Arquitetura de Software: desenvolvimento orientado para arquitetura. Editora Campus. Rio de Janeiro - RJ, 2002.

Arquiteturas em n Camadas

Problema: Pressões do negócio

- Steve Jobs fala:
  - "Internet time" é um novo conceito (final dos anos 1990) e significa que as empresas devem implementar novos sistemas *muito* rapidamente
- Muitos sistemas de empresas devem migrar para Internet/Intranet/Extranet
  - Novos tipos de sistemas devem ser desenvolvidos para usar a tecnologia como *business advantage*, dando um diferencial nos negócios
- Muitos sistemas devem mudar devido a mudanças nos negócios
  - Fusões e aquisições

Resultado: tem *muito* mais software a fazer, *muito mais rapidamente*

- Mas há pressões para manter custos de IT (*Information Technology*) baixos

Problema: Pressões Tecnológicas

- O desenvolvimento de software ficou muito mais complexo nos últimos anos
  - Pelos motivos acima
  - Porque usuários querem funcionalidade mais sofisticada
- Problemas de complexidade
  - O desenvolvimento de muitos grandes sistemas tem fracassado recentemente
  - A figura mais citada: 80% dos projetos são fracassos!
- Resumindo: fazer sistemas de produção customizados do zero in-house:
  - É muito caro
  - Demora muito tempo
  - Não produz boa qualidade

O que grandes empresas, com grandes problemas de desenvolvimento de sistemas, querem, tecnologicamente?

- Melhor flexibilidade
  - Possibilitando satisfazer novos requisitos do negócio (=funcionalidade) rapidamente
- Melhor adaptabilidade
  - Possibilitando customizar uma aplicação para vários usuários, usando várias alternativas de delivery dos serviços da aplicação com impacto mínimo ao núcleo da aplicação
- Melhor manutenabilidade
  - Possibilitando atualizar uma aplicação, mas minimizando o impacto da maioria das mudanças

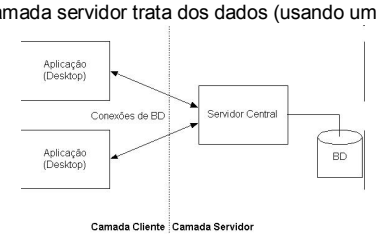
- Melhor reusabilidade
  - Possibilitando rapidamente montar aplicações únicas e dinâmicas
- Melhor aproveitamento do legado
  - Possibilitando reusar a funcionalidade de sistemas legados em novas aplicações
- Melhor interoperabilidade
  - Possibilitando integrar 2 aplicações executando em plataformas diferentes
- Melhor escalabilidade
  - Possibilitando distribuir e configurar a execução da aplicação para satisfazer vários volumes de transação

- Menor tempo de desenvolvimento
  - Equivalente a querer Produtividade do Programador
- Melhor robustez
  - Possibilitando ter soluções com menos defeitos
  - Possibilitando ter confiabilidade e disponibilidade
- Menor risco
  - Possibilitando tudo que falamos acima e ainda não se arriscar a ter projetos fracassados

Resumindo: tudo que ISO 9126 caracteriza como "qualidade de software"

Arquitetura em 2 camadas

- Os primeiros sistemas cliente-servidor eram de duas camadas
  - Camada cliente trata da lógica de negócio e da UI
  - Camada servidor trata dos dados (usando um SGBD)

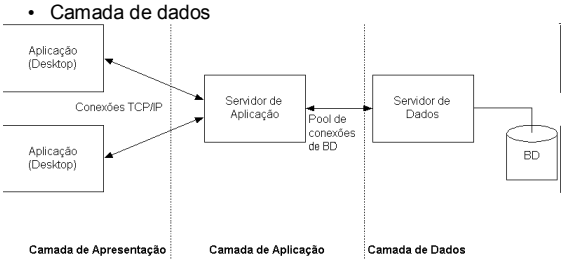


PROBLEMAS!!!

- A arquitetura cliente/servidor em 2 camadas sofria de vários problemas:
  - Falta de escalabilidade (conexões a bancos de dados)
  - Enormes problemas de manutenção (mudanças na lógica de aplicação forçava instalações)
  - Dificuldade de acessar fontes heterogêneas

Arquitetura em 3 camadas

- Camada de apresentação (UI)
- Camada de aplicação (business logic)
- Camada de dados



Arquitetura em 3 camadas

- Problemas de manutenção foram reduzidos, pois mudanças às camadas de aplicação e de dados não necessitam de novas instalações no desktop
- Observe que as camadas são lógicas
  - Fisicamente, várias camadas podem executar na mesma máquina
  - Quase sempre, há separação física de máquinas

