Sistemas Distribuidos de Tiempo Real

Apuntes: TEMA 4

Por: J. Javier Gutiérrez gutierjj@unican.es
http://www.ctr.unican.es/

Grupo de Computadores y Tiempo Real, Universidad de Cantabria

Sistemas distribuidos de tiempo real

PARTE II: Modelos de distribución

- **TEMA 3. Modelo de distribución de Ada**
- **TEMA 4. Modelo de distribución de CORBA y RT-CORBA**

Introducción a CORBA

Definido por el OMG: Object Management Group

Estándar industrial para *DOC:*

- **Distributed Object Computing Middleware**

Modelo cliente-servidor:

- **Reside entre los clientes y los servidores y simplifica el desarrollo de aplicaciones al proporcionar una visión uniforme de las diversas capas de sistema operativo y red**

Versión actual CORBA 3.3 (noviembre 2012)

Introducción a CORBA (cont.)

Basado en los ORBs (*Object Request Brokers*) que permiten a los clientes invocar operaciones en una implementación de un objeto:

- sin necesidad de saber dónde se encuentra dicho objeto
- ni el lenguaje de implementación del mismo
- ni el tipo de plataforma OS/Hardware
- ni los protocolos
- ni redes o buses de interconexión

Todas las operaciones incluso las no remotas se invocan a través del ORB

Modelo de referencia de CORBA

Object Request Broker

- definido en la *especificación CORBA*

Object Services

- conjunto de servicios que soportan funciones básicas para usar e implementar objetos
- definido en la especificación *CORBAservices (Common Object Services Specification)*

Modelo de referencia de CORBA (cont.)

Common Facilities

- conjunto de servicios que las aplicaciones pueden compartir, pero que no son tan fundamentales como los *Object Services*
- definido en la especificación *CORBAfacilities (Common Facilities Architecture)*

Application Objects

- productos proporcionados por un vendedor o un grupo de desarrollo
- se corresponden con la noción tradicional de aplicaciones
- no son estandarizados por la OMG
- constituyen la capa más alta del modelo de referencia

Especificaciones CORBA

OMG Middleware Specifications

- CORBA/IIOP Specifications

NOTE: With the release of CORBA 3.0, the following specifications were removed from the core specification and became stand-alone documents.

Real-Time CORBA	These documents are located in the Specifications Catalog under the Specialized CORBA Specifications category.
Minimum CORBA	

- IDL / Language Mapping Specifications
- Specialized CORBA Specifications
- CORBA Component Model (CCM) Specification

Algunas especificaciones de CORBA

- ***Minimum CORBA***: elimina ciertas características del CORBA completo que no son necesarias para los sistemas de tiempo real ni para los empotrados
- ***Real-Time CORBA***: RT-CORBA define características que permiten un predecibilidad de principio a fin para operaciones en aplicaciones CORBA de prioridades fijas
- POA *Portable Object Adapter*: evolución de BOA (*Basic Object Adapter*) que hacen de mediadores entre los objetos de CORBA y las implementaciones

Minimum CORBA 1.0

Es un subconjunto de CORBA diseñado para sistemas con recursos limitados

Mantienen algunas características costosas en cuanto a tamaño del ORB y *stubs*, incluso cuando la aplicación no hace uso de ellas (TypeCode Features, Exception Features, Inheritance Features)

Se eliminan las características que dan soporte a los aspectos dinámicos de CORBA (*Dynamic Invocation Interface*, *Dynamic Skeleton Interface*, *Dynamic Management of Any Value)*

También se elimina gran parte del Interface Repository

Minimum CORBA 1.0 (cont.)

Soporta un subconjunto de las interfaces y políticas soportadas por el *POA*

Debe soportar al menos un 'language mapping' tal y como está definido por la OMG

- debe soportar el mapping completo exceptuando el de aquellos objetos que se han omitido

RT-CORBA

RT-CORBA es un conjunto de extensiones opcionales de CORBA para tiempo real:

- extensión del CORBA 2.2 y del Messaging Specification

Basado en prioridades fijas

No proporciona portabilidad para un sistema operativo de tiempo real:

- la portabilidad está sustentada en las extensiones POSIX de tiempo real

Categorización de la especificación CORBA

Core:

- modelo de objeto, lenguaje IDL, interfaces, invocación de objetos, repositorio, POAs

Interoperability:

- cómo conectar diferentes ORBs (GIOP)

Interworking:

- cómo trabajar con otras arquitecturas de distribución de objetos

QoS:

- servicios de mensajes, tolerancia a fallos y seguridad

Terminología básica de CORBA

Objeto CORBA:

- entidad virtual capaz de ser localizada por un ORB y de recibir peticiones por parte de un cliente

Objeto destino (target):

- objeto CORBA al que se dirige una petición por parte de un cliente

Cliente:

- entidad que realiza peticiones sobre un objeto CORBA

Servidor:

- aplicación en la que existen uno o más objetos CORBA

Terminología básica de CORBA (cont.)

