1) O que é CORBA?
CORBA (*common object request broker architecture*) é especificação padronizada de um OMA (*object management architecture*) com o objetivo de interoperabilidade entre diferentes sistemas computacionais e linguagens de programação através de um ORB definido. Tal padrão é definido pela OMG.

2) O que é OMG e quem participa?
OMG significa *Object Management Group*, que é um consórcio com a intenção de definir padrões para sistemas distribuídos orientados a objetos. Participam as grandes:

- Adobe Systems Inc.
- AT&T
- Boeing
- Borland Software Corporation
- Department of National Defence
- Ericsson
- Fujitsu
- GNOME Foundation
- Hewlett-Packard
- Hitachi
- Massachusetts Institute of Technology (MIT)
- Motorola
- NASA
- Nokia
- Oracle
- Sun Microsystems
- Alcatel-Lucent
- Telefonica I+D
- Toshiba
- W3 Consortium
- Walt Disney World Co.
- Entre outros.

Ou seja, é um padrão largamente aceito, difundido e organizado.

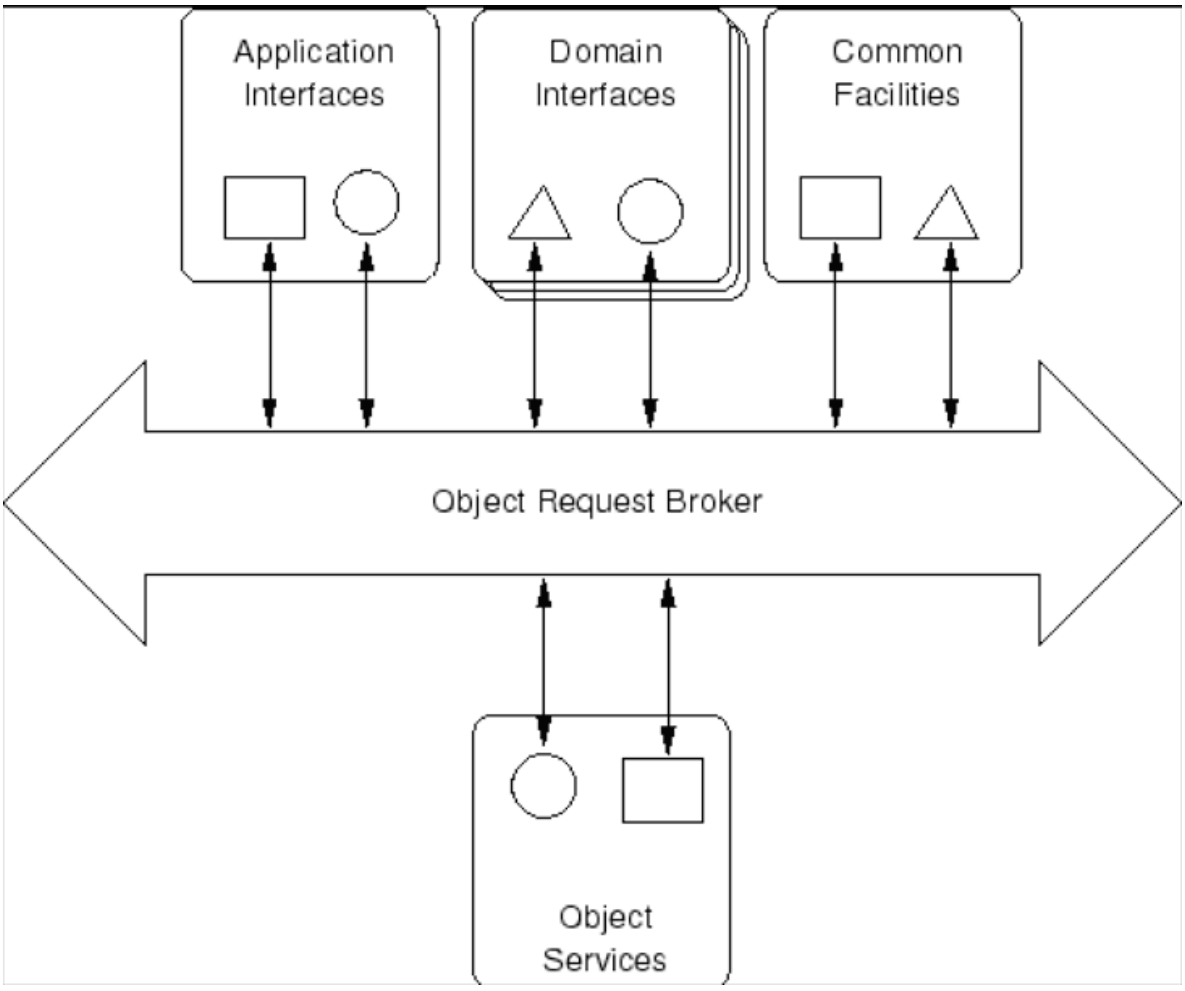
3) Uma das principais filosofias do padrão CORBA é tentar eliminar os problemas típicos em sistemas heterogêneos: a dependência de aspectos incompatíveis entre plataformas. Cite 3 independências propostas pelo padrão, e o que significam:

- Independência de linguagem: pode funcionar como uma "cola" entre linguagens distintas.
- Independência de implementação: o cliente não precisa saber como o serviço é implementado, há um baixo acoplamento entre os módulos.
- Independência de arquitetura: abstração de características do *hardware*, como ordenação de bytes e alinhamento de dados.
- Independência de sistema operacional: clientes e servidores podem ser implementados sistemas operacionais distintos (inclusive, CORBA suporta mais plataformas Microsoft que DCOM e .NET).

- Independência de protocolo/transporte: o ORB decide dinamicamente que protocolo e transporte usar, eliminando a necessidade de código de manipulação de rede nos clientes e servidores.

4) O que é a OMA?

OMA (*object management architecture*) é a estrutura que define a arquitetura sob a qual os objetos locais e remotos irão comunicar-se. A estrutura básica de uma OMA é como a da figura que segue.



5) O que é um ORB?

Em computação distribuída, um ORB (*object request broker*) é uma peça de *middleware* que permite que os programadores façam chamadas de um computador a outro via uma rede. ORB é o componente-chave da OMA (*object management architecture*), sendo muitas vezes chamado de barramento de objetos pela sua funcionalidade de trafegar os dados dos objetos do ambiente local para o remoto e vice versa.

A principal função de um ORB é a transformação das estruturas de dados envolvidas nas chamadas aos objetos remotos no *stream* de bytes que será trafegado pela rede, utilizando o protocolo GIOP.

6) O que significa GIOP?

GIOP (*general interORB protocol*) é um protocolo abstrato de comunicação utilizado entre os ORB's. Os padrões associados a este protocolo são mantidos pela OMG. O GIOP disponibiliza alguns protocolos concretos:

- IIOP (*Internet interORB protocol*): é o protocolo de comunicação entre ORB's CORBA publicado pela OMG, sendo uma implementação do GIOP para ser utilizado na Internet (sobre TCP/IP).
- SSLIOP (*SSL interORB protocol*): é uma implementação do IIOP sobre SSL, permitindo criptografia dos dados e autenticação.
- HTIOP (*HyperText interORB protocol*): é uma implementação do IIOP sobre HTTP, permitindo transparência sob o ponto de vista de *firewalls*.

7) Quais os três principais conceitos do *middleware* (ORB) do CORBA?

- Definição de Interface: em programação orientada a objetos é fundamental que o cliente tenha conhecimento da interface pública do objeto ao qual ele deseja se comunicar para que tal comunicação seja possível através de chamadas aos métodos deste objeto com os parâmetros apropriados. Em CORBA, as interfaces são definidas através de uma linguagem de definição de interfaces conhecida como IDL (*interface definition language*), que podem, então, serem armazenadas em um repositório de interfaces.
- Endereçamento: o cliente precisa, de alguma forma, saber o endereço para o qual deve-se mandar as requisições ao objeto desejado. Este endereço deve conter informações suficientes para identificar o serviço particular (peça de código) para o qual o cliente quer se comunicar. CORBA utiliza referências de objetos como forma de endereçamento.
- Invocação (ou chamada): o cliente precisa ser capaz de construir uma requisição que seja, então, transportada ao servidor, atendida e retornada. Tal atividade pode ser realizada através do uso de objetos *stubs* que são gerados a partir da definição da interface (em IDL). Chamadas aos objetos *stubs* resultam no empacotamento dos parâmetros em uma requisição (*request*) que é enviada de forma transparente para a implementação do objeto localizada no servidor. Uma outra opção é a utilização da *Dynamic Invocation Interface* para criar requisições em tempo de execução, possivelmente utilizando informações de tipo obtidas em tempo de execução do repositório de interfaces. Ambos os meios descritos disponibilizam a abstração necessária para sincronização e codificação.

