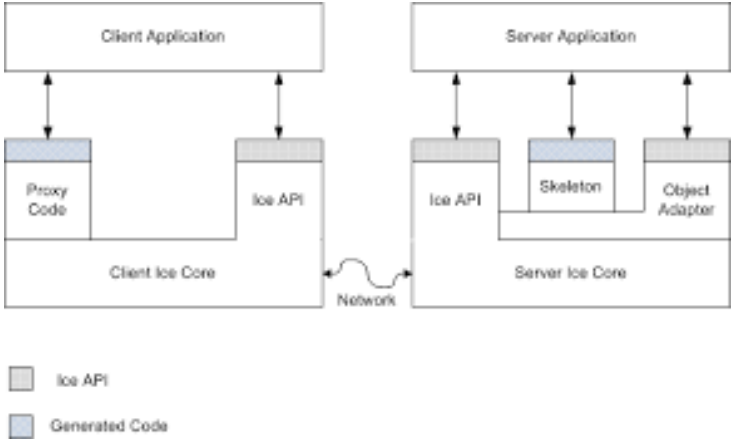
ICE

Computación Distribuida con ZeroC Ice



Introducción

Un **middleware** de comunicaciones es un soisticado sistema de IPC (Inter-Process Communication) orientado a mensajes.

los **middlewares** suelen ofrecer soporte para interfaces concretas entre las entidades que se comunican, es decir, permiten deinir la estructura y semántica para los mensajes.

Introducción

Se puede entender un **middleware** como un software de conectividad que hace posible que aplicaciones distribuidas pueden ejecutarse sobre distintas plataformas **heterogéneas**

Introducción

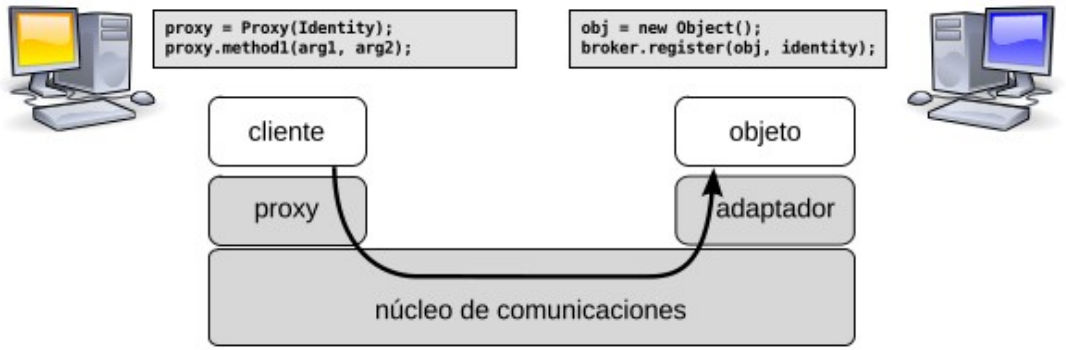
Existen muchos middlewares de comunicaciones: **RPC** (Remote Procedure Call), CORBA, EJB (Enterprise JavaBeans), Java RMI (Remote Method Invo-cation).

En el caso de RPC se indican un conjunto de funciones que podrán ser invocadas remotamente por un cliente.

el ingeniero puede decidir qué entidades del dominio serán accesibles remotamente.

Introducción

Esquema de invocación remota a través de un núcleo de comunicaciones típico de este tipo de middlewares.



El ingeniero puede «particionar»s u diseño, eligiendo qué objetos irán en cada nodo, cómo se comunicarán con los demás, cuáles serán los flujos de información y sus tipos.

Introducción

El cliente utiliza un objeto **proxy** con la misma interfaz definida en la parte del servidor, y que actúa como intermediario. El servidor, por otro lado, utiliza una clase base (una por cada interfaz especificada) encargada de traducir los eventos de la red a invocaciones sobre cada uno de los métodos.

El middleware no puede hacer nada que no puedan hacer los sockets. Pero hay una gran diferencia: con el middleware lo haremos con mucho menos esfuerzo gracias a las abstracciones y servicios que proporciona.

Introducción

El middleware encapsula técnicas de programación de sockets y gestión de concurrencia que pueden ser realmente complejas de aprender, implementar y depurar.

