DISEÑO E IMPLEMENTACIÓN DEL SERVICIO DE EVENTOS DE CORBA PARA JAVA IDL

CÓDIGO FUENTE

2002 **Víctor M. Díaz Caballero**

ÏNDICE DE CONTENIDOS:

PAQUETE COSEVENTCHANNELADMINIMPL

Paquete CosEventChannelAdminImpl

EventChannelServant.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import org.omg.CosEventChannelAdmin.*;

import java.util.Properties;


/**
    Esta clase implelementa el servant del canal, extendiendo la
clase _EventChannelImplBase. Además implementa el interfaz
Runnable, lo cual quiere decir que lanza un hilo de ejecución
propio.
    Este objeto es creado por la aplicación EventChannelFactory y
permanece activo mientras esta aplicación no lo destruya. Crea los
administradores de productores y consumidores y permanece a la
escucha de peticiones por parte de los clientes.
*/
public class EventChannelServant
                        extends _EventChannelImplBase
                        implements Runnable
{
    private ConsumerAdminImpl cadmin;
    private SupplierAdminImpl sadmin;

    private String sargs[];
```

```
    private ORB orb;
    private Thread thread;

    private Misc varios = new Misc ();

    private String Nombre    = new String ();
    private String Tipo      = new String ();
    private String Direccion = new String ();
    private String Puerto    = new String ();
    private EventChannelMonitor monitor;

    private Properties props;

/**
    El constructor recibe como argumento una clase
EventChannelParams donde encuentra todos los parámetros que
necesita: el nombre del canal, el tipo del canal (nombre del
contexto en que se coloca en el servidor de nombres), la referencia
del monitor, la dirección del servidor de nombres y su puerto de
escucha.
    Es aquí donde se crean el administrador de productores y el de
consumidores.
*/
    public EventChannelServant (EventChannelParams params)
    {
        Nombre    = params.getNombre();
        Tipo      = params.getTipo();
        Direccion = params.getDireccion();
        Puerto    = params.getPuerto();
        monitor   = params.getMonitor();
        cadmin    = new ConsumerAdminImpl (params);
        sadmin    = new SupplierAdminImpl (params);

        params.setSupplierAdminRef (sadmin);
    }
/**
    Devuelve la referencia del administrador de consumidores.
*/
    public ConsumerAdmin for_consumers ()
    {
        Mensaje (monitor.INFO, "Un Consumer pide conexion.");
        return cadmin;
    }

/**
    Devuelve la referencia del administrador de productores.
*/
    public SupplierAdmin for_suppliers ()
    {
        Mensaje (monitor.INFO, "Un Supplier pide conexion.");
        return sadmin;
    }

/**
    Método que destruye el canal. Llama a las funciones destroy de
los administradores para que éstos destruyan a los proxies y estos
a su vez a los clientes conectados a ellos.  Se desconecta del ORB,
elimina la referencia del servidor de nombres y destruye el objeto.
*/
    public void destroy ()
    {
```

```
        Mensaje (monitor.INFO, "Va a destruirse el canal.");

        sadmin.destroy();
        cadmin.destroy();

        try
        {
            //Desconectamos del ORB
            orb.disconnect (this);
            orb.shutdown(false);

            // Eliminamos la referencia en el servidor de nombres
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references("NameService");
            NamingContext ncRef =
                    NamingContextHelper.narrow(objRef);
            NameComponent nc = new NameComponent(Nombre, Tipo);
            NameComponent path []= {nc};
            ncRef.unbind (path);

            try
            {
                this.finalize();
            }catch (Throwable t) {}

        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (monitor._ERROR
                             ,varios.TraduceSystemException(ex));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR
                             , _ERROR en clase Canal: " + e);
        }

        Mensaje (monitor.INFO, "Canal Destruido");
    }

/**
    Método encargado de iniciar el ORB y conectar esta clase a él.
También se encarga de darle de alta en el servidor de nombres. Si
ya existe un objeto con ese nombre sale devolviendo un mensaje
explicativo de lo ocurrido, en caso contrario arranca el hilo y
devuelve null.
*/
    public String Conectar ()
    {
        try
        {
            props = new Properties();

            props.put ("org.omg.CORBA.ORBInitialHost",Direccion);
            props.put ("org.omg.CORBA.ORBInitialPort",Puerto);

            // Iniciamos el ORB
            orb = ORB.init(sargs, props);

            // Conectamos el servidor al ORB
            orb.connect (this);
```

```
            // Obtenemos el contexto de nombrado
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references("NameService");
            NamingContext ncRef =
                    NamingContextHelper.narrow(objRef);

            // Vemos si ya existe un canal con ese nombre
            if (YaExiste (ncRef, Nombre))
                return ("Ya existe un canal con ese nombre");

            // Enlazamos la referencia del objeto en el servidor
            // de nombres

            NameComponent nc = new NameComponent(Nombre, Tipo);
            NameComponent path[] = {nc};
            ncRef.bind(path, this);

            // Arrancamos el hilo que nos hará permanecer a la
            // escucha.
            thread = new Thread (this);
            thread.start ();

            return null;
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            return varios.TraduceSystemException(ex);
        }
        catch(Exception e)
        {
            return ("ERROR en clase EventChannelServant: " + e);
        }
    }

/**
    Este hilo no hace nada, tan sólo permanece dormido permitiendo
que el servidor permanezca vivo a la escucha de peticiones.
*/
    public void run ()
    {
        java.lang.Object sync = new java.lang.Object();

        try
        {
            synchronized(sync)
            {
                sync.wait();
            }
        } catch (Exception e) {}
    }

/**
    Método encargado de comprobar si en el contexto de nombrado que
se le pasa como primer argumento, ya existe un objeto con el nombre
que se le pasa como segundo.
    Devuelve true en caso de que así sea y false en caso contrario.
*/
    private boolean YaExiste (NamingContext nacon, String nombre)
    {
        try
```

```
        {
            BindingListHolder bl = new BindingListHolder();
            BindingIteratorHolder blIt =
                                        new BindingIteratorHolder();
            nacon.list(1000, bl, blIt);

            Binding bindings[] = bl.value;
            if (bindings.length == 0)
                return false;
            for (int i=0; i < bindings.length; i++)
            {
                org.omg.CORBA.Object obj =
                          nacon.resolve(bindings[i].binding_name);
                String objStr = orb.object_to_string(obj);
                int lastIx = bindings[i].binding_name.length-1;

                if(bindings[i].binding_name[lastIx].id.equals
                                                        (nombre))
                    return true;
            }
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(ex));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR, "ERROR en clase Canal: " + e);
        }
        return false;
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "EventChannelServant", mensaje);
    }
}
```

SupplierAdminImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CosEventChannelAdmin.*;
import java.util.Vector;

/**
    Esta clase extiende a _SupplierAdminImplBase implementando al
administrador de productores.
*/
public class SupplierAdminImpl extends _SupplierAdminImplBase
{
    // El tamaño inicial de los vectores no es importante pues lo
    // modifican automáticamente. Esto es simplemente para que al
    // principio no haya muchas realocaciones de memoria.
```

```
    private Vector proxypushvector = new Vector (32);
    private Vector proxypullvector = new Vector (32);

    private ConsumerAdminImpl cons_admin_impl;
    private EventChannelMonitor monitor;
    private EventChannelParams event_channel_params;

/**
    El constructor recibe como argumento una clase
EventChannelParams donde encuentra todos los parámetros que
necesita:  la referencia del monitor y la del administrador de
consumidores.
*/

    public SupplierAdminImpl (EventChannelParams params)
    {
        // Obtenemos la referencia a la lista donde enviar los
        // mensajes
        monitor = params.getMonitor();

        // Obtenemos la referencia al administrador de
        // consumidores.
        cons_admin_impl = params.getConsumerAdminRef();

        event_channel_params = params;
    }

/**
    Método invocado por un PushSupplier como primera fase de su
conexión al canal.
    Crea un ProxyPushConsumer y devuelve su referencia.
*/
    public synchronized ProxyPushConsumer obtain_push_consumer ()
    {
        ProxyPushConsumerImpl aux;

        aux = new ProxyPushConsumerImpl (event_channel_params);

        synchronized (proxypushvector)
        {
            proxypushvector.addElement (aux);
        }

        Mensaje (monitor.INFO, "Un Push Supplier ha realizado la
                                primera fase de la conexión.");

        return aux;
    }

/**
    Método invocado por un PullSupplier como primera fase de su
conexión al canal.
    Crea un ProxyPullConsumer y devuelve su referencia.
*/
    public synchronized ProxyPullConsumer obtain_pull_consumer()
    {
        ProxyPullConsumerImpl aux;

        aux = new ProxyPullConsumerImpl (event_channel_params);

        synchronized (proxypullvector)
```

```
        {
            proxypullvector.addElement (aux);
        }

        Mensaje (monitor.INFO, "Un Pull Supplier ha realizado la
                                primera fase de la conexión.");

        return aux;
    }

/**
    Método invocado por un ProxyPushConsumer para indicar que
quiere desconectarse del canal (es decir que el PushSupplier
conectado a él quiere desconectarse).
*/
    public synchronized void ProxyPushDesconecta
                              (ProxyPushConsumerImpl proxypushref)
    {
        boolean conectado;

        synchronized (proxypushvector)
        {
            conectado = proxypushvector.removeElement
                                                  (proxypushref);
        }

        if (conectado == true)
            monitor.Accion (monitor.DESCONECTADO_PUSH_SUPPLIER);
        else
            Mensaje (monitor._ERROR, "El Proxy Push Consumer "
                                     + proxypushref +
                                     " ha intentado desconectarse
                                     sin estar conectado.");
    }

/**
    Método invocado por un ProxyPullConsumer para indicar que
quiere desconectarse del canal (es decir que el PullSupplier
conectado a él quiere desconectarse).
*/
    public synchronized void ProxyPullDesconecta
                              (ProxyPullConsumerImpl proxypullref)
    {
        boolean conectado;

        synchronized (proxypullvector)
        {
            conectado = proxypullvector.removeElement
                                                  (proxypullref);
        }

        if (conectado == true)
            monitor.Accion (monitor.DESCONECTADO_PULL_SUPPLIER);
        else
            Mensaje (monitor._ERROR, "El Proxy Pull Consumer "
                                     + proxypullref +
                                     " ha intentado desconectarse
                                     sin estar conectado.");
    }

/**
```

```
    Método invocado por el servant cuando recibe la orden de
destruirse. Se encarga de eliminar a todos los Proxy Consumers así
como de invocar el método disconnect__supplier de estos para
eliminar también a los Suppliers.
*/
    public void destroy ()
    {
        ProxyPushConsumerImpl aux_push;
        ProxyPullConsumerImpl aux_pull;

        synchronized (proxypullvector)
        {
            synchronized (proxypushvector)
            {
                while (!proxypullvector.isEmpty())
                {
                    aux_pull = (ProxyPullConsumerImpl)
                                        proxypullvector.get (0);
                    proxypullvector.remove (0);
                    aux_pull.disconnect_pull_supplier ();
                    monitor.Accion
                              (monitor.DESCONECTADO_PULL_SUPPLIER);
                }

                while (!proxypushvector.isEmpty())
                {
                    aux_push = (ProxyPushConsumerImpl)
                                           proxypushvector.get (0);
                    proxypushvector.remove (0);
                    aux_push.disconnect_push_supplier ();
                    monitor.Accion
                              (monitor.DESCONECTADO_PUSH_SUPPLIER);
                }
            }
        }
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "SupplierAdmin", mensaje);
    }
}
```

ConsumerAdminImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import java.util.Vector;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CORBA.*;

/**
    Esta clase extiende a _ConsumerAdminImplBase implementando al
administrador de consumidores.
```

```
    En esta clase se encuentra la instancia del buffer. Este objeto
es de vital importancia ya que es el encargado de almacenar los
eventos que recibe el canal.
*/
public class ConsumerAdminImpl extends _ConsumerAdminImplBase
{
    private Vector proxypushvector = new Vector (32);
    private Vector proxypullvector = new Vector (32);

    private ProxyPushSupplierImpl proxypushimpl;
    private ProxyPullSupplierImpl proxypullimpl;
    private Buffer buffer;

    private EventChannelParams event_channel_params;
    private EventChannelMonitor monitor;

    private long eliminados = 0;

/**
    El constructor recibe como argumento una clase
EventChannelParams donde encuentra el parámetro que necesita: la
referencia del monitor.
    Aquí se crea el buffer.
*/
    public ConsumerAdminImpl (EventChannelParams params)
    {
        monitor = params.getMonitor();
        event_channel_params = params;
        params.setConsumerAdminRef (this);

        buffer = new Buffer (params);
    }

/**
    Método invocado por un PushConsumer como primera fase de su
conexión al canal.
    Crea un ProxyPushSupplier y devuelve su referencia.
*/
    public synchronized ProxyPushSupplier obtain_push_supplier ()
    {
        ProxyPushSupplierImpl aux;

        aux = new ProxyPushSupplierImpl (event_channel_params);

        synchronized (proxypushvector)
        {
            proxypushvector.addElement (aux);
        }

        Mensaje (monitor.INFO, "Un Push Consumer ha realizado
                                la primera fase de la conexión.");

        return aux;
    }

/**
    Método invocado por un PullConsumer como primera fase de su
conexión al canal.
    Crea un ProxyPullSupplier y devuelve su referencia.
*/
    public synchronized ProxyPullSupplier obtain_pull_supplier()
```

```
        {
            ProxyPullSupplierImpl aux;

            aux = new ProxyPullSupplierImpl (event_channel_params);

            synchronized (proxypullvector)
            {
                proxypullvector.addElement (aux);
            }

            Mensaje (monitor.INFO, "Un Pull Consumer ha realizado
                                    la primera fase de la conexión.");

            return aux;
        }

    /**
        Método invocado por un ProxyPushSupplier para indicar que
    quiere desconectarse del canal (es decir que el PushConsumer
    conectado a él quiere desconectarse).
    */
        public synchronized void ProxyPushDesconecta
                                    (ProxyPushSupplierImpl proxypushref)
        {
            boolean conectado;

            synchronized (proxypushvector)
            {
                conectado = proxypushvector.removeElement
                                                        (proxypushref);
            }

            if (conectado == true)
                monitor.Accion (monitor.DESCONECTADO_PUSH_CONSUMER);
            else
                Mensaje (monitor._ERROR, "Un Proxy Push Supplier  ha
                                          intentado desconectarse sin
                                          estar conectado.");
        }

    /**
        Método invocado por un ProxyPullSupplier para indicar que
    quiere desconectarse del canal (es decir que el PullConsumer
    conectado a él quiere desconectarse).
    */
        public synchronized void ProxyPullDesconecta
                                    (ProxyPullSupplierImpl proxypullref)
        {
            boolean conectado;

            synchronized (proxypullvector)
            {
                conectado = proxypullvector.removeElement
                                                        (proxypullref);
            }

            if (conectado == true)
                monitor.Accion(monitor.DESCONECTADO_PULL_CONSUMER);
            else
                Mensaje (monitor._ERROR, "Un Proxy Pull Supplier ha
                                          intentado desconectarse sin
```

```
                                        estar conectado.");
    }
/**
    Método invocado por el buffer cuando elimina un evento que ha
quedado obsoleto. Se encarga de incrementar la variable eliminados
que es utilizada en el método try_pull.
*/
    public void elementoEliminadoEnBuffer ()
    {
        eliminados ++;
    }


/**
    Método que añade el Any que recibe como parámetro al buffer.
*/
    public void put (org.omg.CORBA.Any dato)
    {
        buffer.put (dato);
    }

/**
    Método invocado por el servant cuando recibe la orden de
destruirse. Se encarga de eliminar a todos los ProxySuppliers  así
como de invocar el método disconnect__consumer de estos para
eliminar también a los consumidores.
*/
    public void destroy ()
    {
        ProxyPushSupplierImpl aux_push;
        ProxyPullSupplierImpl aux_pull;

        synchronized (proxypullvector)
        {
            while (!proxypullvector.isEmpty())
            {
                aux_pull = (ProxyPullSupplierImpl)
                                        proxypullvector.get (0);
                aux_pull.disconnect_pull_consumer ();
                proxypullvector.remove (0);
                monitor.Accion(monitor.DESCONECTADO_PULL_CONSUMER);
            }
        }

        synchronized (proxypushvector)
        {
            while (!proxypushvector.isEmpty())
            {
                aux_push = (ProxyPushSupplierImpl)
                                        proxypushvector.get (0);
                aux_push.disconnect_push_consumer ();
                proxypushvector.remove (0);
                monitor.Accion(monitor.DESCONECTADO_PUSH_CONSUMER);
            }
        }
    }

/**
    Método invocado por un Proxysupplier que devuelve un evento si
es que hay alguno disponible o null en caso contrario.  Al
```

```
finalizar, el booleanHolder contiene true si hay evento y false en
caso contrario.

    Cuando se invoca el método, el LongHolder contiene el índice
del último evento que pidió. Así, el evento que le corresponde al
Proxy es el índice menos el número de elementos eliminados (o el
primero es caso de que esta resta dé un número negativo).

