Objetos y Componentes Distribuidos

Capa Intermedia

Capa intermedia con Objetos Distribuidos

- Encapsulamiento
- Abstracción Datos
- Soluciones dinámicas y extensibles
- Ejm: Java RMI y Corba

Capa intermedia con Componentes.

- Dependencias implicitas
- Programación compleja
- Detalles relacionados con distribución
- Soporte para depuración
- Ejm: Enterprise JavaBeans y -Fractal

Objetos Distribuidos

Objects	*Distributed objects*	*Description of distributed object*
Object references	Remote object references	Globally unique reference for a distributed object; may be passed as a parameter.
Interfaces	Remote interfaces	Provides an abstract specification of the methods that can be invoked on the remote object; specified using an interface definition language (IDL).
Actions	Distributed actions	Initiated by a method invocation, potentially resulting in invocation chains; remote invocations use RMI.
Exceptions	Distributed exceptions	Additional exceptions generated from the distributed nature of the system, including message loss or process failure.
Garbage collection	Distributed garbage collection	Extended scheme to ensure that an object will continue to exist if at least one object reference or remote object reference exists for that object, otherwise, it should be removed. Requires a distributed garbage collection algorithm.

Componentes Marco independiente del lenguaje con CORBA
1-1 Modelo de Objeto CORBA

Es un modelo de objeto distribuido:

- Los clientes no son necesariamente objetos. Un objeto CORBA se refiere a un objeto remoto
- Su implementación es una interface, tiene una referencia a un objeto remoto y es capaz de responder a invocaciones de métodos

Un objeto CORBA puede implementarse con un lenguajes no-OO

Componentes Marco independiente del lenguaje con CORBA
1.2- LDI (IDL interfaces) Shape y ShapeList

```
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();                         // returns sequence of remote object references  8
        long getVersion() ;
};
```

La gramática de LDI es un subconjunto de ANSIC++
3,5 --- Interfaces, 6 --- Excepción
7 – parámetro cuyo argumento será pasado al servidor en un Request

Módulo LDI (IDL) Whiteboard
Unidades lógicas para interfaces

```
module Whiteboard {
    struct Rectangle{
    ...} ;
    struct GraphicalObject {
    ...};
    interface Shape {
    ...};
    typedef sequence <Shape, 100> All;
    interface ShapeList {
    ...};
};
```

Tipos de Constructores de LDI - 1

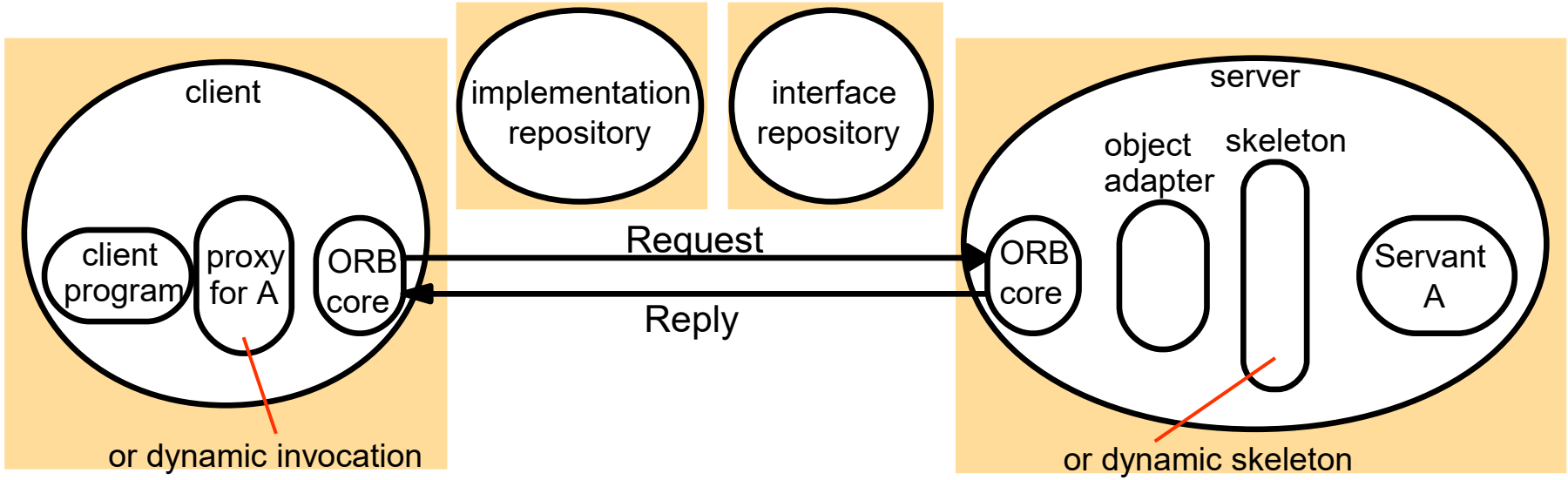
Type	*Examples*	*Use*
sequence	*typedef sequence <Shape, 100> All;* *typedef sequence <Shape> All* bounded and unbounded sequences of *Shapes*	Defines a type for a variable-length sequence of elements of a specified IDL type. An upper bound on the length may be specified.
string	*String name;* *typedef string<8> SmallString;* unbounded and bounded sequences of characters	Defines a sequences of characters, terminated by the null character. An upper bound on the length may be specified.
array	*typedef octet uniqueId[12];* *typedef GraphicalObject GO[10][8]*	Defines a type for a multi-dimensional fixed-length sequence of elements of a specified IDL type.

