Aula 08 - Objetos e componentes distribuídos

Introdução

- Middleware de objetos distribuídos
  - Adota o modelo de programação orientada a objetos para o desenvolvimento de sistemas distribuídos.
    Essa adoção traz diversas vantagens.
  - Vantagens
    - Encapsulamento
      tem-se acesso somente aquilo que o desenvolvedor quer que tenha acesso.
    - Abstração de dados
      Conseguir separar a especificação (o que ele faz) do objeto da implementação (como ele faz)
    - Soluções dinâmicas extensíveis
      dinâmica por implementar objetos e substituir de maneira dinâmica.
- Middleware baseado em componentes
  - Desenvolvida para superar limitações do modelo de middlewares de objetos distribuídos
    - Dependências implícitas
      Nos objetos distribuídos às dependências implícitas não ficam claras na interface, trazendo uma complexidade adicional pois muitos detalhes de baixo nível ficam embutidos no código, como segurança, controle de concorrência e tratamento de falha.
    - Complexidade de programação
    - Falta de separação de aspectos relacionados à distribuição

        - Ausência de suporte para implantação
          A responsabilidade de implantar os objetos distribuídos adequadamente entre as diversas máquinas que compõem o sistema continua sendo responsabilidade do desenvolvedor.

Objetos distribuídos

- Java RMI e CORBA têm muito em comum
- No entanto, RMI é restrito à linguagem JAVA, enquanto CORBA é uma solução multi-linguagem (C++, Java, Python e outras)
  CORBA garante que pode-se desenvolver um sistema em diferentes linguagens e podem interagir.
- Diferenças entre objetos e objetos distribuídos
  - Classe é essencial em orientação a objetos, mas não muito destacada em objetos distribuídos
    A classe especifica métodos e atributos que vão ser instanciados a partir da classe. Já nos objetos distribuídos (como várias linguagens são implementadas), o conceito de classe não é mais tão relevante.
  - Não há herança de implementação, apenas de interface.
    É comum haver herança mas apenas herança de interface nos objetos distribuídos.

- Funcionalidades providas pelo middleware
  - Comunicação entre objetos
    Existem mecanismos para que os objetos troquem mensagens entre si, implementado nos middlewares.
  - Gerenciamento de ciclo de vida
    Processo de criação, migração, deleção de objetos gerenciado pelo middleware.
  - Ativação e desativação

Tornar objeto ativo ou desativado passa a ser responsabilidade do middleware.

- Persistência
  Persistência dos objetos que possuem estados é feito pelo middleware.
- Serviços adicionais
  segurança, controle de transação, atribuição de nomes.

Estudo de caso: CORBA

- Em 1989 formou-se o OMG
- Motivação: Permitir que os objetos distribuídos fossem implementados em qualquer linguagem de programação e pudessem se comunicar com outros
  Permitir que objetos de qualquer linguagem pudessem se comunicar.
- Foi projetada uma linguagem de interface independente de qualquer linguagem específica

- O OBR (Object Request Broker) - ajuda um cliente a invocar distribuído um método em um objeto (localizar, invocar, comunicar a requisição do cliente ao objeto)

Principais componentes do framework

- Uma linguagem de definição de interface (IDL)
- Uma arquitetura
- Uma representação externa de dados (CDR)
- Uma forma padrão para referências a objeto remoto

