Computer Network Programming

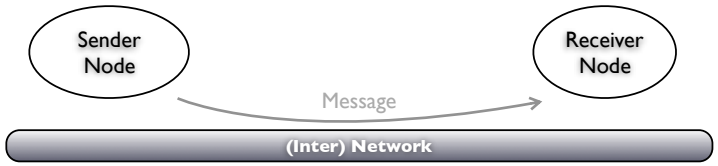
Indirect Communication

Dr. Petru Florin Mihancea

1

The Problem

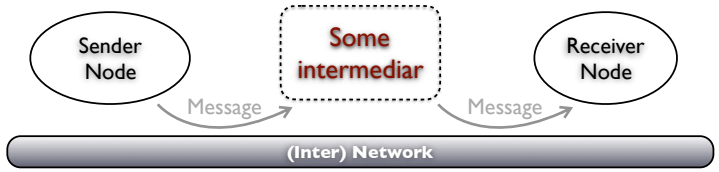
Why?



Space coupling
Communication is directed towards a given receiver(s)

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

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.

2

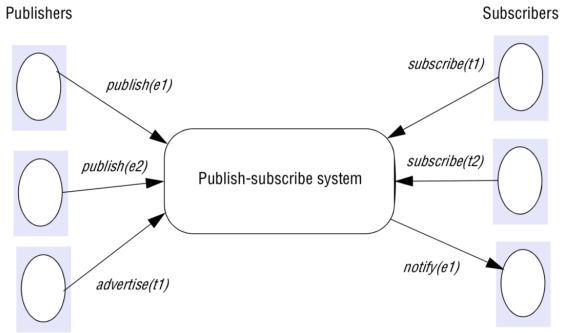
Indirect Communication Approaches

A

Publish-Subscribe Systems

Distributed Event-Based

Basic Functions

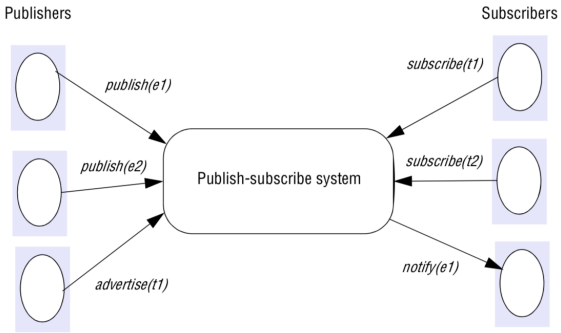


G. Coulourie et. al., Distributed Systems

Publishers

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

Basic Functions

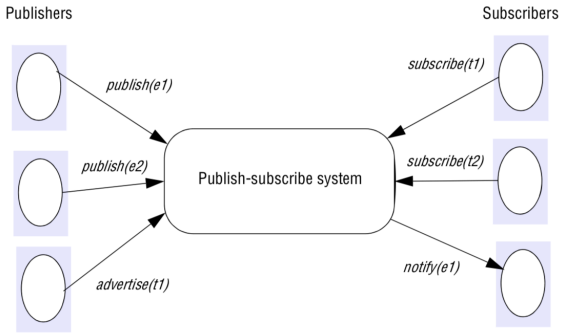


G. Coulourie et. al., Distributed Systems

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

Basic Functions



G. Coulourie 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

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

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

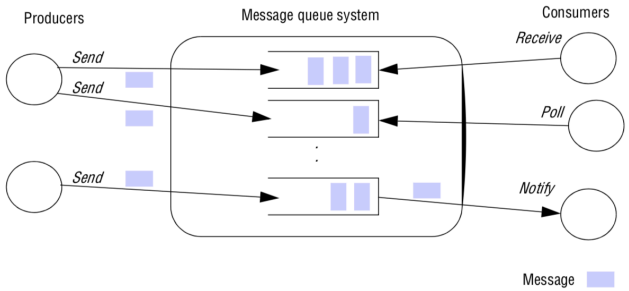
Examples

OpenJMS [http://openjms.sourceforge.net/]
Included in application servers
GlassFish [https://javaee.github.io/glassfish/]

