

Ambientes de Desenvolvimento Avançados

<http://www.dei.isep.ipp.pt/~jtavares/ADAV/ADAV.htm>

Aula 4 Engenharia Informática

2006/2007

José António Tavares
jrt@isep.ipp.pt

1



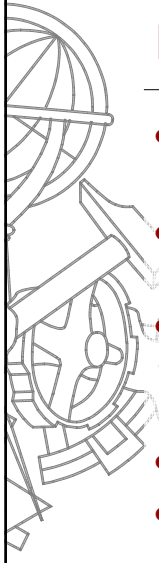
Introdução ao conceito Cliente-Servidor

Baseado num documento de Eng^o *Alexandre Bragança* - 1998/99

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

2



Motivação para os componentes

- O desenvolvimento da *WWW* e *Internet*
 - Sistemas de serviços poucos coordenados
- Técnicas de projecto e linguagens Orientadas aos Objectos
- Movimento da computação baseada em *Mainframes* para computação do tipo **Cliente-Servidor**
- Ritmo rápido das mudanças tecnológicas
- Necessidades económicas de maximizar a reutilização

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

3



Desenvolvimento de sistemas Cliente-Servidor - Agenda

- Características de um sistema cliente-servidor
- Diferentes 'versões' de cliente-servidor
- *Middleware*
- Balanceamento cliente-servidor
- Cliente-servidor 2 camadas vs 3 camadas
- Componentes de uma arquitectura cliente-servidor

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

4



Características de um sistema Cliente-Servidor

- **Serviço**

Cliente-servidor é uma relação entre processos que estão a correr em máquinas diferentes. O processo servidor é o fornecedor dos serviços. O cliente é o consumidor de serviços. Fundamentalmente uma arquitectura cliente-servidor implemente uma separação lógica de funções baseada no conceito de serviço.

- **Recursos partilhados**

Um servidor pode servir vários clientes ao mesmo tempo e gerir os acessos a recursos partilhados.

- **Protocolos assimétricos**

Existe uma relação de muitos-para-um entre clientes e servidor. Os clientes iniciam o diálogo através da requisição de um serviço. Os servidores esperam passivamente os pedidos dos clientes.



Características de um sistema Cliente-Servidor

- **Localização transparente**

O servidor é um processo que pode residir na mesma máquina que o cliente ou numa máquina diferente que esteja ligada através de uma rede. Um programa pode ter o papel de cliente, servidor ou ambos.

- **Independência**

O conceito inerente às arquitecturas cliente-servidor baseia-se em *software* que deve ser independente de *hardware* ou sistemas operativos.

- **Baseado na transmissão de mensagens**

Clientes e servidores devem estar ligados de forma 'fraca', ou seja, não deve ser obrigatório que o servidor esteja a correr para que o cliente possa correr. Sistemas deste tipo são normalmente baseados em mensagens. A mensagem é o mecanismo de transporte para os pedidos e respostas dos serviços.



Características de um sistema Cliente-Servidor

- **Encapsulamento de serviços**

Um servidor deve ser um programa 'especializado'. As mensagens transmitem o pedido de serviço ao servidor. O servidor é que deve ser responsável pela forma como implementa o serviço. A forma de implementar os serviços pode ser melhorada/alterada sem implicações ao nível dos clientes.

- **Escalabilidade**

Os sistemas cliente-servidor podem evoluir facilmente quer por adição de novos clientes quer por evolução para novas máquinas servidoras mais potentes.

- **Integridade**

O código e dados do servidor devem ser mantidos centralmente. Desta forma reduzem-se os custos de manutenção e aumenta-se a integridade dos dados.