RMI CORBA

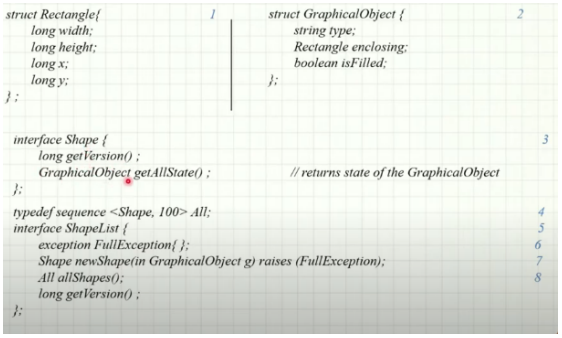
- Aspectos a serem considerados
  - O modelo de objeto do CORBA
    Se trata de qualquer programa que envia requisição para objetos remotos e recebe resposta.
  - A linguagem de definição de interface
  - Mapeamento para a linguagem de programação
- Semelhanças com Java RMI
  É aproveitado o fato de invocar objetos de forma remota.
- Modelo de objeto do CORBA
  - Qualquer programa que envie mensagens de requisição para objetos remotos e receba respostas

- O conceito de classe não existe em CORBA

IDL CORBA

- Especifica um nome e um conjunto de objetos que os clientes podem solicitar
- A gramática da IDL é um subconjunto da linguagem C++

Exemplo



```
struct Rectangle{                                   1
    long width;
    long height;
    long x;
    long y;
};

struct GraphicalObject {                            2
    string type;
    Rectangle enclosing;
    boolean isFilled;
};

interface Shape {                                   3
    long getVersion() ;
    GraphicalObject getAllState() ;        // returns state of the GraphicalObject
};

typedef sequence <Shape, 100> All;                  4
interface ShapeList {                               5
    exception FullException{ };                     6
    Shape newShape(in GraphicalObject g) raises (FullException);   7
    All allShapes();                                8
    long getVersion() ;
};
```

IDL CORBA

Interfaces IDL

- Descreve os métodos disponíveis nos objetos CORBA

Métodos IDL

São especificados dentro da interface.

- Parâmetros são rotulados como in (passado ao cliente para o objeto CORBA), out (passado ao objeto CORBA para o cliente), inout (nas duas direções).
- Exemplo:

```
void getPerson (in string name, out Person p)
```

Semântica de invocação

- Padrão: chamada no máximo uma vez, mas pode especificar a semântica talvez usando a palavra chave oneway

Exceções na IDL CORBA

- excepction FullException{};
  É especificado o que será executado em caso de exceção.
- Shape newShape (in GraphicalObject g) raises (FullException);

Tipos da IDL

- Suporta 15 tipos primitivos;
- CORBA suporta passagem por valor de objetos que não são CORBA;

Atributos

- Interfaces IDL podem ter atributos
- Para cada atributo, são gerados automaticamente dois modelos de acesso: um get e um set

- Podem ser definidos como readonly possui apenas o método get

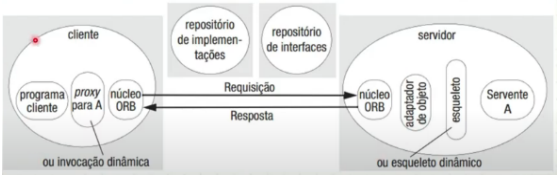
Herança: Regras

- As interfaces IDL podem ser estendidas por meio da herança.
- Uma interface estendida pode redefinir tipos, constantes, exceções, mas não pode redefinir métodos.
  Ao estender uma interface pode-se adicionar novos métodos mas não redefinir métodos pré-existentes.
- O valor de um tipo estendido é válido como parâmetro ou resultado do tipo ascendente.
  Se uma interface A for estendida numa interface B, o tipo B é válido como valor ou resultado de A.
- Uma interface IDL pode estender mais de uma interface
  Z pode estender B e C.
- A IDL não permite que uma interface herde métodos ou atributos com nomes comuns de duas interfaces
  Se B e C definirem um tipo de mesmo nome, Z não pode estender ambas simultaneamente.

Mapeamento de linguagem CORBA

- O mapeamento dos tipos IDL para os tipos de uma linguagem é simples
- Dificuldade com a semântica de passagem de parâmetros da IDL para Java
  - As classes Holder
    Tem a função de na conversão de um objeto out para Java, a fim de transformá-lo em um objeto do tipo holder.
  - Ex:
    - Em IDL: void getPerson (in string name, out Person p);
    - Em Java: void getPerson (String name, PersonHolder p)

A arquitetura CORBA



- Núcleo ORB: comunicação entre as partes.
  Se responsabiliza por fazer a comunicação entre cliente e servidor.

