EJB
Enterprise JavaBeans

(Boa parte destes slides foram obtidos de seminário de Myrthes Aguiar)

MAC 440/5759

Sistemas de Objetos Distribuídos

Prof. Fabio

Departamento de Ciência da Computação

IME - USP

JavaBeans @ EJB

- JavaBeans (~1996); EJB (~1999).
- JavaBeans é um modelo elegante para construção de aplicações baseadas em componentes.
- Em geral, é usado para construção de aplicações locais/centralizadas.
- Comunicação orientada a eventos.
- Interface gráfica auxilia na criação de aplicações com ajuda do mouse.

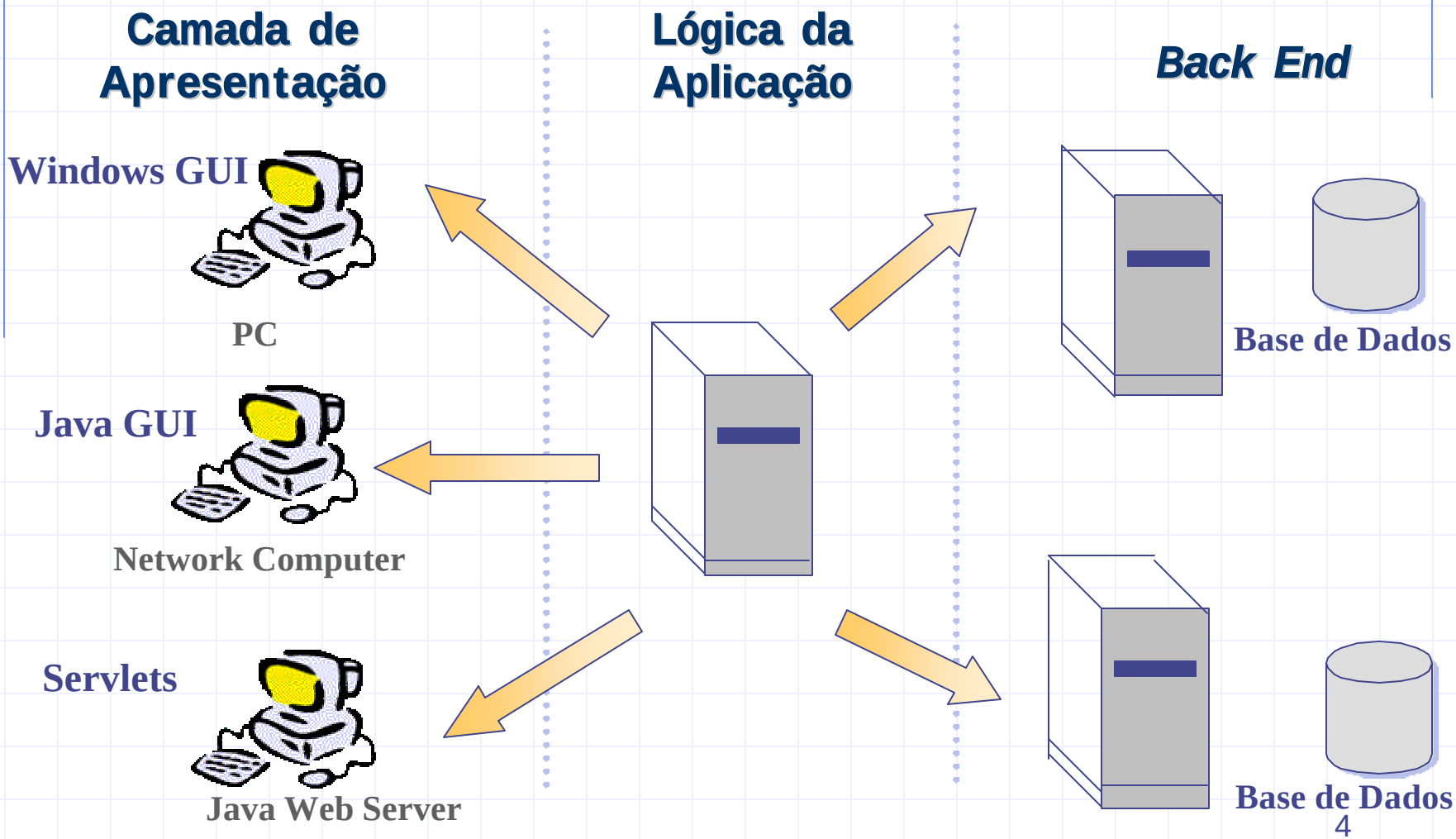
Enterprise JavaBeans -EJB

- Arquitetura de componentes para o desenvolvimento e implantação de ***servidores*** em ambientes distribuídos
- Tecnologia de Monitores de Transações de Componentes - CTMs, sustenta a arquitetura EJB
- CTMs
  - Híbrido entre monitores TP(CICS IBM,TUXEDO BEA) e tecnologias ORB
  - Infra-estrutura para
    - gerenciamento automático de transações
    - distribuição de objetos
    - concorrência
    - segurança
    - recursos
    - ampla população de usuários
    - trabalhos de missão crítica
  - Hoje em dia, chamados de “Servidores de Aplicações”

EJB – outros serviços

- **Gerenciamento de estado**
- **Ciclo de vida**
- **Persistência**

Arquiteturas de 3 camadas



APIs Java Utilizadas em Aplicações EJB / J2EE

API	DESCRIÇÃO	API	DESCRIÇÃO
EJB	define modelo de componente servidor que fornece portablilidade e implementa serviços	Servlets e JSP	Java Servlets e Java Server Pages suporta, HTML dinâmico e gerenciamento de sessão p/ clientes usando browser
JNDI	fornece acesso ao servidor de nomes e diretórios	JMS	Java Messaging Service suporta comunicação assíncrona através de vários sistemas de *messaging*
RMI	cria interfaces remotas p/ computação distribuída na plataforma JAVA	JTA	Java Transaction API fornece demarcação de transação
JAVA IDL	cria interfaces remotas p/ suportar comunicação CORBA. Inclui compilador IDL e um ORB enxuto que suporta IIOP	JTS	Java Transaction Service define um serviço de gerenciamento de transações distribuídas baseado no OTS CORBA
JDBC	fornece acesso uniforme a bancos de dados relacionais		