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

7



Diferentes 'versões' de Cliente-Servidor

- Servidores de ficheiros
- Servidores de bases de dados
- Servidores de transacções
- Servidores de *groupware*
- Servidores de objectos
- Servidores de web

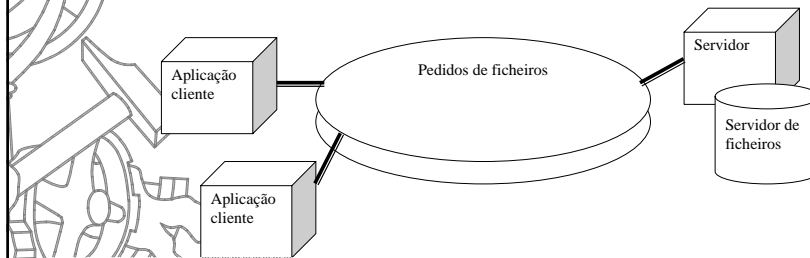
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

8

Diferentes 'versões' de Cliente-Servidor

Servidores de ficheiros



Num sistema deste género o cliente executa pedidos de registos de ficheiros ao servidor de ficheiros através da rede.

É uma forma muito primitiva de serviço de dados e provoca uma troca muito elevada de mensagens pela rede.

São no entanto sistemas necessários para a partilha repositórios de ficheiros em rede (documentos, imagens, desenhos, etc....).

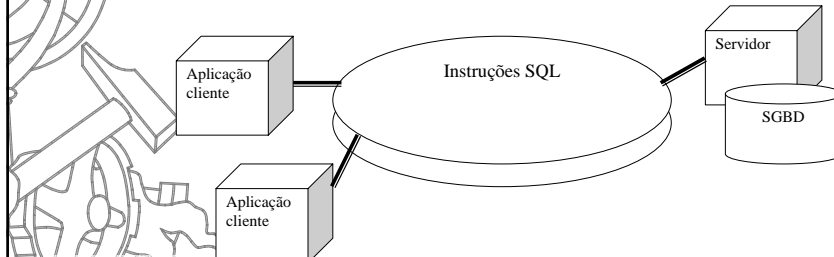
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

9

Diferentes 'versões' de Cliente-Servidor

Servidores de bases de dados



Num servidor de base de dados o que é transmitido na rede (do cliente para o servidor) são instruções de SQL. O resultado da instruções de SQL são envidas para o cliente. O código que processa as instruções SQL e os dados residem na mesma máquina (o servidor). É o servidor que determina quais os registos resultantes da instrução e são apenas estes que são enviados pela rede.

Nos servidores de ficheiros todos os registos são enviados pela rede e é o cliente que determina aqueles que interessam.

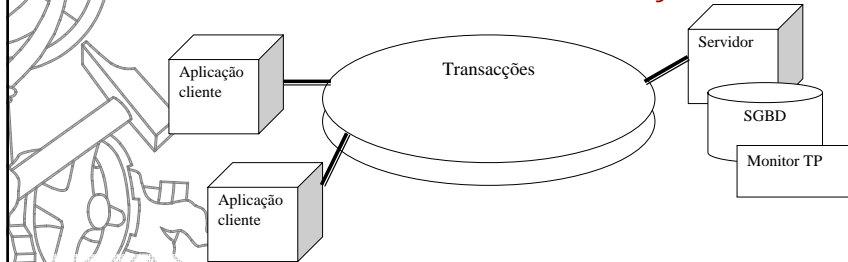
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

10

Diferentes 'versões' de Cliente-Servidor

Servidores de transacções



Com os servidores de transacções os clientes invocam procedimentos remotos que residem no servidor (com uma base de dados). Estes procedimentos remotos são constituídos por grupos de instruções SQL. As instruções do procedimento são executadas na totalidade ou então falha tudo.

Ao contrário do simples servidor de base de dados neste caso o programador tem que escrever código no cliente e no servidor.