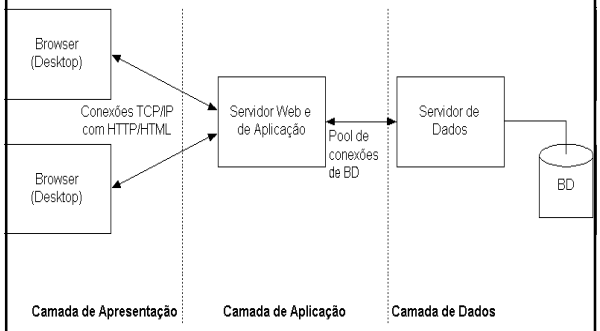
A arquitetura em 3 camadas original sofre de problemas:

- A instalação inicial dos programas no desktop é cara
- O problema de manutenção ainda persiste quando há mudanças à camada de apresentação
- Não se pode instalar software facilmente num desktop que não está sob seu controle administrativo
  - Em máquinas de parceiros
  - Em máquinas de fornecedores
  - Em máquinas de grandes clientes
  - Em máquinas na Internet

Arquitetura em 3/4 camadas Web-Based

- Então, usamos o Browser como Cliente Universal
- Conceito de Intranet
- A camada de aplicação se quebra em duas: Web e Aplicação
- Evitamos instalar qualquer software no desktop e portanto, problemas de manutenção
- Evitar instalação em computadores de clientes, parceiros, fornecedores, etc.
- Às vezes, continua-se a chamar isso de 3 camadas porque as camadas Web e Aplicação freqüentemente rodam na mesma máquina (para pequenos volumes)

Arquitetura em 3/4 camadas Web-Based

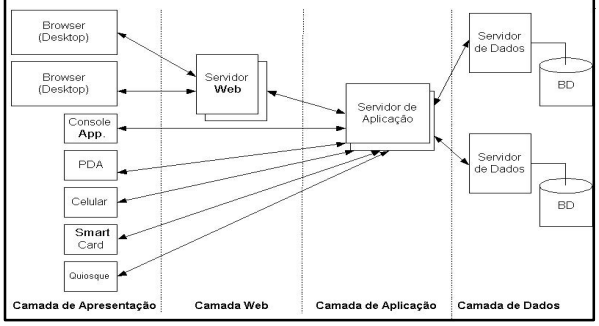


os problemas remanescentes:

1. Não há suporte a Thin Clients (PDA, celulares, smart cards, quiosques, ...) pois preciso usar um browser (pesado) no cliente
- Dificuldade de criar software reutilizável: cadê a componentização?
- Fazer aplicações distribuídas multicamadas é difícil. Tem que:
  - Implementar persistência (*impedance mismatch* entre o mundo OO e o mundo dos BDs relacionais)
  - Implementar tolerância a falhas com fail-over
  - Implementar gerência de transações distribuídas
  - Implementar balanceamento de carga
  - Implementar *resource pooling*

- O truque é introduzir *middleware* num servidor de aplicação que ofereça esses serviços automaticamente
- Além do mais, as soluções oferecidas (J2EE, .Net) são baseadas em componentes

•As camadas podem ter vários nomes:
- •Apresentação, interface, cliente
- •Web
- •Aplicação, Business
- •Dados, Enterprise Information System (EIS)



A Arquitetura J2EE

Componentes de Aplicação

- Aplicações J2EE são compostas de componentes
- Para nós, um componente é uma unidade autocontida de software que pode ser composta numa aplicação em tempo de design (sem compilação)
- Componentes J2EE são escritos em Java

Componentes J2EE na Camada de Apresentação

- "Application client" (cliente não-Web)
  - Tipicamente usa Swing como User Interface (UI)
  - Também chamado "Console Application"
- Applets

Componentes J2EE na Camada Web

- Componentes da camada Web podem incluir vários módulos, incluindo:
  - Páginas HTML/XML estáticas
  - Servlets
    - Programas em Java que rodam no servidor Web e que processam pedidos gerando respostas dinâmicas
- Java Server pages (JSP)
  - Templates HTML mais fáceis de criar, mas contendo "scriplets" (trechos em Java) para a geração de conteúdo dinâmico
  - São convertidas em servlets quando acessadas pela primeira vez
- JavaBeans
  - Componentes tradicionais em Java que podem ser usados em servlets e JSPs

