

JXTA (Juxtapose)

Fernando Mosteiro del Pilar

Computación Distribuída - 2019/2020

Universidad de Santiago de Compostela

Índice

Introducción

Material y métodos

Diseño

Conclusiones

Bibliografía

Introducción

Introducción



El Project JXTA permite construir y desplegar soluciones P2P de una manera más eficiente, definiendo una serie de protocolos basados en XML, soportando aplicaciones y servicios en redes *peer-to-peer*.

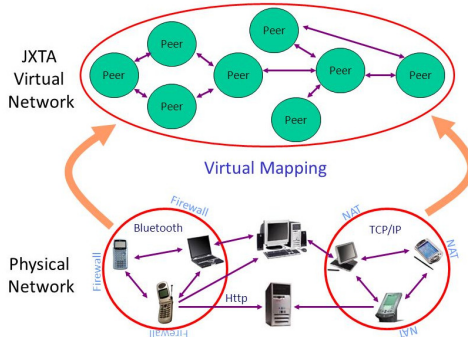
Es un proyecto *open source* creada por Sun Microsystems en el año 2001, responsables de la creación del lenguaje de programación Java, y adquiridos posteriormente por Oracle en 2010.

No está asociado a ningún lenguaje de programación, puede ser fácilmente portado, aunque la implementación en Java es la más madura.

Introducción

JXTA crea una **red virtual** que permite a los nodos en la red física interactuar entre sí, aun cuando algunos de ellos estén detrás de firewalls, NATs o usen distintos transportes de red.

Así se reduce la complejidad de la red y del entorno, y permite una interacción más social y cooperativa. Además, cada nodo es identificado por un ID único, permitiendo que los pares puedan cambiar su dirección pero conservar su número de identificación.



Sobre P2P

(Breve resumen recordatorio sobre las redes *peer-to-peer*, dado que JXTA es capaz de explotar en su totalidad sus usos y aplicaciones)

Las redes P2P son una topología de dispositivos en las que todos o algunos de sus aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí, permitiendo el intercambio directo de información.

Dichas topologías son usadas en una gran variedad de aplicaciones: para compartir ficheros, compartir recursos computacionales o de almacenamiento, mensajería instantánea, trabajo colaborativo, videojuegos online, sistemas de gestión de contenido...

Esta tecnología no es nueva o tremendamente específica, ni está destinada a ser utilizada en un modelo de negocio. Su propósito es el de eliminar servidores y estructuras centralizadas, **facilitar una conexión más directa para los dispositivos, y permitir una computación más colaborativa y social.**

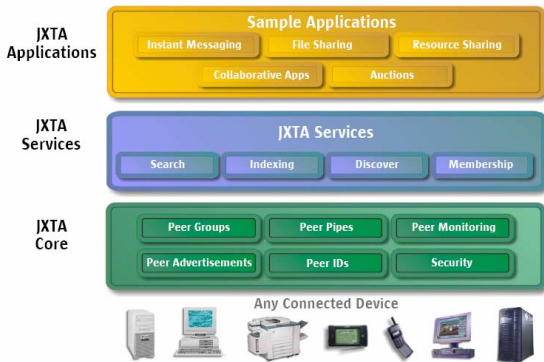
Aspectos clave

Hay una serie de características de JXTA, que conforman la esencia de la plataforma y la distinguen del resto:

- ▶ Representación uniforme de los recursos con documentos XML (anuncios) para describir a los componentes de la red.
- ▶ Canales de comunicación virtuales, la abstracción en nodos y enlaces o canales, sin estar soportada en un sistema central de resolución y direcciones.
- ▶ Uso de identificadores únicos por nodo.
- ▶ Una infraestructura de búsqueda descentralizada basada en tablas hash distribuidas, para la indexación de recursos.

Arquitectura

La arquitectura software de JXTA se divide en tres capas:



A continuación se analizan en detalle los componentes de la capa *core* y la parte de los servicios respecto al grupo de *peers*.

Conceptos y terminología

Existe una serie de términos que describen los componentes principales de JXTA:

- ▶ **JXTA Core**
 - ▶ Peers
 - ▶ Grupos de peers
 - ▶ Anuncios
 - ▶ Canales
 - ▶ Mensajes
 - ▶ IDs
- ▶ **JXTA Services**

Estos componentes se definen debido a su importancia a la hora de implementar un ejemplo simple. Existen más elementos que conforman la plataforma, pero corresponden a un nivel más avanzado.

