

# **PRINCIPIOS DE MIDDLEWARE ORIENTADO A OBJETOS**

Área de Telemática  
Departamento de Sistemas  
Escuela de Ingeniería  
Universidad EAFIT, Medellín, Colombia

## ¿Qué es un *middleware*?

*Middleware* es una capa entre los sistemas operativos de redes (NOS) y las aplicaciones que apunta a resolver la heterogeneidad y la distribución

## Problemas construyendo aplicaciones distribuidas(I)

- La comunicación frecuentemente necesitará enviar parámetros complejos, tal como registros, arreglos o cadenas de caracteres.
- Codificación diferentes de tipos de datos pueden ser utilizada si los clientes y los objetos no son desplegados en la misma plataforma de hardware y si ellos no son escritos en el mismo lenguaje de programación.
- Los parámetros y los valores de retorno pueden referirse a otros objetos distribuidos.
- Los desarrolladores y administradores puede tener que implementar activaciones de objeto, esto es la habilidad de un componente servidor responder a un requerimiento de un cliente aun cuando este no está actualmente corriendo en un proceso del sistema operativo.
- *La seguridad de tipos* denota la garantía que una operación requerida por un cliente es actualmente implementada por un componente servidor y que los parámetros actuales suministrados por el cliente deben coincidir con la definición del servicio. Alcanzar la seguridad de tipos manualmente es una tarea totalmente tediosa y propensa a errores, particularmente si los componentes del cliente y el servidor son desarrollados por diferentes ingenieros.

## Problemas construyendo aplicaciones distribuidas(II)

- Después de enviar un requerimiento, el objeto cliente necesita esperar por los resultados retornen, tanto de un flujo de salida TCP o como por un mensaje de respuesta UDP. De nuevo, la implementación de esta tarea de sincronización manualmente es una tarea tediosa y propensa a errores.
- Algunas veces hay cualidades de servicio que no pueden ser garantizadas por el nivel de red. Esta puede, por ejemplo, llegar a ser necesario por diferentes requerimientos de servicios que estos sean implementados de forma atómica, estos se ejecutan todos completamente o no.

## Objetivo del *middleware*

*Middleware* simplifica la construcción de sistemas distribuidos implementando las capas de sesión y presentación.

## Clasificación de los *Middleware*

Los *middleware* disponibles pueden ser clasificados en tres grandes categorías:

- Orientados a transacciones.
- Orientados a mensajes.
- Orientados a objetos.

## *Middleware* orientado a transacciones

- Es frecuentemente utilizados con aplicaciones de bases de datos distribuidas.
- Utiliza el protocolo de *commit* de dos fase para implementar las transacciones distribuidas.
- CICS de IBM, Tuxedo de BEA, Transac de Encia y COM+ de Microsoft.
- El *middleware* orientado a objetos tiene capacidades de los *middlewares* orientado a transacciones.

## *Middleware* orientado a mensajes

- Soporta la comunicación entre sistemas distribuidos a través del intercambio de mensajes.
- Es utilizado cuando la interacción de sistemas distribuidos implica una forma de comunicación dominante de tipo confiable t asincrónica.
- MQSeries de IBM, MSQueue de Microsoft, JavaMessage de Sun.
- Una de las fortalezas de este *middleware* es que soporta multidifusión.
- Implementa la tolerancia a fallos implementado colas de mensajes que almacenan los mensajes temporalmente en un medio persistente.
- El *middleware* orientado a objetos se está comenzando a integrar con este tipo de *middleware*.

## Llamadas a procedimientos remotos (RPC)

- Una RPC es una llamada a procedimientos a través de los límites de los host.
- Birrell y Nelson desarrollaron las base para los RPC a través de su artículo clásico *Implementing Remote Procedure Calls* en 1984.
- Sun implementó la primera versión comercial de RCP como parte de su plataforma ONC (*Open Network Computing*).
- El origen del *middleware* son los RPC.

## Lenguaje de definición de interfaces (RPC)

```
const NL=64;
struct Jugador {
    struct DoB int dia, int mes, int año;
    string nombre<NL>;
};
program JUGADORPROG {
    version JUGADORVERSION {
        void IMPRIMIR(Jugador)=0;
        int  ALMACENAR(Jugador)=1;
        Jugador CARGAR(int) = 2;
    } = 0;
} = 105040;
```

## Capa de presentación

- La integración de la capa de presentación mapea las estructuras de datos de la aplicación dentro de una forma transmisible y homogéneo.
- Los mapeos entre las representaciones de datos de transporte y aplicación son llamadas *marshalling* y *unmarshalling*
- El *marshalling* puede ser implementado estáticamente o dinámicamente.
- Los *stub* del cliente y el servidor son implementaciones estáticas del *marshalling* y *unmarshalling*

## Ejemplo de *Marshalling* y *Unmarshalling*

```
char *marshall() {
    char *msg;
    msg = new char[4*(sizeof(int)+1) + strlen(name) + 1];
    sprintf(msg, "%d%d%d%d%s", dob.dia, dob.mes, dob.año,
            strlen(name), name);
    return msg;
}

void unmarshall(char *msg) {
    int name_len;
    sscanf(msg, "%d%d%d%d", &dob.dia, &dob.mes, &dob.año,
           &name_len);
    name = new char[name_len+1];
    sscanf(msg, "%d%d%d%d%s", &dob.dia, &dob.mes, &dob.año,
           &name_len, name)
}
```

## Capa de sesión

- La capa de sesión requiere permitir que los clientes puedan localizar un servidor RPC.
- Puede ser hecho estática o dinámicamente.
- La política de *activación* define si un programa de procedimiento remotos está siempre disponible o tiene que ser iniciado bajo demanda.