El middleware se encarga también de gestionar problema inherentes a las comunicaciones: identifica y numera los mensajes, comprueba duplicados, controla retransmisiones, conectividad, asigna puertos...etcétera.

ZeroC Ice

Ice (Internet Communication Engine) es un **middleware** de comunicaciones orientado a objetos desarrollado por la empresa **ZeroC Inc**

ZeroC Ice

Ice soporta múltiples lenguajes (Java, C#, C++, ObjectiveC, Python, Ruby, PHP, etc.)

También es multiplataforma (Windows, GNU/Linux, Solaris, Mac OS X, Android, IOS, etc.)

lo que proporciona una gran flexibilidad para construir sistemas muy heterogéneos o integrar sistemas existentes.

ZeroC Ice

Algunas ventajas importantes de Ice:

1. El desarrollo multi-lenguaje no añade complejidad al proyecto

2. Los detalles de coniguración de las comunicaciones (protocolos, puertos, etc.) son completamente ortogonales al desarrollo del software.

3. El interfaz es relativamente sencillo y el signiicado de las operaciones se puede deducir fácilmente.

ZeroC Ice

Especificación de interfaces :

Cuando planteamos una interacción con un objeto remoto, lo primero es deinir el «contrato», es decir, el protocolo concreto que cliente y objeto (servidor) van a utilizar para comunicarse.

Sin las RPC cada nueva aplicación implica deinir un nuevo protocolo, programar las rutinas de serialización y des-serialización de secuencias de bytes, lo que puede ser bastante complejo.

ZeroC Ice

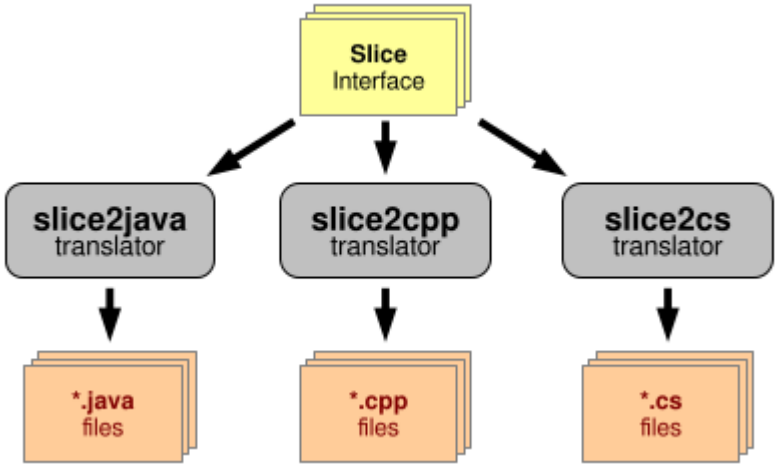
Los middlewares permiten especificar la interfaz mediante un lenguaje de programación de estilo declarativo.

A partir de dicha especificación, un compilador genera código que encapsula toda la lógica necesaria.

A menudo el compilador también genera «esqueletos» para el código dependiente del problema. El ingeniero únicamente tiene que rellenar ese esqueleto con la implementación concreta.

ZeroC Ice

El lenguaje de especificación de interfaces de Ice se llama **Slice** (Specification Language for Ice) y proporciona compiladores de interfaces (translators) para todos los lenguajes soportados



ZeroC Ice

Terminología

Clientes y servidores: hacen referencia a los roles que las diferentes partes de una aplicación pueden asumir durante una petición

Los clientes son entidades activas, es decir, solicita servicios a objetos remotos mediante invocaciones a sus métodos.

Los servidores son entidades pasivas, es decir, proporcionan un servicio en respuesta a las solicitudes de los clientes

Nótese que *servidor* y *cliente* son roles en la comunicación, no tipos de programas. Es bastante frecuente que un mismo programa actúe como servidor (alojando objetos) a la vez que invoca métodos de otros, lo que convierte al sistema en una aplicación *peer-to-peer*.

ZeroC Ice

Objetos: Un objeto Ice es una entidad conceptual o una abstracción que mantiene una serie de características. Es una entidad en el espacio de direcciones remoto o local que es capaz de responder a las peticiones de los clientes.