Petición:

- invocación de una operación sobre un objeto CORBA por un cliente

Referencia a objeto:

- etiqueta usada para identificar, localizar y direccionar un objeto CORBA de forma única

Sirviente (servant):

- entidad de un lenguaje de programación que implementa uno o más objetos CORBA
- los sirvientes existen dentro del contexto de una aplicación servidor

Modelo de objeto

CORBA define un modelo de objeto abstracto que no depende de ninguna implementación concreta

El propósito del objeto es proporcionar servicios a los clientes

El cliente realiza peticiones de servicio, y:

- recibe la respuesta al servicio
- o sólo el servidor realiza la operación asociada en el caso de peticiones en una dirección
- o se produce un fallo de algún tipo

El formato de las peticiones se realiza de acuerdo a una sintaxis y semántica definidas por el lenguaje

Modelo de objeto (cont.)

Se define un *valor* como una instancia particular de un objeto de algún tipo de dato IDL que puede ser usado como parámetro en una llamada

También se define la referencia a un objeto como un valor que designa a un objeto particular

Se definen los *tipos* como entidades para definir objetos

Los objetos ofrecen su funcionalidad a través de las *interfaces*; éstas se pueden agupar en *modules*

Tipos de CORBA

Básicos:

- Short, Long, LongLong, Ulong, UlongLong, Float, Double, LongDouble, Fixed, Char, Wchar, String, Wstring, Boolean, Octet, Enum
- Any: representa cualquier tipo básico o estructurado

Estructurados (constructed):

- Struct, Sequence, Union, Array

Interfaces:

- agrupa el conjunto de operaciones que se pueden hacer sobre un objeto

Tipos de CORBA (cont.)

Tipos valor:

- es una mezcla de una interfaz y una struct

Interfaces abstractas:

- puede representar una interfaz o un tipo valor

Operaciones en CORBA

Una operación es una entidad identificable que suministra un servicio indivisible caracterizada por:

- unos parámetros requeridos en su llamada (`in`, `out`, `inout`)
- la especificación de un resultado
- la identificación de las excepciones que podría elevar
- alguna información del contexto que pudiera afectar a la invocación
- una indicación de la semántica de ejecución que podría esperar el cliente que la invoca: at-most-once, best-effort (oneway)

Formato general:

```
[oneway]<op_type_spec><identifier>(param1,...,paramL)
    [raises(except1,...,exceptN)][context(name1,...,nameM)]
```


El ORB

Es el eje de cualquier implementación de CORBA

Su implementación no requiere que sea un solo componente, sino que implemente sus interfaces:

- operaciones iguales para todos los ORBs
- operaciones específicas de tipos de objetos particulares
- operaciones específicas de un estilo particular de implementación de objetos

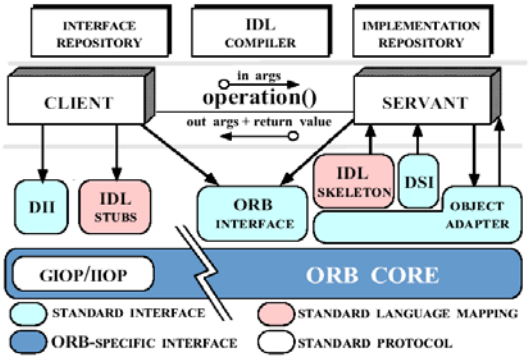
El núcleo del ORB suministra la representación básica de los objetos y la comunicación de peticiones

El ORB (cont.)

El resto de componentes implicados son:

- Los clientes
- Implementaciones de objetos
- Referencias a objetos
- El lenguaje IDL
- Mapeados de IDL a lenguajes de programación
- Stub de cliente
- Interfaz de invocación dinámica
- Implementación del skeleton
- Interfaz de skeleton dinámico
- Adaptador de objetos
- Interfaces de ORB
- Repositorio de interfaces
- Repositorio de implementaciones

Esquema de funcionamiento de CORBA



Components in the CORBA Reference Model

Direccionamiento e invocación de objetos

El uso de un objeto se hace a través de referencias al objeto:

- **se denominan IOR (Interoperable Object Reference) y son opacas al usuario; no se puede extraer ninguna información de las mismas**

La vida de una referencia a objeto es independiente de la vida de su servidor

Un cliente no sabe si el servidor está activado o no en el momento de realizar la petición:

- **un servidor podría estar fuera de servicio temporalmente y más tarde ser restablecido**

Direccionamiento e invocación de objetos (cont.)