Peers (Nodos)

Un *peer* o nodo es cualquier entidad dentro de la red que haga uso de los protocolos JXTA; pueden residir en todo tipo de dispositivos, desde móviles hasta superordenadores. Cada uno de ellos actúa de manera independiente y asíncrona, y está identificado por un ID único.

Los nodos publican sus direcciones de red usando los protocolos JXTA. Cada una de ellas representa un extremo o contacto de un nodo (*peer endpoint*), que es usado para establecer conexiones directas entre 2 nodos.

Los nodos son configurados para encontrar a otros en la red, formar grupos y canales transitorios o permanentes por los que comunicarse.

Las conexiones directas no siempre son posibles, por lo que deben de intervenir nodos intermediarios que puedan enrutar los mensajes a los *peers* separados por límites físicos en la red o en su configuración.

A continuación se analizan los tipos de nodos en la red JXTA, aunque no fue considerado oportuno profundizar en detalle debido a la naturaleza de la asignatura.

Peers (Nodos)

Cada nodo tiene un rol bien definido en el modelo inter pares de JXTA.

- ▶ **Super-peers**

- ▶ **Rendezvous:** Tiene la tarea especial de coordinar los nodos en la red JXTA, provee de las bases necesarias para la propagación de mensajes.

Tienen mecanismos optimizados de enrutamiento apoyados por los *edge-peers*, son eficaces debido a que mantienen una lista de nodos poco consistente, pero en cuanto aumente la tasa de fallos dicha lista es comprobada.

- ▶ **Relay-peers:** Permite que pares que estén detrás de cortafuegos o sistemas NAT tomen parte en la red JXTA, usando protocolos que pueden atravesar *firewalls*.

- ▶ **Edge-peers:** Nodos con bajo ancho de banda, escondidos detrás de cortafuegos o acceden a la red a través de conexiones no dedicadas.

(Existen más tipos de nodos, pero no los analizaremos)

Grupos de *peers*

La red virtual JXTA organiza a los nodos en grupos, donde comparten intereses y servicios.

Un nodo puede pertenecer a múltiples grupos, por defecto al ser instanciados pertenecen al Network Peer Group y a partir de ahí pueden unirse a más.

¿Por qué usar grupos de nodos?

- ▶ Crear un entorno seguro y protegido, controlando políticas de seguridad específicas. Desde un log-in con usuario y contraseña, hasta el uso de criptografía con clave pública al grupo.
- ▶ Crear un entorno delimitado, organizando nodos especializados. Se subdivide la red en regiones abstractas, limitando las búsquedas por parte de sus nodos. Por ejemplo: red de nodos colaborando o compartiendo documentos, compartiendo CPU...

Anuncios (*advertisements*)

Un **anuncio** describe todos los recursos en la red (nodos, grupos, canales, servicios, etc.) en un documento XML (neutral en cuanto al lenguaje de programación). La comunicación en JXTA puede ser tomada como el intercambio de uno o más anuncios a través de la red. Todas las operaciones de resolución (DNS, directorios, sockets...) se resumen en búsquedas de anuncios.

Los protocolos JXTA usan los anuncios para definir y publicar la existencia de los recursos. Los peers son capaces de transmitirlos, publicarlos y/o guardarlos en cache.

Existen múltiples tipos de anuncios, dependiendo de lo que "publicitan": *Peer Advertisement*, *Peer Group Advertisement*, *Pipe Advertisement*...

Pipes (Canales) y mensajes

Los *pipes* son canales de comunicación virtual, capaces de conectar nodos sin un enlace físico, proporcionando abstracción en la comunicación. Los *peers* los usan para enviar o recibir mensajes y cualquier tipo de datos: XML, texto HTML, imagenes, musica, codigo binario, objetos Java...

Son unidireccionales, por lo que los dos extremos de un canal son definidos como: el capaz de recibir (*input pipe*) o el que envía (*output pipe*). Representan y se corresponden con las interfaces de otros nodos en la red, por ejemplo la dirección IP y el puerto TCP.

Asociados de manera dinámica, a uno o más destinatarios. *Point-to-point* (entre un output y un input) o propagados (un output a múltiples inputs). Dentro de la red JXTA, están limitados al mismo grupo de nodos.

