

SISTEMAS OPERATIVOS

TRABAJO MONOGRÁFICO REALIZADO POR

DANIELA NOEMÍ ORTIZ

COMO ADSCRIPTA A LA ASIGNATURA

“SISTEMAS OPERATIVOS”

ABRIL – 2003

“EL CASO DE ESTUDIO CORBA”

EL CASO DE ESTUDIO CORBA

INTRODUCCIÓN

CORBA es un diseño de *middleware* que permite que los programas de aplicación se comuniquen unos con otros con independencia de sus lenguajes de programación, sus plataformas hardware y software, las redes sobre las que se comunican y sus implementadores.

Las aplicaciones se constituyen mediante objetos CORBA, que implementan interfaces definidas en el lenguaje de definición de interfaces de CORBA, IDL.

Los clientes acceden a los métodos de las interfaces de los objetos de CORBA mediante RMI. El componente *middleware* que da soporte a RMI se denomina *Object Request Broker* (Intermediario de Petición de Objetos) u ORB.

La especificación de CORBA ha sido promovida por los miembros del *Object Management Group* (OMG). Se han implementado versiones muy diferentes de ORB, que soportan variedad de lenguajes de programación.

Los servicios CORBA proporcionan un conjunto genérico que puede utilizarse en una gran variedad de aplicaciones; estos servicios incluyen el Servicio de Nombres, los Servicios de Eventos y Notificación, el Servicio de Seguridad, los Servicios de Transacción y Concurrencia y el Servicio de Comercio.

El OMG (*Object Management Group*) fue formado en 1989 con la perspectiva de promocionar la adopción de sistemas de objetos distribuidos para conseguir los beneficios de la programación orientada al objeto para el desarrollo del software y la utilización sobre los sistemas distribuidos, que comenzaban entonces a difundirse.

Para lograr sus objetivos el OMG abogaba por el uso de sistemas abiertos basados en interfaces estándar orientadas al objeto. Estos sistemas podrían construirse partiendo de una base heterogénea de hardware, redes de computadores, sistemas operativos y lenguajes de programación.

Una motivación importante fue el permitir que los objetos distribuidos fueran implementados en cualquier lenguaje de programación y que fueran capaces de comunicarse uno con otro. Así, diseñaron un lenguaje de interfaz independiente de cualquier lenguaje específico de implementación. Se introdujo una metáfora, el Intermediario de petición de objetos cuyo papel es ayudar al cliente a invocar un método de un objeto. Esta función conlleva a localizar el objeto, activar el objeto si fuera necesario y después comunicar la petición del cliente hacia el objeto, que la realiza y responde al cliente.

En 1991 un grupo de compañías se pusieron de acuerdo en una especificación de una arquitectura de intermediario de petición de objetos, conocida como CORBA (Arquitectura Común de Intermediario de Petición de Objeto).

A ésta le siguió, en 1996, la especificación CORBA 2.0 [OMG 1998a], que definió estándares para permitir la comunicación entre implementaciones realizadas por desarrolladores diferentes. Estos estándares se denominan *General Inter.-ORB Protocol* (Protocolo General Inter.-ORB) o GIOP. Se pretendía que este GIOP pudiera ser implementado sobre cualquier capa de transporte con conexiones. La Implementación de GIOP para Internet utiliza TCP/IP y se denomina IIOP (Protocolo Inter.-ORB para Internet).

Los principales componentes del esquema RMI independiente del lenguaje CORBA son los siguientes:

- * Un Lenguaje de definición de interfaces conocido como IDL.
- * Una Arquitectura.
- * El GIOP define una representación de datos externa denominada CDR. También define formatos específicos para los mensajes de un protocolo petición - respuesta.
- * El IIOP define una forma estándar para las referencias a objetos remotos.

La arquitectura CORBA también permite servicios CORBA; un conjunto de servicios genéricos que pueden ser de utilidad para las aplicaciones distribuidas.

CORBA RMI

Para la programación en un sistema RMI multi-lenguaje, es necesario aprender los siguientes nuevos conceptos:

* El modelo de objeto que ofrece CORBA.

* El lenguaje de definición de interfaz y su correspondencia sobre el lenguaje de implementación.

En particular, el programador define interfaces remotas para los objetos remotos y después usa un compilador de interfaces para producir cada proxy y cada esqueleto correspondiente. Aunque en CORBA, el *proxy* se genera en el lenguaje del cliente y el esqueleto en el lenguaje del servidor.

• El modelo de objetos de CORBA: En este modelo los clientes no son objetos, necesariamente; un cliente podría ser cualquier programa que envíe mensajes de petición a los objetos remotos y reciba las respuestas.

El término objeto CORBA se emplea para referirse a los objetos remotos. Así, un objeto CORBA implementa una interfaz de un IDL, tiene una referencia a un objeto remoto y es capaz de responder a las invocaciones de los métodos en su interfaz. Se puede implementar un objeto CORBA en un lenguaje que no sea orientado al objeto. Dado que los lenguajes de implementación pueden tener diferentes nociones de clase, o incluso ninguna, en CORBA no existe el concepto de clase. Sin embargo, sí se pueden pasar como argumentos estructuras de datos de varios tipos y de complejidad arbitraria.

• CORBA IDL: Especifica un nombre y un conjunto de métodos que podrán utilizar los clientes.

Parámetros y resultados en CORBA IDL: Cada parámetro se marca como de entrada o de salida o ambos, mediante las palabras claves *in*, *out* o *inout*.

Estos podrán ser de uno cualquiera de los tipos primitivos: long, boolean, o uno de los tipos construidos struct o un array.

La semántica de paso de parámetros es:

Paso de Objetos CORBA: Cualquier parámetro cuyo tipo esté especificado mediante el nombre de una interfaz IDL.

Paso de tipos Primitivos y Construidos de CORBA: Los argumentos de tipos primitivos y construidos se copian y se pasan por valor. En el destino se crea un nuevo valor en el proceso receptor.

En el método *todasFormas* se combinan estas dos formas de paso de parámetros, cuyo valor devuelto es una cadena de tipo Forma: esto es una cadena de referencias a objetos remotos. El valor devuelto es una copia de la cadena en la que en cada uno de sus elementos hay una referencia a un objeto remoto.

Tipo Object: *Object* es el nombre de un tipo cuyos valores son referencias a objetos remotos. Su efecto es el de ser un Supertipo común de todos los tipos de interfaz IDL.

Excepciones en CORBA IDL: Permite que se definan excepciones en las interfaces y sean lanzadas por sus métodos.

