Sistemas Distribuídos na WEB

(Plataformas para Aplicações Distribuídas)

Tecnologias e Motivações

Sumário

- Paradigmas e Tecnologias para Desenvolvimento de SDs
- Sistemas Distribuídos x Tecnologia da Informação (Prof. Jacques Philippe Sauvé, UFP)
  - Problemas
  - Soluções

Paradigmas

- Troca de Mensagens (Messaging)
- Remote Procedure Call (RPC)
- Objetos Distribuídos
  - CORBA,DCOM e Java RMI
- Componentes Distribuídos
  - J2EE e .Net
- Web Services

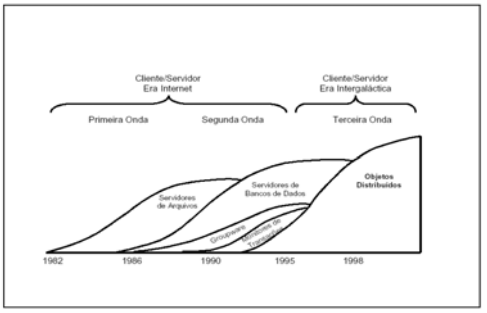
Objetos Distribuídos

- Objetos:
  - Entidades auto-suficientes que encapsulam dados e conjunto de operações sobre os mesmos.
  - Pilares
    - Herança
    - Polimorfirsmo
    - Encapsulamento
  - Reutilização
    - Biblioteca de classes
    - Frameworks
  - Residem num único programa
  - Não têm vida própria fora do programa
  - Não têm acesso a objetos externos

Objetos Distribuídos

- *Orientação a Objetos + Sistemas Distribuídos = Objetos Distribuídos*
  - *Modelo Cliente/Servidor*
  - *Entidades inteligentes independentes*
  - *Residem em qualquer local da rede*
  - *Acessados por outros objetos via invocação remota de métodos*
  - *Continuam a existir enquanto são acessados por outros objetos*
  - *Objetos escritos em diferentes linguagens e compilados em diferentes compiladores podem interagir*

Objetos Distribuídos



As "Ondas" de Cliente/Servidor

Objetos Distribuídos

- *Necessidades:*
  - *Mecanismos de comunicação lógicos que escondam detalhes de implementação*
  - *Compatibilidade entre componentes criados em diversas linguagens*
  - *Suporte a diferentes plataformas (hardware, sistema operacional, etc.)*

Objetos Distribuídos

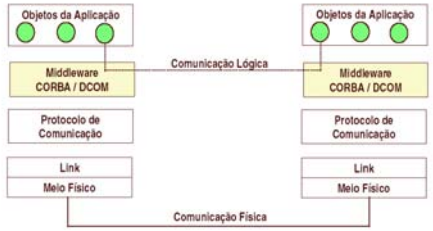
- *Middleware:*
  - *Software distribuído que facilita a interação cliente/servidor*
  - *Permite acesso remoto transparente a serviços e recursos distribuídos em um rede*
  - *O middleware serve como uma camada que provê comunicação estruturada entre clientes e servidores*
  - *Possui uma API utilizada por clientes para requisitar serviços de servidores e controlar a transmissão física de requisições através da rede e do envio dos resultados*

Objetos Distribuídos

- *Middleware:*
  - *Executa tanto no cliente como no servidor*
  - *Exemplos de Middleware:*
    - *Para Bancos de Dados: ODBC, SQL e Oracle Glue;*
    - *Para Groupware: Microsoft Exchange e Lotus Notes;*
    - *Para Internet: HTTP e SSL;*
    - *Para objetos distribuídos: CORBA e DCOM*
  - *Middlewares estão se tornando componentes padrões de sistemas operacionais modernos*

Objetos Distribuídos

Modelo de Objetos Distribuídos



Objetos Distribuídos
(Tecnologia CORBA)