    Al terminar el método el LongHolder contiene el índice
actualizado.
*/
    public Any try_pull (BooleanHolder nuevoEvento
                        , LongHolder indice)
    {
        Any any;
        Long aux = new Long (indice.value - eliminados);

        if (aux.intValue() < 0)
        {
            aux = new Long (0);
            indice.value = eliminados;
        }
        any = buffer.get (aux.intValue(), nuevoEvento);

        if (nuevoEvento.value)
        {
            indice.value ++;
        }

        return any;
    }


/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "ConsumerAdminImpl", mensaje);
    }
}
```

ProxyPushSupplierImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventComm.*;
import org.omg.CORBA.*;

/**
    Esta clase extiende a _ProxyPushSupplierImplBase implementando
al ProxyPushSupplier que es la parte del canal que ve el Push
Consumer y que se encarga de proveerle con los eventos recibidos..
Además implementa el interfaz Runnable, lo cual quiere decir que
lanza un hilo de ejecución propio.
*/
```

```
public class ProxyPushSupplierImpl
                                extends _ProxyPushSupplierImplBase
                                implements Runnable
{
    private boolean conectado = false ;

    private PushConsumer push_consumer_ref;
    private ConsumerAdminImpl cons_admin_impl;
    private Thread thread;

    private Misc varios = new Misc ();
    private EventChannelMonitor monitor;

    // Indice que lleva la cuenta de cuantos eventos llevos ya
    // pedidos.Es un holder para que el ConsumerAdmin pueda
    // actualizarlo en función de los elementos que hayan sido
    // eliminados en el buffer.
    private LongHolder indice = new LongHolder (0);

/**
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita: la referencia al monitor y al administrador de
consumidores
*/
    public ProxyPushSupplierImpl (EventChannelParams params)
    {
        //Obtenemos la referencia al monitor donde enviar los
        //mensajes
        monitor = params.getMonitor();

        //Obtenemos la referencia al administrador de consumidores.
        cons_admin_impl = params.getConsumerAdminRef();
    }

/**
    Método utilizado por el PushConsumer para conectarse al canal.
Es la segunda y última fase de la conexión. Una vez conectado, se
arranca el hilo.
    Se lanza la excepción AlreadyConnected si al invocar este
método, el Push Consumer ya está conectado al canal.
    Se lanza la excepción BAD_PARAM si se recibe null en lugar de
la referencia del Push Consumer. Debe conocerse dicha referencia
para poder enviarle los eventos.
*/
    public void connect_push_consumer (PushConsumer push_consumer)
        throws AlreadyConnected
    {
        if (conectado)
            throw new AlreadyConnected ();

        if (push_consumer == null)
            throw new BAD_PARAM ("Debes pasar tu referencia al
                                  ProxyPushSupplier cuando invoques
                                 el metodo connect_push_consumer");

        push_consumer_ref = push_consumer;

        conectado = true;

        monitor.Accion(monitor.CONECTADO_PUSH_CONSUMER);
```

```
        thread = new Thread (this);
        thread.start();
    }

/**
    Método invocado por el PushConsumer (quién sólo transmite una
orden del consumidor) para desconectarse.
    Desconecta al Proxy invocando el método ProxyPushDesconecta del
administrador de consumidores.
    También es invocado por el método run si detecta fallo en la
comunicación.
*/
    public void disconnect_push_supplier()
    {
        if (conectado)
        {
            conectado = false;
            cons_admin_impl.ProxyPushDesconecta (this);
        }
    }

/**
    Método invocado por el administrador de consumidores informar
al PushConsumer de que lo ha desconectado. Esto será así cuando se
destruya el canal.
*/
    public void disconnect_push_consumer ()
    {
        if (conectado)
        {
            conectado = false;

            try
            {
                push_consumer_ref.disconnect_push_consumer ();
            }
            catch (org.omg.CORBA.SystemException e)
            {
                // Aquí no hacemos nada, simplemente capturamos una
                // posible excepción para en caso de que el
                // consumidor no conteste, el canal pueda seguir
                // destruyendo al resto.
            }

        }
    }

/**
    Saca datos del buffer y los va entregando al Push Consumer.
*/
    public void run ()
    {
        try
        {
            while (conectado)
            {
                // Enviamos al PushConsumer el evento que extraemos
                // del buffer com el método pull ()
                Any aux = pull();
                if (conectado) //He estado bloqueado y no sé que ha
```

```
                    push_consumer_ref.push (aux); //pasado mientras
                }
            }
            catch (org.omg.CORBA.SystemException ex)
            {
                // Si se ha producido un fallo en la comunicación, doy
                // de baja al consumidor en el canal de eventos.
                if (ex instanceof  org.omg.CORBA.COMM_FAILURE)
                {
                    disconnect_push_supplier ();
                }

                Mensaje (monitor._ERROR
                                ,varios.TraduceSystemException(ex));
            }
            catch (Disconnected d)
            {}
        }

    /**
        Este método se queda bloqueado hasta que esté disponible un
    evento que es lo que devuelve.
    */
        private Any pull ()
        {
            // Objeto utilizado ólamente para dormir un segundo.
            java.lang.Object sync = new java.lang.Object();

            Any aux = null;
            BooleanHolder nuevoEvento = new BooleanHolder ();

            try
            {
                do
                {
                    // Pido al ConsumerAdmin un evento nuevo.
                    aux = cons_admin_impl.try_pull
                                                (nuevoEvento, indice);

                    // Si no hay ningún evento nuevo duermo durante un
                    // segundo
                    if (nuevoEvento.value == false)
                    {
                        synchronized (sync)
                        {
                            sync.wait (1000);
                        }
                    }
                    // Mientras siga conectado y no haya recibido
                    // ningún evento nuevo, sigo piediéndolo.
                }while (nuevoEvento.value == false && conectado);
            }
            catch(Exception e)
            {}

            // En este punto ya tengo un evento nuevo. Recordemos que
            // hasta que no salga de esta función, el objeto que la ha
            // invocado permanece bloqueado.

            return (aux);
        }
```

```
/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje(tipo, "ProxyPushSupplierImpl", mensaje);
    }
}
```

ProxyPushConsumerImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventComm.*;
import org.omg.CORBA.*;

/**
    Esta clase extiende a _ProxyPushConsumerImplBase implementando
al ProxyPushConsumer quién se encarga de recibir los eventos del
PushSuppliers e introducirlos en el buffer del canal. Además
implementa el interfaz Runnable, lo cual quiere decir que lanza un
hilo de ejecución propio.
*/
public class ProxyPushConsumerImpl
                              extends _ProxyPushConsumerImplBase
                              implements Runnable
{
    private Queue cola;

    private boolean conectado = false ;
    private boolean quieredesconectar=false;

    private PushSupplier push_supplier_ref;
    private SupplierAdminImpl supp_admin_impl;
    private ConsumerAdminImpl cons_admin_impl;

    private Thread thread;

    private EventChannelMonitor monitor;

/**
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita: el tamaño de la cola circular que crea, la referencia del
monitor, la del administrador de productores y la del administrador
de consumidores.

    Aquí se crea la cola circular donde se almacenan temporalmente
los eventos que se reciben del productor hasta que se introducen en
el buffer del sistema.
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita.
*/
    public ProxyPushConsumerImpl (EventChannelParams params)
    {
```

```
        // Obtenemos la referencia al monitor donde enviar los
        // mensajes
        monitor = params.getMonitor();

        //Obtenemos la referencia al administrador de productores.
        supp_admin_impl = params.getSupplierAdminRef();

        //Obtenemos la referencia al administrador de consumidores.
        cons_admin_impl = params.getConsumerAdminRef();

        // Creamos la cola donde se almacenan temporalmente los
        // eventos que recibimos del productor hasta que se
        // introduzcan en el buffer
        cola = new Queue (params.getTamañoColas());

    }

/**
    Método utilizado por el PushSupplier para conectarse al canal.
Es la segunda y última fase de la conexión. Una vez conectado, se
arranca el hilo.
    El Push Supplier puede optar por pasar aquí su referencia o
pasar un null. De esta decisión dependerá que el canal le informe
al destruirse o no.
    Se lanza la excepción AlreadyConnected si al llamar al método
el Push Supplier ya está conectado al canal.
*/
    public void connect_push_supplier (PushSupplier push_supplier)
        throws AlreadyConnected
    {
        if (conectado)
            throw new AlreadyConnected ();

        push_supplier_ref = push_supplier;

        // Arrancamos el hilo
        thread = new Thread (this);
        thread.start();

        conectado = true ;

        // Informamos al monitor de que nos hemos conectado
        // satisfactoriamente.
        monitor.Accion (monitor.CONECTADO_PUSH_SUPPLIER);
    }

/**
    Método invocado por el PushSupplier para introducir un evento
en la cola circular. Si la cola está llena se bloquea al objeto
invocante hasta que quede libre un hueco.
    Se lanza la excepción Disconnected si el Push Supplier no está
conectado al canal.
*/

    public void push (Any data)
        throws Disconnected
    {
        if (conectado == false)
            throw new Disconnected ();
        else
            cola.put (data);
```

```
    }

/**
    Mientras el Proxy permanezca conectado al canal, el hilo saca
eventos de la cola y los va introduciendo en el buffer. Para esto,
invoca el método get de la cola que lo deja  bloqueado mientras no
llegue  un evento nuevo.
    Si se ha recibido la orden de desconexión impide que se sigan
introduciendo eventos en la cola aunque antes de terminar saca
todos los que ya hubiera y los envía al canal..
*/
    public void run ()
    {
        while (conectado)
        {
            Any aux = cola.get();

            if (aux != null)
                cons_admin_impl.put (aux);

            if (cola.vacia() && quieredesconectar)
            {
                supp_admin_impl.ProxyPushDesconecta (this);
                conectado = false;
            }
            thread.yield();
        }
    }

/**
    Método invocado por el PushSupplier (quién sólo transmite una
orden del productor) para desconectarse.
    Si la cola está vacía introduce en ella un elemento nulo para
despertar al hilo que permanece bloqueado en espera de un evento.
*/
    public void disconnect_push_consumer()
    {
        // Con la variable quieredesconectar aviso al hilo que se
        // encarga de leer los eventos de la cola de que cuando se
        // acaben éstos debe desconectar.

        quieredesconectar = true;

        if (cola.vacia())
        {
            // Introduzco un elemento nulo en la cola para
            // despertar al hilo y que se de cuenta de que tiene
            //que terminar.
            cola.put (null);

        }
    }

/**
    Método invocado por el SupplierAdmin para informar al
PushSupplier de que lo ha desconectado. Esto será así cuando se
destruya el canal. Evidentemente esta notificación sólo tendrá
lugar si el PushSupplier pasó su referencia a este Proxy en la
segunda fase
*/
    public void disconnect_push_supplier ()
```

```
    {
        conectado = false;

        try
        {
            if (push_supplier_ref != null)
                push_supplier_ref.disconnect_push_supplier ();
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            // Aquí no hacemos nada, simplemente capturamos una
            // posible excepción para que en caso de que
            // el productor no conteste, el canal pueda seguir
            // destruyendo al resto.
        }
    }
}
```

ProxyPullSupplierImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CORBA.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventComm.*;

/**
    Esta clase extiende a _ProxyPullSupplierImplBase implementando
al ProxyPullSupplier. Se encarga de esperar peticiones del
PullConsumer que pueden ser try_pull o pull. La diferencia radica
en que si no hay en el buffer ningún evento nuevo, la petición tipo
pull deja al Proxy bloqueado hasta que llegue alguno mientras que
la tipo try_pull no lo bloquea.
*/
class ProxyPullSupplierImpl extends _ProxyPullSupplierImplBase
{
    private boolean conectado = false ;

    private PullConsumer pullconsumer;
    private ConsumerAdminImpl cons_admin_impl;

    private EventChannelMonitor monitor;

    // Indice que lleva la cuenta de cuantos eventos llevos ya
    // pedidos. Es un holder para que el ConsumerAdmin pueda
    // actualizarlo en función de los elementos que hayan sido
    // eliminados en el buffer.
    private LongHolder indice = new LongHolder ();
/**
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita: la referencia del monitor y la del administrador de
consumidores.
*/
    public ProxyPullSupplierImpl (EventChannelParams params)
    {
        // Obtenemos la referencia al monitor donde enviar los
        // mensajes
        monitor = params.getMonitor();
```

```
        //Obtenemos la referencia al administrador de consumidores.
        cons_admin_impl = params.getConsumerAdminRef();
    }


/**
    Método utilizado por el PullConsumer para conectarse al canal.
Es la segunda y última fase de la conexión.
    Se lanza la excepción AlreadyConnected si al llamar al método,
el PullConsumer ya está conectado al canal.
*/
    public void connect_pull_consumer (PullConsumer pull_consumer)
        throws AlreadyConnected
    {
        if (conectado)
            throw new AlreadyConnected ();
        else
        {
            pullconsumer = pull_consumer;
            conectado = true;

            monitor.Accion (monitor.CONECTADO_PULL_CONSUMER);
        }

    }

/**
    La invocación de este método deja bloqueado al llamante hasta
que esté disponible un evento nuevo que es lo que devuelve.
    La petición del evento se realiza a través de la función
try_pull del administrador de consumidores. Si ésta indica que no
hay ningún evento nuevo, dormimos durante un segundo y lo volvemos
a intentar.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public Any pull()
        throws Disconnected
    {
        // Objeto utilizado ólamente para dormir un segundo.
        java.lang.Object sync = new java.lang.Object();

        Any aux = null;
        BooleanHolder nuevoEvento = new BooleanHolder ();

        if (conectado == false)
            throw new Disconnected ();

        try
        {
            do
            {
                // Pido al ConsumerAdmin un evento nuevo.
                aux = cons_admin_impl.try_pull
                                            (nuevoEvento, indice);

                // Si no hay ningún evento nuevo duermo durante
                // un segundo
                if (nuevoEvento.value == false)
                {
```

```
                    synchronized (sync)
                    {
                        sync.wait (1000);
                    }
                }
                // Mientras siga conectado y no haya recibido
                // ningún evento nuevo, sigo piediéndolo.
            }while (nuevoEvento.value == false && conectado);
        }
        catch(Exception e)
        {}

        // En este punto ya tengo un evento nuevo. Recordemos que
        // hasta que no salga de esta función, el objeto que la ha
        // invocado permanece bloqueado.

        return (aux);
    }

/**
    Método que devuelve un evento si es que hay alguno nuevo
disponible. Al terminar, el BooleanHolder contendrá un true si
existe dicho evento y un false en caso contrario. Si no hay ningún
evento disponible el objeto Any devuelto esta vacio.
    La petición del evento se realiza a través de la función
try_pull del administrador de consumidores.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public Any try_pull (BooleanHolder nuevoEvento)
         throws Disconnected
    {
        Any any;

        if (conectado == false)
            throw new Disconnected ();

        any = cons_admin_impl.try_pull (nuevoEvento, indice);

        return any;
    }

/**
    Método invocado por el PullConsumer (quién sólo transmite una
orden del consumidor) para desconectarse. Se encarga únicamente de
trasladar al administrador de consumidores esta petición.
*/
    public void disconnect_pull_supplier()
    {
        if (conectado)
        {
            conectado = false ;
            cons_admin_impl.ProxyPullDesconecta (this);
        }
    }

/**
    Método invocado por el administrador de consumidores para
informar al PullConsumer de que lo ha desconectado. Esto será así
cuando se destruya el canal. Su única función es informar de este
hecho al PullConsumer.
```

```
*/
    public void disconnect_pull_consumer ()
    {
        if (conectado)
        {
            conectado = false;
            try
            {
                if (pullconsumer != null)
                    pullconsumer.disconnect_pull_consumer ();
            }
            catch (Exception e)
            {
                // Aquí no hacemos nada, simplemente capturamos una
                // posible excepción para el caso en que en caso de
                // que el consumidor no conteste, el canal pueda
                // seguir destruyendo al resto.
            }
        }
    }
}
```

ProxyPullConsumerImpl.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CosEventChannelAdmin.*;
import org.omg.CORBA.*;
import org.omg.CosEventComm.*;

/**
    Esta clase extiende a _ProxyPullConsumerImplBase implementando
al ProxyPullConsumer quién se encarga de recibir los eventos del
PullSuppliers e introducirlos en el buffer del canal. Además
implementa el interfaz Runnable, lo cual quiere decir que lanza un
hilo de ejecución propio.
*/
class ProxyPullConsumerImpl
                        extends _ProxyPullConsumerImplBase
                        implements Runnable
{
    private boolean conectado = false ;

    private PullSupplier pull_supplier_ref;
    private SupplierAdminImpl supp_admin_impl;
    private ConsumerAdminImpl cons_admin_impl;
    private EventChannelMonitor monitor;

    private Thread thread;
    private Misc varios = new Misc ();

/**
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita: la referencia del monitor, la del administrador de
productores y la del administrador de consumidores.
*/
    public ProxyPullConsumerImpl (EventChannelParams params)
    {
```

```
        // Obtenemos la referencia al monitor donde enviar los
        // mensajes
        monitor = params.getMonitor();

        //Obtenemos la referencia al administrador de productores.
        supp_admin_impl = params.getSupplierAdminRef();

        //Obtenemos la referencia al administrador de consumidores.
        cons_admin_impl = params.getConsumerAdminRef();
    }

/**
    Método utilizado por el PullSupplier para conectarse al canal.
Es la segunda y última fase de la conexión. Una vez conectado, se
arranca el hilo.