Continua ...

Tipos de Constructores de LDI - 2-

Type	*Examples*	*Use*
record	*struct GraphicalObject {* *string type;* *Rectangle enclosing;* *boolean isFilled;* *};*	Defines a type for a record containing a group of related entities. *Structs* are passed by value in arguments and results.
enumerated	*enum Rand* *(Exp, Number, Name);*	The enumerated type in IDL maps a type name onto a small set of integer values.
union	*union Exp switch (Rand) {* *case Exp: string vote;* *case Number: long n;* *case Name: string s;* *};*	The IDL discriminated union allows one of a given set of types to be passed as an argument. The header is parameterized by an *enum*, which specifies which member is in use.

Componentes Marco Independiente del lenguaje con CORBA
2- Arquitectura



ORB: Object Request Broker

- **Invocación Estática y Dinámica**
- **ORB core: Módulo de comunicación**
- **Object Adapter: puente entre objeto con IDL y la interfaz de lenguaje de programación del Servant. Crea Referencia remota, Despacha RMI hacia sirviente y activa objetos. POA**
- **Skeleton: despachar las invocaciones métodos remotas al servant, (des)empaqueta argumentos**
- **Proxies: (des)empaquetar excepciones/solicitudes**
- **Implementation Repository: activar/localizar servidores en demanda (nombre Object Adapter +rutas de implementaciones de objetos + host y puerto server)**
- **Interface Repository: interfaces registradas- facilita la reflexión**

Componentes Marco independiente del lenguaje con CORBA
4- Referencia a Objetos Remotos

Referencias a objetos interoperable (IOR)

IDL interface type ID Protocoal and address detail Object Key

Interface repository identify	IIOP	Host domain name	Port name	Adapter name	Object name

- Identificador del repositorio de la interface
- IIOP: Internet Inter-ORB. Protocol. Usa TCP, con la dirección del servidor consiste nombre del dominio del host y el numero del puerto
- Identificación del Objeto Corba

IOR persistentes o transitorias

Servicios CORBA

CORBA Service	*Role*	*Further details*
Naming service	Supports naming in CORBA, in particular mapping names to remote object references within a given naming context (see Chapter 9).	[OMG 2004b]
Trading service	Whereas the Naming service allows objects to be located by name, the Trading service allows them to be located by attribute; that is, it is a directory service. The underlying database manages a mapping of service types and associated attributes onto remote object references.	[OMG 2000a, Henning and Vinoski 1999]
Event service	Allows objects of interest to communicate notifications to subscribers using ordinary CORBA remote method invocations (see Chapter 6 for more on event services generally).	[Farley 1998, OMG 2004c]
Notification service	Extends the event service with added capabilities including the ability to define filters expressing events of interest and also to define the reliability and ordering properties of the underlying event channel.	[OMG 2004d]