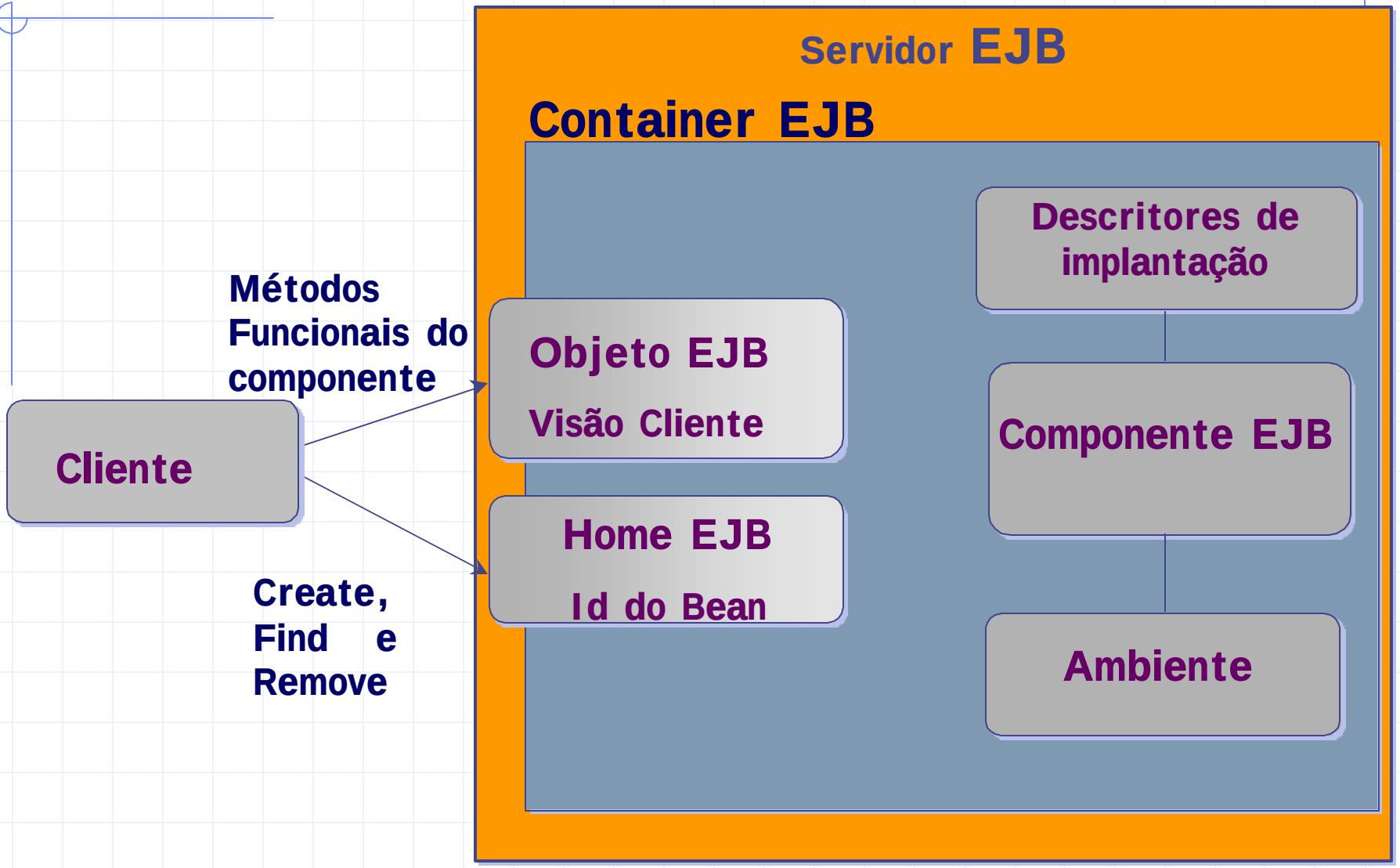
Elementos da Arquitetura EJB

- Servidor EJB
  - servidor de aplicação genérico que fornece um ambiente compatível com a especificação da arquitetura EJB
  - fornece um ou mais containers para os componentes nele implantados
  - responsável pelo gerenciamento e coordenação da alocação de recursos:
    - Threads, processos, memória, conexões a BD
    - serviços: transações, nomes, segurança e persistência
- Container EJB
  - fornece contexto de execução e contexto transacional aos componentes
  - registra o componente no serviço de nomes, cria e destrói instâncias
  - fornece interface remota para o componente
  - gerencia transações, estado e persistência

Elementos da Arquitetura EJB *(cont)*

- Componente EJB
  - **Interface Home**
    - define os métodos de ciclo de vida do componente:
      - criação, remoção e busca
    - através dessa interface, clientes vêem componentes EJB como uma coleção homogênea de instâncias
  - **Interface Remote**
    - define os métodos funcionais do componente
    - representa a visão que o cliente terá do componente
    - expõe todas as interfaces relacionadas à aplicação
  - **Classe do Componente**
    - implementa os métodos funcionais (aspectos funcionais do componente)
- Tipos de Componentes EJB:
  - Componentes Entidades (***Entity Beans***)
  - Componentes Sessão (***Session Beans***)

Arquitetura EJB



Papéis EJB

- Fornecedor de Componentes EJB
  - definição das interfaces home e remote
  - gera arquivo ejb-jar contendo 1 ou mais componentes
  - descritor de implantação
- Montador da Aplicação
  - gera arquivo ejb-jar com os componentes e suas instruções de montagem
- Implantador
  - geração das classes e interfaces adicionais que habilitam o container a gerenciar componentes EJB
- Fornecedor de Servidor EJB / Container EJB
  - ferramentas de implantação
  - suporte de tempo de execução para as instâncias dos componentes
  - API padrão para acesso aos serviços de transação, acesso a BD, ...
- Administrador do Sistema