## *Middleware* orientado a objeto

El *Middleware* orientado a objeto evoluciona más o menos directamente de la idea de procedimientos remotos.

## Lenguaje de definición de interfaces

- Cada *Middleware* orientado a objeto tiene un lenguaje de definición de interfaces.
- Los IDLs soportan el concepto de tipos de objeto como parámetros; el manejo de fallas y la herencia.

## Ejemplo de un IDL

```
interface Jugador : Object {
    typedef struct Fecha {
        short dia; short mes; short año;
    };
    attribute string name;
    readonly attribute Fecha DoB;
};

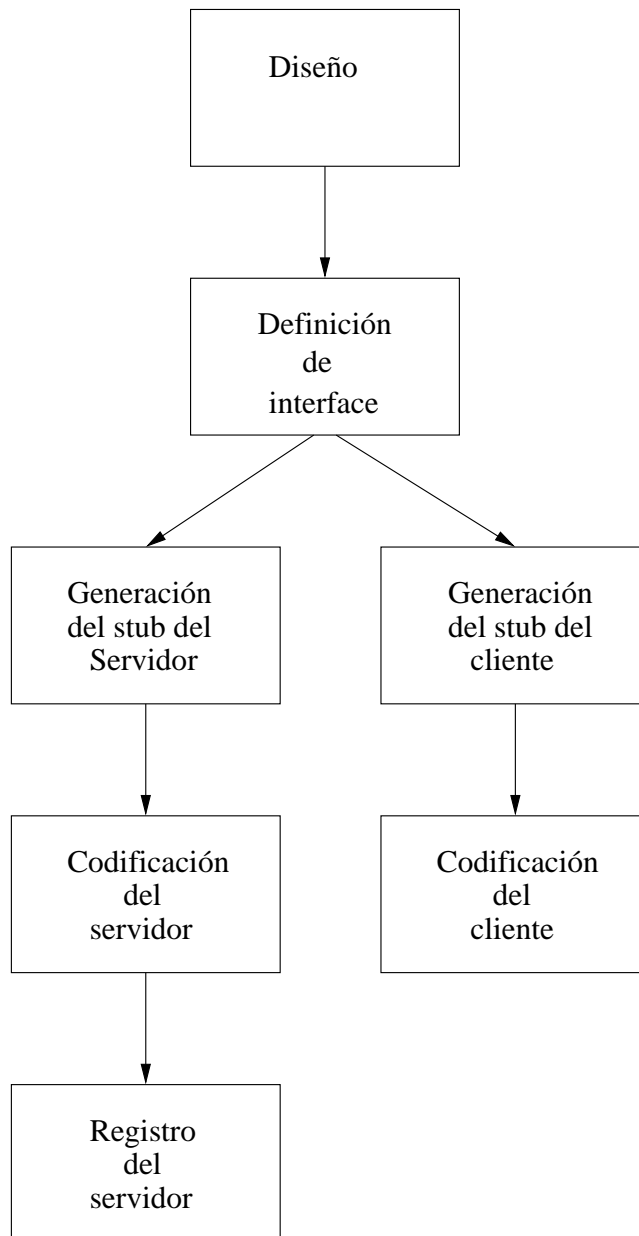
interface BaseDeJugadores : Object {
    exception IDNoEncontrado{};
    short guardar(in Jugador p);
    Jugador cargar(in short id) raises (IDNoEncontrado);
    void imprimir(in Jugador p);
};
```

## Implementación de la capa de presentación

La capa de presentación de un *middleware* orientado a objeto necesita mapear referencias de objetos al formato de transporte.

## Implementación de la capa de sesión

- La capa de sesión necesita mapear las referencias de objetos a *hosts*.
- La capa de sesión implementa las políticas de activación de objetos en el adaptador de objetos.
- Los adaptadores de objetos necesitan estar disponibles para iniciar los servidores, los cuales registran una aplicación en un repositorio o un registro.
- La capa de sesión necesita implementar las operaciones de despacho.
- Las capas de sesión necesitan implementar la sincronización.



## Definición de la interface

- Los IDL suministran un lenguaje para construir todos los conceptos para el modelo soportado de objetos.
- Las definiciones de interfaces adicionan considerables detalles a los diagramas de clases.
- Las interfaces pueden ser vistas como contratos que gobiernan la interacción entre objetos clientes y servidores.
- Las interfaces son también la base para la distribución de información de tipos.
- Los clientes y los servidores son liberados automáticamente de las interfaces.

## Generación del *stub*

- Los *stub* de los clientes y los servidores son *proxies* de servidores y clientes.
- Los *stub* son generados por el compiladores *IDL* que es suministrado por el *middleware*.

## Implementación de los objetos clientes

- Un requerimiento de objeto estático es hecho por una llamada al cliente a través de un método del *stub* del cliente.
- Los *stubs* son tipados y así sus compiladores puede examinar la seguridad de tipos.
- Los *stub* alcanzan transparencia en el acceso.
- El *middleware* puede eliminar los *stub* si ellos residen en el mismo servidor.

## Implementación de los objetos servidores

- Los *stubs* de los servidores generados tiene que llamar a la implementación del servidor que el diseñador de la aplicación estableció.
- Las interfaces y la herencia hace que la implementación de los objetos servidores sean de tipos seguros.

## Registro de los objetos servidores

Los objetos servidores deben ser registrados en el registro o en repositorio de implementación del *middleware*.