    Se lanza la excepción AlreadyConnected si al llamar al método
el PullSupplier ya está conectado al canal.

    Se lanza la excepción BAD_PARAM si se recibe null en lugar de
la referencia del PullSupplier. Debe conocerse dicha referencia
para poder pedirle los eventos.
*/
    public void connect_pull_supplier (PullSupplier pull_supplier)
        throws AlreadyConnected
    {
        if (conectado)
            throw new AlreadyConnected ();

        if (pull_supplier == null)
            throw new BAD_PARAM ("Debe pasarse una referencia al
                                  ProxyPullConsumer al invocar el
                                  metodo connect_push_consumer");

        pull_supplier_ref = pull_supplier;

        monitor.Accion (monitor.CONECTADO_PULL_SUPPLIER);

        conectado = true ;

        //Arrancamos el hilo
        thread = new Thread (this);
        thread.start();
    }

/**
    Mientras el Proxy permanezca conectado al canal, el hilo
realiza peticiones de tipo pull al PullSupplier, estos eventos los
va introduciendo en el Buffer. Es decir, lo que hace es pedir un
evento al PullSupplier, si existe alguno lo envía al Buffer y pide
el siguiente. Si no existe ningún evento nuevo permanece bloqueado
a la espera.
*/
    public void run ()
    {
        Any aux;

        try
        {
            while (conectado)
            {
```

```
                aux = pull_supplier_ref.pull();
                if (conectado) //He estado bloqueado y no sé que ha
                    cons_admin_impl.put (aux); // pasado mientras

            }
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            // Si se ha producido un fallo en la comunicación, doy
            // de baja al consumidor en el canal de eventos.
            if (ex instanceof  org.omg.CORBA.COMM_FAILURE)
            {
                if (conectado)
                    disconnect_pull_consumer ();
            }

            Mensaje (monitor._ERROR
                                ,varios.TraduceSystemException(ex));
        }
        catch (Disconnected disc)
        {
                Mensaje (monitor._ERROR, "Aqui pasa algo raro.");
        }
    }

/**
    Método invocado por el PullSupplier (quién sólo transmite una
orden del productor) para desconectarse. Su única función es
trasladar al administrador de productores esta petición.
*/
    public void disconnect_pull_consumer()
    {
        conectado = false;
        supp_admin_impl.ProxyPullDesconecta (this);
    }

/**
    Método invocado por el SupplierAdmin para informar al
PullSupplier de que lo ha desconectado. Esto será así cuando se
destruya el canal.  Su única función es informar de este hecho al
PullSupplier.
*/
    public void disconnect_pull_supplier()
    {
        conectado = false;
        try
        {
            pull_supplier_ref.disconnect_pull_supplier ();
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            // Aquí no hacemos nada, simplemente capturamos una
            // posible excepción para el caso en que en caso de que
            // el productor no conteste, el canal pueda seguir
            // destruyendo al resto.
        }
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
```

```
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje(tipo, "ProxyPullConsumer", mensaje);
    }
}
```

Buffer.java

```
package org.omg.CosEventChannelAdminImpl;

import java.util.Vector;
import java.util.Date;
import org.omg.CORBA.*;

/**
    Esta clase es el corazón del canal pues es aquí donde se
almacenan los eventos recibidos de los productores y de donde se
extraen para enviarlos a los consumidores. Además establece un
tiempo de vida para los eventos por lo que transcurrido dicho
tiempo, el evento es eliminado.
    El evento es un objeto de tipo Any que, en lugar de almacenarse
tal cual,  se introduce en un objeto tipo Data para poder tener
constancia de su momento de llegada y por lo tanto poder determinar
el momento de su eliminación.
    Implementa el interfaz Runnable, lo cual quiere decir que lanza
un hilo de ejecución propio.
    El tamaño de buffer es ilimitado (o al menos limitado sólo por
la disponibilidad de memoria).
*/

public class Buffer implements Runnable
{
    private Vector datavector = new Vector (1024);
    private Thread thread;
    private boolean buffer_ocupado = false;

    private Data data;

    private EventChannelMonitor monitor;
    private ConsumerAdminImpl consumeradmin;
    private long tiempo_max_de_estancia;

/**
    Constructor de la clase. Recibe como argumento una clase
EventChannelParams donde encontrará todos los parámetros que
necesita: la referencia del monitor, la del administrador de
consumidores y el tiempo de vida de los eventos en segundos.
    Aquí se arranca el hilo.
*/
    public Buffer (EventChannelParams params)
    {
        tiempo_max_de_estancia =
                            params.getTiempoVidaEventos() * 1000;
        monitor    = params.getMonitor();
        consumeradmin = params.getConsumerAdminRef ();

        thread = new Thread (this);
        thread.start();
    }
```

```
/**
    Método que introduce un evento en el buffer.
*/
    public synchronized void put (Any dato)
    {
        while (buffer_ocupado)
        {
            try{wait();}
            catch (InterruptedException e){}
        }

        buffer_ocupado = true ;

        data = new Data (dato);
        datavector.add (data);

        monitor.Accion(monitor.AÑADIDO_ELEMENTO);

        buffer_ocupado = false ;

        notifyAll();
    }

/**
    Método que devuelve un evento del buffer, el que ocupa la
posición indicada por el parámetro entero. Al terminar, se devuelve
el evento (el objeto Any) si es que existe alguno en dicha posición
o un Any vacío en caso contrario.
    En el BooleanHolder se almacena true si existe el evento o
false si no existe.
*/
    public synchronized Any get (int index,
                                 BooleanHolder nuevoEvento)
    {
        Any aux;

        while (buffer_ocupado)
        {
            try{wait();}
            catch (InterruptedException e){}
        }

        buffer_ocupado = true;

        if (index >= datavector.size ())
        {
            nuevoEvento.value = false;
            aux = ORB.init().create_any(); // Any vacío
        }
        else
        {
            data = (Data) datavector.elementAt (index);
            nuevoEvento.value = true;
            aux = data.GetAny();
        }

        buffer_ocupado = false;
        notifyAll ();

        return aux;
```

```
    }

/**
    El hilo se encarga de ver cuando se introdujo el primer
elemento del buffer
    (el más antiguo) y se duerme hasta que llegue la hora de
eliminarlo.
    Una vez eliminado actúa de igual forma con el nuevo elemento
más antiguo.
    Si el buffer queda vacío, espera hasta que llegue algún evento.
*/
    public void run ()
    {
        long aux_time;
        long tiempo;
        Data aux_data;

        while (true)
        {
            try
            {
                synchronized(this)
                {
                    // Si el vector está vacío, espero.
                    while (datavector.size() == 0)
                        wait();
                }
            } catch (InterruptedException e){};

            aux_time = (new Date()).getTime();
            aux_data = (Data) datavector.get(0);
            tiempo =  aux_data.GetTime()
                    + tiempo_max_de_estancia
                    - aux_time;

            if (tiempo > 1000)
            {
                // Duermo hasta que corresponda eliminar el
                // elemento más antiguo.
                try
                {
                    thread.sleep (tiempo);
                } catch (InterruptedException e){}

            }
            EliminaElemento ();
        }
    }

/**
    Una vez eliminado un elemento, aviso al administrador de
consumidores a través de su métod elementoEliminadoEnBuffer () para
que pueda transformar correctamente el índice del Proxy que realice
una petición al índice del Buffer.
*/
    private synchronized void EliminaElemento ()
    {
        while (buffer_ocupado)
        {
            try{wait();}
            catch (InterruptedException e){}
```

```
        }

        buffer_ocupado = true ;

        datavector.remove(0);

        // Aviso al ConsumerAdmin de que se ha eliminado un
        // elemento.
        consumeradmin.elementoEliminadoEnBuffer ();

        monitor.Accion (monitor.ELIMINADO_ELEMENTO);

        buffer_ocupado = false;
        notifyAll ();
    }
}
```

Data.java

```
package org.omg.CosEventChannelAdminImpl;

import java.util.Vector;
import java.util.Date;
import org.omg.CORBA.*;

/**
    Clase auxiliar para el Buffer.
    Se utiliza para almacenar un objeto Any en el Buffer recordando
el momento exacto de su llegada, algo que será vital para
determinar en que momento ha de quedar obsoleto.
*/
public class Data
{
    private long time;
    private Any data;

/**
    Constructor de la clase, realiza una copia del Any que recibe
como argumento y almacena la hora GMT de la construcción.
*/
    public Data (Any dato)
    {
        time = (new Date()).getTime();

        data = org.omg.CORBA.ORB.init().create_any();
        data.insert_any (dato);
    }

/**
    Devuelve la hora GMT en que se construyó este objeto.
*/
    public long GetTime ()
    {
        return time;
    }

/**
    Devuelve el Any que almacena el objeto.
*/
```

```
    public Any GetAny ()
    {
        return data.extract_any();
    }
}
```

Queue.java

```
package org.omg.CosEventChannelAdminImpl;

import java.util.Vector;
import org.omg.CORBA.Any;

/**
    Cola circular de objetos tipo Any. El tamaño de la cola es por
defecto de 64 elementos, pero puede establecerse otro al
construirla.
*/
public class Queue
{
    private int tammax ;
    private int rpos = 0;
    private int apos = 0;
    private Vector cola;

/**
    Constructor que no acepta ningún parámetro, genera una cola
circular de 64 elementos.
*/
    public Queue ()
    {
        // Pongo un elemento más por el espacio en blanco del
        // algoritmo.
        tammax = 65;
        cola = new Vector (tammax);
    }

/**
    Constructor que acepta como parámetro un int que es el tamaño
de esta cola circular.
*/
    public Queue (int tammax_usu)
    {
        // Pongo un elemento más por el espacio en blanco del
        // algoritmo.
        tammax = tammax_usu +1;
        cola = new Vector (tammax);
    }

/**
    Método que devuelve un elemento de la cola (el siguiente al
último que se extrajo). El objeto que invoque este método permanece
bloquedo si la cola está vacía, desbloqueándose cuando se reciba un
nuevo elemento.
*/
    public synchronized Any get()
    {
        if (rpos == tammax) rpos =0;
```

```
        // Mientras la cola esté vacía espero a que se introduzca
        // un elemento.
        while (rpos  == apos)
        {
            try{wait();}
            catch (InterruptedException e){}
        }

        rpos ++;
        notifyAll ();

        return (Any) (cola.get (rpos -1));
    }

/**
    Método que introduce un elemento en la cola. Si está llena, el
método invocante permanece bloqueado hasta que se produce un hueco.
*/
     public synchronized void put (Any dato)
    {
        // Mientras la cola esté llena espero a que se saque un
        // elemento
        while ((apos+1 == rpos)   ||
              ((apos == tammax-1) &&
              (rpos ==0)))
        {
            try{wait();}
            catch (InterruptedException e){}
        }

        // Se pone este if para que no se aumente la capacidad del
        // vector. Con cola.set reemplazo los elementos, pero debe
        // ser apos < size para que no de error. Por eso al
        // principio pongo cola.add.
        if (apos < cola.size ())
            cola.set (apos, dato);
        else
            cola.add (apos, dato);

        apos++;
        if (apos == tammax) apos = 0;

        notifyAll();
    }

/**
    Método que devuelve un boolean true si la cola está llena y
false si no lo está.
*/
    public boolean llena ()
    {
        if ((apos+1 == rpos)   ||
           ((apos == tammax-1) &&
           (rpos ==0)))
            return true;
        else
            return false;
    }

/**
```

```
    Método que devuelve un boolean true si la cola está vacía y
false si no lo está.
*/
    public boolean vacia ()
    {
        if (rpos  == apos)
            return true;
        else
            return false;
    }
}
```

EventChannelParams.java

```
package org.omg.CosEventChannelAdminImpl;

/**
    Clase utilizada para almacenar parámetros necesarios para
    construir el canal de eventos.
    Almacena en ella los siguientes parámetros:
    Nombre del canal de eventos.
    Tipo del canal de eventos.
    Dirección de la máquina donde corre el servidor de nombres.
    Puerto de escucha del servidor de nombres.
    Tamaño de las colas existentes en los Proxies.
    Tiempo de vida de los eventos en el canal.
    Referencia del monitor del canal.
    Referencia del administrador de productores.
    Referencia del administrador de consumidores.
*/

public class EventChannelParams
{
    private String Nombre    = new String ();
    private String Tipo      = new String ();
    private String Direccion = new String ();
    private String Puerto    = new String ();
    private int TamañoColas;
    private int TiempoVidaEventos;
    private ConsumerAdminImpl ConsumerAdminRef;
    private SupplierAdminImpl SupplierAdminRef;
    private EventChannelMonitor Monitor;

/**
    Constructor de la clase. Esta objeto es creado por el objeto
que crea al Servant  y éste lo recibe como parámetro de
construcción. Por esto es necesario establecer desde un principio
todos los parámetros excepto las referencias de los administradores
(que todavía no se han creado).
*/
    public EventChannelParams (String nombre, String tipo,
                               String direccion, String puerto,
                               int tamaño_colas,
                               int t_vida_eventos,
                               EventChannelMonitor monitor)
    {
        Nombre           = nombre;
        Tipo             = tipo;
        Direccion        = direccion;
```

```
        Puerto            = puerto;
        TamañoColas       = tamaño_colas;
        TiempoVidaEventos = t_vida_eventos;
        Monitor           = monitor;
    }

/**
    Constructor idéntico al anterior, la única diferencia es que no
al no recibir la referencia de un monitor, construye un monitor
básico. Puede ser utilizados por aplicaciones que no necesiten
tener control sobre la información de monitorización del canal.
*/
    public EventChannelParams (String nombre, String tipo,
                               String direccion, String puerto,
                               int tamaño_colas,
                               int t_vida_eventos)
    {
        Nombre            = nombre;
        Tipo              = tipo;
        Direccion         = direccion;
        Puerto            = puerto;
        TamañoColas       = tamaño_colas;
        TiempoVidaEventos = t_vida_eventos;
        Monitor           = new BasicMonitor ();
    }

/**
    Guarda la referencia del administrador de productores, es
invocado por el constructor de la clase EventChannelServant.
*/
    public void setSupplierAdminRef
                              (SupplierAdminImpl supplier_admin_ref)
    {
        SupplierAdminRef = supplier_admin_ref;
    }

/**
    Devuelve la referencia al SupplierAdmin.
*/
    public SupplierAdminImpl getSupplierAdminRef ()
    {
        return SupplierAdminRef;
    }

/**
    Guarda la referencia del administrador de consumidores, es
invocado por el constructor de la clase EventChannelServant.
*/
    public void setConsumerAdminRef
                              (ConsumerAdminImpl consumer_admin_ref)
    {
        ConsumerAdminRef = consumer_admin_ref;
    }
/**
    Devuelve la referencia al ConsumerAdmin.
*/
    public ConsumerAdminImpl getConsumerAdminRef ()
    {
        return ConsumerAdminRef;
    }
```

```
/**
    Devuelve el nombre que fue almacenado en construcción.
*/
    public String getNombre ()
    {
        return Nombre;
    }


/**
    Devuelve el tipo que fue almacenado en construcción.
*/
    public String getTipo ()
    {
        return Tipo;
    }


/**
    Devuelve la dirección que fue almacenada en construcción.
*/
    public String getDireccion ()
    {
        return Direccion;
    }


/**
    Devuelve el puerto que fue almacenado en construcción.
*/
    public String getPuerto ()
    {
        return Puerto;
    }


/**
    Devuelve el tamaño de las colas que fue almacenado en
construcción.
*/
    public int getTamañoColas ()
    {
        return TamañoColas;
    }


/**
    Devuelve el tiempo de vida de los eventos que fue almacenado en
construcción.
*/
    public int getTiempoVidaEventos ()
    {
        return TiempoVidaEventos;
    }


/**
    Devuelve la referencia al monitor.
*/
    public EventChannelMonitor getMonitor ()
    {
```

```
        return Monitor;
    }
}
```

EventChannelMonitor.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CORBA.*;

/**
    Interfaz que debe cumplir cualquier clase que se utilice como
monitor del canal de eventos.
*/
public interface EventChannelMonitor
{
    // Tipos de mensaje
    final public int INFO        = 0;
    final public int _ERROR      = 1;
    final public int CONEXION    = 2;
    final public int DESCONEXION = 3;

    // Acciones
    final public int AÑADIDO_ELEMENTO          = 101;
    final public int ELIMINADO_ELEMENTO        = 102;

    final public int CONECTADO_PUSH_SUPPLIER   = 103;
    final public int CONECTADO_PULL_SUPPLIER   = 104;
    final public int CONECTADO_PUSH_CONSUMER   = 105;
    final public int CONECTADO_PULL_CONSUMER   = 106;

    final public int DESCONECTADO_PUSH_SUPPLIER = 107;
    final public int DESCONECTADO_PULL_SUPPLIER = 108;
    final public int DESCONECTADO_PUSH_CONSUMER = 109;
    final public int DESCONECTADO_PULL_CONSUMER = 110;

    final public int CANAL_DESCONECTA          = 111;

    final public int RECIBIDO_PUSH             = 112;
    final public int RECIBIDO_PULL             = 113;
    final public int ENVIADO_PULL              = 114;
    final public int ENVIADO_PUSH              = 115;

/**
    Este método es invocado por todas las clases del sistema para
transmitir al monitor un determinado mensaje, este puede ser de
diversos tipos:  información general, error, indicación de conexión
o de desconexión. Para indicar de que tipo es un mensaje concreto
se utilizan las variables finales de la interfaz:  INFO, _ERROR,
CONEXIÓN y DESCONEXION.
*/
    void Mensaje (int tipo, String Origen, String mensaje);

/**
    Método invocado por todas las clases del sistema para informar
al monitor de que se ha producido una determinada acción tal como
la adición o eliminación de un elemento del buffer, la conexión o
desconexión del canal de algún cliente,  la destrucción del canal,
o el envío o recepción de algún evento. El entero que identifica la
acción es una de las variables finales de la interfaz.
```

```
*/
    void Accion  (int accion);

/**
    Una vez que un Consumer (ya sea de tipo Push o Pull) recibe un
evento puede pasar, naturalmente,. a utilizarlo de la forma que
corresponda a la aplicación concreta. El problema de esto es que
hay que modificar su código en cada caso.
    Para evitar este inconveniente se ha optado por sacar el evento
del propio Consumer y enviarlo al monitor donde será utilizado. Es
en éste método dónde se recibe el evento en el monitor.
*/
    void insert  (Any data);
}
```

