PROGRAMACION CONCURRENTE Y DISTRIBUIDA

VIII.2: ICE Overview

José M. Drake

Notas:

Posibilidades que ofrece Java para la comunicación en red: Socket,RMI y URL.

Arquitectura ICE

- Ice es una plataforma middleware para el desarrollo de aplicaciones basada en objetos distribuidos.
- Es un middleware ligero y abierto que pretende ser compatible con cualquier tipo de plataforma (Windows, Linux, y otras propietarias), y con los principales lenguajes de programación (C,C++, Java, C#, VisualBasic, Phyton, PHP).
- ICE proporciona:
  - APIs y librerías para soportar el paradigma cliente/servidor sobre plataformas distribuidas heterogéneas y utilizando múltiples lenguajes.
  - Servicios y herramientas para soportar las aplicaciones.
- Hereda de CORBA una gran parte de su terminología, aunque difiere de él en algunos conceptos claves.

Notas:

Clientes y servidores

- Representa el modelo evolucionado del paradigma cliente/servidor que corresponde al concepto de objetos distribuidos.
- El concepto de cliente y servidor no representan elementos de una aplicación, sino roles que juegan los elementos en una interacción:
  - Cliente: es el elemento activo en una interacción. Hacen requerimientos a los servidores.
  - Servidor: es el elemento pasivo en una interacción. Proporcionan los servicios que le requieren los clientes.
- Un mismo elemento puede jugar el papel de cliente y servidor:
  - Un servidor para implementar un servicio que ofrece, puede requerir como cliente un servicio de otro elemento.
  - Un cliente puede ofrecer a un servidor una operación de callback. Cuando este la ejecuta está operando como cliente de aquel.

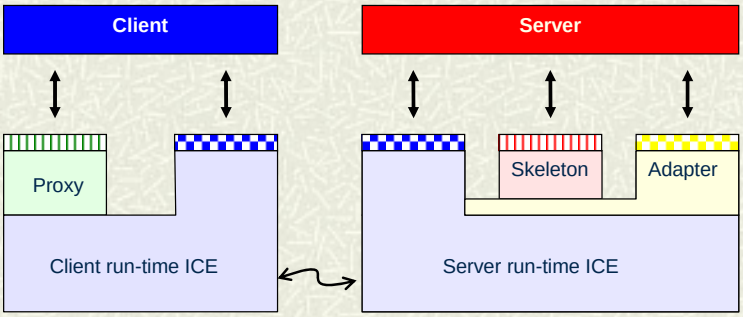
Notas:

Objeto Ice

- Es un ente conceptual local o remoto que puede responder a los requerimientos de clientes:
  - Un objeto ICE tiene una identidad propia que es única en todo el dominio de comunicación de ICE. Si dos objetos tienen la misma identidad ambos son el mismo objeto.
  - Un objeto puede ser accedido por un servant ICE o a través de múltiples servant ICE.
  - Un objeto puede tener múltiples instanciaciones simultaneas en diferentes espacio de direcciones, y sin embargo es el mismo objeto porque tiene la misma identidad.
  - La funcionalidad de un objeto ICE se describe a través de las interfaces que implementan sus facetas. Cada interfaz describe el conjunto de métodos que pueden ser invocados en el puerto que la implementa.
  - Un objeto puede poseer múltiples servicios, cada uno de ellos se denomina faceta y está descrito mediante una interfaz. Una de ellas es la principal. Desde cualquier interfaz, el cliente puede navegar a las otras facetas del mismo objeto.

Notas:

Structura cliente servidor en ICE



- El run-time de ICE es el software de base que existe en cada procesador que soporta objetos ICE y que soporta el intercambio de mensaje que implementa la invocación de mensajes y el retorno de resultados. La interfaz que ofrece se utiliza para tareas de inicialización y administrativas de la distribución.
- El proxy es generado a partir de la definición Slice del servicio. Tiene dos funciones:
  - Proporciona la interfaz local a través del que el cliente realiza la invocación.
  - Proporciona el código para secuencializar y desencuencilizar los datos que se intercambian.
- El esqueleto es generado a partir de la especificación Slice del servidor, y proporciona el código a través del que el run-time de ICE invoca localmente por delegación los métodos que el cliente ha invocado a través el proxy.

