CORBA SEGURO
CORBA SLL

Aplicaciones de Objetos distribuidas seguras

Programación para las Comunicaciones
2000

GRUPO

JUAN MÁRMOL CASTILLO
ALEJANDRO ROCA ALHAMA

RECONOCIMIENTO

Desde aquí nuestro agredecimiento a Diego Sevilla Ruiz, diego.sevilla@um.es, que con su enormes conocimientos sobre CORBA, su paciencia y comentarios nos ha ayudado en la elaboración de este trabajo.

Su trabajo de fin de carrera sobre aplicaciones distribuidas ha sido esencial para la realización de estas practicas.

INTRODUCCIÓN

En las páginas siguientes mostraremos cuales son los conceptos básicos utilizados para construir aplicaciones cliente servidor de objetos distribuidos. Aplicaciones basada en la tecnología propuesta por OMG (Object Management Group), CORBA. (Common Object Request Broker Architecture), arquitectura de canal común de gestión de peticiones a objetos.

El trabajo esta organizado en cuatro capítulos

- Planteamiento, que alternativas hemos considerado y cual ha sido elegida.

- Qué, los mínimos conceptos básicos que necesitamos para poder empezar a construir aplicaciones distribuidas java seguras. No será un presentación del estándar CORBA Seguro, ni una justificación del mismo.

- Cómo, construir una aplicación seguras con el producto elegido. Veremos desde la instalación a la realización de una aplicación de ejemplo. Estudiaremos dos formas de crear aplicaciones java seguras que pueden ser complementarias. Presentamos nuestra aplicación. Y finalmente, en este capítulo, consideraremos alternativas a la solución propuesta, y presentaremos algunas cuestiones de interés práctico que surgen en la implantación de nuestra propuesta.

- Conclusiones, presentaremos los hechos más revelantes de la solución CORBA Seguro en el desarrollo de estas aplicaciones teniendo en cuenta contexto de decisión y capacidad tecnología en el ámbito que nos rodea.

TABLA DE CONTENIDOS

Planteamiento

Objetivo

Esta práctica tiene por objetivo presentar de forma escueta y esquemática la tecnología disponible para la creación de servicios CORBA seguros sobre SLL. Describir las nociones básicas y mostrar mediante programas de ejemplo su funcionamiento.

Justificación

Los servicios de información basados en la tecnología orientada a objetos para aplicaciones distribuidas interoperables propuesta por el OMG (Object Management Group), CORBA, va asumiendo cada vez un mayor protagonismo en mundo de la Red.

Aplicaciones tan ambiciosas como la distribución de información y procesamiento de datos del proyecto GENOMA vienen siendo soportadas por la tecnología CORBA. En los últimos años venimos encontrando nuevos casos de éxito en otros dominios como el financiero, médico e industrial. CORBA es la mejor tecnología existente para el desarrollo de aplicaciones cliente/servidor distribuidas[10].

De un lado, el creciente interés y la consolidación de esta tecnología. Y de otro, su aplicación a dominios en los que la privacidad, autenticación y confidencialidad a través de la Red, son requisitos críticos. Ha empujado al OMG a promover nuevos estándares que cubran estos requisitos, estándares para servicios seguros CORBA.

Se entiende como servicios seguros, aquellos donde la comunicación y accesos a los recursos protegidos son realizados con unos niveles aceptables de

- Confidencialidad,
- Privacidad,
- Autenticación.

Cabría pensar que la protección puede ser alcanzada mediante otros medios como por ejemplo IPSec, de forma que las aplicaciones CORBA obviarán la necesidad de protección. Pero no es sólo un problema de comunicaciones; es también un problema de privilegios y derechos de acceso a los servicios, de saber quien es quien, que puede o no puede pedir de forma confidencial.

Por ello, la autenticación es la principal motivación para permear los gestores de accesos a los servicios, los ORB, con la funcionalidad necesaria al respecto. Debe ser así pues, los ORB son el puente entre los clientes peticionarios y los servicios ofrecidos por la aplicación, antes que la propia aplicación pueda realizar una decisión sobre autorización, el ORB la podría haberla efectuado.

Con todo esto es fácil imaginar escenarios donde la aplicación necesita o asume cierta información de control sobre la identidad, como en los servicios de pago por uso, servicios para banca; y otros donde, la seguridad es ortogonal a los servicios, como un servicio de entrega de registros con datos personales.

Estrategia

Analizada la problemática, consideramos dos posibilidades:

- Modificar un ORB para que las comunicaciones sean realizadas a través de canales seguros SSL.
- Buscar productos del mercado.

Inicialmente pensamos seguir la primera, pero abandonamos esta línea a la vista de lo amplio de la especificación estándar a cubrir, y la perdida del principal objetivo de un servicio CORBA, la interoperatividad, poder acceder desde entornos de ejecución heterogéneos.

Comenzamos a buscar en la Red, productos de acceso gratuito que cubrieran la especificación servicios seguros CORBA en vigor. Así nos encontramos con un producto de Adiron, www.adiron.com, que cubre buena parte de la especificación versión 1.7. de corba seguro En el momento de redactar est documento ya esta disponible otras soluciones, una de las cuales nos merece especial interés, pues la base del producto aquí analizado, el ORB seguro de Object Oriented Concepts, www.ooc.com, pero solo da soporte a las comunicaciones seguras usando FreeSSL, no al estándar coba seguro.

Desarrollo de la practica

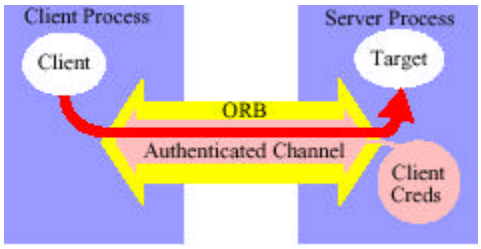
Una vez conseguidos los programas, realizamos una instalación sobre Red Hat 6.2. Superadas las dificultades de instalación y ajuste, obtuvimos la primera parte de este documento. Finalmente, revisamos la bibliografía que teníamos a mano para estudiar otras alternativas y redactar las conclusiones.

¿Qué?

¿QUÉ ES CORBA SEGURO?

CORBA es el acrónimo de Common Object Request Broker Architecture, arquitectura de canal común de gestión de peticiones a objetos, es un marco de desarrollo, framework, estándar para sistemas interoperables de objetos distribuidos. Creado y promovido por el consorcio OMG (Object Management Group), un consorcio de más de 760 empresas que se creó en 1989 con fundadores como Hewlett-Packard o Sun Microsystems.

El estándar define una solicitud CORBA(CORBA request) como una invocación remota de procedimiento aplicada a un objeto dentro del servidor destino.



Una petición CORBA segura (Secure CORBA Request) es una petición normal CORBA que es encapsulada a través de un canal pactado y protegido. El canal es establecido intercambiando la información necesaria entre cliente y servidor de modo que ambos puedan autenticar el uno al otro y viceversa.

La información acerca del canal autenticado es obtenida por medio de la interfaz *Credentials*. En el lado del servidor las credenciales representan la identidad de un cliente.

¿QUÉ ES ORBASEC SL2?

ORBASEC SL2 es un ORB, gestor seguro de peticiones remotas a objetos, que es conforme con las especificaciones de servicio seguros de la arquitectura de canal común de gestión de peticiones a objetos, CORBA, establecida por el OMG. ORBasec SL2, satisface la especificación de nivel de seguridad 2 de la propuesta 1.7 revisión de la especificación de servicio seguro[8].

Las características que ORBasec SL2 soporta son:

- Toda la funcionalidad del ORB, Orbacus 3.3.1 para Java. [9]
- La funcionalidad completa de CORBA Security Level 2.
- Integración del protocolo SECIO, SECure Inter-Operability Protocol.
- Seguridad Suplida.

- Kerberos Version 5 (GSS-API)
- Secure Sockets Layer Version 3 (SSLv3)
- Comunicación desprotegida interoperable (IIOP)

ORBASEC SL2 proporciona al desarrollador de aplicaciones los medios necesarios para plasmar la seguridad en forma de paso de mensajes cifrados y autenticados para realizar y distribuir aplicaciones distribuidas seguras.

¿QUÉ ES ORBACUS?

ORBASEC SL2 esta implementado sobre Orbacus 3.3. para Java, un ORB de Object Oriented Concepts, www.occ.com. Este ha sido incorporado vía el OCI, interfaz de comunicaciones abierta este ORB.

¿QUÉ ES CORBA SECURITY LEVEL 2?

Este es el término usado por la especificación de servicios seguros CORBA que establece un cierto nivel de funcionalidad para el programador de aplicaciones en la forma de API, cuyas características básicas aportadas son:

- Security Manager Object, objeto administrador de seguridad.
- Security Current Object, objeto de la seguridad en la hebra/contexto en curso.
- Credentials Object, el objeto de identidad en curso.
- Principal Authenticator Object, objeto responsable de gestionar la autenticación.
- Diversos runtime Security Policy Objects, objetos para asignación de políticas de seguridad en tiempo de ejecución.

Cada uno de estos objetos puede ser accedido y manipulado para obtener los niveles de seguridad requeridos.

¿QUÉ ES SECIOP?

SECIOP es el acrónimo de SECure Inter-Operability Protocol. Este es el protocolo estándar especificado dentro de la sección de interoperatividad de servicios seguros Corba. Es un protocolo interoperable que usa los estándares formato de señales GSS para entrega de datos autenticados y la protección de datos en una canal de comunicaciones.

¿QUÉ ES LA SEGURIDAD SUPLIDA?

La seguridad reemplazable o intercambiable es un módulo de la especificación de servicios seguros CORBA.

Su principal capacidad es homogeneizar la interfaz de forma que distintos mecanismos de cifrado y autenticación puedan ser empotrados en el ORB y en el SECIOP.

Todo ello motivado por las leyes restricción de exportación de material criptográfico de EEUU y Canadá.

De esta forma se hace posible sustituir unas implementaciones por otras y evitar la debilidad de los métodos exportables según la normativa vigente en EEUU y Canadá, sobre exportación de material criptográfico.

¿QUÉ ES KERBEROS?

Kerberos es una infraestructura de autenticación desarrollada en el MIT estandarizada por la IETF (internet engineering task force).

ORBASEC SL2-GSSKRB es una distribución que contiene un modulo que soporta el GSS-Kerberos del Massachusetts Institute of Technology(MIT). El módulo esta formado por un fichero Java JAR, **GSSKRB.ja**r, y los ficheros librerías de la plataforma de ejecución en cuestión.

El ORBASEC SL2-GSSKRB permite a las aplicaciones interactuar con el estándar de centro de distribución de claves (KDC) para autenticación, RFC 1510[2].

¿QUÉ ES SSL?

SSL es la abreviatura de Secure Socket Layer v3.0. SSL es un protocolo a nivel de puntos de contacto, socket, que permite establecer una conexión segura entre dos entidades de red. Este fue aportado por Netscape, Inc.

La distribución ORBASEC SL2-SSL se apoya en paquete de desarrollo para SSL del Instituto para el procesamiento de información y Comunicaciones Aplicadas de Graz, Austria, es incluido en forma del archivo JAR nombrado **SSL_IAIK.ja**r.

Esto permite a los desarrolladores escribir aplicaciones seguras distribuidas usando certificados X.509 basados en una tecnología pública como DSA y RSA.

No incorpora la interoperatividad con componentes para PKI reconocidos dada la falta de madurez de estándares en este ámbito.

En este trabajo, centramos las pruebas sobre este protocolo.

¿CÓMO SE LICENCIA ORBASEC SL2?

ORBASEC SL2 es distribuido en código abierto que requiere licencias para desarrollo y ocasiones también para ejecución.

Para propósitos académicos, una vez registrado como usuario, se notifican un nombre de entrada y una clave de acceso para extraer de la web de Adiron la distribución.

ORBASEC SL2 necesita de herramientas de terceros para ser usado. Las licencias de estos deben ser proporcionados por sus distribuidores.

- Sun JDK 1.1.x o JDK 1.2 de JavaSoft, Inc., www.sun.com.
- ORBACUS 3.3.1 de Object Oriented Concepts, Inc., www.occ.com.

Además,

REQUISITOS DE ORBASEC SL2-GSSKRB

ORBASEC SL2-GSSKRB necesita un KDC compatible con Kerberos Version 5 accesible y en línea. Puede obtenerse una licencia de:

• MIT Kerberos 1.0.5 o superior (mejor la 1.1.1), del proyecto Athena en Massachusetts Institute of Technology, www.mit.edu.

REQUISITOS DE ORBASEC SL2-SSL

ORBASEC SL2-SSL necesita de

• iSaSiLk 3.0 del IAIK, del Instituto para el procesamiento de la información y las comunicaciones aplicadas de, Graz, de la Universidad de la Tecnología, Graz, Austria. Son también necesarias las librerías *IAIK-JCE 2.51*, una implementación de las interfaces para las extensiones de criptografía para Java.

Además, si se desea emplear cifrado basado en algoritmos propietarios pertenecientes a RSA, Inc.(www.rsa.com):

- Al menos una licencia de desarrollador RSA, Inc.
- Un contrato de asistencia en cliente con Adiron LLC, para posibilitar la utilización de RSA con ORBAsec SL2.

¿QUÉ ES CONTROL? SEGURIDAD ORTOGONAL A IMPLEMENTACIÓN

CONTROL gestor seguro de peticiones remotas a objetos, que es conforme con las especificaciones de servicio seguros de la arquitectura de canal común de gestión de peticiones a objetos, CORBA, establecida por el OMG. ORBasec SL2, satisface la especificación de nivel de seguridad 2 de la propuesta 1.7 revisión de la especificación de servicio seguro[8].

Las características que ORBasec SL2 soporta son:

• Toda la funcionalidad del ORB, Orbacus 3.3.1 para Java.[9]
• La funcionalidad completa de CORBA Security Level 2.
• Integración del protocolo SECIO, SECure Inter-Operability Protocol.
• Seguridad Suplida.
• Kerberos Version 5 (GSS-API)
• Secure Sockets Layer Version 3 (SSLv3)
• Comunicación desprotegida interoperable (IIOP)

CONTROL proporciona al desarrollador de aplicaciones y al administrador los medios necesarios para plasmar la seguridad en forma de paso de mensajes cifrados y

autenticados, y control de acceso automático para realizar, desarrollar y distribuir aplicaciones distribuidas seguras.

CONTROL es un ORB que extiende ORBAsec SL2. A todo lo dicho sobre ORBAsec CONTROL añade la capacidad efectuar decisiones de acceso y gestionar el control de accesos mediante la manipulación de un interceptor, el responsable de capturar y encaminar las peticiones a los servants.

CONTROL permite que el control de acceso a los recursos y las decisiones de acceso sean hechos de forma automática y transparente para las aplicaciones. De forma que están una vez producidas necesiten ninguna o mínima modificación.

¿CÓMO SE LICENCIA CONTROL?

CONTROL es distribuido en código abierto que requiere licencias para desarrollo y ocasiones también para ejecución.

Para propósitos académicos, una vez registrado como usuario, se notifican un nombre de entrada y una clave de acceso para extraer de la web de Adiron la distribución.

CONTROL necesita de herramientas de terceros para ser usado. Las licencias de estos deben ser proporcionados por sus distribuidores.

- Sun JDK 1.1.x o JDK 1.2 de JavaSoft, Inc., www.sun.com.
- ORBACUS 3.3.1 de Object Oriented Concepts, Inc., www.occ.com.

Debemos revisar el apartado correspondiente a ORBasec, pues tiene los mismos requisitos de instalación y distribución.

¿QUÉ ES SAL? UN LENGUAJE PARA CONTROL DE ACCESO EN CORBA SEGURO

El proceso de conceder acceso en cualquier sistema está basado en los conceptos de ***actores***, y ***recursos***, que pueden ser accedidos por dichos actores. Se denominan actores a cualquier entidad que pueda ser autenticada, tales como usuarios, procesos o máquinas. El nivel de autenticación siempre será función del nivel de seguridad subyacente.