BasicMonitor.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CORBA.*;

/**
    Clase que implementa un monitor muy básico, limitándose a
mostrar en la consola los mensajes que recibe. Se ha creado para
proporcionar automáticamente un monitor a las aplicaciones que no
incorporen el suyo propio.
    Hay que destacar que si una aplicación decide no implementar el
interfaz EventChannelMonitor deberá modificar el código de las
funciones push de la clase PushConsumerImpl y las funciones
try_pull y pull de la clase PullConsumerImpl para utilizar los
eventos que reciban ya que, para dotar de mayor generalidad al
canal, estas funciones se limitan a pasar los eventos al monitor.
*/

public class BasicMonitor implements EventChannelMonitor
{

/**
    Muestra el mensaje indicando previamente su origen.
*/
    public synchronized void Mensaje (int tipo, String origen,
                                      String mensaje)
    {
        System.out.println (origen + " :: " + mensaje);
    }

/**
    Muestra el mensaje correspondiente a la acción llevada a cabo.
*/
    public synchronized void Accion (int accion)
    {
        switch (accion)
        {
            case AÑADIDO_ELEMENTO:
                    System.out.println ("Añadido elemento al
                                         buffer.");
                    break;
            case ELIMINADO_ELEMENTO:
                    System.out.println ("Eliminado elemento del
```

```
                                        buffer.");
                    break;
                case CONECTADO_PUSH_SUPPLIER:
                    System.out.println ("Conectado Push Supplier al
                                        canal.");
                    break;
                case CONECTADO_PULL_SUPPLIER:
                    System.out.println ("Conectado Pull Supplier al
                                        canal.");
                    break;
                case CONECTADO_PUSH_CONSUMER:
                    System.out.println ("Conectado Push Consumer al
                                        canal.");
                    break;
                case CONECTADO_PULL_CONSUMER:
                    System.out.println ("Conectado Pull Consumer al
                                        canal.");
                    break;
                case DESCONECTADO_PUSH_SUPPLIER:
                    System.out.println ("Desconectado Push Supplier
                                        del canal.");
                    break;
                case DESCONECTADO_PULL_SUPPLIER:
                    System.out.println ("Desconectado Pull Supplier
                                        del canal.");
                    break;
                case DESCONECTADO_PUSH_CONSUMER:
                    System.out.println ("Desconectado Push Consumer
                                        del canal.");
                    break;
                case DESCONECTADO_PULL_CONSUMER:
                    System.out.println ("Desconectado Pull Consumer
                                        del canal.");
                    break;
                case ENVIADO_PUSH:
                    System.out.println ("Enviado evento tipo push
                                        al canal.");
                    break;
                case RECIBIDO_PUSH:
                    System.out.println ("Recibido evento tipo push
                                        del canal.");
                    break;
                case RECIBIDO_PULL:
                    System.out.println ("Recibido evento tipo pull
                                        del canal.");
                    break;
                case ENVIADO_PULL:
                    System.out.println ("Recibido evento tipo push del
                                         canal.");
                    break;
                case CANAL_DESCONECTA:
                    System.out.println ("El canal ha desconectado a
                                        este objeto.");
                    break;
                default:
                    System.out.println ("Accion desconocida!");
            }
        }

    /**
        Método utilizado para introducir un evento en el monitor.
```

```
    Lo que se haga despues con él dependará de cada caso.
*/
    public void insert (Any data)
    {
        //No puedo hacer nada porque no sé que viene dentro del Any
    }
}
```

Misc.java

```
package org.omg.CosEventChannelAdminImpl;

import org.omg.CORBA.SystemException.*;
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

/**
    Clase auxiliar con métodos utilizados en diversos sitios a lo
largo del proyecto.
    Es una especie de cajón de sastre sin más pretensión que la de
no repetir un mismo método en diversas clases.
*/
public class Misc
{
/**
    Método que genera un panel de error con el mensaje que se le
pase como parámetro.
*/
    public void PopUpError (JPanel panel, String Mensaje)
    {
        JOptionPane.showMessageDialog(   panel,
                                         Mensaje,
                                         "Error",
                                         JOptionPane.ERROR_MESSAGE);
    }

/**
    Método que crea y devuelve un panel compuesto por un JLabel
con el texto label, un JTextField con el texto textField que será
editable o no dependiendo del argumento editable.
    Los parámetros descr y texto son los mensajes que aparecerán al
permanecer el ratón sobre el JLabel y el JTextField
respectivamente.
*/
    public JPanel CreaLabelTexto (String label, String descr,
                                  JTextField textField,
                                  String texto, boolean editable)
    {
        JPanel aux = (new JPanel ()
                        {public Insets getInsets ()
                        {
                            return new Insets (4, 4, 4, 4);}}
                        );
        aux.setLayout (new GridLayout ());

        JLabel jLabel = new JLabel   (label);
        jLabel.setToolTipText (descr);
        aux.add (jLabel, null);
```

```
        textField.setBorder(BorderFactory.createEtchedBorder());
        textField.setEditable (editable);
        textField.setText(texto);
        textField.setToolTipText(descr);

        aux.add (textField, null);

        return aux;
    }

/**
    Método que devuelve el String explicativo de la excepción de
tipo org.omg.CORBA.SystemException que se le pase como parámetro.
Este mensaje depende del tipo de la excepción y de su código minor.
*/
    public String TraduceSystemException
                          (org.omg.CORBA.SystemException excepcion)
    {
        if (excepcion instanceof  org.omg.CORBA.COMM_FAILURE)
        {
            switch (excepcion.minor)
            {
                case 1: return ("Unable to connect to the host and
                                 port specified in the object
                                 reference, or in the object
                                 reference obtained after
                                 location/object forward.");
                case 2: return ("Error occurred while trying to
                                 write to the socket. The socket
                                 has been closed by the other side,
                                 or is aborted.");
                case 3: return ("Error occurred while trying to
                                 write to the socket. The
                                 connection is no longer alive.");
                case 6: return ("Unable to successfully connect to
                                 the server after several
                                 attempts.");

                default: return ( "Recibida la excepcion "
                                +  excepcion.getClass()
                                +" con el código menor "
                                +  excepcion.minor
                                +" desconocido.");
            }
        } else if (excepcion instanceof org.omg.CORBA.BAD_PARAM)
        {
            switch (excepcion.minor)
            {
                case 1: return ("A null parameter was passed to a
                                 Java IDL method.");
                default: return ( "Recibida la excepcion "
                                +  excepcion.getClass()
                                +" con el código menor "
                                +  excepcion.minor
                                +" desconocido.");
            }
        } else if (excepcion instanceof
                                    org.omg.CORBA.DATA_CONVERSION)
        {
            switch (excepcion.minor)
```

```
        {
            case 1: return ("Encountered a bad hexadecimal
                             character while doing ORB
                             string_to_object operation.");
            case 2: return ("The length of the given IOR for
                             string_to_object() is odd. It must
                             be even.");
            case 3: return ("The string given to
                             string_to_object() does not start
                             with IOR: and hence is a bad
                             stringified IOR.");
            case 4: return ("Unable to perform ORB
                             resolve_initial_references
                             operation due to the host or the
                             port being incorrect or
                             unspecified, or the remote host
                             does not support the Java IDL
                             bootstrap protocol.");

            default: return ( "Recibida la excepcion "
                            +  excepcion.getClass()
                            +" con el código menor "
                            +  excepcion.minor
                            +" desconocido.");
        }
    } else if (excepcion instanceof org.omg.CORBA.INTERNAL)
    {
        switch (excepcion.minor)
        {
            case 3: return ("Bad status returned in the IIOP
                             Reply message by the server.");
            case 6: return ("When unmarshaling, the repository
                             id of the user exception was found
                             to be of incorrect length.");
            case 7: return ("Unable to determine local hostname
                             using the Java APIs
                             InetAddress.getLocalHost().
                                              getHostName().");
            case 8: return ("Unable to create the listener
                             thread on the specific port.
                             Either the port is already in use,
                             there was an error creating the
                             daemon thread, or security
                             restrictions prevent listening.");
            case 9: return ("Bad locate reply status found in
                             the IIOP locate reply.");
            case 10: return ("Error encountered while
                              stringifying an object
                              reference.");
            case 11: return ("IIOP message with bad GIOP v1.0
                              message type found.");
            case 14: return ("Error encountered while
                              unmarshaling the user
                              exception.");
            case 18: return ("Internal initialization error.");
            default: return ( "Recibida la excepcion "
                            +  excepcion.getClass()
                            +" con el código menor "
                            +  excepcion.minor
                            +" desconocido.");
        }
```

```
} else if (excepcion instanceof org.omg.CORBA.INV_OBJREF)
{
    switch (excepcion.minor)
    {
        case 1: return ("An IOR with no profile was
                         encountered.");
        default: return ( "Recibida la excepcion "
                        +  excepcion.getClass()
                        +" con el código menor "
                        +  excepcion.minor
                        +" desconocido.");
    }
} else if (excepcion instanceof org.omg.CORBA.MARSHAL)
{
    switch (excepcion.minor)
    {
        case 4: return ("Error occured while unmarshaling
                         an object reference.");
        case 5: return ("Marshalling/unmarshaling
                         unsupported IDL types like wide
                         characters and wide strings.");
        case 6: return ("Character encountered while
                         marshaling or unmarshaling a
                         character or string that is not
                         ISO Latin-1 (8859.1) compliant. It
                         is not in the range of 0 to
                         255.");
        default: return ( "Recibida la excepcion "
                        +  excepcion.getClass()
                        +" con el código menor "
                        +  excepcion.minor
                        +" desconocido.");
    }
} else if (excepcion instanceof org.omg.CORBA.NO_IMPLEMENT)
{
    switch (excepcion.minor)
    {
        case 1: return ("Dynamic Skeleton Interface is not
                         implemented.");
        default: return ( "Recibida la excepcion "
                        +  excepcion.getClass()
                        +" con el código menor "
                        +  excepcion.minor
                        +" desconocido.");
    }
} else if (excepcion instanceof org.omg.CORBA.OBJ_ADAPTER)
{
    switch (excepcion.minor)
    {
        case 1: return ("No object adapter was found
                         matching the one in the object
                         key when dispatching the request
                         on the server side to the object
                         adapter layer.");
        case 2: return ("No object adapter was found
                         matching the one in the object key
                         when dispatching the locate
                         request on the server side to the
                         object adapter layer.");
        case 4: return ("Error occured when trying to
                         connect a servant to the ORB.");
```

```
                default: return ( "Recibida la excepcion "
                                +  excepcion.getClass()
                                +" con el código menor "
                                +  excepcion.minor
                                +" desconocido.");
            }
        } else if (excepcion instanceof
                                    org.omg.CORBA.OBJECT_NOT_EXIST)
        {
            switch (excepcion.minor)
            {
                case 1: return ("Locate request got a response
                                 indicating that the object is not
                                 known to the locator.");
                case 2: return ("Server id of the server that
                                 received the request does not
                                 match the server id baked into the
                                 object key of the object reference
                                 that was invoked upon.");
                case 4: return ("No skeleton was found on the
                                 server side that matches the
                                 contents of the object key inside
                                 the object reference.");
                default: return ( "Recibida la excepcion "
                                +  excepcion.getClass()
                                +" con el código menor "
                                +  excepcion.minor
                                +" desconocido.");
            }
        } else if (excepcion instanceof org.omg.CORBA.UNKNOWN)
        {
            switch (excepcion.minor)
            {
                case 1: return ("Unknown user exception encountered
                                 while unmarshaling: the server
                                 returned a user exception that
                                 does not match any expected by the
                                 client.");
                case 3: return ("Unknown runtime exception thrown
                                 by the server implementation.");
                default: return ( "Recibida la excepcion "
                                +  excepcion.getClass()
                                +" con el código menor "
                                +  excepcion.minor
                                +" desconocido.");
            }
        }

        return ("Excepcion desconocida");
    }
}
```

PAQUETE COSEVENTCOMMIMPL

Paquete CosEventCommImpl

PullConsumerImpl.java

```
package org.omg.CosEventCommImpl;

import java.util.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventChannelAdminImpl.*;
import org.omg.CORBA.*;
import org.omg.CosEventComm.*;
import org.omg.CosNaming.*;

/**
  Esta clase extiende a _PullConsumerImplBase implementando al
PullConsumer. Este consumidor no recibe automáticamente los eventos
del canal sino que tiene que pedirlos. Una vez recibidos, los
eventos se podrían utilizar en esta misma clase, pero para hacerla
lo más general posible, se ha optado por utilizarlos fuera.
*/
public class PullConsumerImpl extends _PullConsumerImplBase
{
    private boolean conectado = false;

    private ProxyPullSupplier proxypullsupplier;
    private EventChannelMonitor monitor;
    private Properties properties_actuales = new Properties();
    private ORB orb = null;

    private Misc varios = new Misc ();
```

```
/**
    El constructor de la clase, no acepta ningún parámetro. El
sistema creará un monitor básico que se limitará a mostrar los
mensajes en la consola.
*/
    public PullConsumerImpl ()
    {
        monitor = new BasicMonitor ();
    }

/**
    Otro constructor de la clase. Acepta una referencia a un
monitor (objeto que implemente la interfaz EventChannelMonitor) en
el que escribir los mensajes.
*/
    public PullConsumerImpl (EventChannelMonitor monitor_ref)
    {
        if (monitor_ref == null)
            monitor = new BasicMonitor ();
        else
            monitor = monitor_ref;
    }

/**
    Método llamado por el consumidor para conectar el Pull Consumer
al canal de eventos.

    Recibe los argumentos:
            Canal: Nombre del canal al que tiene que conectarse.
            Tipo:  Tipo del canal al que tiene que conectarse.
            Direccion: Dirección de la máquina donde corre el
                       servidor de nombres.
            Puerto: Puerto de escucha del servidor de nombres.
            DarRef: True si se quiere que el administrador de
                    consumidores conzoca su referencia

    Se lanza la excepción AlreadyConnected si al llamar al método,
el Pull Consumer ya está conectado al canal.
*/
    public void connect (String Canal, String Tipo,
                         String Direccion, String Puerto,
                         boolean DarRef)
        throws AlreadyConnected
    {
        try
        {
            String sargs []={};

            if (conectado) throw new AlreadyConnected ();

            Properties properties_nuevas = new Properties ();
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);

            if (!properties_nuevas.equals(properties_actuales))
            {
                // Colocamos este if para que no iniciemos el ORB
                // siempre que nos conectemos sino solamente cuando
                // cambien sus propiedades.
```

```
                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);
                orb = ORB.init (sargs, properties_actuales);
            }

            // Buscamos el servidor de nombres
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references ("NameService");

            NameComponent nc = new NameComponent(Canal, Tipo);
            NameComponent path[] = {nc};
            EventChannel eventchannel = EventChannelHelper.narrow
                (NamingContextHelper.narrow(objRef).resolve(path));

            // Primera fase de la conexión
            proxypullsupplier =
              eventchannel.for_consumers().obtain_pull_supplier ();

            // Ya conozco la referencia del ProxyPullSupplier que
            // ha creado el ConsumerAdmin para mí. Ahora realizo la
            // segunda fase de la conexión.
            if (DarRef)
                proxypullsupplier.connect_pull_consumer(this);
            else
                proxypullsupplier.connect_pull_consumer(null);

            Mensaje (monitor.INFO,
                              "Pull Consumer conectado al canal.");

            conectado = true;
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(ex));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR, e.toString());
        }
    }

/**
    Método invocado por el Consumidor cuando quiere desconectar al
PullConsumer del canal.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de invocar este método.
*/

    public void disconnect ()
        throws Disconnected
    {
        if (!conectado)
            throw new Disconnected ();

        proxypullsupplier.disconnect_pull_supplier();

        Mensaje (monitor.INFO,
```

```
            "Pull Consumer desconectado a petición del consumidor.");

        conectado = false;

        try
        {
            this._release();
            this.finalize();
        }
        catch (java.lang.Throwable thr){}
    }

