Sistemas Distribuidos de Tiempo Real

Apuntes: TEMA 6

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 III: Middlewares de distribución

- **TEMA 5. Middleware de distribución para el modelo Ada**
- **TEMA 6. Middleware de distribución esquizofrénico para lenguaje Ada: CORBA y DSA**
- **TEMA 7. Middlewares de distribución de tiempo real**

Introducción a PolyORB

Suministra una solución uniforme para construir aplicaciones distribuidas basada en estándares:

- **CORBA**
- **DSA de Ada 95**
- **paradigmas de distribución basados en servicios Web**
- **MOM: Message Oriented Middleware**
- **middleware específico de aplicación**

Se basa en dos subsistemas que permiten la interacción de particiones:

- **la API hacia el que construye los objetos de aplicación**
- **el protocolo del middleware para la interacción entre nodos**

PolyORB como middleware genérico

La arquitectura de PolyORB define un marco uniforme en el que interaccionan los diferentes modelos de distribución

Proporciona un conjunto de componentes sobre los que elaborar diferentes instancias:

- ***personalidades*** que corresponden a cada modelo de distribución

Las personalidades son visiones mutuamente exclusivas de la misma arquitectura y poseen los dos subsistemas:

- nivel de API (personalidad de aplicación)
- nivel de protocolo (personalidad de protocolo)

PolyORB como middleware genérico (cont.)

El desacoplo entre las personalidades de aplicación y protocolo permite soportar varias personalidades en el mismo sistema y la interoperación entre las mismas:

- interoperabilidad middleware-to-middleware (M2M)

Personalidades de aplicación soportadas:

- CORBA
- DSA
- Message Oriented Middleware for Ada (MOMA)
- Ada Web Server (AWS)

PolyORB como middleware genérico (cont.)

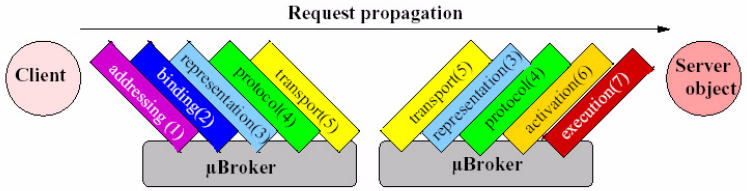
Personalidades de protocolo soportadas:

- GIOP:
  - nivel de transporte de la especificación CORBA
  - es un protocolo genérico sobre el que se proporcionan varias instancias:
    - IIOP: TCP/IP
    - MIOP: IP multicast
    - DIOP: UDP/IP
- SOAP: intercambio de información estructurada y tipificada entre pares (en XML)

Características de PolyORB

Identifica un número reducido de servicios para soportar los diferentes modelos de distribución:

- **identifica 7 pasos en el procesado de peticiones remotas, cada uno de los cuales define un servicio fundamental:**



Características de PolyORB (cont.)

Los servicios son componentes genéricos para los que se proporciona una implementación general:

- **los desarrolladores pueden dar una versión alternativa**

Cada instancia del middleware es un ensamblado coherente de estas entidades, siendo el *µBroker* el encargado de coordinar los servicios y del correcto tratamiento de la petición remota

En la arquitectura del middleware los servicios son *pipes* y *filtros* que manipulan los valores y se lo pasan al siguiente componente

Características de PolyORB (cont.)

En los siete pasos que se consideran se realizan las siguientes actividades:

1. **El cliente obtiene una referencia al servidor usando el servicio de direccionamiento (servidor de nombres, por ejemplo)**
2. **El servicio de enlace establece una conexión con el servidor usando uno de los canales de comunicaciones (por ejemplo: sockets)**
3. **Los parámetros de la petición se mapean en una representación adecuada para la transmisión (lo hace el servicio de representación). Se obtiene un stream de bytes (por ejemplo CORBA CDR)**
4. **Un protocolo de comunicaciones realiza la transmisión entre los nodos cliente y servidor**

Características de PolyORB (cont.)

5. El servicio de transporte establece el canal de comunicación entre los dos nodos
6. En la recepción de una petición remota el middleware asegura que hay una entidad disponible para ejecutar la petición mediante el servicio de activación
7. El servicio de ejecución asigna los recursos necesarios para procesar la petición

Políticas en PolyORB

Política de ejecución de tareas:

- No_Tasking
- Full_Tasking
- Ravenscar

Política de uso de tareas:

- No_Tasking
- Thread_Per_Session
- Thread_Per_Request
- Thread_Pool
  - se configura con tres valores con el mismo significado que en GLADE

Políticas en PolyORB

Políticas de control del ORB:

- No_Tasking
  - se hace un lazo en el que se procesan los trabajos internos y luego se monitorizan las peticiones remotas
- Workers
  - todas las tareas son iguales y alternativamente van procesando trabajos y esperando a las peticiones remotas
- Half_Sync/Half_Async
  - un thread dedicado monitoriza las peticiones remotas y se las va pasando a los threads del pool
- Leader/Followers
  - las tareas del pool van monitorizando y ejecutando las peticiones remotas (espera y ejecuta, después la siguiente)

Aplicación CORBA con PolyORB

Los pasos para generar una aplicación CORBA serían:

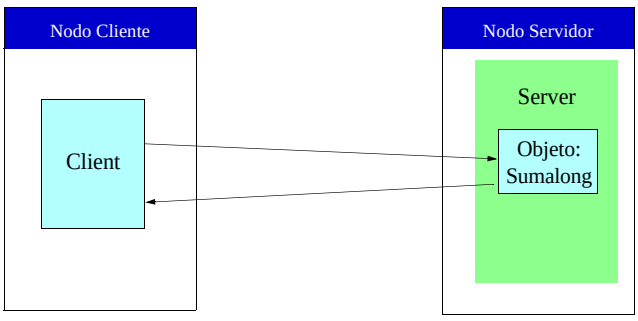
1. Generar el fichero o ficheros IDL con las interfaces de los objetos que se van a poder usar remotamente
2. Realizar la compilación IDL al lenguaje de programación (Ada) a usar con la opción de generar las implementaciones
3. En cada fichero de implementación añadir el código Ada correspondiente al objeto
4. Realizar el programa del cliente(s) y del servidor(es)
5. Alternativamente usar un servidor de nombres o las referencias IOR explícitas de los objetos generados
6. Compilar los programas
7. Ejecutar el programa

Aplicación DSA con PolyORB

El desarrollo de la aplicación es DSA:

- igual que para GLADE
- se compila con `po_gnatdist` sobre un fichero `.cfg`
- es necesario configurar un servidor de nombres

Ejemplo de suma remota en CORBA



1. Edición de suma.idl

Interfaz que ofrece una operación de suma de dos enteros

```
interface Suma {
  long Sumalong (in long op1, in long op2);
};
```


2. Compilación de suma.idl

Llamada al compilador:

```
idlac -i suma.idl
```

- la opcion `-i` genera las implementaciones

Se generan los siguientes ficheros Ada:

- *stubs* de cliente: suma.ads, suma.adb
- *skeletons* de servidor: suma-skel.ads, suma-skel.adb
- operaciones de ayuda: suma-helper.ads, suma-helper.adb
- implementación del sirviente:
  - suma-impl.ads
  - suma-impl.adb: aquí es donde se escribe el código del objeto

3. Edición de suma-impl.adb

```
package body Suma.Impl is
   function Sumalong
     (Self : access Object;
      op1 : in CORBA.Long;
      op2 : in CORBA.Long)
     return CORBA.Long
   is
      Result : CORBA.Long;
   begin
      --  Insert implementation of Sumalong
      Result:=op1+op2;  -- Línea añadida
      return Result;
   end Sumalong;
end Suma.Impl;
```

4. Edición del cliente

```
with Ada.Text_IO;
with CORBA.ORB;
with CORBA.Object;

with Suma;
with Suma.Helper;

with PolyORB.Setup.Client;
pragma Warnings (Off, PolyORB.Setup.Client);

with PolyORB.CORBA_P.Naming_Tools;
```


4. Edición del cliente (cont.)

```
procedure Client is
   use Ada.Command_Line;
   use Ada.Text_IO;
   use PolyORB.Utils.Report;

   Num1, Num2, Result : CORBA.Long;
   Ref : CORBA.Object.Ref;
   Mysuma : Suma.Ref;

begin
   CORBA.ORB.Initialize ("ORB");
   --  Getting object reference from nameserver
   Ref:= PolyORB.CORBA_P.Naming_Tools.Locate("AddServer");
   Mysuma := Suma.Helper.To_Ref(Ref);
```


4. Edición del cliente (cont.)

```
   --  Checking if it worked
   if Suma.Is_Nil (Mysuma) then
      Put_Line ("main : cannot invoke on a nil reference");
      return;
   end if;
   --  Calling remote operation
   Num1 := 12;
   Num2 := 13;
   Result := Suma.Sumalong(Mysuma, Num1, Num2);
   --  Printing result
   Put_Line ("Suma = " & CORBA.Long'Image (Result));

end Client;
```

4. Edición del servidor

```
with Ada.Text_IO;

with CORBA.Impl;
with CORBA.Object;
with CORBA.ORB;

with PortableServer.POA.Helper;
with PortableServer.POAManager;

with Suma.Impl;

with PolyORB.CORBA_P.CORBALOC;

with PolyORB.CORBA_P.Naming_Tools;
```


