Processamento Distribuído

- Redes de Computadores
  - Software para o uso da rede
    - Compartilhamento de Arquivos e Impressoras
    - Serviços de Rede (autenticação, acesso a recursos, etc)
  - Emergência de um novo paradigma : sistemas distribuídos
- Sistemas Distribuídos
  - Sistemas Operacionais Distribuídos
    - Software do sistema é distribuído ao longo da rede
  - Aplicações Distribuídas
    - Novos métodos de programação
    - Tecnologias Básicas
      - Sockets, RPC, Objetos Distribuídos

Programação em Rede

- Suite de Programação Internet
  - IP - Internet Protocol
  - TCP - Transport Control Protocol - orientado a conexão
  - UDP - User Datagram Protocol - sem conexão
- Comunicação
  - comutação de pacotes de dados - pacotes IP
- Dispositivo de Acesso à Rede
  - Endereço IP - números de 32 bits - NNN.NNN.NNN.NNN
  - Domain Name System (DNS) - associação de nomes a endereços IP - DNS Servers
- Portas
  - endereço dentro de um computador (16 bits)
  - número associado a um tipo de serviço

Programação em Rede

- TCP - Serviço Orientado a conexão
  - estabelece uma conexão entre uma origem (porta/end.IP) e um destino (porta/end.IP) que perdura até que a mesma seja explicitamente encerrada
  - comunicação confiável
- UDP - Serviço sem Conexão
  - Não estabelece um vínculo direto entre origem e destino
  - envia datagramas sem confirmação de resposta e sem técnicas de correção de erros
  - mais rápidos que o TCP
- Unicast x Multicast
  - unicast - comunicação ponto a ponto
  - multicast - grupo de hosts recebendo um mesmo endereço IP

Programação com Sockets

- Programação Socket
  - Base para programação em rede utilizando TCP/IP
  - Apareceu como uma abstração para programação em rede dentro de sistemas UNIX, tornando-o compatível com o paradigma básico de I/O do UNIX
- Paradigma básico de I/O do UNIX
  - open-read-write-close
  - insuficiente para gerenciar todos os serviços e protocolos de rede
- Socket
  - generalização do mecanismo de acesso a arquivos do UNIX, provendo um ponto de conexão para comunicação
  - programas aplicativos solicitam a criação de um socket ao sistema operacional, quando necessário

Sockets e Java

- java.net
  - InetAddress - encapsula endereços IP e suporta conversão entre notação decimal pontuada e nomes de hosts
  - Socket, ServerSocket, DatagramSocket, MulticastSocket - implementa sockets clientes e servidores
  - SocketImpl e DatagramSocketImpl (classes) e SocketImplFactory (interface) - auxiliam a criação de sockets customizados
  - URL, URLConnection, HttpURLConnection e URLEncoder - implementam conexões Web de alto nível
  - FileNameMap (interface) - usada no mapeamento de nomes de arquivos a tipos MIME

Sockets e Java

- Classe InetAddress

boolean equals(Object obj)	Compares this object against the specified object.
byte[] getAddress()	Returns the raw IP address of this InetAddress object.
static InetAddress[] getAllByName(String host)	Determines all the IP addresses of a host, given the host's name.
String toString()	Converts this IP address to a String.
static InetAddress getByName(String host)	Determines the IP address of a host, given the host's name.
String getHostAddress()	Returns the IP address string "%d.%d.%d.%d".
String getHostName()	Returns the hostname for this address.
static InetAddress getLocalHost()	Returns the local host.
int hashCode()	Returns a hashcode for this IP address.
Boolean isMulticastAddress()	Utility routine to check if the InetAddress is a IP multicast address.

Sockets e Java

- Classe Socket

`Socket(InetAddress address, int port)`	Creates a stream socket and connects it to the specified port number at the specified IP address.
`Socket(String host, int port)`	Creates a stream socket and connects it to the specified port number on the named host.
`void close()`	Closes this socket.
`InetAddress getInetAddress()`	Returns the address to which the socket is connected.
`InputStream getInputStream()`	Returns an input stream for this socket.
`InetAddress getLocalAddress()`	Gets the local address to which the socket is bound.
`int getLocalPort()`	Returns the local port to which this socket is bound.
`OutputStream getOutputStream()`	Returns an output stream for this socket.
`int getPort()`	Returns the remote port to which this socket is connected.