/**
    Método invocado por el Canal de Eventos para informar de su
destrucción. Sólo podrá hacerlo si este objeto proporcionó su
referencia al ProxyPullSupplier en la segunda fase de la conexión.
*/
    public void disconnect_pull_consumer ()
    {
        conectado = false;

        try
        {
            monitor.Accion(monitor.CANAL_DESCONECTA);

            this._release();
            this.finalize();
        }
        catch (java.lang.Throwable thr){}
    }


/**
    Este método realiza una petición de tipo pull al canal de
eventos, por lo tanto queda bloqueado por éste hasta que haya un
evento disponible. Una vez recibido, el evento se envía al monitor
para que sea utilizado allí.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public void pull()
        throws Disconnected
    {
        Any data;

        if (conectado == false)
            throw new Disconnected ();
        try
        {
            data = proxypullsupplier.pull ();

            monitor.insert (data);
        }
        catch (org.omg.CORBA.SystemException se)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(se));

        }

    }
```

```
/**
    Este método realiza una petición de tipo try_pull al canal de
eventos, por lo tanto no queda bloqueado en ningún momento. En caso
de haber algún evento disponible, se envía al monitor para que sea
utilizado allí.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public void try_pull ()
         throws Disconnected
    {

        Any data;

        // El BooleanHolder contendrá un true si existe dicho
        // evento y un false en caso contrario.
        org.omg.CORBA.BooleanHolder hay_dato =
                                new org.omg.CORBA.BooleanHolder ();

        if (conectado == false)
            throw new Disconnected ();
        try
        {
            data = proxypullsupplier.try_pull (hay_dato);

            if (hay_dato.value)
            {
                monitor.insert (data);
            }
            else
                Mensaje (monitor.INFO,
                            "No hay nuevos eventos disponibles.");
        }
        catch (org.omg.CORBA.SystemException se)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(se));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR, e.toString());
        }
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "PullComsumerImpl", mensaje);
    }
}
```

PullSupplierImpl.java

```
package org.omg.CosEventCommImpl;
```

```
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventChannelAdminImpl.*;
import org.omg.CosEventComm.*;
import java.util.*;

/**
    Esta clase extiende a _PullSupplierImplBase implementando al
PullSupplier, que se encarga de introducir los eventos que recibe
en el canal según el modelo pull. En general, los eventos se
podrían producir en esta misma clase pero, para hacerla lo más
general posible, se ha optado por generarlos fuera.
    Dispone de una cola circular donde se almacenan temporalmente
los eventos que se reciben del productor hasta que los pide el
canal.
*/
public class PullSupplierImpl extends _PullSupplierImplBase
{
    // Cola donde se almacenan temporalmente los eventos que
    // recibimos del productor hasta que los pida el canal.
    private Queue cola = new Queue ();

    private ProxyPullConsumer proxypull;
    private boolean conectado = false;

    private EventChannelMonitor monitor;
    private Misc varios = new Misc ();

    private Properties properties_actuales = new Properties();
    private ORB orb = null;

/**
    El constructor de la clase, no acepta ningún parámetro. El
sistema creará un monitor básico que se limitará a mostrar los
mensajes en la consola.
*/
    public PullSupplierImpl ()
    {
        monitor = new BasicMonitor ();
    }

/**
    Otro constructor de la clase. Acepta una referencia a un
monitor (objeto que implementa la interfaz EventChannelMonitor).
*/
    public PullSupplierImpl (EventChannelMonitor monitor_ref)
    {
        if (monitor_ref == null)
            monitor = new BasicMonitor ();
        else
            monitor = monitor_ref;
    }

/**
    Método invocado por el productor para conectar el Pull Supplier
al canal de eventos.
    En la segunda fase de la conexión debe proporcionar
obligatoriamente su referencia al ProxyPullConsumer pues éste la
necesita para pedirle los eventos.
```

```
    Recibe los argumentos:
            Canal: Nombre del canal al que tiene que conectarse.
            Tipo:  Tipo del canal al que tiene que conectarse.
            Direccion: Dirección de la máquina donde corre el
                       servidor de nombres.
            Puerto: Puerto de escucha del servidor de nombres.

    Se lanza la excepción AlreadyConnected si al invocar el método,
el Pull Supplier ya está conectado al canal.
*/
    public void connect (String Canal, String Tipo,
                         String Direccion, String Puerto)
        throws AlreadyConnected
    {
        try
        {
            String sargs []={};

            if (conectado) throw new AlreadyConnected ();

            Properties properties_nuevas = new Properties ();
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);

            if (!properties_nuevas.equals(properties_actuales))
            {
                // Colocamos este if para que no iniciemos el ORB
                // siempre que nos conectemos
                // sino solamente cuando cambien sus propiedades.

                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);
                orb = ORB.init (sargs, properties_actuales);
            }

            // Buscamos el servidor de nombres
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references ("NameService");

            NameComponent nc = new NameComponent(Canal, Tipo);
            NameComponent path[] = {nc};
            EventChannel eventchannel = EventChannelHelper.narrow
                (NamingContextHelper.narrow(objRef).resolve(path));

            // Primera fase de la conexión
            proxypull =
              eventchannel.for_suppliers().obtain_pull_consumer ();

            // Ya conozco la referencia del ProxyPullConsumer que
            // ha creado el SupplierAdmin para mí. Ahora realizo la
            // segunda fase de la conexión.
            proxypull.connect_pull_supplier(this);

            conectado = true;
            Mensaje (monitor.INFO,
                              "Pull Supplier conectado al canal.");
```

```
            }
            catch (org.omg.CORBA.SystemException ex)
            {
                Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(ex));
            }
            catch(Exception e)
            {
                Mensaje (monitor._ERROR, e.toString());
            }
        }

    /**
        Método invocado por el productor cuando quiere desconectar al
    PullSupplier del canal.
        Se lanza la excepción Disconnected si no se han completado las
    dos fases de la conexión antes de invocar este método.
    */
        public void disconnect ()
            throws Disconnected
        {
            if (!conectado)
                throw new Disconnected ();

            proxypull.disconnect_pull_consumer();

            Mensaje (monitor.INFO, "Pull Supplier desconectado a
                                    petición del productor.");

            conectado = false;

            if (!cola.llena())
            {
                Any aux = ORB.init().create_any(); // Lo utilizo para
                cola.put (aux);  // desbloquear al proxy que esta
                                 // esperando en el método push.
            }

            try
            {
                this._release();
                this.finalize();
            }
            catch (java.lang.Throwable thr) {}
        }

    /**
        Método invocado por el Canal de Eventos para informar de su
    destrucción.
    */
        public void disconnect_pull_supplier ()
        {
            conectado = false;

            try
            {
                monitor.Accion(monitor.CANAL_DESCONECTA);

                this.finalize();
            }
            catch (java.lang.Throwable thr) {}
```

```
    }


/**
    La invocación de este método deja bloqueado al llamante hasta
que esté disponible un evento (objeto Any) y se le envíe. Este
evento se saca de la cola donde lo ha introducido el productor.
*/
    public Any pull()
    {
        Any dato = cola.get();

        monitor.Accion(monitor.ENVIADO_PULL);

        return dato;
    }

/**
    Método que devuelve un evento si es que hay alguno disponible.
Al finalizar el método devuelve el objeto Any y el BooleanHolder
contiene true si existe dicho evento y false en caso contrario.
*/
    public Any try_pull (BooleanHolder booleanholder)
    {
        Any dato = ORB.init().create_any();

        if (cola.vacia())
        {
            booleanholder.value = false;
        }
        else
        {
            booleanholder.value = true;
            dato = cola.get();
        }

        monitor.Accion(monitor.ENVIADO_PULL);

        return dato;
    }

/**
    Con este método, el productor introduce un evento en el
PullSupplier, no en el canal. Este evento se almacena en una cola y
conforme el canal los demanda se van sacando de ella.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de invocar este método.
*/

    public void push (Any data)
        throws Disconnected
    {
        if (conectado == false)
            throw new Disconnected ();
        else
            cola.put (data);
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
```

```
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "PullSupplierImpl", mensaje);
    }
}
```

PushConsumerImpl.java

```
package org.omg.CosEventCommImpl;

import org.omg.CORBA.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventChannelAdminImpl.*;
import org.omg.CosNaming.*;
import org.omg.CosEventComm.*;
import java.util.*;

/**
    Esta clase extiende a _PushConsumerImplBase implementando al
PushConsumer. Este consumidor una vez conectado al canal recibe
todos los eventos que lleguen a él . En general, los eventos se
podrían utilizar en esta misma clase pero, para hacerla lo más
general posible, se ha optado por generarlos fuera.
*/
public class PushConsumerImpl extends _PushConsumerImplBase
{
    private boolean conectado = false;

    private ProxyPushSupplier proxypushsupplier;
    private EventChannelMonitor monitor;

    private Properties properties_actuales = new Properties();
    private ORB orb = null;

    private Misc varios = new Misc ();

/**
    El constructor de la clase, no acepta ningún parámetro. El
sistema creará un monitor básico que se limitará a mostrar los
mensajes en la consola.
*/
    public PushConsumerImpl ()
    {
        monitor = new BasicMonitor ();
    }

/**
    Otro constructor de la clase. Acepta una referencia a un
monitor (objeto que implementa la interfaz EventChannelMonitor) en
el que escribir los mensajes.
*/

    public PushConsumerImpl (EventChannelMonitor monitor_ref)
    {
        if (monitor_ref == null)
            monitor = new BasicMonitor ();
        else
```

```
            monitor = monitor_ref;
    }

/**
    Método invocado por el consumidor para conectar el Push
Consumer al canal de eventos. En la segunda fase de la conexión ,
está obligado a entregar su referencia al Proxy ya que en caso
contrario éste no podría enviarle los eventos.

    Recibe los argumentos:
            Canal: Nombre del canal al que tiene que conectarse.
            Tipo:  Tipo del canal al que tiene que conectarse.
            Direccion: Dirección de la máquina donde corre el
                       servidor de nombres.
            Puerto: Puerto de escucha del servidor de nombres.

    Se lanza la excepción AlreadyConnected si al invocar el método,
el Push Consumer ya está conectado al canal.
*/
    public void connect (String Canal, String Tipo,
                         String Direccion, String Puerto)
        throws AlreadyConnected
    {
        try
        {
            String sargs []={};

            if (conectado) throw new AlreadyConnected ();

            Properties properties_nuevas = new Properties ();
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);

            if (!properties_nuevas.equals(properties_actuales))
            {
                // Colocamos este if para que no iniciemos el ORB
                // siempre que nos conectemos sino solamente cuando
                // cambien sus propiedades.

                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
                properties_actuales.put
                         ("org.omg.CORBA.ORBInitialPort", Puerto);
                orb = ORB.init (sargs, properties_actuales);
            }

            // Buscamos el servidor de nombres
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references ("NameService");

            NameComponent nc = new NameComponent(Canal, Tipo);
            NameComponent path[] = {nc};
            EventChannel eventchannel = EventChannelHelper.narrow
                (NamingContextHelper.narrow(objRef).resolve(path));

            // Primera fase de la conexión
            proxypushsupplier =
              eventchannel.for_consumers().obtain_push_supplier ();
```

```
            // Ya conozco la referencia del ProxyPushSupplier que
            // ha creado el ConsumerAdmin para mí.
            // Ahora realizo la segunda fase de la conexión.
            // Debo dar al ProxyPushSupplier mi referencia si no lo
            // hago lanzará la excepcion BAD_PARAM.

            proxypushsupplier.connect_push_consumer
                                (PushConsumerHelper.narrow (this));

            conectado = true;

            Mensaje (monitor.INFO, "Push Consumer conectado al
                                    canal.");

        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(ex));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR, e.toString());
        }
    }

/**
    Método invocado por el Consumidor cuando quiere desconectar al
Push Consumer del canal.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public void disconnect ()
        throws Disconnected
    {
        if (!conectado)
            throw new Disconnected ();

        proxypushsupplier.disconnect_push_supplier();

        Mensaje (monitor.INFO, "Push Consumer desconectado a
                                petición del consumidor.");

        conectado = false;

        try
        {
            this._release();
            this.finalize();
        }
        catch (java.lang.Throwable thr){}
    }

/**
    Método invocado por el Canal de Eventos para informar de su
destrucción.
    Provoca la destrucción de este objeto.
*/
    public void disconnect_push_consumer ()
    {
        conectado = false;
```

```
        try
        {
            monitor.Accion(monitor.CANAL_DESCONECTA);

            this._release();
            this.finalize();
        }
        catch (java.lang.Throwable thr){}
    }


/**
    Este método es invocado por el ProxyPushSupplier para
introducir un evento en el PushConsumer y de aquí es enviado al
consumidor.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de llamar a este método.
*/
    public void push(Any data)
        throws Disconnected
    {
        if (conectado == false)
            throw new Disconnected ();

        monitor.insert (data);
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
        monitor.Mensaje (tipo, "PushComsumerImpl", mensaje);
    }
}
```

PushSupplierImpl.java

```
package org.omg.CosEventCommImpl;

import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosEventChannelAdmin.*;
import org.omg.CosEventComm.*;
import org.omg.CosEventChannelAdminImpl.*;
import java.util.*;

/**
    Esta clase extiende a _PushSupplierImplBase implementando al
PushSupplier, que se encarga de introducir los eventos que recibe
en el canal según el modelo push. En general, los eventos se
podrían producir en esta misma clase pero, para hacerla lo más
general posible, se ha optado por generarlos fuera.
*/
public class PushSupplierImpl extends _PushSupplierImplBase
{
```

```
    private ProxyPushConsumer proxypush;
    private boolean conectado = false;

    private EventChannelMonitor monitor;
    private Misc varios = new Misc ();

    private Properties properties_actuales = new Properties();
    private ORB orb = null;

/**
    Un constructor de la clase, no acepta ningún parámetro. El
sistema creará un monitor básico que se limitará a mostrar los
mensajes en la consola.
*/
    public PushSupplierImpl ()
    {
        monitor = new BasicMonitor ();
    }

/**
    Otro constructor de la clase. Acepta una referencia a un
monitor (objeto que implementa la interfaz EventChannelMonitor).
*/
    public PushSupplierImpl (EventChannelMonitor monitor_ref)
    {
        if (monitor_ref == null)
            monitor = new BasicMonitor ();
        else
            monitor = monitor_ref;
    }

/**
    Método invocado por el productor para conectar el Push Supplier
al canal de eventos.

    Recibe los argumentos:
            Canal: Nombre del canal al que tiene que conectarse.
            Tipo:  Tipo del canal al que tiene que conectarse.
            Direccion: Dirección de la máquina donde corre el
                       servidor de nombres.
            Puerto: Puerto de escucha del servidor de nombres.
            DarRef: True si se quiere que el administrador de
                    consumidores conzoca su referencia

    Se lanza la excepción AlreadyConnected si al invocar el método,
el PushSupplier ya está conectado al canal.
*/
    public void connect (String Canal, String Tipo,
                         String Direccion, String Puerto,
                         boolean DarRef)
        throws AlreadyConnected
    {
        try
        {
            String sargs []={};

            if (conectado) throw new AlreadyConnected ();

            Properties properties_nuevas = new Properties ();
            properties_nuevas.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
```

```
            properties_nuevas.put
                        ("org.omg.CORBA.ORBInitialPort", Puerto);

            if (!properties_nuevas.equals(properties_actuales))
            {
                // Colocamos este if para que no iniciemos el ORB
                // siempre que nos conectemos sino solamente cuando
                // cambien sus propiedades.