Los **mensajes** son la unidad básica de intercambio de información entre nodos. Tras haber establecido los canales correspondientes, JXTA usa los servicios *Endpoint Service* y el *Pipe Service* para enviarlos.

Uso de ficheros XML

Los componentes de JXTA "promocionan" o "publican" mensajes representados por documentos XML.

Dichos anuncios permiten a los nodos de la red a encontrar recursos y servicios, publicar los propios, y a determinar como conectarse e interactuar con otros nodos.

Estos intercambios son asíncronos, basados en un modelo de consulta/respuesta. Una comunicación basada en ficheros de un formato específico, **permite que sean interpretados por cualquier lenguaje de programación** y que sean transmitidos a través de multitud de redes.



Seguridad en JXTA

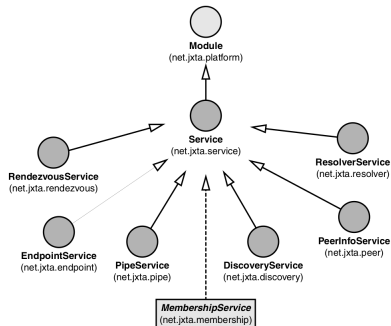
Aunque no sea su principal objetivo, los protocolos de JXTA incluyen unas cuantas medidas de seguridad, a la hora de enviar mensajes entre nodos de un grupo limitado.

- ▶ Se implementa *Transport Layer Security* (TLS) en los enlaces entre peers, usando criptografía asimétrica para autentificar a la contraparte con quien se están comunicando.
- ▶ Se incluye una librería de criptografía (simple).
- ▶ Cada peer tiene su certificado raíz, clave pública de certificación en los anuncios, *framework* de autentificación, esquema de logueo con contraseñas...

Servicios

JXTA incluye servicios de red que puede que no sean estrictamente necesarios para que una red P2P funcione correctamente, pero son convenientes en un entorno de este estilo. Ofrecen un conjunto de funciones que un *peer* proveedor ofrece y publica, usando las *pipes*. Los nodos se comunican buscando, encontrando e invocando a estos servicios de red.

La mayoría de servicios son en relación al grupo de *peers*, donde colaboran entre ellos y, aunque algún nodo falle, el servicio sigue estando disponible.



Servicios

El siguiente listado define el grupo de servicios esenciales que **cada nodo debe implementar** para participar en la red JXTA:

Endpoint Service Enviar y recibir mensajes entre nodos.

Resolver Service Enviar consultas a nodos para obtener información.

Además los servicios que comunmente son implementados **para cada grupo de nodos**:

Discovery Service Buscar recursos del grupo: otros nodos, grupos, canales o servicios.

Membership Service Establecerse dentro de un grupo de nodos.

Pipe Service Crear y gestionar los canales entre nodos del grupo.

Access Service

Monitoring Service

Futuro de JXTA

La mayoría de la documentación en la red es antigua, **muy antigua**. Lo más reciente que se encuentra habla de las intenciones de los colaboradores en el proyecto, que consisten en:

- ▶ Mejorar la escalabilidad vertical (optimización de los recursos de los dispositivos) y horizontal (optimización de los servicios en redes a gran escala).
- ▶ Nuevos servicios, derechos digitales de autor, sistemas de gestión de contenidos, integración con servicios Web ...
- ▶ Estandarización y definición de un estándar, con ayuda de organizaciones públicas.
- ▶ Intenciones de formar una comunidad y conseguir momentum: más miembros, proyectos, foros de discusión, integrar tecnologías...

Pero la gran mayoría de los enlaces que aparecen en la documentación están caídos o referencian a Wayback Machine (web para consultar la historia o modificaciones de las páginas a través del tiempo).

Futuro de JXTA

Digamos que JXTA no está abandonado oficialmente, hay colaboradores que "parchean" las últimas versiones oficiales.

Tras la compra de Sun Microsystems por parte de Oracle, el proyecto quedó en un limbo tras no establecer ninguna administración. Y para continuar era necesario "limpiar" el código base por su antigüedad, pero no hubo suficientes voluntarios.

Las últimas noticias sobre colaboradores voluntarios haciendo *forks* del proyecto datan de 2013. Por lo que al investigar la tecnología nos encontraremos con muchos mensajes como el siguiente:



We're sorry the java.net site has closed.