8) O que é a IDL?

IDL (*interface definition language*) é a linguagem desenvolvida para definir as interfaces. A partir de uma definição criada em IDL, o compilar IDL gera o *stub* e o *skeleton*, sendo o

primeiro utilizado para invocar os objetos remotos e o segundo utilizado como esqueleto para a implementação do objeto.

9) Quais são os passos necessários para uma invocação dinâmica?

É importante salientar que é responsabilidade do cliente especificar os tipos de parâmetros e os resultados esperados. Há quatro passos básicos, feitos pelo ORB, e são os seguintes:

1. Identificar o objeto que se quer invocar
2. Recuperar a interface desse objeto, buscando-a no Repositório de Interfaces
3. Construir a invocação
4. Invocar a requisição e receber os resultados

10) O que é DSI e qual a diferença entre essa interface e DII?

DSI (*dynamic skeleton interface*) é a Interface de Esqueleto Dinâmica. Ela permite que uma implementação de objetos seja alcançada através de uma interface que provê acesso aos nomes das operações e parâmetros de maneira análoga à interface de Invocação Dinâmica (DII). A diferença básica entre as duas é que DII é uma interface do lado do cliente, enquanto DSI é para a implementação de objetos.

11) Na prática, como é que posso desenvolver uma aplicação usando objetos distribuídos com CORBA?

Em primeiro lugar, é preciso ter instalado um ambiente de desenvolvimento, onde as principais peças são o compilador IDL para alguma linguagem e o servidor ORB. Existem diversos pacotes de ferramentas para desenvolvimento de aplicações CORBA, conforme será visto a seguir. Após esse pré-requisito cumprido, pode-se descrever o desenvolvimento de um programa simples através dos seguintes passos:

- Passo 1: Escrever a interface dos objetos através da linguagem de descrição IDL. Essa interface irá apenas definir quais serão os campos e métodos dos objetos, porém a implementação destes é feita em uma linguagem de programação comum, geralmente orientada objetos. Entre as linguagens regulamentadas pela OMG citadas na bibliografia estão C, C++, Java, Ada, COBOL, Smalltalk, Objective C, e Lisp. Além disso, no site oficial da OMG encontra-se a especificação dos mapeamentos de linguagem para CORBA Scripting Language, Python, Pl/1, MOF 2.0 e XML. Uma rápida busca na web verificou também que estão sendo desenvolvidas ferramentas para Perl, Ruby, Pascal, PHP, TCL, Haskell, Visual Basic, entre outras.

O exemplo abaixo mostra a definição em IDL das interfaces de um objeto que provê informações de tempo:

Arquivo: Time.idl

```
struct Instante {
       short hora, minuto, segundo;
};
interface Relogio {
       Instante get_instante();
```

```
};
```

- Passo 2: Uma vez definidas as interfaces em IDL, elas devem ser compiladas (ou mapeadas) para a linguagem de programação alvo, gerando uma coleção de módulos, entre eles os *stubs* e os *skeletons*. Os *stubs* são responsáveis pelo empacotamento (*marshaling*) de parâmetros, enquanto os *skeletons* devem ser editados para conter a implementação dos objetos. Dependendo do tipo de linguagem, a implementação é feita de maneiras diferentes. Geralmente, em linguagens OO, os protótipos dos métodos estão no *skeleton*, e cabe ao desenvolvedor do servidor criar os corpos dos métodos. Dependendo da versão do CORBA, uma camada adicional (POA) é criada pelo compilador, e deve ser estendida para que se implemente a funcionalidade dos objetos.