- *Common Object Request Broker Architecture*
- *Desenvolvido pelo Object Management Group (OMG)*
  - *Fundado em Maio de 1989 por 3Com, American Airlines, Canon, Data General, HP, Phillips, Sun e Unisys*
  - *Hoje é um dos maiores grupos de padronização com mais de 700 companhias e organizações pelo mundo*
  - *Arquitetura de Gerenciamento de Objeto – Object Management Architecture (OMA)*
    - *Objetos são os elementos chave, que provêem interfaces através das quais disponibilizam funcionalidade a outros objetos*
  - *Linguagens que suportam: C++, C, Java, Smalltalk, Cobol, Ada, etc.*

Objetos Distribuídos
(Tecnologia CORBA)

- *Provê serviços fundamentais para gerenciamento de objetos*
- *Separação total da implementação de um objeto e da interface*
  - *Para cliente acessar servidor, basta conhecer a interface;*
  - *Sistema operacional do servidor é escondido do cliente.*
- *Permite descoberta de outros objetos na rede*
- *Objetos podem existir em qualquer lugar na rede*
- *Localização de objetos é completamente transparente*

Objetos Distribuídos
(Tecnologia DCOM)

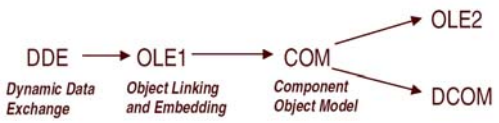
- Distributed Component Object Model
- Desenvolvido pela Microsoft
  - Surgiu em 1990
  - Sofreu várias alteração de nome até chegar no estado atual
  - Primeira versão de DCOM disponível no Windows NT 4.0
- Usada pelos componentes ActiveX
- Linguagens suportadas: Visual C++, Visual J++, Visual Basic
- Separa interface da funcionalidade usando uma IDL

Objetos Distribuídos
(Tecnologia DCOM)

- Não suporta a visão tradicional de objetos
  - "objetos" DCOM não possuem estado, são uma coleção de interfaces
  - Quando uma referência a um objeto é solicitada, ela é arbitrária
  - Referências subseqüentes podem gerar referências diferentes

Objetos Distribuídos
(Tecnologia DCOM)

Resumo da Evolução



Objetos Distribuídos
(Tecnologia DCOM)

- Evolução
  - DDE
    - Basicamente um clipboard para o copiar e colar
    - Para o desenvolvedor não era nada fácil de utilizar
    - Dados trocados por mensagens Windows com dois parâmetros: LPARAM de 32 bits e WPARAM que podia ser de 32 ou 16 bits de acordo com o sistema operacional.
    - Uma vez que o número e o tipo de parâmetros é fixo no Windows, o único jeito de trocar estruturas de dados complexas era através do LPARAM como ponteiro para a área a enviar
    - Permitia que um aplicativo acessasse a área de memória de outro (leva a muitos erros)

Objetos Distribuídos
(Tecnologia DCOM)

- Evolução
  - OLE 1
    - Construído a partir do DDE
    - Introduz objetos como abstrações para partes isoladas do aplicativo que podem chamar umas as outras
    - Quando um aplicativo deseja modificar um objeto OLE1, a modificação é feita no contexto da aplicação que originalmente criou o objeto
    - OLE1 necessitava manipular objetos difíceis de manipular com DDE
  - COM
    - Uma grande modificação no DDE, usado como base para o OLE2
    - É um padrão binário que especifica a área de memória dos objetos que podem ser acessados por outros aplicativos

Objetos Distribuídos
(Tecnologia DCOM)

- Evolução
  - DCOM
    - Estende o modelo COM permitindo comunicação via rede
    - Objetos acessam proxies dentro da própria aplicação que enviam mensagens para o objeto real com o nome das funções chamadas e os parâmetros passados
    - O objeto real pode residir fisicamente em um outro computador
    - Inspirado no Distributed Computing Environment (DCE) criado pela Open Software Fundation (OSF)
      - Usa chamadas a procedimentos remotos (RPC) do DCE e uma IDL que é uma pequena modificação da DCE IDL