Most Open Source projects previously hosted on java.net have been relocated. Please contact the corresponding project administrator for relocation information.

For Java related projects: <http://www.oracle.com/technetwork/java/index.html>

Material y métodos

Material y métodos

Binarios y .jar

Como se ha mencionado anteriormente, la documentación que se encuentra en la red es antigua, por lo que la mayoría de los enlaces de descarga están caídos. Incluso muchos mirrors que han sido creados a causa de este problema también presentan dificultades.

Además, en caso de encontrar los binarios correctos y de la versión deseada, aún necesitamos las dependencias correspondientes, lo que provoca nuevas dificultades.

Las librerías JAR de JXTA y de sus dependencias pueden descargarse desde el siguiente enlace.

Maven

Es la opción más recomendable, sobre todo si está integrado en el IDE en el que trabajamos (en mi caso IntelliJ de JetBrains).

Material y métodos

Maven

La gran ventaja de utilizar Maven es que la descarga de las librerías y sus dependencias es automática, por lo que tenemos todo lo necesario para un correcto funcionamiento de la manera más sencilla.

Si el proyecto está basado en Maven, basta con añadir la dependencia en la lista, dentro del fichero POM.

La última versión oficial de JXTA es la 2.7, pero algunos programas en versiones anteriores rompen al actualizar. Además, la 2.5 es la última versión que funcionaba sin problema desde el IDE IntelliJ con Maven.

Home » [net.jxta](#) » [jxta-jxse](#) » 2.5



JXTA Platform » 2.5

JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports.

Organization	Project JXTA
HomePage	https://jxta-jxse.dev.java.net/
Date	(May 20, 2010)
Files	pom (8 KB) jar (1.7 MB) View All
Repositories	Central Spring Lib M Spring Plugins
Used By	3 artifacts

[Maven](#) [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- https://mvnrepository.com/artifact/net.jxta/jxta-jxse -->
<dependency>
  <groupId>net.jxta</groupId>
  <artifactId>jxta-jxse</artifactId>
  <version>2.5</version>
</dependency>
```

☒ Include comment with link to declaration

Compile Dependencies (3)

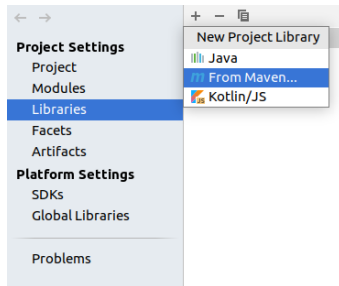
Category/License	Group / Artifact	Version	Updates
BouncyCastle	bouncycastle » bcprov-jdk15	136	1.46
Java Spec CDDL GPL 2.0	javax.servlet » servlet-api	2.3	4.0.1
Apache 2.0	jetty » jetty	4.2.27	6.1.26

Material y métodos

La opción anterior es recomendable para un proyecto completo, con un gran número de dependencias y librerías.

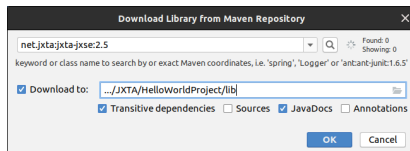
En nuestro caso, al ser un proyecto simple y de ejemplo, usaremos una opción del IDE IntelliJ que permite descargar librerías de Maven, dependencias y documentación al directorio del proyecto, y las incorpora en la ejecución a través de la opción *classpath*.

Primero iremos a **Project Settings** -> **Libraries**, hacemos click en el icono de **New Project Library** y seleccionaremos **From Maven...**

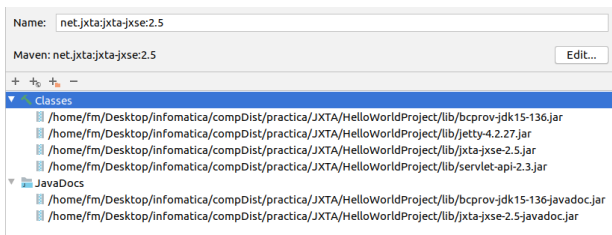


Material y métodos

Se abre la siguiente ventana, donde insertamos las coordenadas Maven correspondientes. Seleccionamos el directorio donde queremos que se descarguen las librerías. Y además marcamos las opciones **Transitive dependencies** y **JavaDocs**.



Tras la descarga vemos en el listado de librerías del proyecto `jxta-jxse-2.5.jar` y sus dependencias.

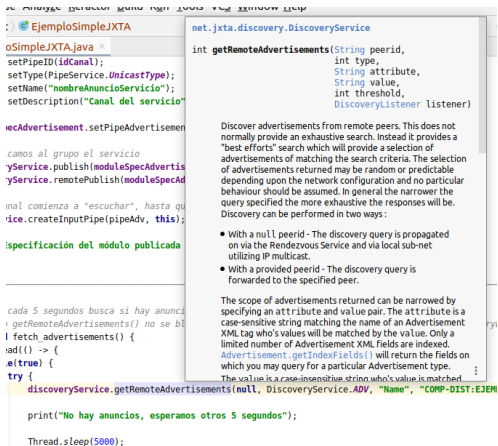


Material y métodos

Con esto conseguimos todo lo necesario para comenzar a trabajar con JXTA.

Además, como hemos marcado la opción Javadocs a la hora de descargar las librerías, podemos acceder a toda la documentación del paquete net.jxta, que es de mucha ayuda debido a los pocos recursos que existen en la web.

(Ctrl + Q sobre un método u objeto para mostrar su documentación)



The screenshot shows an IDE with two panels. The left panel displays the source code of `SimpleJXTA.java`, which includes comments in Spanish and Java code for setting up a JXTA service. The right panel shows the Javadoc for the `net.jxta.discovery.DiscoveryService` class, specifically the `getRemoteAdvertisements` method. The Javadoc includes a detailed description of the method's purpose, two bullet points explaining different search criteria (null peerid vs. provided peerid), and the scope of the returned advertisements.