B

Message Queues

Basic Functions

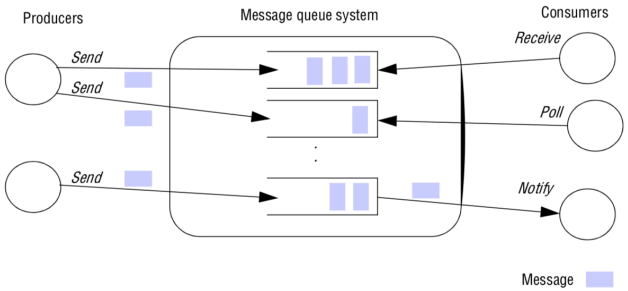


G. Coulourie et. al., Distributed Systems

Producers

- add messages to a specific queue via the send operation

Basic Functions



G. Coulourie et. al., Distributed Systems

Consumers

- consumes messages form a queue via
  - blocking receive operation
  - non-blocking pool operation
  - notification operation

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

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/]

3

Java Message Service (JMS)
Middleware

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.

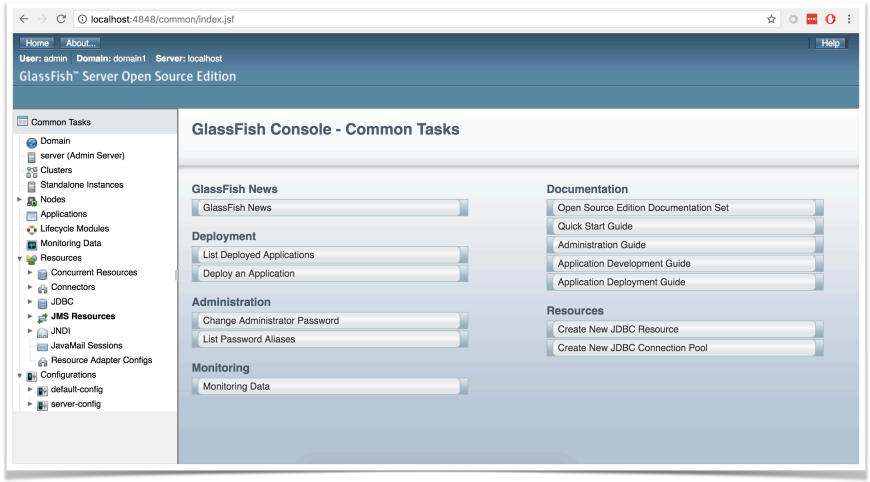
Prepare Glassfish JMS Provider

1. **Download it and unpack**
   https://javaee.github.io/glassfish/download
   or the prepared packet from the lab page :)

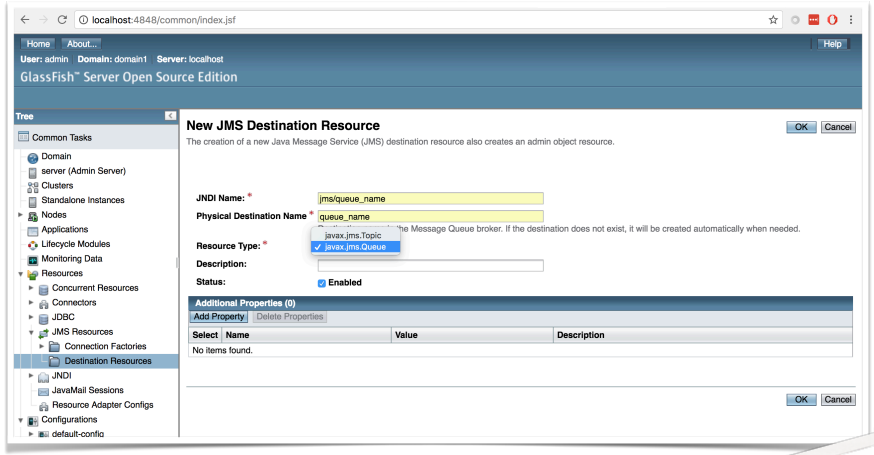
2. **In a terminal, go to the bin folder and run**
   [sh] asadmin start-domain —verbose

3. **In a browser, go to**
   http://localhost:4848/

Prepare Glassfish JMS Provider

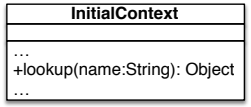


Create a Destination

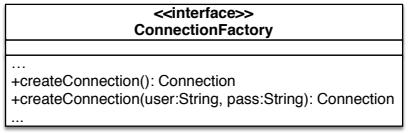


Thus, a destination is a queue or a topic

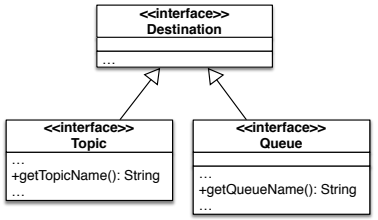
Locating Relevant Objects




```
javax.naming.InitialContext
    context = new InitialContext();
```