Objetos Distribuídos
(Tecnologia Java RMI)

- JAVA Remote Method Invocation
- RMI estende a linguagem JAVA para suportar objetos distribuídos
  - Permite objetos invocar métodos remotos usando a mesma sintaxe da invocação local
  - Entretanto, objeto que faz invocação remota deve lidar com diferentes exceções (RemoteExceptions)
  - O uso de objetos remotos não é transparente, deve ser explicitado através da implementação da interface Remote.
  - Semântica de passagem de parâmetros é diferente

Problemas

- Classes de problemas
  - Pressões do negócio
  - Pressões tecnológicas

Problemas
Pressões do negócio

- Steve Jobs: "Internet time" é um novo conceito (1997-1998) 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

Problemas
Pressões do negócio

- 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
  - Hoje (2001), de 4 a 6% da receita é gasta com TI! É altíssimo!

Problemas
Pressões tecnológicas

- Evolução tecnológica força mudanças de sistemas
  - Muitas empresas acabam de fazer a transição para cliente/servidor e agora a palavra de ordem é "client/server is dead!"
  - Isto é, client/server em 2 camadas
- Sistemas devem migrar para arquiteturas distribuídas N-tier
  - Para ter escala
  - Para ter acesso Internet mas com acesso a sistemas legados
    - Não podemos jogar o legado fora
  - Para juntar sistemas operando em plataformas diferentes

Problemas
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

Problemas
O que queremos?

- *O que grandes empresas, com grandes problemas de desenvolvimento de sistemas, querem?*
- *Na visão do usuário, software de qualidade:*
  - *Tem que satisfazer as necessidades do usuário*
  - *Tem que ser feito no prazo combinado*
  - *Tem que ser feito dentro do custo combinado*
  - *Tem que ter nível aceitável de defeitos*

Problemas
O que queremos?

- *Mas queremos ver a visão do desenvolvedor ...*
- **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*

Problemas
O que queremos?

- **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*

Problemas
O que queremos?

- **Melhor escalabilidade**
  - *Possibilitando distribuir e configurar a execução da aplicação para satisfazer vários volumes de transação*
- **Menor tempo de desenvolvimento**
  - *Possibilitando viver em "Internet time" e com baixo orçamento*
- **Melhor robustez**
  - *Possibilitando ter soluções com menos defeitos*
- **Menor risco**
  - *Possibilitando tudo que falamos acima e ainda não se arriscar a ter projetos fracassados*

Problemas
Orientação a Objetos não está resolvendo?

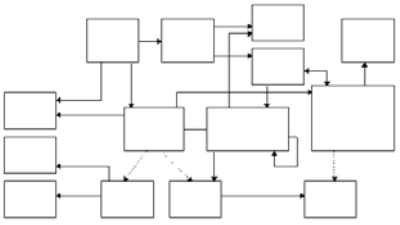
- Muitos grandes projetos baseados em OO estão fracassando recentemente
- Aparentemente, OO não está à altura
- OO é apenas uma Enabling Technology
  - Benefícios não são automáticos mas dependem do uso correto da tecnologia
- Precisamos descobrir o que acrescentar a OO para melhorar as coisas

Problemas
Orientação a Objetos não está resolvendo?

- Onde OO peca?
  - OO (sozinha) não produz software reutilizável automaticamente
  - OO (sozinha) não tem boa escalabilidade
  - OO (sozinha) não provê boa encapsulação (esconder informação)
- Isto pode ser visto com o resultado do uso de OO sem disciplina
  - Hyperspaghetti Objects

Problemas
Orientação a Objetos não está resolvendo?

Hyperspaghetti Objects



Problemas
Orientação a Objetos não está resolvendo?