```
SimpleJXTA.java
setPipeID(idCanal);
setType(PipeService.UnicastType);
setName("nombreAnuncioServicio");
setDescription("Canal del servicio");

recAdvertisement.setPipeAdvertisement(
//camos al grupo el servicio
yService.publish(moduleSpecAdvertis
yService.remotePublish(moduleSpecAd

//al comienza a "escuchar", hasta qu
rice.createInputPipe(pipeAdv, this);

//especificación del módulo publicada

cada 5 segundos busca si hay anuncio
getRemoteAdvertisements() no se bl
fetch_advertisements() {
ad() -> {
e(true) {
try {
discoveryService.getRemoteAdvertisements(null, DiscoveryService.ADV, "Name", "COMP-DIST:EJEM

print("No hay anuncios, esperamos otros 5 segundos");

Thread.sleep(5000);
```

net.jxta.discovery.DiscoveryService

`int getRemoteAdvertisements(String peerid, int type, String attribute, String value, int threshold, DiscoveryListener listener)`

Discover advertisements from remote peers. This does not normally provide an exhaustive search. Instead it provides a "best efforts" search which will provide a selection of advertisements of matching the search criteria. The selection of advertisements returned may be random or predictable depending upon the network configuration and no particular behaviour should be assumed. In general the narrower the query specified the more exhaustive the responses will be. Discovery can be performed in two ways:

- With a null peerid - The discovery query is propagated on via the Rendezvous Service and via local sub-net utilizing IP multicast.
- With a provided peerid - The discovery query is forwarded to the specified peer.

The scope of advertisements returned can be narrowed by specifying an attribute and value pair. The attribute is a case-sensitive string matching the name of an Advertisement XML tag whose values will be matched by the value. Only a limited number of Advertisement XML fields are indexed. `Advertisement.getIndexFields()` will return the fields on which you may query for a particular Advertisement type. The value is a case-insensitive string whose value is matched

Diseño

Para el ejemplo básico de uso de la tecnología, se ha implementado un pequeño **programa que representa un nodo dentro de un grupo determinado**. Consiste en lo siguiente:

1. Cada ejecución crea un nodo distinto, pero este siempre se une al mismo subgrupo.
2. Ya dentro del grupo, el nodo publica un servicio en el que informa de su existencia, y al que va incorporado un canal de entrada a dicho nodo.
3. Se busca por anuncios de este mismo estilo, pero por parte de otros nodos del grupo.
4. Cuando encuentra un anuncio por parte de otro nodo, se establece el canal de salida y se envía un mensaje saludando.
5. Se vuelve al paso 3.

Nuestra clase implementa dos interfaces `DiscoveryListener` y `PipeMsgListener`, que definen los dos métodos que nos servirán para identificar cuando haya un nuevo nodo en el grupo y cuando se reciba un nuevo mensaje.