Notas:

Structura cliente sevidor en ICE

- El adaptador es parte del API de ICE y realiza en el servidor las siguientes tareas:
  - Traduce en el lado del servidor, los mensajes que llegan en las invocaciones a invocaciones de métodos concretos del servidor.
  - Constituye el elemento que se asocia a una o varias terminaciones de comunicación concretas (dirección + puerto) que son los destino de los mensajes que invocan los métodos del servidor.
  - Es el responsable de la creación de los proxies que pueden ser pasados a los clientes. El adaptador conoce el tipo, la identidad y los detalles de comunicación para acceder al servidor.
- Desde el punto de vista de negocio, sólo hay dos procesos, el del cliente que invoca y el del servidor que ejecuta la invocación. El run-time de ICE proporcionan los elementos ocultos que existen entre ambos.
- Cuando el proxy es indirecto, existe el tercer proceso de localizador, que se utiliza para traducirlo a una información concreta de acceso.

Notas:

Proxy ICE

- Para que un cliente pueda invocar los métodos de las facetas de un objeto ICE, debe disponer de un proxy correspondiente al objeto.
- El proxy es un elemento que existe en el espacio de direcciones del cliente, y representa en él al objeto con el que se comunica.
- Cuando el cliente invoca un método del proxy, el run-time de ICE:
  1. Localiza el objeto ICE.
  2. Activa el servidor correspondiente al objeto si no está activo.
  3. Se activa el objeto dentro del servidor.
  4. Se transmiten los parámetros in del método invocado.
  5. Se espera a que el método termine su ejecución en el servidor.
  6. Retorna al cliente el valor retornado por el método y los parámetros out (o la excepción si es lanzada por el servidor o el run-time)
- El proxy contiene toda la información necesaria para llevar a cabo estas tareas:
  - La dirección (IP + port) que permite acceder al servidor.
  - La identidad que identifica unívocamente el objeto y el server a través del que se accede.
  - Opcionalmente el identificador de la faceta a la que referencia el proxy.

Notas:

Generación del proxy

- Todo proxy puede ser formulado como un string que lo define unívocamente. El run-time de ICE tiene capacidad de crear el proxy a partir del string que lo representa.
- Hay dos tipos de proxies:
  - Proxy directo: Encapsula toda la información (identidad y dirección de acceso).

    "SimplePrinter:tcp –h server1 –p 10001"
  - Proxy indirecto: Contiene la identidad del objeto (well-known object), o la identidad mas el identificador del adaptador a través del que se accede.

    "SimplePrinter"

    "SimplePrinter@PrinterAdapter"

    El proxy indirecto no tiene información de la dirección del objeto. Esta debe obtenerse a través de un servicio de localización.
- ICE admite replicación, esto es, un mismo objeto puede ser accedido a través de múltiples direcciones y adaptadores:

  "SimplePrinter:tcp –h server1 –p 10001 :tcp –h server2 –p 10002"

Notas:

Servant

- El servant o esqueleto es el elemento que introduce ICE para invocar por delegación el método del servidor en su espacio de memoria.
- Un servant es una instancia de una clase cuyos métodos han sido escritor por el diseñador del servicio, y que se registran en el run-time de ICE para este invoque los correspondiente métodos cuando lo haga el cliente a través del proxy.
- Un único servant puede:
  - Dar acceso a un único objeto ICE, el cual está referenciado en el propio servant.
  - O en cada invocación , dar acceso a un diferente objeto ICE. En este caso la identidad del objeto debe ser incluida en el requerimiento.
- Un objeto puede ser accedido a través de múltiples servant diferentes. (Puede ser para acceder con diferentes protocolos, o con diferentes características de comportamiento)

Notas:

Tipos de invocaciones

- Invocación síncrona: El cliente que invoca se suspende hasta que el método invocado ha terminado en el servidor y los resultados han sido retornados. El es método que se utiliza por defecto.
- Invocación asíncrona (AMI): El cliente en la invocación entrega al proxy un objeto de callback, y tras haber conocido que el método ha sido invocado al servidor, recibe el control y continua su ejecución. Cuando la ejecución del método ha terminado, y los resultados han sido transferidos al proxy, este ejecuta el callback, y a través de él, los transfiere al cliente. El servidor no tiene información de si la invocación ha sido síncrona o asíncrona.
- Ejecución asíncrona (AMD): Cuando en el lado del servidor se recibe la invocación, se crea un nuevo thread, que se realiza su ejecución (en concurrencia con otras ejecuciones previas). El servidor recibe información del inicio y finalización de la ejecución a fin de habilitar o inhibir su ejecución, o el retorno de resultados.
- Invocación One way: Sólo es posible en métodos que no retornan resultados. El cliente espera a que la invocación se haya iniciado en el servidor, y luego continua sin esperar a que el método se haya ejecutado.
- Invocación Datagram: Sólo es posible en métodos que no retornan resultados. El cliente tras transferir la invocación, continua sin esperar a confirmar que haya llegado al servidor.
- Invocación Batched Oneway y Datagram: Varias invocaciones de tipo OneWay o Datagram relativas a un mismo servidor son agrupadas en un único mensaje.

Notas:

Excepciones

- Si se presenta un fallo en la ejecución de cualquier invocación de un método a través de ICE se eleva una excepción.
- Las excepciones pueden ser de dos tipos:
  - Excepciones de Ice run-time: Están predefinida en ICE y son generadas en el run-time de ICE. Pueden ser consecuencias de fallos de comunicación, fallos de timeouts, fallos de localización de recursos, etc.
  - Excepciones de usuario: Son generadas como consecuencia de fallos que se detectan en el código del servidor y que han sido declaradas en la especificación Slice del servidor. Estas excepciones pueden contener un conjunto arbitrario de datos que describen la naturaleza u origen de la misma.

Notas:

Propiedades ICE

- El run-time de ICE se configura a traves de propiedades.
- Las propiedades son parejas de nombre-valor:

  “Ice.Default.Protocol=tcp”
- Las propiedades pueden ser establecidas a través de:
  - Un fichero editable de propiedades por de inicialización.
  - En el proceso de instanciación del cliente o servidor como parámetros del correspondiente método main.
  - En tiempo de ejecución, a través de una API específica definida en ICE.

Notas:

Slice (Specification Language for ICE)

- Es un lenguaje abstracto que permite describir las interfaces de los servicios, los métodos que incluyen, los datos que se intercambian y las excepciones que se pueden lanzar.
- Ice describe una correspondencia simple y estricta entre la especificación Slice y los lenguajes C, C++, Java, C#, VisualBasic, .NET, Python y PHP.
- Para cada lenguaje de programación existe un precompilador de Slice, que genera todo el código fuente necesario para construir el cliente o el servidor.

Notas:

Protocolos ICE

- ICE define el protocolo de comunicación que soporta a través del sistema de comunicaciones la invocación de métodos y el retorno de resultados.
- Define protocolos para TCP/IP, UDP y SSL
- El protocolo ICE define:
  - Los tipos de mensajes que utiliza ICE.
  - La máquina de estados que define el protocolo.
  - Las reglas de codificación que se utilizan en cada tipo de mensaje.
  - La cabecera que describe el tipo de mensaje.
- El protocolo ICE soporta mecanismos de compresión, que pueden ser configurados mediante propiedades ICE.
- El protocoloICE soporta sesiones de comunicación bidireccionales, las cuales son necesarias para atravesar los mecanismos de firewall.