Sockets e Java

- Classe ServerSocket

`ServerSocket(int port)`	Creates a server socket on a specified port.
`ServerSocket(int port, int backlog)`	Creates a server socket and binds it to the specified local port number.
`ServerSocket(int port, int backlog, InetAddress bindAddr)`	Create a server with the specified port, listen backlog, and local IP address to bind to.
`Socket accept()`	Listens for a connection to be made to this socket and accepts it.
`void close()`	Closes this socket.
`InetAddress getInetAddress()`	Returns the local address of this server socket.
`int getLocalPort()`	Returns the port on which this socket is listening.
`Protected void implAccept(Socket s)`	Subclasses of ServerSocket use this method to override accept() to return their own subclass of socket.
`String toString()`	Returns the implementation address and implementation port of this socket as a String.

Sockets e Java

- Streams
  - Sequências de bytes que se movimentam de uma origem a um destino
- java.io
  - InputStream
    - FilterInputStream, BufferedInputStream, DataInputStream, LineNumberInputStream, PushBackInputStream, ...
  - OutputStream
    - FilterOutputStream, BufferedOutputStream, DataOutputStream, PrintStream, ByteArrayOutputStream, FileOutputStream, ...
  - Reader
    - BufferedReader, CharArrayReader, FilterReader, InputStreamReader, ...
  - Writer
    - BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, ...

Sockets e Java

Cliente

```
…
Socket cl = new Socket(destination,port);
…
BufferedReader inS = new BufferedReader(
cl.getInputStream())
DataOutputStream outS = new DataOutputStream(
cl.getOutputStream());
…
/* use inS and outS */
```

Servidor

```
…
ServerSocket s = new ServerSocket(PORT);
Socket cl = s. accept();
…
String dnm = cl.getInetAddress().getHostName();
int dpt = cl.getPort();
…
BufferedReader inS = new BufferedReader( new
    InputStreamReader (cl.getInputStream()));
DataOutputStream outS = new DataOutputStream(
    cl.getOutputStream());
…
/* use inS and outS  */
```

Programação RPC

- Programação Socket
  - apesar de conveniente, apresenta um problema incurável:
    - paradigma input/output
  - Implementar toda a programação de rede em cima de I/O tem uma série de inconveniências
- RPC - Remote Procedure Call
  - um paper de Birrel and Nelson (1984) introduziu uma maneira completamente diferente de atacar o problema
  - permitir programas chamar rotinas localizadas em outras máquinas
  - quando o processo na máquina A chama uma rotina na máquina B, o processo em A é suspenso, e a execução da rotina chamada tem início em B. A informação que vai de A para B se dá por meio de parâmetros retornando como o resultado da chamada

Programação RPC

- Transparência na Programação
  - Do mesmo modo que uma chamada de sistema, uma chamada remota de procedimento (RPC) deve ser transparente para o programador
  - Stub do Cliente: empacota os parâmetros e envia ao servidor. Depois disso, o stub fica esperando uma mensagem de retorno do servidor, bloqueando o socket. Uma vez que o servidor retorna, ele desempacota a mensagem do servidor, copia o resultado para a rotina que fez a chamada do RPC e retorna de modo usual
  - Stub do Servidor: quando a mensagem do stub do cliente chega ao servidor, o kernel a passa para o stub do servidor, que tipicamente está com um socket bloqueado esperando por mensagens. O stub desempacota os parâmetros, chama a rotina desejada e a seguir re-empacota o resultado da chamada e envia a resposta ao stub do cliente. Em seguida retorna ao loop de espera por nova requisição

Programação RPC

- Transparência na Programação
  - O programa não tem a menor idéia que o trabalho está sendo feito remotamente
  - Todos os detalhes de passagem de mensagens ficam escondidos dentro dos dois stubs
- Gerando stubs
  - Se a troca de mensagens é feita manualmente, erros obscuros podem acontecer
  - Portanto, a geração de stubs deve ser feita automaticamente
  - Um compilador de stubs deve ser utilizado, portanto para gerar os stubs do cliente e do servidor
  - e.g. - SunRPC: arquivo .x - linguagem RPC, um programa - rpcgen gera os arquivos de stubs (em linguagem C)
  - diversos mecanismos de RPC:DCE RPC, Microsoft RPC, ONC RPC, etc

Modelo Cliente-Servidor

- Aplicações convencionais do tipo Cliente-Servidor
  - apenas transformam aplicações monolíticas separando-as em duas metades
- Futuro do Modelo Cliente-Servidor
  - extrapolar de aplicações restritas a redes locais para redes de alcance global
- Como ?
  - Usando uma arquitetura baseada em componentes
  - Objetos Distribuídos são a chave para esta revolução
  - Aplicações envolvendo objetos distribuídos são muito adequadas para a criação de sistemas cliente/servidor mais flexíveis, pois tanto dados quanto a lógica da aplicação estão encapsulados dentro dos objetos, permitindo sua localização em qualquer lugar de um sistema distribuído