Common Object Request Broker Architecture (CORBA) define un estándar de sistemas distribuidos basados en objetos. Los sistemas distribuidos CORBA dependen en los servicios de seguridad de CORBA para proveer del significado necesario para autenticar actores y presentarlos a las aplicaciones de una manera estándar. Para lograr una mejor gestión de las decisiones de acceso en sistemas de gran escala, la seguridad de CORBA introduce la noción de *derechos establecidos* (required rights). Las operaciones definidas en las interfaces son mapeadas en un espacio plano denominado derechos (rights), los cuales deben ser establecidos para un actor para poder acceder a una operación particular. También se proporciona un mapeo de actores a derechos.

El Lenguaje de Accesos a Servidor (SAL) expresa la lógica de las decisiones de acceso basadas en el modelo de seguridad basado en credenciales de CORBA.

Utilización con CORBA

El uso de SAL en arquitectura CORBA actualmente se aplica al control de accesos a objetos dentro de un sólo servidor, aunque no se excluye que varios servidores puedan compartir las mismas descripciones. Cuando un servidor es inicializado, puede tomar una descripción SAL o RRAPSAL desde un fichero, compilarla a SAL normal, la cual está en una estructura InterfaceControl (ICS), y crear el evaluador “eval_ic ICS”. Este evaluador es la implementación que el objeto de CORBA Seguro utiliza. Descripciones complejas en SAL y RRAPSAL pueden ser precompiladas en forma SAL normal y quizás ser optimizadas para rendimientos mayores.

Las actualizaciones dinámicas son realizadas mientras un servidor está ejecutándose modificando la descripción SAL y proporcionándosela al servidor a través de una interfaz de administración segura. Normalmente, una herramienta de administración realiza la compilación RRAPSAL hacia la forma SAL normal antes de proporcionarla al servidor, liberando a éste de realizar la transformación compleja.

Extensiones

SAL permite expresiones regulares para su uso con los valores de los atributos de seguridad. Las construcciones que consultan las credenciales del servidor pueden ser escritas para soportar reglas como “el atributo AccessId del cliente debe coincidir con el del servidor”. Las estructuras de predicados con derechos pueden ser escritas utilizando expresiones de derechos combinándolas con los operadores “any”, “all”, “and”, y “or”.

De momento SAL no soporta el uso de dominios en CORBA. En CORBA, los objetos pertenecen a dominios, los cuáles gobiernan las políticas de acceso.

Cómo

En primer lugar veremos como trabajar con ORBASEC, para seguidamente analizar la forma de trabajar con CONTROL.

ANTES DE COMENZAR

Instalación

La versión instalada para esta practicas ha sido la 2.1.14.

Problemas y soluciones

La instalación ha tenido varios problemas detectados sobre red hat 6.2:

Diferencias entre las variables de entorno asignadas en los archivos cabecera para los archivos de compilación, y las rutas usadas realmente para localizar ciertos archivos. Solución editar alguno de los archivos cabecera.

Alguna librería, libstd++, es referenciada por un nombre distinto del que aparece en la distribución red hat 6.2. Solución ligar el nombre usado en archivos de compilación con el nombre de archivo actual.

Los scripts de prueba utilizan un procesador de comandos, tksh, que no asumían el paso de ciertas variables de entorno, como CLASSPATH, motivado por la configuración de la plataforma. Solución editar los scripts utilizados para usar bash.

El orden correcto de los archivos jar, en la variable de ambiente CLASSPATH. Distinto finalmente del indicado en la documentación.

Los programas de prueba, no consiguen comparar la salida esperada con la salida resultado de las ejecuciones, debido a la utilización de los ficheros iak version 2.6., que introducen mensajes en la salida estándar que no pueden ser suprimidos. Solución modificar los tests.

Alguno de estos problemas han supuesto una gran cantidad de tiempo, por ejemplo determinar la librería correcta libstd++: investigar en distintas distribuciones, determinar la que debe ser realmente,…

Medio de distribución

La distribución entregada vía ftp con ftp.adiron.com es un único archivo en formato ZIP y/o tar comprimido, archivo que una vez descomprimido se despliega en una serie de directorios donde están todos los archivos fuente.

Una vez desplegados debemos procesar con la utilidad Make, en linux, o NMAKE, en WINNT, con esto son construidos:

- Los archivos de clases java, ficheros jar, de los módulos del ORB y CONTROL principalmente, además de los enlaces con los módulos java IAIK.
- Los programas de ejemplos.
- Los ficheros de librería dinámica, dll, enlazados a las librerías de la plataforma usada para el servicio del centro de distribución de claves Kerberos.

Productos y versiones

IAIK-JCE2.6ev iSaSiLk3.0	Librerias de desarrollo iak para criptografia Y conexiones seguras y uso de certificados X509 de
ORBAsec-2.1.4	Orbasec, marco de desarrollo de aplicaciones corba seguras de www.adiron.com
JOB-3.3.2	Orbacus, marco de desarrollo de aplicaciones corba no seguras.
jdk1.2	Marco de desarrollo Java, de www.sun.com

Variables de entorno

Path

```
/usr/kerberos/bin:/usr/kerberos/bin:/usr/kerberos/bin:/sbin:/usr/sbin:/usr/bin:
/bin:/usr/X11R6/bin:/usr/local/bin:/opt/bin:/root/bin:/usr/local/bin:.:/opt/ORB
acus/bin:/opt/jdk1.2/bin:/root/bin:/usr/local/bin:.:/usr/local/bin
```

Alias

```
alias java='java -Xbootclasspath/p:$CLASSPATH'
```

El JDK 1.2

La plataforma JDK 1.2 viene con alguna restricción molesta que puede ser evitada mediante algún rodeo.

La primera es la separación de la CLASSPATH en dos secciones una de usuario y otra de sistema.

La del sistema se conoce como *bootclasspath*, las clases en esta no pueden ser sobrescritas por clases de la sección de usuario. Esto parece indicarnos que las clases clave de java no pueden ser remplazadas. Sun instala un cliente ORB en módulo de ejecución, impidiendo que otros ORB tomen el control. Para eliminar esta restricción, se incluye el modificador –Xbootclasspath, como se puede observar en los ejemplos.

Los Navegadores

Para que puedan funcionar los ejemplos basados en applet y servlet, los navegadores deben proporcionar un entorno de ejecución JDK 1.2 o superior.

Comenzando

A continuación vamos a desgranar un ejemplo, que nos servirá de esquema modelo para el desarrollo de una aplicación.

El ejemplo

El ejemplo es la aplicación “Hola mundo”, un servidor Hello, ofrece un único servicio, que será accedido desde un cliente, con las credenciales de Bart Simson, como resultado de solicitar la ejecución del servicio, se presenta en la salida estándar las credenciales del cliente. El programa es una modificación del ejemplo usado en Orbacus.

Implementar una aplicación con ORBasec SL2, con cliente y servidor se realiza de la misma forma que cualquier otra aplicación CORBA:

- Definir las interfaces en IDL.
- Generar automáticamente los archivos prototipo de plantilla para servidor y enganche para cliente, skeleton y stub, respectivamente, en la nomenclatura CORBA.
- Extender los prototipos base implementando el cuerpo de los métodos.
- Ejecutar, inicializando convenientemente los servicios del ORB.

Las diferencias son en tiempo de ejecución:

- ORBASEC SL2 debe ser inicializado en ambos lados en el cliente y el servidor; y
- El cliente y el Servidor se deben autenticar así mismos mediante **PrincipalAuthenticato**r, un objeto instanciado por ORBasec SL2 administrador del proceso de autenticación.

Una vez que servidor y cliente han sido autenticados, el objeto de las *Credenciales* puede ser modificado para reflejar característica soportadas y requeridas por los mecanismos de seguridad subyacentes. Adicionalmente, diferentes políticas de seguridad pueden ser establecidas sobre el cliente.

Este apartado muestra un ejemplo demostrativo de la autenticación e inicialización por medio del PrincipalAuthenticator.

El código IDL

El código IDL es el mismo que el usado en una aplicación corba no segura.

```
// IDL
module hello {
interface Hello
{
void hello();
```

Como cualquier otra aplicación CORBA sobre Orbacus, traducimos el código IDL

```
jidl --output-dir generated Hello.idl
```

Puesta en práctica de Hello

En la implementación Java del *servant* Hello incluimos el código para imprimir en la salida estándar las credenciales del cliente que realizar la invocación.

```
// Java
package hello;
import org.omg.CORBA.*;
import org.omg.Security.*;
import org.omg.SecurityLevel2.*;
public class Hello_impl extends _HelloImplBase
{
     public ORB orb;
     public void hello()
     {
          try {
               Current current = CurrentHelper.narrow(
               orb.resolve_initial_references(
                                   “SecurityCurrent”));
               ReceivedCredentials c =
                  current.received_credentials();
               orbasec.corba.CredUtil.dumpCredentials(System.out,c);
               System.out.flush();
               } catch (Exception e) {
                    e.printStackTrace(System.out);
               }
     }
}
```

Cuando el método hello de un Objeto Hello es invocado, el ORB solicita una referencia al objeto **SecurityLevel2::Current**; este objeto es usado para obtener información concreta del hilo en curso, información sobre la seguridad establecida en el contexto. Advertimos que necesitamos información sobre seguridad del hilo en cuestión, porque en general, si un servant es un servicio de solicitudes, el ORB puede escoger un hilo distinto para la ejecución de la operación del servant. (Este tema es discutido con detenimiento en la página 84 del manual de Orbasec).

En nuestro ejemplo, queremos obtener los detalles de las credenciales recibidas, el objeto **SecurityLevel2::ReceivedCredentials** de la invocación recibida del cliente vía la asociación de seguridad establecida. Este objeto representa las propiedades de una asociación segura realizada entre el cliente y el servidor. (Este tema es presentado con

detenimiento en pagina 143 del manual). La clase de utilidades **orbasec.corba.CredUtil** usa las interfaces del estándar seguridad CORBA nivel 2 para visualizar las credenciales en forma legible.

Puesta en práctica del servidor

Como en cualquier otra aplicación CORBA, debemos escribir una clase que contenga un método *main* que inicie el servant Hello. Aquí además indicaremos al ORB que inicialize ORBasec SL2 asignando la propiedad **org.omg.CORBA.ORBClass** al valor "orbasec.ORB".

```
// Java
import org.omg.CORBA.*;
import java.util.Properties;
public static void main(String[] args)
{
     // ORB, BOA, and SL2 initialization
     java.util.Properties props = new java.util.Properties();
     props.put( "org.omg.CORBA.ORBClass", "orbasec.ORB" );
     ORB orb = org.omg.CORBA.ORB.init( args, props);
     BOA boa = orb.BOA_init( args, props );
     ...
}
```

Podemos encontrar más detalles sobre la inicialización de Orbasec SL2 en la página 57 del manual.

Una vez inicializado Orbasec, podemos entonces preguntar al ORB por la referencia al objeto administrador de autenticación **SecurityLevel2::SecurityManager**, desde el cual podemos obtener funcionalidad relativa a la seguridad para el servidor. Este objeto tiene por contexto el proceso de la máquina java en ejecución, es global a todos los hilos de esa máquina.

```
// Get SecurityLevel2::Current from ORB
org.omg.CORBA.Object obj =
     orb.resolve_initial_references("SecurityManager");
org.omg.SecurityLevel2.SecurityManager security_manager =
     org.omg.SecurityLevel2.SecurityManagerHelper.narrow(obj);
...
```

La interfaz **SecurityLevel2::SecurityManager** es discutida en detalle en página 73 del manual de Orbasec.

Una vez inicializado orbasec, con la referencia al **SecurityManage**r, obtenemos una referencia al **SecurityLevel2::PrincipalAuthenticato**r, objeto empleado en asignación de las credenciales del servidor.

```
// Obtain PrincipalAuthenticator from SecurityManager
org.omg.SecurityLevel2.PrincipalAuthenticator pa;
```

```
pa = security_manager.principal_authenticator();
```

La interfaz del **PrincipalAuthenticator** es detallada en la página 91 del manual.

Autenticando el servidor usando SSL

La autenticación del servidor usando los mecanismos de seguridad de Adiron, nos proporciona un valor predefinido para el parámetro *mechanism* de la clase de utilidades **orbasec.corba.MechUtil**. Este especifica el nombre de los mecanismos junto con una lista separada por comas de las alternativas de cifrado. En este caso, ha sido elegida certificados DSA-signed Diffe-Hillman Ephemeral, usando cifrado exportable. Los mecanismos de cifrado deben ser compatibles con las claves y certificados empleados en los datos de autenticación.
Podemos encontrar más información al respecto en la página 109 del manual de orbasec
El nombre de seguridad será obtenido a partir de los datos de autenticación. En Kerberos si debe ser asignado el *security_name.*

```
String mechanism =
orbasec.corba.MechUtil.SSL_DHE_DSS_EXPORT_MECH;
String security_name = “”;
```

ORBASEC SL2 ofrece numerosos métodos para la usar la autenticación con SSL. Estos son vistos en la página 113 del manual. En este ejemplo, usamos uno, SSL KeyStore.

Para usar los métodos de autenticación de ORBASEC SL2 debemos proporcionar un **Security::AuthenticationMethod** (un entero **in**t), junto con una referencia **CORBA::Any**, que contendrá una estructura de datos CORBA compatible con los datos del método de autenticación. Aquí, se ha utilizado el método **SecLev2::SecSSLKeyStoreWithStorePass** y su estructura de datos asociada. Este permite usar una cadena certificados y clave privada almacenados en un fichero Java KeyStore.

```
int method =
orbasec.SecLev2.SecSSLKeyStoreWithStorePass.value;
orbasec.SecLev2.SSLKeyStoreWithStorePass data =
orbasec.corba.AuthUtil.default_SSLKeyStoreWithStorePass();
data.keystore = "FILE:./demo.keystore";
data.storetype = "IAIKKeyStore";
data.storepass = "mystorepass";
data.alias = "homer";
data.keypass = "mypassword";
data.usage =
orbasec.SecLev2.CredentialsUsage.SecAcceptOnly;
org.omg.CORBA.Any auth_data = orb.create_any();
orbasec.SecLev2.SSLKeyStoreWithStorePassHelper.insert(
auth_data, data );
```

Obsérvese que usamos el método factoría **orbasec.corba.AuthUtil default_SSLKeyStoreWithStorePass** para crear una instancia por defecto de una estructura **SSLKeyStoreWithStorePass**. Con ello aseguramos que la estructura es

inicializada a los valores por defecto correctos. Entonces usamos el ORB para crear una instancia *Any*, empleando la clase **SSLKeyStoreWithStore-PassHelper** para insertar la estructura en objeto *Any* creado.

El parámetro **auth_data** en el método en el método **authenticate** es un **CORBA::Any**, que contiene la información relativa a SSL. El ejemplo especifica que:
• El KeyStore se encuentra en el archivo *demo.keystore;*
• El tipo de este es IAIKKeyStore. Se asigna el valor a "", para utilizar el tipo por defecto. Esto depende del administrador de la plataforma operativa.
• La contraseña para verificar el contenido del keystore, "mystorepass";
• Las credenciales, identidad, una vez establecida, puede ser usada para recibir solo conexiones seguras desde los clientes, y/o desde los ORB, etc.; según sea indicado.

Una vez asignados los valores para el nombre de seguridad, los mecanismos de cifrado y autenticación, la operación **authenticate** del objeto **PrincipalAuthenticato**r, es invocada usando además los siguientes valores:

```
org.omg.Security.SecAttribute privileges[] =
     new org.omg.Security.SecAttribute[0];
org.omg.SecurityLevel2.CredentialsHolder creds_holder =
     new org.omg.SecurityLevel2.CredentialsHolder();
org.omg.Security.OpaqueHolder
     continuation_data =
          new org.omg.Security.OpaqueHolder(),
     auth_specific_data =
          new org.omg.Security.OpaqueHolder();
pa.authenticate(
     method,
     mechanism,
     security_name,
     auth_data,
     privileges,
     creds_holder,
     continuation_data,
     auth_specific_data
);
```

El parámetro de privilegios establece que privilegios deben ser autenticados a través de los mecanismos de seguridad, pero esto no tiene sentido en SSL.