```
public class EjemploSimpleJXTA implements DiscoveryListener,
    PipeMsgListener {
    [...]

    @Override
    public void discoveryEvent(DiscoveryEvent evento) {...}

    @Override
    public void pipeMsgEvent(PipeMsgEvent event) {...}

    [...]
}
```

La función `discoveryEvent()` "salta" cuando `discoveryService.getRemoteAdvertisements()` encuentra el anuncio que informa de la existencia de otro nodo.

Y `pipeMsgEvent()` "salta" cuando se llama a la función `outputPipe.send()` desde otro nodo que ha encontrado a este.

El `main()` comienza generando un nuevo puerto aleatorio, debido a que al coexistir múltiples nodos en un mismo dispositivo, los procesos y sus conexiones pueden entrar en conflicto dado que JXTA utiliza el protocolo TCP.

A continuación el constructor de la clase crea el objeto `NetworkManager` e inicializa todos los parámetros necesarios antes de crear el nodo.

Se genera un nombre aleatorio para el nodo, se genera el ID único, se crea un nuevo directorio que almacenará la configuración y la caché del nodo (importante, sin este paso todos los nodos leen de los mismo ficheros) y se crea el gestor de la red.

Lo anteriormente comentado corresponde al código de la siguiente diapositiva.

Nota: el código está altamente abreviado

```

public static void main(String[] args) throws PeerGroupException,
    IOException {
    (...)

    int port = 9000 + new Random().nextInt(100);
    EjemploSimpleJXTA hello = new EjemploSimpleJXTA(port);
    hello.start();
    hello.fetch_advertisements();
}

private EjemploSimpleJXTA(int port) {
    peerNombre = "Peer" + new Random().nextInt(10000000);
    peerId = IDFactory.newPeerID(PeerGroupID.defaultNetPeerGroupID,
        peerNombre.getBytes());
    archivoConfiguracion = new File(".") + System.getProperty("file.
        separator") + peerNombre);
    networkManager = new NetworkManager(NetworkManager.ConfigMode.
        ADHOC, peerNombre, archivoConfiguracion.toURI());

    (...)
}

```

A continuación se llama al método `start()`, que arranca la red. El método `networkManager.startNetwork()` devuelve el objeto correspondiente al grupo al que pertenece el nodo por defecto (l. 1-2 siguiente diapositiva).

A continuación se asocia a un nuevo grupo, el que determinamos que sería nuestro subgrupo (l. 4), y se obtienen sus servicios que nos interesan para publicar servicios y enviar mensajes (l. 6-7).

Se crea el canal *input* y el anuncio que lo "promocionará" al grupo (l. 9-11), y ambos componentes se asocian (l. 13-17).

```
1 private void start() throws PeerGroupException, IOException {
2     PeerGroup myPeerGroup = networkManager.startNetwork();
3
4     elSubgrupo = myPeerGroup.newGroup(idGrupo, mAdv, nombreGrupo,
5         descripcionGrupo);
6
7     pipeService = elSubgrupo.getPipeService();
8     discoveryService = elSubgrupo.getDiscoveryService();
9
10    idUnicast = IDFactory.newPipeID(elSubgrupo.getPeerGroupID(),
11        nombreUnicast.getBytes());
12
13    PipeAdvertisement advertisement1 = (PipeAdvertisement)
14        AdvertisementFactory.newAdvertisement(PipeAdvertisement.
15            getAdvertisementType());
16
17    advertisement1.setPipeID(idUnicast);
18
19    pipeService.createInputPipe(advertisement1, this);
20
21    discoveryService.addDiscoveryListener(this);
22 }
```


Y se crean y publican el anuncio que promociona la existencia del nodo (l. 18-24), y el anuncio que realmente establece la conexión entre nodos (l. 27-39), asociado a un canal (l. 29 y 34).