Objetos Distribuídos

- Objetos Clássicos
  - unidade de serviço encapsulando código e dados
  - provê diversas facilidades para reutilização de código
  - existe somente dentro de um programa isolado
- Objetos Distribuídos
  - unidade de serviço localizada em qualquer lugar de uma rede
  - armazenados como pedaços independentes de código que podem ser acessados por clientes remotos via invocação de métodos
  - linguagens de programação e compiladores envolvidos são totalmente transparentes para seus clientes
  - clientes não necessitam saber onde se localizam os objetos ou que sistema operacional está sendo executado remotamente
  - pedaços inteligentes de software que podem trocar mensagens entre si de modo transparente em qualquer lugar do mundo

Objetos e Componentes

- Componentes
  - pedaços isolados de software que podem se interconectar automaticamente via redes, aplicações, linguagens, ferramentas e sistemas operacionais
- Objetos Distribuídos
  - são por definição, componentes
  - nem todo componente é um objeto e nem todos os componentes são distribuídos
- Tecnologia de Componentes
  - promete alterar radicalmente a maneira como se desenvolve software
  - e.g. objetos distribuídos permitem a criação de sistemas cliente-servidor sofisticados simplesmente interconectando-se componentes
  - meta é atingir o nível de interoperabilidade encontrado por exemplo em circuitos integrados

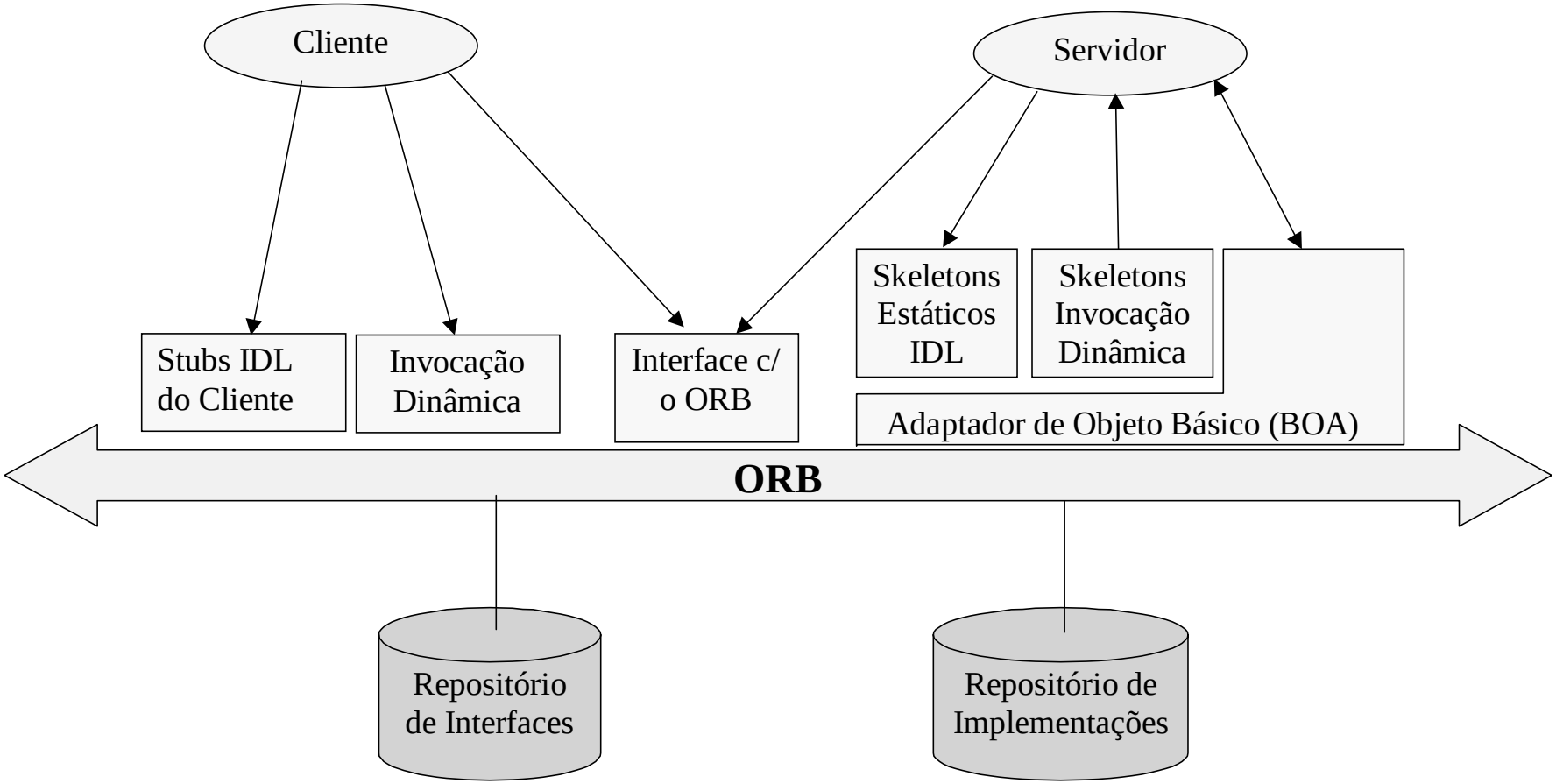
O que exatamente é um Componente ?

- É algo que se pode comprar e vender
- Não é uma aplicação completa
- É algo que pode ser combinado a outros componentes, de maneiras imprevisíveis
- Possui uma interface muito bem especificada, que define sua operação
- É um "objeto inter-operável"
  - termo cunhado por Ray Valdes em 1995 significando uma entidade de software que seja independente do sistema operacional utilizado
- É um "objeto extendido"
  - deve agir como um objeto suportando encapsulamente, herança e polimorfismo

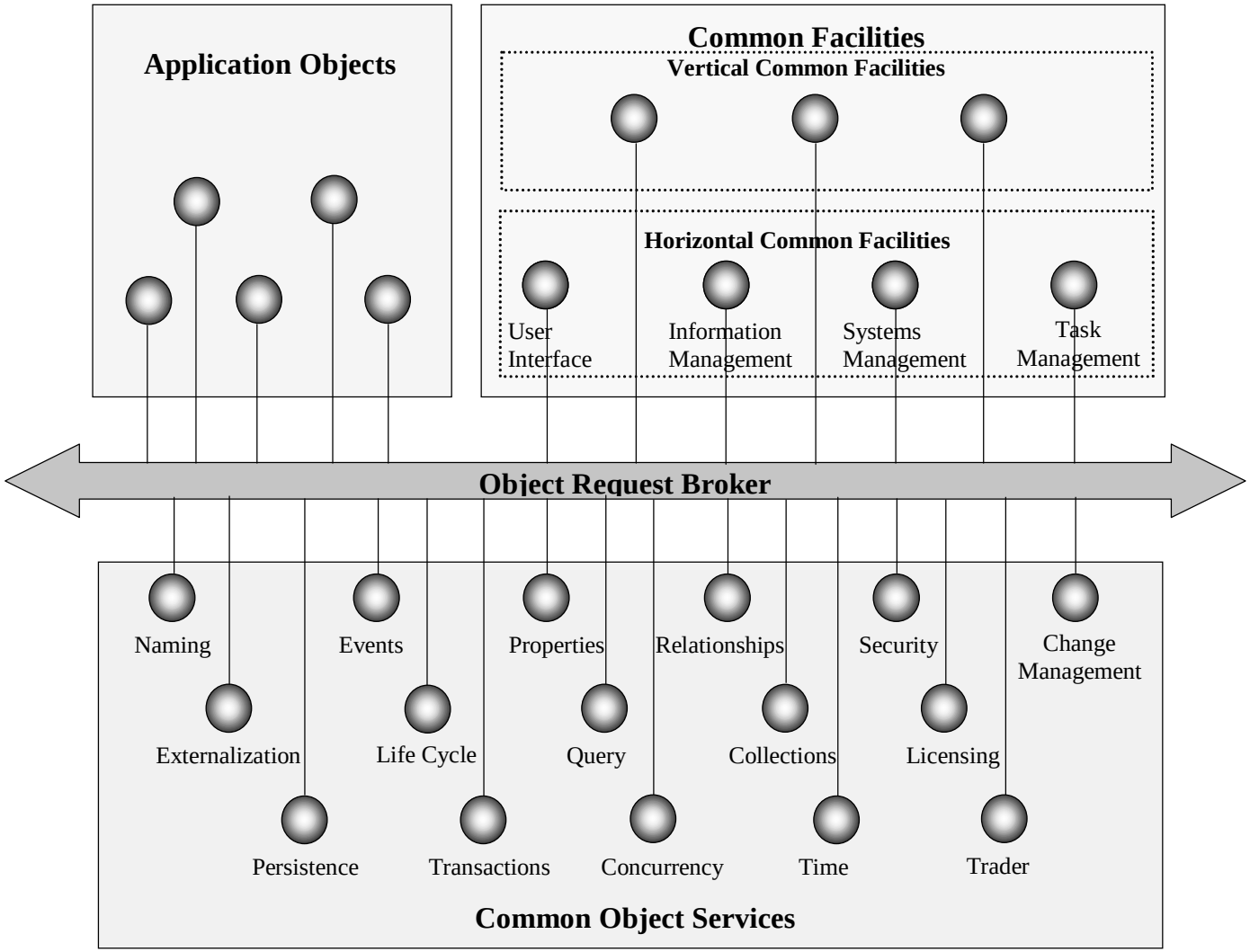
CORBA - Common Object Request Broker Architecture

