ICE Evaluation

# Web page

Web page’s design in vertical format with top menu.

**Advantages:**

* Home page has a resume of the product, instead of the typical corporation image. In this resume is pointed out all programming language supported, the different licenses that are offered and other highlights of ICE.
* In the right side of the homepage there is a dynamic textbox that shows the current project where ICE is used.

# Documentation

## Web’s documentation

**Advantages:**

* ICE middleware has a good online documentation.

### Resumen principal

Destacan la simplicidad para entender Ice y la facilidad para programar con él. Aunque sea simple, dicen que es lo bastante flexible para acomodarse incluso en aplicaciones críticas.

Sobre las APIs enfatizan en la simplicidad y la facilidad de uso. Según ellos todas las APIs son thread-safe and exception-safe y que en las APIs de C++ están hechas para que resulte difícil tener memory leaks o memoria corrompida.

Puntualizan su uso de tecnología avanzada, recalcando que no es simplemente un mecanismo RPC. Soporta tanto llamadas síncronas como asíncronas, puede coexistir con firewalls debido a su soporte bidireccional, puede aumentar su eficiencia usando batching para los mensajes.

Dicen permitir la creación de sistemas tolerantes a fallos, usando multiples instancias de un servidor en diferentes ordenadores, con transparencia si una máquina falla o se desconecta.

Según su web, ICE está diseñado para aplicaciones que requieren rendimiento y escalabilidad. Usan un protocolo binario para minimizar el ancho de banda. Usan poca CPU y memoria y tiene eficientes estructuras internas de datos.

Para sistemas distribuidos reales tiene varios servicios que más funcional que los más sofisticados sistemas RPC. Soporta SSL y varios mecanismos de autentificación. Tienen integración con Web.

### Características ICE

#### Lenguage de definición de interfaces

Tiene su propio lenguaje para definir las interfaces y tipos de datos: Slice (Specification Language for Ice). Las estructuras definidas, tales como mapas, se mapean a estructuras conocidas en cada lenguaje. Los mapeos de lenguaje son type-safe, thread-safe and exception-safe. Para C++, Ice añade un gestor de memoria automático para evitar memory leaks.

#### Características RPC

Ofrece invocación y o despacho de forma síncrona o asíncrona. Las invocaciones síncronas son bloqueantes y las asíncronas no lo son. La invocación asíncrona deja libre al hilo y el usuario es notificado más tarde con el resultado. En el lado del servidor también ofrece modo asíncrono, donde no ata un hilo de procesamiento a cada petición.

Soporta tipos dinámicos que no se conocen hasta runtime. Está pensado para crear aplicaciones router o puentes entre protocolos.

#### Caracteristicas de transporte

Se puede elegir entre TCP, SSL y UDP. La gestión de la conexión es totalmente transparente. Ice abre y cierra conexiones según las necesita. ICE usa una codificación compacta binaria que conserva el ancho de banda y es muy eficiente en la serialización y deserialización. Se puede activar compresión también.

## Manuals

**Disadvantages:**

* ICE doesn’t offer manuals in PDF format.

## Installation

**Disadvantages:**

* ICE doesn’t offer an installation manual.

# Software

## Paradigm

The paradigm is RMI.

## Modelo de objetos

Ice es una plataforma middleware para el desarrollo de aplicaciones basada en objetos distribuidos. Representa el modelo evolucionado del paradigma cliente/servidor que corresponde al concepto de objetos distribuidos.

Un 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.

La estructura 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.
* 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.

Un proxy en 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.

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”*

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)

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.

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.

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.

## API

API is written in C++ and it’s oriented to objects.

**Advantages:**

* It uses the C++ templates. Some examples are the creation of smart pointers or guard for mutex.
* API is simple for the user usage. API uses strings that avoid to write code by user. An example is the string “tcp –p 1000” that creates a TCP transport that opens the port 1000.

## Behavior

## Supported programming languages

**Advantages:**

* ICE lets develop distributed application n C++, Java, C#, Python, Ruby, PHP and ActionScript.
* ICE offers option for mobile platforms as Android.

## Platforms

ICE is supported in Windows, Linux, and Mac in 32bit and 64bit. Also it is supported in Solaris.

* **Advantages:**  
  ICE is supported in Android.

## Supported types

Types supported by ICE: basic types, enumerations, structures, sequence and dictionaries. Also it is supported interfaces, operations, exceptions, pointers (proxy operator), and classes.

**Advantages:**

* **ICE supports inherence in classes and interfaces.**
* **There are exceptions.**
* **The serialization mechanism could be specified (ej: CDR).**

## Programming model

Programming with ICE consists in four steps:

1. Define types and services with Slice IDL language.
2. Generates code in the desired programming language using the previous definitions.
3. Escribir la aplicación cliente y compilarla.
4. Escribir la aplicación servidor y compilarla.

Desde el cliente, la forma de conseguir el objeto a usar es:

ConverterPrx cvt = ConverterPrxHelper.checkedCast(

communicator.stringToProxy("converter:tcp -p 10000 -h host.domain.com"));

La forma de crear el objeto en el servidor es:

Ice.ObjectAdapter adapter = communicator.createObjectAdapterWithEndpoints(

"converter", "tcp -p 10000");

adapter.add(new ConverterI(), communicator.stringToIdentity("converter"));

adapter.activate();

En el servidor se pueden crear objetos *ObjectAdapter*. A estos objetos se les puede añadir sirvientes

## Instalación

Para Windows tiene un instalador sencillo. Solo pregunta donde guardar ICE y sus demos.

En Linux está guardado en un fichero rpm.

# Detalles técnicos

## Rendimiento

En los foros hablan de que ICE tiene un buen rendimiento. La peña esta contenta con ello. No he sido capaz de encotrar ninguna gráfica.

Según su whitepaper sobre rendimiento, son los mejores en comparación con WTF en binario y soap.

# Modelo de negocio

## Licencia

La licencia principal es la licencia GPL (GNU General Public License). Licencias comerciales están disponibles para los clientes que desean usar Ice en entorno de software cerrado.

## Proyectos que lo usan

Lo usan en muchos proyectos, tanto libres como cerrados. Algunos de ellos son aplicaciones de trabajan con formatos media (video, audio).

# Notas

El modelo de desarrollo que hemos pensado es similar al de ICE.