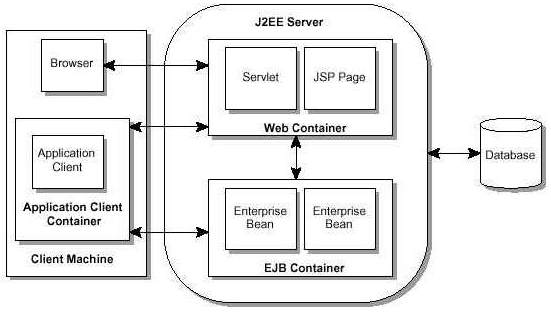
Componentes J2EE na Camada de Aplicação

Componentes da camada de aplicação são chamados Enterprise Java Beans (EJB)

- Há vários tipos de EJBs:
  - Session Beans
    - Representam uma conversação transiente com um cliente
    - Quando o cliente termina, a session bean some
  - Entity Bean
    - Representam dados persistentes gravados num banco de dados (tipicamente uma linha de uma tabela)
  - Message-Driven Bean
    - Uma combinação de um session bean com um Listener de mensagem Java Message Service (JMS)
    - Permite que um componente de aplicação (o message bean) receba mensagens assíncronas
    - Isso significa que podemos acoplamento muito fraco entre pedaços da aplicação: importante quando máquinas remotas estão envolvidas e podem nem estar no ar, ou pelo menos, poderão não responder de forma síncrona a chamadas remotas
    - Não falaremos desse tipo de Bean nesta disciplina

A camada de dados

- Observe que a camada de chamamos "de dados" pode ser um banco de dados ou outra coisa:
  - Por exemplo, pode ser um sistema ERP, CRM ou outro sistema legado
  - Por esse motivo, a camada freqüentemente é chamada de "camada EIS"



Containers e Serviços

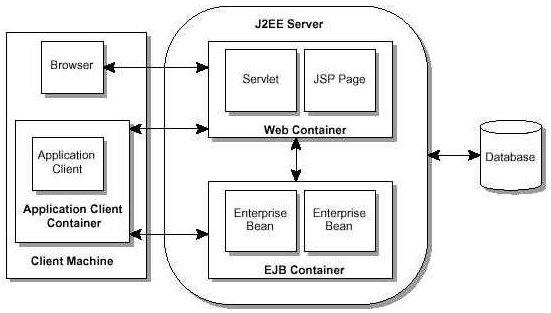
- A chave da arquitetura J2EE é que muito trabalho normalmente feito pelo programador é poupado, já que é feito automaticamente pelo middleware
- O programador se concentra no Business Logic
- A entidade que faz essa mágica é o Container
- Um container "envolve" um componente de forma a capturar mensagens dirigidas ao componente e fornecer serviços automáticos a este

- Portanto, antes de ser usado, um componente (seja cliente, Web ou EJB) deve:
  - Ser montado numa aplicação
  - Ser "deployed" (implantado) dentro de um container
- O container pode ser configurado em tempo de deployment
  - Com declarative programming, isto é, mudança de atributos
- Exemplos do que se faz no deployment ao configurar um container:
  - Estabelecer segurança
  - Estabelecer o tratamento transacional
  - Mapear nomes entre a aplicação e os recursos disponíveis
- O container também gerencia serviços não configuráveis:
  - O lifecycle dos componentes (achar, criar, destruir, ...)
  - Pooling de recursos (conexões de bancos de dados, por exemplo)
  - Persistência de dados

Tipos de Containers

- Os seguintes tipos de containers existem e executam no servidor J2EE:
  - Container EJB: um tal container para acolher algumas ou todas as Enterprise Beans (EJBs) de uma aplicação
  - Web container: um tal container para acolher algumas ou todas as JSPs e servlets de uma aplicação
- Os seguintes tipos de containers existem e executam na máquina cliente:
  - Application Client Container: para executar uma aplicação "console"
- Observe que servlets e JSPs podem executar sem um "J2EE server" completo
  - Podem executar num servidor Web com suporte especial sem ter suporte a EJB
  - Por exemplo: Apache Tomcat

Portanto, na figura abaixo, onde se vê "J2EE server", podemos ter, na realidade, máquinas diferentes



As APIs do J2EE

- Java 2 Platform, Standard Edition (J2SE™)
  - O antigo JDK
- Enterprise JavaBeans Technology
- JDBC API
  - Para acessar Bancos de Dados
- Java Servlet Technology
- JavaServer Pages (JSP) Technology
- Java Message Service (JMS)
  - Para comunicação assíncrona distribuída, fracamente acoplada e confiável
- Java Transaction API (JTA)
  - Para a demarcação de transações

- JavaMail™ Technology
  - Para que aplicações possam enviar mail
- Java API for XML Processing (JAXP)
  - Para implementar B2B, relatório XML, etc.
- J2EE Connector Architecture
  - Para se conectar de forma simples a vários sistemas de informação corporativos (ERP, BD
- Java Authentication and Authorization Service (JAAS)
  - Para prover serviços de autenticação e autorização

Computação distribuída

A computação distribuída, ou sistema distribuído, é uma referência à computação paralela e descentralizada, realizada por dois ou mais computadores conectados através de uma rede, cujo objetivo é concluir uma tarefa em comum.