Entity Beans

- Fornece uma visão em forma de objeto dos dados na BD (entidades)
- Permite acesso compartilhado por múltiplos usuários
- Possui “vida longa” → igual a do dado associado na BD
- Uma instância de um bean entidade, sua chave primária e sua interface remota sobrevivem a eventual quedas do container
- Queda não transparente ao cliente → exceção ao chamar método
- Criação de um bean entidade → inserção de novo registro na BD
- Tipos de Persistência:
  - Gerenciada pelo container
  - Gerenciada pelo componente

Session Beans

- Representa uma sessão de um único cliente
- Não representa dados na BD, mas pode atualizá-los
- Vida relativamente curta
- Removido quando ocorre queda do container ➡ cliente tem de restabelecer um novo objeto sessão para continuar o trabalho
- Úteis para descrever e gerenciar interações entre os componentes entidades e implementar um fluxo de trabalho
- Tipos:
  - Sem estado - não mantém o estado entre as chamadas dos métodos
  - Com estado { - dedicado a um cliente pelo tempo de vida da instância
    - mantém estado conversacional: pode guardar dados relativos ao cliente entre chamadas de métodos

Visão do Cliente

- ***Componente Session*** ➡ objeto não persistente em execução em um servidor
- ***Componente Entity*** ➡ objeto que fornece uma visão OO de entidades armazenadas num BD, ou de entidades persistentes implementadas por aplicação corporativa já existente
- ***Container*** ➡ gera ou fornece as classes que implementam as interfaces Home e Remote
- ***Cliente*** ➡ utiliza JNDI para localizar e obter uma referência para interface Home
- ***Através da Home*** ➡ o cliente obtém uma referência para interface Remote

Exemplo de Aplicação:
Sistema de Reserva para Cruzeiros Marítimos

```
// Definition of the EJB Remote Interface
package com.titan.cabin;
import java.rmi.RemoteException;

public interface Cabin extends javax.ejb.EJBObject {
    public String getName() throws RemoteException;
    public void setName(String str) throws RemoteException;
    public int getDeckLevel() throws RemoteException;
    public void setDeckLevel(int level)
                        throws RemoteException;
    public int getShip() throws RemoteException;
    public void setShip(int sp) throws RemoteException;
    public int getBedCount() throws RemoteException;
    public void setBedCount(int bc) throws RemoteException;
}
```

Sistema de Reserva para Cruzeiros Marítimos

Interface CabinHome

```
package com.titan.cabin;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface CabinHome extends javax.ejb.EJBHome {

    public Cabin create (int id)
        throws CreateException, RemoteException;

    public Cabin findByPrimaryKey (CabinPK pk)
        throws FinderException, RemoteException;
}
```

Sistema de Reserva para Cruzeiros Marítimos

Classe CabinBean (a implementação)

```
package com.titan.cabin;
import javax.ejb.EntityContext;
public class CabinBean implements javax.ejb.EntityBean {
    public int id, deckLevel, ship, bedCount;
    public String name;
   public CabinPK ejbCreate(int id){
        this.id = id;
        return null;
    }
    public void ejbPostCreate(int id){// Do nothing. Required.}
    public String getName(){ return name; }
    public void setName(String str){ name = str; }
    public int getShip(){ return ship; }
    public void setShip(int sp) { ship = sp; }
    public int getBedCount(){ return bedCount; }
    public void setBedCount(int bc){ bedCount = bc; }
    public int getDeckLevel(){ return deckLevel;}
    public void setDeckLevel(int level ){ deckLevel = level; }
```

Classe CabinBean (a implementação)

```
    public void setEntityContext(EntityContext ctx){
        // Not implemented.
    }
    public void unsetEntityContext(){
        // Not implemented.
    }
    public void ejbActivate(){  // Not implemented.
    }
    public void ejbPassivate(){ // Not implemented.
    }
    public void ejbLoad(){      // Not implemented.
    }
    public void ejbStore(){     // Not implemented.
    }
    public void ejbRemove(){    // Not implemented.
    }
}
```

Classe Primary Key

```
package com.titan.cabin;
public class CabinPK implements java.io.Serializable {
    public int id;
    public int hashCode( ){
        return id;
    }
    public boolean equals(Object obj){
        if(obj instanceof CabinPK){
            return (id == ((CabinPK)obj).id);
        }
        return false;
    }
    public String toString(){
       return String.valueOf(id);
    }
}
```