                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialHost", Direccion);
                properties_actuales.put
                       ("org.omg.CORBA.ORBInitialPort", Puerto);
                orb = ORB.init (sargs, properties_actuales);
            }

            // Buscamos el servidor de nombres
            org.omg.CORBA.Object objRef =
                    orb.resolve_initial_references ("NameService");

            NameComponent nc = new NameComponent(Canal, Tipo);
            NameComponent path[] = {nc};
            EventChannel eventchannel = EventChannelHelper.narrow
                (NamingContextHelper.narrow(objRef).resolve(path));

            // Primera fase de la conexión
            proxypush =
              eventchannel.for_suppliers().obtain_push_consumer ();

            // Ya conozco la referencia del Proxy Push Consumer que
            // ha creado el SupplierAdmin para mí. Ahora realizo la
            // segunda fase de la conexión.
            if (DarRef)
                proxypush.connect_push_supplier(this);
            else
                proxypush.connect_push_supplier(null);

            conectado = true;
            Mensaje (monitor.INFO, "Push Supplier conectado al
                                    canal.");

        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (monitor._ERROR,
                                varios.TraduceSystemException(ex));
        }
        catch(Exception e)
        {
            Mensaje (monitor._ERROR, e.toString());
        }
    }

/**
    Este método es invocado por el Productor y se encarga
únicamente de introducirlo en el ProxyPushConsumer.
    Se lanza la excepción Disconnected si no se han completado las
dos fases de la conexión antes de invocar este método.
*/
    public void push (org.omg.CORBA.Any dato)
        throws Disconnected
```

```
    {
        if (conectado)
        {
            proxypush.push (dato);
            monitor.Accion (monitor.ENVIADO_PUSH);
        }
        else
            throw new Disconnected ();
    }

/**
    Método invocado por el Productor cuando quiere desconectar al
PushSupplier del canal. Se lanza la excepción Disconnected si no se
han completado las dos fases de la conexión antes de invocar este
método.
*/
    public void disconnect ()
        throws Disconnected
    {
        if (!conectado)
            throw new Disconnected ();

        proxypush.disconnect_push_consumer();

        Mensaje (monitor.INFO, "Push Supplier desconectado a
                                petición del productor.");

        conectado = false;

        try
        {
            this._release();
            this.finalize();
        }
        catch (java.lang.Throwable thr) {}
    }

/**
    Método invocado por el Canal de Eventos para informar de su
destrucción. Sólo podrá hacerlo si se le ha dado la referencia de
este objeto en la segunda fase de la conexión.
*/
    public void disconnect_push_supplier ()
    {
        conectado = false;

        try
        {
            monitor.Accion(monitor.CANAL_DESCONECTA);

            this.finalize();
        }
        catch (java.lang.Throwable thr) {}
    }

/**
    Envía el mensaje al monitor indicando el tipo y origen del
mismo.
*/
    private void Mensaje (int tipo, String mensaje)
    {
```

```
            monitor.Mensaje (tipo, "PushSupplierImpl", mensaje);
        }
    }
```

PAQUETE EVENTCHANNELFACTORY

Paquete EventChannelFactory

EventChannelFactory.java

```
package Applications.EventChannelFactory;

import javax.swing.UIManager;
import java.awt.*;

/**
    Clase que contiene el main de la aplicación y que se encarga de
instanciar la clase PricipalFrame.
*/
public class EventChannelFactory
{
    public EventChannelFactory ()
    {
        PrincipalFrame frame = new PrincipalFrame();

        frame.validate();

        // Centramos la ventana
        Dimension screenSize =
                      Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
```

```
        frame.setLocation((screenSize.width - frameSize.width)/2,
                          (screenSize.height - frameSize.height)/2
                         );

        frame.setVisible(true);
    }

    public static void main(String[] args)
    {
        try
        {
            UIManager.setLookAndFeel
                        (UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
        }
        new EventChannelFactory();
    }
}
```

PrincipalFrame.java

```
package Applications.EventChannelFactory;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.Vector;
/**
    Como su nombre indica, esta clase compone el marco principal de
la aplicación. Este marco posee un menú con tres opciones:

    Crear Canal.    Instancia un objeto de la clase Channel.
    Destruir Canal. Destruye el canal que esté seleccionado en ese
                    momento.
    Salir. Sale de la aplicación destruyendo previamente todos los
           canales que se hubieran creado.

*/
public class PrincipalFrame extends JFrame
{
    private JMenuBar menuBar1 = new JMenuBar();
    private JMenu menu;
    private JMenuItem menuItem;

    private JToolBar toolBar = new JToolBar();
    private JButton jButton  = new JButton();
    private ImageIcon image1;
    private JTabbedPane jTabbedPane = new JTabbedPane();

/**
    Método que se encarga de destruir el canal que esté
seleccionado.
*/
    private void DestruirCanal ()
    {
        Channel c = (Channel) jTabbedPane.getSelectedComponent();

        if (c!= null)
```

```
        {
            c.Destruir ();
            jTabbedPane.remove(c);
        }
        else
            JOptionPane.showMessageDialog
                       (this,
                       "Necesita seleccionar el canal a destruir.",
                       "Error",
                        JOptionPane.ERROR_MESSAGE);
    }

/**
    Método que crea un canal con toda la información necesaria para
el mismo.
*/
    private void CrearCanal ()
    {
        final Channel canal = new Channel ();
        final JDialog dial  = new JDialog (this,
                                           "Datos del nuevo canal",
                                            false);
        dial.getContentPane().add (canal);
        dial.pack ();
        dial.setVisible (true);
        dial.setLocationRelativeTo (this);
        dial.setDefaultCloseOperation(dial.DISPOSE_ON_CLOSE);

        canal.Creado.addActionListener(new ActionListener ()
        {public void actionPerformed (ActionEvent e)
        {
            jTabbedPane.addTab (canal.Nombre.getText(),
                                null,
                                canal,
                                null);
            dial.dispose();
        }});

        canal.Cancelar.addActionListener(new ActionListener ()
        {public void actionPerformed (ActionEvent e)
        {
            dial.dispose();
        }});
    }
/**
    Este método conforma el menú de la aplicación.
*/
    private void CrearMenu ()
    {
        menu = new JMenu();
        menu.setText ("Acciones");

        menuItem = new JMenuItem();
        menuItem.setText("Crear Canal");
        menuItem.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                CrearCanal ();
            }
        });
```

```
            menu.add(menuItem);

            menuItem = new JMenuItem();
            menuItem.setText("Destruir Canal");
            menuItem.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent e)
                {
                    DestruirCanal ();
                }
            });
            menu.add(menuItem);

            menu.addSeparator();

            menuItem = new JMenuItem();
            menuItem.setText("Salir");
            menuItem.addActionListener(new ActionListener()
            {
                public void actionPerformed(ActionEvent e)
                {
                    fileExit_actionPerformed(e);
                }
            });
            menu.add(menuItem);

            menuBar1.add(menu);
            this.setJMenuBar(menuBar1);
        }


    /**
        Método que construye el marco.
    */
        public PrincipalFrame()
        {
            enableEvents(AWTEvent.WINDOW_EVENT_MASK);
            try
            {
                CrearMenu ();

                this.getContentPane().setLayout(new BorderLayout());
                this.setSize(new Dimension(600, 500));
                this.setTitle("Event Channel Factory");
                this.getContentPane().add(jTabbedPane,
                                          BorderLayout.CENTER);

                CrearCanal();
            }
            catch(Exception e)
            {
                System.err.println("ERROR: " + e);
            }
        }

    /**
        Método que se ejecuta al salir de la aplicaión.
    */
        private void fileExit_actionPerformed(ActionEvent e)
        {
            // Destruimos todos los canales
```

```
        for (int i = 0; i < jTabbedPane.getComponentCount(); i ++)
        {
            Channel c = (Channel) jTabbedPane.getComponentAt(i);
            c.Destruir();
        }

        System.exit(0);
    }

    //Modificado para poder salir cuando se cierre el sistema
    protected void processWindowEvent(WindowEvent e)
    {
        super.processWindowEvent(e);
        if(e.getID() == WindowEvent.WINDOW_CLOSING)
        {
            fileExit_actionPerformed(null);
        }
    }
}
```

Channel.java

```
package Applications.EventChannelFactory;

import org.omg.CosEventChannelAdminImpl.*;
import org.omg.CORBA.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
    Clase que  se encarga de crear un canal nuevo y presentar un
entorno gráfico de monitorización del mismo.

    En primer lugar se demandan los datos del nuevo canal a crear.
    Que son:

    Nombre del canal a crear (con el que se dará de alta en el
        servicio de nombrado).
    Tipo del canal a crear (contexto en el servicio de nombrado).
    Dirección IP de la máquina donde corre el servidor de nombres.
    Puerto de escucha del servidor de nombres.
    Vida de los eventos. Tiempo transcurrido el cual los eventos
        son eliminados.
    Tamaño de la cola del ProxyPushConsumer.
*/

public class Channel extends JPanel
                     implements EventChannelMonitor, ActionListener
{
    private JTextArea textArea = new JTextArea ();

    private Misc varios = new Misc();

    private BorderLayout borderLayout1 = new BorderLayout ();

    private JButton Aceptar     = new JButton ("Aceptar");
```

```
    public  JButton Cancelar    = new JButton ("Cancelar");
    public  JButton Creado      = new JButton ("Creado");
    private JButton Monitorizar = new JButton ("Monitorizar");

    public  JTextField Nombre    = new JTextField();
    private JTextField Tipo      = new JTextField();
    private JTextField Direccion = new JTextField();
    private JTextField Puerto    = new JTextField();

    private JTextField TiempoVidaEventos = new JTextField();
    private JTextField TamañoColas       = new JTextField();
    private JTextField NumActualEventos  = new JTextField();
    private JTextField NumTotalEventos   = new JTextField();
    private JTextField NumPushSuppliers  = new JTextField();
    private JTextField NumPullSuppliers  = new JTextField();
    private JTextField NumPushConsumers  = new JTextField();
    private JTextField NumPullConsumers  = new JTextField();

    private JPanel      panel_1_2 = new JPanel ();
    private JSplitPane  panel_2   = new JSplitPane
                                        (JSplitPane.VERTICAL_SPLIT);
    private JPanel      panel_2_1 = (new JPanel ()
                            {public Insets getInsets()
                            {
                                return new Insets (5, 5, 5, 5);}}
                            );
    private JScrollPane panel_2_2 = new JScrollPane();

    private EventChannelServant eventchannelservant;

/**
    Constructor de la clase. No acepta parámetros.
*/
    public Channel ()
    {
        try
        {
            this.setPreferredSize(new Dimension(550, 175));
            this.setLayout(borderLayout1);

            this.add (panel_2, BorderLayout.CENTER);
            this.add (panel_1_2, BorderLayout.SOUTH );

            Aceptar.addActionListener(this);
            panel_1_2.add (Aceptar , null);

            Monitorizar.setVisible (false);
            Monitorizar.addActionListener(this);
            panel_1_2.add (Monitorizar, null);
            panel_1_2.add (Cancelar, null);

            panel_2.add (panel_2_1, JSplitPane.LEFT );
            panel_2.setEnabled(false);

            panel_2_2.getViewport().add (textArea, null);

            panel_2_1.setLayout (new GridLayout(3, 2, 2, 2));

            panel_2_1.add (varios.CreaLabelTexto (
                          "Nombre:", "Nombre del canal.",
                           Nombre, "CANAL" , true), null);
```

```
            panel_2_1.add (varios.CreaLabelTexto (
                          "Tipo:", "Tipo de canal.",
                           Tipo, "", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
                "Dirección:", "Dirección del servidor de nombres.",
                 Direccion, "127.0.0.1", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
             "Puerto:", "Puerto de escucha del servidor de nombres"
              Puerto, "900", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
             "Vida de los eventos:", "Tiempo (en segundos) tras el
                                 cual los eventos son eliminados.",
              TiempoVidaEventos, "60"       , true), null);

            panel_2_1.add (varios.CreaLabelTexto (
             "Tamaño de las colas:", "Tamaño máximo de las colas de
                                      los Proxies.",
              TamañoColas, "32", true), null);
        }
        catch(Exception e)
        {
            varios.PopUpError(this, "Error en clase Channel " + e);
            System.out.println ("Error en clase Channel " + e);
        }
    }
/**
    Añadimos más elementos informativos al panel.

    Número de PushSuppliers conectados al canal.
    Número de PullSuppliers conectados al canal.
    Número de PushConsumers conectados al canal.
    Número de PullConsumers conectados al canal.
    Número total de eventos gestionados por el canal desde su
        creación.
    Número de eventos que almacena actualmente el canal.
*/
    private void AmpliaInformacion ()
    {
        panel_2.setVisible (false);
        panel_2_1.setLayout (new GridLayout(6, 2, 2, 2));

        panel_2_1.add (varios.CreaLabelTexto (
            "Número de eventos:", "Número de eventos actualmente en
                                   el canal.",
             NumActualEventos, "0", false), null);

        panel_2_1.add (varios.CreaLabelTexto (
            "Número total de eventos:", "Número total de eventos
                                     que han pasado por el canal.",
             NumTotalEventos , "0", false), null);

        panel_2_1.add (varios.CreaLabelTexto (
            "Push Suppliers:", "Número de Push Suppliers
                                actualmente conectados al canal.",
             NumPushSuppliers, "0", false), null);

        panel_2_1.add (varios.CreaLabelTexto (
```

```
            "Pull Suppliers:", "Número de Pull Suppliers
                                actualmente conectados al canal.",
             NumPullSuppliers, "0", false), null);

        panel_2_1.add (varios.CreaLabelTexto (
             "Push Consumers:", "Número de Push Consumers
                                actualmente conectados al canal.",
              NumPushConsumers, "0", false), null);

        panel_2_1.add (varios.CreaLabelTexto (
             "Pull Consumers:", "Número de Pull Consumers
                                actualmente conectados al canal.",
              NumPullConsumers, "0", false), null);

        panel_2.add (panel_2_2, JSplitPane.RIGHT );
        panel_2.resetToPreferredSizes();
        panel_2.setVisible(true);
    }

/**
   En este método se controlan los botones pulsados.
*/
    public void actionPerformed(ActionEvent e)
    {
        if (e.getActionCommand().equals("Aceptar"))
        {
            Aceptar (e);
        }
        else if (e.getActionCommand().equals("Monitorizar"))
        {
            panel_2.setVisible (!panel_2.isVisible());
        }
    }

/**
    Método que realmente crea el canal si se ha pulsado "Aceptar".
*/
    private void Aceptar (ActionEvent e)
    {
        if (Nombre.getText().equals (""))
        {
            varios.PopUpError (this, "El canal necesita un
                                      nombre.");
            return;
        }
        try
        {
            int tamaño_colas = String2int (TamañoColas.getText());
            if (tamaño_colas == -1)
            {
                varios.PopUpError (this, "El tamaño de la cola debe
                                          ser un entero no negativo.");
                return;
            }

            int tiempo_vida_eventos = String2int
                                     (TiempoVidaEventos.getText());
            if (tiempo_vida_eventos == -1)
            {
                varios.PopUpError (this, "El tiempo de vida de los
                         eventos debe ser un entero no negativo.");
```

```
                return;
            }

            EventChannelParams params = new EventChannelParams (
                                    Nombre.getText()
                                  , Tipo.getText()
                                  , Direccion.getText()
                                  , Puerto.getText()
                                  , tamaño_colas
                                  , tiempo_vida_eventos
                                  , this);

            eventchannelservant = new EventChannelServant (params);

            String resultado_conexion;

            resultado_conexion = eventchannelservant.Conectar ();
            if (resultado_conexion == null)
            {
                Mensaje (INFO, "", "Canal "
                                  + Nombre.getText()
                                  + " activo.");

                Aceptar.setVisible(false);
                Cancelar.setVisible(false);
                Monitorizar.setVisible(true);

                Nombre.setEditable(false);
                Tipo.setEditable(false);
                Direccion.setEditable(false);
                Puerto.setEditable(false);
                TamañoColas.setEditable(false);
                TiempoVidaEventos.setEditable(false);
                AmpliaInformacion();
                Creado.doClick();
            }
            else
            {
                varios.PopUpError (this, resultado_conexion);
            }
        }
        catch(Exception ex)
        {
            varios.PopUpError (this,"Error en clase Channel "+ex);
            System.out.println ("Error en clase Channel " + ex);
        }
    }
/**
    Metodo que destruye el canal.
*/
    public void Destruir ()
    {
        eventchannelservant.destroy();
    }

/**
    Método que transforma un String representado a un número entero
en un int.
    Difiere del método intValue() de la clase Integer en que
devuelve -1 tanto si el String no representa a un entero como si
éste es negativo.
```

```
*/
    private int String2int (String texto)
    {
        int valor;

        try
        {
            Integer aux = new Integer (texto);
            valor = aux.intValue();
        }
        catch (Exception e){ return -1;}

        if (valor >= 0)
            return valor;
        else
            return -1;
    }

/*
    Ahora viene la parte en la que se implementa el interfaz
Monitor
*/
    public synchronized void Mensaje (int tipo,
                                      String origen,
                                      String mensaje)
    {
        if (tipo == this._ERROR)
            mensaje = "Error en " + origen + ". "+ mensaje;

        textArea.append (mensaje + "\n");
    }

    private int mumeroPushSuppliers = 0;
    private int mumeroPushConsumers = 0;
    private int mumeroPullSuppliers = 0;
    private int mumeroPullConsumers = 0;
    private int numTotalEventos     = 0;
    private int numActualEventos    = 0;

    public void Accion (int accion)
    {
        switch (accion)
        {
            case AÑADIDO_ELEMENTO:
                numTotalEventos++;
                numActualEventos++;
                NumActualEventos.setText(""+numActualEventos);
                NumTotalEventos.setText(""+numTotalEventos);
                break;
            case ELIMINADO_ELEMENTO:
                numActualEventos--;
                NumActualEventos.setText(""+numActualEventos);
                break;
            case CONECTADO_PUSH_SUPPLIER:
                mumeroPushSuppliers ++;
                NumPushSuppliers.setText (""+mumeroPushSuppliers);
                Mensaje (CONEXION, "", "Conexión de un Push
                                        Supplier al canal.");
                break;
            case CONECTADO_PULL_SUPPLIER:
                mumeroPullSuppliers ++;
```