```
18      ModuleClassAdvertisement advertisement = (ModuleClassAdvertisement)
      AdvertisementFactory.newAdvertisement( ModuleClassAdvertisement .
      getAdvertisementType() );
19
20      ModuleClassID moduleClassID = IDFactory.newModuleClassID();
21      advertisement.setModuleClassID(moduleClassID);
22
23      discoveryService.publish(advertisement);
24      discoveryService.remotePublish(advertisement);
25
26
27      moduleSpecAdvertisement = (ModuleSpecAdvertisement) AdvertisementFactory .
      newAdvertisement( ModuleSpecAdvertisement . getAdvertisementType() );
28
29      idCanal = IDFactory.newPipeID( elSubgrupo.getPeerGroupID(), nombreServicio .
      getBytes() );
30      PipeAdvertisement pipeAdv = (PipeAdvertisement) AdvertisementFactory .
      newAdvertisement( PipeAdvertisement . getAdvertisementType() );
31
32      pipeAdv.setPipeID(idCanal);
33
34      moduleSpecAdvertisement.setPipeAdvertisement(pipeAdv);
35
36      discoveryService.publish(moduleSpecAdvertisement);
37      discoveryService.remotePublish(moduleSpecAdvertisement);
38
39      pipeService.createInputPipe(pipeAdv, this);
40 }
```

Después configurar nuestro nodo y grupo, comienza un hilo que continuamente busca anuncios de otros nodos. Sobre `getRemoteAdvertisements()`, según el orden de parámetros:

- No diferencia por ID de nodo
- De ambos tipos, anuncios de nodo y de grupo
- El campo "Name" es igual a "COMP -DIST:EJEMPLO"
- Máximo número de anuncios es 1
- No queremos callback

```
private void fetch_advertisements() {
    new Thread(() -> {
        while(true) {
            try {
                discoveryService.getRemoteAdvertisements(null,
DiscoveryService.ADV, "Name", "COMP-DIST:EJEMPLO", 1, null);

                print("No hay anuncios, esperamos otros 5 segundos")
            ;

                Thread.sleep(5000);
            } catch (InterruptedException e) { e.printStackTrace();
            }
        }
    }).start();
}
```

Cuando el *discovery service* encuentra un nuevo nodo, se dispara el siguiente método. En él se crea el canal de salida por el que saldrá el mensaje hacia dicho nodo. El `idUnicast` es el ID del pipe de salida que definimos anteriormente.

```
1 public void discoveryEvent(DiscoveryEvent evento) {
2     String idNodoEncontrado = "urn:jxta:" + evento.getSource().
3         toString().substring(7);
4     String contenidoMensaje = "Hola nodo!";
5
6     PipeAdvertisement anuncio = (PipeAdvertisement)
7         AdvertisementFactory.newAdvertisement(PipeAdvertisement.
8             getAdvertisementType());
9
10    anuncio.setPipeID(idUnicast);
11
12    Set<PeerID> nodosDestinatarios = new HashSet<>();
13
14    try {
15        nodosDestinatarios.add((PeerID) IDFactory.fromURI(new URI(
16            idNodoEncontrado));
17    } catch (URISyntaxException e) { e.printStackTrace(); }
18
19    try {
20        OutputPipe canalDeSalida = pipeService.createOutputPipe(
21            anuncio, nodosDestinatarios, 10000);
22    } catch (IOException e) { e.printStackTrace(); }
```

Se crea el mensaje y se cubren los campos de origen y contenido (en el fondo es un XML). El mensaje es enviado y dispara el método `pipeMsgEvent()` de la interfaz `PipeMsgListener` en el nodo destino.

```
18     if (canalDeSalida != null) {
19         Message mensaje = new Message();
20
21         MessageElement origen = new ByteArrayMessageElement("
22         From", null, peerId.toString().getBytes(StandardCharsets.
23         ISO_8859_1), null);
24         MessageElement contenido = new ByteArrayMessageElement("Msg
25         ", null, contenidoMensaje.getBytes(StandardCharsets.ISO_8859_1)
26         , null);
27
28         mensaje.addMessageElement(origen);
29         mensaje.addMessageElement(contenido);
30
31         try {
32             canalDeSalida.send(mensaje);
33         } catch (IOException e) { e.printStackTrace(); }
34     }
```

En dicho método se interpreta el mensaje y sus campos.