Estes sistemas usualmente designam-se de OLTP (*Online Transaction Processing*)

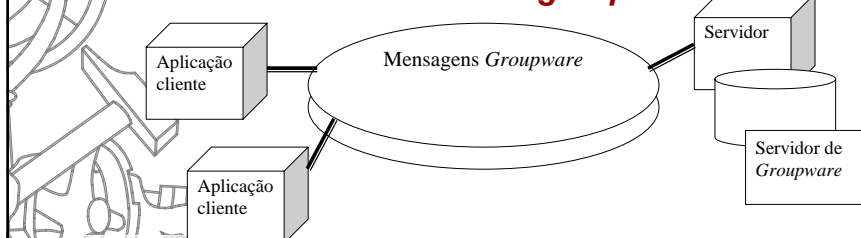
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

11

Diferentes 'versões' de Cliente-Servidor

Servidores de *groupware*



O objectivo dos sistemas *groupware* é o de facilitar a gestão de informação semi-estruturada (ou não-estruturada) tal como texto, imagem, e-mail, etc..

Para além disso normalmente estes sistemas também implementam capacidades de automação de *workflow*.

Estes sistemas suportam-se sobre sistemas de transmissão de mensagens. Existem diversos sistemas que embora se possam interligar são implementados de formas diferentes.

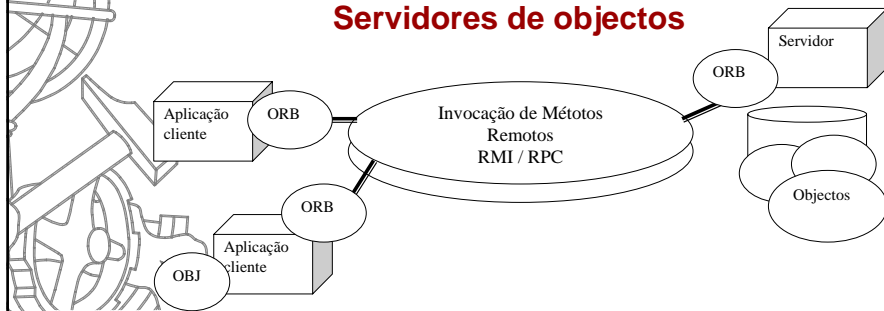
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

12

Diferentes 'versões' de Cliente-Servidor

Servidores de objectos



Um sistema deste tipo é implementado através de um conjunto de objectos que podem comunicar entre si. Objectos cliente comunicam com objectos servidores através do *Object Request Broker* (ORB).

Quando o cliente invoca um método num objecto remoto o ORB localiza a instância do objecto servidor, invoca o método e retorna o resultado ao objecto cliente.

As tecnologias concorrentes nesta área são o CORBA e o DCOM.

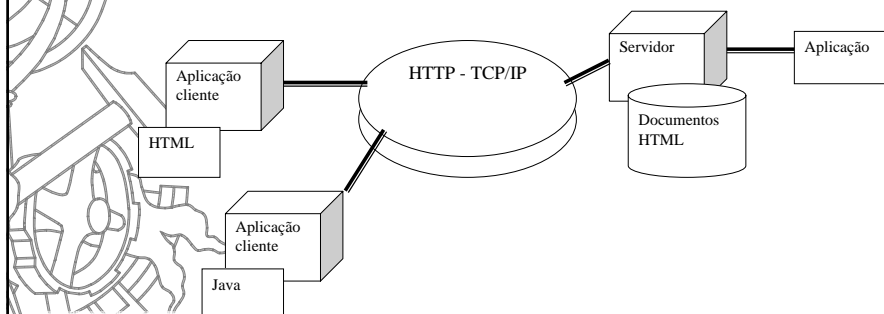
2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

13

Diferentes 'versões' de Cliente-Servidor

Servidores de web



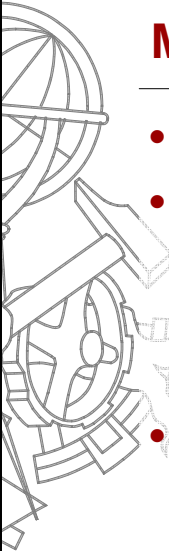
O novo modelo introduzido pela internet consiste em clientes 'leves', 'portáteis' e 'universais' que comunicam com servidores 'super pesados' (servem milhares ou milhões de clientes).

A comunicação é feita por um protocolo do tipo RPC designado por HTTP.

