Note: This information was taken from the JavaONE presentation by Akhil Arora.
Learning Objectives

• This module will help you...
  – Learn about the architecture of JXTA for J2ME
  – Learn the JXTA for J2ME APIs
  – Learn how the JXTA for J2ME APIs can be used to build a simple peer-to-peer (p2p) application

Learning Objectives

• In this module, we'll be discussing JXTA for J2ME, which provides JXTA-compatible capabilities on very small footprint devices.
• We'll first describe the Project JXTA for J2ME architecture, and then review the JXTA for J2ME APIs.
• We'll end the module with code examples illustrating how the APIs can be used to build a simple P2P application.
What is JXTA for J2ME?

- Provides a peer-to-peer infrastructure for small devices
- Interoperates with Project JXTA protocols
- Provides a simple and easy-to-use API
- Has a small footprint: only about 10k
- MIDP-1.0 compliant
- Supports iAppli
- Network and carrier-independent

What is JXTA for J2ME?

- Let's start with the basic question: what is JXTA for J2ME?
- JXTA for J2ME provides a peer-to-peer infrastructure for small devices, like cell phones and PDAs.
- It is compatible with and interoperates with Project JXTA protocols. Using JXTA for J2ME, any MIDP device can participate in P2P activities with other MIDP devices. At the same time, a MIDP device is able to participate, with some restrictions, in P2P activities with JXTA peers running on desktops or servers.
- JXTA for J2ME is designed to be simple and easy-to-use by developers. It features a simple to use API that makes it easy to get started writing applications.
- JXTA for J2ME has a small footprint – only 10k – which is small enough to be used with cell phones and PDAs. It is MIDP-1.0 compliant, supports iAppli, and is both network- and carrier-independent.
Project JXTA Virtual Network

- JXTA for J2ME peers participate in the Project JXTA virtual network as edge peers. At the virtual level, JXTA peers appear to communicate directly with other JXTA peers. The virtual network hides the underlying physical network topology.

- At the physical level, JXTA for J2ME peers use HTTP to communicate with JXTA Relay peers. These Relay peers enable communication with other peers in the JXTA network.
Architecture

- The main components in the JXTA for J2ME architecture are the JXTA for J2ME peers, JXTA relays, and JXTA Proxies.
- JXTA for J2ME peers send messages to a JXTA Relay, which forwards them to a JXTA Proxy.
- The JXTA Relay is also used by JXTA J2SE clients that may be behind firewalls. In this case, the messages bypass the proxy.
Architecture (continued)

- HTTP Relay
  - Relays messages between HTTP clients
  - Provides a Message Store for disconnected operation
- JXTA Proxy
  - Stores JXTA Advertisements
  - Filters JXTA traffic
  - Processes JXTA's XML messages and converts them to a compact, binary, easy-to-parse format

Architecture (continued)

- As we saw on the previous architecture diagram, JXTA for J2ME peers interact with JXTA relays and JXTA proxies.
- HTTP Relays are responsible for relaying messages between HTTP clients. They also provide a message store for disconnected operation. A peer can reconnect after a period of inactivity and retrieve messages that have been stored for it.
- A Proxy is needed because of the limited storage and processing capabilities on the small footprint devices. The JXTA Proxy stores JXTA advertisements and filters JXTA traffic. It also processes JXTA’s XML messages and converts them into a compact binary format that is easier for the handheld device to parse.
JXTA Relay

- Runs on publicly-addressable peers
- Provides connectivity for peers that are behind NAT and/or firewall
- Provides a message store for disconnected operation
- Lease management
- Uses the HTTP protocol
- GET and POST messages in a single round-trip to minimize latency

JXTA Relay
- The JXTA Relay runs on publicly-addressable peers and provides connectivity for peers that are behind NAT and/or firewall. The use of JXTA relays is not new or unique to JXTA for J2ME – it is also used by other JXTA peers.
- JXTA Relays provide a message store for disconnected operation.
- Lease management – when clients connect, they are assigned a lease, during which time the relay agrees to hold messages for them. If a client does not reconnect before this lease expires, their queued messages may be discarded.
- JXTA Relays use the HTTP protocol to communicate with peers, and GET and POST message in a single round trip to minimize latency.
JXTA Proxy

- Is a JXTA Service
- Provides services for clients that
  - Have limited storage
  - Are on networks with limited bandwidth
- Translates client requests to JXTA protocols and the other way around
  - Currently supports Discovery and Pipe protocols
- Asynchronous

JXTA Proxy

- The JXTA Proxy Service provides services for clients that have limited storage and/or are on networks with limited bandwidth.
- It translates client requests to JXTA protocols, and the other way around. Currently, the JXTA Proxy service supports Discovery and Pipe protocols.
- The JXTA Proxy service, like other JXTA services, uses an asynchronous communication model.
API Design Goals

- Minimize the number of classes: no interfaces, factories, listeners, inner classes
- Minimize the number of threads
- Have a low conceptual weight, low barrier-to-entry
- Hide complexity for the casual developer, while allowing low-level access for the advanced developer
- Use method overloading

API Design Goals

- The API is designed to hide the complexity for new or casual developers. It minimizes the number of classes, and avoids the use of Interfaces, Factories, Listeners, Threads, and Inner Classes. This reduces the size of the API, lowers the conceptual weight, and presents a low barrier-to-entry — which enables new programmers to start writing applications for JXTA for J2ME very quickly.
- A further goal of the API is to hide complexity for the casual developer while allowing low-level access for the advanced developer.
- The API uses method overloading to minimize the number of methods, so as to minimize the API footprint.
API Overview

- **PeerNetwork**
  - Abstraction for a Network of Peers in a particular Group
- **Message**
  - Abstraction for a JXTA Message
- **Element**
  - Each JXTA Message can be composed of multiple elements

The API consists of only three classes:
- **PeerNetwork** – Abstraction for a network of peers in a particular peergroup. This includes operations that can be invoked on the JXTA network.
- **Message** – Methods to create and manipulate JXTA messages.
- **Element** – Methods to construct and manipulate the basic components of JXTA messages
API: PeerNetwork

static PeerNetwork createInstance(String peerName)
   – Factory method, used to create an instance of a PeerNetwork.

byte[] connect(String relayURL, byte[] state)
   – Connect to a relay.

create(PEER|GROUP|PIPE, String name, String arg)
   – Create a JXTA peer, group, or pipe.

search(PEER|GROUP|PIPE, String query)
   – Search for JXTA peers, groups, or pipes.

API: PeerNetwork

- createInstance – this method is used to create an instance of a PeerNetwork.
- connect – once you have created an instance of a PeerNetwork, the connect method is used to connect to a relay. This method internally manages all connection-related issues. Once you are connected, you can start sending messages – for example, search messages to find other JXTA resources. The state is treated like a cookie; it is used for session management.
- create – this method is used to create a JXTA peer, peergroup, or pipe.
- search – this method is used to search for JXTA peers, peergroups, or pipes.
API: PeerNetwork (continued)

- **listen**(String pipeName, String pipeld, String pipeType)
  - *Open a pipe for input.*
- **close**(String pipeName, String pipeld)
  - *Close an input pipe.*
- **send**(String pipeName, String pipeld, String pipeType, Message data)
  - *Send data to the specified pipe.*
- **Message poll**(int timeout)
  - *Poll the relay for messages addressed to this peer.*

API: PeerNetwork (continued)

- **listen** – this method is used to open a pipe for input.
- **close** – as its name implies, this method is used to close an input pipe.
- **send** – this method is used to send data to a specified pipe.
- **poll** – this method is used to poll the relay for messages addressed to this peer. Messages will be stored on the relay until the JXTA for J2ME peer polls for them.
API: Message

**Message** (Element[] elements)
- Construct a Message from an array of Elements.

**getElementCount()**
- Return the number of Elements contained in this Message.

**getElement(int index)**
- Return the Element contained in this Message at the specified index.

**getSize()**
- Returns the size in bytes of this Message.

API: Message
- The Message class is used to manipulate JXTA messages. Each Message contains one or more Elements.
- **Message** - this method is used to create a Message from an array of Elements.
- **getElementCount** - This method returns the number of Elements contained in this Message.
- **getElement** - this method returns the Element at the specified index.
- **getSize** - this method returns the size, in bytes, of this Message.
API: Element

Element(String name, byte[] data, String nameSpace, String mimeType)
   - Create an Element from its parts.

String getName()
   - Return the name of the Element.

byte[] getData()
   - Return the data in the Element.

String getNameSpace()
   - Return the namespace used by the Element name.

String getMimeType()
   - Return the MIME type of the data in the Element.

API: Element

- The Element class is used to manipulate JXTA Elements, the components that are used to build JXTA Messages. Each Element contains a name and name space, data, and a MIME type.

- Element – this method is used to create an Element from its constituent parts.

- getName
gedata
getNameSpace
getMimeType  – These methods return the various pieces of an Element.
JXME Example: Chat

- IRC Style chat group chat
- Supports 1-1 IM style chat as well
- Interoperates with J2SE Group Chat and JXTA-C Group Chat

JXME Example: Chat

An example Chap applications is available for JXTA for J2ME platform. This IRC-style chat group chat applications supports 1-1 Instant Messaging-style chat as well. The application interoperates with both J2SE Group Chat and JXTA-C Group Chat.
Example: part 1

```java
class Sample implements Runnable {
    String RELAY = "http://209.25.154.233:9700";
    PeerNetwork peer = null;
    Sample() {
        peer = PeerNetwork.createInstance("sample-peer");
        peer.connect(RELAY, null);
        peer.listen("sample-pipe", null,
                PeerNetwork.PROPEGATE_PIPE);
    }
}
```

Example: part 1

- This example illustrates how to use the JXTA for J2ME APIs.
- The example starts by calling `PeerNetwork.createInstance()` to create an instance of a PeerNetwork. We pass in one argument, which is the name of our new peer.
- A connection to the relay needs to be established before any other operations can be invoked. We call `PeerNetwork.connect()` to connect to our JXTA Relay. This method takes two arguments: the address of the JXTA relay, and a persistent state (null in this example).
- Next, we create an open a propagate pipe by calling `PeerNetwork.listen()`. This method takes three arguments: the name of the pipe, the pipe ID, and the type of pipe (unicast or propagate). Because we pass in a null pipe ID, a new pipe will be created for us. The pipe ID for this new pipe will be returned asynchronously in a response message. We will also receive an asynchronous response indicating the status of this call to `listen()` -- whether it completed successfully or an error occurred.
Example: part 2

WaitForPipeId() {
    do {
        Message msg = peer.poll(1000);
        if (msg != null) {
            pipeId = processPipeIdResponse(msg);
        }
    } while (pipeId == null);

Example: part 2

- This method waits until we receive the pipe ID for our newly created input pipe, which was created by our call to listen().
- Here, we poll for messages with a timeout of 1000 milliseconds. If we received a message, we call our method processPipeIdResponse() to check if it was the response we're looking for – pipe ID of the input pipe we created.
- We continue to loop until we get the pipe ID.
Example: part 3

```java
String processPipeIdResponse(Message m) {
    for (int i=0; i < m.getElementCount(); i++) {
        Element e = m.getElement(i);
        if (Message.REQUESTID_TAG.equals(e.getName())) {
            requestId = new String(e.getData());
        } else if (Message.ID_TAG.equals(e.getName())) {
            pipeId = new String(e.getData());
        }
    }
    return pipeId;
}
```

Example: part 3
- This method processes the messages we receive, and checks if contains the pipe ID for our pipe.
- This method loops through and looks at each Element. We're looking for a message that contains an element with tag Message.REQUESTID_TAG and another element with tag Message.ID_TAG – the pipe ID we're looking for. If we find it, we return the pipe ID.
Example: part 4

send(String text) {
    Element[] el = new Element[2];
    el[0] = new Element("JxtaTalkSenderName",
                       "sample-peer".getBytes(), null, null);
    el[1] = new Element("JxtaTalkSenderMessage",
                        text.getBytes(), null, null);
    Message m = new Message(el);
    peer.send("sample-pipe", pipeId,
              PeerNetwork.PROPOGATE_PIPE, m);
}

Example: part 4

- This method is used to create and send a message.
- We first create an array containing two Elements. The first element has a tag of JxtaTalkSenderName and a value/data of “sample-peer”.
- The second element has a tag of JxtaTalkSenderMessage, and a value/data of the message that we want to send.
- We then pass this array of Elements to the Message constructor to create our Message.
- Once we have our Message, we call PeerNetwork.send() to send the message. We pass this method the name of our pipe, the pipe ID (which we received previously), the type of pipe, and the message we want to send.
Example: part 5

Run() {
    Message message = null;
    while (true) {
        m = peer.poll(1000);
        if (m != null) {
            for (int i = 0; i < m.getElementCount(); i++) {
                Element e = m.getElement(i);
                if ("JxtaTalkSenderId" == e.getName()) {
                    // Process e...
                }
            }
        }
    }
}

Finally, we loop forever waiting for messages.

We call PeerNetwork.poll() to check for any messages. If we've received a message, we loop through each of its Elements and check if it contains an Element with the tag/name of "JxtaTalkSenderId". If it does, it's one of the messages we're expecting, and we can do whatever processing is appropriate.
Resources

• Go to http://jxme.jxta.org for the source code, white papers, API documentation, sample applications and programmer's guide
• Send mail to discuss@jxme.jxta.org for help

The best starting place for more information is the http://jxme.jxta.org web site. Here you can find the source code, white papers, API documentation, and sample applications.
• You can also join a mailing list or send mail to discuss@jxme.jxta.org for help.
End – JXTA for J2ME