Definição

- Um sistema distribuído é:
- "coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente";

Andrew Tanenbaum

- "coleção de computadores autônomos interligados através de uma rede de computadores e equipados com software que permita o compartilhamento dos recursos do sistema: hardware, software e dados"

George Coulouris

Definição

- O suporte completo de um sistema de banco de dados distribuídos implica que uma única aplicação seja capaz de operar de modo transparente sobre dados dispersos em uma variedade de banco de dados diferentes, gerenciados por vários SGBDs diferentes, em execução em uma variedade de máquinas diferentes que podem estar rodando em diversas plataformas diferentes e uma variedade de sistemas operacionais. Onde o modo transparente diz respeito à aplicação operar sob um ponto de vista lógico como se os dados fossem gerenciados por um único SGBD, funcionando em uma única máquina com apenas um sistema operacional.

Definição

- Assim, a computação distribuída consiste em adicionar o poder computacional de diversos computadores interligados por uma rede de computadores ou mais de um processador trabalhando em conjunto no mesmo computador, para processar colaborativamente determinada tarefa de forma coerente e transparente, ou seja, como se apenas um único e centralizado computador estivesse executando a tarefa. A união desses diversos computadores com o objetivo de compartilhar a execução de tarefas, é conhecida como sistema distribuído.

Organização

- Organizar a interação entre cada computador é primordial. Visando poder usar o maior número possível de máquinas e tipos de computadores, o protocolo ou canal de comunicação não pode conter ou usar nenhuma informação que possa não ser entendida por certas máquinas. Cuidados especiais também devem ser tomados para que as mensagens sejam entregues corretamente e que as mensagens inválidas sejam rejeitadas, caso contrário, levaria o sistema a cair ou até o resto da rede.

Organização

- Outro fator de importância, é a habilidade de mandar softwares para outros computadores de uma maneira portável de tal forma que ele possa executar e interagir com a rede existente. Isso pode não ser possível ou prático quando usando hardware e recursos diferentes, onde cada caso deve ser tratado separadamente com cross-compiling ou reescrevendo software.

Modelos de computação distribuída

- **Cliente/Servidor**
  - O *cliente* manda um pedido para o *servidor* e o *servidor* o retorna.
- **Peer-to-peer (P2P)**
  - É uma arquitetura de sistemas distribuídos caracterizada pela descentralização das funções na rede, onde cada nodo realiza tanto funções de servidor quanto de cliente.
- **Objetos Distribuídos**
  - Semelhante ao peer-to-peer, mas com um Middleware "mediador" intermediando o processo de comunicação.

Software

- **Fracamente acoplados**
  - Permitem que máquinas e usuários de um sistema distribuído sejam fundamentalmente independentes e ainda interagir de forma limitada quando isto for necessário, compartilhando discos, impressoras e outros recursos.
- **Fortemente acoplados**
  - provê um nível de integração e compartilhamento de recursos mais intenso e transparente ao usuário caracterizando sistemas operacionais distribuídos.

Cluster

Um cluster, ou aglomerado de computadores, é formado por um conjunto de computadores, que utiliza um tipo especial de sistema operacional classificado como sistema distribuído. Muitas vezes é construído a partir de computadores convencionais (personal computers), os quais são ligados em rede e comunicam-se através do sistema, trabalhando como se fossem uma única máquina de grande porte. Há diversos tipos de cluster. Um tipo famoso é o cluster da classe Beowulf, constituído por diversos nós escravos gerenciados por um só computador.

Tipos de cluster

- Cluster de Alto Desempenho:
  - Também conhecido como cluster de alta performance, ele funciona permitindo que ocorra uma grande carga de processamento com um volume alto de gigaflops em computadores comuns e utilizando sistema operacional gratuito, o que diminui seu custo.

Tipos de cluster

- Cluster de Alta Disponibilidade:
  - São clusters os quais seus sistemas conseguem permanecer ativos por um longo período de tempo e em plena condição de uso. Sendo assim, podemos dizer que eles nunca param seu funcionamento; além disso, conseguem detectar erros se protegendo de possíveis falhas..

Tipos de cluster

- Cluster para Balanceamento de Carga:
  - Esse tipo de cluster tem como função controlar a distribuição equilibrada do processamento. Requer um monitoramento constante na sua comunicação e em seus mecanismos de redundância, pois se ocorrer alguma falha, haverá uma interrupção no seu funcionamento.