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

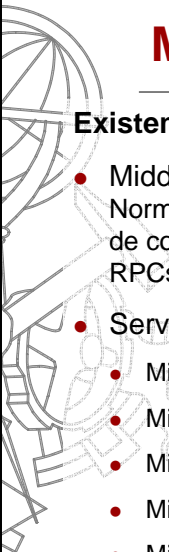
14



Middleware

- 'Aquila' que se encontra entre o cliente e o servidor
- Começa com a API que o cliente utiliza para invocar um serviço, inclui a transmissão do pedido pela rede assim como da resposta. O Middleware não inclui o *software* que executa o serviço (esta é a função do servidor)
- O Middleware não inclui o interface (esta é uma função do cliente)

2006/2007 ADAV 15
Ambientes de Desenvolvimento Avançados



Middleware

Existem 2 categorias de middleware

- Middleware genérico
Normalmente inclui tudo o que tem que ver com transporte (*stacks* de comunicação, serviços de directório, serviços de autenticação, RPCs, etc.).
- Serviços de Middleware
 - Middleware específico de base de dados (ODBC...)
 - Middleware específico de groupware (MAPI, VIM...)
 - Middleware específico de serviços de objectos (CORBA, DCOM...)
 - Middleware específico de internet (HTTP, SSL...)
 - Middleware de gestão específico (ORB...)

2006/2007 ADAV 16
Ambientes de Desenvolvimento Avançados



Componentes de uma arquitectura Cliente-Servidor

- **Cliente**

Baseia-se fundamentalmente no interface gráfico da aplicação. Cada vez mais este é um “*Object Oriented User Interface*” (OOUI). Acede a serviços através do middleware

- **Servidor**

Executa a parte de serviços da aplicação. Utiliza normalmente servidores específicos de software (SGBDs (relacionais ou OO), Monitores TP, servidores de groupware, servidores de objectos e servidores de Web)

- **Middleware**

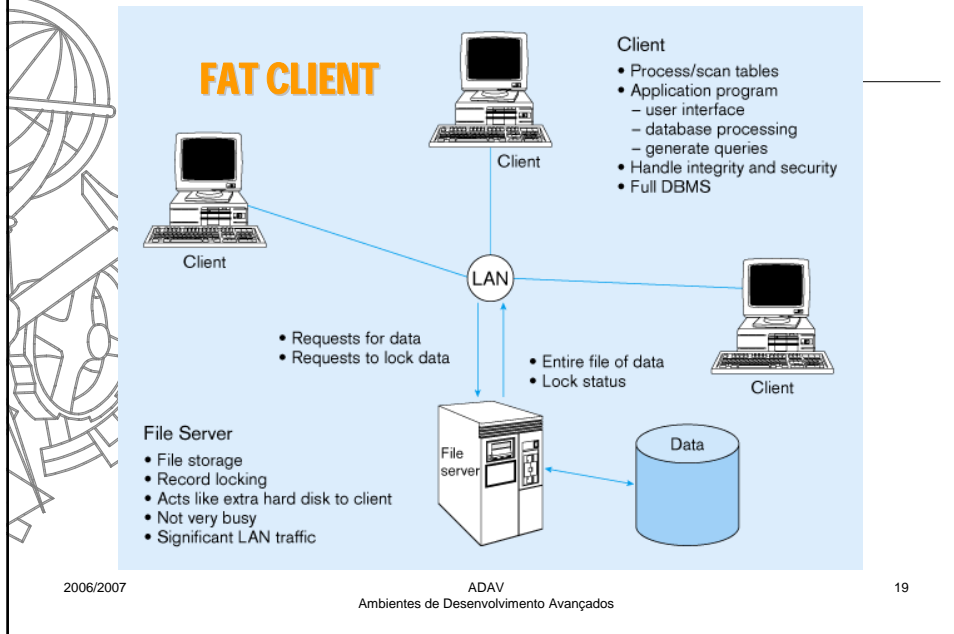
Executa no cliente e no servidor. Normalmente está dividido em stacks de transporte (TCP/IP, NetBios, IPX/SPX, SNA, ...), Sistemas Operativos de Rede ou serviços de rede (NOS = serviços de directório, segurança, RPC, mensagens, etc.) e serviços específicos de middleware (ODBC, Mail, ORB, HTTP, ...)