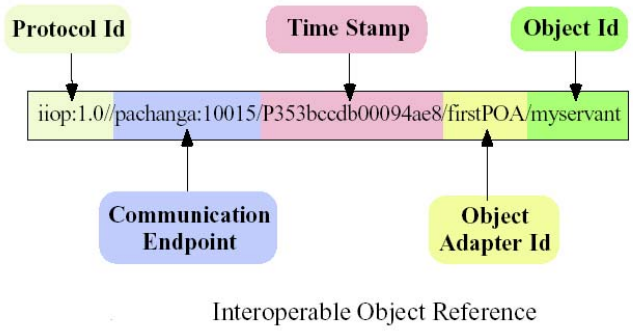
Formalmente una IOR es una tupla con un tipo IDL declarado y un conjunto de perfiles

Los perfiles pueden tener diferentes niveles, por ejemplo, el más utilizado es el IIOP (Internet Inter-ORB Protocol) que puede ser descompuesto en cinco componentes:

- **un número de la revisión IIOP**
- **el nombre del host (o la dirección IP en la notación punto decimal)**
- **el número de puerto TCP en el que esté escuchando el servidor**
- **una clave de objeto específica del servidor, y**
- **etiquetas opcionales**

Direccionamiento e invocación de objetos (cont.)

Ejemplo de IOR con IIOP:



Interoperable Object Reference

Direccionamiento e invocación de objetos (cont.)

El cliente necesita el IOR para invocar a un objeto:

- para ello puede utilizar un servicio de nombres o cualquier otro mecanismo que le proporcione la referencia

Una vez que se conocen la interfaz y la dirección de un objeto, los métodos pueden ser invocados y se ejecutarán remotamente

Los objetos se definen en el lenguaje IDL y el compilador genera los *stubs* en el lenguaje de programación elegido:

- un objeto stub encapsula una referencia a objeto específica
- cuando un cliente adquiere una referencia a objeto, el ORB crea un objeto stub y el código generado se encarga de convertir el método invocado en una petición y se la pasa al ORB para que contacte con el servidor

Direccionamiento e invocación de objetos (cont.)

En la parte del servidor el compilador IDL genera un *skeleton*:

- el programador deriva sus clases de ese esqueleto e implementa los métodos
- el esqueleto generado recibe las peticiones del ORB y realiza la llamada sobre el objeto invocado

Los stubs y los skeletons usan información estática:

- los nombres de los métodos y de los parámetros se deciden en tiempo de compilación

Direccionamiento e invocación de objetos (cont.)

Como alternativa se tiene una invocación dinámica:

- Interfaz de Invocación Dinámica (DII) en la parte de cliente
- Interfaz de Esqueleto Dinámica (DSI) en la parte del servidor

Mediante el uso de DII un cliente puede realizar una petición utilizando información obtenida en tiempo de ejecución:

- se obtiene en un Almacén de Interfaces (Interface Repository)

En la parte del servidor, un sirviente puede utilizar el DSI para responder las peticiones:

- el uso de DII en la parte del cliente no implica utilizar el DSI en la parte del servidor ni viceversa

Componentes del cliente

Stub del cliente

- capa intermedia entre el cliente y el núcleo del ORB
- define cómo se invocan los servicios que proporcionan los objetos servidores
- se encargan de codificar la operación y sus parámetros, y de enviarla de forma remota

Interfaz de invocación dinámica o DII (Dynamic Invocation Interface)

- permite descubrir en tiempo de ejecución métodos para ser invocados

Componentes del cliente (cont.)

Almacén de interfaces

- permite obtener y modificar la descripción de todos los objetos que están registrados en él:
  - métodos que soporta
  - parámetros que requiere
- es como una base de datos distribuida

Interfaz del ORB

- bibliotecas de servicios locales para realizar labores auxiliares en la aplicación

Componentes del servidor

Esqueleto (skeleton) del servidor

- proporciona los elementos necesarios para que los clientes invoquen los servicios exportados por el objeto
- hace transparente el proceso de comunicación

Esqueleto de interfaces dinámicas o DSI (Dynamic Skeleton Interface)

- mecanismo de enlazado en tiempo de ejecución para servidores que necesitan manejar peticiones realizadas dinámicamente

Componentes del servidor (cont.)