```
                    NumPullSuppliers.setText (""+mumeroPullSuppliers);
                    Mensaje (CONEXION, "", "Conexión de un Pull
                                            Supplier al canal.");
                    break;
                case CONECTADO_PUSH_CONSUMER:
                    mumeroPushConsumers ++;
                    NumPushConsumers.setText (""+mumeroPushConsumers);
                    Mensaje (CONEXION, "", "Conexión de un Push
                                            Consumer al canal.");
                    break;
                case CONECTADO_PULL_CONSUMER:
                    mumeroPullConsumers ++;
                    NumPullConsumers.setText (""+mumeroPullConsumers);
                    Mensaje (CONEXION, "", "Conexión de un Pull
                                            Consumer al canal.");
                    break;
                case DESCONECTADO_PUSH_SUPPLIER:
                    mumeroPushSuppliers --;
                    NumPushSuppliers.setText (""+mumeroPushSuppliers);
                    Mensaje (DESCONEXION, "", "Desconexión de un Push
                                               Supplier del canal.");
                    break;
                case DESCONECTADO_PULL_SUPPLIER:
                    mumeroPullSuppliers --;
                    NumPullSuppliers.setText (""+mumeroPullSuppliers);
                    Mensaje (DESCONEXION, "", "Desconexión de un Pull
                                               Supplier del canal.");
                    break;
                case DESCONECTADO_PUSH_CONSUMER:
                    mumeroPushConsumers --;
                    NumPushConsumers.setText (""+mumeroPushConsumers);
                    Mensaje (DESCONEXION, "", "Desconexión de un Push
                                               Consumer del canal.");
                    break;
                case DESCONECTADO_PULL_CONSUMER:
                    mumeroPullConsumers --;
                    NumPullConsumers.setText (""+mumeroPullConsumers);
                    Mensaje (DESCONEXION, "", "Desconexión de un Pull
                                               Consumer del canal.");
                    break;
                default:
                    Mensaje (_ERROR, "", "Accion desconocida! "
                                          + accion);
            }
        }

        public void insert (Any data)
        {
        }

    }
```

PAQUETE CLIENTSFACTORY

Paquete ClientsFactory

```
package Applications.ClientsFactory;

import javax.swing.UIManager;
import java.awt.*;

/**
    Clase que contiene el main de la aplicación y que se encarga de
instanciar la clase PricipalFrame.
*/
public class ClientsFactory
{
/**
    Constructor de la aplicación.
*/
    public ClientsFactory()
    {
        PrincipalFrame frame = new PrincipalFrame();
        frame.validate();

        //Centrar la ventana
        Dimension screenSize =
                       Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width)/2,
                          (screenSize.height - frameSize.height)/2
                          );
        frame.setVisible(true);
    }
```

```
    public static void main(String[] args)
    {
        try
        {
            UIManager.setLookAndFeel
                       (UIManager.getSystemLookAndFeelClassName());
        }
        catch(Exception e)
        {
        }
        new ClientsFactory();
    }
}
```

PrincipalFrame.java

```
package Applications.ClientsFactory;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
    Como su nombre indica, esta clase compone el marco principal de
la aplicación.
*/
public class PrincipalFrame extends JFrame
                            implements ActionListener
{
    private JMenuBar menuBar1 = new JMenuBar();
    private JMenu menu;
    private JMenuItem menuItem;
    private BorderLayout borderLayout1 = new BorderLayout();
    private JTabbedPane jTabbedPane    = new JTabbedPane();

/**
    Se encarga de ejecutar la acción seleccionada en el menu.
*/
    private void Ejecutar (int orden)
    {
        switch (orden)
        {
            case 0:
                PushSupplierPanel push_supplier_panel =
                                        new PushSupplierPanel ();
                jTabbedPane.addTab ("Push Supplier",
                                    push_supplier_panel);
                jTabbedPane.setSelectedComponent
                                   (push_supplier_panel);
                break;
            case 1:
                PullSupplierPanel pull_supplier_panel =
                                        new PullSupplierPanel ();
                jTabbedPane.addTab ("Pull Supplier",
                                    pull_supplier_panel);
                jTabbedPane.setSelectedComponent
                                  (pull_supplier_panel);
```

```
                break;
            case 2:
                PushConsumerPanel push_consumer_panel =
                                            new PushConsumerPanel ();
                jTabbedPane.addTab ("Push Consumer",
                                     push_consumer_panel);
                jTabbedPane.setSelectedComponent
                                    (push_consumer_panel);
                break;
            case 3:
                PullConsumerPanel pull_consumer_panel =
                                            new PullConsumerPanel ();
                jTabbedPane.addTab ("Pull Consumer",
                                     pull_consumer_panel);
                jTabbedPane.setSelectedComponent
                                    (pull_consumer_panel);
                break;
            case 5:
                Destruir (-1);
                break;
            case 7:
                System.exit (0);

        }
    }

/**
   Se destruye el cliente que se indique en el índice o todos si el
indice es -1.
*/
    public void Destruir (int indice)
    {
        if (indice == -1)
        {
            indice = jTabbedPane.getSelectedIndex();
            if (indice == -1)
            {
                JOptionPane.showMessageDialog(
                  this,
                  "Necesita seleccionar el componente a destruir.",
                  "Error",
                  JOptionPane.ERROR_MESSAGE);
                return;
            }
        }

        if (jTabbedPane.getComponentAt(indice) instanceof
                                                PushSupplierPanel)
        {
            PushSupplierPanel c = (PushSupplierPanel)
                                 jTabbedPane.getComponentAt(indice);
            jTabbedPane.remove(c);
            c.Destruir();
        }
        else if (jTabbedPane.getComponentAt(indice) instanceof
                                                PushConsumerPanel)
        {
            PushConsumerPanel c = (PushConsumerPanel)
                               jTabbedPane.getComponentAt(indice);
            jTabbedPane.remove(c);
            c.Destruir();
```

```
            }
            else if (jTabbedPane.getComponentAt(indice) instanceof
                                                    PullConsumerPanel)
            {
                PullConsumerPanel c = (PullConsumerPanel)
                jTabbedPane.getComponentAt(indice);
                jTabbedPane.remove(c);
                c.Destruir();
            }
            else if (jTabbedPane.getComponentAt(indice) instanceof
                                                    PullSupplierPanel)
            {
                PullSupplierPanel c = (PullSupplierPanel)
                jTabbedPane.getComponentAt(indice);
                jTabbedPane.remove(c);
                c.Destruir();
            }
        }

         String ordenes [] =
         {          "Crear Push Supplier",
                   "Crear Pull Supplier",
                   "Crear Push Consumer",
                   "Crear Pull Consumer",
                   "",
                   "Destruir",
                   "",
                   "Salir"
        };
    /**
        Método que conforma el menú con seis opciones:

        Crear PushSupplier. Crea un PushSupplier e instancia un objeto
            de la clase PushSupplierPanel.
        Crear PullSupplier. Crea un PullSupplier e instancia un objeto
            de la clase PullSupplierPanel.
        Crear PushConsumer. Crea un PushConsumer e instancia un objeto
            de la clase PushConsumerPanel.
        Crear PullConsumer. Crea un PullConsumer e instancia un objeto
            de la clase PullConsumerPanel.
        Destruir. Destruye el cliente que esté seleccionado en ese
            momento.
        Salir. Sale de la aplicación destruyendo previamente todos los
            clientes que se hubieran creado.
    */
        private void CrearMenu ()
        {
            menu = new JMenu();
            menu.setText ("Acciones");

            for (int i =0; i <ordenes.length; i++)
            {
                menuItem = new JMenuItem();
                if (ordenes[i].equals (""))
                    menu.addSeparator();
                else
                {
                    menuItem.setText(ordenes[i]);
                    menuItem.addActionListener(this);
                    menu.add(menuItem);
                }
```

```
        }

        menuBar1.add(menu);
        setJMenuBar(menuBar1);
    }

/**
    Se recibe el evento de que se ha seleccionado una acción en el
menú.
*/
    public void actionPerformed (ActionEvent e)
    {
        for (int i =0; i <ordenes.length; i++)
        {
            if (e.getActionCommand().equals (ordenes [i]))
                Ejecutar (i);
        }
    }

/**
    Constructor de la clase
*/
    public PrincipalFrame()
    {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try
        {
            CrearMenu ();

            getContentPane().setLayout(borderLayout1);
            setSize(new Dimension(450, 450));
            setTitle("Clients Factory");

            getContentPane().add(jTabbedPane, BorderLayout.CENTER);
        }
        catch(Exception e)
        {
            System.err.println("ERROR: " + e);
        }
    }

/**
    Método modificado para poder salir cuando se cierre el sistema
*/
    protected void processWindowEvent(WindowEvent e)
    {
        super.processWindowEvent(e);
        if(e.getID() == WindowEvent.WINDOW_CLOSING)
        {
            // Destruimos todos los componentes

            int num_componentes = jTabbedPane.getComponentCount();
            for (int i = num_componentes-1; i >= 0 ; i --)
            {
                Destruir (i);
            }

            System.exit(0);
        }
    }
}
```

PullConsumerPanel.java

```
package Applications.ClientsFactory;

import org.omg.CosEventCommImpl.*;
import org.omg.CosEventChannelAdminImpl.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import org.omg.CORBA.*;
import java.util.*;

/**
    Clase que  se encarga de conectar el nuevo PullConsumer a un
canal y presentar un entorno gráfico de monitorización del mismo.
*/
public class PullConsumerPanel
                    extends JPanel
                    implements EventChannelMonitor, ActionListener
{
    private PullConsumerImpl pullconsumerimpl =
                                        new PullConsumerImpl (this);
    private int recibidos = 0;
    private Misc varios = new Misc();

    private JTextArea textArea = new JTextArea ();

    private JButton Conectar     = new JButton ("Conectar");
    private JButton Desconectar  = new JButton ("Desconectar");
    private JButton Pull         = new JButton ("Pull");
    private JButton Try_Pull     = new JButton ("Try_Pull");

    private JTextField Canal     = new JTextField();
    private JTextField Tipo      = new JTextField();
    private JTextField Direccion = new JTextField();
    private JTextField Puerto    = new JTextField();
    private JTextField Recibidos = new JTextField();

    private JComboBox comboBox   = new JComboBox ();

    private JSplitPane  panel_2   = new JSplitPane
                                        (JSplitPane.VERTICAL_SPLIT);
    private JPanel      panel_2_1 = (new JPanel ()
                            {public Insets getInsets ()
                            {
                                return new Insets (4, 4, 4, 4);}}
                            );
    private JScrollPane panel_2_2 = new JScrollPane();

/**
    Constructor de la clase
*/
    public PullConsumerPanel()
    {
        try
        {
            this.setPreferredSize(new Dimension(650, 175));
            this.setLayout(new BorderLayout());
            this.add (panel_2, BorderLayout.CENTER);
```

```
panel_2.add (panel_2_2, JSplitPane.RIGHT );
panel_2.add (panel_2_1, JSplitPane.LEFT );
panel_2.setEnabled(false);

panel_2_2.getViewport().add (textArea, null);

panel_2_1.setLayout (new GridLayout(5, 2, 2, 2));

panel_2_1.add (varios.CreaLabelTexto (
    "Canal:", "Nombre del canal al que se va a
               conectar.",
     Canal, "CANAL", true), null);

panel_2_1.add (varios.CreaLabelTexto (
    "Tipo de canal:", "Tipo del canal al que se va a
                       conectar."
   , Tipo, "", true), null);
panel_2_1.add (varios.CreaLabelTexto (
    "Dirección:", "Dirección del servidor de nombres.",
     Direccion, "127.0.0.1", true), null);

panel_2_1.add (varios.CreaLabelTexto (
    "Puerto:", "Puerto de escucha del servidor de
                nombres.",
     Puerto, "900", true), null);

JPanel aux = new JPanel ();
JLabel jLabel = new JLabel   ("Enviar mi referencia:");
jLabel.setToolTipText("Enviar o no la propia
                      referencia al canal de eventos.");
aux.add (jLabel, null);

comboBox.addItem ("Sí");
comboBox.addItem ("No");
comboBox.setToolTipText ("Enviar o no la propia
                      referencia al canal de eventos.");
aux.add (comboBox, null);

panel_2_1.add (aux, null);
panel_2_1.add (varios.CreaLabelTexto (
   "Eventos recibidos:", "Número global de eventos
                          recibidos.",
    Recibidos, "0", false), null);

Conectar.setToolTipText ("Conectar el Push Consumer al
                          canal de eventos.");
Conectar.addActionListener(this);
panel_2_1.add   (Conectar , null);

Desconectar.setToolTipText ("Desconectar el Push
                       Consumer del canal de eventos.");
Desconectar.addActionListener(this);
panel_2_1.add   (Desconectar , null);

Pull.setToolTipText ("Pide un evento al canal
                       bloqueándose hasta recibirlo.");
Pull.addActionListener(this);
panel_2_1.add   (Pull , null);

Try_Pull.setToolTipText ("Pide un evento al canal. No
 se bloquea en caso de no haber eventos disponibles.");
```

```
            Try_Pull.setNextFocusableComponent(Canal);
            Try_Pull.addActionListener(this);
            panel_2_1.add   (Try_Pull , null);
        }
        catch(Exception exx)
        {
            varios.PopUpError (this, "ERROR en constructor de
                                      PullConsumerPanel: " + exx);
            System.out.println ("Error en clase PullConsumerPanel."
                                + exx);
        }
    }
/**
    Método que recibe el evento de botón pulsado y actúa en
consecuencia.
*/
    public void actionPerformed (ActionEvent e)
    {
        try
        {
            if (e.getActionCommand().equals("Conectar"))
            {
                if (comboBox.getSelectedItem().toString().equals
                                                        ("Sí"))
                    pullconsumerimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText(), true);
                else
                    pullconsumerimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText(), false);
            }
            else if (e.getActionCommand().equals("Desconectar"))
            {
                pullconsumerimpl.disconnect();
                recibidos = 0;
                Recibidos.setText("0");
            }
            else if (e.getActionCommand().equals("Pull"))
            {
                pullconsumerimpl.pull ();
            }
            else if (e.getActionCommand().equals("Try_Pull"))
            {
                pullconsumerimpl.try_pull ();
            }

        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (ERROR, "PullConsumerPanel",
                            varios.TraduceSystemException(ex));
        }
        catch(Exception exx)
        {
            Mensaje (ERROR, "PullConsumerPanel", "ERROR al
                                                conectar: " + exx);
        }
    }
```

```
    public void Destruir ()
    {
        try
        {
            pullconsumerimpl.disconnect();
        }
        catch(Exception exx){}
    }

/*
    Ahora viene la parte en la que se implementa el interfaz
EventChannelMonitor
*/
    public void Mensaje (int tipo, String origen, String mensaje)
    {
        if (tipo == this._ERROR)
            mensaje = "Error en " + origen + ". "+ mensaje;

        textArea.append (mensaje + "\n");
    }

    public void Accion (int accion)
    {
        switch (accion)
        {
            case RECIBIDO_PULL:
                recibidos ++;
                Recibidos.setText(""+ recibidos);
                break;
            case CANAL_DESCONECTA:
                Mensaje (INFO, "PullConsumerPanel", "El canal ha
                                desconectado al Pull Consumer.");
                break;
        }
    }

/**
    El evento ya está en el consumidor y por panto se puede hacer
con él lo que sea necesario.
    En éste caso analizamos el contenido y en función de él
componemos el mensaje que va a la pantalla.
*/
    public void insert (Any data)
    {
        String mens ="";

        try
        {
            Vector v    = (Vector)  data.extract_Value ();
            String tipo = (String)  v.elementAt (0);
            Integer num = (Integer) v.elementAt (1);
            Date date   = (Date)    v.elementAt (2);

            if (num.intValue() == 0)
                mens="Evento individual ";
            else
                mens="Evento número "
                    + num.toString()
                    + " de ráfaga ";
```

```
            mens = mens + "tipo " + tipo + ". ";

            mens = mens + date.toString().substring(0, 19);

            Mensaje (INFO, "", mens);

            Accion  (RECIBIDO_PULL);
        }
        catch(Exception ex)
        {
            Mensaje (INFO, "", "Recibido un evento que no nos
                                corresponde.");
        }

    }
}
```

PushConsumerPanel.java

```
package Applications.ClientsFactory;

import org.omg.CosEventCommImpl.*;
import org.omg.CosEventChannelAdminImpl.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.util.*;
import org.omg.CORBA.*;

/**
    Clase que  se encarga de conectar el nuevo PushConsumer a un
canal y presentar un entorno gráfico de monitorización del mismo.
*/
public class PushConsumerPanel
                    extends JPanel
                    implements EventChannelMonitor, ActionListener
{
    private int recibidos = 0;
    private Misc varios = new Misc();
    private PushConsumerImpl pushconsumerimpl =
                                        new PushConsumerImpl (this);

    private JTextArea textArea  = new JTextArea ();

    private JButton Conectar     = new JButton ("Conectar");
    private JButton Desconectar  = new JButton ("Desconectar");

    private JTextField Canal     = new JTextField();
    private JTextField Tipo      = new JTextField();
    private JTextField Direccion = new JTextField();
    private JTextField Puerto    = new JTextField();
    private JTextField Recibidos = new JTextField();

    private JSplitPane  panel_2   = new JSplitPane
                                        (JSplitPane.VERTICAL_SPLIT);
    private JPanel      panel_2_1 = (new JPanel ()
                            {public Insets getInsets ()
                            {
                                return new Insets (4, 4, 4, 4);}}
```

```
                                );
    private JScrollPane panel_2_2 = new JScrollPane();