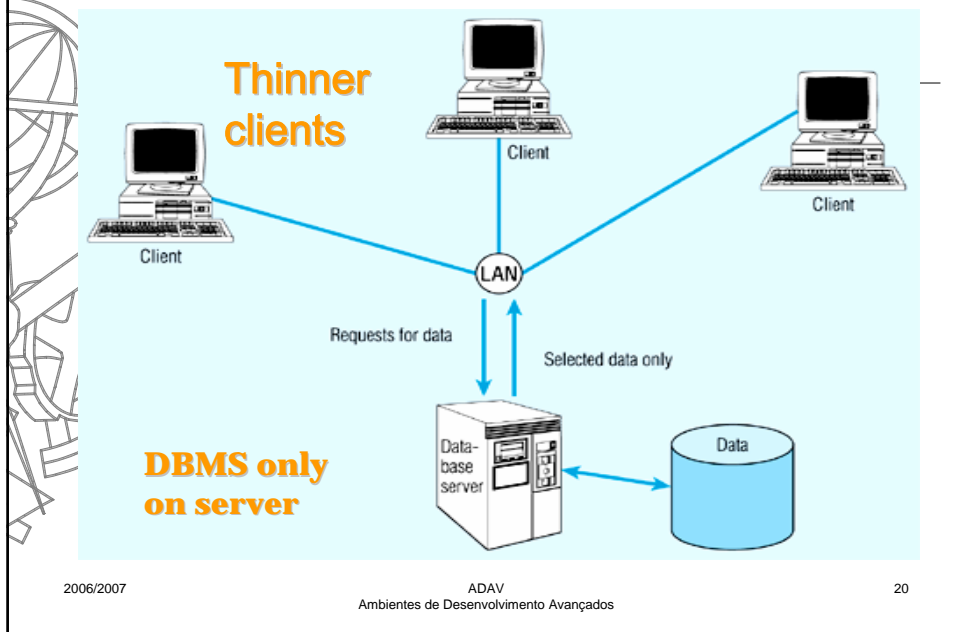
Balanceamento Cliente-Servidor

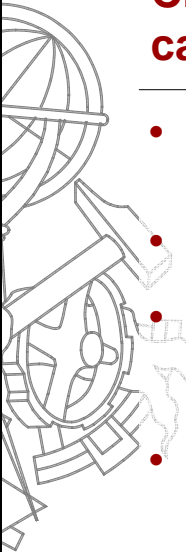
- Como distribuir uma aplicação entre cliente e servidor?
- *Groupware* e servidores de Web são servidores 'pesados'
- Servidores de ficheiros e de base de dados são exemplos de clientes 'pesados'
- Objectos distribuídos estão no meio (normalmente...)

Balanciamento cliente-servidor



Balanciamento cliente-servidor





Cliente-servidor 2 camadas vs 3 camadas (2-tier vs 3-tier)

- As aplicações podem normalmente ser divididas logicamente em interface, lógica de negócio e base de dados...
- No chamado cliente-servidor a 2 níveis a lógica de negócio está dividida entre cliente e servidor
- No cliente-servidor a 3 níveis (ou n níveis) existe um nível independente só para a lógica de negócio. Esta solução permite um desempenho e uma evolução superior.
- O exemplo por excelência desta arquitectura são as soluções de objectos distribuídos. As soluções Web também se enquadram normalmente desta categoria.