No exemplo acima, após a compilação (para a linguagem C++), foram gerados os cabeçalhos (*headers*) para que o cliente conhecesse a interface dos objetos (Time.hh) e também os *skeletons* para que o servidor implementasse os métodos definidos na interface (POA_Relogio). O código abaixo demonstra a implementação da classe Relogio, com o método get_instante().

```
#include <ctime>
#include "Time.hh"

using namespace std;

class Relogio_impl : public virtual POA_Relogio {
public:
       virtual Instante get_instante() throw (SystemException)
       {
               time_t now = time(0);
               struct tm * tp = gmtime(&now);

               Instante i;
               i.hora = tp->tm_hour;
               i.minuto = tp->tm_min;
               i.segundo = tp->tm_sec;
               return i;
       }
};
```

- Passo 3: Uma vez criados os objetos da aplicação, é preciso desenvolver uma aplicação cliente e outra servidor. O servidor é um programa que inicia o ORB, registra os objetos criados e então espera por uma requisição de clientes. Para processar as requisições, o mecanismo é semelhante ao de *callbacks*. Uma aplicação cliente registra-se também junto ao ORB e então realiza as requisições ao servidor. No caso do cliente, ele não sabe que está fazendo uma requisição ao servidor, pois ele simplesmente chama um método que será redirecionado pelo *stub* para o servidor.

Seguindo o exemplo, o código do servidor seria da seguinte forma:

```
//Arquivo: server.cc
```

```
#include <iostream>
#include "Time.hh"

using namespace std;
using namespace CORBA;
using namespace PortableServer;

int main(int argc, char **argv)
{
       // Inicialização típica: ORB, POA, ativação...
       ORB_var orb = ORB_init(argc, argv);
       Object_var obj = orb->resolve_initial_references("RootPOA");
       POA_var poa = POA::_narrow(obj);
       POAManager_var mgr = poa->the_POAManager();
       mgr->activate();

       // cria um objeto Relogio
       Relogio_impl relogio;
       // obtém uma referência para imprimir no console
       Relogio_var ref = relogio._this();
       cout << orb->object_to_string(ref) << endl;

       orb->run();
}
```

A aplicação cliente, por fim, executa as chamadas aos objetos remotos. No exemplo, o cliente recebe a referência do objeto criado pelo servidor. Numa aplicação real, a passagem dessa referência deve ser feita através da rede.

```
//Arquivo: client.cc
#include <iostream>
#include "Time.hh"

using namespace std;
using namespace CORBA;

// envie como argumento a string da referência informada pelo servidor
int main(int argc, char **argv)
{
       ORB_var orb = ORB_init(argc, argv);
       // obtém uma referência a partir da string
       Object_var obj = orb->string_to_object(argv[1]);
       // faz um upcast de Object para Relogio
       Relogio_var rel = Relogio::_narrow(obj);
       // chama o método
       Instante ins = rel->get_instante();

       cout << "Horario UTC: " <<
               ins.hora << ":" <<
               ins.minuto << ":" <<
               ins.segundo << endl;
}
```

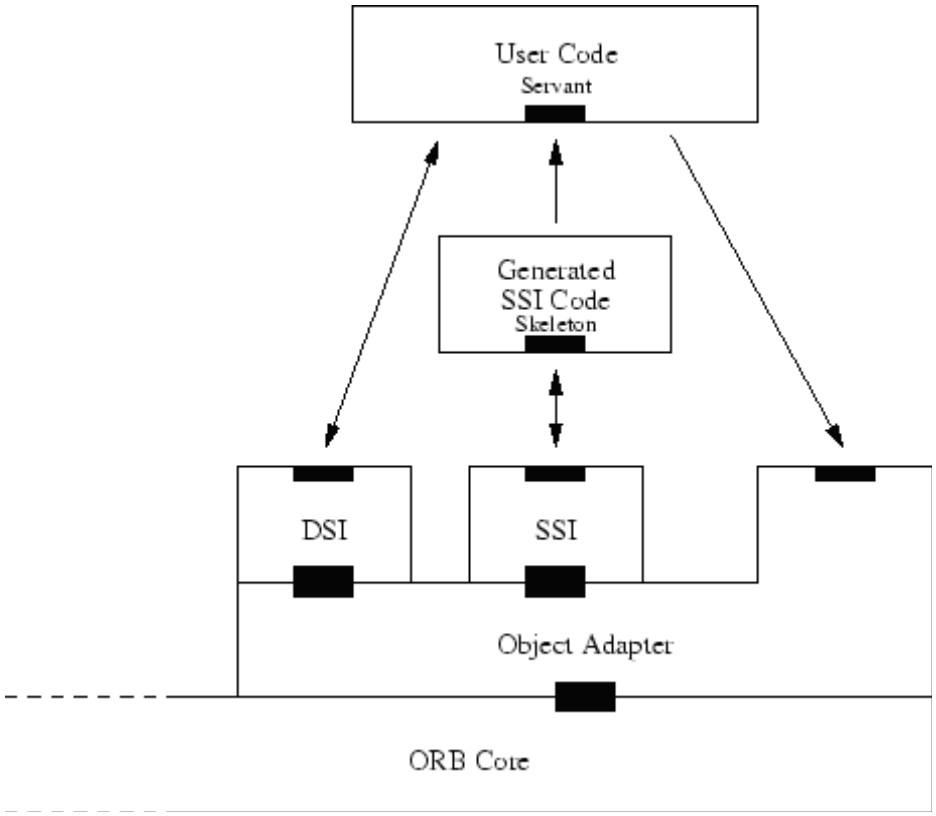
- Passo 4: Na execução, uma vez iniciado o servidor, ele passa a se comportar como um *daemon*, ou seja, um processo que fica bloqueado à espera de requisições.

Conforme as requisições chegam, ele ativa os objetos necessários e responde aos clientes o que foi pedido. Dessa forma, temos um programa simples que utiliza CORBA para implementar objetos distribuídos.

Diversos tutoriais para iniciantes se encontram disponíveis na web. O tutorial da Sun, utilizando Java como linguagem e o suporte nativo de Java como ferramenta, também foi estudado para esse trabalho e é recomendado:
http://java.sun.com/developer/onlineTraining/corba/corba.html

12) Qual a função de um OA?
Um OA (*object adapter*) serve como mediador entre o ORB e o serviço (*servant*), disponibilizando ao ORB uma interface consistente para interagir com o código do usuário presente no serviço (*servant*), sendo flexível em cooperar com o serviço. Diferentes *object adapters* podem ser disponibilizados para lidar com os vários requerimentos dos servidores e, então, a implementação mais apropriada deve ser escolhida.
Obs.: BOA e POA são exemplos de OA's introduzidos em diferentes versões do CORBA.

13) O que é o BOA?
O BOA (*basic object adapter*) é o OA (*object adapter*) introduzido na primeira versão do CORBA. Seus propósitos eram simplistas, sendo este um *object adapter* genérico para ser usado para propósitos simples. O BOA é como no seguinte esqueleto.



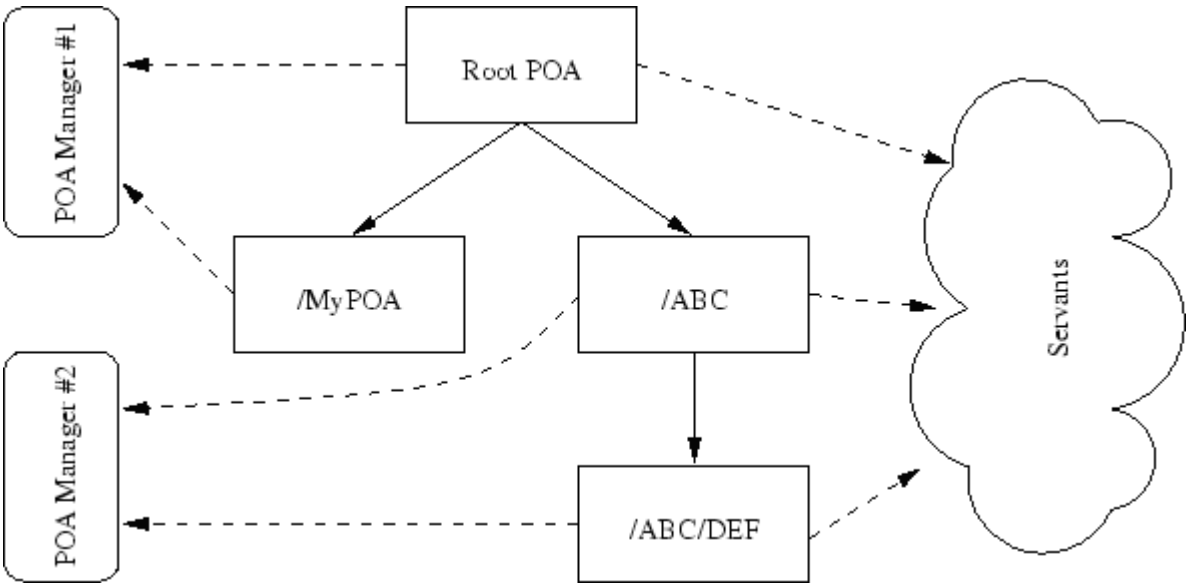
14) O que é o POA? Quais suas vantagens sobre o BOA?

O POA (*portable object adapter*) é um OA (*object adapter*) criado para ser mais portável, suprindo as limitações que existiam devido a simplicidade do BOA. Foi introduzido no CORBA 2.0, sendo definido para ser universal, um perfeito substituto para o BOA.

Vantagens do POA sobre o POA:

- Portabilidade: é possível utilizar código fonte com ORB's de diferentes desenvolvedores sem necessidade de modificação do código.
- Flexibilidade: ao prover o controle do ciclo de vida dos serviços (*servants*) e ao possuir a capacidade de sempre estar pronto para requisições, o POA pode ser usável até em servidores com requisitos exóticos, existindo padrões que permitem a programação de servidores simples da maneira mais fácil possível.

O POA tem alguns conceitos diferenciados. Ao invés de ser um componente *singleton* como o BOA (i.e. só existe uma única instância do BOA no servidor), podem existir diversas instâncias do POA, organizadas em uma hierarquia (a raiz desta hierarquia é criada pelo ORB) como no exemplo a seguir.



15) Porque utilizar o POS?

O serviço de persistência de objetos permite que um objeto sobreviva ao término da aplicação que o criou ou do cliente que o utilizou.

O ORB possui a habilidade de manter a referência a um objeto de forma persistente mas isso não garante que um determinado objeto esteja disponível apenas porque a sua referência ainda é válida.

16) Quais são as etapas do POS?

1. A cada objeto é associado um indicador para saber se é persistente.
2. O gerenciador de objetos persistentes (POM) escolhe quais serviços vão armazenar os objetos.

3. São esses serviços os responsáveis pelo armazenamento dos objetos dentro das bases de dados.

17) Qual é o papel do POM na arquitetura do POS?
O POM provê uma interface uniforme para a implementação das operações de persistência de um objeto, independente do PDS utilizado ou seja, devido à existência do POM e de um protocolo de comunicação entre PO e PDS, é possível trocar arbitrariamente o PDS que acessará o *datastore*, de forma "*plug-and-play*". Isto ocorre porque, ao receber uma requisição de operação de persistência, o POM transfere a requisição para um PDS capaz de suportar a combinação protocolo/*datastore* necessária para o objeto. Para que isso seja possível, o POM precisa saber quais são os PDSs disponíveis e quais combinações protocolo/*datastore* cada um deles suporta.

18) Qual é a função do PDS (*Persistent Data Services*)?
O PDS é o componente responsável por interagir com o objeto (PO) para extrair e inserir dados nele, através de um protocolo e, também, por interagir com o *datastore* para inserir e extrair dados dele.

19) O que é um IR (*interface repository*) e porque é necessário?
IR, ou repositório de interfaces, é uma base de dados em tempo de execução que contém as especificações de interfaces de cada objeto que o ORB reconhece.
É necessária a sua existência porque um ORB precisa saber a definição dos objetos com os quais ele está trabalhando.
Um modo para obter estas definições é incorporar a informação dentro de *stubs*, sendo outro modo aquele em que a informação é acessível dinâmicamente através do IR.

20) Qual a principal vantagem do CORBA em relação a outras tecnologias de objetos distribuídos como DCOM, Java/RMI e .NET Remoting?
A principal vantagem do CORBA em relação a soluções proprietárias como DCOM e .NET Remoting é que estas últimas não permitem a interoperabilidade de diferentes máquinas e plataformas, dificultando sua aceitação como padrão em ambientes não-Microsoft. A principal vantagem do CORBA sobre Java/RMI é o suporte a diversas linguagens de programação.

