JAVAWORLD

NEWS

Peer-to-peer applications made easy

Get started developing cross-platform P2P applications using Jxta and its Java binding

By Sayed Ibrahim Hashimi

JavaWorld | May 9, 2005 1:00 AM PT

It has been said that Kazaa, the peer-to-peer (P2P) file-sharing application, causes more network traffic then any other application. The Kazaa Website states that it has had more than 385,000,000 downloads! For a comparison, I viewed Download.com's top downloads, which lists Ad Aware as the most popular download, with just 117,000,000 downloads. From Download.com's top 25 downloads, I recognized 11 P2P applications. Just from these observations alone, P2P applications are obviously growing in popularity. But file sharing is not the only type of P2P application. Most of the operations of a typical instant messaging application are P2P. Other examples are forums and distributed databases. And the list just continues to grow.

To create P2P applications like these, you must have a means for discovering and interacting with other peers. Most of the difficulties involved in creating P2P applications are related to maintaining the network of peers, formatting and passing messages, discovering other peers, and other similar issues. Project Jxta and its Java binding handle these aspects of your application. By using Jxta, you can focus on your application, not generic P2P issues.

Jxta is a shortened version of the word juxtapose, which means side-by-side. The Jxta Programmer's Guide defines Jxta as "an open computing platform designed for P2P computing." It is specific neither to any platform nor any programming language. It was conceived at Sun Microsystems and has been released to the open source community to maintain and grow. Along with its release, an initial Java implementation was issued. I focus on that implementation in this article as I discuss how to use Jxta in a Java environment. I also cover the six most common operations of Jxta applications implemented in Java and introduce the tools you need to start writing your own P2P applications. After reading this article, I hope that you will have realized how easy and exciting it can be to create P2P applications. P2P applications will continue to grow not only in popularity, but also in diversity, and tomorrow's developers must start learning these technologies today to stay on the cutting edge.

Java and Jxta

The first step to using Jxta is downloading it from the Jxta downloads page. As most readers will agree, sometimes open source projects can be difficult to acquire and configure for use. Jxta is an example of a great open source project that is also **very** easy to download and use right away. If you are having a hard time and need more information about downloading and using Jxta, refer to the Jxta Programmer's Guide.

When you first run a Jxta-enabled application from a new directory, you will be provided with the GUI configurator.

What exactly is a peer? According to Daniel Brookshire (a well-known Jxta committer and so-called "champion"), it is a "virtual communication point," where different peers can run on the same device. The device is not limited to a PC; it can be a cell phone, a server, or even an item as simple as a sensor. There are special peers, the two that we need to be aware of are rendezvous and relay. A rendezvous peer allows peers to communicate outside the scope of the local subnet, and a relay peer is used to relay information through firewalls.

Let's start by going over the six most common Jxta application operations, as defined in "The Costs of Using Jxta" (IEEE Computer Society, September 2003). They are listed below in the order in which they typically occur.

1. **Starting Jxta:** Starting Jxta is pretty simple and simply a matter of a few lines of code.
2. **Joining a peer group:** A peer group is a set of peers having a common set of interests that have been grouped together. In this article, I cover joining existing peer groups and creating new ones.
3. **Publishing advertisements:** Advertisements, simply stated, are what Jxta is all about. Jxta uses advertisement to discover peers, peer groups, and other resources in a platform-independent manner. I discuss reading, creating, and sending new advertisements later in this article.
4. **Opening an input pipe:** A pipe is one mechanism peers use to communicate with each other. Pipes are "virtual communication channels"—virtual in that pipe users don't know the other peer's actual address. I discuss using pipes to send messages in this article's section on pipes.
5. **Discovering other peer resources:** Before you can communicate with other peers, you must first find some, which I will also discuss.
6. **Opening an output pipe:** Output pipes are used to send messages to

other peers. There are two classes of output pipes: point-to-point, or one-to-one, and propagation, or one-to-many.

Now that you know where this article will take you, let's start our journey.

Peer groups

Peer groups are simply a collection of peers with some set of common interests. Peer groups, like peers, can provide services, yet a peer-group service is not necessarily dependent on a specific peer that fulfills requests to it. As long as a single peer in the group provides the service, then the service is available. Every peer is a member of the world peer group and also, typically, the net peer group, and can elect to join and leave other groups at will. What is the motivation for creating peer groups? Here are a few reasons:

- **Maintain secure region:** If you have a secure peer group, peers in the group don't have to expose their critical information.
- **Provide common services:** Typically, many peers will want to use/provide the same services as other peers, so doing so in the group just makes sense. For example, you could provide a printer or a distributed database service to all peers in the group.
- **Limit ID scope:** Pipe names are matched with the group that they are created in. If two pipes have the same name, but were not created in the same group, then there are no issues with addressing them.

Let's examine how we can create and join a peer group. The methods provided by the `PeerGroup` interface are listed below.