- Adaptador de objeto:
  Faz o mapeamento do objeto CORBA com o objeto cliente.
  - Faz a ligação entre objetos CORBA com interfaces IDL e as interfaces da linguagem de programação das classes serventes.
  - Cria referências de objeto remoto para objetos CORBA.
    Também realiza todas a invocações remotas para o servente adequado.
  - Envia cada RMI para o servente apropriado.
  - Ativa e desative os serventes.
  - POA – Portable Object Adapter
    Em algumas implementações se denomina POA.

A arquitetura CORBA

- **Esqueletos:** Geradas na linguagem do servidor por um compilador IDL
  Representante de determinado servente do lado do servidor.
  - Desempacota as requisições
  - Empacota os resultados e exceções nas mensagens de resposta.

- **Stubs/proxies de cliente**
  Realiza a mesma função do esqueleto, do lado do cliente.
  - Escritos na linguagem do cliente
  - Gerados a partir de uma interface IDL por um compilador IDL
  - Empacotam as requisições
  - Desempacotam exceções e resultados nas repostas.

A arquitetura CORBA

Repositório de implementações
Através dele é possível localizar um determinado servidor ou objeto que tenha invocações remotas a serem realizadas.

- Responsável por ativar servidores registrados sob demanda;
- Localizar servidores em execução.
- Armazena o mapeamento dos nomes dos adaptadores de objeto para o nome do arquivo contendo a implementação do objeto.

- Quando uma implementação de objeto é ativada no servidor, o hostname e o número da porta são adicionados ao mapeamento

A arquitetura CORBA

Repositório de interfaces

Através dele é possível localizar objetos de maneira dinâmica e realizar invocações remotas.

- Fornece informações sobre interfaces IDL registradas para clientes e servidores
- É possível fornecer os nomes dos métodos de uma dada interface, assim como os nomes e tipos dos argumentos e exceções.
- Adiciona um recurso de reflexão para CORBA.
  Na reflexão são feitas requisições ao objeto para que ele forneça informações necessárias para construção do proxy.
- ID de repositório: identificador atribuído a uma interface
- Nem todos os ORBS fornecem repositório de interfaces.

Referências de objeto remoto

Endereço para que se chegue a um objeto.

- IOR – Referências de objeto interoperáveis
- IOR Transientes - Duram tanto quanto o processo que contém esses objetos
- IORS persistentes – Duram entre as invocações dos objetos CORBA.

Formato IOR

ID de tipo de interface IDL	Protocolo e detalhes do endereço			Chave de objeto	
identificador de repositório de interfaces	IIOP	nome de domínio do *host*	número da porta	nome do adaptador	nome do objeto

Serviços CORBA

- Serviço de nomes
  Responsável pelo mapeamento dos nomes para referência do objeto remoto.
- Serviço de negócio
  Permite localizar objetos pelos atributos.
- Serviço de evento
  Para enviar notificações para assinantes de determinado serviço.
- Serviço de notificação
  Pode adicionar filtros para o serviço de eventos.
- Serviço de segurança
  Ele implementa autenticação, controle de acesso e auditoria. Tudo relacionado a segurança.
- Serviço de transação
  Pode criar transações baseadas seguindo a mesma ideia de banco de dados.
- Serviço de controle de concorrência

utiliza travas para que não haja condições de corrida.

- Serviço de estado persistente
  Garante que mesmo que uma máquina desligue o estado de um objeto fique salvo.
- Serviço de ciclo de vida
  Garante que possa criar, modificar e destruir objetos.