Descritores de Implantação

- Contém informações sobre o ambiente no qual o componente deve ser executado.
- Até EJB 1.0
  - Era um objeto Java seriado gravado em disco
- De EJB 1.1 em diante
  - Especificado através de um arquivo XML
- No momento da implantação da aplicação utiliza-se um arquivo jar contendo:
  - Código dos componentes necessários
  - Arquivo XML com descritor da implantação

Descritor de Implantação (Deployment Descriptor)

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
   JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-
   jar_1_1.dtd">
 <ejb-jar>
   <enterprise-beans>
        <entity>
           <description>
                  This Cabin enterprise bean entity represents a
                 cabin on a cruise ship.
          </description>
          <ejb-name>CabinBean</ejb-name>
          <home>com.titan.cabin.CabinHome</home>
          <remote>com.titan.cabin.Cabin</remote>
          <ejb-class>com.titan.cabin.CabinBean</ejb-class>
          <persistence-type>Container</persistence-type>
          <prim-key-class>com.titan.cabin.CabinPK</prim-key-class>
          <reentrant>False</reentrant>
```

Descritor de Implantação

```
        <cmp-field><field-name>id</field-name></cmp-field>
        <cmp-field><field-name>name</field-name></cmp-field>
        <cmp-field><field-name>deckLevel</field-name></cmp-field>
         <cmp-field><field-name>ship</field-name></cmp-field>
        <cmp-field><field-name>bedCount</field-name></cmp-field>
    </entity>
</enterprise-beans>
```

Descritor de Implantação

```
<assembly-descriptor>
  <security-role>
      <description>
          This role represents everyone who is allowed full
          access to the cabin bean.
      </description>
       <role-name>everyone</role-name>
  </security-role>
   <method-permission>
       <role-name>everyone</role-name>
       <method>
            <ejb-name>CabinBean</ejb-name>
            <method-name>*</method-name>
       </method>
   </method-permission>
```

Descritor de Implantação

```
        <container-transaction>
              <method>
                    <ejb-name>CabinBean</ejb-name>
                    <method-name>*</method-name>
              </method>
        <trans-attribute>Required</trans-attribute>
        </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

Exemplo de Cliente

```
public class Client_1 {
  public static void main(String [] args){
    try {
     Context jndiContext = getInitialContext();
     Object obj = jndiContext.lookup("java:env/ejb/CabinHome");
     CabinHome home = (CabinHome)
        javax.rmi.PortableRemoteObject.narrow(obj, CabinHome.class);
     Cabin cabin_1 = home.create(1);
     System.out.println("created it!");
     cabin_1.setName("Master Suite");
     cabin_1.setDeckLevel(1);
     cabin_1.setShip(1);
     cabin_1.setBedCount(3);

     CabinPK pk = new CabinPK();
     pk.id = 1;
     System.out.println("keyed it! ="+ pk);
```

Exemplo de Cliente (cont.)

```
            Cabin cabin_2 = home.findByPrimaryKey(pk);
            System.out.println("found by key! ="+ cabin_2);
            System.out.println(cabin_2.getName());
            System.out.println(cabin_2.getDeckLevel());
            System.out.println(cabin_2.getShip());
            System.out.println(cabin_2.getBedCount());

        } catch (java.rmi.RemoteException re)
                {re.printStackTrace();}
            catch (javax.naming.NamingException ne)
                {ne.printStackTrace();}
            catch (javax.ejb.CreateException ce)
                {ce.printStackTrace();}
            catch (javax.ejb.FinderException fe)
                {fe.printStackTrace();}
    }
```

Exemplo de Bean de Sessão

- *Session Beans* encapsulam operações complexas com as entidades do sistema.
- Retiram o gerenciamento do negócio do cliente
- Simplificam ao máximo o código do cliente