Semántica de invocación: La invocación remota en CORBA define, por defecto, la semántica como máximo una vez. Sin embargo, IDL puede especificar que la invocación de un método concreto tenga semántica, puede ser mediante la palabra clave *Oneway*. El cliente no se bloquea en las peticiones *oneway*, que solamente puede emplearse para los métodos sin resultado.

• El Servicio de Nombres de CORBA: Es un enlazador que proporciona operaciones como *rebind* para que los servidores registren referencias a objetos remotos de objetos CORBA mediante su nombre y *resolve* para que los clientes las busquen mediante este nombre. Los nombres se estructuran al modo jerárquico, y cada nombre de una ruta se encuentra en una estructura denominada *Name-Component*.

Ejemplo: Son dos Interfaces IDL *Forma* y *ListaForma*.

```
struct Rectángulo{
    long anchura;
    long altura;
    long x;
    long y;
};
struct ObjetoGráfico{
    string tipo;
    Rectangle enmarcado;
    boolean estaRelleno;
};
interface Forma {
    long dameVersion ();
    ObjetoGráfico dameTodoEstado(); // devuelve el estado del ObjetoGráfico
};
typedef secuencia <Forma, 100> Todo;
interface ListaForma{
    exception ExcepcionLlena { };
    Forma nuevaForma(in ObjetoGráfico g) raises (ExcepcionLlena);
    Todo TodasFormas(); // devuelve una secuencia de referencias a objetos remotos
    long dameVersion ();
};
```

Este ejemplo muestra las definiciones de dos *struct*. Las secuencias y las cadenas se definen mediante *typedef*, que muestra una secuencia de elementos de tipo *Forma* de longitud 100.

La estructura *ObjetoGráfico* pasada como argumento produce una copia nueva de esta estructura en el servidor.

El parámetro de *nuevaForma* es un parámetro *in* que indica que el argumento debería pasarse desde el cliente al servidor en el mensaje de Petición.

- *Pseudo Objetos CORBA*: Las implementaciones de CORBA proporcionan algunas interfaces sobre las funciones del ORB que los programadores necesitan conocer. Estas se denominan pseudo-objetos dado que no pueden utilizarse como si fueran objetos CORBA. Tienen interfaces IDL y vienen implementadas en bibliotecas.

Ejemplo: Clase *SirvienteListaForma* del programa servidor para la interfaz CORBA *ListaForma*.

```
import org.omg.CORBA.*;
class SirvienteListaForma extends _ListaFormaImplBase{
    ORB elORB;
    private Forma laLista[];
    private int version;
    private static int n = 0;
    private SirvienteListaForma(ORB orb){
        elOrb = orb;
        // inicialización de las otras variables de instancia
    }
    public Forma nuevaForma(ObjetoGráfico g) throws ListaFormaPackage.ExcepcionLlena{
        version++;
        Forma f=new SirvienteForma(g versión);
        if(n >= 100) throw new ListaFormaPackage.ExcepcionLlena();
        laLista[n++]=f;
        elOrb.connect(f);
    }
}
```

```

return s;
}
public Forma[] todasFormas(){...}
public int dameVersion () {...}
}

```

Lo destacable de este ejemplo es que:

* ORB = es el nombre de una interfaz que representa la funcionalidad del ORB a la que necesitan acceder los programadores. Esta incluye:

- El método *init*, que deberá llamarse para inicializar el ORB.
- El método *connect*, que se emplean para registrar objetos CORBA en el ORB.
- También otros métodos que permiten conversiones entre referencias a objetos remotos y cadenas de texto.

* El método *nuevaforma* es un método factoría puesto que crea objetos CORBA.

Cuando un servidor crea una instancia de una clase sirviente, deberá registrarla frente al ORB que convierte la instancia en un objeto CORBA y le da una referencia a un objeto remoto. A menos que se haya hecho esto, no podrá recibir invocaciones remotas.

* Las etapas necesarias para producir programas clientes - servidor en CORBA que, emplean las interfaces IDL son:

- *Programa Servidor*: El programa deberá contener implementaciones de una o más interfaces IDL. Para un servidor escrito en un lenguaje orientado al objeto como es Java o C++, estas implementaciones vienen dadas en las clases sirvientes y los objetos CORBA son instancias de estas clases.

- * *Las clases sirvientes*: Cada clase sirviente extiende la clase esqueleto correspondiente e implementa una interfaz IDL empleando las signaturas de los métodos que se definen en la interfaz Java equivalente. Las clases sirvientes que implementa la interfaz *ListaForma* se denominan *SirvienteListaForma*.

- * *El servidor*: Primero crea e inicializa el ORB, posteriormente crea una instancia de *sirvientelistaforma*, que es un objeto Java, lo convierte en un objeto CORBA al registrarlo frente al ORB e invoca la operación *rebind* para registrar al servidor frente al Servicio de Nombres es entonces cuando se espera las peticiones de los clientes.

Los servidores al hacer uso del Servicio de Nombres primero obtienen un contexto raíz después crean un *namecomponent*, definen una ruta y finalmente emplean el método *rebind* para registrar el nombre y la referencia al objeto remoto. Los clientes llevan a cabo los mismos pasos pero emplean el método *resolve*.

- *Programa Cliente*: Crea e inicializa un ORB, entonces contacta con el Servicio de Nombres para obtener una referencia a un objeto remoto *listaforma* mediante su método *resolve*. Tras ello, invoca su método *todasformas* para conseguir una secuencia de referencia a objetos remotos de todas las formas que posee el servidor en la actualidad. Después invoca el método *dametodoestado* dándole como referencia al primer objeto remoto de la secuencia obtenida; el resultado se proporciona como una instancia de la clase *objetográfico*.

El método *dametodoestado* parece afirmar de que es posible pasar objetos por valor en CORBA, dado que tanto el cliente como el servidor trabajan con instancias de la clase *objetográfico*. Sin embargo no es tan así: el objeto CORBA devuelve un elemento *struct*, y los clientes que emplearan otro lenguaje diferente lo verían también de forma diferente.

Los programas cliente deberían atrapar siempre las excepciones CORBA *systemexception*, que informan de los errores debidos a la distribución. Los programas clientes también deberían atrapar a las excepciones definidas en la interfaz IDL.

