

# Project JXTA – Guide to a peer-to-peer framework (Part 2)

---

Ekaterina Chtcherbina

Siemens AG, Corporate Technology  
Munich

Thomas Wieland

Siemens AG, Munich  
University of Applied Sciences, Coburg

# Agenda (part 2)

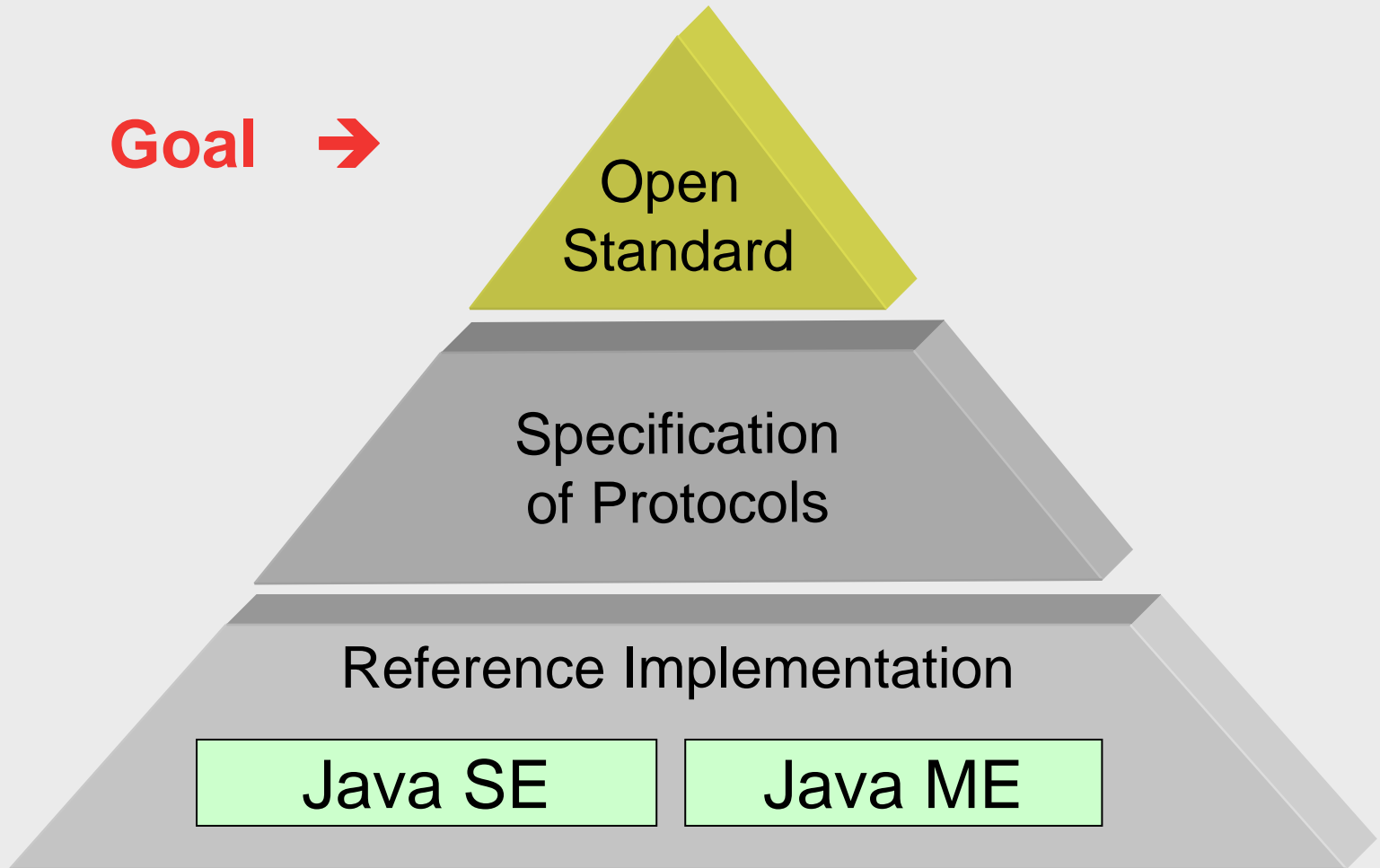
- JXTA Bindings
- Download & Configuration
- Deep into Protocols
- Demo
- Creating a simple JXTA App that can
  - discover other peers and
  - send messages to them

# JXTA Bindings

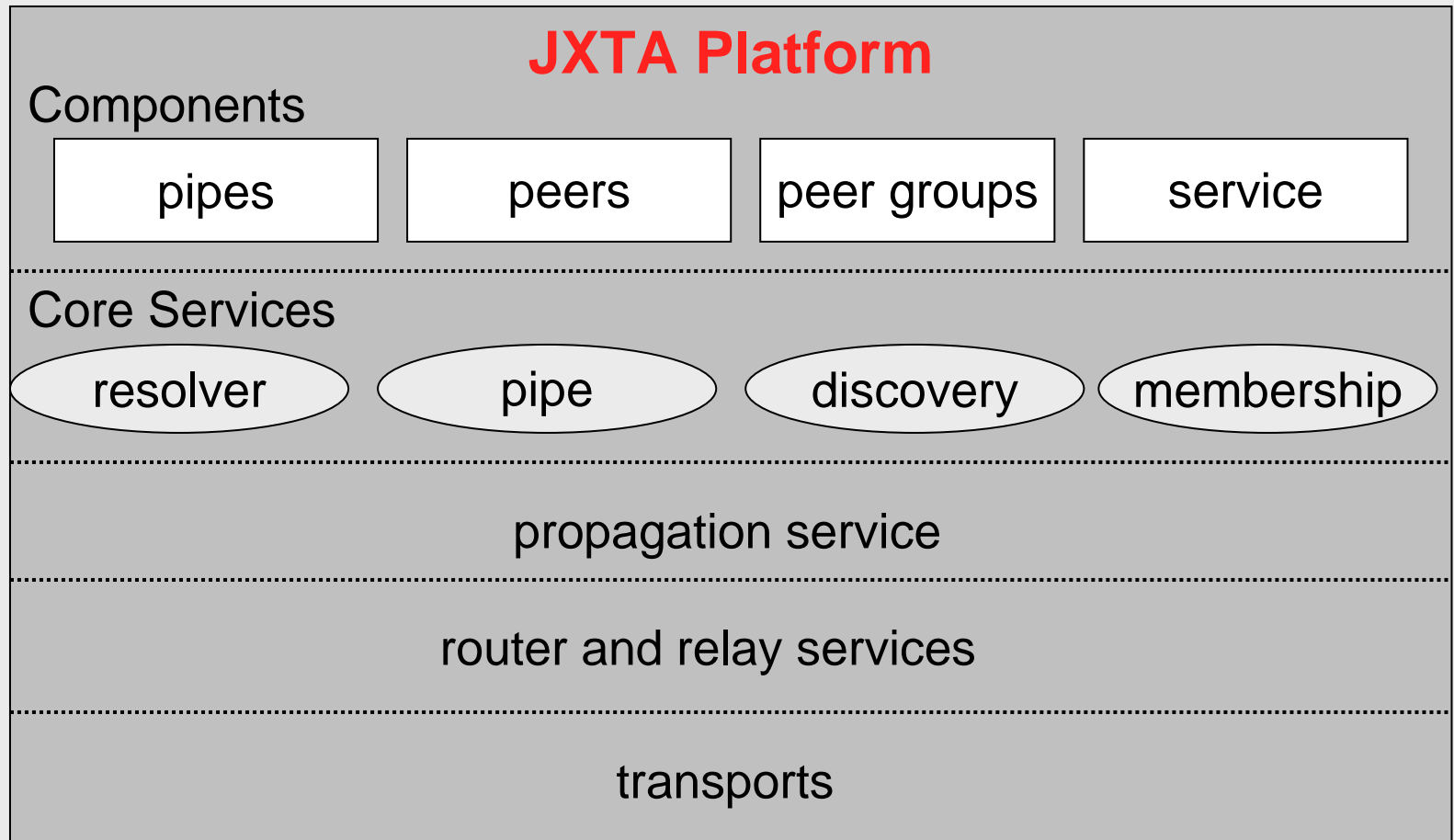
---

# JXTA Goal

**Goal** →



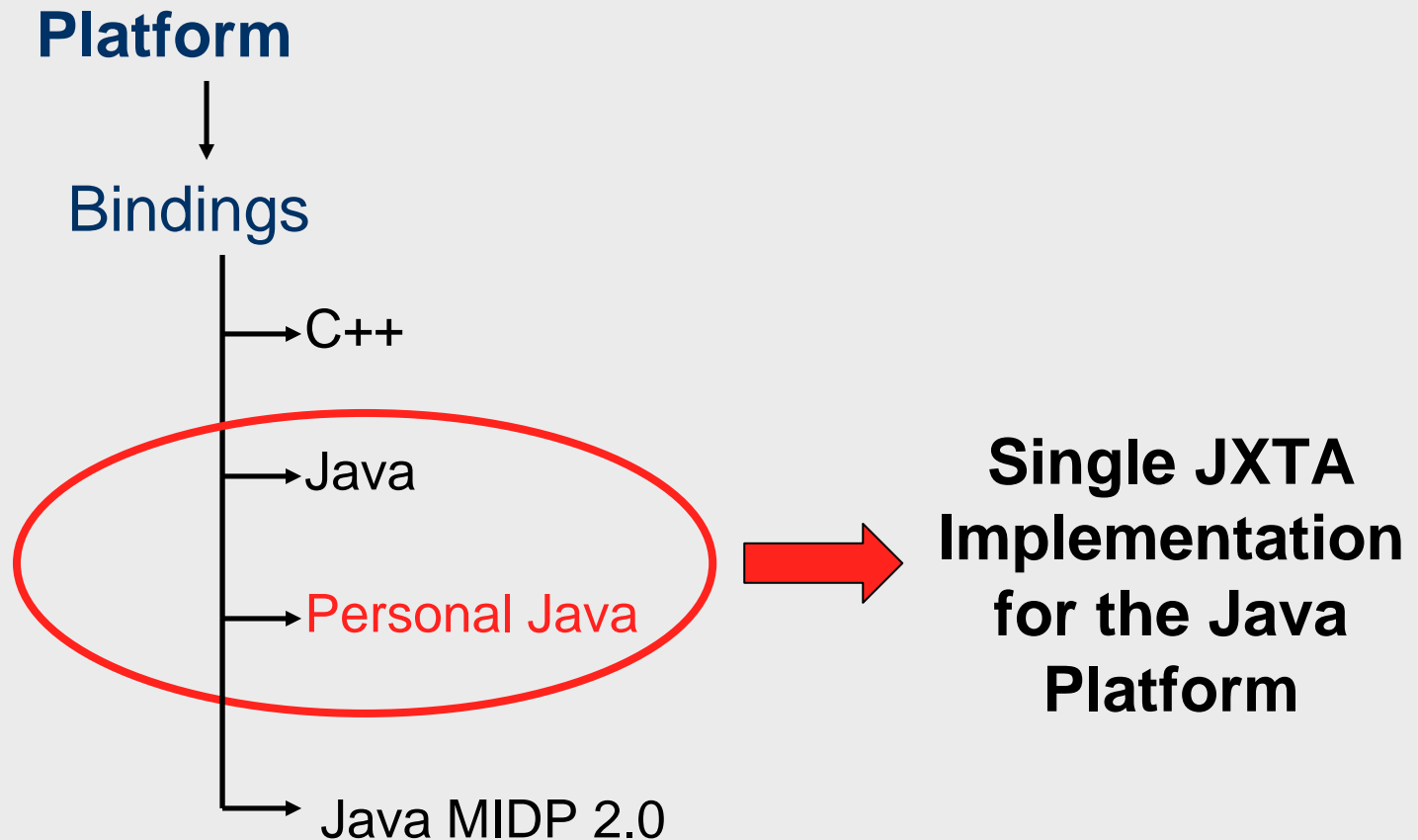
# JXTA Architecture



# JXTA Projects

- Core
  - Peer Groups: mechanisms to/for create and delete, join, advertise, discover, communication, security, content sharing
  - Peer Pipes: transfer of data, content, and code in a protocol-independent manner
  - Peer Monitoring: including access control, priority setting, traffic metering and bandwidth balancing
- Services
  - expand upon the capabilities of the core and facilitate application development
  - mechanisms for searching, sharing, indexing, and caching code and content to enable cross-application bridging and translation of files
- Applications
  - built using peer services as well as the core layer

# JXTA Bindings



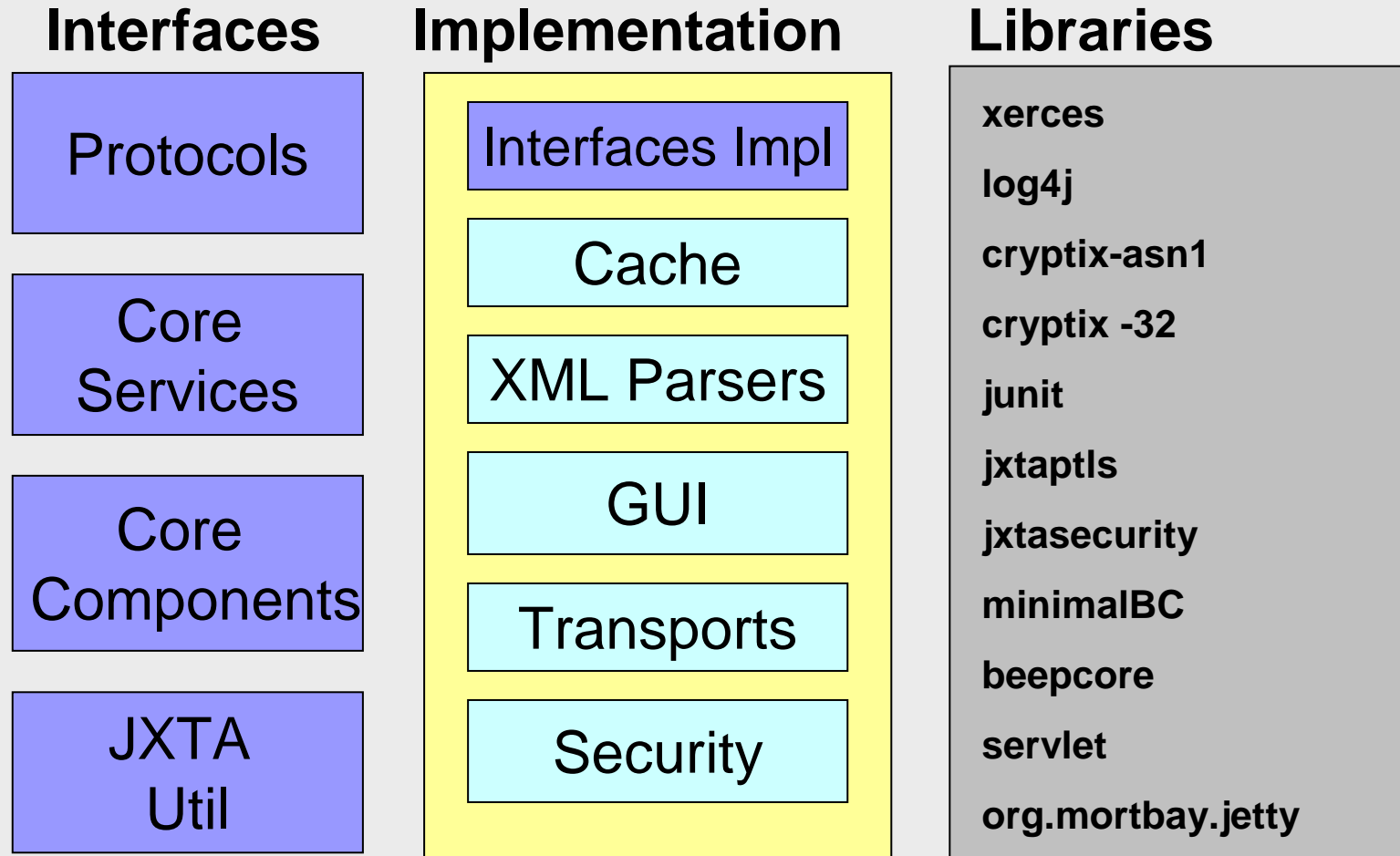
# Interoperability Level



# JXTA Java Bindings

- Set of Java classes that implement the **JXTA protocol** and a set of **objects and services** that enable P2P communication
- The code contains two main packages
  - net.jxta => **interfaces**
  - net.jxta.impl => **implementations**
- Daily & Stable Builds

# Source Code



# Download & Configuration

---

# Download & Installation

- Supported Platforms
  - Windows, Unix, Solaris, Mac OSX ...
- Download JXTA Platform @
  - <http://download.jxta.org/easyinstall/install.html>
  - OR <http://download.jxta.org/stablebuilds/index.html>
- System Requirements
  - JVM

# Configuring the JXTA Core

- Configuration interface is displayed each time when either configuration file does not exist yet or any time the platform needs to be reconfigured
- Settings are stored in the **PlatformConfig** file
- Peer certificate is stored under **\pse** directory (e.g. password under **\pse\etc\passwd**)

# Configuration Interface I

The screenshot shows a window titled "JXTA Configurator" with a blue title bar. Below the title bar, there is a text prompt: "See 'http://shell.jxta.org/index.html' for config help". The window contains four tabs: "basic", "advanced", "Rendezvous/Relays", and "Security". The "basic" tab is selected, and the text "Basic settings" is displayed above the form fields. The form includes a "Peer Name" field with the value "Mr. Fischer" and a "(Mandatory)" label. Below this is a checkbox labeled "Use a proxy server if behind firewall". Underneath the checkbox, there are two input fields for "Proxy address": "myProxy.myDomain" and "8080". At the bottom of the window are "OK" and "Cancel" buttons.

Peer Name

Proxy Settings

# Configuration Interface II

The screenshot shows the 'JXTA Configurator' window with the 'advanced' tab selected. The window title is 'JXTA Configurator' and it includes a help link: 'See "http://shell.jxta.org/index.html" for config help'. The 'Security' sub-tab is active, and the text 'Experienced Users Only' is displayed. A 'Trace Level' dropdown is set to 'error'. The 'TCP settings' section is expanded, showing 'Enable if direct LAN connection.' with a checked 'Enabled' checkbox. Below this, there are two rows of settings: one for 'Manual' (unchecked) with a dropdown set to '139.23.186.77' and a text box containing '9701', and another for '(Optional) Public NAT address' with a text box containing '9701'. The 'HTTP settings' section is also expanded, showing 'Must enable if behind a firewall/NAT' with a checked 'Enabled' checkbox. Below this, there are two rows of settings: one for 'Manual' (unchecked) with a dropdown set to '139.23.186.77' and a text box containing '9700'. At the bottom of the window are 'OK' and 'Cancel' buttons.

Port for  
broadcasting

If you want to  
start 2 instances  
of the application  
on the same PC  
this port should be  
different

# Configuration Interface I

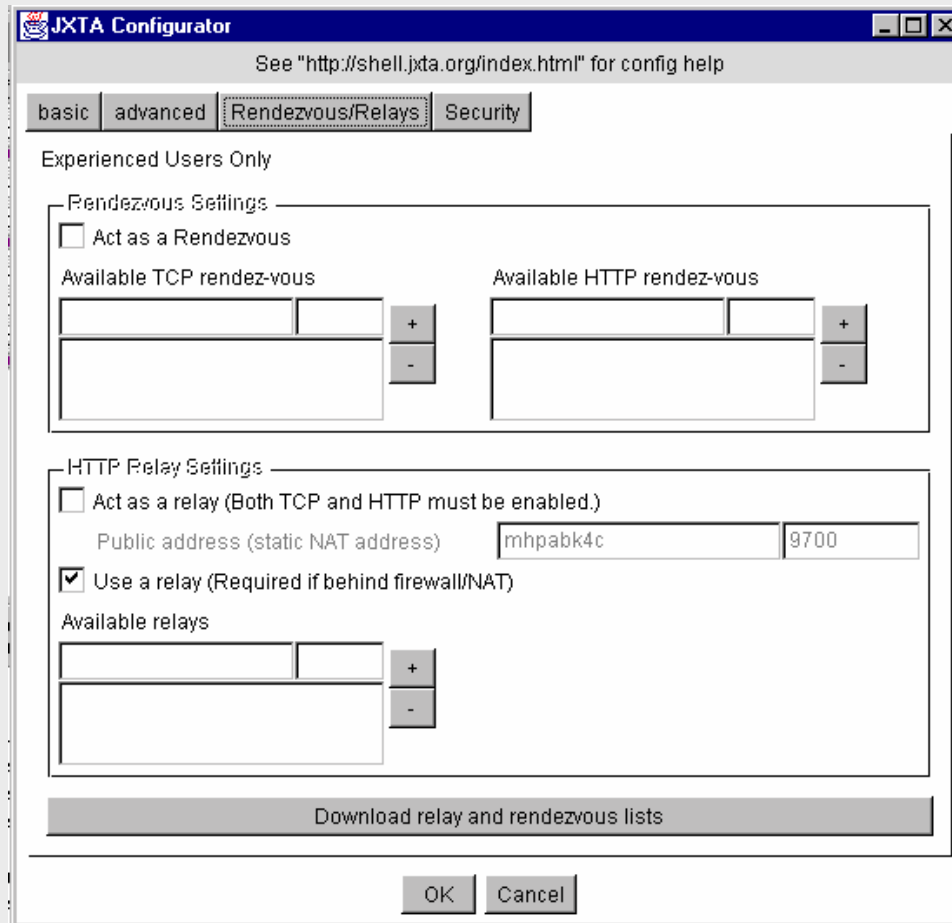
The screenshot shows the JXTA Configurator window with the following elements:

- Window title: JXTA Configurator
- Help text: See "http://shell.jxta.org/index.html" for config help
- Navigation tabs: basic, advanced, Rendezvous/Relays, Security
- Section: Basic settings
- Peer Name field: Mr. Fischer (Mandatory)
- Proxy checkbox:  Use a proxy server if behind firewall
- Proxy address field: myProxy.myDomain
- Proxy port field: 8080
- Buttons: OK, Cancel

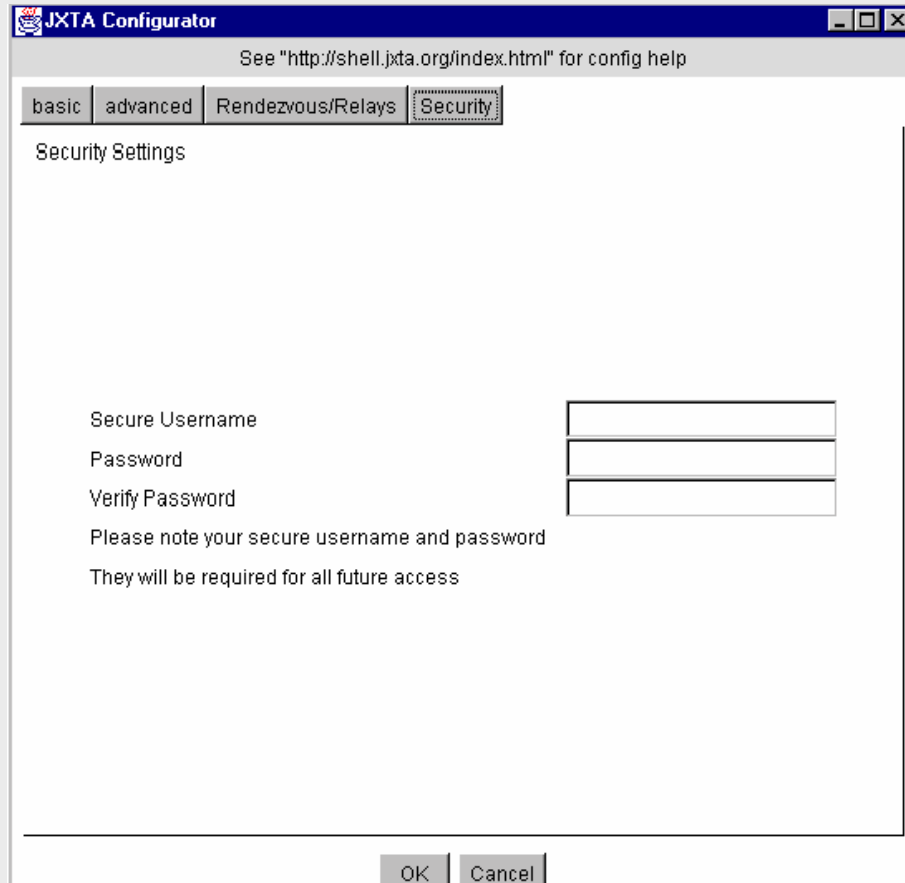
Peer Name

Proxy Settings

# Configuration Interface III



# Configuration Interface IV



The image shows a screenshot of the JXTA Configurator application window. The title bar reads "JXTA Configurator" and includes standard window controls. Below the title bar, a message says "See 'http://shell.jxta.org/index.html' for config help". There are four tabs: "basic", "advanced", "Rendezvous/Relays", and "Security", with "Security" being the active tab. The main content area is titled "Security Settings" and contains the following fields and text:

- Secure Username:
- Password:
- Verify Password:

Please note your secure username and password  
They will be required for all future access

At the bottom of the dialog are "OK" and "Cancel" buttons.

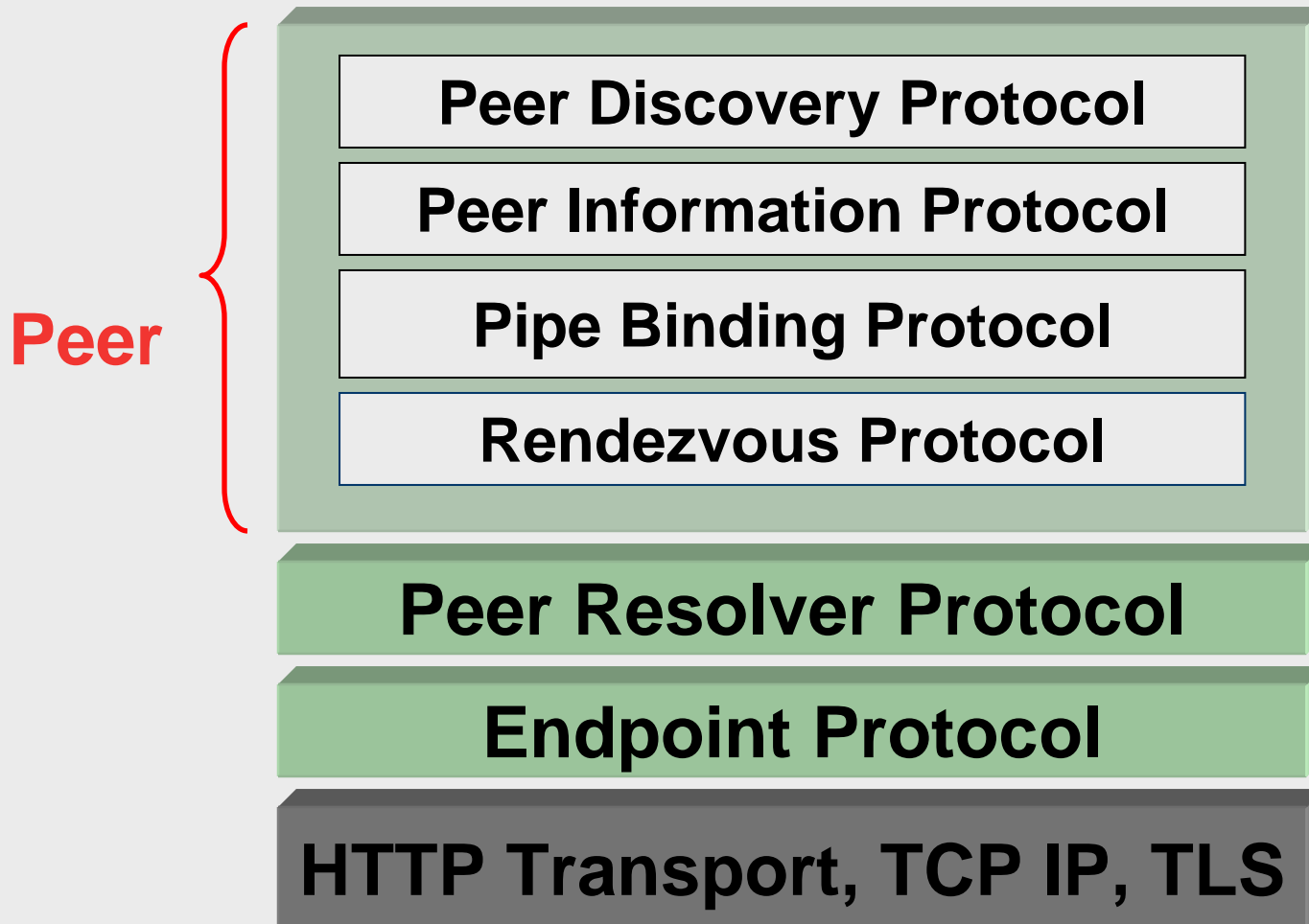
# JXTA Settings

- Mandatory
  - Name
    - Peer Name that is translated into the unique PeerID
  - TCP Settings
    - IP Address and port for broadcasting
  - Security
    - User name + Password
- Optional
  - Firewall
    - Proxy Name + HTTP Settings
  - Rendezvous service
    - For those how want to sacrifice their PC resources for the JXTA network

# Protocols

---

# JXTA Protocols



# Endpoint Protocol

- A set of messages to find **routing information (hops & gateways)** for sending messages between the source and the destination peers
- If routing information is not found in the cache then a **route resolver query** is sent to the available relay peers
- **EndpointService** is responsible for the work

# Endpoint Protocol (2)

```
<?xml version="1.0"?>
<jxta: EndpointRouter>
  <Src>
    urn:jxta:uuid- 59616261646162614A7874615
    0325033FF8F1B4D77E4495780B483A5C89CEC9103
  </Src>
  <Dest>
    urn:jxta:uuid- 5961627651162614A7874615
    0325033FF8F1B4D77E4495780B483A5C89CEC9542
  </Dest>
  <TTL> 10000 </TTL>
  <Gateway> Ordered Sequence of Gateways </Gateway>
</jxta: EndpointRouter>
```

Peer ID of  
the Source

Peer ID of the  
Destination

# Peer Resolver Protocol

- Enables peers to send **generic query requests** against “handlers” that are registered in a peer group
- **ResolverService** does the job
- Registered handlers must implement **QueryHandler**
- It is the **basis** for other query protocols – PDP and PIP
- Each service can register a handler

# Peer Resolver Protocol (2)

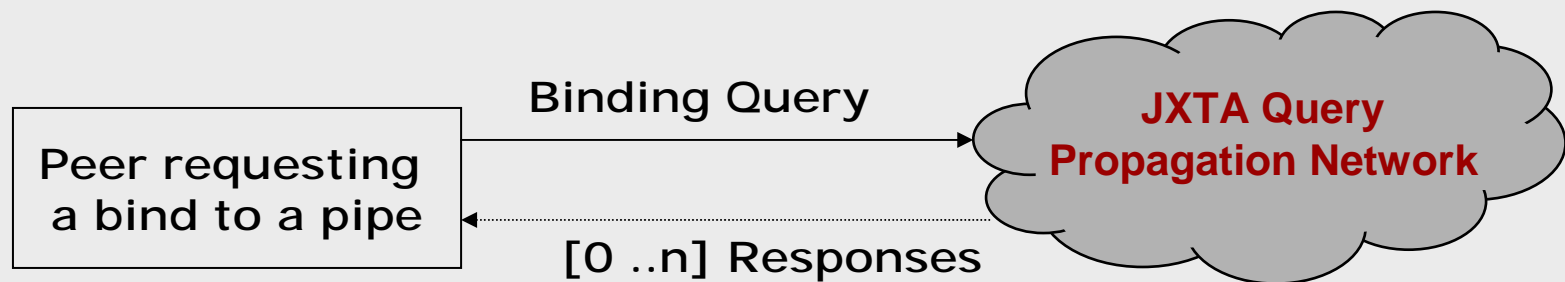
```
<?xml version="1.0"?>
<jxta: ResolverQuery>
  <Credential> Credential </Credential>
  <HandlerName> Name </HandlerName>
  <QueryID> Incremental Query ID <QueryID>
  <Query> Query String < /Query >
  <SrcPeerId>
    urn:jxta:uuid- 59616261646162614A7874615
    0325033FF8F1B4D77E4495780B483A5C89CEC9103
  </ SrcPeerId >
</jxta: ResolverQuery >
```

# Peer Resolver Protocol (3)

```
<?xml version="1.0"?>
<jxta: ResolverResponse>
  <Credential> Credential </Credential>
  <HandlerName> Name </HandlerName>
  <QueryID> Incremental Query ID <QueryID>
  <Response> Response String < /Response >
</jxta: ResolverResponse >
```

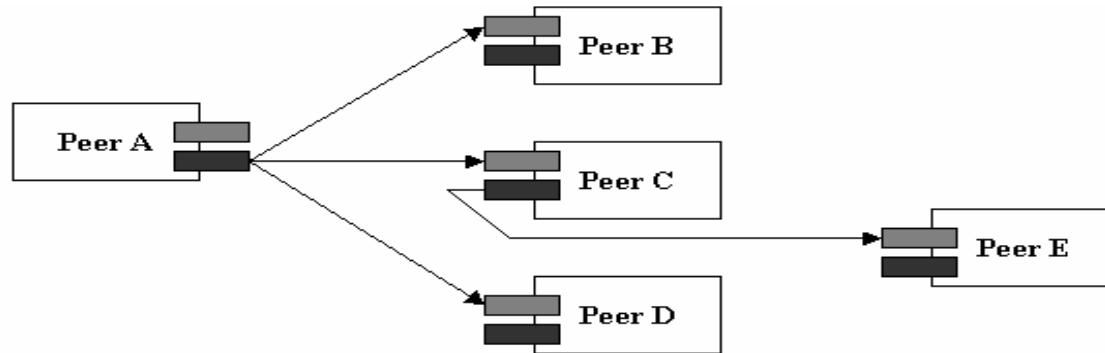
# Pipe Binding Protocol

- Local **PipeService** is responsible for the work
  - re-establishment of the pipe against a constantly changing non-deterministic network topology



# Pipe Binding Protocol (2): Communication Channel

- **Multicast or Propagate Message:** checking the existence of the output pipe at the endpoint



- **Unicast Message:** sending a message to the certain peer



■ Input pipe  
■ Output pipe

# Pipe Binding Protocol (3)

```
<?xml version="1.0"?>
<jxta: PipeBindingQuery>
  <Credential> URI </Credential>
  <Peer>
    urn:jxta:uuid- 5961627651162614A7874615
    0325033FF8F1B4D77E4495780B483A5C89CEC9542
  </Peer>
  <Cached> true or false </Cached>
  <PipeID>
    urn:jxta:uuid- 5961627651162614A7874615
    13033FF8F1B4D77E4495780B483A5C89CEC9542
  </PipeID>
</ jxta: PipeBindingQuery >
```

Optional  
Peer ID

Whether to  
look in cached  
info

- Returns true or false (found or not found)

# Pipe Binding Protocol (4)

```
<?xml version="1.0"?>
<jxta: PipeBindingResponse>
  <Credential> URI </Credential>
  <PipeId>
    urn:jxta:uuid- 5961627651162614A7874615
    0325033FF8F1B4D77E4495780B483A5C89CEC9542
  </PipeId>
  <Cached> true or false </Cached>
  <Found> True or false </Found>
</ jxta: PipeBindingResponse >
```

# Peer Discovery Protocol

- Is used to discover any **published** peer resources
- **DiscoveryService** does the job
- PDP uses
  - IP broadcast in the local network
  - Rendezvous peers in the WAN

# Peer Discovery Protocol (2)

```
<?xml version="1.0"?>
<jxta: DiscoveryQuery>
  <Type> AdvertisementType </Type>
  <PeerAdv> PeerAdv of the Requestor </PeerAdv>
  <Attr> Query attribute </Attr>
  <Value>
    Value desired for the attribute
  <Value>
</jxta:DiscoveryQuery>
```

# Peer Discovery Protocol (3)

```
<?xml version="1.0"?>
<jxta:PA xmlns:jxta="http://jxta.org">
  <PID>
    urn:jxta:uuid-59616261646162614A78746150325033
    AF808359DC3C4D888279B6A10496C52403
  </PID>
  <GID> urn:jxta:jxta-WorldGroup </GID>
  <Name> Mr. Baker </Name>
  <Svc>
    <MCID> urn:jxta:uuid-DEADBEEFDEAFBABAFAFEEDBABE0000000805 </MCID>
    <Parm>
      <Addr> tcp://169.254.216.13:9705/ </Addr>
      <Addr>
        jxta://uuid-59616261646162614A78746150325033
        AF808359DC3C4D888279B6A10496C52403/
      </Addr>
    </Parm>
  </Svc>
</jxta:PA>
```

# Peer Information Protocol

- Is used to obtain **status information** of a remote peer such as
  - Uptime
  - PeerAdvertisement
- Allows for wide use in **commercial (enabling billing)** and internal JXTA applications
  - Determining the usage of the service and bill the service consumers for their use
  - Re-routing of the traffic according to the node's behavior/availability
- Response can be **full or** just an **acknowledgement** (alive/not alive)

# Peer Information Protocol (2)

```
<?xml version="1.0"?>
<jxta: Ping>
  <Credential> Credential </Credntial>
  <SourcePid> Peer ID </ SourcePid >
  <TargetPid> Peer ID </TargetPid>
  <Option>
    Full or Acknowledgement
  <Option>
</jxta:Ping>
```

# Peer Information Protocol (3)

```
<?xml version="1.0"?>
<jxta: PeerInfo>
  <Credential> Credential </Credntial>
  <SourcePid> Peer ID </SourcePid >
  <TargetPid> Peer ID </TargetPid>
  <Uptime> Uptime </Uptime>
  <TimeStamp> TimeStamp </TimeStamp>
  <PeerAdv> The whole Peer Advertisement </PeerAdv>
</jxta:PeerInfo>
```

# Peer Rendezvous Protocol

- Is responsible for **propagating messages** within a Peer Group
- Defines a simple protocol that allows for
  - Peers to connect to a service
  - Control the propagation of the message (TTL, loopback etc)

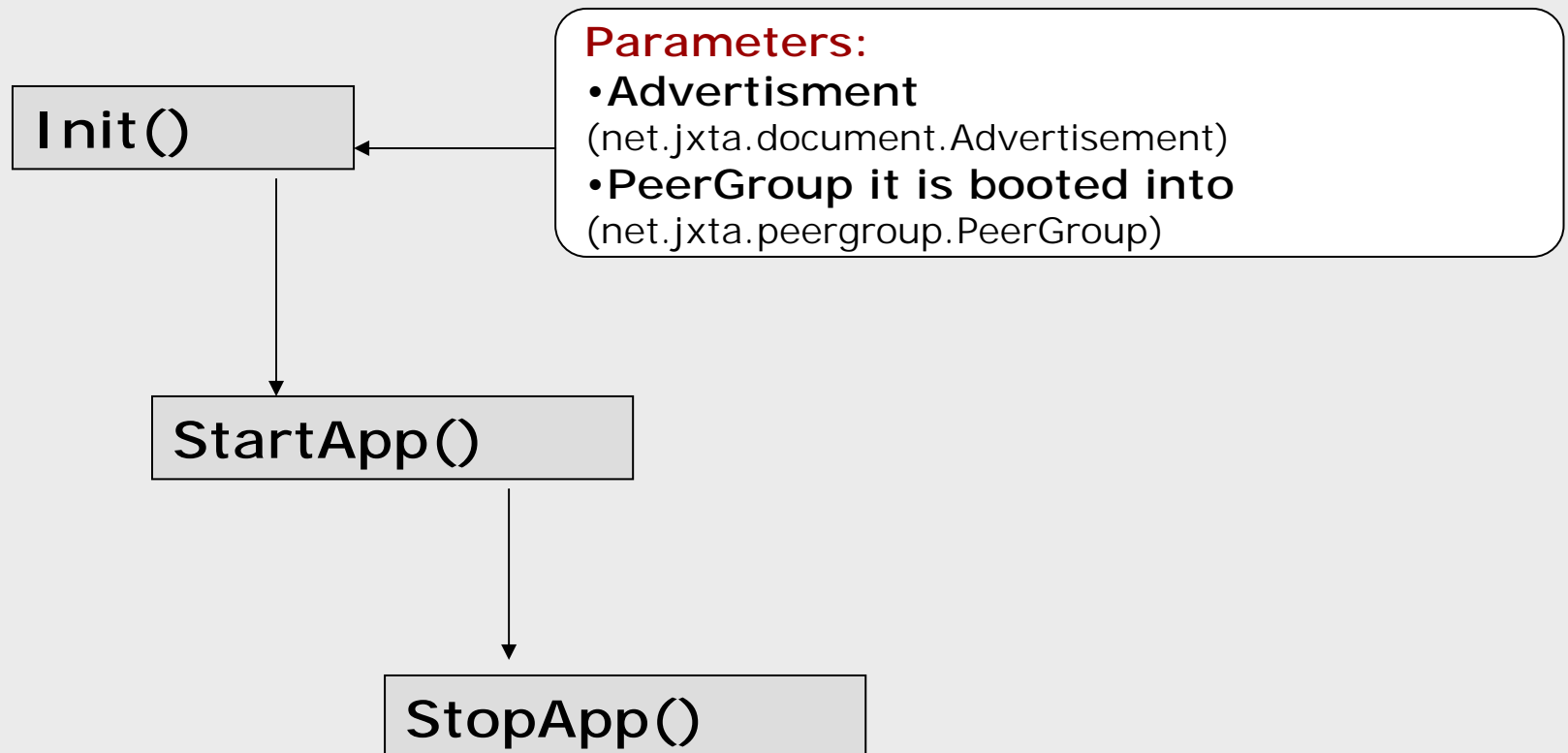
# Demo

---

# Creating An Application

---

# Service life cycle



# Starting Your App

- Starting the platform from your App

```
import net.jxta.platform.Application;  
public class myJXTAApp implements Application { }
```

- If you want to control the boot process

```
import net.jxta.peergroup.PeerGroup;  
import net.jxta.exception.PeerGroupException;  
import net.jxta.peergroup.PeerGroupFactory;  
try {  
    netPeerGroup = PeerGroupFactory.newNetPeerGroup();  
} catch (PeerGroupException e) { }
```

# Initializing Discovery

## ■ Starting DiscoveryService

```
discoveryService = netPeerGroup.getDiscoveryService();
```

## ■ Publishing advertisements locally

```
try {  
    discoveryService.publish(peerAdv,  
                             DiscoveryService.PEER, -1, 50 * 1000);  
    discoveryService.remotePublish(peerAdv,  
                                   DiscoveryService.PEER);  
} catch (IOException e) { }
```

# Discovering Resources

## ■ Getting advertisements from the local

```
try {  
  
enum=discoveryService.getLocalAdvertisements(discovery.PEER,  
                                             null, null);  
} // when nothing is stored  
  catch (Exception e) {}
```

## ■ Remote Discovery

```
try {  
    enum= discoveryService.getRemoteAdvertisements(null,  
                                                  discovery.PEER, Attr, Val, Threshold);  
} // when nothing is stored  
  catch (Exception e) {}
```

# Initializing the Input Pipe

- With the help of the Pipe Service

```
inputPipeAdv = (PipeAdvertisement)
    AdvertisementFactory.newAdvertisement(
        PipeAdvertisement.getAdvertisementType());
inputPipeAdv.setName((peerAdv.getID()).toString());
inputPipeAdv.setPipeID(IDFactory.newPipeID
    (netPeerGroup.getPeerGroupID()));
inputPipeAdv.setType(PipeService.UnicastType);
try{
    pipeSvc.createInputPipe(inputPipeAdv,
        new PipeMsgListener());
} catch (Exception e) {}
```

# Creating an Output Pipe

- Creating an output pipe with the Pipe Service

```
try {  
    pipesvc.createOutputPipe(foundInputPipeAdv,  
                             outputPipeListener);  
} catch (IOException e)
```

- We need two things
  - PipeAdv of the input pipe of the peer on the other side
  - Listener that listens for responses

# Sending a Message

- 5 steps to send a message to a specific peer
  - 1 step: find an advertisement of the input pipe of another peer
  - 2 step: send a propagate message
  - 3 step: generate a message, attach the content

```
Message msg = createNewMessage();
```

- 4 step: create an output pipe connected to the input pipe of another peer
- 5 step: send a message

# Thanks for your attention

Any questions?



Download the slides from  
[www.drwieland.de](http://www.drwieland.de)