Exemplo de Cliente e Servidor CORBA

- Itens gerados pelo compilador de interface IDL
  - Interfaces Java equivalentes – duas por interface IDL
    - HelloOperations: Define as operações na interface IDL
    - Hello: Implementa as operações
  - Esqueleto do servidor para cada interface:
    - HelloPOA
  - Stubs do cliente, um para cada interface IDL:
    - HelloStub
  - Classes helpers e holders.
    - O método narrow na classe Helper converte referência de um objeto para a classe a que pertence.
    - Classes Holder lidam com argumentos out e inout

Definição de Interface

Hello.idl

```
module HelloApp
{
  interface Hello
  {
  string sayHello();
  oneway void shutdown();
  };
};
```

Implementando o servidor

- O servente HelloImpl é a implementação do Hello da inteface IDL
- O servente é uma subclasse de HelloPOA, gerado pelo compilador CORBA.
- Contém um método para cada operação IDL
  - sayhello()
  - shutdown()
- Todo o código extra para trabalhar com ORB, tal como o marshaling, é provido pelo esqueleto

HelloImpl

```java
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;

import java.util.Properties;

class HelloImpl extends HelloPOA {
  private ORB orb;

  public void setORB(ORB orb_val) {
    orb = orb_val;
  }

  // implement sayHello() method
  public String sayHello() {
    return "\nHello world !!\n";
  }

  // implement shutdown() method
  public void shutdown() {
    orb.shutdown(false);
  }
```

```java
public class HelloClient
{
  static Hello helloImpl;

  public static void main(String args[])
    {
      try{
        // Cria e inicializa o ORB
        ORB orb = ORB.init(args, null);

        // Obtém o root naming context
        org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");

        // Obtém a referência do NamingContext
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        // Resolve a referência do objeto no Naming
        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

      } catch (Exception e) {
        System.out.println("ERROR : " + e) ;
        e.printStackTrace(System.out);
        }
  }
```

HelloServer

```java
public class HelloServer {

  public static void main(String args[]) {
    try{
      // Cria e inicializa o ORB
      ORB orb = ORB.init(args, null);

      // Obtem a referência do rootPOA e ativa o POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // Cria o servente e registra-o no ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // Obtem uma referencia a partir do servente
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      // Obtém o root naming context NameService
      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Obtém a referência do NamingContext
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

```java
      // Vincula a referência do objeto ao nome
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);

      System.out.println("HelloServer ready and waiting ...");

      // Aguarda invocações dos clientes
      orb.run();

      catch (Exception e) {
        System.err.println("ERROR: " + e);
        e.printStackTrace(System.out);
      }

      System.out.println("HelloServer Exiting ...");