Notas:

Servicios ICE

- ICE proporciona un conjunto de servicios que sirven de soporte de aplicaciones distribuidas:
  - IceGrid: Es un servicio de localización de servicios:
    - Resuelve proxies indirectos.
    - Permite automatiza las instanciacion de los servicios
    - Permite replicación y balanceo de carga.
  - IceBox: Permite organizar el inicio y finalización de un conjunto de servidores de una forma coordinada, así como desplegarlos como librerías dinámicas en vez de como procesos.
  - IceStorm: Es un servicio de gestión de eventos que hacen posible desacoplar clientes y servidores.
  - IcePatch2: Es un servicio de distribución de actualizaciones a los clientes.
  - Glacier2: Es un servicio de firewall que permite la comunicación segura entre clientes y servidores a través de trafico encriptado.

Notas:

ICE HelloWorld: Especificación Slice

- Edición del fichero Slice que describe el servidor

Fichero Printer.ice

```
module Demo {
   interface Printer {
      void printString(string s);
   };
};
```

- Compilación de la especificación Slice:

  $ slice2java –output-dir generated Printer.ice

  La compilación genera en la carpeta generated un amplio conjunto de ficheros fuente Java, que pueden ser utilizados para implementar los clientes del servicio y el propio servicio.

Notas:

ICE HelloWorld: Código de implementación del servidor

PrinterI.java

```
public class PrinterI extends Demo._PrinterDisp {
    public void printString(String s, Ice.Current current) {
        System.out.println(s);
    }
}
```

- El servidor hereda a través de la clase abstracta _PrinterDisp todos los recursos de ICE. Esta clase define como métodos abstractos, aquellos que han sido definido con Slice y que deben ser implementados en el servidor.
- El parámetro de tipo Ice.Current aparece en todos los métodos de la interfaz, y da acceso a los recursos del sistema y a los parámetros de la invocación. En este caso no se utiliza.

Notas:

ICE HelloWorld: Código del Servidor

```
public class Server {
   public static void main(String[] args) {
       int status = 0;
       Ice.Communicator ic = null;
       try {
            ic = Ice.Util.initialize(args);
            Ice.ObjectAdapter adapter =
                  ic.createObjectAdapterWithEndpoints("SimplePrinterAdapter", "default -p 10000");
            Ice.Object object = new PrinterI();
            adapter.add(object,Ice.Util.stringToIdentity("SimplePrinter"));
            adapter.activate();
            ic.waitForShutdown();
        } catch (Ice.LocalException e) {e.printStackTrace(); status = 1;
        } catch (Exception e) {System.err.println(e.getMessage()); status = 1;
        }
        if (ic != null) { // Clean up
           try {
                ic.destroy();
           } catch (Exception e) { System.err.println(e.getMessage()); status = 1;
           }
        }
        System.exit(status);
    }
}
```


Notas:

ICE HelloWorld: Código del cliente

```
public class Client {
   public static void main(String[] args) {
      int status = 0;
      Ice.Communicator ic = null;
      try {
          ic = Ice.Util.initialize(args);
          Ice.ObjectPrx base = ic.stringToProxy("SimplePrinter:default -p 10000");
          Demo.PrinterPrx printer= Demo.PrinterPrxHelper.checkedCast(base);
          if (printer == null) throw new Error("Invalid proxy");
          printer.printString("Hello World!");
      } catch (Ice.LocalException e) { e.printStackTrace(); status = 1;
      } catch (Exception e) { System.err.println(e.getMessage()); status = 1;}
      if (ic != null) {   // Clean up
          try {
              ic.destroy();
          } catch (Exception e) { System.err.println(e.getMessage());status = 1;}
      }
      System.exit(status);
   }
}
```


Notas:

ICE HelloWorld: Compilación y ejecución

- La compilación se realiza utilizando el entorno Java habitual
  - $ javac **-d classes -classpath classes:$ICEJ_HOME/lib/Ice.jar**
    -source Server.java PrinterI.java generated/Demo/*.java
  - $ **javac -d classes -classpath classes:$ICEJ_HOME/lib/Ice.jar**
    -source Client.java PrinterI.java generated/Demo/*.java
- La ejecución se realiza en cada entorno de forma convencional
  - $ **java Server**
  - $ **java Client**

Notas: