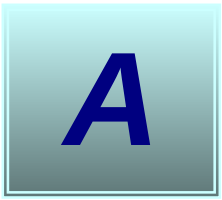
Apéndice



Conectividad

1. SOAP ____ 2

2. CORBA ____ 4

3. JMS ____ 6

4. RMI ____ 8

SOAP

SOAP (Simple Object Access Protocol) es un protocolo basado en XML que permite comunicar componentes y aplicaciones mediante HTTP. Es como hacer RPC mediante HTTP y XML. De hecho es la evolución del protocolo XML-RPC que permite hacer llamadas a procedimientos remotos usando XML como lenguaje común y HTTP como protocolo de transporte.

No tiene el problema de firewalls que existe con CORBA por ejemplo, por trabajar sobre HTTP. Además si se quiere que la comunicación SOAP sea segura alcanza con usar el protocolo HTTPS en lugar de HTTP.

La comunicación entre dos aplicaciones A y B con SOAP funciona de la siguiente forma (suponiendo que B esta del lado del servidor):

La aplicación A realiza un call al servidor (servidor web donde se encuentra B) pasándole el nombre del método de B, y sus parámetros, luego B recibe la llamada y responde. Tanto la llamada como la respuesta son transportadas en la forma de documentos XML. Entre A y B existe un servlet quien se encarga de hacer las "traducciones" y que se puede obtener de alguna de las implementaciones que existen de SOAP, en [implSoap] se da una lista de las mismas. Existen diferentes implementaciones por ejemplo para java, C, Delphi, Applescript, Visual FoxPro, etc, e incluso existen algunas para múltiples lenguajes (se debe elegir una implementacion que este en leguaje de B, en el caso del ejemplo). En el proyecto utilizamos la implementación para java (multiplataforma) de ApacheSOAP [apacheSOAP].

El servlet corre en el servidor web (debe ser un servidor web que soporte servlets, como por ejemplo Tomcat [tomcat], Resin [resin]) y funciona como un traductor, es decir recibe el call en XML de A, obtiene del mismo el nombre del método, los parámetros, y basado en un archivo de configuración (que denominamos deploymentDescriptor.xml, que indica donde se encuentra el objeto o clase -B- que contiene ese método) invoca el método con los parámetros recibidos, éste le retorna un valor (si es que retorna), codifica ese valor nuevamente en una respuesta en XML y lo envía a A.

La importancia de SOAP radica en que es un protocolo independiente del lenguaje, plataforma, muy simple, y que permite además enviar tipos definidos por el usuario, por ejemplo en el caso de java se puede enviar en el call (o en la respuesta) un objeto de tipo "persona" el cual sea definido por el usuario.

A continuación explicamos dos ejemplos realizados, el primero utilizando Tomcat como servidor web (se explica también la configuración del mismo para correr el servlet de SOAP) y java, y sin tipos definidos por el usuario. El segundo utilizando Resin como servidor web, java, y el tipo definido por el usuario llamado "libro".

Ejemplo 1. Servidor web: Tomcat, Lenguaje: java, Impl. De SOAP: Apache SOAP 2.2
(El código fuente se encuentra bajo el directorio doc/Anexos/A-Conectividad/SOAP/1)

Configuración Tomcat para correr el servlet de SOAP.

1. Bajar el software de apache soap (es un .jar) desde [apacheSoap] (suponemos que bajamos la versión 2.2)
2. Descomprimirlo bajo un directorio digamos C:/java/ agregar al classpath el jar C:/java/soap-2_2/soap.jar
3. Bajar y agregar al classpath los siguientes .jar:
    a. mail.jar [JavaMail]
    b. activation.jar [activation]
    c. un jaxp compatible tal como Apache Xerces (versión 1.1.2 o posterior) [apacheXerces]. Si se tiene otro parser de XML debe tenerse cuidado de que este esté primero en el classpath.
4. Agregar C:/java/soap-2_2 al classpath
5. Agregar la siguiente línea al archivo tomcat.bat que se encuentra bajo %TOMCAT_HOME%/bin: set CP=path-de-xerces\xerces.jar;%CLASSAPTH%;%CP%
6. Para hacer el deploy del archivo soap.war que viene bajo soap-2_2/webapps agregar soap.war bajo %TOMCAT_HOME%/webapps

7. Crear un nuevo contexto en %TOMCAT_HOME%/conf/server.xml agregando la línea: <Context path="/soap" docBase="C:/java/soap-2_2/webapps/soap" debug="1" reloadable="true"> </Context>
8. Para verificar que fue configurado correctamente iniciar tomcat y acceder a http://localhost:8080/soap (donde 8080 es el puerto en que esta corriendo Tomcat).