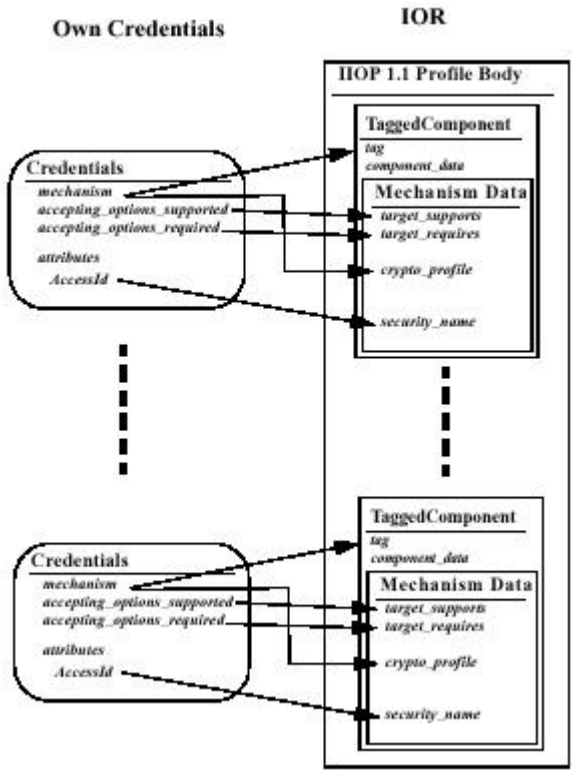
Una vez que el servidor es autenticado, las credenciales son devueltas en una estructura **CredentialsHolder**; también son incorporadas al objeto gestor de autenticación, SecurityManager, en su atributo la lista **own_credentials** para poder ser fácilmente accesible desde otras partes del programa.

Los datos **continuation_data** y **auth_specific_data** no tienen cabida en SSL.

Una instancia de la clase Hello_impl es creada, el IOR para el servant es escrito en un archivo, y el BOA inicia el servicio para admitir solicitudes desde los clientes con el método **impl_is_ready**.

IMPORTANTE IOR

El servidor debe autenticarse así mismo como participante para obtener un objeto con sus credenciales antes de que su IOR sea accesible a los clientes. Esto es importante porque el IOR contiene datos sobre los mecanismos de seguridad que el cliente debe usar para comunicar de forma segura con el servidor.



Implementación del Cliente

El código del cliente es muy similar a como ha sido desarrollado el servidor. El principal factor a tener en cuenta, como en toda aplicación corba, es alcanzar las referencias a los servicios, los IOR. En este ejemplo, la referencia al servidor es almacenada en un archivo hello.ref.

```
// Java
import org.omg.CORBA.*;
import java.util.Properties;
public void main(String[] args)
{
```

```
// ORB and SL2 initialization
java.util.Properties props = new java.util.Properties();
props.put( “org.omg.CORBA.ORBClass”, “orbasec.ORB” );
ORB orb = org.omg.CORBA.ORB.init( args, props );
// Get SecurityLevel2::SecurityManager from ORB
org.omg.CORBA.Object obj =
orb.resolve_initial_references(“SecurityManager”);
org.omg.SecurityLevel2.SecurityManager security_manager =
     org.omg.SecurityLevel2.SecurityManagerHelper.narrow(obj)
// Obtain PrincipalAuthenticator
org.omg.SecurityLevel2.PrincipalAuthenticator pa;
     pa = security_manager.principal_authenticator();
```

Autenticación del cliente usando SSL

La autenticación del cliente usando SSL es similar al proceso descrito para el servidor; especificamos el mecanismo SSL mediante los valores predefenidos de la clase de utilidad orbasec.corba.MechUtil utility, como antes certificados DSA-signed Diffe-Hillman Ephemeral usando cifrado exportable.

Nuevamente usamos el mismo método de autenticación **SecSSLKeyStoreWithStorePass**.

Para más detalles consultar la página 113 del manual.

```
String mechanism =
orbasec.corba.MechUtil.SSL_DHE_DSS_EXPORT_MECH;
String security_name = “”;
int method =
     orbasec.SecLev2.SecSSLKeyStoreWithStorePass.value;
orbasec.SecLev2.SSLKeyStoreWithStorePass data =
     orbasec.corba.AuthUtil.default_SSLKeyStoreWithStorePass();
data.keystore = "FILE:./demo.keystore";
data.storetype = "IAIKKeyStore";
data.storepass = "mystorepass";
data.alias = "bart";
data.keypass = "mypassword";
data.usage =
     orbasec.SecLev2.CredentialsUsage.SecInitiateOnly;
org.omg.CORBA.Any auth_data = orb.create_any();
orbasec.SecLev2.SSLKeyStoreWithStorePassHelper.insert(auth_data,data
```

De nuevo seguimos el mismo proceso que el servidor.

Usamos el método factoría **orbasec.corba.AuthUtil default_SSLKeyStoreWithStorePass** para crear una instancia por defecto de una estructura **SSLKeyStoreWithStorePass**. Con ello aseguramos que la estructura es inicializada a los valores por defecto correctos. Entonces usamos el ORB para crear una

instancia *Any*, empleando la clase **SSLKeyStoreWithStore-PassHelper** para insertar la estructura en objeto *Any* creado.

El parámetro **auth_data** en el método en el método **authenticate** es un **CORBA::Any**, que contiene la información relativa a SSL. El ejemplo especifica que:

• El KeyStore se encuentra en el archivo *demo.keystore;*
• El tipo de este es IAIKKeyStore. Se asigna el valor a "", para utilizar el tipo por defecto. Esto depende del administrador de la plataforma operativa.
• La contraseña para verificar el contenido del keystore, "mystorepass";
• Las credenciales, identidad, una vez establecida, puede ser usada para recibir solo conexiones seguras desde el cliente, y/o desde los ORB, etc.; según sea indicado.

Una vez asignados los valores para el nombre de seguridad, los mecanismos de cifrado y autenticación, la operación **authenticate** del objeto **PrincipalAuthenticato**r, es invocada usando además los siguientes valores:

```
org.omg.Security.SecAttribute privileges[] =
     new org.omg.Security.SecAttribute[0];
org.omg.SecurityLevel2.CredentialsHolder creds_holder =
     new org.omg.SecurityLevel2.CredentialsHolder();
org.omg.Security.OpaqueHolder
     continuation_data =
          new org.omg.Security.OpaqueHolder(),
     auth_specific_data =
          new org.omg.Security.OpaqueHolder();
pa.authenticate(
     method,
     mechanism,
     security_name,
     auth_data,
     privileges,
     creds_holder,
     continuation_data,
     auth_specific_data
);
```

La operación **authenticate** del gestor de autenticación debe ser invocada después de inicializar ORBasec SL2 y antes de hacer ninguna petición al objeto Hello.

Compilando los programas de demostración

La compilación de este ejemplo es usando la utilidad make en el directorio `sl2/demo/ssl-hello`.

Obtendremos una salida similar a la siguiente:

```
mkdir classes
mkdir generated
jidl --output-dir generated Hello.idl
CLASSPATH=./classes:$CLASSPATH \
javac -deprecation -d classes \
```

```
generated/hello/*.java
```

Ejecución

Ejecutar el ejemplo implica arrancar el Servidor y seguidamente el Cliente. El servidor se inicia primero para grabar el fichero Hello.ref, que contiene la referencia CORBA por la que el servidor puede ser accedido.

Recuérdese definir correctamente la variable de entorno CLASSPATH y en su caso introducir un alias para el ejecutable java, que incluya el parámetro xbootclasspath=$CLASSPATH.

Ejecución del Servidor

```
java hello.Server
```

Obtenemos la siguiente salida de la que hemos quitado los mensajes de las libreria IAIK.

```
Own Credentials:
>>>>> Credentials:
credentials_type = SecOwnCredentials
mechanism = SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
accepting_options_supported =
[Integrity,Confidentiality,DetectReplay,DetectMisordering,EstablishTrus
tInTarge
t,EstablishTrustInClient,NoDelegation]
accepting_options_required =
[Integrity,DetectReplay,DetectMisordering,NoDelegation]
invocation_options_supported = []
invocation_options_required = []
7 Security Attributes: (definer,family,type,def_auth,value)
SecAttribute(41244,1,2,<empty>,1485)
SecAttribute(41244,2,2,<empty>,CN=AdironCA, O=Adiron, L=Syracuse, C=US)
SecAttribute(41244,2,1,<empty>,L=Springfield, CN=homer, O=MYREALM.COM,
C=US)
SecAttribute(0,1,2,<empty>,L=Springfield, CN=homer, O=MYREALM.COM,
C=US)
SecAttribute(41244,1,1,<empty>,192.168.0.0)
SecAttribute(41244,0,0,<empty>,
SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA)
SecAttribute(0,1,1,<empty>,)
<<<<< Credentials
Hello Server is Ready.
```

El servidor imprime el certificado devuelto tras la autenticación, las credenciales obtenidas vía el gestor de autenticación **PrincipalAuthenticator** y su método **authenticate**. Vemos que los parámetros **invocation_options_supported** y **invocation_options_required** están vacíos, porque las credenciales han sido asignadas usando la constante de modo de uso **SecAcceptOnly.**

Ejecución del Cliente

```
java hello.Client
```

Obtenemos la siguiente salida de la que hemos quitado los mensajes de las libreria IAIK.

```
Own Credentials:
>>>>> Credentials:
credentials_type = SecOwnCredentials
mechanism = SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
accepting_options_supported = []
accepting_options_required = []
invocation_options_supported =
[Integrity,Confidentiality,DetectReplay,DetectMisordering,Establi
shTrustInTarge
t,EstablishTrustInClient,NoDelegation]
invocation_options_required =
[Integrity,DetectReplay,DetectMisordering,NoDelegation]
6 Security Attributes: (definer,family,type,def_auth,value)
SecAttribute(41244,2,2,<empty>,CN=AdironCA, O=Adiron, L=Syracuse,
C=US)
SecAttribute(41244,2,1,<empty>,L=Springfield, CN=bart,
O=MYREALM.COM,
C=US)
SecAttribute(0,1,2,<empty>,L=Springfield, CN=bart, O=MYREALM.COM,
C=US)
SecAttribute(41244,1,1,<empty>,192.168.0.1)
SecAttribute(41244,0,0,<empty>,
SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA)
SecAttribute(0,1,1,<empty>,)
<<<<< Credentials
Getting Hello Reference
Hello's Credentials
>>>>> Credentials:
credentials_type = SecTargetCredentials
mechanism = SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
accepting_options_supported = []
accepting_options_required = []
invocation_options_supported = []
invocation_options_required = []
association_options_used =
[Integrity,Confidentiality,DetectReplay,DetectMisordering,Establi
shTrustInClien
t,NoDelegation]
9 Security Attributes: (definer,family,type,def_auth,value)
SecAttribute(41244,2,2,<empty>,CN=AdironCA, O=Adiron, L=Syracuse,
C=US)
SecAttribute(41244,2,1,<empty>,L=Springfield, CN=homer,
O=MYREALM.COM,
C=US)
```

```
SecAttribute(0,1,2,<empty>,L=Springfield, CN=homer,
O=MYREALM.COM, C=US)
SecAttribute(41244,1,4,<empty>,1485)
SecAttribute(41244,1,3,<empty>,192.168.0.0)
SecAttribute(41244,1,2,<empty>,1514)
SecAttribute(41244,1,1,<empty>,192.168.0.1)
SecAttribute(41244,0,0,<empty>,
SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA)
SecAttribute(0,1,1,<empty>,)
<<<<< Credentials
```

Después de que el cliente obtenga y muestre sus propia credencial, obtiene la referencia al objeto Hello, y muestra sus credenciales. Análogamente, al servidor, se ha utilizado el modo **SecInitiateOnly**.

Para continuar con la demo, pulsamos h, y obtenemos la siguiente salida:

```
Received Credentials:
>>>>> Credentials:
credentials_type = SecReceivedCredentials
mechanism = SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
accepting_options_supported = []
accepting_options_required = []
invocation_options_supported = []
invocation_options_required = []
association_options_used =
[Integrity,Confidentiality,DetectReplay,DetectMisordering,NoDeleg
ation]
delegation_mode = SecDelModeNoDelegation
delegation_state = SecInitiator
9 Security Attributes: (definer,family,type,def_auth,value)
SecAttribute(41244,2,2,<empty>,anonymous)
SecAttribute(41244,2,1,<empty>,anonymous)
SecAttribute(0,1,2,<empty>,anonymous)
SecAttribute(41244,1,4,<empty>,1514)
SecAttribute(41244,1,3,<empty>,192.168.0.1)
SecAttribute(41244,1,2,<empty>,1485)
SecAttribute(41244,1,1,<empty>,192.168.0.0)
SecAttribute(41244,0,0,<empty>,SSL_IAIK,SSL_DHE_DSS_EXPORT_WITH_D
ES40_CBC_SHA)
SecAttribute(0,1,1,<empty>,)
<<<<< Credentials
```

Están son las credenciales impresas por la implementación del objeto Hello, Hello_impl. Recordamos que no soporta delegación de credenciales, por lo que estas no pueden ser usadas para crear o iniciar nuevas asociaciones de seguridad.

Podemos observar que **Integrity** y **Confidentiality** son ambas usadas, pero no **EstablishTrustInClient** y **EstablishTrustInTarget**, indicando que ni el servidor ni el cliente se autentican así mismos para el otro.

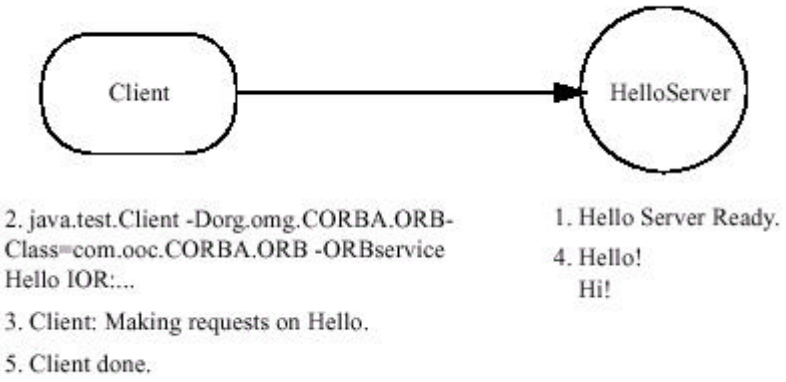
IOR y seguridad

Debemos hacer constar que aunque en esta práctica usamos explícitamente los IOR, lo que puede aparecer como una limitación a distribución de la aplicación, esto tiene por que ser así. La solución para este extremo son los servicios de localización por nombre y por perfil de propiedades o atributos, los *name services* y los *trading servicies*, con los cuales podemos utilizar nombres lógicos y consultas sobre propiedades para acceder a los objetos que ofrecen los servicios CORBA SLL.

CONTROL

Comenzando

En este apartado intentaremos configurar y poner en marcha un servidor CONTROL. Empezaremos con un ejemplo básico cliente/servidor. Entonces usaremos



CONTROL como ORB.

Para entender cómo funciona CONTROL, comenzaremos con el ejemplo de los apartados anteriores, teniendo un objeto y un cliente que puedan interactuar sin seguridad utilizando la arquitectura CORBA sin utilizar CONTROL.

Ejemplo sin CONTROL

Este ejemplo consta de un cliente (programa “Hello”) y un programa servidor. El cliente invoca las operaciones “hello” y “hi” del servidor y el servidor imprime las palabras “Hello!” o “Hi!” según corresponda. El ejemplo incluye un script que arranca ambos, cliente y servidor, volcando la salida del terminal a ficheros de texto para su posterior examen.

Para continuar, nos aseguraremos que en el path existe el directorio ./classes seguidos de los correspondientes ficheros “jar” en este mismo orden:

./classes:Control.jar:OBX.jar:SL2.jar:SSL_IAIK.jar:OB.jar

En el directorio del ejemplo teclearemos
make all

Este comando debería generar los directorios “generated” y “classes. Debemos asegurarnos antes que los ejecutables java, javac, jidl, irserv y irfeed se encuentran la variable de entorno PATH.

Run0

Este comando se asegura que todo se está ejecutando correctamente, asegurando la ejecución solitaria de ORBAcus ORB, sin ORBAsec SL2’s ORB y sin el ORB de CONTROL.