- Norma da OMG definindo a operação, gerenciamento, serviços e facilidades para objetos distribuídos
- Características
  - arquitetura completamente orientada a objetos
  - separa interface de serviços de sua implementação
  - provê um conjunto padrão de serviços e facilidades
  - integra múltiplas linguagens de programação
  - permite sua inserção dentro de um sistema operacional
  - independente do sistema operacional
  - padrão aberto adotado por múltiplas empresas de software
  - permite a interoperabilidade entre produtos de diferentes empresas

Arquitetura CORBA



Facilidades e Serviços CORBA



Java e Objetos Distribuídos

- java.rmi
  - Pacote RMI - Remote Method Invocation
  - Implementação nativa do Java para objetos distribuídos
  - exige que os objetos distribuídos sejam objetos Java
- org.omg.CORBA
  - Implementação Java de um ORB CORBA
  - permite objetos de múltiplas linguagens
  - é computacionalmente mais custoso do que o RMI
- Sockets x RMI x CORBA
  - caso a comunicação seja simples, deve-se utilizar Sockets
  - caso se deseje abstrair a passagem de mensagens, e o sistema distribuído utilizará somente Java, deve-se utilizar RMI
  - caso seja necessário (acessar objetos desenvolvidos/ oferecer objetos para programas) em outras linguagens, utilizar CORBA

Java RMI

- Registry do RMI
  - rmiregistry, rodando na máquina do servidor
- Objeto Servidor
  - instalar um RMISecurityManager
  - definir interface que extende a interface Remote
  - implementar esta interface
  - cadastrar nome utilizando Naming.rebind()
- Geração de Stubs e Skeletons
  - rmic
- Objeto Cliente
  - instala um RMISecurityManager
  - obtém stub do servidor via Naming.lookup()
  - Acessa normalmente os métodos do servidor

Java CORBA

- Arquivo IDL - Nome.idl
- idltojava
  - gera os seguintes arquivos:
    - _NomeImplBase.java - skeleton do servidor. É extendida por NomeImplBase no programa servidor
    - _NomeStub.java - stub do cliente - implementa Nome.java
    - Nome.java - interface correspondente ao arquivo IDL
    - NomeHelper.java - funcionalidades auxiliares
    - NomeHolder.java - possui implementação da classe Nome

Exemplo de Programa Servidor

```
// Create and initialize the ORB
      ORB orb = ORB.init(args, null);
// Create the servant and register it with the ORB
      HelloServant helloRef = new HelloServant();
      orb.connect(helloRef);
// Get the root naming context
      org.omg.CORBA.Object objRef =
           orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);
// Bind the object reference in naming
      NameComponent nc = new NameComponent("Hello", "");
      NameComponent path[] = {nc};
      ncRef.rebind(path, helloRef);
// Wait for invocations from clients
      java.lang.Object sync = new java.lang.Object();
      synchronized(sync){
        sync.wait();
      }
```

Exemplo de Programa Cliente

```
// Create and initialize the ORB
     ORB orb = ORB.init(args, null);

// Get the root naming context
     org.omg.CORBA.Object objRef =
           orb.resolve_initial_references("NameService");
     NamingContext ncRef = NamingContextHelper.narrow(objRef);

// Resolve the object reference in naming
     NameComponent nc = new NameComponent("Hello", "");
     NameComponent path[] = {nc};
     Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

// Call the Hello server object and print results
     String Hello = helloRef.sayHello();
     System.out.println(Hello);
```