O exemplo mais moderno desse paradigma é o BOINC, que é um framework de grade computacional no qual diversos projetos podem rodar suas aplicações, como fazem os projetos World Community Grid, SETI@Home, ClimatePrediction.net, Einstein@Home, PrimeGrid e OurGrid.

Referências

- Tanenbaum, Andrew S., *Distributed Systems: Principles and Paradigms*, pg. 2

Blackboard ou Quadro Negro

Blackboard

- Um Sistema de Quadro Negro é uma aplicação de inteligência artificial baseado no modelo de arquitetura blackboard, onde uma base de conhecimento comum, o quadro-negro ", é iterativa atualizado por um grupo diversificado de fontes de conhecimentos específicos, começando com uma especificação do problema e termina com uma solução.Cada fonte de conhecimento atualiza o quadro-negro com uma solução parcial, quando os seus condicionalismos internos correspondem ao estado negro. Desta forma, os especialistas trabalham juntos para resolver o problema. O modelo de quadro-negro foi originalmente concebido como uma forma de lidar com problemas complexos e mal definidos, onde a solução é a soma de suas partes.

Justificativa da denominação BlackBoard

- Herda da idéia de pessoas que trabalham juntas na frente de um quadro (blackboard) para resolver uma tarefa.
- O cenário a seguir fornece uma simples metáfora que dá algumas dicas sobre como o sistema blackboard funciona:
  - Um grupo de especialistas está sentado em uma sala com uma pedra grande. Eles trabalham como uma equipe para brainstorm uma solução para um problema, usando o quadro-negro como o trabalho cooperativo para o desenvolvimento da solução. A sessão começa quando as especificações do problema são gravados na pedra. Os especialistas de todo o relógio na lousa, procurando uma oportunidade de aplicar seus conhecimentos para a solução de desenvolvimento. Quando alguém escreve algo no quadro negro que permite que outro especialista para aplicar os seus conhecimentos, os registros segundo especialista o seu contributo no quadro-negro, esperemos que permitam outros especialistas para, em seguida, aplicar os seus conhecimentos. Este processo de adição de contribuições para o quadro-negro continua até que o problema foi resolvido.

A que se aplica

- Útil para problemas no qual não há uma solução determinística
- Uma coleção de programas independentes que trabalham cooperativamente em uma estrutura de dados comum (blackboard)
- Vários subsistemas especializados agregam seu conhecimento para conseguir uma possível solução aproximada para o problema
- Os subsistemas especializados são independentes uns dos outros
- BlackBoard= repositório de dados compartilhados

Histórico

Introduzido por Newell e Simon em 1972 inicialmente na teoria de IA.

HEARSAY II - primeiro sistema blackboard conhecido (sistema de reconhecimento de voz).

Introdução

- **Coleção de módulos (ou agentes) independentes que trabalham de maneira cooperada em uma estrutura de dados única (blackboard).**
- **Cada módulo é especializado em resolver uma parte específica do processamento.**
- **Todos os módulos trabalham em conjunto na solução do problema, mas são independentes entre si (não possuem comunicação direta entre eles).**
- **Uma central de controle determina o estado do processamento e coordena os módulos. O módulo mais adequado para a tarefa a ser realizada é chamado pelo controle.**
- **A aplicação do sistema negro consiste em três componentes principais:**

Componentes

1. Os módulos de software especializados,são chamados de fontes de conhecimento (KSS).
   - Como os especialistas humanos em um quadro-negro, cada fonte de conhecimento proporciona conhecimentos específicos necessários para a aplicação. A capacidade de suporte à interação e cooperação entre diversas SKs cria uma enorme flexibilidade na concepção e manutenção de aplicativos. Como o ritmo da tecnologia se intensificou, torna-se cada vez mais importante para ser capaz de substituir os módulos de software que se tornam ultrapassadas ou obsoletas.

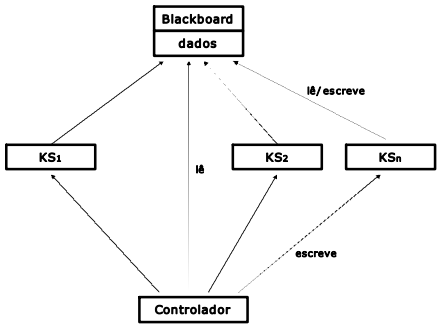
Componentes

2. O quadro:
   - Um repositório compartilhado de problemas, soluções parciais, sugestões e contribuíram com informações. O negro pode ser pensado como uma biblioteca dinâmica "das contribuições para o problema atual que foram recentemente" publicado "por outras fontes de conhecimento.