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

21



Cliente-servidor 3 camadas

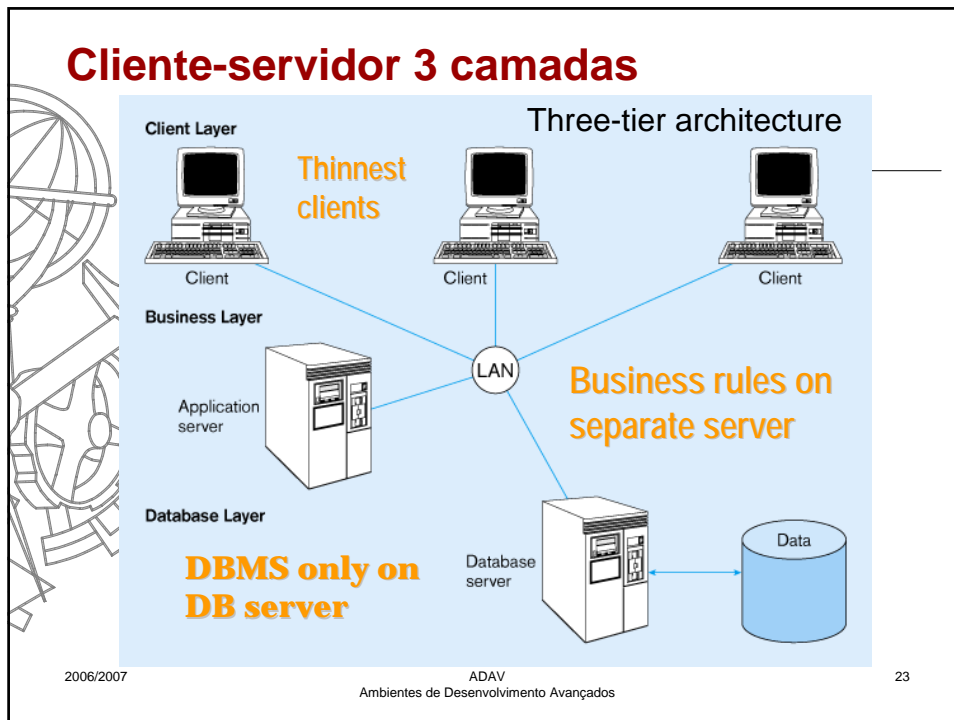
- Em geral, estes incluem outra camada do tipo servidor entre o cliente e o usual servidor;
- Este servidor adicional pode ser usado para diferentes propósitos;
- Frequentemente os programas de aplicação residem no servidor adicional (*Application Server*)
- Na maior parte dos casos os cliente apenas contém a Interface com o Utilizador e fazem um processamento mais leve. O armazenamento de dados é limitado ou não existe.

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

22

Cliente-servidor 3 camadas



Vantagens das arquitecturas de 3 camadas

- **Escalabilidade** – middle tier can be used to reduce the load on a database server by using a transaction processing (TP) monitor to reduce the number of connections to a server, and additional application servers can be added to distribute application processing
- **Flexibilidade tecnológica** – easier to change DBMS engines – middle tier can be moved to a different platform. Simplified presentation interfaces make it easier to implement new interfaces
- **Redução de custo a longo prazo** – use of off-the-shelf components or services in the middle tier can reduce costs, as can substitution of modules within an application rather than a whole application

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

24



Vantagens das arquitecturas de 3 camadas

- **Melhor adaptação dos sistemas às necessidades de negócio** – new modules can be built to support specific business needs rather than building more general, complete applications
- **Serviço de apoio aos clientes melhorado** – multiple interfaces on different clients can access the same business process
- **Vantagem competitiva** – ability to react to business changes quickly by changing small modules of code rather than entire applications

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

25



Desafios para as arquitecturas de 3 camadas

- **Elevados custo de curto prazo** – presentation component must be split from process component – this requires more programming
- **Ferramentas, treino e experiência** – currently lack of development tools and training programmes, and people experienced in the technology
- **Standards incompatíveis** – few standards yet proposed
- **Falta de ferramentas compatíveis do utilizador final** – many end-user tools such as spreadsheets and report generators do not yet work through middle-tier services

2006/2007

ADAV
Ambientes de Desenvolvimento Avançados

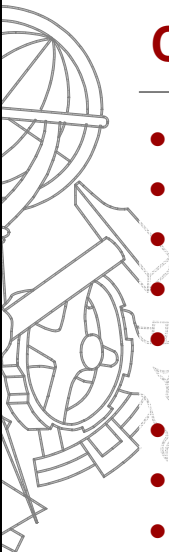
26



O que é um componente e o que não é?