Adaptador de objetos

- se encarga de gestionar las peticiones que llegan al servidor y de asignar referencias e instanciar objetos

Almacén de implementaciones

- contiene la información acerca de las clases que soporta el servidor, los objetos instanciados y sus identificadores

Adaptadores de objetos

Un adaptador de objeto es un mediador entre el ORB y las implementaciones de los objetos CORBA, consiguiendo que las peticiones que llegan a través del ORB sean procesadas por la implementación del objeto

Cuando se realiza una invocación el ORB de la parte del cliente es responsable de:

- interpretar los métodos de los objetos
- localizar el servidor donde se encuentra el objeto y
- enviar una petición a ese servidor

Adaptadores de objetos (cont.)

En la parte del servidor, la petición es recibida por el ORB donde se realizan tres pasos:

- el ORB debe encontrar el adaptador de objetos en el que está implementado el objeto y pasar la petición a ese adaptador
- el adaptador de objetos debe encontrar el sirviente que implementa el objeto
- si el sirviente utiliza un esqueleto estático, la petición es interpretada por el código generado por IDL y se invoca al método deseado

Antes de que todo esto se realice el adaptador de objetos tiene que conocer al sirviente

Adaptadores de objetos (cont.)

Las responsabilidades de un adaptador de objetos son:

- Registro de los objetos: proporcionar funciones para registrar implementaciones para los objetos CORBA
- Generación de referencias a objetos: generar referencias para los objetos que tengan registrados
- Activación de procesos servidores: activar los objetos registrados
- Multiplexación de peticiones a los objetos registrados: asegurar que todas las peticiones sean recibidas por los objetos, soportar la concurrencia de peticiones
- Gestión de las invocaciones: deben despachar las peticiones de objetos registrados

Adaptadores de objetos (cont.)

CORBA permite múltiples adaptadores de objetos:

- son diferentes para cada lenguaje de implementación
- sólo proporciona dos adaptadores

Inicialmente OMG propuso BOA (Basic Object Adapter): fuera de uso

En CORBA 2.2 se introdujo POA (Portable Object Adapter) para cubrir dos aspectos:

- portabilidad: usar código fuente con distintos ORBs comerciales sin tener que cambiar el código
- flexibilidad: proporcionar herramientas para controlar el ciclo de vida de los sirvientes y la recepción de peticiones

BOA

Cubría de forma mínima las funciones del adaptador de objetos:

- el registro de objetos
- la generación de referencias
- la activación de implementaciones y
- la gestión de las peticiones

Introduce el concepto de almacen de implementaciones que se corresponde con el almacen de interfaces en el lado del cliente

POA

Trabaja con tres entidades fundamentales:

- Referencias a objeto: es responsable de crearlas
- Identificadores de objeto: cada objeto es identificado por una secuencia de bytes de forma única
  - cuando POA crea un nuevo objeto introduce su identificador en la clave del objeto dentro de la referencia al objeto
  - cuando un cliente invoca una petición, el ORB del cliente utiliza dicha referencia para determinar los puntos de la comunicación donde se encuentra el objeto y envía allí la petición
  - el ORB servidor utiliza la clave del objeto para determinar qué POA del servidor controla dicho objeto y redirecciona la petición a ese adaptador, que busca al sirviente y le pasa la petición
- Sirvientes: una aplicación puede crear y registrar sirvientes cuando sea necesario

POA (cont.)

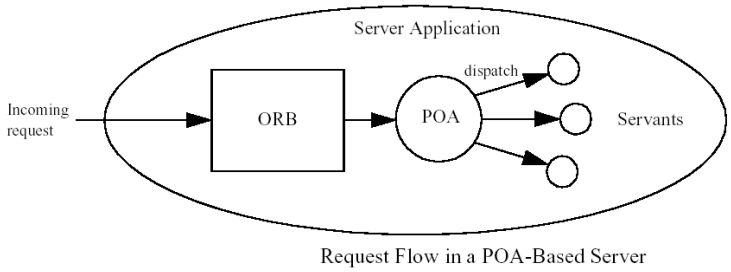
Se pueden tener varias instancias de POA organizadas jerárquicamente en un servidor

El ORB crea una instancia de POA raíz y los siguientes POA se puecen crear como hijos suyos

Cada POA mantiene su propio mapa de objetos activos:

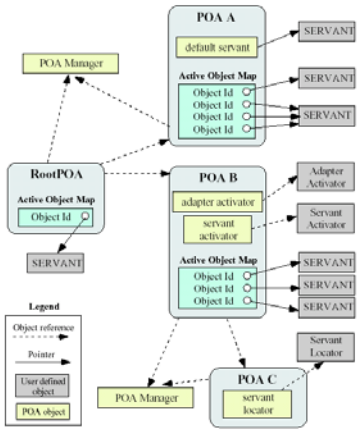
- una tabla que relaciona los objetos activos con los sirvientes
- los objetos son activados a instancia de un POA y se asocian con él identificados por un único identificador de objeto (IOR)

POA: Atención a invocación de objeto



Request Flow in a POA-Based Server

Ejemplo de arquitectura de POAs



Gestión de objetos en POA

La especificación CORBA ha definido un conjunto de políticas de gestión de objetos para el POA, y el usuario puede definir los valores para estas políticas en la creación de un nuevo POA:

- **Política de threads**
- **Política de vida**
- **Política de identificador único**
- **Política de asignación de identificación**
- **Política de procesamiento de peticiones**
- **Política de activación**
- **Política de retención de sirvientes**

Interoperabilidad en CORBA

La interoperabilidad de CORBA ofrece una pasarela que hace que diferentes implementaciones de ORBs sean compatibles

Esta parte del estándar se asienta en la capa de transporte del modelo OSI

La arquitectura de interoperabilidad entre ORBs está basada en el protocolo GIOP (General Inter - ORB Protocol)

- especifica la sintaxis de transferencia de un conjunto de mensajes estándares para la comunicación entre ORBs basada en un protocolo de transporte orientado a conexión

Interoperabilidad en CORBA (cont.)

La especificación consta de los siguientes elementos:

- CDR (Common Data Representation): define cómo deben codificarse los datos para su transferencia entre clientes y servidores
- Formato de los mensajes GIOP: define ocho tipos de mensajes que pueden intercambiar los ORBs del cliente y del servidor
  - dos tipos necesarios para la comunicación: Request y Reply
  - el resto son mensajes de control
- Condiciones de la capa de transporte:
  - debe ser orientado a conexión
  - las conexiones son full-duplex
  - las conexiones son simétricas
  - comunica si se produce una pérdida de conexión

Interoperabilidad en CORBA (cont.)

GIOP especifica la mayoría de los detalles de protocolo que son necesarios para que se puedan comunicar los servidores y los clientes

GIOP es un protocolo abstracto y, por tanto, es independiente de un tipo de transporte particular

El IIOP (Internet Inter.-ORB Protocol) es una implementación concreta del protocolo GIOP sobre el protocolo de transporte TCP/IP

- necesita especificar cómo las IOR codifican la información del direccionamiento de TCP/IP
- es el principal protocolo de interoperabilidad utilizado por CORBA

Lenguaje IDL

El IDL permite separar la interfaz de la implementación de los objetos CORBA

Ofrece una forma de especificar las interfaces de los objetos de forma independiente de los lenguajes de programación elegidos para implementar los métodos

- esto difiere de los sistemas centralizados, en los que tanto la especificación como la implementación se definen en un único sistema y utilizando un único lenguaje: es el caso de Ada por ejemplo

Vamos a ver el lenguaje IDL mediante los ejemplos de definiciones de tipos y funciones de la distribución de PolyORB

Lenguaje IDL: tipos

```
interface all_types {

  // Simple types

  boolean echoBoolean(in boolean arg) ;
  short echoShort(in short arg) ;
  long echoLong(in long arg) ;
  unsigned short echoUShort(in unsigned short arg) ;
  unsigned long echoULong(in unsigned long arg) ;
  unsigned long long echoULLong(in unsigned long long arg) ;
  float echoFloat(in float arg) ;
  double echoDouble(in double arg) ;
  char echoChar(in char arg) ;
  wchar echoWChar(in wchar arg) ;
```


Lenguaje IDL: tipos (cont.)

```
  // Simple types

  octet echoOctet (in octet arg) ;
  string echoString (in string arg) ;
  wstring echoWString (in wstring arg) ;
  all_types echoRef (in all_types arg);
  Object echoObject (in Object arg);

  typedef all_types otherAllTypes;
  typedef Object otherObject;
  otherAllTypes echoOtherAllTypes (in otherAllTypes arg);
  otherObject echoOtherObject (in otherObject arg);
```

Lenguaje IDL: tipos (cont.)

```
// Bounded strings

typedef string<12> BoundedStr;
BoundedStr echoBoundedStr (in BoundedStr arg);

typedef wstring<11> BoundedWStr;
BoundedWStr echoBoundedWStr (in BoundedWStr arg);

// Enum

enum Color { Red, Green, Blue };
Color echoColor (in Color arg);
```