this figure continues on the next slide

Servicios CORBA -2-

Security service	Supports a range of security mechanisms including authentication, access control, secure communication, auditing and nonrepudiation (see Chapter 11).	[Blakely 1999, Baker 1997, OMG 2002b]
Transaction service	Supports the creation of both flat and nested transactions (as defined in Chapters 16 and 17).	[OMG 2003]
Concurrency control service	Uses locks to apply concurrency control to the access of CORBA objects (may be used via the transaction service or as an independent service).	[OMG 2000b]
Persistent state service	Offers a persistent object store for CORBA, used to save and restore the state of CORBA objects (implementations are retrieved from the implementation repository).	[OMG 2002d]
Lifecycle service	Defines conventions for creating, deleting, copying and moving CORBA objects; for example, how to use factories to create objects.	[OMG 2002e]

De Objetos Distribuidos a Componentes Distribuidos

1-Dependencias Implícitas.

Hay un requerimiento claro para especificar no sólo las interfaces que ofrece un objeto, sino también las dependencias que objeto tiene con otros objetos en la configuración distribuida

2-Interacción con el middle ware.

Necesidad de simplificar la programación de aplicaciones distribuidas, para presentar una separación entre los código relacionados con operación en el middleware y el código asociado con la solicitud

3- Separación relacionada a la distribución.

Soporte a toda la gama de servicios del sistema distribuido, y las complejidades de este tipo de servicios debe estar oculto siempre que sea posible para el programador

4- Soporte al desarrollo.

Middleware deben proporcionar apoyo intrínseco al desarrollo de manera que el software distribuido puede ser instalado y desplegado al igual que el software para una sola máquina , con las complejidades de la implementación ocultos por parte del usuario

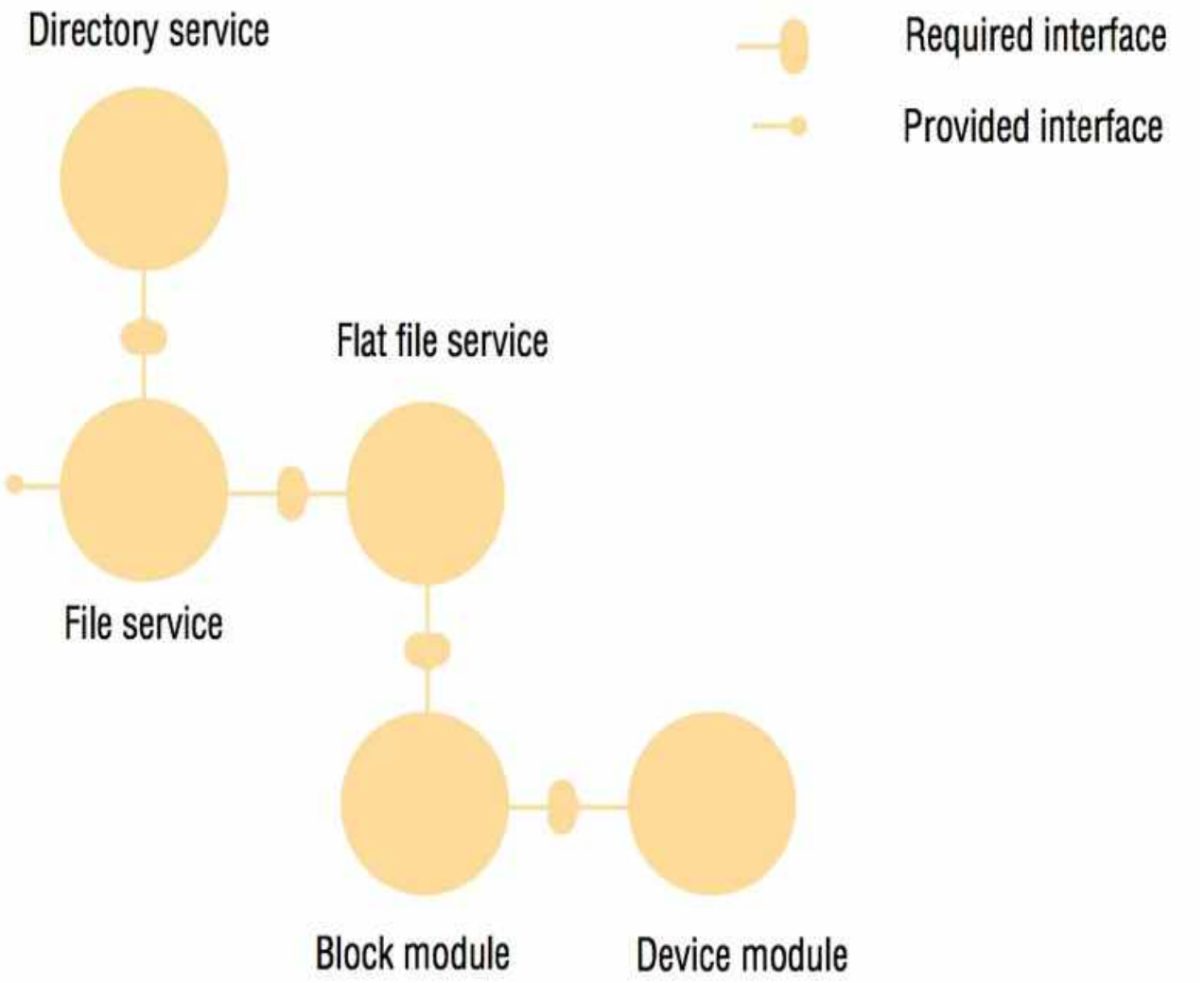
Componentes

Componentes : Un componente de software es una unidad de composición con interfaces especificadas contractualmente y solamente explicitas las dependencias del contexto.

Un componente se especifica en términos de un contrato, el cual incluye :

- un conjunto de interfaces propuestas que el componente ofrece como servicios a otros componentes;
- un conjunto de interfaces necesarias - es decir, las dependencias que este componente tiene en términos de otros componentes que deben estar presentes y conectados a este componente para que funcione correctamente

Un ejemplo de la arquitectura de software



Tipos de Interfaces:

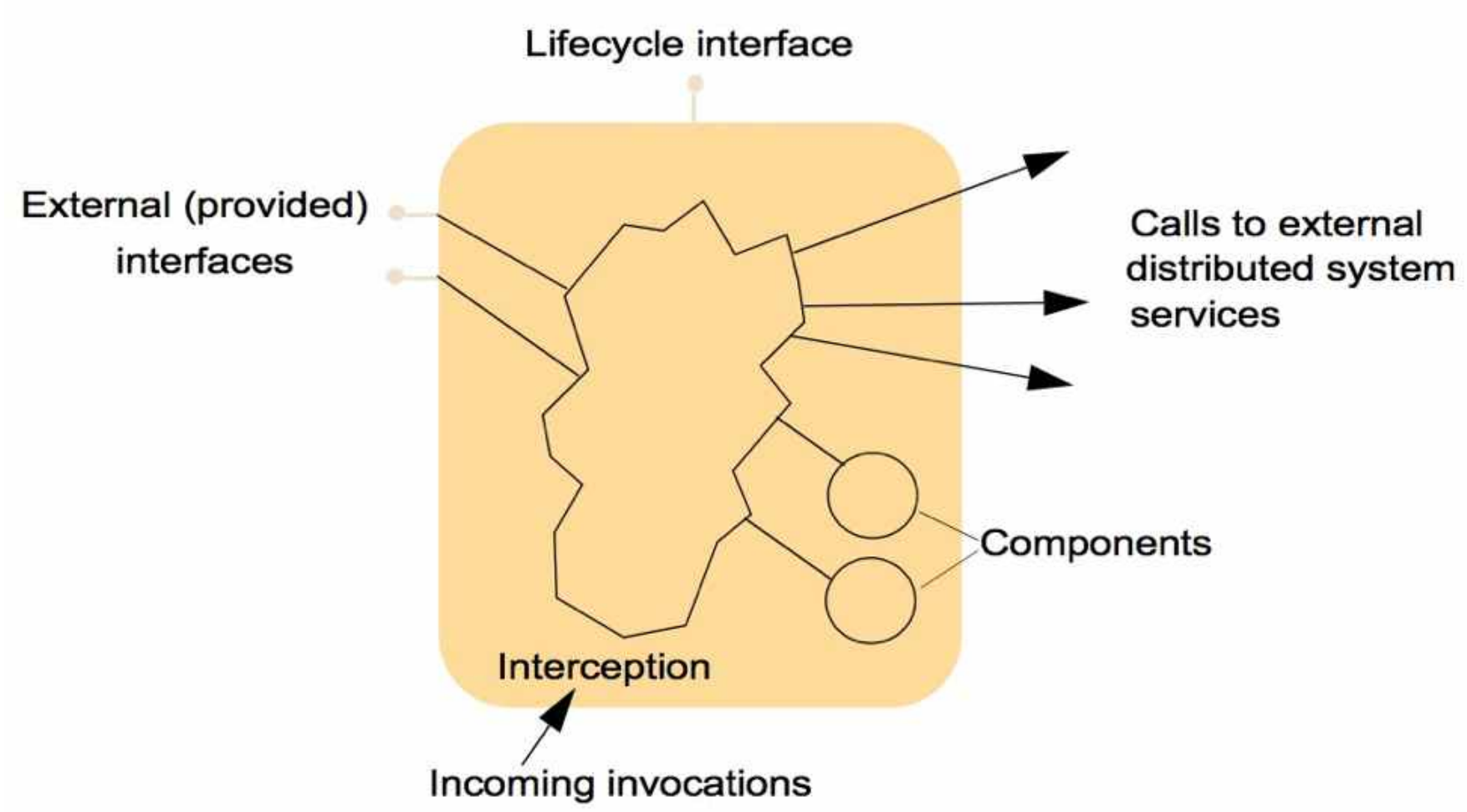
- Soportan invocación de métodos remotos, ejm CORBA y Java RMI
- Soportan eventos distribuidos, ejm arquitecturas comunicación indirecta

Programación basada en Componentes

- Relacionado con el desarrollo de componentes y su composición
- La meta es soportar un estilo de desarrollo de software que, al igual que el desarrollo de hardware, usa componentes y los ensambla para desarrollar un servicio sofisticado

- Recipientes consiste de:
  - un front-end (quizás basado en la web) para el cliente;
  - un recipiente que contiene uno o más componentes que implementan la aplicación o lógica de negocios;
  - servicios del sistema que gestionan los datos relacionados en el almacenamiento persistente.
- Su propósito es proporcionar un alojamiento en el lado del servidor para los componentes y proporcionar la necesaria separación, donde los componentes se ocupan de la aplicación y el contenedor del sistemas distribuido y middleware

Estructura de un contenedorr



Aplicaciones en el Servidor

Technology	*Developed by*	*Further details*
WebSphere Application Server	IBM	[www.ibm.com]
Enterprise JavaBeans	SUN	[java.sun.com XII]
Spring Framework	SpringSource (a division of VMware)	[www.springsource.org]
JBoss	JBoss Community	[www.jboss.org]
CORBA Component Model	OMG	[Wang *et al.* 2001]
JOnAS	OW2 Consortium	[jonas.ow2.org]
GlassFish	SUN	[glassfish.dev.java.net]

Middleware basada en Componentes

Middleware basado en componentes da soporte al desarrollo de configuraciones de componentes.

Software se empaquetan como arquitecturas de software (componentes y sus interconexiones) junto con *descriptores de despliegue* que describen cómo las configuraciones deben ser desplegados en un entorno distribuido

Descriptores de despliegue son escritos en XML y tienen información para asegurar que :

- Los componentes están conectados a través de protocolos adecuados y con soporte al middleware ;
- el middleware subyacente y la plataforma están configurados para proporcionar el nivel adecuado de apoyo a la configuración de los componentes (por ejemplo, en CORBA, esto podría incluir la configuración de POA) ;
- los servicios de sistemas distribuidos asociados están preparados para proporcionar el nivel adecuado de seguridad, soporte de transacciones y así

Caso de Estudio: Enterprise JavaBeans - EJB

- Es un modelo de componentes (beans) del lado del servidor
- Soporta la arquitectura de tres niveles
- EJB proporciona los siguientes roles:
  - Proveedor de componentes, desarrolla aplicaciones lógicas
  - Ensamblador de aplicaciones, ensambla los componentes en recipientes
  - Implementador, asegura que una aplicación pueda ser desarrollada en un ambiente operacional
  - Proveedor de servicios, proporciona servicios en el SD
  - Proveedor de persistencia, correspondencia entre los datos a la BD
  - Proveedor de recipientes, configurar los recipientes
  - Administrador de sistemas, monitoreo y hacer ajustes para asegurar operaciones correctas

El modelo de componentes de EJB

EJB proporciona dos estilos de componentes:

- **Componentes de sesiones**: implementan una tarea en particular dentro de la lógica de la aplicación de un servicio, ejm realizar una compra en un comercio electrónico (estado, sin estado, persistencia,...)
- **Componentes orientados al mensajes**: clientes interactúan con el componente de sesión usando una invocación remota o local.
  - Orientado a soportar comunicación indirecta, en especial por medio de colas de mensajes.
  - EJB proporciona una interfaz de estilo escucha (***listener***) que refleja la naturaleza orientada a eventos del componente asociado

POJOs y anotaciones

- Enterprise JavaBeansPOJOs (plain old Java objects)
  - Un POJO es una instancia de una clase que no extiende ni implementa nada en especial. Beans es un POJO plano con anotaciones
  - Enterprise Java Bean (EJB) es un componente de negocio Java Enterprise, y para su ejecución necesita un contenedor EJB/J2EE (JBoss, WAS, OAS, ...)
- Anotaciones: una forma de metadatos, proporcionan datos sobre un programa que no es parte del programa en sí. No tienen ningún efecto directo sobre el código.

  Ejm *@TransactionManagement (Bean) @TransactionManagement (Container) @TransactionAttribute(Required)*

Atributos de Transacciones en EJB.

Attribute	*Policy*
REQUIRED	If the client has an associated transaction running, execute within this transaction; otherwise, start a new transaction.
REQUIRES_NEW	Always start a new transaction for this invocation.
SUPPORTS	If the client has an associated transaction, execute the method within the context of this transaction; if not, the call proceeds without any transaction support.
NOT_SUPPORTED	If the client calls the method from within a transaction, then this transaction is suspended before calling the method and resumed afterwards – that is, the invoked method is excluded from the transaction.
MANDATORY	The associated method must be called from within a client transaction; if not, an exception is thrown.
NEVER	The associated methods must not be called from within a client transaction; if this is attempted, an exception is thrown.

Dependencia e Intercepciones

- Dependencia: los contenedores son responsables de manejar y resolver las relaciones entre los componentes y sus dependencias
- Intercepción de métodos:
  - De llamadas a métodos asociados con interfaz de negocio: ejm, identificarse (logging) antes de una operación de compra o prevenir que ciertos usuarios realizen compras
  - Eventos del ciclo de vida: ejem, creación o eliminación de componentes

Métodos asociados con Contexto de Invocaciones en EJB

Signature	*Use*
public Object getTarget()	Returns the bean instance associated with the incoming invocation or event
public Method getMethod()	Returns the method being invoked
public Object[] getParameters()	Returns the set of parameters associated with the intercepted business method
public void setParameters(Object[] params)	Allows the parameter set to be altered by the interceptor, assuming type correctness is maintained
public Object proceed() throws Exception	Execution proceeds to next interceptor in the chain (if any) or the method that has been intercepted

Fractal

- Fractal: es un modelo de componentes ligeros que proporciona los beneficios de una programación basada en componentes para desarrollar SD
- Soporta programación con interfaces
- Proporciona una representación explicita de la arquitectura de software del sistema, evitando las dependencias implicitas
- Apuesta por el minimalismo, sin soporte a funcionalidades adicionales del componente

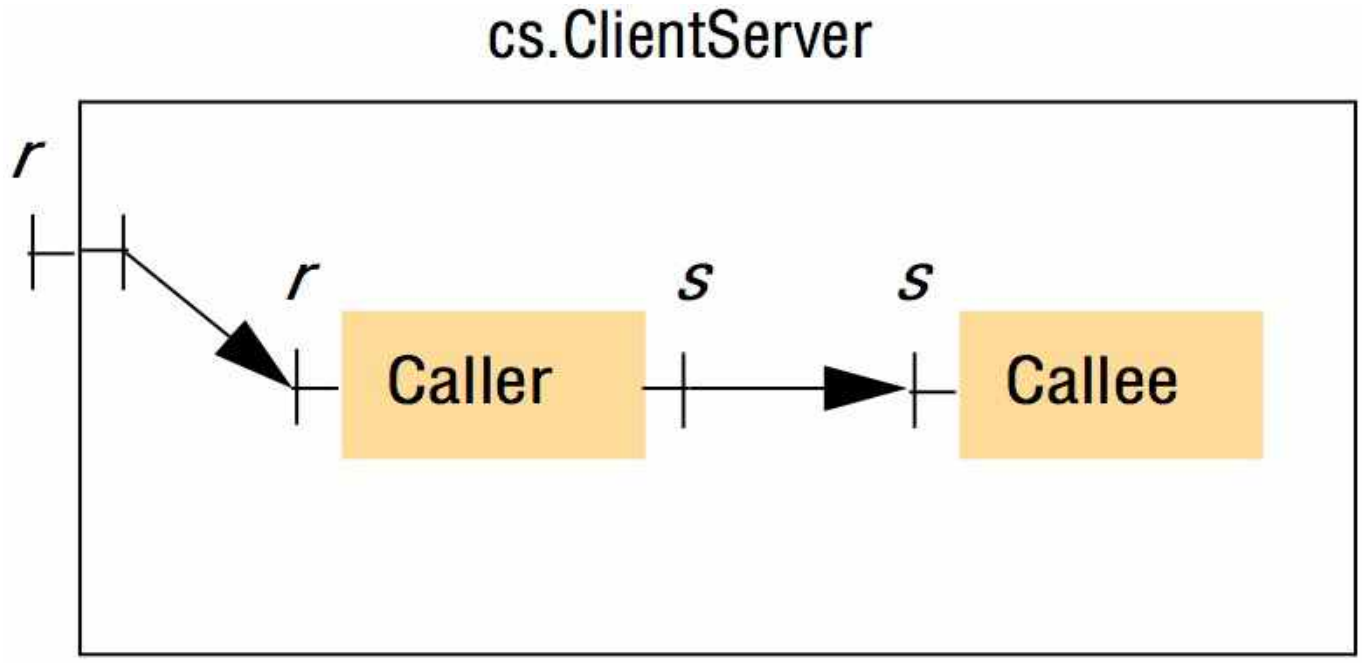
Fractal: Modelo de Programación

- Fractal define un modelo de programación independiente del idiomas, incluyendo:
  - Julia y AOKell (en Java, el último ofrece soporte para programación orientada a aspecto)
  - Cecilia y Think (basado en C) ;
  - FracNet (de .NET) ;
  - FracTalk (de Smalltalk) ;
  - Julio (basado en Python) .
  - Julia y Cecilia son tratados como la referencia para la implementación en Fractal
- Fractal es un estándard (www.ow2.org), open-source