Por información mas detallada ver [instSOAP]

Ejemplo 2. Servidor web: Resin, Lenguaje: java, Impl. De SOAP: Apache SOAP 2.2
(El código fuente se encuentra bajo el directorio doc/Anexos/A-Conectividad/SOAP/2)
Configuración Resin para correr el servlet de SOAP.

1. Repetir los puntos 1 a 4 del Ejemplo1.
2. Copiar la carpeta soap-2_2/webapps/soap bajo Resin/webapps
3. Iniciar Resin
4. Verificar que fue configurado correctamente accedediendo a http://localhost:8080/soap (donde 8080 es el puerto en que esta corriendo Resin).

Ejemplo 3. Servidor web: Resin, Lenguaje: java, Impl. De SOAP: Apache SOAP 2.2
(El código fuente se encuentra bajo el directorio doc/Anexos/A/SOAP/Ejemplos/3). En este directorio se encuentran una serie de ejemplos que aclaran algunas dudas surgidas. En el directorio raíz de cada ejemplo o versión se encuentra documentación particular de cada uno.
Antes que nada se deben crear dos variables de ambiente:

Anexos_SOAP = <Directorio de los anexos SOAP> EJ: D:\doc\Final\doc\Anexos\A-Conectividad\SOAP
Resin_Home = <Directorio de instalacion del Resin Web Server> EJ: D:\Servers\Resin

Para estos ejemplos se debe compilar al directorio %Resin_Home%/classes.
Para ejecutarlos existe en cada directorio un archivo "Comandos.txt" que se puede ejecutar para hacer correr el ejemplo.

CORBA

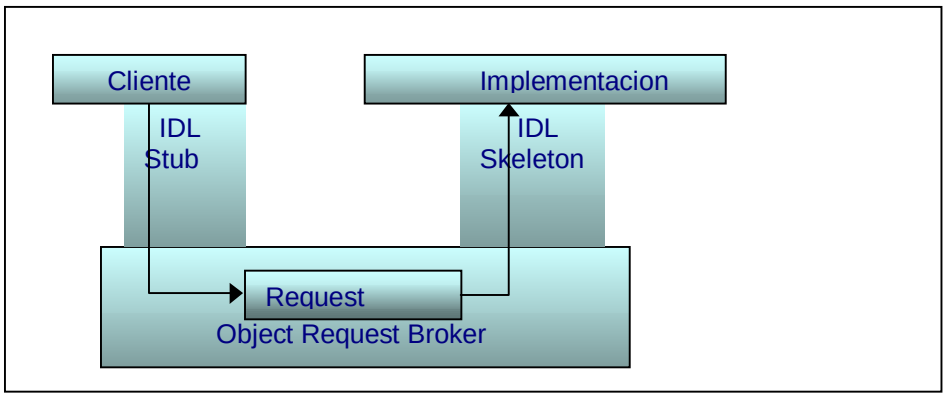
CORBA (Common Object Request Broker Architecture) es una especificación producida por la OMG de la arquitectura e infraestructura que las aplicaciones usan para trabajar conjuntamente sobre una red. Es importante considerar que es una especificación (de software en este caso), esta especificación se puede bajar desde [especCORBA], no una implementación, una lista de implementaciones se puede ver en [implCORBA] (y en [implCORBAFree] se puede ver una de implementaciones libres) . Un programa basado en CORBA, que utiliza una implementación de cualquier vendedor, sobre cualquier computadora, sistema operativo, lenguaje de programación y red, puede interoperar usando el protocolo estandar IIOP, con otro programa basado en CORBA de cualquier otro vendedor o el mismo, en cualquier sistema operativo, lenguaje y red. Esta interoperabilidad resulta de dos partes claves de la especificación: IDL (Interface Definition Language) y los protocolos GIOP y IIOP.