Lenguaje IDL: tipos (cont.)

```
// Array of enum

typedef Color Rainbow[7];
Rainbow echoRainbow (in Rainbow arg);

// Exceptions

exception my_exception {long info;};

void testException (in long arg) raises (my_exception);
void testUnknownException (in long arg);
void testSystemException (in long arg);
```


Lenguaje IDL: tipos (cont.)

```
// Unions

union myUnion switch (long) {
  case 1: long    Counter;
  case 2: boolean Flag;
  case 3: Color   Hue;
  default: long Unknown;
};

myUnion echoUnion (in myUnion arg);
```

Lenguaje IDL: tipos (cont.)

```
// Unions

union myUnionEnumSwitch switch (Color) {
  case Red:   long   foo;
  case Green: short  bar;
  case Blue:  string baz;
};
myUnionEnumSwitch echoUnionEnumSwitch
                        (in myUnionEnumSwitch arg);

union noMemberUnion switch (boolean) {
  case FALSE: long falseVal;
};
noMemberUnion echoNoMemberUnion (in noMemberUnion arg);
```


Lenguaje IDL: tipos (cont.)

```
// Arrays

typedef long simple_array[5];
simple_array echoArray (in simple_array arg);

// Multi-dimensional arrays

typedef long matrix[3][3];
matrix echoMatrix (in matrix arg);

typedef long bigmatrix[30][15];
bigmatrix echoBigMatrix (in bigmatrix arg);
```


Lenguaje IDL: tipos (cont.)

```
// Nested arrays

typedef simple_array nested_array[3];
nested_array echoNestedArray (in nested_array arg);

// Big arrays

typedef long sixteenKb[64][64];
sixteenKb echoSixteenKb (in sixteenKb arg);
```

Lenguaje IDL: tipos (cont.)

```
// Structs

struct simple_struct {
    long a;
    string s;
};
simple_struct echoStruct (in simple_struct arg);

struct array_struct {
    long a[10];
    unsigned short b;
};
array_struct echoArrayStruct (in array_struct arg);
```


Lenguaje IDL: tipos (cont.)

```
// Structs

struct composite_struct {
    fixed<12,3> fixedMember;
    sequence<sequence<octet> > seqseqMember;
    long double matrixMember[3][4];
};

struct nested_struct {
  simple_struct ns;
};
nested_struct echoNestedStruct (in nested_struct arg);
```


Lenguaje IDL: tipos (cont.)

```
// Sequences

typedef sequence<short> U_sequence;
U_sequence echoUsequence (in U_sequence arg);

typedef sequence<short,10> B_sequence;
B_sequence echoBsequence (in B_sequence arg);

// Fixed point

typedef fixed<18,2> Money;
Money echoMoney (in Money arg);
```

Lenguaje IDL: tipos (cont.)

```
  // Attributes

  readonly attribute long Counter;
  attribute Color myColor;

  void StopServer ();  // Shut down server
};
```


Lenguaje IDL: métodos

```
interface all_functions {

  // attributes

  attribute short the_attribute ;

  readonly attribute short the_readonly_attribute ;
```


Lenguaje IDL: métodos (cont.)

```
  // procedures

  void void_proc() ;

  void in_proc(in short a, in short b, in short c) ;

  void out_proc(out short a, out short b, out short c) ;

  void inout_proc(inout short a, inout short b) ;

  void in_out_proc
     (in short a, in short b, out short c, out short d) ;
```

Lenguaje IDL: métodos (cont.)

```
// procedures

void in_inout_proc
   (in short a, inout short b, in short c, inout short d) ;

void out_inout_proc
   (out short a, inout short b, inout short c, out short d) ;

void in_out_inout_proc
   (in short a, out short b, inout short c) ;
```


Lenguaje IDL: métodos (cont.)

```
// functions

short void_fun() ;

short in_fun(in short a, in short b, in short c) ;

short out_fun(out short a, out short b, out short c) ;

short inout_fun(inout short a, inout short b) ;

short in_out_fun
   (in short a, in short b, out short c, out short d) ;
```