- Ocorre mais quando software cresce em tamanho
  - Um objeto pode referenciar qualquer outro
  - Referências demais não permitem arrancar um objeto e reutilizá-lo sozinho
  - Manutenção difícil, causando instabilidade
  - Tirar bugs introduz novos bugs
- Resultado: o programa está muito complexo e não gerenciável

Problemas
Orientação a Objetos não está resolvendo?

- Onde OO peca mais?
  - OO (sozinha) não permite a construção de sistemas verdadeiramente extensíveis
    - Para ser verdadeiramente extensível, um sistema deve permitir a adição tardia de uma extensão sem necessitar de uma verificação global de integridade
    - Em outras palavras, para adicionar algo, somos obrigados a passar por compilação e/ou link edição, testes de integração, etc.
    - Queremos fazer mais ou menos como num sistema operacional
      - Aplicações estendem o SO mas não precisamos de um teste total de sistema
      - Temos problemas de instalação e configuração, apenas

Problemas
Orientação a Objetos não está resolvendo?

- O uso de herança em OO cria um acoplamento muito forte entre os pedaços
  - Por este motivo, a extensão de uma classe usando herança (de implementação) requer freqüentemente que se tenha o código fonte da classe sendo estendida
  - Muitos fornecedores de bibliotecas OO fornecem código fonte por este motivo

Soluções

- Precisamos de novas abordagens:
  - Aplicações Multicamadas Distribuídas
  - Desenvolvimento Baseado em Componentes

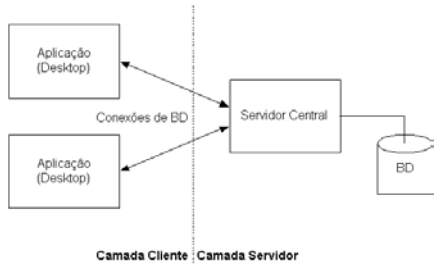
Soluções
Aplicações Multicamadas Distribuídas

- Faremos um resumo histórico das arquiteturas de aplicações
- **Arquitetura centralizada**
  - Dominantes até década de 80 como arquitetura corporativa
  - Problema básico: interface não amigável

Soluções
Aplicações Multicamadas Distribuídas

- **Arquitetura em 2 camadas**
  - *Sistemas em camadas surgiram para:*
    - *Melhor aproveitar os PCs da empresa*
    - *Oferecer sistemas com interfaces gráficas amigáveis*
  - *Integrar o desktop e os dados corporativos*
    - *Em outras palavras, permitiram aumentar a escalabilidade de uso de Sistemas de Informação*
    - *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)*

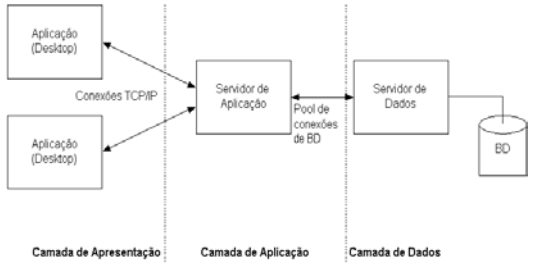
Soluções
Aplicações Multicamadas Distribuídas



Soluções
Aplicações Muticamadas Distribuídas

- **Arquitetura em 3 camadas**
  - *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 (legado CICS, 3270, ...)*
  - *Inventou-se a arquitetura em 3 camadas*
    - *Camada de apresentação (UI)*
    - *Camada de aplicação (business logic)*
    - *Camada de dados*

Soluções
Aplicações Muticamadas Distribuídas



Soluções
Aplicações Muticamadas Distribuídas

- *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*

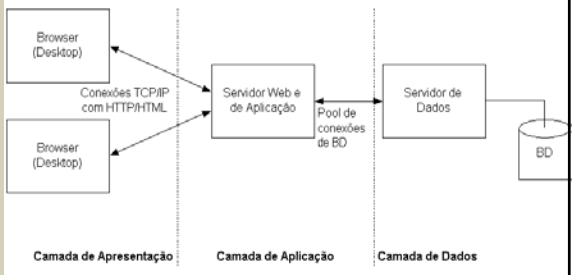
Soluções
Aplicações Muticamadas Distribuídas