  }
}
```

HelloClient

Passos para execução

- Compilação
  - idlj -fall Hello.idl
  - javac *.java HelloApp/*.java
- Execução no servidor
  - orbd -ORBInitialPort 1050
  - java HelloServer -ORBInitialPort 1050
- Execução no cliente
  - java HelloClient -ORBInitialHost maq1 -ORBInitialPort 1050

De objetos a componentes

- Problemas de middleware orientado a objetos
  - Dependências implícitas

    Não ficam visíveis na interface, ou seja, vários detalhes de programação ficam sob

responsabilidade do programador.
    - As interfaces não fornecem todas as informações necessárias ao funcionamento de um objeto
    - Um objeto pode usar serviços que não estão visíveis externamente(segurança, nomes, transações);
  - Interação com o middleware
    - Muitos detalhes de baixo nível
    - O programador fica exposto a muitos detalhes, tal como nomes, POA e o ORB.

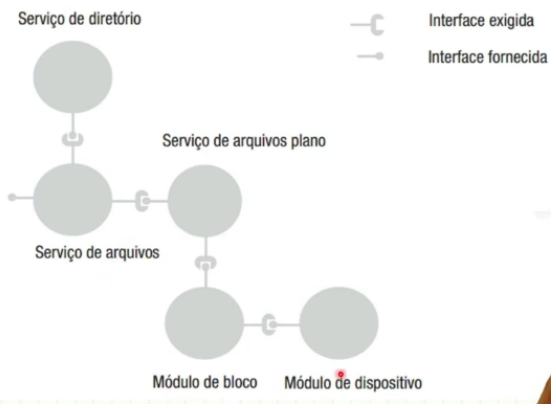
Problemas de middleware orientado a objetos

- Falta de separação de aspectos relacionados à distribuição
  - Mistura de código da aplicação com chamadas a serviços do middleware, tal como segurança, transações e replicação.
- Falta de suporte para distribuição
- Os objetos precisam ser distribuídos manualmente entre as máquinas

Essência dos componentes

- Um componente de software é uma unidade de composição com interfaces especificadas por contratos e contendo somente dependências contextuais explícitas.
- Um componente inclui:
  - Interfaces fornecidas – serviços providos pelo componente
  - Interfaces exigidas – dependências de outros componentes.
    o que o componente precisa para seu funcionamento

Exemplo de arquitetura de software



Serviços necessários para implementação dos diretórios de um sistema de arquivos.

Cada componente tem uma interface fornecida (o que ele disponibiliza) e alguns têm interface exigida (o que ele precisa para seu funcionamento).

Sistemas baseados em componentes

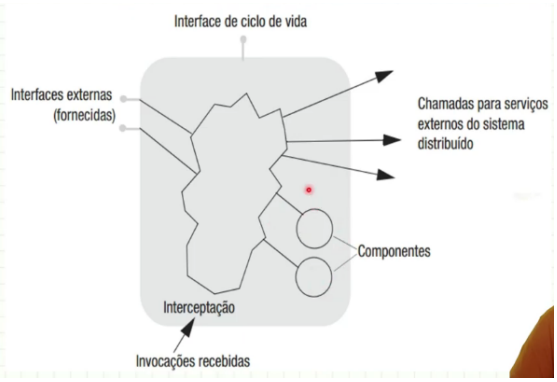
Diversos componentes realizando atividades específicas reunidos em um só componente de modo a atender alguma demanda.

- Suporta um estilo de desenvolvimento semelhante ao desenvolvimento de hardware.
- Componentes padrão são reunidos para desenvolver serviços mais sofisticados
- Montagem de software;

Componentes e sistemas distribuídos

- Middlewares baseados em componentes
  - Enterprise JavaBeans
  - CCM(Corba Component Model)
    Evolução do middleware baseado em objetos distribuídos. Caiu em desuso.
- Contêineres
  - Mantém um ou mais componentes que implementam a lógica da aplicação
  - Trata dos problemas não funcionais, aqueles não relacionados diretamente à aplicação
    Trata de gerenciamento de transações, serviços de nomes... para tirar a responsabilidade do desenvolvedor.

Contêiner



- Intercepta as invocações recebidas e garante as propriedades desejadas pela aplicação(segurança, por exemplo);
- No exemplo anterior, o contêiner faria todas as chamadas relacionadas ao POA;

- Middleware que suporta o padrão contêiner é chamado de servidor de aplicação.

Servidores de aplicação

Tecnologia	Desenvolvida por	Mais detalhes
WebSphere Application Server	IBM	[www.ibm.com]
Enterprise JavaBeans	SUN	[java.sun.com XII]
Spring Framework	SpringSource (uma divisão da VMware)	[www.springsource.org]
JBoss (Wildfly)	JBoss Communit	[www.jboss.org]
CORBA Component Model	OMG	[Wang *et al.* 2001]
JOnAS	OW2 Consortium	[jonas.ow2.org]
GlassFish	SUN	[glassfish.dev.java.net]

Estudo de caso: Enterprise Java Beans

- Especificação de arquitetura de componentes da plataforma JAVA EE
- Componentes são conhecidos como **beans** no EJB
- Fornece suporte para a arquitetura de três camadas físicas
- O contêiner fornece suporte para importantes serviços.
- O contêiner oculta do desenvolvedor dos beans detalhes dos serviços providos.

O EJB

- O EJB é uma arquitetura pesada e destinada a certos tipos de aplicação.
  usada em aplicações de grande porte, como internet e comércio eletrônico no geral.
- Comum em aplicações de comércio eletrônico.
- O modelo de componente do EJB
  - Bean fornece uma ou mais interfaces de negócio com interfaces remotas ou locais (interface fornecida).
  - Bean=conjunto de interfaces de negócio locais e remotas + classe bean que implementa as interfaces.
  - Estilos de bean suportados
    - Beans de sessão
      usados em aplicações que precisam prover determinado serviço, como: uma compra de um usuário dispara uma sessão.
      - Com estado
        quando se quer manter histórico da conversa realizada

no bean de sessão.

- Sem estado sem histórico.

- Beans baseados em mensagens
  Criado no EJB para suportar comunicação indireta (mandar mensagem sem saber o destinatário.)

POJO e anotações

- POJO – Plain old java objects (objeto java puro)
  A lógica da programação em java, onde se implementa a regra de negócio.
- Um bean é um POJO complementado com anotações
- Anotações em Java permitem associar metadados a pacotes, classes, métodos, parâmetros e variáveis.
- Ex: Anotações podem ser utilizadas para especificar o estilo do bean
  especificam quais serviços e como serão utilizados por determinado bean.

Anotações

Também utilizadas para especificar se uma interface é remota ou local