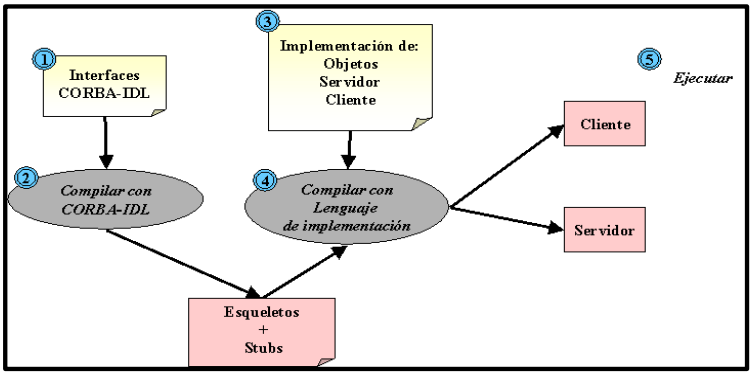

Lenguaje IDL: métodos (cont.)

```
// functions

short in_inout_fun
   (in short a, inout short b, in short c, inout short d) ;

short out_inout_fun
   (out short a, inout short b, inout short c, out short d) ;

short in_out_inout_fun
   (in short a, out short b, inout short c) ;
```

Lenguaje IDL: métodos (cont.)

```
  // oneway procedures

  oneway void oneway_void_proc() ;

  oneway void oneway_in_proc(in short a, in short b) ;

  short oneway_checker() ;

};
```


Desarrollo de una aplicación en CORBA



Introducción a RT-CORBA

Un sistema RT-CORBA debe componerse al menos de cuatro componentes que independientemente proporcionen predecibilidad:

- **Mecanismos de planificación en el OS**
- **ORB de tiempo real**
- **Transporte para las comunicaciones**
- **Aplicaciones**

RT-CORBA no describe un RT-IOP específico:

- **la especificación utiliza mecanismos de extensión disponibles en IIOP:**
  - **GIOP ServiceContexts, IOR Profiles, IOR Tagged Components**

Introducción a RT-CORBA (cont.)

RT-CORBA define un conjunto de extensiones a CORBA; las más destacadas son:

- **ORB de tiempo real: define una extensión de la interfaz de ORB**
- **Planificación de threads: usa el thread como entidad de planificación**
- **Prioridad RT-CORBA: define una prioridad universal independiente de la plataforma**
  - **define las prioridades nativas y**
  - **los mapeados de prioridad: nativas/RT-CORBA**
- **Priority models: client propagated / server declared**
- **RT-CORBA Mutexes y Herencia de prioridad**

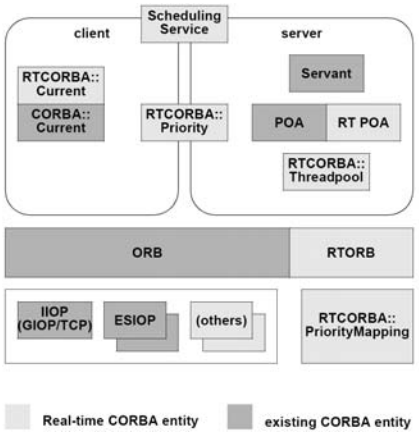
Introducción a RT-CORBA (cont.)

- **Threadpools: usa los conjuntos de threads para ejecutar servicios con las siguientes características**
  - **creación de threads previa: reduce la latencia e incrementa la predecibilidad**
  - **particionado de threads: disponer de varios pools asociados con diferentes POAs**
  - **limitación del uso de threads: número máximo que puede usar un POA o conjunto de POAs**
  - **buffer para peticiones adicionales: si no hay suficientes threads se encolan las peticiones**
- **Priority Banded Connections: para reducir la inversión de prioridad si se usa un protocolo de transporte sin prioridades, se permite que el cliente se comunique por diferentes conexiones (cada una con un rango de prioridades distinto)**

Introducción a RT-CORBA (cont.)

- **Non-Multiplexed Connections: permite a un cliente definir conexiones privadas (no compartidas con otros clientes)**
- **Invocation Timeouts: en las esperas de respuesta a una petición**
- **Client and Server Protocol Configuration: interfaces que permiten la selección de protocolos en los lados del cliente y del servidor**
- **Configuración de RT-CORBA:**
  - **configuración de los threadpools en el servidor (servidor)**
  - **modelo de prioridad utilizado (servidor)**
  - **creación de conexiones por bandas de prioridad (cliente)**
  - **creación de conexiones no multiplexadas (cliente)**
  - **selección y configuración del protocolo**

Modelo de RT-CORBA



Modelo de prioridades

En el módulo `RTCORBA` se define el tipo `Priority` con valores de 0 a 32767

Se definen también funciones de cambio de prioridad entre prioridades nativas y CORBA

La interfaz `Current` define la prioridad base del thread actual

Se definen dos políticas de propagación de prioridades:

- `CLIENT_PROPAGATED`: la prioridad del cliente se propaga al sirviente
- `SERVER_DECLARED`: establece la prioridad del sirviente
  - en este hay opción de sobreescribir la prioridad basada en el objeto concreto

Modelo de prioridades (cont.)