Componentes

3. O shell de controle:
   - Que controla o fluxo da atividade de resolução de problemas no sistema. Assim como os especialistas ansioso humanos precisam de um moderador para impedi-los de atropelamento em um traço louco para pegar o giz, KSS precisa de um mecanismo para organizar a sua utilização na forma mais eficaz e coerente. Em um sistema de quadro-negro, este é fornecido pelo shell de controle.

Notação



Passos para implementação em Blackboard

- **Definir o problema;**
- **Definir o espaço de soluções para o problema;**
- **Dividir a solução em passos;**
- **Dividir a solução em classes e suas tarefas específicas;**
- **Definir o vocabulário do Blackboard;**
- **Especificar o controle do sistema;**
- **Implementar as classes Knowledge Sources.**

Implementações

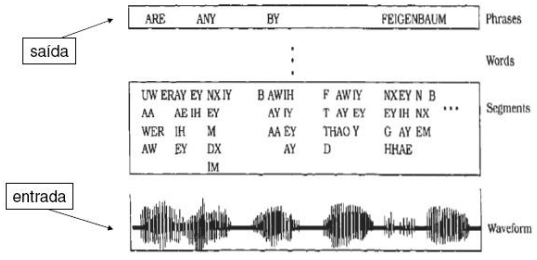
- Como exemplos famosos de início de sistemas de quadro-negro escolar são os boatos II sistema de reconhecimento de fala e Copycat Douglas Hofstadter e projetos Numbo.
- Exemplos mais recentes incluem implantado aplicações do mundo real, como o Plano de Componente do Sistema de Controle da Missão RADARSAT-1, um satélite de observação da Terra desenvolvido pelo Canadá para acompanhar a evolução do ambiente e recursos naturais da Terra.

Blackboard

- O padrão Blackboard é útil para problemas para os quais não se conhecem estratégias de soluções deterministas
- No Blackboard, uma série de subsistemas especializados combinam seu conhecimento para construir uma solução parcial ou aproximada para o problema

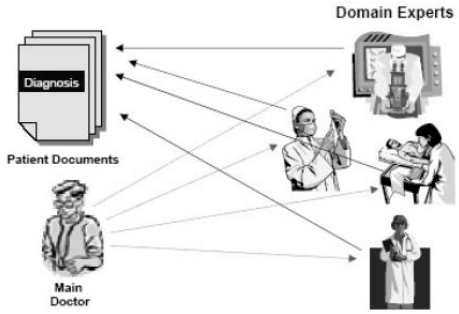
Um exemplo

- Considere um sistema de software para reconhecimento de fala
  - Entrada
    - Fala gravada (wave form)
    - Sistema aceita palavras e sentenças completas
  - Saída
    - Representação textual para as frases correspondentes
    - As transformações envolvidas requerem conhecimento acústico-fonético, lingüístico e estatístico



- Um procedimento divide a forma de onda em segmentos significativos no contexto da fala (fonemas)
- Outro procedimento verifica a sintaxe de candidatos a frases
- Os procedimentos trabalham em diferentes domínios

Outro Exemplo



Problema

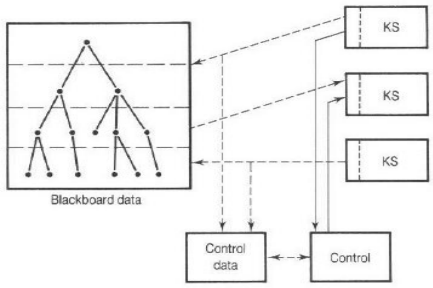
- Ataca problemas que não possuem uma solução determinista realizável, em geral transformando dados primitivos em estruturas de dados de alto nível, como diagramas, tabelas ou linguagem natural
  - Visão, reconhecimento de imagem, reconhecimento de fala
- Decomposição em subproblemas se expande por diversas áreas especializadas
- Soluções para subproblemas demandam diferentes representações e paradigmas
- Não existe estratégia pré-definida para combinação das soluções dos subproblemas (*not hard-coded)*

Solução

- Uma coleção de programas independentes que trabalham de forma cooperativa sobre uma estrutura de dados comum
- Cada programa é especializado na solução de uma parte do problema
- Os programas são independentes, não conhecem uns aos outros
- Um controle central avalia o estado corrente do processamento e coordena os programas especializados

Solução

- O conjunto de possíveis soluções é chamado de espaço de soluções (*solution space), e está organizado em níveis de* abstração



Estrutura

Classe	**Colaboradores**
Blackboard	• -
Responsabilidades:	
• Gerenciar dados centralizados	