- **Arquitetura em 3/4 camadas Web-Based**
  - *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*

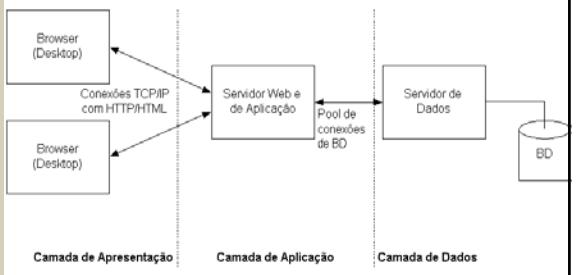
Soluções
Aplicações Muticamadas Distribuídas

- *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)*

Soluções
Aplicações Muticamadas Distribuídas



Soluções
Aplicações Muticamadas Distribuídas



Soluções
Aplicações Muticamadas Distribuídas

- **Arquitetura distribuída em n camadas**
  - *Os problemas remanescentes:*
  - *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?*

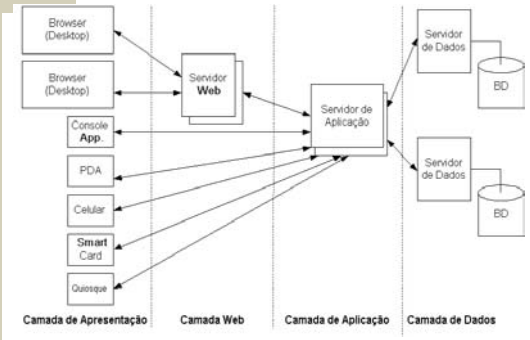
Soluções
Aplicações Muticamadas Distribuídas

- **Arquitetura distribuída em n camadas**
  - *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*
    - *etc.*

Soluções
Aplicações Muticamadas Distribuídas

- *As empresas não querem contratar PhDs para implementar sistemas de informação!*
- *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*

Soluções
Aplicações Muticamadas Distribuídas



Soluções
Aplicações Muticamadas Distribuídas

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

Soluções
Componentes

- *Frameworks + Componentes*
- *Idéias Básicas:*
  - *Vamos examinar mercados bem sucedidos para ver o que eles fazem diferente do que é feito no mercado de software*
  - *Como mercados maduros dão boa relação qualidade/preço?*
  - *Mercado industrial ensina que a manufatura (carros, p. ex.) não tem muitos dos problemas da construção de software*
    - *Porque a manufatura usa Componentes Pré-Fabricados*

Soluções
Componentes

- *Idéias Básicas:*
  - *Pense em carros, bicicletas, hardware (com circuitos integrados)*
  - *Por que usar componentes é bom?*
    - *A empresa não tem que fazer tudo*
    - *Ela se concentra no que ela é boa*
    - *Ela compra componentes de outros especialistas*
    - *É a boa e velha "terceirização"*
    - *Desta forma, pode-se focar melhor apenas no que falta*

Soluções
Componentes

- Resultados:
  - Componentes reutilizáveis
  - Compre, não construa porque deve ser mais barato
  - Pare de escrever aplicações do zero cada vez que inicia um projeto
  - Muda a ênfase da programação (construção) para a composição (montagem)
    - É como usar bloquinhos Lego

Soluções
Componentes

- Precisamos de uma forma simples de dinamicamente conectar os componentes entre si mas não em tempo de compilação ou link-edição
  - Quero usar uma ferramenta visual para
    - Configurar os componentes
    - Interconectar os componentes (estabelecer associações)
  - Estou "programando" sem codificar
  - Também chamado de
    - Attribute programming
    - Visual programming
    - Interactive programming
    - Rapid Application Development (RAD)