Existe la posibilidad de realizar transformaciones de prioridades, y utilizar modelos diferentes a las dos políticas propuestas:

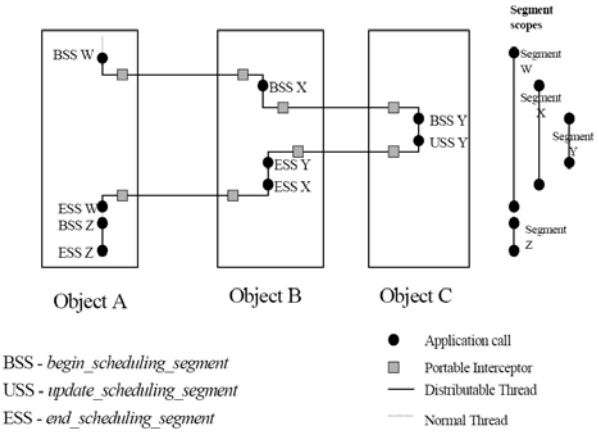
- *inbound*: la transformación ocurre antes de que se use la prioridad concreta en el lado del servidor
- *outbound*: la transformación ocurre en el momento de hacer una invocación desde el código de aplicación del sirviente

Thread distribuido

Es un concepto similar al de la transacción distribuida:

- **introduce la secuenciación para separar la planificación del despacho**
- **admite varias políticas de planificación, caracterizadas por sus parámetros de planificación y planificadores**
- **define el segmento de planificación como entidad planificable con sus parámetros asociados**
- **define los puntos de planificación como instantes en los que el planificador altera la planificación en curso**
- **el thread distribuido está compuesto de segmentos y puntos de planificación**

Thread distribuido (cont.)



Políticas de planificación

RT-CORBA contempla las siguientes políticas de planificación:

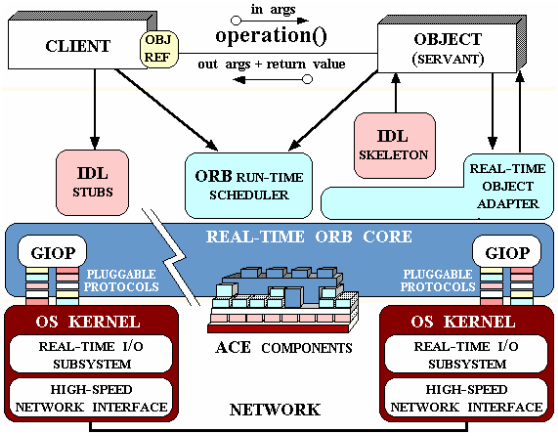
- **prioridades fijas**
- **EDF (Earliest Deadline First)**
- **LLF (Least Laxity First)**
  - **laxity=deadline-hora actual-ejecución pendiente**
- **MAU (Maximized Accrued Utility)**
  - **usa una función de utilidad asociada a cada thread que establece su planificación**

TAO

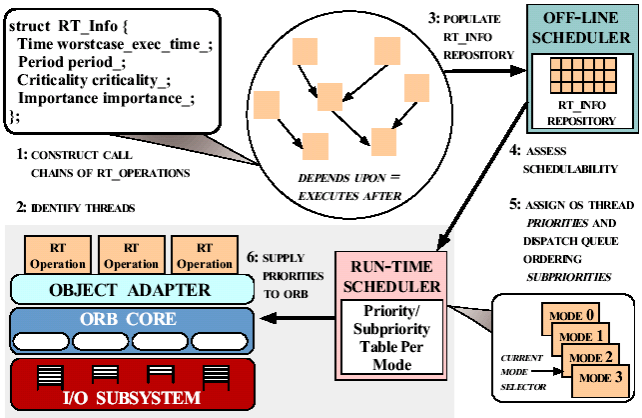
Implementación de CORBA con funcionalidad de RT-CORBA y servicios de planificación global añadidos

- **http://www.cs.wustl.edu/~schmidt/TAO.html**

Arquitectura de TAO



Global Scheduling Service en TAO



RT_INFO en TAO

struct que define QoS para una operación:

- **Tiempo de peor caso (tiempo real estricto)**
- **Tiempo típico de ejecución (tiempo real estadístico)**
- **Cached execution time**
- **Periodo (0 -> *reactive*)**
- ***Criticality* (primer orden)/ *Importance* (segundo orden)**
- ***Quantum* (round robin para misma prioridad)**
- **Información de dependencias: RT_Info de las operaciones de que depende directamente**
- **Prioridad de OS, subprioridad**
- **The queue number for this RT_Info**
- **Número de threads internos**