Capítulo 4 de:
Szyperski, Clemens et al. *Component Software - Beyond Object-Oriented Programming*. Second Edition

2006/2007 ADAV 27
Ambientes de Desenvolvimento Avançados



Conteúdo

- Componentes
- Objectos
- Componentes e Objectos
- Módulos
- Abstração e Reutilização : *WhiteBox* vs *BlackBox*
- Interfaces
- Dependências de Contexto Explícitas
- Componentes – “Peso”

2006/2007 ADAV 28
Ambientes de Desenvolvimento Avançados




O que é um componente?

- **“A software package which offers *service through interfaces*”**
[Peter Herzum and Oliver Sims, “Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise”, John Wiley & Sons, Incorporated, 1999].
- **“A coherent package of software artifacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other components to build something larger”**
[D.F. D’Souza and A.C. Wills, “Objects, Components, And Frameworks with UML – The Catalysis Approach” Addison-Wesley, 1998].
- **“A component is a unit of composition with contractually specified interfaces and *explicit context dependencies* only. A software component can be deployed independently and is subject to composition by third parties.”**
[C. Szyperski, “Component Software: Beyond Object-Oriented Programming” Addison-Wesley, 1998].



O que não é um componente?

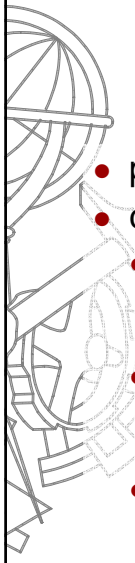
Component isn't an object, not in sense of simply being an object in a Java or C++ program, although it is true at runtime.



Programação orientada a componentes

- A sign of **maturity**
- Evolved from **Object-Oriented**
- Large scale **reuse**
- **Reconfigurable** capabilities
- **Off-the-shelf** components
- **Evolutionary** refinement
- **Economically** scaling

2006/2007 ADAV 31
Ambientes de Desenvolvimento Avançados




Termos e Conceitos

Componentes

- pre-built binary units
- characteristic properties of a component are:
 - **unit of independent deployment** – it needs to be well separated from its environment and other components;
 - **unit of a third-party composition** – it need to be sufficient self-contained and needs to come with clear specification of what it requires and provides;
 - **has no (external) observable state** – it is required that the component cannot be distinguished from copies of its own.

2006/2007 ADAV 32
Ambientes de Desenvolvimento Avançados




Termos e Conceitos

Objectos

- objects are not sold, bought or deployed, the unit of deployment is something more static such as a class or rather a library or framework of classes
- characteristic properties of an object are:
 - a unit of instantiation with a unique identity;
 - may have state and this can be externally observable;
 - encapsulates its state and behavior,
 - instances of classes or clones of *prototype* objects;
 - Initialization: *constructor* (static procedure) or *object factory* (separate object)

2006/2007 ADAV 33
Ambientes de Desenvolvimento Avançados

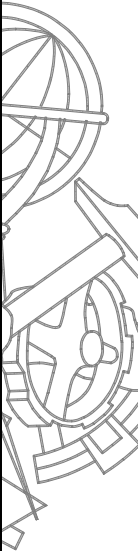


Termos e Conceitos

Componentes e Objectos

- a component is likely to act through objects – normally consists of one or more classes;
- however, there is no need for a component to contain classes only, or even contain classes at all – might contain global procedures or static variables; might be implemented in functional or assembly language;
- state maintained by an object is abstracted by that object's reference – a component that does not maintain observable state cannot (observably) maintain references even to the objects it creates;
- just as classes can depend on other classes using inheritance, components can depend on other components – the superclasses of a class do not necessarily need to reside in the same component as the class itself;