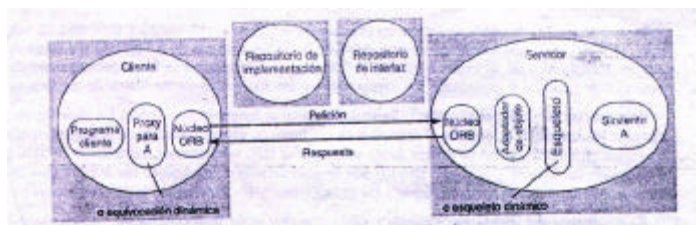
- *Devoluciones de Llamadas*: Pueden implementarse en CORBA de modo similar a la *interfazretrollamadatablero*. Esta interfaz se implementa como un objeto CORBA por parte del cliente, permitiendo que el servidor envíe al cliente un número de versión cuando se añadan objetos nuevos. Pero antes de que el servidor pueda hacer esto, el cliente necesita informar al servidor de la referencia a este objeto remoto. Para posibilitar esto, la interfaz requiere métodos adicionales tales como *registra* y *desregistra*.

Después de que el cliente haya obtenido una referencia al objeto y creado una instancia de *RetrollamadaTablero*, el método registra del objeto para informar al servidor de que está interesado en recibir retrollamadas. El objeto del servidor es el responsable de mantener una lista de clientes interesados y de notificar a todos ellos cada vez que se incrementa el número de versión al añadir un objeto nuevo. El método *retrollamada* se declara como *oneway* de modo que el servidor puede utilizar llamadas asíncronas para evitar retrasos cuando notifica a cada cliente.

LA ARQUITECTURA DE CORBA

La arquitectura está diseñada para dar soporte al papel de un intermediario de peticiones de objetos que permita que los clientes invoquen métodos en objetos remotos, donde tanto los clientes como los servidores puedan implementarse en variedad de lenguajes de programación.

La arquitectura CORBA contiene tres componentes mayores: el adaptador de objeto, el repositorio de implementación y el repositorio de interfaz. Que se muestran a continuación.



CORBA distingue entre invocaciones estáticas y dinámicas. Las invocaciones estáticas se emplean cuando se conoce en tiempo de compilación la interfaz remota del objeto CORBA, lo que permite emplear un resguardo de cliente y un esqueleto de servidor. Si la interfaz remota no es conocida en tiempo de compilación, debe emplearse una invocación dinámica.

El núcleo de ORB: Proporciona una interfaz que incluye lo siguiente:

- Operaciones que permiten su arranque y parada.
- Operaciones para la conversión entre referencias a objetos remotos y cadenas de texto.
- Operaciones para obtener listas de argumentos para las llamadas que emplean invocación dinámica.

El adaptador de objetos: Su papel es servir de puente sobre el hueco que media entre los objetos con interfaces IDL y las interfaces del lenguaje de programación de las correspondientes clases sirviente. Este papel también cubre la existente entre la referencia remota y los módulos de despacho. Este tiene los siguientes cometidos:

- Crea referencias a objetos remotos para los objetos CORBA.
- Despacha cada RMI vía un esqueleto hacia el sirviente apropiado.
- Activa objetos.

Un adaptador de objeto da a cada objeto CORBA un único *nombre de objeto*, que forma parte de su referencia a objeto remoto. Cada vez que se activa un objeto se empleará este mismo nombre. El nombre de objeto puede venir especificado por el programa de aplicación o ser generado por el adaptador de objeto. Cada objeto CORBA se registra frente a su adaptador de objeto, que puede mantener una tabla a objetos remotos que relaciona los nombres de objetos CORBA con sus sirvientes.

Cada adaptador de objeto tiene su propio nombre, que también forma parte de las referencias a objeto remoto de todos los objetos CORBA que gestiona. También este nombre puede venir especificado por el programa de aplicación o generarse automáticamente.

Esqueletos: Las clases esqueleto se generan en el lenguaje del servidor mediante un compilador de IDL. Las invocaciones de métodos remotos se despachan mediante el esqueleto apropiado a un sirviente concreto, y el esqueleto desempaqueta los argumentos desde los mensajes de petición y empaqueta las excepciones y los resultados en mensaje de respuesta.

Resguardo / proxy del cliente: Se encuentran en el lenguaje cliente. La clase de un *proxy* o un conjunto de procedimientos de resguardo se genera desde una interfaz IDL, mediante un compilador IDL para el lenguaje del cliente. Cada uno empaqueta los argumentos de los mensajes de invocación y desempaqueta las excepciones y los resultados de las respuestas.

Repositorio de implementación: Cada uno es responsable de activar los servidores registrados bajo demanda y localizar los servidores que están en ejecución en cada momento. Para hacer referencia a los servidores, cuando se registran y se activan se emplea el nombre del adaptador del objeto.

Un repositorio de implementación almacena la correspondencia de los nombres de ciertos adaptadores de objetos con las rutas de los archivos que contienen los objetos de la implementación. Los objetos de implementación y los nombres del adaptador de objetos se registran usualmente frente al repositorio de implementación cuando se instalan los programas servidores. Cuando se activan las implementaciones de los objetos en los servidores, se añade a la relación el nombre del computador y el número de puerto donde se ubica el servidor.

Entrada del repositorio de implementación:

| Nombre de adaptador de objeto. | Ruta de implementación de objeto | Nombres host y número de puerto del servidor |
|--------------------------------|----------------------------------|--|
|--------------------------------|----------------------------------|--|

Un repositorio de implementación permite almacenar generalmente cierta información extra sobre cada servidor, por ejemplo: información de control de acceso sobre quién está capacitado para activarlo o para invocar sus operaciones. En él es posible replicar información para proporcionar disponibilidad o tolerancia a fallos.

Repositorio de interfaz: Su papel es proporcionar información sobre las interfaces IDL registradas a los clientes y a los servidores que lo requieran. Para una interfaz de un cierto tipo puede aportar los nombres de los métodos y para cada método los nombres y los tipos de los argumentos y las excepciones. Así, el repositorio de interfaz añade posibilidades de reflexión a CORBA.

El compilador de IDL asigna un identificador de tipo único a cada tipo IDL de cada interfaz que compila.

En las referencias a objetos remotos se incluye un identificador de tipo para los objetos CORBA de ese tipo.

Este identificador se denomina ID del repositorio porque puede usarse como clave para las interfaces IDL registradas en el repositorio de interfaz. No todos los ORB proporcionan un objeto de interfaz.

Interfaz de invocación dinámica: En algunas aplicaciones, puede que un cliente sin la clase *proxy* apropiada necesite invocar un método de un objeto remoto.

Esta permite que los clientes realicen invocaciones dinámicas sobre objetos remotos CORBA. Se emplea cuando no es práctico emplear proxies. El cliente puede obtener información necesaria sobre los métodos disponibles para un objeto CORBA dado desde el repositorio de interfaz. El cliente puede utilizar esta información para construir una invocación.

Interfaz de esqueleto dinámica: Esta permite a un objeto CORBA aceptar invocaciones sobre una interfaz para la que no hay un esqueleto, posiblemente porque no se conocía el tipo de su interfaz en tiempo de compilación. Cuando un esqueleto dinámico recibe una invocación, inspecciona los contenidos de la petición para descubrir el objeto destino, el método que se invoca y los argumentos.