Para ejecutar el ejemplo basta escribir el comando

run 0

Este comando ejecutará el ejemplo arrancando el servidor HelloServer en segundo plano. Entonces ejecutará el cliente en primer plano. Es un script UNIX escrito para *csh* (nota: usando JDK 1.2, se asume la existencia del siguiente alias-> alias java=’java –Xbootclasspath:/$CLASSPATH’)

```
#!/bin/tcsh
#
# Start the server in the background.
# This is job 1
#
java test.HelloServer \
-Dorg.omg.CORBA.ORBClass=com.ooc.CORBA.ORB \
|& tee server.out &
sleep 1
#
# Start the client
#
java test.Client
-Dorg.omg.CORBA.ORBClass=com.ooc.CORBA.ORB \
-ORBservice Hello ‘cat Hello.ref‘
|& tee client.out
#
# Clean up
#
kill %1 %2
```

La salida de los programas las podemos encontrar en los ficheros “server.out” y “client.out”. La salida se muestra también en pantalla usando pipes y el comando UNIX “tee”.

Especial atención merece la salida “server.out” ya que al mostrar las líneas “Hi!” y “Hello!” podemos estar seguros de que las solicitudes del cliente han sido recibidas y realizadas.

Usando SSL

Ahora vamos a ejecutar el código del mismo ejemplo con diferentes configuraciones, permitiendo el uso de autenticación, credenciales SSL, para ambos, cliente y servidor. Recordamos que las credenciales son las interfaces del programador para consulta de atributos de seguridad pertenecientes a la propia aplicación y a las de cualquier cliente que realice peticiones.

Run1

Usaremos el script **run1**, muy parecido al anterior pero con algunos cambios:

```
#!/bin/tcsh
# Start the server in the background.
# This is job 1
#
java test.HelloServer \
-ORBconfig h1.config \
|& tee server.out &
sleep 1
#
# Start the client
#
java test.Client
-ORBconfig c1.config \
|& tee client.out
#
# Clean up
#
kill %1 %2
```

Los cambios son los siguientes:

- En ningún momento especificamos el ORB; como la variable de entorno CLASSPATH contiene el fichero Control.jar, la clase será automáticamente cargada para el ORB, y
- Usamos un fichero de configuración ORB para la configuración (utilizando el argumento –ORBconfig).

El fichero de configuración *h1.config* contiene la configuración de *HelloServer* para el script *run1*. El fichero de configuración *c1.config* contiene la configuración para *run1*.

El programa HelloServer consigue su configuración utilizando el autenticador de CONTROL para inicializar las credenciales SSL del servidor:

```
control.authentication=ssl
control.authentication.ssl.auth_method=pwsupplied
control.authentication.ssl.mutual_auth=true
control.authentication.ssl.usage=AcceptOnly
control.authentication.ssl.keystore=FILE:demo.keystore
control.authentication.ssl.keystore.storetype=IAIKKeyStore
control.authentication.ssl.keystore.storepass=”blahblah”
control.authentication.ssl.keystore.keyalias=marge
control.authentication.ssl.keystore.keypass=”mypassword”
```

El ciente consigue configurarse utilizando también el autenticador de CONTROL para inicializar sus credenciales SSL:

```
control.authentication=ssl
control.authentication.ssl.auth_method=pwsupplied
control.authentication.ssl.mutual_auth=true
control.authentication.ssl.usage=InitiateOnly
control.authentication.ssl.keystore=FILE:demo.keystore
control.authentication.ssl.keystore.storetype=IAIKKeyStore
control.authentication.ssl.keystore.storepass=”blahblah”
control.authentication.ssl.keystore.keyalias=bart
control.authentication.ssl.keystore.keypass=”mypassword”
control.service.Hello=FILE:Hello.ref
```

El fichero de configuración del cliente contiene una propiedad “control.service” indicando que el ORB del cliente encontrará la referencia para el servicio *Hello* en el fichero *“Hello.ref”*, el cual está generado por *HelloServer*.

En los ficheros de configuración se han usado contraseñas proporcionadas en el mismo fichero de configuración como método de autenticación. Esto se ha hecho para automatizar el ejemplo, si se quiere utilizar contraseñas que se soliciten al usuario de forma interactiva, basta con utilizar los métodos de autentificación “pwtext” y “pwgui” en sus respectivos ficheros de configuración.

Run1

Para ejecutar el ejemplo, basta con teclear:

```
run 1
```

Los resultados de la ejecución se guardarán en el fichero *”client.out”* indicando que ambas operaciones *“hi”* y *“hello”* han sido permitidas.

CONTROL con Políticas de Acceso Iniciales

El siguiente paso es ejecutar un ejemplo con una política de acceso por defecto escrita en SAL (Server Access Languaje). Usaremos la siguiente política de acceso SAL para permitir el acceso a la operación “hi” y denegarlo para el acceso a “hello”. El fichero con esta configuración es “Server2.sal” y contiene lo siguiente:

```
(Apply ((OperationControl (
     (“hi” Allow)
     (“hello” Disallow)))
Disallow))
```

El fichero de configuración “h2.config” contiene una línea más que “h1.config” indicando al servidor dónde encontrar la política de control. Esta línea es:

```
control.access.description=FILE:Server2.sal
```

El fichero de configuración para el cliente, “c2.config”, mantiene lo mismo que “c1.config”. El script “run2” básicamente permanece inalterado, con la salvedad de indicar que es la salida de run2.

Run2

Para ejecutar el ejemplo, basta con teclear:

```
run 2
```

Los resultados de la ejecución se guardarán en el fichero *”client.out”* indicando que la operación *“hi”* ha sido permitida y la operación *“hello”* ha sido denegada.

CONTROL con Otras Políticas de Acceso

Vamos ahora a desarrollar un ejemplo con una política SAL más compleja. La siguiente política es equivalente a la anterior, sin embargo, la anterior política no hacía ninguna mención a la interfaz del objeto. Para ilustrar como las políticas pueden ser definidas y usadas por nombre dependiendo del nombre de la interfaz, definimos la siguiente política usada por *“run3”* y contenida en el fichero *“Server3.sal”*:

```
(OperationControl HiNoHello “IDL:test/Hello:1.0”
     ((“hi” Allow)
     (“hello” Disallow)
     (“_is_a” Allow)
     (“_interface” Allow)
     (“_implementation” Allow)
     (“_hash” Allow)
     (“_is_dynamic” Allow)
     (“_is_equivalent” Allow)
     (“_non_existent” Allow)
     (“*” Disallow)))
(InterfaceControl TheControl
     ((“IDL:test/Hello:1.0” HiNoHello)))
(Apply ((InterfaceControl TheControl Disallow))
```

El fichero de configuración *“h3.config”* contiene lo mismo que *“h2.config”* excepto para la siguiente propiedad:

```
control.access.description=FILE:Server3.sal
```

Run3

Para ejecutar el ejemplo, basta con teclear:

```
run 3
```

Los resultados de la ejecución se guardarán en el fichero *"client.out"* indicando que la operación *"hi"* ha sido permitida y la operación *"hello"* ha sido denegada.

CONTROL con Políticas de Acceso basadas en Predicados

Ahora vamos a desarrollar u ejemplo con una política más compleja usando predicados de credenciales simples. La siguiente línea en "Server4.sal" contiene una línea, la cual carga los tipos de atributos que se definen para ORBAsec SL2.

```
(import "Attributes.sal")
```

Las credenciales basadas en predicados para los actores (usuarios, procesos o máquinas a ser autenticados) del cliente son:

```
(CredentialsPred isBart
     (SubjectId "L=Springfield, CN=bart, O=MYREALM.COM, C=US"))

(CredentialsPred isAnyonefromMyRealm
     (SubjectId '.*O=MYREALM.COM.*'))
```

Estos dos predicados son VERDAD si el SubjectId es el nombre de Bart, que es el actor cliente en el ejemplo o cualquiera conteniendo "O_MYREALM.COM", que también es cierto para el nombre de Bart. Reescribimos la política del anterior ejemplo para usar estos predicados:

```
(OperationControl HiNoHello "IDL:test/Hello:1.0"
     (("hi" ((isBart Disallow) (true Allow)))
     ("hello" ((isAnyoneFromMyRealm Allow)
     (true Disallow)))
     ("_is_a" Allow)
     ("_interface" Allow)
     ("_implementation" Allow)
     ("_hash" Allow)
     ("_is_dynamic" Allow)
     ("_is_equivalent" Allow)
     ("_non_existent" Allow)
     ("*" Disallow)))
(InterfaceControl TheControl
(("IDL:test/Hello:1.0" HiNoHello)))

(Apply ((InterfaceControl TheControl Disallow))
```

El fichero de configuración "h4.config" contiene los mismo que "h3.config" excepto para la siguiente propiedad:

```
control.access.description=FILE:Server4.rsal
```

La razón de que el fichero contenga en esa propiedad un fichero con la extensión “rsal” en vez de un fichero con la extensión “sal” es que el fichero “Server4.sal” debe ser reducido primero. El fichero “Server4.sal” contiene una declaración import, la cual es ilegal para la inicialización del servidor. Esta restricción existe porque no hay forma de especificar el path de búsqueda para encontrar tales ficheros. Esto se manifiesta cuando hay que conseguir cambiar la política de forma remota. Para evitar este problema , reducimos el fichero “Server4.sal” en el fichero “Server4.rsal” usando el siguiente comando antes de arrancar el servidor:

```
java control.tools.SALReduce Server4.sal > Server4.rsal
```

Run4

Para ejecutar el ejemplo, basta con teclear:

```
run 4
```

Los resultados de la ejecución se guardarán en el fichero *”client.out”* indicando que la operación *“hello”* ha sido permitida y la operación *“hi”* ha sido denegada.

Cambio Dinámico de las Políticas de Acceso

Ahora vamos a intentar demostrar el uso de la herramienta “ControlAdmin” para la actualización dinámica de las políticas de acceso. Usaremos dos ficheros de políticas diferentes, *“Server5.1.sal”* y *“Server5.2.sal”*.

El primer fichero es la política inicial y el segundo la política con la que queremos actualizar el servidor. Ambos ficheros importan un fichero común, *“Server5.sal*”, del cual *“Server5.1.sal”* y *“Server5.2.sal”* contienen una mínima descripción. El fichero *“Server5.1.sal”* contiene:

```
(Apply ((InterfaceControl Control1) Disallow))
```

Y el fichero *“Server5.2.sal”* contiene:

```
(Apply ((InterfaceControl Control2) Disallow))
```

Las definiciones de la interfaces llamadas *“Control1”* y *“Control2”* se encuentran en el fichero *“Server5.sal”*. Este fichero contiene las políticas de los ejemplos previos con ligeras ampliaciones.

Primero, se ha añadido un Predicado de Credencial para proteger el uso de AccessManager del objeto servidor. Este predicado de credencial se usará posteriormente para asegurar que el actor que invoque al AccessManager es el mismo actor que el servidor.

```
(CredentialsPredicate isSamePrin
     (SubjectId (ValueOf Server SubjectId)))
```

Usamos este predicado en el desarrollo de una operación de control para proteger las operaciones del AccessManager, restringiendo el acceso a ellos sólo al actor que opere en el servidor.

```
(OperationControl SAM_SamePrinOnly
     "IDL:adiron.com/AccessControl/AccessManager:1.0"
          (("*" ((isSamePrin Allow)
               (true Disallow)))
```

Añadimos una nueva operación de control para el objeto Hello que permitirá el acceso de Bart a la operación *"hi"*, pero no permitirá el acceso a nadie desde "MYREALM.COM" para acceder a la operación *"hello*"

Esta política se forma cambiando el significado de los controles *"Allow"* y *"Disallow"* en la operación de control "BartNoHi" del ejemplo anterior.

```
(OperationControl HiNoHello "IDL:test/Hello:1.0"
     (("hi" ((isBart Allow) (true Disallow)))
     ("hello" ((isAnyoneFromMyRealm Disallow)
     (true Allow)))
     ("_is_a" Allow)
     ("_interface" Allow)
     ("_implementation" Allow)
     ("_hash" Allow)
     ("_is_dynamic" Allow)
     ("_is_equivalent" Allow)
     ("_non_existent" Allow)
     ("*" Disallow)))
```

Ahora desarrollamos las dos interfaces de control. Cada interfaz contendrá la misma política protegiendo a AccessManager, pero diferenciándose en la política que usan para proteger el objeto Hello.

```
(InterfaceControl Control1
     (("IDL:adiron.com/AccessControl/AccessManager:1.0"
     SAM_SamePrinOnly)
     ("IDL:test/Hello:1.0" BartNoHi)))
     (InterfaceControl Control2
          (("IDL:adiron.com/AccessControl/AccessManager:1.0"
          SAM_SamePrinOnly)
          ("IDL:test/Hello:1.0" BartHiNoMyRealm)))
```

El fichero *"h5.config"* contiene lo mismo que el fichero *"h4.config"*, con una línea añadida, la que contiene la propiedad "control.access.AccessManager". Este propiedad indica al servidor que exponga el IOR de su AccessManager directamente escribiéndola en un fichero llamado "ServerAccessManager.ior". Este IOR se usará en las llamadas a ControlAdmin.

```
control.access.AccessManager=FILE:ServerAccessManager.ior
```

Las llamadas a ControlAdmin serán realizadas por el script *"run5"* y básicamente tienen la forma:

```
java \
control.tools.ControlAdmin \
--access-manager FILE:ServerAccessManager.ior \
--ssl-pwsupplied demo.keystore IAIKKeyStore "blahblah" \
marge "mypassword" \
--import-dir ./ \
--description FILE:Server5.2.sal \
--update-reduced
```

El argumento al que apunta la opción *"—access-manager"* contiene el nombre del fichero que contiene el IOR al AccessManager del objeto servidor. El argumento *"—description"* especifica el fichero de la descripción de la nueva política de acceso. El argumento *"—import-dir"* contiene el nombre del directorio actual , porque el fichero de descripción contiene importaciones de declaraciones que se encuentran en el directorio actual. La opción *"—update-reduced"* implica que la herramienta Control-Admin incluirá la descripción de accesos y llamará al objeto AccessManager del servidor para actualizar su política con la nueva descripción.

El argumento "--ssl-pwsupplied" tiene el mismo significado que en los ejemplos previos.

Run5

El script "run5" realizará las siguientes tareas:

1. Reducirá el fichero "Server5.1.sal" en el fichero "Server5.1.rsal".
2. Arrancar el servidor Hello.
3. Ejecutar el cliente, el cual tiene garantizado el acceso a la operación "hi" y denegado para la operación "hello".
4. Usar la herramienta ControlAdmin para recuperar la política actual para el servidor Hello. Sin embargo, usa un actor distinto al usado como servidor. Este comando fallará y los accesos a las operaciones del AccessManager del servidor serán denegadas.
5. Usa la herramienta ControlAdmin para recuperar la actual política para Hello. Usa el mismo actor que actúa como servidor. Este comando será aceptado, y los accesos a las operaciones del AccessManager del servidor serán aceptadas.
6. Usa la herramienta ControlAdmin para actualizar la política con la definida en el fichero "Server5.1.sal". No es necesario reducir este fichero ya que ControlAdmin lo realiza de forma automática. Este comando es ejecutado como el mismo actor que el servidor y tendrá éxito.
7. Ejecutar el cliente de nuevo, al cual tendrá se le denegará el acceso a la operación "hi" pero se le concederá acceso a la operación "hello".

Otras formas de usar CONTROL

CONTROL puede utilizarse de otras maneras y junto a:

- Kerberos.
- SSL.
- Versiones antiguas de IIOP.
- Servicios de nombres para IOR de AccessManager
- Propietarios y dominios.
- Privilegios requeridos y concedidos.

Programas de seguridad (usando interfaces CORBA de seguridad tales como ORBAsec SL2).

EL PUZZLE

ORBASEC

En los siguientes a apartados veremos brevemente cuatro elementos de Orbasec que son piezas clave en la realización de soluciones corba segura.

El nivel de detalle de la exposición no quiere suplir al manual del producto, pretende ser una descripción que facilite entender como llevar a cabo un programa corba con orbasec.

Inicialización de SL2

La forma de inicializar el ORB de Orbasec es la forma standard del método estático:

```
org.omg.CORBA.ORB.init
```

de esta forma se puede inicializar el ORB completo el ORB para applets Java y el adaptador básico de objetos, BOA (basic object adapter) en la forma estándar.

