

Computer Network Programming

## Indirect Communication

Dr. Petru Florin Mihancea

V20180301

1

# The Problem

## Why?



### Space coupling

Communication is directed towards a given receiver(s)

### Time coupling

Sender and receiver(s) exist at the same moment of time

Dr. Pavan Kumar Bhattacharya

## Why?



### Space uncoupling

Sender dose not know the receiver(s) identity

### Time uncoupling

Sender and receiver(s) have independent lifetimes

Many approaches: Group communication, Publish-Subscribe, Message Queues, Distributed Shared Memory, etc.

Dr. Péter Pázmány Pálffy

2

## **Indirect Communication Approaches**

Dr. Petru Florin Mihăncă

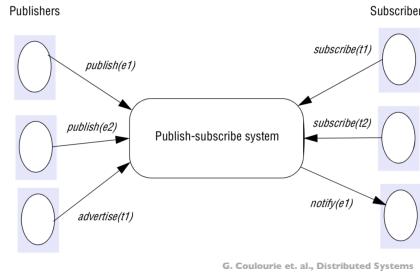
# A

## **Publish-Subscribe Systems**

**Distributed Event-Based**

Dr. Péter Pázmány Pálffy

## Basic Functions



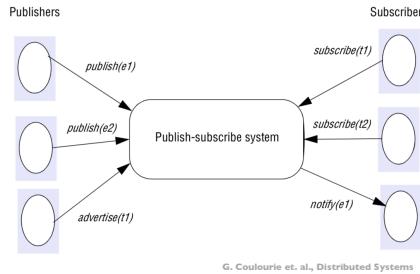
G. Coulouris et. al., Distributed Systems

### Publishers

- announce an event via **publish(e)** operation

Dr. Punit Parikh Bhavnani

## Basic Functions



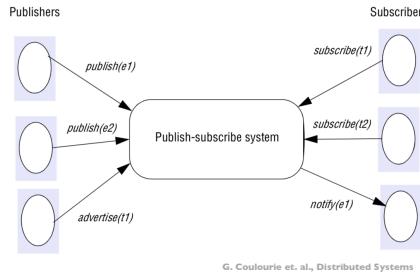
G. Coulouris et. al., Distributed Systems

### Subscribers

- express interest in some events via **subscribe(*t*)** operation
  - *t* is a **filter** used to specify the events of interest
- get notified via **notify(*e*)** when an event of interest occurs

Dr. Paula Flores Pintorres

## Basic Functions



G. Coulouris et. al., Distributed Systems

## Extra elements

- **unsubscribe(t)** revoking subscriber interest in an event
- **advertise(t)** enables a publisher to declare the kinds of events it produces

Dr. Paula Flores Pintorres

## Specifying Events of Interest

### Channel-based

Subscribe to a **named channel** and get all events sent to it

### Topic-based

Subscribe to a **topic** and an event **explicitly specifies its topic**

### Content-based

Subscribe with a **query** specifying combined event attributes

### Type-based

Filter based on the **types** of the event **objects**

Other: Observe object state change, context (e.g. location), complex event patterns in time

Dr. Paul Perna Miltzoucas

## Publish-Subscribe Implementations

### Architectures

#### Centralized

A node acting as the central event broker

#### Network of brokers

Several nodes cooperate just to manage the events

#### Peer-to-peer

No distinction between publisher, subscriber and broker

### Event routing in distributed implementations

Flooding, Filtering, Rendezvous

### Examples

**OpenJMS** [<http://openjms.sourceforge.net/>]

Included in application servers

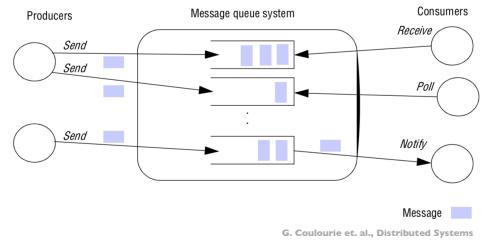
**GlassFish** [<https://javaee.github.io/glassfish/>]

B

# Message Queues

Dr. Petru Florin Mihăncăea

## Basic Functions

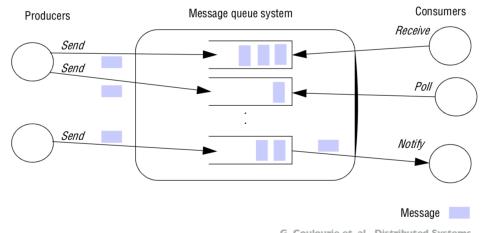


### Producers

- add messages to a specific **queue** via the **send** operation

Dr. Péter Pázmány Pálffy

## Basic Functions



G. Coulouris et. al., Distributed Systems

## Consumers

- consumes messages from a queue via
  - blocking **receive** operation
  - non-blocking **poll** operation
  - notification operation

Dr. Pavan Parimi Bhattacharya

## **Additional Features**

### **Queue policy**

Simple FIFO, Priority queues, Properties-based selection

### **Message persistency**

#### **Reliable delivery**

Messages will be delivered once but we do not know when

### **Other**

Transactions and transformations

Dr. Péter Pázmány Pálffy

## Message Queues Implementations

### Architectures

#### Centralized

A node acting as the central message queue manager

#### Distributed

Cooperating message queue managers

### Examples

OpenJMS, WebSphereMQ

Included in application servers

GlassFish [https://javaee.github.io/glassfish/]

Dr. Peter Perner Pfeiffer

3

# **Java Message Service (JMS) Middleware**

Dr. Petru Florin Mihăncă

## JMS API

Allows applications to create, send and receive messages to/from:

**queues** (message queues systems)

**topics** (publish-subscribe systems)

### JMS Provider

an implementation of this API like

**OpenJMS, GlassFish, etc.**

Dr. Péter Pázmány Pálffy

## Prepare Glassfish JMS Provider

### 1. Download it and unpack

<https://javaee.github.io/glassfish/download>  
or the prepared packet from the lab page :)

### 2. In a terminal, go to the bin folder and run

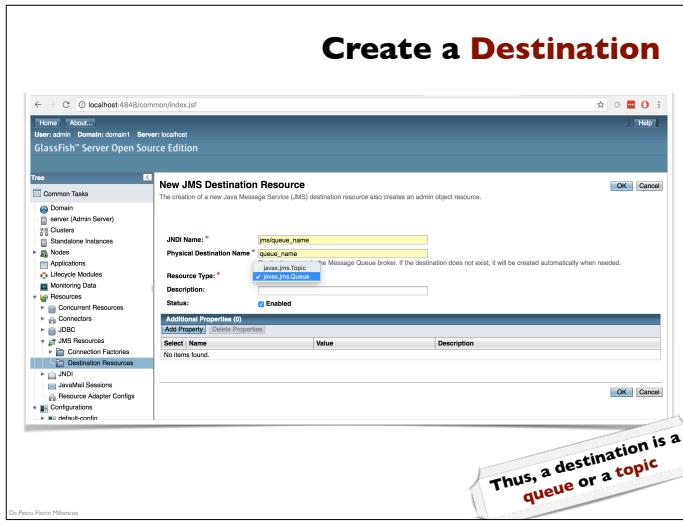
[sh] asadmin start-domain --verbose

### 3. In a browser, go to

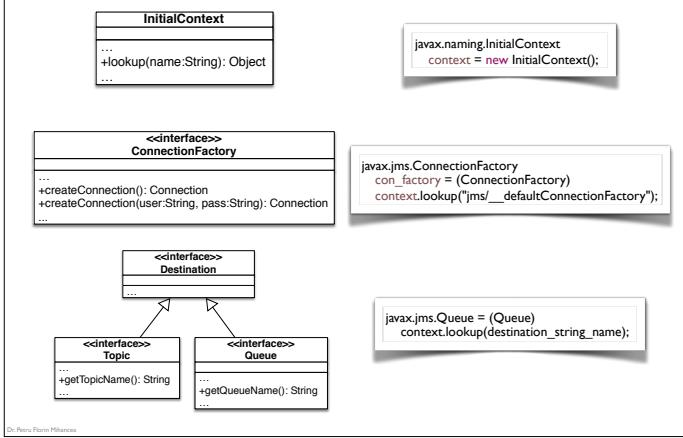
<http://localhost:4848/>

## Prepare Glassfish JMS Provider

The screenshot shows the GlassFish Server Open Source Edition console interface. At the top, there is a header bar with links for Home, About, User: admin, Domain: domain, and Server: localhost. Below the header is a title bar "GlassFish® Server Open Source Edition". The main area is titled "GlassFish Console - Common Tasks". On the left, there is a sidebar titled "Common Tasks" with several categories: Domain, Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources, and Configurations. Under Resources, there are sub-categories: Container Resources, Connectors, JDBC, JNDI, JavaMail Sessions, and JMS Resources. Under JMS Resources, there is a sub-category: JMS Adapter Configs. Under Configurations, there are sub-categories: default-config and server-config. The right side of the screen displays sections for GlassFish News, Deployment (List Deployed Applications, Deploy an Application), Administration (Change Administrator Password, List Password Aliases), Documentation (Open Source Edition Documentation Set, Quick Start Guide, Administration Guide, Application Development Guide, Application Deployment Guide), and Resources (Create New JDBC Resource, Create New JDBC Connection Pool). At the bottom left, there is a link "Go Back From Preferences".



## Locating Relevant Objects



## Connection and Session

```
<<interface>>
Connection

+createSession(int sessionMode): Session
...
+start(): void
+stop(): void
...
+close(): void
```

```
javax.jms.Connection con = con_factory.createConnection();
javax.jms.Session ses =
    con.createSession(JMSContext.AUTO_ACKNOWLEDGE);
...
con.start();
...
con.close();
```

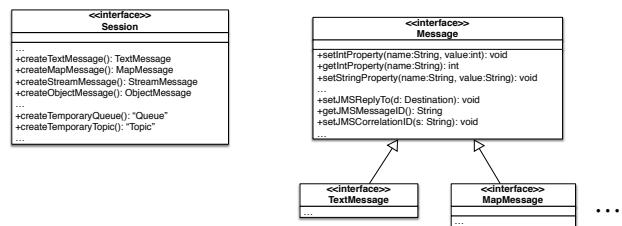
**Other JMSContext constants  
may be used to require  
transacted sessions, explicit  
client ACK, etc.**

```
<<interface>>
Session

...
+createTextMessage(): TextMessage
+createMapMessage(): MapMessage
+createStreamMessage(): StreamMessage
+createObjectMessage(): ObjectMessage
...
+createTemporaryQueue(): "Queue"
+createTemporaryTopic(): "Topic"
...
+createProducer(d.Destination): MessageProducer
+createConsumer(d.Destination): MessageConsumer
...
+createConsumer(d.Destination, selector:String): MessageConsumer
+createDurableSubscriber(c.Topic, name:String, selector:String, noLocal:boolean): "MessageConsumer"
...
```

On Paris Paris Mihai

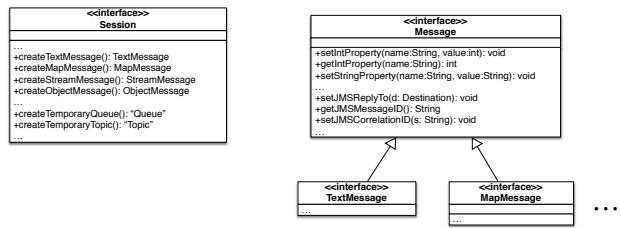
## Messages



### Message properties

**key-value pairs enabling message filtering**

## Messages



### Temporary destinations

a way of getting back an answer

`messageID` and `correlationID` are usually used to pair the request/reply messages

## Consumers and Producers

```
<<interface>>
Session

+createProducer(d:Destination): MessageProducer
+createConsumer(d:Destination): MessageConsumer
...
+createConsumer(d:Destination, selector:String): MessageConsumer
+createDurableSubscriber(c:Topic, name:String, selector:String, noLocal:boolean): "MessageConsumer"
...
```

```
<<interface>>
MessageProducer

+send(m:Message): void
+setPriority(p:int): void
+close(): void
...
```

**Sends messages to the associated destination  
priorities can be included**

```
<<interface>>
MessageConsumer

+receive(): Message
+setMessageListener(l: MessageListener): void
+close(): void
...
```

**Gets messages from the associated destination  
synchronously** (blocking receive)  
**asynchronously** (the listener is called by the JMS  
infrastructure when a message is available)

## Consumers and Producers

```
<<interface>>
Session

+createProducer(d:Destination): MessageProducer
+createConsumer(d:Destination): MessageConsumer
...
+createConsumer(d:Destination, selector:String): MessageConsumer
+createDurableSubscriber(c:Topic, name:String, selector:String, noLocal:boolean): "MessageConsumer"
...
```

```
<<interface>>
MessageProducer

+send(m:Message): void
+selPriority(p:int): void
+close(): void
...
```

Sends messages to the associated destination  
priorities can be included

```
<<interface>>
MessageConsumer

+receive(): Message
+setMessageListener(l: MessageListener): void
+close(): void
...
```

Gets messages from the associated destination  
synchronously or asynchronously  
depending on the infrastructure when it is used

selectors - SQL92 string over  
properties for message filtering  
durable - the named consumer will get the topic  
messages when it is back "online"

Dr. Paul Poirier

# 4

## Demo Application

Same application but the server waits for requests set in a **message queue** and answers to the client via a **temporary queue**

Dr. Péter Pázmány Pálffy