Fractal: Modelo del Componente

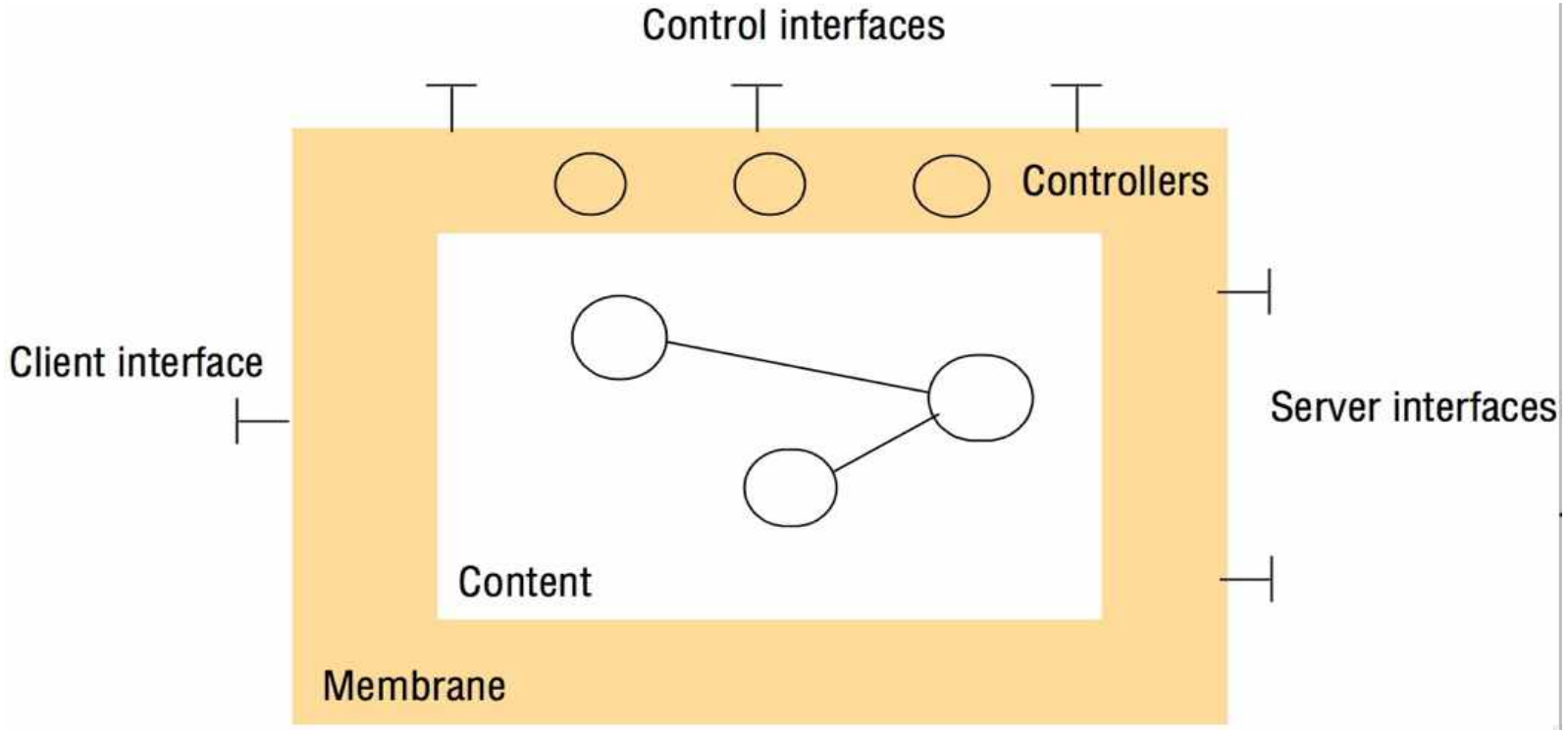
- Interfaz del servidor: soporta invocaciones de operaciones
- Interfaz del cliente: soporta realización de invocaciones
- Enlaces:
  - Enlaces primitivos: correspondencia directa entre las interfaces del cliente y el servidor
  - Enlaces compuestos: arquitectura de software compleja que implementa comunicaciones entre interfaces diferentes en máquinas diferentes. Ejm: CORBA y Fractal

Un ejemplo de configuración de un componente con Fractal



- Modelo de componente es jerárquico
- Usa XML (Fractal ADL, arquitecture definition language)

Estructura de un componente Fractal



- Membrana: capacidades de control asociada con los componentes por medio de controladores y asocian con el contenido
- Interfaz pueden ser internas o externas
- Controladores: manejar ciclo de vida, reflexión-introspección, intercepción

Fractal

http://fractal.ow2.org/

http://fractal.ow2.org/tutorials/helloworld.html