Código de legado: Este término se refiere al código existente que no fue diseñado previendo los objetos distribuidos. Se puede convertir un fragmento de código de legado en un objeto CORBA definiendo una interfaz IDL para él y proporcionando la implementación de un adaptador de objeto y los esqueletos.

LENGAJE DE DEFINICIÓN DE INTERFAZ DE CORBA

Este ofrece mecanismos para definir módulos, interfaces, tipos atributos y firmas de métodos. IDL tiene las mismas reglas léxicas que C++ pero tiene palabras adicionales para dar soporte a la distribución, por

ejemplo: *any*, *attribute*, *in*, *out*, etc. La gramática de IDL es un subconjunto de ANSI C++ con constructores adicionales para soportar firmas de métodos.

Módulos de atributos: La construcción módulo permite agrupar en unidades lógicas las interfaces y otras definiciones de tipo IDL. Module define un alcance léxico, que previene la colisión de los nombres definidos en el interior de un módulo con los nombres definidos fuera de él.

Interfaces IDL: Describe los métodos disponibles en los objetos CORBA que implementan esa interfaz. Podrían diseñarse clientes de un objeto CORBA partiendo del único conocimiento de esta interfaz IDL. Una interfaz define un conjunto de operaciones y atributos y depende generalmente de un conjunto de tipos definidos como ésta.

Métodos IDL: La forma general de la signatura de un método es:

```
[oneway] <tipo_devuelto> <nombre_método> ( parámetro I, ... , parámetro L)
  [raíces (except1, ... , exceptN)] [contexto (nombre1, ... , nombreM)]
```

donde las expresiones entre corchetes son opcionales. Como ejemplo de un método que contenga sólo las partes necesarias, considere:

```
void damePersona (in string nombre, out Persona p);
```

La expresión *oneway* indica que el cliente que invoca el método no se bloqueará mientras; el objeto destino lleva a cabo el método. Además, las invocaciones *oneway* se realizan con la semántica de llamadas pudiera ser, vemos el siguiente ejemplo:

```
oneway void retrollamada(in int version);
```

En este ejemplo, donde el servidor llama al cliente cada vez que se añade una nueva forma, la pérdida ocasional de una petición no es un problema para el cliente, dado que la llamada indica exactamente que es improbable que se pierda el número de la última versión y las llamadas subsiguientes.

La expresión opcional *raises* indica excepciones definidas por el usuario que pueden lanzarse para terminar la ejecución del método. Por ejemplo, considere el siguiente ejemplo:

```
exception ExcepcionLlena { };
Forma nuevaForma (in ObjetoGrafico g) raises (ExcepcionLlena);
```

El método *nuevaForma* especifica mediante la expresión *raises* que puede lanzar una excepción llamada *ExcepcionLlena*, definida dentro de la interfaz *ListaForma*; pueden definirse excepciones que contengan variables.

Tipos IDL construidos:

| Tipo | Ejemplos | Uso |
|-----------------|--|---|
| Sequence | <pre>typedef sequence <Forma, 100> Todo; typedef sequence <Forma> Todo; secuencias de Formas con límite y sin límite</pre> | Define un tipo para una sec. de longitud vble. de elementos de un tipo IDL especificado. Puede especificarse una talla superior límite. |
| string | <pre>string nombre; typedef string <8> CadenaCorta, secuencias de caracteres con límite</pre> | Define secuencias de caracteres terminadas en el carácter nulo. Puede especificarse una talla superior límite. |

| | | |
|-------------------|---|--|
| array | typedef octet idUnico [12]; typedef ObjetoGrafico OG[10][8]; | Define un tipo para una secuencia multidimensional de talla fija de elementos de un tipo IDL especificado. |
| record | struct ObjetoGrafico { struct tipo; Rectangulo enmarcado; boolean estaRelleno; }; | Define un tipo para un registro que contiene un grupo de entidades relacionadas. Cada <i>struct</i> se pasa por valor en los argumentos y en los resultados |
| enumerated | enum Arbitr (Exp Numero, Nombre); | El tipo enumerado en IDL hace corresponder un nombre de tipo sobre un pequeño conjunto de valores enteros. |
| union | union Exp switch (Arbitr) case Exp: string voto; case Numero: long n; case Nombre: string s; | La unión IDL discriminada permite pasar como argumento uno de un conjunto de tipos dado. La cabecera está parametrizada por un <i>enum</i> . que especifica qué miembro está en uso |

Cuando se lanza una excepción que contiene variables, el servidor puede utilizar las variables para devolver información al cliente acerca del contexto de la excepción.

CORBA puede producir también excepciones de sistema sobre los problemas con los servidores; tales como estar ocupados o la imposibilidad de ser activados, problemas con la comunicación y problemas del lado del cliente. Los programas cliente deberían gestionar las excepciones de usuario y de sistema. La expresión opcional *context* se utiliza para aportar relaciones entre cadenas de nombres y cadenas de texto de valores.

Tipos IDL: IDL soporta quince tipos primitivos, que incluyen short (16 bits), long (32 bits), unsigned short, unsigned long, float (32 bits), double (64 bits), char, boolean (TRUE, FALSE), octet (8 bits), y any (que podrá representar cualquier tipo primitivo o construido). Mediante la palabra clave *const*, es posible declarar constantes de la mayoría de los tipos primitivos y constantes de cadenas de texto. IDL proporciona un tipo especial llamado *Object*, cuyos valores son referencias a objetos remotos. Si un parámetro o el resultado es de tipo *Object*, entonces el argumento correspondiente podría referirse a cualquier objeto CORBA

Todas las cadenas o secuencias que se empleen como argumentos deben definirse en un *typedef*. Ninguno de los tipos de datos primitivos o construidos podrán contener referencias.

Atributos: Los atributos son como los campos clase públicos en Java. Los atributos se pueden definir como *readonly* (sólo lectura), donde se necesite. Los atributos son privados a los objetos CORBA, pero se genera un par de métodos de acceso, para cada atributo declarado, de modo automático por el compilador de IDL.

Herencia: Las interfaces IDL se pueden extender. Por ejemplo, si la interfaz B extiende la interfaz A, quiere decir que pudiera haber añadido tipos, constantes, excepciones, métodos y atributos nuevos a los de A. Una interfaz extendida puede redefinir tipos, constantes y excepciones, pero no podrá redefinir métodos; un valor de un tipo extendido es válido como valor de parámetro o resultado del tipo padre. Por ejemplo: el tipo B es válido como valor de un parámetro o resultado del tipo A.

```
Interface A { };
Interface B: A { };
Interface Z: B, C { };
```

Además, una interfaz IDL podrá extender más de una interfaz. Por ejemplo la interfaz Z, extiende B y C (aparte de aquellos que redefine).