La forma de configurar ORBasec SL2, es mediante argumentos y propiedades que podemos consultar en el manual a partir de la página 58 a 72.

Las propiedades establecidas durante la inicialización pueden ser asignadas mediante:

- Fichero de configuración
- Java.util.Properties
- Definiciones de las propiedades del sistema en programa o en la línea de comandos parámetro –D
- Opciones expresadas en la línea de comando de invocación de programa java.

Debemos cuidar que orden entre los archivos SL2.jar y OB.jar en la variable CLASSPATH, debe ser el indicado anteriormente.

Veamos un ejemplo de como se lleva a cabo

```
// Java
java.util.Properties props = new java.util.Properties();
props.put( “org.omg.CORBA.ORBClass”, “orbasec.ORB” );
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( args, props );
org.omg.CORBA.BOA boa = orb.BOA_init( args, props );
```

Si es utilizado el JDK o otra clase de otro proveedor para org.omg.CORBA.ORB deberemos asignar la propiedad org.omg.CORBA.ORBClass a orbasec.orb, mirar el ejemplo anterior.

Entre otras propiedades podemos determinar:

- El tipo de conexiones aceptadas.
- Puertos y direcciones de los protocolos ssliop y iiop.
- Activar o desactivar el uso de SSL anónimo.
- Abrir una sesión kerberos automática.

Servicios seguros

Durante la inicialización, ORBasec SL2 añade cierto número de objetos iniciales responsables de los servicios de seguridad.

Estos son obtenidos como referencias iniciales fuera del ORB ORBasec.

Forma de obtenerlos

```
import org.omg.CORBA.*;
import java.util.Properties;
public void main(String[] args)
{
Properties props = new Properties();
ORB orb =
ORB.init( args, props) );
BOA boa = orb.BOA_init( args, props );
// authenticate
...
// create references to secure ORB services
((orbasec.ORB)orb).add_initial_services();
...
}
```

Objetos

SECURITY MANAGER

Se introduce la interfaz SecurityLevel2::SecurityManager, según la especificación 1.7. de CORBA seguro.

Este objeto local mantiene la información de estado acerca de la única instancia ORB que se ejecuta en la JVM.

Obtener su referencia
Se obtiene a partir de una referencia inicial como aparece en el siguiente código:

```
// Java
org.omg.CORBA.ORB orb = // The SL2 initialized ORB;
org.omg.SecurityLevel2.SecurityManager security_manager =
org.omg.SecurityLevel2.SecurityManagerHelper.narrow(
orb.resolve_initial_references("SecurityManager")
);
```

Primordialmente usaremos este objeto para obtener acceso al PrincipalAuthenticator e información sobre los mecanismos de seguridad y credenciales.

Más detalles en las páginas 73 a 86 del manual

POLICYCURRENT

Objeto local mantiene la información sobre las políticas en uso en hilo o contexto actual de ejecución.

Obtener su referencia
Se obtiene a partir de una referencia inicial como aparece en el siguiente código:

```
// Java
org.omg.CORBA.ORB orb = // The SL2 initialized ORB;
org.omg.SecLev2.PolicyCurrent policy_current =
     orbasec.SecLev2.PolicyCurrentHelper.narrow(
               orb.resolve_initial_references("PolicyCurrent")
     );
```

Podemos ver la funcionalidad de este objeto en la página 87 del manual.

Gestor principal autenticación

Es el responsable de las operaciones de autenticación y establecimiento de los mecanismos de seguridad que se vayan a usar.

Este objeto **SecurityLevel2::SecurityManager,** del que existe una única instancia en la JVM en ejecución, es la implementación singular de la interfaz del mismo nombre.

Una programador usa este objeto para inicializar las credenciales propias de la aplicación, adquiriendo el objeto credenciales correspondiente. El **PrincipalAuthenticator** entonces sitúa esta credenciales en el objeto

SecurityManager y en la lista de credenciales propias del objeto **SecurityLevel2::Current**.

La operación esencial ofrecida por este objeto es **authenticate**, que comienza la secuencia de autenticación que puede implicar varios pasos.

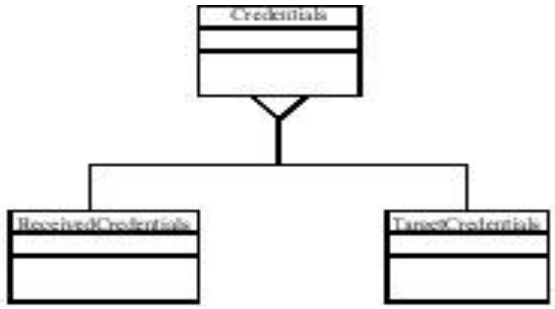
Obtener su referencia
Se obtiene a partir de una referencia inicial como aparece en el siguiente código:

```
// Java
org.omg.SecurityLevel2.SecurityManager mgr =
// .... get the security manager object
org.omg.SecurityLevel2.PrincipalAuthenticator pa =
mgr.principal_authenticator();
```

Para más detalles sobre este objeto consultar las páginas de la 91 a 123.
Para más detalles sobre el objeto **SecurityManager** consultar las páginas 74.

Credenciales

Las credenciales son objetos para consultar los atributos de seguridad pertenecientes a la aplicación y a los clientes que realizan las peticiones. También pueden ser examinadas las credenciales del servidor. Así se habla de credenciales “propias”, “recibidas” y “objetivo”.

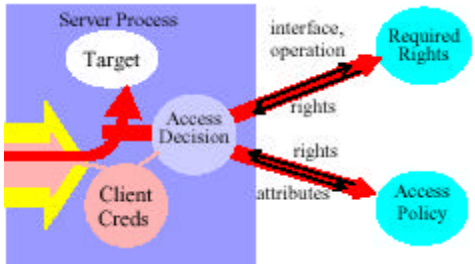


Las credenciales propias son obtenidas a partir del **PrincipalAuthenticator.**
Las recibidas son validas en el contexto de ejecución concreto de servicio de una petición. Representan la asignación de un contexto de seguridad entre client y el objetivo. Esta credenciales son consultadas para obtener la identidad y calcular los privilegios del solicitante.

Las credenciales objetivo, son las credenciales de un objeto inmersas en la referencia al objeto, IOR. Posiblemente se necesite comprobar los derechos de un objeto antes de ejecutar peticiones sobre este.

Control y Sal. Construcciones

Todas las expresiones en SAL son representadas por estructuras entre paréntesis o por nombres. En el nivel superior, SAL contiene declaración de construcciones.




```
(type-tag name typed-structure)
```

La declaración de construcciones son tuplas que asocian un nombre con una estructura entre paréntesis de tal forma que una sustitución directa de la estructura sea posible.

Por ejemplo, una expresión *attribute-family* tiene la siguiente estructura:

```
( number number)
```

Una declaración para un determinado *attribute-family* tiene la siguiente forma:

```
(AttributeFamily Corba1 (0 1))
```

Utilizando las características de sustitución del lenguaje, si una expresión *attribute-type* tiene la forma:

```
( attribute-family number )
```

Entonces las expresiones (Corba1 2) y ((0 1) 2) denotan la misma construcción attribute-type. La última expresión se dice que ha sido expandida. Cuando una expresión ha sido expandida completamente, se dice que se encuentra en *forma normal*.

Tipos de Atributos de Seguridad

En CORBA, un atributo de seguridad pertenece a una familia (la cual se define por una estructura de dos números) y tiene un tipo (el cual se define por una familia y un tipo número). Ejemplos de algunas familias, atributos de seguridad y tipos son los siguientes:

```
(AttributeFamily Corba1 (0 1))
(AttributeType AccessId (Corba1 2))
(AttributeType PrimaryGroupId (Corba1 3))
```

CredentialsPred

Una CredentialsPred es una estructura que especifica una consulta booleana de una interfaz. La evaluación implica la consulta de las Credenciales para los atributos de tipos especificados y su equivalente valor correspondiente al valor del atributo en la estructura CredentialsPred. Por ejemplo, es común la consulta al sistema del cliente que

tiene un determinado valor del atributo “AccessId” y compararlo con un valor conocido. La siguiente declaración CredentialsPred ilustra esta especificación:

```
(CredentialsPred isBart (AccessId “bart@simpson”))
```

Esta declaración define un predicado de credenciales, el cual es verdadero si el objeto Credencial tiene un atributo del tipo “AccessId” y el valor del atributo es igual a “bart@simpson”.

La construcción de CredentialsPred pueden realizarse por medio de operadores conjuntivos y disyuntivos, **and** y **or**, tal como en el siguiente ejemplo:

```
 (CredentialsPred isBartOrHomer (or (AccessId
“bart@simpson”) (AccessId “homer@simpson)))
```

El operador unario **not** es indefinido para esta construcción. Tratando con negaciones nos encontramos con una dificultad, ya que la negación de una construcción *(attribute-type strin*g) podría significar la negación de su conjunción. El resultado sería: el objeto credencial no tiene un atributo “AccessId” o el valor de su atributo no es igual a “bart@simpson”. Esta sentencia nos lleva a problemas cuando tratamos con Credenciales. Pero es posible que la Credencial contenga más de un atributo “GroupId”, para aliviar esta aparente carencia de expresividad, las negaciones de las decisiones de acceso son delegadas en los *controles*.

Controles

Los Controles son construcciones que son evaluadas contra las Credenciales, nombres de interfaz y nombres de operaciones y generan las decisiones de acceso: permitir (“*allow*”) o denegar (“*disallow*”). En casos especiales, pueden generar el valor no aplicable (“*DoesNotApply”*), el cual puede generarse en situaciones en las que el control para una determinada interfaz u operación no está definida.

En SAL hay tres tipos de controles: CredentialsControl, OperationControl e InterfaceControl.

CredentialsControl

Una estructura CredentialsControl es una construcción que es evaluada contra las credenciales del cliente y genera una decisión de acceso. Una estructura CredentialsControl contiene una secuencia de pares entre paréntesis, cada par consiste en una estructura CredentialsPred y un valor de control **Allow** o **Disallow**. El siguiente ejemplo ilustra una declaración CredentialsControl:

```
(CredentialsControl OnlyBart
((isBart Allow)
(true Disallow)))
```

La evaluación de una estructura CredentialsControl se procesa evaluando cada estructura CredentialsPred contra el mismo objeto Credenciales sucesivamente hasta que

el valor “true” es devuelto, similar a una estructura if-then-else. Así que la siguiente construcción CredentialsControl:

“((isBart Allow) (true Disallow))”

genera “Allow” si la estructura CredentialsPred asociada con “isBart” se evalúa como “true”.

Por el contrario el control generará “Disallow” porque el predicado de la última claúsula es “true”.

En las siguientes secciones continuaremos refiriéndonos a las interfaces y operaciones definidas por las siguiente definiciones IDL CORBA:

```
module test {
     interface Hello {
          void hi();
          void hello();
     };
     interface Goodbye {
          void goodbye();
     };
};
```

OperationControl

Una estructura OperationContrrol es una construcción que es evaluada contra el nombre de la operación y las Credenciales del cliente. Esta evaluación genera una decisión de acceso. Una estructura OperationControl contiene una secuencia de pares formados por el nombre de una operación y una estructura CredentialsControl. El siguiente ejemplo muestra una declaración de estructura OperationControl:

```
(OperationControl HelloBartOnly “IDL:/test/Hello:1.0”
((“hi” OnlyBart)
(“hello” ((true Allow)))))
```

Esta declaración difiere de la previa en que contiene un argumento extra, “IDL:/test/Hello:1.0”. Este identificador es el nombre de la interfaz CORBA que se usa para registrar la interfaz “Hello” con el repositorio de interfaces CORBA, la entidad en CORBA que alberga la definición de interfaz. Este identificador se usa al nivel declaración para avisar en el chequeo de tipos, es decir, asegurar que las operaciones llamadas en el actual control están contenidas en la interfaz. Una restricción contextual a la sintaxis del OperationControl es que no está permitido que aparezca el nombre de una operación dos veces.

La evaluación de una OperationControl procede comparando el nombre de la operación de la solicitud CORBA con los nombres de las operaciones listadas en el control OperationControl. Si hay una coincidencia, entonces la estructura CredentialsControl se evalúa contra la Credencial de la solicitud. Si no hay coincidencia la evaluación genera "DoesNotApply".

InterfaceControl

Una estructura InterfaceControl es una construcción que se evalúa contra el nombre de la interfaz, el nombre de la operación, y las credenciales del cliente. Esta evaluación genera una decisión de acceso. Una estructura InterfaceControl contiene una secuencia de pares formados por nombres de interfaz y estructuras OperationControl, como muestra el siguiente ejemplo:

```
(InterfaceControl HelloGoodbye
(("IDL:/test/Hello:1.0" HelloBartOnly)
("IDL:/test/Goodbye:1.0" (("goodbye" ((true Allow)))))))
```

La evaluación de una estructura InterfaceControl selecciona la estructura OperationContro asociada por el nombre de interfaz, entonces evalúa la estructura OperationControl contra el nombre de la operación y las Credenciales. Si no hay coincidencia con el nombre de interfaz, la evaluación genera "DoesNotApply".

Una restricción contextual sobre una estructura InterfaceControl es que no está permitido que el nombre de una interfaz aparezca dos veces.

Construcción AccessDecision

Las construcciones definidas anteriormente pueden ser usadas para crear librerías de declaraciones. Estas declaraciones, por otro lado, pueden ser usadas para componer una política de decisiones de acceso que será aplicada a un conjunto de objetos. La siguiente construcciones define la decisión de acceso en función de un servidor:

```
(AccessDecision (InterfaceControl HelloGoodbye) Disallow)
```

Sólo puede haber una construcción AccessDecision en una descripción SAL.

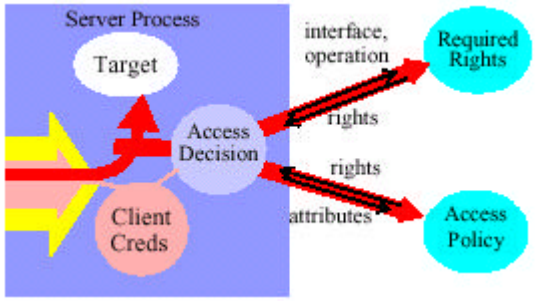
La evaluación de una construcción AccessDecision consiste en una evaluación del InterfaceControl contra las Credenciales del cliente, el nombre de la interfaz y el nombre de la operación. Si el resultado de la evaluación es "DoesNotApply", la decisión de acceso por defecto es el valor especificado como tercer argumento.

Extendiendo el Modelo

La especificación de Seguridad CORBA sugiere además un diferente modelo/implementación para la función de decisión de acceso basada en *derechos establecidos* (required rights). Este modelo trabaja con las bases de que interfaces y operaciones se agrupan juntas en un espacio plano denominado *derechos* (rights). Por lo tanto, puede haber un grupo como conjunto de operaciones en diferentes interfaces dentro de los mismos derechos, tal como "get" y "set", y entonces definir la lógica de control de acceso en base a esos derechos. Esos derechos se dice que han sido

establecidos (*required*). Este mapeo está definido por la Seguridad CORBA por el objeto "RequiredRights".

A la hora de definir políticas de acceso basadas en derechos, se hace necesario la existencia de un mapeo de actores a derechos. Este mapeo se define por la Administración de Seguridad CORBA con el objeto "AccessPolicy".



El objeto AccessDecision posee una operación booleana, denominada "access_allowed". La evaluación de esta operación se define como la consulta de RequiredRights para los derechos establecidos para el nombre de una interfaz particular del objeto en concreto y el nombre de la operación de la solicitud, y la consulta del objeto AccessPolicy para los derechos concedidos al cliente, como podemos ver en la figura.

Si se encuentra la especificación de derechos establecidos, entonces se dice que el acceso ha sido concedido para la solicitud y la función de decisión de acceso devuelve "true", en caso contrario la función devuelve "false".

Extendiendo el Lenguaje

Extender el lenguaje para el soporte RR/AP (RequiredRights/AccessPolicy) reuqiere añadir nuevas construcciones para manejar derechos. Estas extensiones del lenguaje se denominan RRAPSAL.

Rights

Una de las construcciones básicas en RRAPSAL es la estructura de datos Right. En Seguridad CORBA, los derechos se definen en familias, de forma parecida a como se definen tipos de atributos. Un derecho tiene una familia, la cual se forma por medio de dos enteros no negativos y un valor formado por una cadena. Esta definición se ilustra por medio del siguiente ejemplo de algunos de los derechos estándar de CORBA:

```
(RightFamily Corba0 (0 0))
(Right Get (Corba0 "get"))
(Right Set (Corba0 "set"))
(Right Manage (Corba0 "manage"))
```

CredentialsRights

El componente AccessPolicy de este modelo RR/AP mapea los atributos de las Credenciales a una lista de derechos. Este mapeo se especifica por cláusulas conteniendo

una estructura CredentialsPred y una lista de derechos. El siguiente ejemplo ilustra la estructura CredentialsRights:

```
(CredentialsRights MyAccessPolicy
     ((isBart (Get Set))
     (isAdmin (Set Manage))))
```

La semántica de evaluación es un poco diferente ya que cada una de las cláusulas y todas deben ser evaluadas contra las Credenciales. La unión de los derechos resultantes constituyen la respuesta. Si ninguna estructura CredentialsPred se evalúa como verdadera, se devuelve una lista de derechos vacía.

OperationRights

La estructura OperationRights es una construcción que especifica un mapeo de nombres de operaciones a listas de derechos. Como en el siguiente ejemplo:

```
(OperationRights HelloRights "IDL:/test/Hello:1.0"
     (("hi" none)
     ("hello" (Manage Set)))
```

Como el OperationControl, esta declaración contiene un argumento extra, "IDL:/test/Hello:1.0" para el mismo propósito de chequeo de tipos. Como antes, una restricción contextual de la estructura OperationRights es que no está permitido que ningún nombre de operación aparezca dos veces.

La evaluación de una estructura OperationRights contra nombres de operaciones produce una lista de derechos. Si no existe un mapeo para una operación particular existente, se devuelve "NoMapping".

InterfaceRights

La estructura InterfaceRights es una construcción que realiza el mapeo de interfaces y operaciones a una lista de derechos. Este mapeo se realiza especificando una estructura OperationRights para cada nombre de interfaz. Un ejemplo de declaración es la siguiente:

```
(InterfaceRights MyRequiredRights
     (("IDL:/test/Hello:1.0" HelloRights)
     ("IDL:/test/Goodbye:1.0" (("goodbye" Get)))))
```

Una restricción contextual de la estructura InterfaceRights es que no está permitido que ningún nombre de interfaz aparezca dos veces.

La evaluación de una estructura InterfaceRights contra un nombres de interfaz y un nombre de operación produce una lista de derechos. Si no existe un mapeo para una interfaz u operación particular existente, se devuelve "NoMapping".

Combinando InterfaceRights y CredentialsRights

Cuando usamos el modelo , la evaluación de la estructura CredentialsRights debe ser combinada con la evaluación de la estructura InterfaceRigths. La evaluación de cada estructura se realiza por medio de composición funcional. Aplicamos el evaluados de la estructura InterfaceRights contra los nombres de interfaces y operaciones. Esta aplicación producirá los derechos establecidos. Entonces aplicamos el subconjunto de funciones a la lista de derechos establecidos contra la lista de derechos producidos aplicando el evaluador de la estructura CredentialsRights contra las Credenciales.

La estructura AccessDecision se extiende para este propósito, añadiendo un nuevo tipo de construcción de control, InterfaceRightsControl, como es la siguiente:

```
(AccessDecision
     (InterfaceRightsControl MyRequiredRights
                             MyAccessPolicy )
     Disallow)
```

La construcción InterfaceRightsControl especifica que las dos construcciones, InterfaceRights y CredentialsRights, deben ser evaluadas como hemos comentado anteriormente. Si la evaluación de la estructura InterfaceRights devuelve “NoMapping”, entonces la construcción completa devuelve “DoesNotApply” y se devuelve el valor por defecto.

Análisis de Evaluaciones

El separar la descripción de control de acceso en mapeos de credenciales a derechos y nombres de interfaces y operaciones a derechos, puede generar una gestión de control de accesos más fácil; pero la directa evaluación por separado de los objetos RequiredRights y AccessPolicy en CORBA es ineficiente.

Cuando una solicitud alcanza llega a un servidor, la nterfaz y la operación ya son conocidas, y por tanto, los derechos establecidos son fácilmente deducibles. Es mejor preguntar si un derecho particular está concedido, en vez de preguntar por todos los derechos concedidos a Credenciales y entonces realizar un cálculo de subconjuntos. Desafortunadamente, esta implementación está imposibilitada debido a la utilización de un objeto AccessPolicy, el cual sólo pregunta por todos los derechos concedidos a las Credenciales.

Para hacer la función AccessDecision más eficiente, la construcción InterfaceRightsControl se reduce a una estructura InterfaceControl. El usar una interfaz InterfaceControl sobre una construcción InterfaceRightsControl hace la evaluación de la decisión de acceso más eficiente, ya que no hay que evaluar todas las estructuras CredentialsPred que aparecen en la estructura CredentialsRights. La función AccessDecision sólo necesita evaluar las estructuras CredentialsPred que están asociadas con derechos particulares Además, obteniendo antes la descripción de control de acceso completa, reduciéndola, y evaluando la decisión de control de acceso de forma local al servidor es más eficiente que hacer llamadas en dos objetos remotos y realizar una combinación de sus resultados.

LA APLICACIÓN

A partir de los ejemplos que vienen de Orbasec y los de [10], hemos creado nuestro propio ejemplo para comprobar por nosotros mismos las características de la solución.

El dominio escogido ha sido la Banca.

La prueba consiste en una pequeña aplicación en la que un usuario a través de un Applet empotrado en una página web, accede a los saldos de quien se solicita. El programa crea nuevas cuentas con saldos aleatorios cuando se solicita el saldo de alguien inexistente en la relación de cuentas. Para más detalles consultar las páginas 118 a 123 de [10].

El proceso seguido ha sido el indicado aquí:

- Definición de IDL
- Códificación de las clases de implementación, incorporando la inicialización ORBasec.
- Y ya está, a jugar…

El código se incluye en los anexos.

ALTERNATIVAS

Si revisamos la bibliografía el trabajo de fin de carrera de Diego Sevilla nos permite comparar distintas tecnologías disponibles para la realización de aplicaciones distribuidas, y podemos llegar a la conclusión de que esta, CORBA seguro es la mejor solución.

No obstante, **RMI** en entornos de aplicaciones de java puro, donde es usado un modelo de objetos java exclusivamente, es la alternativa a considerar[11], aunque deberemos crear un marco de desarrollo que cubra:

- La seguridad de la distribución de las aplicaciones, mediante el uso de RMISecurityManager o subclase de esta, instanciada vía el método setSecurityManager de la clase System.
- La seguridad de transporte, creando una RMISocketFactory a la medida, por ejemplo basadas en las librerias IAK.
- La autenticación y el control de acceso, pues RMI no da soporte alguno a estos conceptos, ni siquiera como interfaz, debemos construir la solución en base las IAK.

Como podemos imaginar, esto implica un mayor esfuerzo de desarrollo.

Debemos tener en cuenta, los problemas de integración de los productos ante una gran distribución de una aplicación. Esto puede forzarnos a emplear RMI como la única posibilidad.

En el momento de redactar esta documentación, ya esta disponible la solución de object oriented concepts para CORBA con conexiones seguras sobre FreeSSL, no conforma la especificación de CORBA Seguro.

Debemos ser conscientes de la importancia de elegir este tipo de soluciones pues de esta forma esta abierta a ser fácilmente incorporada a nuevas tecnologías como las de componentes corba, donde estos servicios son asumidos por la especificación de componentes corba, actualmente en revisión.

Habitualmente, las especificaciones corba, son pronto bastante estables y también es tenida en cuenta la sustitución tecnológica, por lo que una inversión en este tipo de solución tiene muchas posibilidades de ser dotada de una continuidad y una mantenibilidad altas.

CORBA SLL Y ...

Cortafuegos

Las políticas de filtrado de paquetes, restricción de puertos y protocolos aplicadas en los enrutadores, apoderados de red y cortafuegos, pueden conducir al mal funcionamiento de las aplicaciones CORBA SLL y CORBA en general.

Todas las comunicaciones CORBA se realizan sobre paquetes TCP. Con la posibilidad de escoger y fijar los puertos para los puntos de acceso.

Pero como es habitual, que las organizaciones limitan el intercambio con la Red a paquetes TCP para HTTP, incluso HTTP es filtrado para evitar el acceso a contenidos o la descarga de caballos de Troya. La mayoría de los productos para ORB, contemplan esta posibilidad.

HTTP

El encapsulamiento de las peticiones CORBA en http, se realiza mediante programas pasarela interpuestos que escuchan peticiones vía http y las traduce a llamadas CORBA habituales y viceversa. Estos programas son en si proxys orb enchufados en puerto para http. En nuestro caso, el ORB subyacente ORBACUS, permite alcanzar esta funcionalidad.

Pero no hemos sabido averiguar como es posible hacer funcionar CONTROL ni ORBASEC sobre túneles http.

Proxys

La necesidad de proteger las redes de las organizaciones frente a la Red, impulsa la instalación de servicios de apoderados o representantes, que protegen y aíslan de tráfico de aplicaciones mal intencionado.

Los ORB constituyen en si mismos apoderados. Además podemos ver que las especificaciones del OMG, recogen los servicios de representación o apoderado, proxy,

para servicios corba. Algunos de los productos comerciales como el que soporta nuestra propuesta, ORBACUS disponen de servicios complementarios que cubren la representación.

De hecho debemos tener en cuenta que los ORB, permiten el acceso a los servicios sin tener en cuenta el contexto real de ejecución donde se realiza el computo, la localización del proceso servidor, servant. Todo ello, evita que la implementación se vea afectada; el desarrollo del código de la aplicación no tiene por qué considerar este aspecto.

La situación del servicio ORB en una DMZ, con control sobre los flujos de información requiere de un esfuerzo de administración pero no de programación.

Interoperatividad

Continuando con el tema de proxys, un orb actuando como apoderado, soluciona la interoperatividad entre ORB que no soportan SLL y los que si, de forma que se pueda evitar el paso por la Red o la Intranet de peticiones desprotegidas. Para ello, dispondremos un apoderado creado sobre el ORB seguro conectado al ORB no seguro que media con otra plataforma(lenguaje de programación), situados ambos en la misma máquina del cliente. La conexión interna dentro de la propia máquina puede también ser protegida mediante IPSec o SSH. El orb apoderado seguro, redirige las peticiones a objetos remotos sobre un ORB seguro distante en la Red. Las repuestas siguen el camino inverso.

XLM y SOAP

SOAP

Actualmente SOAP esta en el candelero del debate Corba, así comentaremos brevemente respecto al papel de SOAP en CORBA SLL.

SOAP, es otro protocolo para la gestión dinámica peticiones RPC para Microsoft DCOM, soportado también sobre flujos XML. Recientemente ha sido adoptada la correspondencia con CORBA, haciendo posible la interoperatividad de peticiones.

Desde el punto de servidor seguro CORBA ORBasec y CONTROL no hay inconveniente alguno para usar SOAP mediante pasarela, simplemente es un nivel previo que debe interceptar las peticiones SOAP, que seguidamente son encaminadas al servicio seguro.

Pero en nuestra modesta opinión, SOAP (sopa, enredo) solo contribuye a enredar y complicar el uso y la programación de aplicaciones distribuidas.

En cuanto a XML, no creemos que deba usarse para todo, como tampoco se debe usar HTTP para crear un diccionario.

Por otra parte, dada la proliferación de herramientas libres al alcance para XML, si creemos que las descripciones SAL, podrían ganar en claridad y facilidad de uso utilizando una representación XML, editores XML y finalmente compiladores XML, que

convirtieran las descripciones de la políticas de seguridad y privilegios a formatos más acordes para su aplicación.

Proteger el Servicio

Cuando utilizamos CONTROL, o ORBASEC, debemos tener en cuenta que un cliente podría intentar acceder al administrador de autenticación para obtener la lista de credenciales activa. Debemos ser muy cuidadosos con este extremo, poniendo los medios necesarios para evitar esta situación.

Utilizar políticas de acceso restrictivas por defecto, acceso a nada, las políticas deben ser explícitas indicando servicio, privilegio e identidad (credencial).

Incluir claramente la política que evita acceder a la lista de credenciales activas tanto en lectura como actualización, el atributo lista **own_crentials** del SecurityManager. Es responsabilidad de las aplicaciones eliminar las credenciales caducas y fuera de uso.

Corba y IPSec

Para algunos sistemas ya implantados la solución más rápida para adoptar un servicio seguro sobre CORBA es implantar comunicaciones IPSEC y VPN, especialmente en la comunicación entre ORB, usando el IIOP, para sistemas de trading u orb distribuidos sobre la Red ya en marcha. Esta solución si necesita de unas decisiones de acceso, obliga a permear la aplicación con detalles de los servicios IPSec.

IPSEC puede ser usado de forma complementaria para evitar el spoofing y hajacking a nivel de los datos de transporte. En algunos sistemas puede y quizás deba usarse, por ejemplo en una jerarquía de servicios de localización de objetos.

Otro propósito fundamental donde IPSEC o SHH, puede ser aplicado es el diseño de conexiones seguras entre ORB sin soporte para SSL y ORB con soporte SLL, dentro de una misma máquina o entre sitios remotos.

Una situación práctica que surge cuando no tenemos disponible un ORB para el lenguaje que deseamos utilizar, en [11] podemos ver el ORB Fnorb para Python, pero este no da soporte para SSL, por lo cual deberemos instalar en el cliente nuestro orb seguro, realizando las veces de apoderado conectado con IIOP al ORB Fnorb, en tubería segura IPSEC o SSH, si es preciso.

IPV6

En IP versión 6, la autenticación, y la privacidad son parte integrante del modelo de ejecución, por lo que las aplicaciones se pueden beneficiar de los servicios aportados al

respecto para los niveles superiores al transporte. Esto simplificará a buen seguro la capa intermedia del código de la aplicación y la Red, contribuyendo a una mayor eficacia.

¿Esto puede ser un problema para las aplicaciones CORBA SLL en funcionamiento? Opinamos que no pues el estándar incorpora la sustitución de la tecnología para la seguridad mediante la especificación de seguridad suplida, Security Replacebilty, de forma que se hace posible sustituir unas implementaciones por otras. Minimizando el impacto del cambio en las aplicaciones.

Conclusiones

Si escogemos CORBA y necesitamos proteger y autenticar los servicios, deberemos optar por escoger una solución que siga los estándares establecidos por el OMG.

Mantenibilidad y continuidad

Habitualmente, las especificaciones corba, son pronto bastante estables y también es tenida en cuenta la sustitución tecnológica, por lo que una inversión en este tipo de solución tiene muchas posibilidades de ser dotada de una continuidad y una mantenibilidad altas.

Igual a Igual

Un aspecto que debemos destacar es la posibilidad que brinda la tecnología CORBA para que las aplicaciones sean de igual a igual, no solo cliente a servidor. En este contexto es si cabe aún más importante la capacidad de autenticación y privacidad que CORBA seguro ofrece.

En cuanto a los productos aquí revisados

SAL y CONTROL

SAL es un lenguaje para el control del modelo de credenciales de seguridad en CORBA, basado en interfaces y operaciones. El lenguaje expresa políticas de control de acceso no triviales por definición de predicados sobre las credenciales de un cliente. Las negaciones puede ser expresadas usando el control **Disallow**. Una descripción escrita en forma normal SAL junto con el evaluador “eval_ic” sirve como implementación del objeto AccessDecision en CORBA seguro, el cual es la entidad que controla las decisiones de acceso de las solicitudes en el lado servidor.

La potencia descriptiva del modelo de políticas de Accesos/Derechos establecidos de CORBA seguro está soportado por una extensión de SAL, llamada RRAPSAL. El modelo RR/AP se beneficia del hecho de generar una descripción más manejable; sin embargo, carece de cierta expresividad que la semántica de RR/AP no permite por la introducción de controles negativos.

La reducción de RRAPSAL en forma normal SAL resulta en una más eficiente evaluación que la evaluación directa de RRAPSAL. Obviamente es más rápida que la evaluación del modelo de objetos CORBA RR/AP, los cuáles deben realizar llamadas remotas a los objetos RequiredRights y AccessPolicy. La inicialización de los servidores con completas descripciones para RR/AP en RRAPSAL compiladas para la forma normal de SAL hace la evaluación de decisión de acceso más eficiente.

Facilidad de uso

La dificultad de SAL, estriba en su notación que en nuestra opinión dificulta su uso. La sintaxis de SAL no es fácil de entender, ni usar.

Transparencia

La transparencia es la cualidad más destacable de CONTROL y SAL, las aplicaciones no deben preocuparse de incluir código alguno respecto a la seguridad, lo cual puede ser en si una ventaja, al aislar al programador del conocimiento de las políticas de seguridad y privilegios.

ORBASEC

Orbasec aunque no satisface todos los requisitos expresados en la norma del OMG para servicios CORBA seguros, alcanza un nivel de funcionalidad practico más que suficiente para cualquier aplicación.

Usaremos ORBAsec cuando no haya necesidad de establecer políticas de acceso con distintos niveles de privilegio, como servicios públicos sobre conexiónes SSL anónimas.

Facilidad de uso

Orbasec es Java, los módulos de carga iniciales donde se inician el ORB y las conexiones se amplían, pero esto puede automatizarse mediante patrones de código o soluciones ad hoc.

Transparencia

Orbasec no es transparente a las aplicaciones, pero el código introducido puede delimitarse perfectamente, evitando el enmarañamiento del código original no seguro.

Sal y orbasec se complementan

Orbasec puede ser utilizado como complemento de SAL y Control. Juntos permiten cubrir situaciones como las que surgen ante la creación de auditorías, trazas acceso a servicios, bitacora de accesos, delegación entre servicios, multidifusión…

Los tiempos de respuesta

El trabajo de Diego Sevilla [11] nos permite comparar los tiempos de respuesta de CORBA frente a otras, pero evidentemente la conexión SLL, supone una sobrecarga.

La operación más pesada, es la negociación del secreto compartido para establecer la entidad de seguridad en el establecimiento de la conexión SLL.

En las prueba realizadas se aprecia en el primer acceso un tiempo de respuesta muy superior a otras sucesivas, retraso por el establecimiento de la conexión SLL, pero debemos además tener en cuenta que las librerías usadas para generar números aleatorios y la plataforma de pruebas, no deberían usarse en entornos de producción.

El cifrado también supone una sobrecarga, pero si consideramos que la mayoría de los mensajes correspondientes a solicitudes de invocación sobre objetos son cortos, que posiblemente la operación marshaling puede ser aprovechada para el cifrado y descifrado y la latencia de comunicaciones en la Red, la sobrecarga añadida debe ser mínima frente el tiempo total de respuesta.

Estas sobrecargas deben ser tenido en cuenta en el diseño de las políticas de servicio del ORB[9 pág. 151-155].

En la actualidad, podemos encontrar el mercado aceleradores para servicios internet que incluyen el establecimiento de conexiones SSL muy usado en comercio electrónico y banca. Estos aceleradores no son más que ordenadores con algo de circuitería especial adecuada a tal fin. Así, que no es descartable que pronto podamos encontrar este tipo de soluciones adaptadas para los servicios CORBA y en particular para CORBA seguro.

Aceptación del mercado español, Murcia en particular.

En nuestra opinión, en el ámbito de dirección sistemas de información y de la administración de sistemas, las soluciones corba no tienen una aceptación comparable a otros tipo de soluciones para aplicaciones distribuidas, basadas en la tecnología DCOM o Java Servlet. Realmente, ni siquiera las aplicaciones distribuidas son consideradas o se confunde el término con aplicaciones cliente/servidor de tres o cuatro capas centradas en servicios web. Se construyen las aplicaciones centradas en servicios web basadas en DCOM o Java Servlet como fórmula para la distribución de aplicaciones por la Red e intranets.

Algunos productos de gran aceptación en el mercado de las grandes cuentas como los de Oracle, incluyen empotrados en sus marcos de desarrollo la tecnología CORBA, pero eluden su mención en los titulares. Oracle Developer 6 y Application Server, se apoyan en CORBA pero la gran mayoría de sus usuarios no son conscientes de ello. Ni de las ventajas que pueden obtener

Al menos, tenemos la satisfacción de que desde nuestra Facultad esta siendo impulsado el conocimiento de las tecnologías CORBA, con lo que es de esperar que en ocho a diez años, cuando los actuales nuevos ingenieros superiores, pasen a ocupar puestos de responsabilidad y además las ahora tecnologías emergentes CORBA habrán alcanzado un nivel madurez importante, veremos como comienzan a implantarse soluciones con las aportaciones y beneficios de trabajar en entornos CORBA.

Resumen de la conclusiones

Tras analizar las solución CORBA seguro consideramos tres aspectos esenciales el impacto en el desarrollo de aplicaciones, el nivel de aceptación del mercado, la madurez del producto.

Desarrollo de aplicaciones distribuidas

- Tiene todas las ventajas de una solución CORBA[11] añadidos a la seguridad. Además de conexiones mediante canal seguro, permite la construcción de aplicaciones con la posibilidad de gestión de privilegios completa.

- El impacto y complejidad sobre el desarrollo son mínimos frente a los beneficios. La solución expuesta es más sencilla que usar RMI con seguridad. Podemos plasmar nuestra solución de forma ortogonal al desarrollo e incluso disponer de las herramientas para establecer políticas de control y acceso en tiempo de ejecución. La solución estará dotada de una capacidad de continuidad, adaptabilidad al cambio y mantenibilidad muy altas.

- Posibilidad de desarrollo de aplicaciones de igual a igual, cliente y servidor asumen ambos papeles. La autenticación, el cifrado y las capacidades de los ORB permiten implementar soluciones en las que el cliente y servidor intercambien servicios.

Aceptación

- Faltan algunos años, hasta que este tipo de soluciones sea cotidiano y usadas profusamente en nuestro ámbito próximo. La Facultad afortunadamente apuesta por la formación en este campo.

Madurez

- El producto soporta gran parte del estándar para CORBA seguro. Es más que una conexión vía canal seguro entre ORB y cliente. La gestión dinámica de privilegios y la definición de complejos esquemas de autorización son realizables.

- El producto, tiene que ser completado con herramientas que guíen y faciliten la definición de políticas de seguridad ortogonales a las aplicaciones. Aunque siempre se las puede construir uno.

Todo lo expuesto no lleva a afirmar que esta solución debe ser, por lo menos, valorada en el desarrollo de nuevos sistemas de información y de sistemas que deban renovarse. Sistemas en los que sea necesario distribuir la carga de procesamiento entre cliente y servidores, y que además la seguridad sea un objetivo del proyecto.

BIBLIOGRAFÍA SOBRE SEGURIDAD DE CORBA

Blakely, Bob, “CORBA Security: An Introduction to Safe Computing with Objects”, The Addision-Wesley Object Technology Series, ISBN: 0201325659, October, 1999.
Bob Blakely es uno de muchos autores contribuyentes importantes a la especificaciones de Seguridad de Corba.
Blakely, Bob, “CORBA Security: An Introduction to Safe Computing with Objects”, The Addision-Wesley Object Technology Series, ISBN: 0201325659, October, 1999.
2. Kohl J, Neuman C., “The Kerberos Network Authentication Service (V5)”, Net-work Working Group RFC 1510, September 1993.
3. Java Cryptography Architecture and API Reference and Specification, http:// java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.html
4. The Object Management Group, “The Common Object Request Broker: Archi-tecture and Sepcification”, Version 2.2, Feburary 1998.
5. The Object Management Group, “CORBAservices: Common Object Services Specification”, November 1997.
6. The Object Management Group, “Security Service Specification”, Version 1.2 Draft 4.1, 5 January, 1998.
7. The Object Management Group, “Security Service Specification”, Version 1.5, March 1999.
8. The Object Management Group, “Security Service Specification”, Version 1.7, Jan 2000.
9. Object Oriented Concepts, Inc. “ORBACUS C++ and Java”, Version 3.3, 2000.
10. Diego Sevilla.Aplicaciones Distribuidas en Internet/Intranets: de los sockets a los Objetos Distribuidos. Noviembre 1998.
11. Jaime Jaworski.Java 1.2 al descubierto. Prentice Hall,ISBN 84-8322-061-X. 1998.

ANEXOS

ANEXO FUENTES DEL PROGRAMA DEL BANCO

Definición de Interfaces

```
// Bank.idl

module Bank {
  interface Account {

    float balance();
  };

  interface AccountManager {

    Account open(in string name);

  };

};
```

Client.java

```
// Client.java
//tabstop=4
//****************************************************************************
// Project: ORBAsec SL2
//
// ---------------------------------------------------------------------------
// Copyright (C) 2000 Adiron, LLC.
//                    All rights reserved.
// ---------------------------------------------------------------------------
// $Id$
//****************************************************************************

/*
 * NOTE: To run this application, because we are using IAIK-JCE 2.51
 * your JDK needs to have the following permission added in the
 * grant section of in the JDK's security java.policy file,
 * which is located in:
 *              $JAVA_HOME/jre/lib/security/java.policy
 *
 * grant {
 *   permission java.security.SecurityPermission "removeProviderProperty.IAIK";
 * }
 */
import java.awt.*;

public class Client
extends java.applet.Applet
{
      // Base64 Encoding of a PKCS#8 format Bart's DSA private key
      // encrypted with "mypassword" and pbeWithMD5AndDES_CBC.
      final byte[] encoded_ecrypted_private_key = (
      "-----BEGIN ENCRYPTED PRIVATE KEY-----\n"+
      "MIHnMBoGCSqGSIb3DQEFAzANBAjTl0UvLU/XuwIBAQSByB4BkTxezkQU5NCD3SBF\n"+
      "PCN5uINhEsqPrW2CJNim0W3oosEQvAbmbnKRKLhgb2sxE3F0hejQ6Q4YX3ER3qM5\n"+
      "QxEiMUygYMWsBShYNdm66HDm4ZpzZfe97xXt3x30cljDGMr0gouXh9ViCunCKMoZ\n"+
```

```
"fYSb+nMjqiKAmZ0UvS+w89aXfYvz0sAIHam4jVS2b97xG9i1gR3i2vZdU5J1tU62\n"+
"wz16PKwNFsaFIKC6n5/dq6V/BoN8Tr8OwCPH42QDmTrAzZRsdhj/jX87\n"+
"-----END ENCRYPTED PRIVATE KEY-----\n"
).getBytes();

// Base64 Encoding of a PKCS#8 format of Bart's unecrypted
// DSA private key.
final byte[] encoded_private_key = (
"-----BEGIN PRIVATE KEY-----\n"+
"MIHEAgEAMIGmBgUrDgMCDDCBnAJBAMFsutNNR17FOWaV1pS8i8R+WY4jtanXxc7I\n"+
"LWW2gn1E6VN4SEcwwL/x9MtW9HxuUQVL6JIA8w1D3E/vliTUZlsCFQC3uBC1jAk0\n"+
"9kKHjzYLltfMJrU+TQJATFPHJr2/u6ZUnX5zGTnGyTqGmifF2xe6PKxYnXs+AD+n\n"+
"NfKQz9B6PvEPNRVfGi73AzWve2pSEaEQNRj7pE6XGAQWAhQZtJRt4cC+Dib9gv+P\n"+
"VxUw6qCnOQ==\n"+
"-----END PRIVATE KEY-----\n"
).getBytes();

// Corresponding Base64 Encoding off Bart's X.509 Certificate chain
// belonging to the private key. Issuer Cert is SpringfieldCA.
final byte[] encoded_cert_chain = (
"-----BEGIN CERTIFICATE-----\n"+
"MIIB0jCCAZICAQAwCQYFKw4DAg0FADBCMRQwEgYDVQQHEwtTcHJpbmdmaWVsZDEq\n"+
"MCgGA1UEAxMhU3ByaW5nZmllbGQgQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk5\n"+
"MTIwNzE1MDUyNVoXDTA0MTIwNzE1MDUwN1owLTEUMBIGA1UEBxMLU3ByaW5nZmll\n"+
"bGQxFTATBgNVBAMTDEJhcnQgU2ltcHNvbjCB7jCBpgYFKw4DAgwwgZwCQQDBbLrT\n"+
"TUdexTlmldaUvIvEflmOI7Wp18XOyC1ltoJ9ROlTeEhHMMC/8fTLVvR8blEFS+iS\n"+
"APMNQ9xP75Yk1GZbAhUAt7gQtYwJNPZCh482C5bXzCa1Pk0CQExTxya9v7umVJ1+\n"+
"cxk5xsk6hponxdsXujysWJ17PgA/pzXykM/Qej7xDzUVXxou9wM1r3tqUhGhEDUY\n"+
"+6ROlxgDQwACQHPbYV8Dv9JBZFuoDj758xfz2PxFn+W61y34yLbFZ7o6NHuQVjqO\n"+
"1L8JxqEzl2VUwKKtAMvn630HBdM3sins6XowCQYFKw4DAg0FAAMvADAsAhQGpFNZ\n"+
"4fElNd6jhFptaO6dzsKQpgIULf/IqTp7GEDLbgqwyqi4Lipjp+k=\n"+
"-----END CERTIFICATE-----\n"+
"-----BEGIN CERTIFICATE-----\n"+
"MIIB6DCCAagCAQAwCQYFKw4DAg0FADBCMRQwEgYDVQQHEwtTcHJpbmdmaWVsZDEq\n"+
"MCgGA1UEAxMhU3ByaW5nZmllbGQgQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk5\n"+
"MTIwNzE1MDEyOFoXDTA0MTIwNzE1MDExOFowQjEUMBIGA1UEBxMLU3ByaW5nZmll\n"+
"bGQxKjAoBgNVBAMTIVNwcmluZ2ZpZWxkIENlcnRpZmljYXRlIEF1dGhvcml0eTCB\n"+
"7zCBpgYFKw4DAgwwgZwCQQDBbLrTTUdexTlmldaUvIvEflmOI7Wp18XOyC1ltoJ9\n"+
"ROlTeEhHMMC/8fTLVvR8blEFS+iSAPMNQ9xP75Yk1GZbAhUAt7gQtYwJNPZCh482\n"+
"C5bXzCa1Pk0CQExTxya9v7umVJ1+cxk5xsk6hponxdsXujysWJ17PgA/pzXykM/Q\n"+
"ej7xDzUVXxou9wM1r3tqUhGhEDUY+6ROlxgDRAACQQCJ6zY8kbMQW/KfUAt4vwUq\n"+
"ObZ2ekNTdXb24JFvUDR3GYQhqc+U+eIG9D7TMD+DR2hEid2cPpeRAwTQzz70Ye56\n"+
"MAkGBSsOAwINBQADLwAwLAIUSOF5PvSGIPhblgMH5BnQLCzVqD8CFA1iJ69asvYz\n"+
"ymYNOnvtlBQytak3\n"+
"-----END CERTIFICATE-----\n"
).getBytes();

// Corresponding Base64 Encoding off a X.509 Certificate chain
// which is the signer of Bart's certificate, SpringfieldCA.
final byte[] encoded_trusted_certs = (
"-----BEGIN CERTIFICATE-----\n"+
"MIIB6DCCAagCAQAwCQYFKw4DAg0FADBCMRQwEgYDVQQHEwtTcHJpbmdmaWVsZDEq\n"+
"MCgGA1UEAxMhU3ByaW5nZmllbGQgQ2VydGlmaWNhdGUgQXV0aG9yaXR5MB4XDTk5\n"+
"MTIwNzE1MDEyOFoXDTA0MTIwNzE1MDExOFowQjEUMBIGA1UEBxMLU3ByaW5nZmll\n"+
"bGQxKjAoBgNVBAMTIVNwcmluZ2ZpZWxkIENlcnRpZmljYXRlIEF1dGhvcml0eTCB\n"+
"7zCBpgYFKw4DAgwwgZwCQQDBbLrTTUdexTlmldaUvIvEflmOI7Wp18XOyC1ltoJ9\n"+
"ROlTeEhHMMC/8fTLVvR8blEFS+iSAPMNQ9xP75Yk1GZbAhUAt7gQtYwJNPZCh482\n"+
"C5bXzCa1Pk0CQExTxya9v7umVJ1+cxk5xsk6hponxdsXujysWJ17PgA/pzXykM/Q\n"+
"ej7xDzUVXxou9wM1r3tqUhGhEDUY+6ROlxgDRAACQQCJ6zY8kbMQW/KfUAt4vwUq\n"+
"ObZ2ekNTdXb24JFvUDR3GYQhqc+U+eIG9D7TMD+DR2hEid2cPpeRAwTQzz70Ye56\n"+
```

```
  "MAkGBSsOAwINBQADLwAwLAIUSOF5PvSGIPhblgMH5BnQLCzVqD8CFA1iJ69asvYz\n"+
  "ymYNOnvtlBQytak3\n"+
  "-----END CERTIFICATE-----\n"
  ).getBytes();



  org.omg.SecurityLevel2.SecurityManager security_manager_ = null;
  private TextField _nameField, _balanceField;
  private Button _checkBalance;
  private Bank.AccountManager _manager                    = null;

  public void init()
{
        //
        // ORB and SL2 initialization
        //
        java.util.Properties props = new java.util.Properties();
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init( this, props );
        try {
              security_manager_ =
                    org.omg.SecurityLevel2.SecurityManagerHelper.narrow(
                    orb.resolve_initial_references("SecurityManager"));
        } catch ( org.omg.CORBA.ORBPackage.InvalidName in ){
              in.printStackTrace();
        }

        authenticate();

        //
        // Create client object
        //
        String ior = getParameter("ior");
        org.omg.CORBA.Object obj = orb.string_to_object(ior);
        if(obj == null)
        throw new RuntimeException();

           _manager = Bank.AccountManagerHelper.narrow(obj);
        //
        //
        //
        //Aspecto visual

            // Este GUI utiliza un enrejado de 2x2 elementos.^M
            setLayout(new GridLayout(2, 2, 5, 5));
            // Añadir los cuatro elementos.
            add(new Label("Nombre de la Cuenta"));
            add(_nameField = new TextField());
            add(_checkBalance = new Button("Comprobar Saldo"));
            add(_balanceField = new TextField());
            // Hacel el campo "balance" no editable.
            _balanceField.setEditable(false);

        //java.awt.Button button = new java.awt.Button("Hello");
        // button.addActionListener( button_listener_ );
        // this.add( button );
}
```

```
    public void
      authenticate()
    {
      org.omg.SecurityLevel2.PrincipalAuthenticator pa =
                  security_manager_.principal_authenticator();

            org.omg.Security.MechandOptions[] mechs =
                  security_manager_.supported_mechanisms();
            for(int i=0;i<mechs.length;i++)
                  System.out.println(mechs[i].mechanism_type);

            // Use SSL with cipher suites needing DSA signed Diffe-Hillman
            // Ephemeral certificates, using only non-exportable encryption
            String  mechanism =
                  orbasec.corba.MechUtil.SSL_DHE_DSS_EXPORT_MECH;
            String      security_name = ""; // assigned from AuthMethod
/*
            // This authentication method is used with the unencrypted
            // private key
            int method =
                  orbasec.SecLev2.SecSSLPrivateKeyAndCertificates.value;
            orbasec.SecLev2.SSLPrivateKeyAndCertificates data =

      orbasec.corba.AuthUtil.default_SSLPrivateKeyAndCertificates();
            data.encoded_private_key         = encoded_private_key;
            data.key_algorithm               = "DSA";
            data.key_format                  = "PKCS#8";
            data.security_provider           = "IAIK";
            data.encoded_certificate_chain   = encoded_cert_chain;
            data.encoded_trusted_authorities = encoded_trusted_certs;

            org.omg.CORBA.Any                    auth_data                    =
org.omg.CORBA.ORB.init().create_any();
            orbasec.SecLev2.SSLPrivateKeyAndCertificatesHelper.insert(
                  auth_data, data );
*/
            // This autentication key involves a password protected
            // encrypted private key.
            int method =

      orbasec.SecLev2.SecSSLEncryptedPrivateKeyAndCertificates.value;
            orbasec.SecLev2.SSLEncryptedPrivateKeyAndCertificates data =

      orbasec.corba.AuthUtil.default_SSLEncryptedPrivateKeyAndCertificates();
            data.encoded_private_key         = encoded_ecrypted_private_key;
            data.key_algorithm               = "DSA";
            data.key_format                  = "PKCS#8";
            data.key_pass                    = "mypassword";
            data.security_provider           = "IAIK";
            data.encoded_certificate_chain   = encoded_cert_chain;
            data.encoded_trusted_authorities = encoded_trusted_certs;

            org.omg.CORBA.Any                    auth_data                    =
org.omg.CORBA.ORB.init().create_any();
            orbasec.SecLev2.SSLEncryptedPrivateKeyAndCertificatesHelper.insert(
                  auth_data, data );

            org.omg.Security.SecAttribute privileges[] =
                  new org.omg.Security.SecAttribute[0];
            org.omg.SecurityLevel2.CredentialsHolder creds =
```

```
                new org.omg.SecurityLevel2.CredentialsHolder();
            org.omg.CORBA.AnyHolder
             continuation_data =
                new org.omg.CORBA.AnyHolder(),
             auth_specific_data =
                new org.omg.CORBA.AnyHolder();

            pa.authenticate(
                method,
                mechanism,
                security_name,
                auth_data,
                privileges,
                creds,
                continuation_data,
                auth_specific_data
            );
      }

      void
      print_credentials()
      {
            System.out.println("Own Credentials:");
            orbasec.corba.CredUtil.dumpCredentials(System.out,
                security_manager_.own_credentials() );
            System.out.flush();
      }

public boolean action(Event ev, Object arg) {
          if(ev.target == _checkBalance) {
               // Pedirle al AccountManager que abra una cuenta con un nombre.
               // Coger el nombre del campo "_nameField".
               Bank.Account account = _manager.open(_nameField.getText());
               // Actualizar el balance en el GUI.
               _balanceField.setText(Float.toString(account.balance()));
               return true;
          }
    return false;
  }

}
```

Server

```
//tabstop=4
//****************************************************************************
// Project: ORBAsec SL2 PROGRAMACION PARA LAS COMUNI
//
// ---------------------------------------------------------------------------
// Copyright (C) 2000 Juan Mármol Castillo.
//                    Alejandro Roca Alhama
// ---------------------------------------------------------------------------
// $Id$
//****************************************************************************

import org.omg.CORBA.*;
import java.io.*;
import Bank.*;

public class Server
```

```
{
      public static void main(String args[])
      {
            try {
                  //
                  // ORB, BOA, and SL2 initialization
                  //
                  java.util.Properties props = new java.util.Properties();
                  props.put( "org.omg.CORBA.ORBClass", "orbasec.ORB" );
                  ORB orb = org.omg.CORBA.ORB.init( args, props );
                  BOA boa = orb.BOA_init( args, props );

            //
            // Get SecurityLevel2::Current from ORB
            //
            org.omg.SecurityLevel2.SecurityManager security_manager =
                  org.omg.SecurityLevel2.SecurityManagerHelper.narrow(
                  orb.resolve_initial_references("SecurityManager"));

            //
            // Authenticate using PrincipalAuthenticator
            //
            org.omg.SecurityLevel2.PrincipalAuthenticator pa;
            pa = security_manager.principal_authenticator();

                  // Use SSL with cipher suites needing DSA signed Diffe-
Hillman
                  // Ephemeral certificates, using only exportable encryption
                  String  mechanism =
                        orbasec.corba.MechUtil.SSL_DHE_DSS_EXPORT_MECH;
                  String      security_name = "";// assigned from AuthMethod

                  int                           method                          =
orbasec.SecLev2.SecSSLKeyStoreWithStorePass.value;
                  orbasec.SecLev2.SSLKeyStoreWithStorePass data =

      orbasec.corba.AuthUtil.default_SSLKeyStoreWithStorePass();
                  data.keystore  = "FILE:./demo.keystore";
                  data.storetype = "IAIKKeyStore";
                  data.storepass = "mystorepass";
                  data.keyalias  = "homer";
                  data.keypass   = "mypassword";
                  data.usage                                                    =
orbasec.SecLev2.CredentialsUsage.SecAcceptOnly;

                  org.omg.CORBA.Any auth_data = orb.create_any();
                  orbasec.SecLev2.SSLKeyStoreWithStorePassHelper.insert(
                        auth_data, data );

                  org.omg.Security.SecAttribute privileges[] =
                        new org.omg.Security.SecAttribute[0];
                  org.omg.SecurityLevel2.CredentialsHolder creds_holder =
                        new org.omg.SecurityLevel2.CredentialsHolder();
                  org.omg.CORBA.AnyHolder
                   continuation_data =
                        new org.omg.CORBA.AnyHolder(),
                   auth_specific_data =
                        new org.omg.CORBA.AnyHolder();

                  pa.authenticate(
```

```
                        method,
                        mechanism,
                        security_name,
                        auth_data,
                        privileges,
                        creds_holder,
                        continuation_data,
                        auth_specific_data
                  );

                  // Change the credentials requirements so that we
                  // request a client certificate for authentication.
                  short                       assoc_opts                      =
creds_holder.value.accepting_options_required();
                  assoc_opts |= org.omg.Security.EstablishTrustInClient.value;
                  creds_holder.value.accepting_options_required(assoc_opts);

                  //
                  // Print credentials to screen
                  //
                  org.omg.SecurityLevel2.Credentials[] credlist =
                        security_manager.own_credentials();
                  System.out.println("Own Credentials:");
                  orbasec.corba.CredUtil.dumpCredentials(System.out,credlist);
                  System.out.flush();

                  //
                  // Create implementation object
                  //

                        //Creamos el Servant
                  // El objeto que despachara las peticiones

                           Bank.AccountManager manager =
                                 new AccountManagerImpl(orb);



                  //
                  // Save reference
                  //
                  try {
                        String           ref = orb.object_to_string(manager);
                        String           refFile;
                        FileOutputStream file;
                        PrintWriter      out;

                        refFile = "Server.ref";
                        file    = new FileOutputStream(refFile);
                        out     = new PrintWriter(file);
                        out.println(ref);
                        out.flush();
                        file.close();

                        refFile = "ClientApplet.html";
                        file    = new FileOutputStream(refFile);
                        out     = new PrintWriter(file);
                        out.println(
                              "<h1>ORBASEC Practicas CORBA SEGURO</h1>\n"+
                              "<h2>Saldo en el Banco</h2>\n"+
```

```
                              "<h3>Juan Mármol y Alejandro Alhama</h3>\n"+
                              "<hr>\n"+
                              "<center>\n"+
                              "<applet\n" +
                              "    codebase=.\n" +
                              "    code=\"Client.class\"\n" +
                              "    archive=\"SL2.jar,SSL_IAIK.jar,
                                     iaik_jce_full.jar,
                                     iaik_ssl.jar\"\n" +
                              "    width=200\n" +
                              "    height=80>\n" +
                               "       <param name=org.omg.CORBA.ORBClass
                                   value=orbasec.ORB>\n" +
                              "       <param name=ior value="+ref+">\n" +
                              "</applet>\n"+
                                "</center>\n"+
                              "<hr>\n"
                        );


                        out.flush();
                        file.close();
                    } catch(IOException ex) {
                        System.err.println("Can't write to `" +
                                           ex.getMessage() + "'");
                        System.exit(1);
                    }

                    //
                    // Run implementation
                    //
                    System.out.println("\nBankManager   esta   listo   esperando
peticiones...");
                    System.out.flush();

                    //Modelo de concurrencia bloqueante
                    boa.impl_is_ready(null);

                    System.exit(0);
                } catch(Exception ex) {
                    System.err.println(ex.getMessage());
                    ex.printStackTrace();
                    System.exit(1);
                }
            }
}
```

Applet generado

ORBASEC Practicas CORBA SEGURO

Saldo en el Banco

Juan Mármol y Alejandro Alhama

```
<h1>ORBASEC Practicas CORBA SEGURO</h1>
<h2>Saldo en el Banco</h2>
<h3>Juan Mármol y Alejandro Alhama</h3>
<hr>
<center>
<applet
    codebase=.
    code="Client.class"
    archive="SL2.jar,SSL_IAIK.jar,iaik_jce_full.jar,iaik_ssl.jar"
    width=200
    height=80>
       <param name=org.omg.CORBA.ORBClass value=orbasec.ORB>
       <param name=ior
value=IOR:000000000000001c49444c3a42616e6b2f4163636f756e744d616e616765723a312e3
0000000001000000000000003c000101000000000a3132372e302e302e3100000000000010000
00003a40a64b00060ae0000000000000000100000014000000080000000fe00da0420>
</applet>
</center>
<hr>
```

ANEXO RMI

Ejemplo obtenido de Java al descubierto[11]. Ha sido incluido aquí para que el lector tenga acceso a un ejemplo de uso RMI y le ayude a formar su propia opinión.

Cliente

```
package ju.ch38.client;
import ju.ch38.server.*;
import java.rmi.*;

public class MyClient {
 static String hostName="athome.jaworski.com";
 public static void main(String args[]) {
  try {
   MyServer server = (MyServer) Naming.lookup("//"+hostName+"/MyServer");
   int n = server.getDataNum();
   for(int i=0;i<n;++i) {
    System.out.println(server.getData(i));
   }
  } catch (Exception ex) {
   System.out.println(ex);
  }
 }
}
```

servidor

```
package ju.ch38.server;

import java.rmi.*;

public interface MyServer extends Remote {
 int getDataNum() throws RemoteException;
 String getData(int n) throws RemoteException;
}
package ju.ch38.server;

import java.rmi.*;
import java.rmi.server.*;

public class MyServerImpl extends UnicastRemoteObject
 implements MyServer {
 static String hostName="athome.jaworski.com";
 static String data[] = {"Remote","Method","Invocation","Is","Great!"};
 public MyServerImpl() throws RemoteException {
  super();
 }
 public int getDataNum() throws RemoteException {
  return data.length;
 }
 public String getData(int n) throws RemoteException {
  return data[n%data.length];
 }
 public static void main(String args[]){
  System.setSecurityManager(new RMISecurityManager());
  try {
   MyServerImpl instance = new MyServerImpl();
   Naming.rebind("//"+hostName+"/MyServer", instance);
```

```
 System.out.println("I'm registered!");
} catch (Exception ex) {
 System.out.println(ex);
```

ANEXO EJEMPLO DE ORBACUS CON FREESSL

Este quier mostrar una pequeña muestra de código para que el lecto pueda forma su propio criterio sobre la utilización de este producto. Para mas información www.ooc.com.

CLIENTE

Politicas

```
// Java
com.ooc.FSSL.Certificate[] myCerts = new com.ooc.FSSL.Certificate[2];
myCerts[0] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("client.der"));
myCerts[1] = fsslManager.create_certificate(
    com.ooc.FSSL.FSSL.load_file("ca.der"));
int id = fsslManager.create_context(
    myCerts,
    com.ooc.FSSL.FSSL.load_file("clientkey.der"),
    com.ooc.FSSL.FSSL.string_to_PassPhrase("foobar"),
    myTrustDecider,
    com.ooc.FSSL.FSSL.get_default_ciphers());
org.omg.CORBA.PolicyManager policyManager =
    org.omg.CORBA.PolicyManagerHelper.narrow(
        orb.resolve_initial_references("ORBPolicyManager"));
org.omg.CORBA.Policy[] pl = new org.omg.CORBA.Policy[1];
pl[0] = fsslManager.create_context_policy(id);
policyManager.add_policy_overrides(pl);
```

Identificación

```
// Java
com.ooc.OCI.FSSLIOP.TransportInfo fssliopInfo = null;
if(!obj._non_existent())
{
    org.omg.CORBA.portable.ObjectImpl objImpl =
        (org.omg.CORBA.portable.ObjectImpl)obj;
    com.ooc.CORBA.Delegate objDelegate =
        (com.ooc.CORBA.Delegate)objImpl._get_delegate();

    com.ooc.OCI.TransportInfo info =
        objDelegate.get_oci_transport_info();
    fssliopInfo = com.ooc.OCI.FSSLIOP.TransportInfoHelper.narrow(info);
}
```