```
public void pipeMsgEvent(PipeMsgEvent event) {  
    try {  
        Message msg = event.getMessage();  
  
        byte[] msgBytes = msg.getMessageElement("Msg").getBytes(true  
    );  
        byte[] fromBytes = msg.getMessageElement("From").getBytes(  
            true);  
  
        String from = new String(fromBytes);  
        String message = new String(msgBytes);  
    } catch (Exception e) { e.printStackTrace(); }  
}
```

Al ejecutar el programa dos veces, esta es la salida de uno de los nodos (se recortan los IDs por su longitud) :

```
[03:09:44] Comienza nodo JXTA en el puerto 9027
[03:09:44] Configuracion del peer OK
[03:09:44] Grupo definido {
    id: urn:jxta:uuid-677275706[...]
    nombre: grupo_usc
    descripcion: Grupo de prueba, ejemplo simple de JXTA
}
[03:09:44] Nuevo peer {
    id: urn:jxta:uuid-596162616[...]
    nombre: Peer255815
}
[03:09:44] Estas dentro del grupo
[03:09:44] Canal unicast de entrada OK, id: urn:jxta:uuid
-677275706[...]
[03:09:44] Canal multicast de entrada OK, id: urn:jxta:uuid
-677275706[...]
[03:09:44] Clase modulo publicada OK
[03:09:44] Especificacion del modulo publicada OK, canal escuchando
[03:09:44] No hay anuncios, esperamos otros 5 segundos
[03:09:44] Anuncio encontrado, hay un nuevo nodo en la ciudad!
[03:09:44] Canal de salida OK
[03:09:44] Mensaje enviado OK
[03:09:48] Mensaje nuevo recibido
[03:09:48] Mensaje del nodo: urn:jxta:uuid-596162616[...]
    Hola nodooo!!
```

Conclusiones

Conclusiones

JXTA es una gran herramienta en cuando a estructuración de *peers* e intercambio de información en una red P2P.

Es **genérica**, implicando que es válida para multitud de aplicaciones; y **no específica** como otras plataformas, que se centran de manera exclusiva en transferir archivos (BitTorrent, Napster), streaming de video o mensajería instantánea.

Por lo visto y abarcado en este análisis, **es una tecnología robusta, fiable y completa**. Se han revisado algunos manuales y guías, y la sensación es la de que aún queda mucho por explorar sobre las capacidades de JXTA.

A su vez, aunque todo parezca a su favor y sea una muy buena plataforma, es evidente que **no es de las más utilizadas**, ha perdido presencia en la red hasta llegar a ser **complicado el encontrar enlaces activos** e información reciente.

Conclusiones

Tras haber analizado JXTA, creo que **la tecnología aporta:**

- ▶ Una organización de la red virtual en grupos de nodos, abstracción con IDs, y la capacidad de delimitar el acceso y funcionalidades.
- ▶ Intercambio de mensajes asíncronos basados en la consulta/respuesta, publicando anuncios en los servicios del grupo, a la espera de ser descubiertos por el resto de nodos.
- ▶ Comunicación entre nodos a través de canales unidireccionales y con uno o más destinatarios.
- ▶ El uso de documentos XML permite una fácil portabilidad e intercambio con otros lenguajes de programación.
- ▶ Un nivel de seguridad más que suficiente para el uso cotidiano, usando cifrado o logueos al entrar a un grupo.

Conclusiones

Tras revisar otras tecnologías similares, como IPOP (IP-over-P2P), también se habla del uso de redes virtuales y de abstracción con identificadores para los nodos, pero no están enfocadas en un entorno tan social o delimitado por grupos.

Aunque los niveles de seguridad de JXTA no son notablemente altos, otras tecnologías no se preocupan por este aspecto, sino que se centran en la transmisión y enrutamiento de la información.

Por lo que he leído por parte de personas que se "ensuciaron las manos" con esta tecnología, JXTA ofrece una amplia variedad de funcionalidades y posibilidades.

Aún así no se puede negar que ha perdido *momentum*, y que la compra y abandono por parte de Oracle le ha perjudicado. Deduzco que las aplicaciones P2P más populares, lo son porque están implementadas en plataformas más específicas y "livianas".

Bibliografía

Bibliografía

- **JXTA™** The Language and Platform Independent Protocol for P2P Networking - Sitio web oficial (WaybackMachine)
- **JXTA Protocols** by Daniel Brookshier
- **JXTA Java™ Standard Edition v2.5: Programmers Guide**
- **How do I discover peers and send messages in JXTA-JXSE 2.6?**
pregunta de @ib.lundgren en Stack Overflow
- **Why has JXTA been abandoned? Any alternatives out there?**
pregunta de @Stephen K en Stack Overflow