Cuando una interfaz como Z, extiende más de una interfaz, existe la posibilidad de que pueda heredar un tipo, constante o excepción con el mismo nombre desde dos interfaces diferentes.

IDL no permite que una interfaz herede métodos o atributos con nombres comunes desde dos interfaces diferentes.

Todas las interfaces IDL heredan del tipo *Object*, lo que implica que todas las interfaces IDL son compatibles con el tipo *Object*. Esto posibilita definir operaciones IDL que puedan tomar como argumento *devolver*, como resultado una referencia a un objeto remoto de cualquier tipo.

Nombres de tipo IDL: Por ejemplo, el tipo IDL para el tipo *Forma* de la interfaz podría ser:

IDL: Tablero/ Forma: 1.0

Este ejemplo ilustra las tres partes del nombre de un tipo IDL; el prefijo IDL, un nombre de tipo y un número de versión para prefijar una cadena de texto adicional al nombre del tipo con el objeto de distinguir sus propios tipos de los del resto de desarrolladores. Los nombres de tipo IDL de las interfaces se incluyen en las referencias a objetos remotos.

Extensiones a CORBA: Se han añadido algunas características nuevas de especificación de CORBA. Estas incluyen la posibilidad de pasar objetos no-CORBA por valor y una variante asíncrona de RMI.

Objetos que pueden pasarse por valor: los argumentos y los resultados IDL de tipos primitivos o contruidos, mientras que aquellos que se refieren a objetos CORBA se pasan por referencia.

Se ha añadido a CORBA el soporte para pasar por valor objetos no-CORBA. Un *valuetype* es un *struct* con firmas de métodos adicionales. Los argumentos y resultados *valuetype* se pasan por valor: esto es, se pasa el estado al lugar remoto y se emplea para producir un nuevo objeto en el destino. Los métodos de éste nuevo objeto pueden ser invocados localmente, provocando que su estado pueda diferir del estado del objeto original. El pasar la implementación de los métodos no es algo sencillo dado que el cliente y el servidor pueden usar lenguajes diferentes. Sin embargo, si el cliente y el servidor están implementados en Java, el código podrá descargarse. Para ser implementado en C++, el código requerido deberá estar ya presente tanto en el cliente como en el servidor.

RMI asíncrono: El RMI asíncrono de CORBA se adapta bien para su uso en entornos donde los clientes pueden quedar desconectados temporalmente; la especificación de mensajería de CORBA [OMG 1998d] propone una forma alternativa de RMI para abastecer a aquellos clientes susceptibles de desconectarse temporalmente.

REFERENCIAS A OBJETOS REMOTOS EN CORBA

CORBA-2.0 especifica un formato para las referencias a objetos remotos que es adecuada para su uso, tanto si el objeto es activable remotamente como si no. Las referencias que utilizan este formato se denominan referencias a objetos remotos interoperables IOR. La siguiente figura contiene el detalle del IOR.

Formato de IOR:

| Nombre de tipo de interfaz. | Protocolo y dirección detallada | | | Clave de objeto | |
|--|---------------------------------|----------------------------|------------------|---------------------|-------------------|
| Identificador de repositorio de interfaz | IIOP | Nombre de dominio del host | Número de puerto | Nombre de adaptador | Nombre de objeto. |

- El primer campo de un IOR especifica el nombre de tipo de la interfaz IDL del objeto CORBA. Si el ORB tiene un repositorio de interfaz, este nombre de tipo es también el identificador de la interfaz IDL en el repositorio de interfaz.
- El segundo campo especifica el protocolo de transporte y los detalles necesarios para que ese protocolo de transporte identifique al servidor. El protocolo Internet Inter.-ORB (IIOP) emplea TCP/ IP .
- El tercer campo es empleado por el ORB para identificar un Objeto CORBA.

- *IOR transitorio*: El núcleo del ORB del servidor recibe el mensaje de petición que contiene el nombre del adaptador de objeto y el nombre del destino. Emplea el nombre del adaptador del objeto para localizar el sirviente. El IOR transitorio de un objeto CORBA dura sólo el tiempo en que dura el proceso que lo aloja, éste contiene los detalles de las direcciones al servidor que contiene el objeto CORBA.
- *IOR persistente*: Perdura entre las activaciones de los Objetos CORBA y contendrá los detalles de las direcciones del repositorio de implementación. Este recibe la petición, extrae el nombre del adaptador de objeto del IOR de la petición. Dado que el nombre del adaptador de objeto está en su tabla, intenta, si es necesario, activar el objeto CORBA de la dirección del *host* especificado.

En ambos casos, el ORB cliente envía el mensaje de petición al servidor cuya dirección proporciona el IOR.

Una vez activado el objeto CORBA, el repositorio de implementación devuelve su dirección detallada al ORB del cliente, que lo emplea como destino de los mensajes de petición RMI, lo que incluye el nombre del adaptador de objeto y el nombre del objeto. Esto permite al núcleo del ORB del servidor encontrar el adaptador de objeto, el cual emplea el nombre del objeto para localizar al sirviente, como antes.

El segundo campo es un IOR, que especifique el nombre de dominio del *host* y el número de puerto, puede aparecer varias veces con el fin de permitir que un objeto o un repositorio de implementación esté replicado en varias ubicaciones diferentes.

El mensaje de respuesta en el protocolo petición-respuesta incluye un encabezamiento con información que permite llevar a cabo el mecanismo de IOR persistente. En particular, incluye una entrada de estatus que puede indicar si la petición debiera dirigirse a un servidor diferente, en cuyo caso el cuerpo de la respuesta incluye un IOR que contiene la dirección del servidor del objeto recientemente activado.

CORRESPONDENCIA CON EL LENGUAJE EN CORBA

Los tipos primitivos en IDL se corresponden con tipos primitivos en Java. Las secuencias y las cadenas en IDL, se corresponden con las cadenas (*array*) en Java. Una excepción de IDL se corresponde con una clase Java que proporciona variables de instancia a los campos de la excepción y los constructores. La relación con C++ es igualmente directa.

Sin embargo, aparecen algunas dificultades con la correspondencia en la semántica de paso de parámetros de IDL con Java. En particular, IDL permite que los métodos devuelvan varios valores separados vía parámetros de salida, mientras que Java sólo puede tener un solo resultado. Para soslayar esta diferencia se proporcionan clases *Holder* (propietario), que aun así requiere que el programador haga un uso explícito de ellas.