/**
    Constructor de la clase.
*/
    public PushConsumerPanel()
    {
        try
        {
            this.setPreferredSize(new Dimension(650, 175));
            this.setLayout(new BorderLayout());
            this.add (panel_2, BorderLayout.CENTER);

            panel_2.add (panel_2_2, JSplitPane.RIGHT );
            panel_2.add (panel_2_1, JSplitPane.LEFT );
            panel_2.setEnabled(false);

            panel_2_2.getViewport().add (textArea, null);

            panel_2_1.setLayout (new GridLayout(4, 2, 2, 2));

            panel_2_1.add (varios.CreaLabelTexto (
                "Canal:", "Nombre del canal al que se va a
                           conectar.",
                 Canal, "CANAL", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
                "Tipo de canal:", "Tipo del canal al que se va a
                                   conectar.",
                 Tipo, "", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
                "Dirección:", "Dirección del servidor de nombres.",
                 Direccion, "127.0.0.1", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
                "Puerto:", "Puerto de escucha del servidor de
                            nombres.",
                 Puerto, "900", true), null);

            panel_2_1.add (varios.CreaLabelTexto (
                "Eventos recibidos:", "Número global de eventos
                                       recibidos.",
                 Recibidos, "0", false), null);

            panel_2_1.add (new JPanel ());

            Conectar.setToolTipText ("Conectar el Push Consumer al
                                      canal de eventos.");
            Conectar.addActionListener(this);
            panel_2_1.add   (Conectar , null);

            Desconectar.setToolTipText ("Desconectar el Push
                                         Consumer del canal de eventos.");
            Desconectar.setNextFocusableComponent(Canal);
            Desconectar.addActionListener(this);
            panel_2_1.add   (Desconectar , null);
        }
        catch(Exception exx)
        {
```

```
            varios.PopUpError (this, "ERROR en constructor de
                                      PushConsumerPanel: " + exx);
            System.out.println ("Error en clase PushConsumerPanel."
                               + exx);
        }
    }

/**
    Método que recibe el evento de botón pulsado y actúa en
consecuencia.
*/
    public void actionPerformed (ActionEvent e)
    {
        try
        {
            if (e.getActionCommand().equals("Conectar"))
            {
                pushconsumerimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText());
            }
            else if (e.getActionCommand().equals("Desconectar"))
            {
                pushconsumerimpl.disconnect();
                recibidos = 0;
                Recibidos.setText("0");
            }
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (ERROR, "PullConsumerPanel",
                            varios.TraduceSystemException(ex));
        }
        catch(Exception exx)
        {
            Mensaje (ERROR, "PullConsumerPanel","ERROR al
                                               conectar: " + exx);
        }
    }

    public void Destruir ()
    {
        try
        {
            pushconsumerimpl.disconnect();
        }
        catch(Exception exx){}
    }

/*
    Ahora viene la parte en la que se implementa el interfaz
EventChannelMonitor
*/
    public void Mensaje (int tipo, String origen, String mensaje)
    {
        if (tipo == this._ERROR)
            mensaje = "Error en " + origen + ". "+ mensaje;

        textArea.append (mensaje + "\n");
    }
```

```
    public void Accion (int accion)
    {
        switch (accion)
        {
            case RECIBIDO_PUSH:
                recibidos ++;
                Recibidos.setText(""+ recibidos);
                break;
            case CANAL_DESCONECTA:
                Mensaje (INFO, "PushConsumerPanel", "El canal ha
                                desconectado al Push Consumer.");
                break;
        }
    }

/**
    El evento ya está en el consumidor y por panto se puede hacer
con él lo que sea necesario.
    En éste caso analizamos el contenido y en función de él
componemos el mensaje que va a la pantalla.
*/

    public void insert (Any data)
    {
        String mens ="";

        try
        {
            Vector v    = (Vector)  data.extract_Value ();
            String tipo = (String)  v.elementAt (0);
            Integer num = (Integer) v.elementAt (1);
            Date date   = (Date)    v.elementAt (2);

            if (num.intValue() == 0)
                mens="Evento individual ";
            else
                mens= " Evento número "
                    +   num.toString()
                    + " de ráfaga ";

            mens = mens + "tipo " + tipo + ". ";

            mens = mens + date.toString().substring(0, 19);

            Mensaje (INFO, "", mens);

            Accion  (RECIBIDO_PUSH);
        }
        catch(Exception ex)
        {
            Mensaje (INFO, "", "Recibido un evento que no nos
                                corresponde.");
        }

    }
}
```

PushSupplierPanel.java

```
package Applications.ClientsFactory;

import org.omg.CosEventCommImpl.*;
import org.omg.CosEventChannelAdminImpl.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import org.omg.CORBA.*;
import java.util.*;

/**
    Clase que  se encarga de conectar el nuevo PushSupplier a un
canal y presentar un entorno gráfico de monitorización del mismo.
*/
public class PushSupplierPanel
                    extends JPanel
                    implements EventChannelMonitor, ActionListener
{

    private PushSupplierImpl pushsupplierimpl =
                                        new PushSupplierImpl (this);
    private Any data;
    private int enviados = 0;
    private Misc varios = new Misc();

    private JTextArea textArea  = new JTextArea ();
    private JButton Conectar    = new JButton ("Conectar");
    private JButton Desconectar = new JButton ("Desconectar");
    private JButton Evento      = new JButton ("Evento");
    private JButton Rafaga      = new JButton ("Rafaga");

    private JTextField Canal    = new JTextField();
    private JTextField Tipo     = new JTextField();
    private JTextField Direccion= new JTextField();
    private JTextField Puerto   = new JTextField();
    private JTextField Enviados = new JTextField();

    private JSplitPane  panel_2   = new JSplitPane
                                        (JSplitPane.VERTICAL_SPLIT);
    private JPanel panel_2_1      = (new JPanel ()
                            {public Insets getInsets ()
                            {
                                return new Insets (4, 4, 4, 4);}}
                            );
    private JScrollPane panel_2_2 = new JScrollPane();
    private JComboBox comboBox    = new JComboBox ();

/**
    Constructor de la clase.
*/
    public PushSupplierPanel()
    {
        try
        {
            this.setPreferredSize(new Dimension(650, 175));
            this.setLayout(new BorderLayout());
            this.add (panel_2, BorderLayout.CENTER);
```

```
panel_2.add (panel_2_2, JSplitPane.RIGHT );
panel_2.add (panel_2_1, JSplitPane.LEFT );
panel_2.setEnabled(false);

panel_2_2.getViewport().add (textArea, null);

panel_2_1.setLayout (new GridLayout(5, 2, 2, 2));

panel_2_1.add (varios.CreaLabelTexto (
    "Canal:", "Nombre del canal al que se va a
               conectar.",
     Canal, "CANAL", true), null);

panel_2_1.add (varios.CreaLabelTexto (
     "Tipo de canal:", "Tipo del canal al que se va a
                        conectar.",
      Tipo,"", true), null);

panel_2_1.add (varios.CreaLabelTexto (
     "Dirección:", "Dirección del servidor de nombres."
    , Direccion, "127.0.0.1", true), null);

panel_2_1.add (varios.CreaLabelTexto (
     "Puerto:", "Puerto de escucha del servidor de
                 nombres.",
      Puerto, "900", true), null);

JPanel aux = new JPanel ();
JLabel jLabel = new JLabel   ("Enviar mi referencia:");
jLabel.setToolTipText ("Enviar o no la propia
                        referencia al canal de
                        eventos.");

aux.add (jLabel, null);

comboBox.addItem ("Sí");
comboBox.addItem ("No");
comboBox.setToolTipText ("Enviar o no la propia
                          referencia al canal de
                          eventos.");
aux.add (comboBox, null);

panel_2_1.add (aux, null);
panel_2_1.add (varios.CreaLabelTexto (
    "Eventos enviados:", "Número global de eventos
                          enviados.",
     Enviados, "0", false), null);

Conectar.setToolTipText ("Conectar el Push Supplier al
                          canal de eventos.");
Conectar.addActionListener(this);
panel_2_1.add   (Conectar , null);

Desconectar.setToolTipText ("Desconectar el Push
                             Supplier del canal de
                             eventos.");
Desconectar.addActionListener(this);
panel_2_1.add   (Desconectar , null);

Evento.setToolTipText ("Enviar un evento al canal.");
```

```
            Evento.addActionListener(this);
            panel_2_1.add   (Evento , null);


            Rafaga.setToolTipText("Enviar una ráfaga de 2000
                                  eventos consecutivos al canal.");
            Rafaga.setNextFocusableComponent(Canal);
            Rafaga.addActionListener(this);
            panel_2_1.add (Rafaga , null);
        }
        catch(Exception exx)
        {
            varios.PopUpError (this, "ERROR en constructor de
                                     PushSupplierPanel: " + exx);
            System.out.println ("Error en clase PushSupplierPanel."
                                                              + exx);
        }
    }
/**
    Método que recibe el evento de botón pulsado y actúa en
consecuencia.
*/
    public void actionPerformed (ActionEvent e)
    {
        try
        {
            if (e.getActionCommand().equals("Conectar"))
            {
                if (comboBox.getSelectedItem().toString().equals
                                                          ("Sí"))
                    pushsupplierimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText(), true);
                else
                    pushsupplierimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText(), false);
            }
            else if (e.getActionCommand().equals("Desconectar"))
            {
                pushsupplierimpl.disconnect();
                enviados = 0;
                Enviados.setText("0");
            }
            else if (e.getActionCommand().equals("Evento"))
            {
                Vector v = new Vector ();
                v.addElement(new String ("Push"));
                v.addElement(new Integer (0));
                v.addElement(new Date());

                data = ORB.init().create_any();
                data.insert_Value(v);
                pushsupplierimpl.push (data);
            }
            else if (e.getActionCommand().equals("Rafaga"))
            {
                for (int i=1; i<2001 ; i++)
                {
```

```
                    Vector v = new Vector ();
                    v.addElement(new String ("Push"));
                    v.addElement(new Integer (i));
                    v.addElement(new Date());

                    data = ORB.init().create_any();
                    data.insert_Value(v);
                    pushsupplierimpl.push (data);
                    System.out.println ("Push rafaga " + i);
                }
            }
        }
        catch (org.omg.CORBA.SystemException ex)
        {
            Mensaje (ERROR, "PullConsumerPanel",
                            varios.TraduceSystemException(ex));
        }
        catch(Exception exx)
        {
            Mensaje (ERROR, "PullConsumerPanel", exx.toString());
        }
    }

/**
    Se destruye el PushSupplier.
*/
    public void Destruir ()
    {
        try
        {
            pushsupplierimpl.disconnect();
        }
        catch(Exception exx){}
    }

/*
    Ahora viene la parte en la que se implementa el interfaz
EventChannelMonitor
*/

    public void Mensaje (int tipo, String origen, String mensaje)
    {
        if (tipo == this._ERROR)
            mensaje = "Error en " + origen + ". "+ mensaje;

        textArea.append (mensaje + "\n");
    }

    public void Accion (int accion)
    {
        switch (accion)
        {
            case ENVIADO_PUSH:
                enviados ++;
                Enviados.setText(""+ enviados);
                break;
            case CANAL_DESCONECTA:
                Mensaje (INFO, "PushSupplierPanel", "El canal ha
                                desconectado al push Supplier.");
                break;
        }
```

```
    }

    public void insert (Any data)
    {
    }
}
```

PullSupplierPanel.java

```
package Applications.ClientsFactory;

import org.omg.CosEventCommImpl.*;
import org.omg.CosEventChannelAdminImpl.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import org.omg.CORBA.*;
import java.util.*;

/**
    Clase que  se encarga de conectar el nuevo PullSupplier a un
canal y presentar un entorno gráfico de monitorización del mismo.
*/

public class PullSupplierPanel
                    extends JPanel
                    implements EventChannelMonitor, ActionListener
{
    private PullSupplierImpl pullsupplierimpl =
                                        new PullSupplierImpl (this);
    private Any data;
    private JTextArea textArea = new JTextArea ();
    private int enviados = 0;
    private Misc varios = new Misc();

    private JButton Conectar     = new JButton ("Conectar");
    private JButton Desconectar  = new JButton ("Desconectar");
    private JButton Evento       = new JButton ("Evento");
    private JButton Rafaga       = new JButton ("Rafaga");

    private JTextField Canal     = new JTextField();
    private JTextField Tipo      = new JTextField();
    private JTextField Direccion = new JTextField();
    private JTextField Puerto    = new JTextField();
    private JTextField Enviados  = new JTextField();

    private JSplitPane  panel_2   = new JSplitPane
                                        (JSplitPane.VERTICAL_SPLIT);
    private JPanel      panel_2_1 = (new JPanel ()
                            {public Insets getInsets ()
                            {
                                return new Insets (4, 4, 4, 4);}}
                            );
    private JScrollPane panel_2_2 = new JScrollPane();

/**
    Constructor de la clase.
*/
```

```
public PullSupplierPanel()
{
    try
    {
        this.setPreferredSize(new Dimension(650, 175));
        this.setLayout(new BorderLayout());
        this.add (panel_2, BorderLayout.CENTER);

        panel_2.add (panel_2_2, JSplitPane.RIGHT );
        panel_2.add (panel_2_1, JSplitPane.LEFT );
        panel_2.setEnabled(false);

        panel_2_2.getViewport().add (textArea, null);

        panel_2_1.setLayout (new GridLayout(5, 2, 2, 2));

        panel_2_1.add (varios.CreaLabelTexto (
            "Canal:", "Nombre del canal al que se va a
                       conectar.",
             Canal, "CANAL", true), null);

        panel_2_1.add (varios.CreaLabelTexto (
             "Tipo de canal:", "Tipo del canal al que se va a
                                conectar.",
             Tipo, "", true), null);

        panel_2_1.add (varios.CreaLabelTexto (
             "Dirección:", "Dirección del servidor de nombres."
            , Direccion, "127.0.0.1", true), null);

        panel_2_1.add (varios.CreaLabelTexto (
            "Puerto:", "Puerto de escucha del servidor de
                        nombres.",
             Puerto, "900", true), null);

        panel_2_1.add (varios.CreaLabelTexto (
             "Eventos enviados:", "Número global de eventos
                                   enviados.",
              Enviados, "0", false), null);

        panel_2_1.add (new JPanel ());

        Conectar.setToolTipText ("Conectar el Pull Supplier al
                                  canal de eventos.");
        Conectar.addActionListener(this);
        panel_2_1.add   (Conectar , null);

        Desconectar.setToolTipText ("Desconectar el Pull
                                     Supplier del canal de
                                     eventos.");
        Desconectar.addActionListener(this);
        panel_2_1.add   (Desconectar , null);

        Evento.setToolTipText ("Generar un evento para el
                                canal.");
        Evento.addActionListener(this);
        panel_2_1.add   (Evento , null);

        Rafaga.setToolTipText ("Generar una ráfaga de 2000
                                eventos consecutivos para el
                                canal.");
```

```
            Rafaga.setNextFocusableComponent(Canal);
            Rafaga.addActionListener(this);
            panel_2_1.add (Rafaga , null);
        }
        catch(Exception exx)
        {
            varios.PopUpError (this, "ERROR en constructor de
                                    PullSupplierPanel: " + exx);
            System.out.println ("Error en clase PullSupplierPanel."
                                                            + exx);
        }
    }

/**
    Método que recibe el evento de botón pulsado y actúa en
consecuencia.
*/
    public void actionPerformed (ActionEvent e)
    {
        try
        {
            if (e.getActionCommand().equals("Conectar"))
            {
                pullsupplierimpl.connect (
                                Canal.getText(), Tipo.getText(),
                                Direccion.getText(),
                                Puerto.getText());
            }
            else if (e.getActionCommand().equals("Desconectar"))
            {
                pullsupplierimpl.disconnect();
                enviados = 0;
                Enviados.setText("0");
            }
            else if (e.getActionCommand().equals("Evento"))
            {
                Vector v = new Vector ();
                v.addElement(new String ("Pull"));
                v.addElement(new Integer (0));
                v.addElement(new Date());

                data = ORB.init().create_any();
                data.insert_Value(v);
                pullsupplierimpl.push (data);
            }
            else if (e.getActionCommand().equals("Rafaga"))
            {
                for (int i=1; i<2001 ; i++)
                {
                    Vector v = new Vector ();
                    v.addElement(new String ("Pull"));
                    v.addElement(new Integer (i));
                    v.addElement(new Date());

                    data = ORB.init().create_any();
                    data.insert_Value(v);
                    pullsupplierimpl.push (data);
                    System.out.println ("Pull rafaga " + i);
                }
            }
        }
```

```
            catch (org.omg.CORBA.SystemException ex)
            {
                Mensaje (ERROR, "PullConsumerPanel",
                                varios.TraduceSystemException(ex));
            }
            catch(Exception exx)
            {
                Mensaje (ERROR, "PullConsumerPanel", exx.toString());
            }
        }

    /**
        Se destruye el PullSupplier.
    */
        public void Destruir ()
        {
            try
            {
                pullsupplierimpl.disconnect();
            }
            catch(Exception exx){}
        }

    /*
        Ahora viene la parte en la que se implementa el interfaz
    EventChannelMonitor
    */
        public void Mensaje (int tipo, String origen, String mensaje)
        {
            if (tipo == this._ERROR)
                mensaje = "Error en " + origen + ". "+ mensaje;

            textArea.append (mensaje + "\n");
        }

        public void Accion (int accion)
        {
            switch (accion)
            {
                case ENVIADO_PULL:
                    enviados ++;
                    Enviados.setText(""+ enviados);
                    break;
                case CANAL_DESCONECTA:
                    Mensaje (INFO, "PullSupplierPanel", "El canal ha
                                    desconectado al Pull Supplier.");
                    break;
            }
        }

        public void insert (Any data)
        {
        }
    }
```