21) Cite algumas semelhanças e diferenças entre RPC e CORBA.
RPC (*remote procedure call*) é uma tecnologia que podemos pensar como sendo uma evolução da programação com *sockets*, permitindo a execução remota de procedimentos. RPC é um mecanismo de comunicação fundamental onde existem apenas chamadas a procedimentos. Pode-se utilizar um serviço de diretórios para conectar-se ao servidor desejado, porém RPC não implementa nenhuma tecnologia que permita um endereçamento mais flexível e transparente (em contraponto, CORBA, através do ORB, permite o registro de objetos remotos e locais e assim permite uma transparência na localização dos objetos

através das suas referências). Os *stubs* do cliente e do servidor se comunicam diretamente, sem uma entidade intermediária para resolução de referências que de fato não existem em RPC (CORBA tem um ORB que realiza a resolução das referências). Tipos de dados como uma lista encadeada não podem ser passados como parâmetros por RPC, pois esse não suporta referências. Todo tipo de dado deve ser serializado antes da comunicação, havendo uma perda no poder de expressão desta tecnologia (CORBA permite o uso de referências remotas e locais de forma totalmente transparente).

22) Compare CORBA com outras tecnologias.

Web Services

CORBA é um padrão desenvolvido para permitir a criação de aplicações com objetos distribuídos. Os Web services é um padrão desenvolvido para disponibilizar pequenos serviços no formato de procedimentos remotos. Seria mais fácil comparar os Web services com RPC devido às suas semelhanças.

Uma vantagem dos Web services é a transparência do ponto de vista do protocolo de comunicação entre o cliente e o servidor (que provê o serviço). Web services utiliza um protocolo chamado SOAP, baseado em XML, de fácil leitura e fácil programação. Tal protocolo é transportado sobre o protocolo HTTP, sendo, portanto, transparente do ponto de vista dos *firewalls*.

CORBA é um padrão mais poderoso devido à capacidade de permitir objetos distribuídos, e não apenas procedimentos distribuídos. O protocolo de comunicação utilizado é o GIOP, que tem diversas implementações como IIOP (GIOP sobre TCP/IP) e HTIOP (GIOP sobre HTTP), sendo este último também transparente aos *firewalls* como o protocolo SOAP/HTTP.

Java RMI

A diferença que mais se destaca entre RMI e CORBA é o fato de que, enquanto RMI é uma implementação específica, CORBA é um padrão genérico. Dessa forma, RMI é totalmente baseado na tecnologia Java, enquanto existem diversas implementações de CORBA em diversas linguagens diferentes. No caso específico, portanto, de uma implementação Java de CORBA e do RMI, pode-se apontar algumas diferenças específicas. O protocolo de comunicação de ambas, por exemplo, é diferente: CORBA utiliza o IIOP, enquanto RMI usa o JRMP. As referências a objetos, no caso do CORBA, são resolvidas pelos *Object Adapters* e pelo ORB. No caso do RMI, a própria máquina virtual Java é responsável por essa tarefa. Experiências demonstram que o desempenho de sistemas CORBA é maior que os baseados em RMI. No caso do CORBA, a modularidade é maior, uma vez que a definição da interface é separada da implementação. Dessa forma, uma mesma interface pode gerar mais de uma implementação. No caso do RMI, interface e implementação estão embutidas no código Java. A curva de aprendizado de ambas as tecnologias é praticamente a mesma, com a diferença de que programadores que não estejam habituados com Java devem primeiro aprender essa linguagem para então usar RMI.

DCOM

Uma das vantagens que o DCOM leva na comparação é o fato de que, assim como CORBA, DCOM é uma especificação implementada em diversas linguagens. Uma vantagem do DCOM é o fato de que as ferramentas de desenvolvimento, como

IDE's e depuradores, são bem melhores que os para CORBA. Algumas tarefas comuns como a geração de referências e o registro de objetos são feitos automaticamente pelo CORBA, enquanto no DCOM é necessário que o programador especifique essas tarefas. A maneira de identificar objetos é muito semelhante em ambas as tecnologias, porém as nomenclaturas variam bastante. DCOM especifica diretivas de *garbage collection* para os objetos, porém CORBA ignora completamente esse mecanismo. Uma falha do DCOM em relação ao CORBA é a portabilidade. DCOM é preferencialmente usado em plataformas Windows (apesar de algumas tentativas de porte), enquanto CORBA é portável para praticamente qualquer plataforma.