A pesar de que las implementaciones en C++ de CORBA pueden gestionar parámetros *out e inout* de una manera bastante natural, los programadores de C++ padecen un diferente conjunto de problemas; con los parámetros, relacionado con la gestión de almacenamiento.

Cuando un método del servidor ha acabado su ejecución, los argumentos out y los resultados se liberan y el programador deberá duplicarlos si aún los necesitara.

Las referencias a objeto pasadas a los clientes deberán liberarse cuando no se vayan a necesitar, lo mismo para los parámetros de longitud variable.

En general los programadores que utilicen IDL no sólo tendrán que aprender la notación IDL en sí, sino que también tendrán que comprender cómo se correlacionan sus parámetros con los parámetros del lenguaje de implementación.

SERVICIOS DE CORBA

CORBA incluye especificaciones para los servicios que pudieran necesitarse en los objetos distribuidos.

El **Servicio de Nombres** permite buscar los objetos CORBA por nombre.

El **Servicio de Comercio** permite que sean localizados por sus atributos (es un servicio de directorio). Su base de datos contiene una relación de tipos de servicio y sus atributos asociados con referencias a objetos remotos de objetos CORBA. Este tipo de servicio es un nombre, y cada atributo es un par nombre-valor. Los clientes hacen consultas especificando restricciones sobre los valores de los atributos, y sus preferencias del orden en que quieren recibir las ofertas concordantes.

El **Servicio de Transacción de Objetos** (*Object Transaction Service*) permite que los objetos distribuidos CORBA participen en transacciones tanto sencillas como anidadas. El cliente especifica una transacción, como una secuencia de llamadas RMI, que comienzan por *begin* (comenzar) y terminan por *commit* (consumar) o *rollback* (abortar, retractar). El ORB adhiere un identificador de transacción a cada invocación remota y trata con las peticiones *begin*, *commit*, *rollback* (*abort*). Los clientes también pueden suspender y retomar las transacciones. El servicio de transacciones lleva a cabo un protocolo de consumación en dos fases.

El **Servicio de Control de Concurrencia** emplea bloqueos para aplicar el control de concurrencia al acceso a objetos CORBA. Puede utilizarse desde el interior de las transacciones o aisladamente.

El **Servicio de Objetos Persistentes**, según el cual los objetos persistentes pueden almacenarse en una forma pasiva en un almacén de objetos persistentes, mientras no están siendo usados, y pueden activarse cuando se los necesite. A pesar de que un ORB activa objetos CORBA con referencias a objetos persistentes, obteniendo sus implementaciones desde el repositorio de implementación, no se hacen responsables de almacenar y restaurar el estado de los objetos CORBA. En su lugar, cada objeto CORBA es responsable de guardar y recuperar su propio estado, por ejemplo mediante archivos. El *Servicio de Objetos Persistentes* (POS) de CORBA está dispuesto para servir como almacén de objetos persistentes de objetos CORBA. El POS puede implementarse de variedad de formas.

La arquitectura de POS permite que se disponga de un conjunto de almacenes de datos; cada objeto persistente tiene un identificador persistente que incluye la identidad de su almacén de datos y un número de objeto dentro de este almacén. Tanto el cliente como el objeto CORBA pueden decidir cuándo almacenar su estado, enviando una petición al POS. Para permitir que el cliente controle su persistencia, un objeto CORBA hereda una interfaz que incluye opciones para almacenarlo, restaurarlo o eliminarlo.

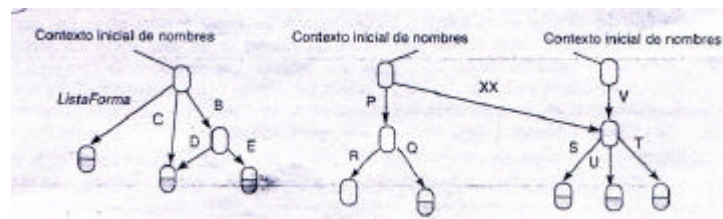
La implementación de un objeto CORBA persistente debe elegir un protocolo de comunicación con un almacén de datos.

La especificación POS ofrece una selección de protocolos alternativos para esta tarea, proporcionando un abanico de funcionalidades.

EL SERVICIO DE NOMBRES DE CORBA

El Servicio de Nombres de CORBA es un sofisticado ejemplo de enlazador. Permite enlazar nombres con referencias a objetos remotos de CORBA con contextos de nombres.

Un contexto de nominación es el alcance dentro del que se aplica un conjunto de nombres; cada uno de estos nombres en un contexto será único. Este es un ejemplo del servicio de Nombres de CORBA:



Los contextos pueden estar anidados para proporcionar un espacio de nombres jerárquicos, como se muestra en el gráfico en el que se ven los objetos CORBA al modo habitual, y los contextos como óvalos sencillos.

Un contexto inicial de nominación proporciona una raíz para un conjunto de enlaces. En la práctica, cada instancia de un ORB tiene un único contexto inicial de nominación, pero los servidores de nombres asociados con diferentes ORB pueden formar federaciones.

Los programas cliente y servidor solicitan el contexto inicial de nominación desde el ORB.

Dado que puede haber varios contextos iniciales, los objetos no tienen nombres absolutos: los nombres siempre se interpretan con respecto a un contexto inicial de nominación.

Para resolver un nombre con varios componentes el servicio de nombres busca en el contexto de arranque un enlace que concuerde con el primer componente. Si tal enlace existe, será bien una referencia a un objeto remoto o una referencia a otro contexto de nominación. Si el resultado es un contexto de nominación, se resuelve el segundo nombre en ese contexto. El procedimiento se repite hasta que se hayan resuelto todos los componentes del nombre y se obtenga una referencia a un objeto remoto, a menos que falle la concordancia por el camino.

Los nombres que emplea el *Servicio de Nombres* de CORBA son del tipo *NameComponent* y constan de 2 cadenas, una para el nombre y otra para la clase del objeto: este proporciona un solo atributo cuyo uso se previó para las aplicaciones y puede contener cualquier información descriptiva de utilidad.

Los clientes emplean el método *resolve* para buscar referencias a objetos por su nombre. El tipo del valor es *Object*. El tipo del resultado deberá ser estrechado, antes de poder ser utilizado para invocar un método en un objeto remoto de la aplicación.

El argumento de *resolve* es de tipo *Name*, que está definido como una secuencia de componentes de nombre. Esto viene a decir que el cliente debe construir una secuencia de componentes de nombre antes de realizar la llamada.