Classe	**Colaboradores**
Knowledge Source	• Blackboard
Responsabilidades:	
• Avalia sua própria aplicabilidade	
• Processa um resultado	
• Atualiza o blackboard	

Classe	**Colaboradores**
Controle	• Blackboard
	• Knowledge Source
Responsabilidades:	
• Monitorar o Blackboard	
• Planejamento da ativação das KSs	

Estrutura

- Blackboard
  - O blackboard é um repositório central de dados;
  - Sua principal responsabilidade é a gerência de elementos do espaço de soluções e dados de controle;
  - O blackboard oferece uma interface que permite que todas as fontes de conhecimento leiam dados e gravem dados nele;

Estrutura

- Knowledge source
  - Knowledge sources (KS) são subsistemas independentes, separados, que resolvem aspectos específicos do problema;
  - KS não se comunicam diretamente; apenas interagem através do que lêem e escrevem no blackboard;
  - Nenhuma KS pode resolver o problema sozinha;
  - Cada KS é responsável por conhecer as condições em que pode contribuir para uma solução
    - (condition-part, action-part)

Estrutura

- Controle
  - O componente de controle executa um laço que monitora as mudanças no blackboard e decide qual a próxima ação;
  - Ele agenda avaliações e ativações de KSs de acordo com alguma estratégia de aplicação de conhecimento;
  - A base de tal estratégia são os dados no blackboard;
  - Podem existir KSs de controle;

Blackboard – forças

- Uma busca completa no espaço de soluções não é viável (tempo)
- Experimentação/facilidade de mudança
  - Domínios imaturos demandam experimentos com diferentes algoritmos
- Há diferentes algoritmos (não relacionados) que resolvem problemas parciais
- Diferentes representações de dados
  - Entradas, resultados intermediários e resultados finais possuem diferentes representações
- Um algoritmo trabalha com resultados de outros algoritmos
- Incerteza nos dados e soluções aproximadas estão envolvidos
- Paralelismo em potencial
  - Se possível, deve-se evitar solução estritamente seqüencial

Blackboard – implementação

- Defina o problema
- Defina o espaço de soluções (parciais...)
- Defina a solução do problema em pequenos passos
- Divida o conhecimento em KSs e algumas subtarefas
- Defina o vocabulário do blackboard
- Defina a estratégia de controle
- Implemente as KSs

☺ Vantagens

- Possibilita experimentação
- Facilidade de mudança e manutenção
  - Fontes de conhecimento e controle
- Reuso de fontes de conhecimento
- Suporte a tolerância a falhas e robustez
  - Hipóteses suportadas por dados

☹ Desvantagens

- Dificuldade para testar;
- Não há garantia de chegar a uma boa solução;
- Dificuldade para estabelecer uma boa estratégia de controle;
- Baixa eficiência;
- Alto esforço de desenvolvimento;
- Não suporta paralelismo;

Referências Bibliográficas

Software Architecture: a Roadmap. David Garlan. *22º International Conference on Software Engineering, 2000.*

A brief survey of software architecture. Rikard Land. Technical Report, *Departamento de Engenharia de Computação, Mälardalen University, Fevereiro 2002.*

Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. Frank Buschmann et. al. Wiley and Sons, 1996.

Extending UML for modeling and design of multi-agent systems. Krishna Kavi, *et. al.*

Referências Bibliográficas

The reflective blackboard architectural pattern. Otavio R. da Silva, Alessandro F. Garcia, Carlos J.P. de Lucena. *Software Engineering for Large-Scale Multi-Agent Systems, Springer-Verlag, LNCS 2603, Abril 2003.*

Pipes and Filters

(Dutos e filtros)

Dutos e filtros

- Esse padrão oferece uma estrutura para sistemas que processam fluxos de dados
  - Cada passo de processamento é encapsulado em um filtro
  - O dado é passado pelos dutos entre filtros adjacentes
  - Recombinação de filtros permite a construção de famílias de sistemas relacionados

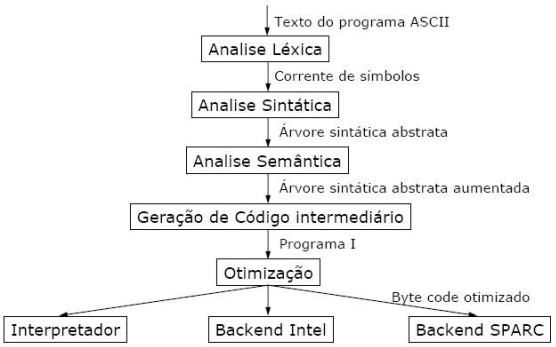
Componentes

- Filtro
- Data source
- Data sink
- Duto

Um exemplo

- Construção de um compilador portável para uma linguagem L
  - L é compilada para uma linguagem intermediária I que roda em uma máquina virtual VM
  - VM será implementada por um interpretador ou backends específicos para diferentes plataformas
  - Um backend irá traduzir o código para instruções de máquina de um processador específico para melhor desempenho