Proxies: Para que un cliente sea capaz de comunicarse con un objeto ha de tener acceso a un proxy para el objeto. Un proxy es un componente local al espacio de direcciones del cliente, y representa al objeto (posiblemente remoto).

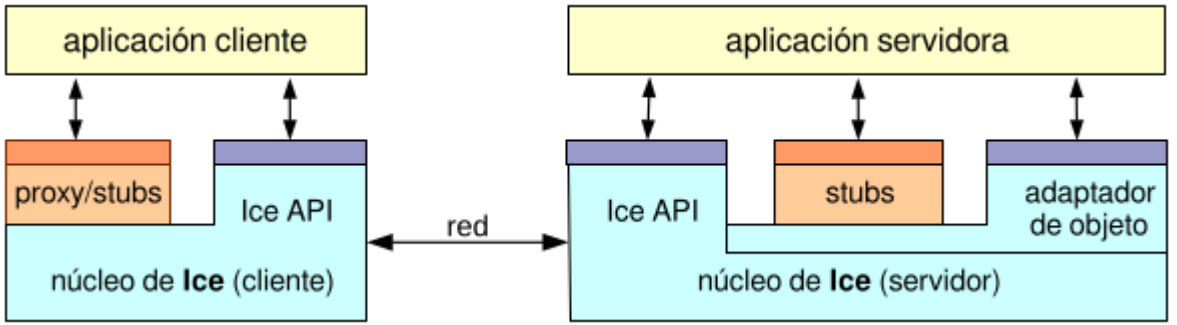
ZeroC Ice

Sirvientes (servants): Las peticiones de los clientes deben terminar en una entidad de procesamiento en el lado del servidor que proporcione el comportamiento para la invocación de una operación.

El componente en la parte del servidor que proporciona el comportamiento asociado a la invocación de operaciones se denomina sirviente. Un sirviente encarna a uno o más objetos distribuidos.

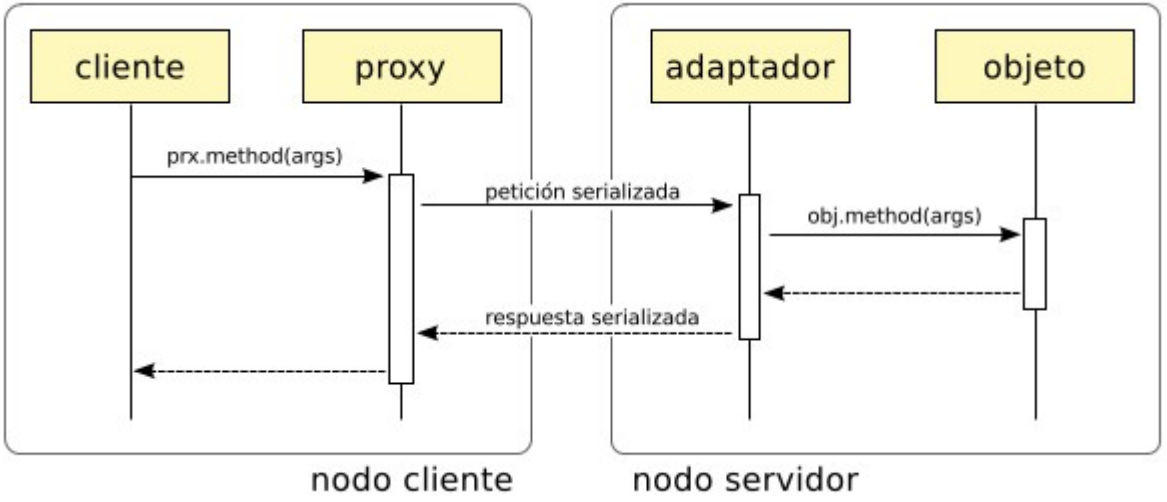
ZeroC Ice

En la siguiente figura se muestran los componentes principales del middleware y su relación en una aplicación típica que involucra a un cliente y a un servidor.



ZeroC Ice

En el siguiente diagrama de secuencia se describe una interacción completa correspondiente a una invocación remota síncrona, es decir, el cliente queda blo queado hasta que la respuesta llega de vuelta.



ZeroC Ice

Workshop