Los servidores de los objetos remotos emplean la operación *bind* para registrar los nombres de sus objetos y *unbind* para eliminarlos. La operación *bind* enlaza cierto nombre con una referencia a un objeto remoto y se invoca en el contexto en el que se añade el enlace.

La operación *bind-new-context* se usa para crear un contexto nuevo y para enlazarlo con el nombre dado en el contexto sobre el que fue invocada. Otro método llamado *bind_context* enlaza un nombre de contexto, dado a un nombre dado en el contexto sobre el que fue invocado. El método *unbind* puede usarse para contextos y nombres.

La operación *list* está preparada para ojear la información disponible en un contexto en el *Servicio de Nombres*. Devuelve una lista de enlaces desde el *NameContext* objetivo. Cada enlace consta de un nombre y un tipo; un objeto o un contexto. Algunas veces, un nombre de contexto puede contener un número muy grande de enlaces, en cuyo caso sería deseable que pudiera proporcionarlos como resultado de una sola invocación. Por esta razón, el método *list* devuelve cierto número máximo de enlaces como resultado de la llamada, si es el caso de que faltan enlaces por enviar, se puede arreglar para que envíe los resultados en tandas. Esto es posible dado que retorna un iterador como resultado secundario. El cliente usa el iterador para recuperar el resto de los resultados, uno cada vez.

El tipo *bindingList* es una secuencia de enlaces, cada uno de las cuales contiene un nombre y su tipo, que es bien un contexto o bien una referencia a un objeto remoto. El tipo *BindingIterator* proporciona un método *next_n* para acceder al próximo conjunto de enlaces; su primer argumento especifica cuántos enlaces se desean y en el segundo recibimos una secuencia de enlaces. El cliente llama al método *lista* dándole como primer argumento el número máximo de enlaces que se recibirán inmediatamente a través del segundo argumento. El tercer argumento es un iterador, que podrá ser usado para obtener el resto de los enlaces, si los hubiera.

El espacio de nombres de CORBA permite la federación de *Servicio de Nombres*, usando un esquema en el que cada servidor proporciona un subconjunto del grafo de nombres.

La implementación Java de *Servicio de Nombres* de CORBA es muy simple y se denomina transitoria porque almacena todos sus enlaces en memoria volátil.

SERVICIO DE EVENTOS DE CORBA

La especificación del *Servicio de Eventos* de CORBA define interfaces que permiten a los objetos de interés, denominados *proveedores*, comunicar notificaciones a sus subscriptores, llamados *consumidores*. Las notificaciones se comunican como argumentos o resultados de invocaciones síncronas ordinarias de métodos remotos CORBA.

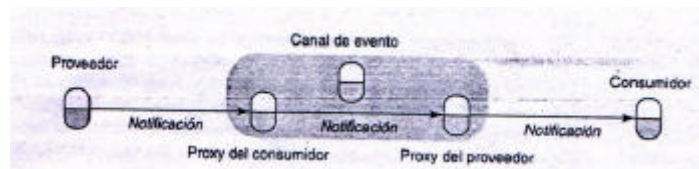
Las notificaciones pueden propagarse impulsadas (*pull*) por el *proveedor* hacia el *consumidor*, en este caso, el *consumidor* implementa la interfaz *PushConsumer* que incluye un método *Push* que toma cualquier tipo de dato CORBA como argumento. Los *consumidores* registran sus referencias a objetos remotos con los *proveedores*. Los *proveedores* invocan el método *push*, pasando una notificación como argumento. Lo pueden hacer como extraídas (*pull*) por parte del *consumidor* desde el *proveedor*. Este implementa la interfaz *PullSupplier*, que incluyen método *pull* que recibe cualquier tipo de dato CORBA como valor de retorno. Los *proveedores* registran sus referencias a objetos remotos frente a los *consumidores*. Los *consumidores* invocan el método *pull* y reciben como resultado una notificación.

La notificación en sí misma se transmite como un argumento o resultado cuyo tipo es *any*, que indica que los objetos que intercambian notificaciones deben haberse puesto de acuerdo sobre los contenidos de las notificaciones. Los programadores de aplicaciones sin embargo, podrán definir sus propias interfaces IDL con notificaciones de cualquier tipo deseado.

Los canales de eventos son objetos CORBA que pueden emplearse para permitir que múltiples proveedores se comuniquen con múltiples consumidores de modo asíncrono. Un canal de eventos actúa como búfer entre los proveedores y los consumidores. También puede multidifundir las notificaciones a los consumidores.

Los procesos proveedores en la aplicación ofrecen ellos mismos las suscripciones obteniendo los *proxy* de consumidor desde el canal de eventos y conectando los consumidores a ellos.

La presencia de *proxy de proveedores* y *proxy de consumidores* hace posible construir cadenas de canales de eventos en las que cada canal suministra notificaciones que serán consumidas por el siguiente canal.



Estos pueden programarse para llevar a cabo algunos de los papeles de los observadores. Sin embargo, las notificaciones no conllevan forma alguna de identificadores, y así los patrones de reconocimiento o de filtrado de notificación deberán basarse en la información de tipo puesta en las notificaciones por las aplicaciones.

SERVICIO DE NOTIFICACIÓN DE CORBA

El *Servicio Notificación* de CORBA [OMG 1998c] extiende el *Servicio de Eventos* de CORBA, conservando todas sus características, incluyendo canales de eventos. El servicio de eventos no proporciona ningún soporte para el filtrado de eventos o para especificar requisitos de reparto. Sin el uso de filtros, todos los consumidores conectados a un canal tendrán que recibir las mismas notificaciones que cualquier otro. Y sin la posibilidad de especificar requisitos de reparto, todas las notificaciones que se envíen a través de este canal tendrán las garantías de reparto incluidas en la implementación.

El servicio de notificación añade las siguientes nuevas funciones:

- Las notificaciones se pueden definir como estructuras de datos.

- Los consumidores de eventos pueden usar filtros que especifican exactamente en qué eventos están interesados. Los filtros pueden adherirse a cada proxy de un canal. Cada proxy dirigirá las notificaciones a los consumidores de eventos según las restricciones especificadas en los filtros.
- Los proveedores de eventos poseen mecanismos para descubrir en qué eventos están interesados los consumidores.
- Los consumidores de eventos pueden descubrir los tipos de eventos que ofrecen los proveedores en un canal, lo que les permite suscribirse a nuevos eventos según se hagan disponibles.
- Es posible configurar las propiedades de un canal, un proxy o un evento particular. Estas propiedades incluyen la confiabilidad del reparto de eventos, la prioridad de los eventos, el ordenamiento requerido y la política para descartar eventos almacenadas.
- Como extra adicional se da un repositorio de tipos de eventos. Proporciona acceso a la estructura, lo que le hace conveniente para definir restricciones de filtrado.