Bean de Sessão: Travel Agent

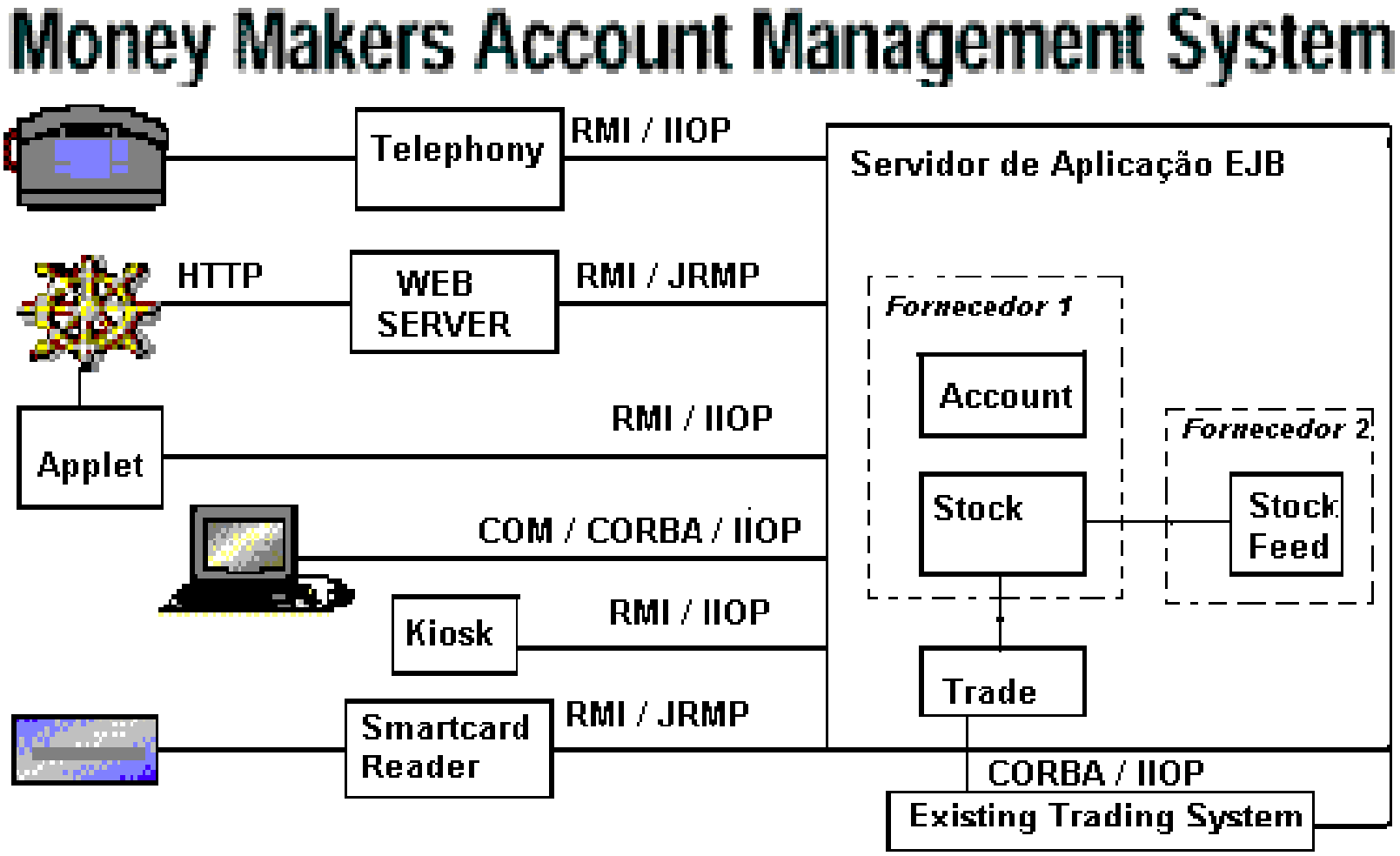
```
public interface TravelAgent extends javax.ejb.EJBObject {
    public void setCruiseID(int cruise)
        throws RemoteException, FinderException;
    public int getCruiseID()
        throws RemoteException, IncompleteConversationalState;
    public void setCabinID(int cabin)
        throws RemoteException, FinderException;
    public int getCabinID()
        throws RemoteException, IncompleteConversationalState;
    public int getCustomerID( )
        throws RemoteException, IncompleteConversationalState;

    public Ticket bookPassage(CreditCard card, double price)
        throws RemoteException,IncompleteConversationalState;

    public String [] listAvailableCabins(int bedCount)
        throws RemoteException,IncompleteConversationalState;
}
```

Restrição no acesso aos *Beans*: RMI-IIOP

- As interfaces Remote e Home de todo componente tem que satisfazer as restrições impostas por *RMI over IIOP.*
- Possibilita que qualquer *bean* possa ser acessado por clientes em qq. linguagem através de CORBA.

- Restrições:
  1. Qdo. fizer herança múltipla de interfaces, não usar polimorfismo (dois métodos com o mesmo nome).
  2. Tipos *Serializable* não podem implementar a interface **`java.rmi.Remote.`**

Os mesmos *Beans* podem ser acessados através de diferentes protocolos e contextos



Gerenciamento de Transações

- Container é responsável pela:
  - Geração de transações para as interações do cliente com o componente
  - Detecção de transações requisitadas pelo cliente
  - Decisão por executar o método chamado dentro da transação do cliente, dentro de uma nova transação ou permitir que o componente gerencie os limites da transação

- Java Transaction API (JTA) ➡ define as interfaces necessárias para a interação com um servidor de transações

- Atributo de suporte a transações especificado no descritor de implantação (**TransactionAttribute**)

Atributos de Transações

- TransactionAttribute

  - **TX_BEAN_MANAGED** ➡ demarcação manual de transações usando JTA
  - **TX_NOT_SUPPORTED** ➡ o componente não pode ser executado dentro do contexto de uma transação.
  - **TX_SUPPORTS** ➡ o componente pode ser executado com ou sem o contexto de transação.
  - **TX_REQUIRED** ➡ o componente precisa ser executado dentro do contexto de uma transação.
  - **TX_REQUIRES_NEW** ➡ o componente precisa ser executado dentro do contexto de uma nova transação.
  - **TX_MANDATORY** ➡ o componente precisa sempre ser executado dentro do contexto de uma transação.

Mais sobre Descritores de Implantação

- **Tipos de Informação:**
  - **ESTRUTURAIS** → descrevem a estrutura do componente e declaram as dependências externas.
  - **DE MONTAGEM** → descrevem como o(s) componentes(s) contidos num arquivo ejb-jar se relacionam de modo a formar uma unidade de implantação.
- **Na implantação**
  - Informações adicionais como por exemplo, atribuição de valores para variáveis de ambiente e mapeamento entre campos do bean entidade e a base de dados.

EJB Versão 2.0

- Funcionalidades:
  - **Introdução de componentes dirigidos por mensagens** (integração JMS)
  - **Suporte para relacionamentos gerenciados pelo container, para componentes do tipo entidade implantados em um mesmo servidor EJB**
  - **Interfaces local home e local** (análogas à **home** e **remote** da versão 1.1)
    - Clientes Locais
    - Acessos mais "leves", sem o custo das chamadas remotas
  - **EJB QL** ➡ permite a navegação pela rede de entity beans definida pelos relacionamentos
  - **Suporte para implementação de métodos adicionais na home com lógica da aplicação independente de instância específica**
  - **Protocolo de interoperabilidade baseado em IIOP**

EJB Versão 2.0

- **Tipo de Componentes**
  - ***Session beans***
  - ***Entity beans***

  Características similares às especificadas na versão 1.1

  - ***Message-driven beans***
    - Executam após recepção de mensagem do cliente
    - Podem ser cientes de transações
    - Não representam dados num BD, mas podem consultar e atualizar
    - Sem estado
    - Removidos após queda do container

EJB vs. CCM

- EJB é um subconjunto do CCM
  - Com exceção dos componentes orientados a mensagens
- Ambos permitem integração de sistemas heterogêneos
- EJB usa CORBA para integração com outras linguagens e sistemas legados
- CORBA usa IDL, CIDL e XML para especificações
- Java usa interfaces Java e XML
- CCM possui o conceito de facetas, receptáculos e fontes e consumidores de eventos