Um exemplo



Problema

- Imagine que você está construindo um sistema que deve processar ou transformar um stream de dados de entrada
  - A implementação do sistema como componente único não é possível por diversas razões
  - O sistema deve ser implementado por várias pessoas,
  - A tarefa pode ser decomposta naturalmente em diversos passos de processamento, e
  - Os passos de processamento podem ser modificados, reordenados ou reutilizados em outros contextos

Solução

- A arquitetura de Dutos e Filtros divide as tarefas do sistema em diversos passos de processamento seqüencial
- Os passos estão conectados pelo fluxo de dados através do sistema: a saída de um passo é a entrada do passo seguinte
- Cada passo é implementado por um componente chamada filtro
- Um filtro consome e entrega dados de forma incremental

Solução

- A entrada para o sistema é feita através de uma fonte de dados (data source), por exemplo, um arquivo texto
- A saída flui para um sorvedouro de dados (data sink), por exemplo, um arquivo, programa, etc.
- A fonte de dados, os filtros e a saída de dados estão conectados seqüencialmente por dutos (dutos)
- Cada duto implementa o fluxo de dados entre passos de processamento adjacentes.
- A seqüência de filtros combinados através de dutos é chamada de *processing pipeline*

Estrutura

Classe	**Colaboradores**
Filter	• Pipe
Responsabilidades:	
• Pega o dado de entrada • Processa uma função sobre o dado de entrada • Fornece o dado de saída	

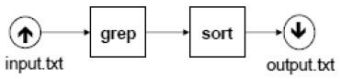
Classe	**Colaboradores**
Pipe	• Data Source • Data Sink • Filter
Responsabilidades:	
• Transferência de dados • Armazenamento temporário de Dados • Sincronização de atividades vizinhas	

Classe	**Colaboradores**
Data Source	• Pipe
Responsabilidades:	
• Fornece entrada para a linha de processamento	

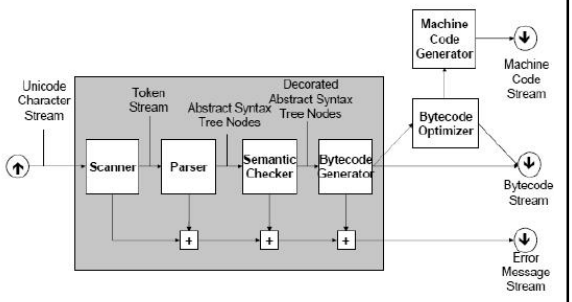
Classe	**Colaboradores**
Data Sink	• Pipe
Responsabilidades:	
• Consume a saída.	

Dutos e filtros

- Sistemas operacionais e compliladores são os melhores exemplos de arquitetura dutos e filtros
- Exemplo SO
  - Unix shell:
    - cat input.txt | grep “text” | sort > output.txt



Exemplo compilador



Dutos e filtros – forças

- Melhorias futuras do sistema devem ser capazes de suportar troca de passos ou recombinação
- Pequenos passos de processamento são fáceis de serem reutilizados em diferentes contextos
- Passos não adjacentes não devem compartilhar informação
- Existem diferentes tipos de fontes de dados de entrada
- Deve ser possível apresentar ou armazenar dados de saídas em diversas formas
- Armazenamento de resultados intermediários pelos usuários é atividade propensa a erros
- Deve-se prever a possibilidade de processamento paralelo dos passos

Dutos e filtros – implementação

- Divida a tarefa do sistema em uma seqüência de passos de processamento
  - Cada estágio deve depender apenas da saída do seu predecessor direto
  - Todos os estágios são conectados pelo fluxo de dados
- Defina o formato dos dados que serão passados por cada duto
- Decida como implementar cada conexão de duto
- Projete implemente os filtros
- Projete o tratamento de erros

☺ Vantagens

- Flexibilidade para modificação/substituição de filtros;
- Flexibilidade para recombinação de processos;
- Reuso de atividades de processo;
- Ajuda na prototipação rápida

☹ Desvantagens

- Compartilhamento de informação de estado é caro ou inflexível
- Eficiência ganha com processamento paralelo pode ser contestada
  - Custo de transferência de dados pode ser alto
  - Alguns filtros podem exigir que toda entrada esteja disponível (não incremental)
  - Sincronização de filtros pode gerar paradas
- Overhead de conversão de dados (uniformidade)
- Tratamento de erros é um ponto crítico