El tipo Evento Estructura presentado por el servicio de notificación proporciona una estructura de datos en la que encajan una gran variedad de tipos de notificaciones diferentes. Se pueden definir filtros en términos de las componentes de un tipo de Evento Estructurado, este consta de una cabecera de evento y un cuerpo de evento. La *cabecera* contiene:

| <i>Tipo de dominio</i> | <i>Tipo de evento</i> | <i>Nombre de evento</i> | <i>Requisito</i> |
|------------------------|-----------------------|-------------------------|----------------------|
| <<casa>> | <<alarma antirrobo>> | <<21/03, 2 p.m.>> | <<prioridad, 1.000>> |

El tipo de dominio se refiere al dominio de definición. El tipo de evento categoriza el tipo del evento de modo único dentro del dominio.

El nombre de evento identifica de modo único la instancia específica del evento que está siendo transmitido.

El resto de la cabecera contiene una lista de pares <nombre, valor>, proyectadas para ser usadas al especificar la fiabilidad y otros requisitos sobre el reparto de eventos.

El *cuerpo* de un evento estructurado contiene la siguiente información:

| <i>Parte Filtrable</i> | | | |
|--------------------------|-------------------------|----------------------|--------------|
| <i>Nombre, valor</i> | <i>Nombre, valor</i> | <i>Nombre, valor</i> | <i>Resto</i> |
| <<campana>>, <<sonando>> | <<puerta>>, <<abierta>> | <<gato>>, <<fuera>> | |

La primera parte del cuerpo del evento contiene una secuencia de pares <nombre, valor> proyectadas para su uso por filtros.

El resto del cuerpo del evento está propuesto para transmitir datos referentes a un evento particular por ejemplo, cuando la alarma antirrobo se apague, pudiera contener una fotografía digital del interior de las estancias.

Los objetos filtro se emplean desde los *proxy* para realizar decisiones sobre si encaminar o no cada notificación. Un filtro se diseña como una colección de restricciones, cada una de las cuales es una estructura con dos componentes:

- Una lista de estructuras de datos, cada una de las cuales indica un tipo de evento en términos de su nombre de dominio y tipo de evento.
- Una cadena de texto que contiene una expresión booleana que incluye los tipos de eventos ya listados.

SERVICIO DE SEGURIDAD DE CORBA

Este servicio incluye:

- Autenticación de principales (usuarios y servidores); generando credenciales para los principales (esto es certificados confirmando sus derechos).
Se puede aplicar control de acceso a los objetos CORBA cuando reciben invocaciones remotas.
- Auditoría de las invocaciones a métodos remotos por parte de los servidores.
- Medios para el no repudio. Cuando un objeto lleva a cabo una invocación remota en nombre de un principal, el servidor crea y almacena las credenciales que prueban que se hizo la invocación al servidor en representación del principal peticionario.

Para garantizar la aplicación correcta de la seguridad a las invocaciones de métodos remotos, el servicio de seguridad requiere cooperación en nombre del ORB. Para realizar una invocación segura a un método remoto, se envían las credenciales del cliente en el mensaje de petición.

Si las credenciales son válidas, se utilizan para decidir si el principal tiene derecho a acceder al objeto remoto utilizando el método del mensaje de petición. Esta decisión se hace consultando un objeto que contiene información sobre qué principales tienen derecho a acceder a cada método del objeto destino. Si el cliente tiene suficientes derechos, se lleva a cabo la invocación y se devuelve el resultado al cliente, junto con las credenciales del servidor, si fuera necesario. El objeto destino también podría almacenar los detalles de la invocación en un histórico o guardar las credenciales de no repudio.

CORBA permite especificar una variedad de políticas de seguridad según los requisitos. Una política de protección del mensaje declara si deben autenticarse el cliente o el servidor (o ambos).

Se proporciona a los usuarios un tipo especial de *credencial*, denominada *privilegio*, a la medida de sus funciones. Los objetos se agrupan en *dominios*. Cada dominio tiene una única política de control de acceso que especifica los derechos de acceso a los objetos del dominio de conjuntos de usuarios con privilegios concretos. Para dar cabida a una variedad impredecible de métodos, cada método se clasifica en términos de uno de cuatro métodos genéricos: 1) Los métodos *get* sólo devuelven partes del estado de un objeto, 2) los métodos *set* alteran el estado del objeto, 3) los métodos *use* hacen que el objeto realice algún trabajo, 4) los métodos *manage* realizan funciones especiales no proyectadas para este uso general. Dado que los objetos CORBA tienen una variedad de interfaces diferentes, hay que especificar los derechos de acceso de cada nueva interfaz en términos de los métodos genéricos anteriores. Esto implica que los diseñadores de la aplicación estarán involucrados en la aplicación de control de acceso, la determinación de los atributos de privilegio apropiados (por ejemplo: grupos o funciones) y en ayudar al usuario en la obtención de los privilegios apropiados para su tarea.

CONCLUSIONES

El principal componente de CORBA es el *Intermediario de Peticiones de Objetos (Object Request Broker)* u ORB; permite que clientes escritos en un lenguaje invoquen operaciones de objetos remotos (objetos CORBA) escritos en otro lenguaje.

El protocolo *General Inter-ORB (GIOP)* de CORBA incluye una representación externa de datos llamada CDR, que permite que los clientes y los servidores se comuniquen independientemente de su hardware. También incluye una especificación para un protocolo petición-respuesta independiente del sistema operativo subyacente.

El protocolo *Internet Inter-ORB (IIOP)* implementa el protocolo petición-respuesta sobre TCP/IP.

Un objeto CORBA implementa las operaciones de una interfaz IDL.

La arquitectura de CORBA permite activar los objetos CORBA bajo demanda, lo que se logra mediante el repositorio de implementación, que aloja una base de datos de las implementaciones indexadas

por sus nombres de adaptadores de objetos. Un repositorio de interfaz es una base de datos de definiciones de interfaces IDL indexada por un ID de repositorio que está incluida en un IOR.

Los servicios CORBA proporcionan funcionalidades sobre RMI que podrían utilizarse en aplicaciones distribuidas para servicios adicionales como servicios de nombres y de directorios, notificaciones de eventos, transacciones o seguridad.

BIBLIOGRAFÍA

1. G. Coulouris, J. Dollimore, T. Kindberg. **Sistemas Distribuidos – Conceptos y Diseño – 3/E.** Addison Wesley, España, 2001. ISBN 84-7829-049-4.
2. W. Stallings. **Sistemas Operativos – 4/E.** Prentice Hall, España, 2001. ISBN 84-205-3177-4.