```
@Remote public interface Orders {...}
@Local public interface Calculator {...}
```

Gerenciamento de transações no EJB

- Gerenciado pelo Bean: inclusão do código de transações dentro do bean.

```
@Stateful
@TransactionManagement (BEAN)
public class eShop implements Orders {
    @Resource javax.transaction.UserTransaction ut;
    public void MakeOrder (...) {
        ut.begin ();
        ...
        ut.commit ();
    }
}
```

Gerenciamento de transações no EJB

- Gerenciado pelo contêiner
  modo ideal.
  - Evita código explícito dentro do bean

```
@Stateful public class eShop implements Orders {
    ...
    @TransactionAttribute (REQUIRED)
    public void MakeOrder (...) {
    ...
    }
}
```

Injeção de dependência

Padrão de programação muito comum.

- O contêiner é responsável por gerenciar e solucionar as relações entre um

componente e suas dependências(interfaces exigidas).

- O contêiner usa reflexão para isso.
  solucionar dependências invocando métodos do componente.
- A anotação @Resource

Gerenciamento de transações no EJB

- Gerenciado pelo Bean: inclusão do código de transações dentro do bean.

```
@Stateful
@TransactionManagement (BEAN)
public class eShop implements Orders {
    @Resource javax.transaction.UserTransaction ut;
    public void MakeOrder (...) {
        ut.begin ();
        ...
        ut.commit ();
    }
}
```

Interceptação em EJB

- Tipos de operações em beans suportadas
  - Chamada de métodos associadas a uma interface de negócio
  - Eventos do ciclo de vida quando se cria ou se destrói um objeto.

Interceptação de métodos

Associando um método de interceptação a uma classe

```
@Stateful
public class eShop implements Orders {
    public void MakeOrder (...) {
        ...
    }
    @AroundInvoke
    public Object log(InvocationContext ctx) throws Exception {
        System.out.println ("The following method was invoked: "+
                            ctx.getMethod().getName());
        return invocationContext.proceed();
    }
}
```

Interceptação de eventos de ciclo de vida

- Permite associar interceptadores à criação e à exclusão de componentes.
  - @PostConstruct
  - @PreDestroy

```
@Stateful
public class eShop implements Orders {
    ...
    public void MakeOrder (...) {...}
    ...
    @PreDestroy void TidyUp() {... }}
}
```