Soluções
Componentes

- Isso leva ao conceito de "design time" (ou Assembly Time)
  - E, na realidade, a um novo processo de desenvolvimento
- O que são Componentes?
  - Muitas definições foram oferecidas ao longo dos últimos anos (ver definições)

Soluções
Componentes

- Apesar da confusão nas definições, podemos enxergar algo em comum
- Vamos iniciar com a definição mais geral de D'Souza
  - Componente geral: "Um pacote coerente de artefatos de software que pode ser desenvolvido independentemente e entregue como unidade e que pode ser composto, sem mudança, com outros componentes para construir algo maior."

Soluções
Componentes

- Baseando-se nesta definição, todas as coisas que seguem poderiam ser consideradas "componentes", pelo menos em espírito
- Usando essa definição, um componente pode incluir:
  - Código executável
  - Código fonte
  - Projetos (designs)
  - Especificações
  - Testes
  - Documentação
  - etc.

Soluções
Componentes

- Mas nós queremos falar apenas de componentes de implementação
- Definição de D'Souza, mais uma vez:
  - Componente de implementação: "Um pacote coerente de implementação de software que (a) pode ser desenvolvido independentemente e entregue como unidade; (b) tem interfaces explícitas e bem definidas para os serviços que oferece; (c) tem interfaces explícitas e bem definidas para os serviços que requer; e (d) pode ser composto com outros componentes, talvez após a customização de algumas propriedades mas sem modificar os componentes em si."

Soluções
Componentes

- Examinando esta definição (e as outras), podemos ver o que componentes têm de diferente comparados a bibliotecas ou outros artefatos de software
  - São 3 características básicas
    - Construção de aplicações por montagem
    - Um componente explicita suas interfaces
    - Um componente é uma unidade de empacotamento (packaging), entrega (delivery), implantação (deployment), e carga (loading)

Soluções
Componentes versus Objetos

- As diferenças geram controvérsia
- Instanciação
  - A instanciação de um componente não gera outro componente mas um "objeto" criado a partir de um protótipo (o componente)
  - Um componente freqüentemente é visto como um factory de objetos e não um objeto em si
- Em termos de linguagem OO, um componente pode conter vários objetos
  - Componentes têm granularidade maior, freqüentemente

Soluções
Componentes versus Objetos

- Componentes freqüentemente tratam de sua persistência
  - Objetos não fazem isso
- Objetos não são empacotados como componentes
  - Componentes são sujeitos à composição em tempo de design
- Se o componente obedece às suas interfaces, ele pode ser implementado em qualquer linguagem (mesmo sem ser OO!) e rodar em qualquer plataforma
  - Objetos são manipulados com linguagens OO apenas

Soluções
Categorias de Componentes

- Podemos classificar componentes sob várias dimensões
  - Escopo
    - Componentes de Especificação (diagramas, documentos)
    - Componentes de Implementação (classes, ...)
  - Objetivo
    - Domínio (voltado ao problema)
    - Tecnologia (para o suporte, infra-estrutura)
  - Abstração
    - Geral (aplicação em muitos domínios: horizontal)
    - Específico (aplicação em um único ou poucos domínios: vertical)

Soluções
Categorias de Componentes

- Podemos classificar componentes sob várias dimensões
  - Granularidade
    - Fina (um widget GUI)
    - Grossa (shopping-cart, agência bancária, fatura, contas a receber)
      - As oportunidades de reuso podem ser menores mas tais componentes são cruciais para melhorar a produtividade no desenvolvimento

Soluções
Categorias de Componentes

- Podemos classificar componentes sob várias dimensões
  - Localização
    - Cliente
    - Servidor (Middle tier)
  - Serviços providos
    - Gerência de configuração e de propriedades, Notificação de eventos, Acesso a metadados e reflexão,Persistência,Controle de transação e concorrência, Segurança, Licenciamento, Controle de versão, Autotestes e Auto-instalação