4. Edición del servidor (cont.)

```
--  Setup server node: use no tasking default configuration
with PolyORB.Setup.No_Tasking_Server;
pragma Warnings (Off, PolyORB.Setup.No_Tasking_Server);

procedure Server is
begin
   declare
      Argv : CORBA.ORB.Arg_List :=
             CORBA.ORB.Command_Line_Arguments;
   begin
      CORBA.ORB.Init (CORBA.ORB.To_CORBA_String ("ORB"), Argv);
      declare
         Root_POA : PortableServer.POA.Local_Ref;
         Ref : CORBA.Object.Ref;
```


4. Edición del servidor (cont.)

```
         Obj : constant CORBA.Impl.Object_Ptr :=
                                  new Suma.Impl.Object;
      begin
         --  Retrieve Root POA
         Root_POA := PortableServer.POA.Helper.To_Local_Ref
           (CORBA.ORB.Resolve_Initial_References
            (CORBA.ORB.To_CORBA_String ("RootPOA")));
         PortableServer.POAManager.Activate
           (PortableServer.POA.Get_The_POAManager (Root_POA));
         --  Set up new object
         Ref := PortableServer.POA.Servant_To_Reference
           (Root_POA, PortableServer.Servant (Obj));
```

4. Edición del servidor (cont.)

```
         --  Output IOR
         Ada.Text_IO.Put_Line
           ("'"
            & CORBA.To_Standard_String
              (CORBA.Object.Object_To_String (Ref))
            & "'");
         Ada.Text_IO.New_Line;
         --  Output corbaloc
         Ada.Text_IO.Put_Line
           ("'"
            & CORBA.To_Standard_String
            (PolyORB.CORBA_P.CORBALOC.Object_To_Corbaloc (Ref))
            & "'");
```


4. Edición del servidor (cont.)

```
         --  Register the object in the name server
         PolyORB.CORBA_P.Naming_Tools.Register
           ("AddServer",Ref);
         --  Launch the server
         CORBA.ORB.Run;
      end;
   end;
end Server;
```


5. Servidor de nombres

```
   -- Inicialización del Servicio
   PolyORB.CORBA_P.Server_Tools.Initiate_Well_Known_Service
     (PortableServer.Servant (Root_NC), "NameService", Ref);
   CORBA.ORB.Register_Initial_Reference
     (CORBA.ORB.To_CORBA_String ("NamingService"), Ref);
   -- Se pinta el IOR y el CORBAloc
   Ada.Text_IO.Put_Line
     ("POLYORB_CORBA_NAME_SERVICE="
      & CORBA.To_Standard_String
        (CORBA.Object.Object_To_String (Ref)));
   Ada.Text_IO.Put_Line
     ("POLYORB_CORBA_NAME_SERVICE="
      & CORBA.To_Standard_String
          (PolyORB.CORBA_P.CORBALOC.Object_To_Corbaloc (Ref)));
```

6. Compilación del programa

Compilación del cliente:

```
gnatmake client.adb ‘polyorb-config‘
```

Compilación del servidor:

```
gnatmake servidor.adb ‘polyorb-config‘
```

Compilación del servidor de nombres:

```
gnatmake nameserver.adb ‘polyorb-config‘
```

Se pueden hacer `Makefiles` en los que se pasan las rutas de compilación y enlazado

7. Ejecución del programa

Colocar cada ejecutable (client, server, nameserver) en un directorio separado

Especificar para cada uno su fichero de configuración:

- `polyorb.conf`

7. Ejecución: cliente - `polyorb.conf`

```
# CORBA parameters
[corba]
name_service=corbaloc:iiop:1.2@127.0.0.1:5557/NameService/
000000024fF0000000080000000

# IIOP parameters
[iiop]

# IIOP default address
polyorb.protocols.iiop.default_addr=127.0.0.1

# IIOP default port
polyorb.protocols.iiop.default_port=5559
```

7. Ejecución: cliente - polyorb.conf (cont.)

```
# Parameters for tasking
[tasking]

# Default storage size for all threads spawned by PolyORB
storage_size=262144

# Number of threads by Thread Pool tasking policy
min_spare_threads=4
max_spare_threads=4
max_threads=4
```


7. Ejecución: servidor - polyorb.conf

```
# CORBA parameters
[corba]
name_service=corbaloc:iiop:1.2@127.0.0.1:5557/NameService/
000000024fF0000000080000000

# IIOP parameters
[iiop]

# IIOP default address
polyorb.protocols.iiop.default_addr=127.0.0.1

# IIOP default port
polyorb.protocols.iiop.default_port=5558
```


7. Ejecución: servidor de nombres- polyorb.conf

```
# IIOP parameters
[iiop]

# IIOP default address
polyorb.protocols.iiop.default_addr=127.0.0.1

# IIOP default port
polyorb.protocols.iiop.default_port=5557
```