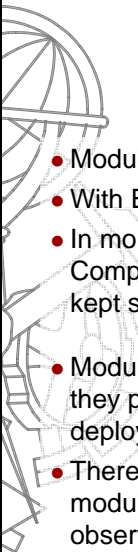
2006/2007 ADAV 34
Ambientes de Desenvolvimento Avançados



Exemplos de Componentes:

- Potential Examples:
 - Procedures ?
 - Classes ?
 - Modules ?
 - entire application ?
- Are not components:
 - C macros
 - C++ templates
 - Smalltalk blocks

2006/2007 ADAV 35
Ambientes de Desenvolvimento Avançados



Termos e Conceitos

Módulos

- Modular languages – Modula-2 (Wirth, 1982) and Ada;
- With Eiffel, it was claimed that a class is a better module (Meyer, 1988);
- In more recent languages designs – such as Oberon, Modula-3, Component Pascal and C# – the notions of modules and classes are kept separated;
- Modules as opposed to classes can be seen as **minimal components**: they package multiple classes, can be compiled separately and deployed independently;
- There are cases where modules do not qualify as components – modules can be build to use global (static) variables to expose observable state.

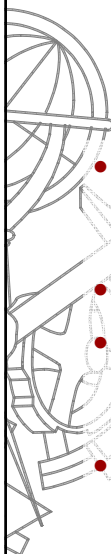
2006/2007 ADAV 36
Ambientes de Desenvolvimento Avançados



Termos e Conceitos

Abstração e Reutilização : *WhiteBox vs BlackBox*

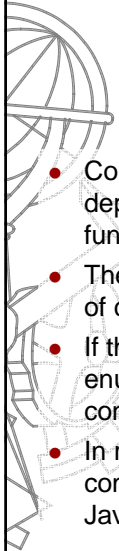
- In ideal **blackbox** abstraction a client cannot see beyond the interface, implementations are reused without building on anything else than their interface
- In **whitebox** abstraction the interface may still enforce encapsulation and limit what clients can do but the implementation is fully available and implementation inheritance allows for substantial interference
- **Glassbox** reuse allows inspection of the implementation but not interference
- **Grayboxes** are those that reveal a controlled part of their implementation



Termos e Conceitos

Interfaces

- Define component's access points – allow clients of a component, usually components themselves, to access services provided;
- Interface specifies signature and behavior;
- Normally, multiple interfaces are provided corresponding to different access points, each representing a service that component offers;
- Emphasizing the contractual nature of the interface specifications is important because the component and its clients are developed in mutual ignorance.

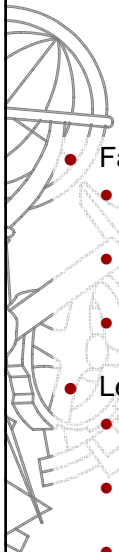


Termos e Conceitos

Dependências de Contexto Explícitas

- Components have to specify their needs – specification of what the deployment environment will need to provide so that the component can function;
- These needs are called context dependencies, referring to the context of composition and deployment;
- If there were only one software component world, it would suffice to enumerate requires interfaces of the other components to specify all context dependencies;
- In reality, there are several component worlds that partial coexist, partial compete and partially conflict with each other – OMG’s CORBA, Sun’s Java and Microsoft’s COM and CLR.

2006/2007 ADAV 39
Ambientes de Desenvolvimento Avançados



Termos e Conceitos

“Peso” de Componentes

- **Fat Components**
 - The component is self-contained and can function under weak environmental guarantees
 - The context dependencies are reduced making the component more robust over time
 - But a component with everything bundled in is not a component anymore
- **Lean Components**
 - Other components are (re)-used to achieve the component's services
 - The context dependencies increase making the component more vulnerable in case of context evolution
 - Re-use is maximized, use is compromised

2006/2007 ADAV 40
Ambientes de Desenvolvimento Avançados