- `newGroup(Advertisement pgAdv)`: typically used for instantiating a group that already exists with the discovered group advertisement
- `newGroup(PeerGroupID gid, Advertisement impl, String name, String description)`: typically used to construct new peer groups
- `newGroup(PeerGroupID gid)`: used to instantiate an existing, and published, peer group with only the peer group ID (`gid`)

Creating peer groups

Creating a basic peer group is relatively straightforward. Let's look at some code:

```
try
{
   //We will create a new group based on the netPeerGroup so let's copy its
   //impl advertisement and modify it.
   ModuleImplAdvertisement implAdv =
     netPeerGroup.getAllPurposePeerGroupImplAdvertisement();

   myPeerGroup = netPeerGroup.newGroup(
         null,              //Create a new group id for this group.
         implAdv,           //Use the above advertisement.
         "Group name",      //This is the name of the group.
         "Group description" //This is the description of the group.
         );

   System.out.println("---Peer group created successfully, id: " +
     myPeerGroup.getPeerGroupAdvertisement().getID() );
   //Now that the group is created, it is automatically published and stored locally,
   //but we need to publish it remotely so other peers can discover it.
   discoveryService.remotePublish( myPeerGroup.getPeerGroupAdvertisement() );
   System.out.println("---Published peer group advertisement remotely");
}
catch (Exception e)
{
   System.out.println("An error occurred");
   e.printStackTrace();
}
```

The call to `newGroup()` creates and publishes the group to the local cache. Most likely, you will want to publish this advertisement to other peers when you create it, which you can do by calling `remotePublish()`. This method will push the peer group advertisement to other peers. If you need to make sure that you send the advertisement to peers on another subnet, you must ensure you are connected to a rendezvous peer. To do this, use the following code, assuming your rendezvous peer is up and configured correctly:

```
 private void connectToRdv(PeerGroup peerGroup)
{

    if( rdv == null)
    {
        //Get the rdv service
        rdv = peerGroup.getRendezVousService();
    }

    //Make sure that we are connected before proceeding
    while( !rdv.isConnectedToRendezVous() )
    {
        try
        {
            Thread.sleep(5000);
        }
        catch (InterruptedException e1)
        {
            System.out.println("rdv connect interrupted");
            e1.printStackTrace();
        }
    }
}
```

Joining peer groups

Joining a peer group can be more difficult then actually creating one. Even if we have an unsecured peer group, we still must create credentials, empty credentials, and send these credentials to the peer group we are trying to join.

Since we have a peer group advertisement, we need to create all the credentials necessary and join the group. Before we look at the `joinGroup()` method, let's look at one of the classes it uses, the `MembershipService` class. There are three methods in `MembershipService` that we are interested in, specifically `apply()`, `join()`, and `resign()`. We pass to the `apply()` method the type of authentication desired, and if that type is supported, it returns to us an `Authenticator`. We use this `Authenticator` to actually join the group. We pass it as an argument to the `join()` method, and it verifies our credentials. When a peer wants to leave a group, the call to `resign()` facilitates this.

Now let's look at the `joinGroup()` method:

```
private void joinGroup()
{
  //Assuming myPeerGroup has been instantiated
   //before calling this method.
   System.out.println("Trying to join the peer group");
   try
   {
      //Create the document that will identity this peer.
      StructuredDocument identityInfo = null;
         //No identity information required for our group.

      AuthenticationCredential authCred =
        new AuthenticationCredential(
          myPeerGroup, //Peer group that it is created in
          null, //authentication method. );
      MembershipService membershipService =
        myPeerGroup.getMembershipService();
        Authenticator auth = membershipService.apply(authCred);

      //See if the group is ready to be joined.
      //Authenticator currently makes no distinction between
      //failed and unfinished authentication.
      if( auth.isReadyForJoin() )
      {
         Credential myCred = membershipService.join(auth);
         System.out.println("Joined myPeerGroup");

         System.out.println("Group id: " +
            myPeerGroup.getPeerGroupID() );
      }
      else
      {
         System.out.println("Unable to join the group");
      }
   }
   catch (Exception e)
   {
      System.out.println("An error occurred");
      e.printStackTrace();
   }
}
```

Now that we have successfully joined the group, we are able to employ provided peer group services and send messages to the members. When developing P2P applications, thinking about where you want your peer group boundaries ahead of time will assist you in the long run. Keep in mind that peer group boundaries can span many networks.

The joining process can seem daunting at first, but it is pretty straightforward once you do it a few times. Now it is possible to employ many different methods to secure a peer group—the complexity of the joining process depends on the type of authentication desired. I do not discuss these methods here.

Pipes

As explained earlier, a pipe is a virtual channel of communication between two peers. Pipes can be confusing for beginners because newbies try to relate them to what they already know—sockets. While I discuss pipes, keep in mind that they are much more abstract then sockets.

In the most basic form, there are two types of pipes; input pipes and output pipes. Applications use input pipes to receive information, and output pipes, to send information. Pipes can be used in two addressing modes:

- **Unicast (point-to-point) pipes:** These pipes connect one output pipe to a single input pipe, but a single input pipe can receive messages from different output pipes
- **Propagate pipes:** These pipes connect a single output pipe to many different input pipes

Pipes are an unreliable, unidirectional, and asynchronous means of communication. Beefed-up implementations of pipes are available that provide reliability, bidirectional capabilities, and secure transit.

To create a pipe, first you must create a pipe advertisement and publish it. Then you need to get the pipe service from the peer group and use it to create the pipe. Each pipe has a pipe ID associated with it, which is used to address the pipe.

To create a new pipe ID, we use the `IDFactory` in the `net.jxta.id` package. Here is a sample of how to create and print the ID:

```
ID id = IDFactory.newPipeID( peerGroup.getPeerGroupID() );
 System.out.println( id.toURI() );
```

Note: `peerGroup` is the peer group for which you want to create the pipe.

So two peers can communicate with each other, they must know the pipe IDs for the pipes with which they wish to communicate. There are a few ways to ensure that they both know this information:

- Both peers read in the same pipe advertisement from a file
- The pipe ID is hard-coded into the applications
- Publish and discover the pipe ID at runtime
- Pipe ID is generated from a well-known ID