Las aplicaciones CORBA están compuestas de objetos, que representan alguna cosa del mundo real. Normalmente existen muchas instancias de un objeto, por ejemplo en un sitio de comercio electrónico existen muchas instancias del objeto tarjeta, todos idénticos en funcionalidad pero asignados a diferentes clientes. En las aplicaciones CORBA existe normalmente una sola instancia. Para cada tipo de objeto (como tarjeta) se define una interface en IDL, que es parte de lo que el objeto servidor (tarjeta) ofrece a las aplicaciones clientes que lo invocan. Cualquier aplicación cliente que invoque una operación debe usar esa interface en IDL para especificar la operación que desea realizar y sus argumentos. La definición de la interface en IDL es independiente del lenguaje de programación, pero mapea a todos los lenguajes mas populares, por ejemplo existen compiladores de IDL para java, C,C++, COBOL,Smalltalk,Ada, Pitón, etc. [idlCompiladores]

La separación entre la interface y la implementación, facilitada por IDL, es la esencia de CORBA, es decir como hace posible la interoperabilidad, con transparencia. La interface es estrictamente definida (siguiendo IDL) y es publicada a todos los clientes, en cambio la implementación de un objeto esta oculta del resto del sistema (clientes), es decir encapsulada.

Cuando se compila un archivo IDL se generan clases stubs y skeletons, stubs para el lado del cliente, y skeletons del lado del servidor que cumplen el rol de proxies para clientes y servidores respectivamente. Luego deben implementarse esas interfaces, lo que constituye la implementación del objeto. En CORBA cada instancia de objeto tiene su propia object reference, un token que lo identifica. Los clientes usan esa referencia para dirigir sus invocaciones. Tanto los stubs como skeletons corren sobre un (o varios) ORB. El ORB (Object Request Broker) es el que se ocupa de rutear un request desde un cliente a un objeto, y la respuesta al su destino.

El siguiente esquema muestra la arquitectura CORBA básica:



Cliente que envía un request a una implementación de un objeto

Actualmente CORBA es una tecnología que se esta dejando de lado, una de las principales razones es el uso de puertos variables de CROBA cuando se tiene firewalls entre medio de las aplicaciones que intentan comunicarse.

En el proyecto utilizamos la implementación de Visibroker para CORBA. Y tanto los stubs, skeletons e implementaciones de los objetos en CORBA fueron generados por dMOF. Sí se implementaron clientes CORBA para probar dichos objetos. Por ejemplo el cliente que crea un paquete transformacional:

Ejemplo 1. Cliente CORBA que crea un paquete transformacional.

(El código fuente se encuentra bajo el directorio doc/Anexos/A-Conectividad/CORBA/1)

JMS

JMS (Java Message Service) es una API de java diseñada por Sun que permite enviar, recibir, leer y crear mensajes [JMS], [RMI]. Las características principales de JMS son:

- Permite la comunicación distribuida que es débilmente acoplada.
- Asincrónica: Un proveedor de JMS provee los mensajes a los clientes cuando estos llegan (no deben requerirlos ellos cuando llegan).
- Confiable: esta API asegura que un mensaje es liberado solo una vez.

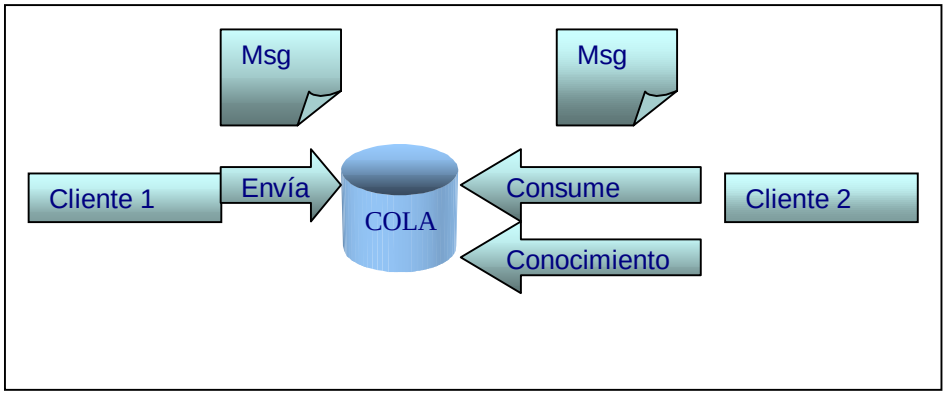
- Y se usa en general cuando:
- Se tienen componentes que pueden ser fácilmente reemplazables
- Una aplicación puede correr cuando no todos los componentes están activos
- El modelo de la aplicación permite a un componente enviar información a otro y continuar operando sin recibir la respuesta inmediata.

Las arquitectura se compone de los siguientes elementos:

1. **JMS provider**: implementa las interfaces de JMS y provee las características administrativas y de control.
2. **Clientes JMS**: programas en java que producen y consumen mensajes
3. **Mensajes**: objetos que comunican información entre clientes JMS
4. **Administered objects**: objetos JMS creados por el administrador para uso del los clientes
5. **Non-JMS clients**: programas que usan el cliente nativo del producto de mensajería en lugar de JMS API.

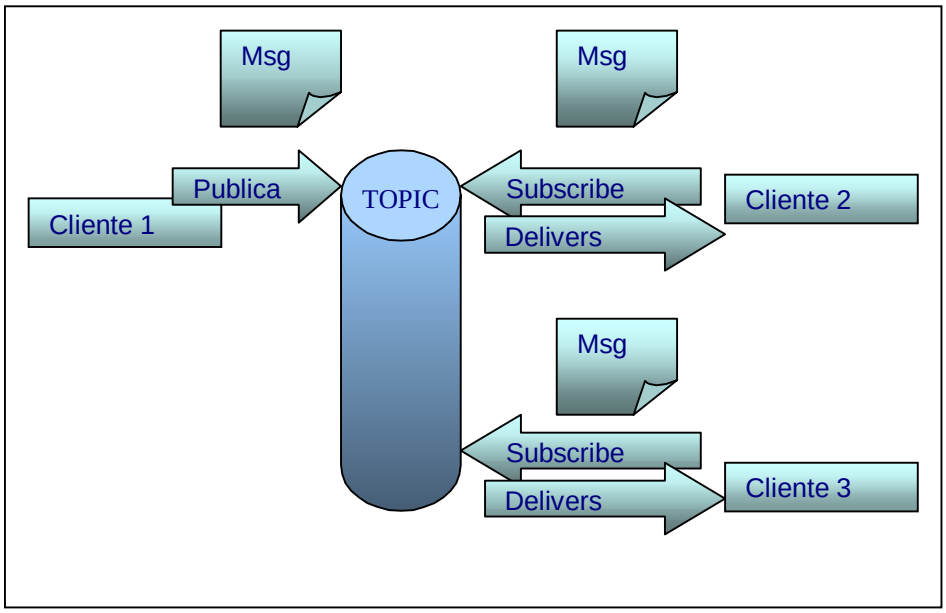
Y existen dos aproximaciones, que son mensajería point to point y Publish/Subscribe.

Point-to-Point. Cada mensaje se envía a una cola y el cliente recibidor lo extrae de la misma. No existen dependencias de tiempo entre el que envía y recibe, el que recibe el mensaje puede ir a buscarlo, este o no corriendo cuando el cliente envía el mensaje. Cada mensaje tiene un solo consumidor.



Mensajería punto a punto

Publish/Subscribe Cada mensaje tiene múltiples consumidores. Existe dependencia de tiempo, el cliente que se subscribe a un "topic" puede consumir solo mensajes que fueron publicados después de que el cliente creo una suscripción, y el que se subscribe debe continuar activo para consumir mensajes.



Mensajería publica/Subscribe

Los mensajes que se envían tienen tres partes en su estructura:

- **Header:** contiene valores que tanto el cliente como el proveedor necesitan para identificar y rutear el mensaje, como por ejemplo un id único, la cola o topic a la cual el mensaje es enviado, un timestamp, nivel de prioridad.

- **Properties**: se pueden usar para proporcionar compatibilidad con otros sistemas de mensajes o para crear message selectors.

- **Body**: donde va la información a enviar propiamente dicha. Incluso se pueden enviar objetos java en esta parte.

RMI

RMI (Remote Method Invocation) es una especificación que esta implementada con APIs de java diseñadas por Sun.

En el modelo de objetos distribuidos de la plataforma java un objeto remoto es uno cuyos métodos pueden ser invocados desde cualquier otra maquina virtual, potencialmente en diferente host. Un objeto de este tipo esta descrito por una o mas interfaces remotas escritas en java.

RMI es análogo a CORBA, salvo que provee un modelo mucho mas simple para objetos distribuidos basados en el lenguaje java. RMI usa como protocolo de transporte el llamado JRMP (Java RMI protocol), el cual es específico de RMI de Java y no puede ser usado para interoperar con otros lenguajes, esta es una de las principales limitaciones de RMI.

RMI es nativo de java, y por ello depende de varias características como serialización de objetos java y definiciones de interfaces java entre otras.

Esta opción se estudio en el proyecto pero fue descartada debido a sus limitaciones de permitir interoperar con otros lenguajes.