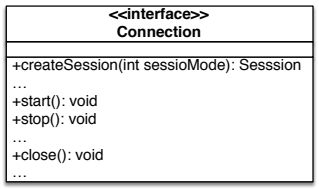



```
javax.jms.ConnectionFactory
    con_factory = (ConnectionFactory)
    context.lookup("jms/__defaultConnectionFactory");
```



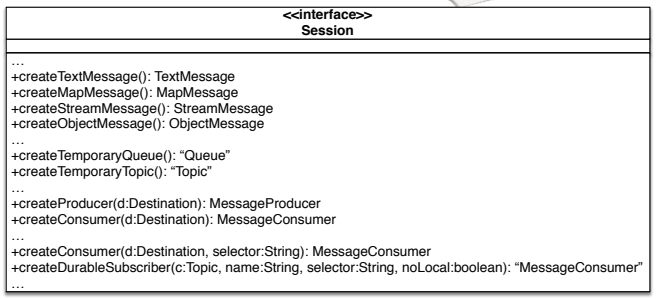

```
javax.jms.Queue = (Queue)
    context.lookup(destination_string_name);
```


Connection and Session

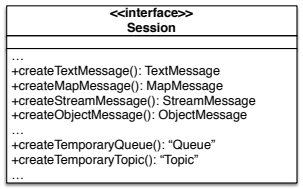


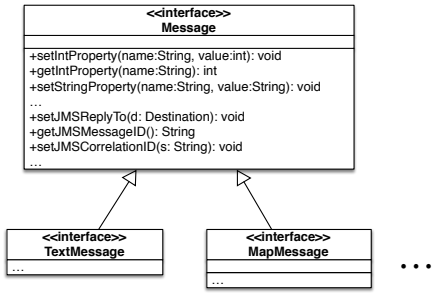

```
javax.jms.Connection con = con_factory.createConnection();
javax.jms.Session ses =
    con.createSession(JMSContext.AUTO_ACKNOWLEDGE);
…
con.start();
…
con.close();
```

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



Messages



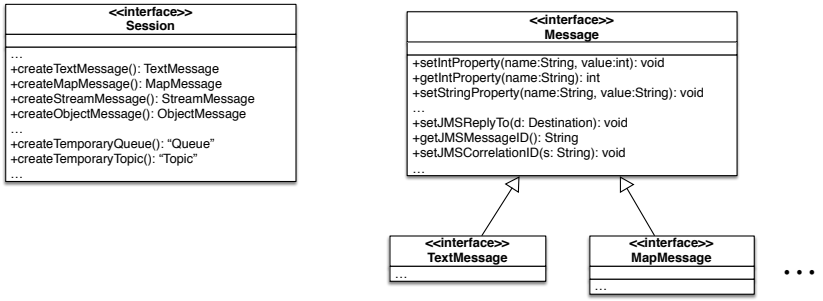


. . .

Message properties

key-value pairs enabling message filtering

Messages

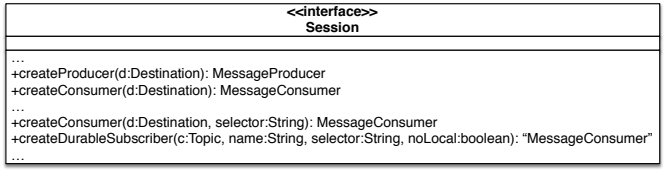


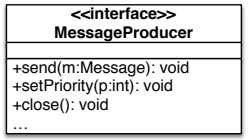
Temporary destinations

a way of getting back an answer

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

Consumers and Producers

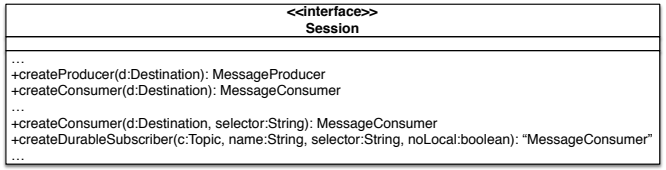


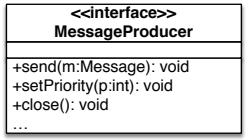


Sends messages to the associated destination
priorities can be included

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





Sends messages to the associated destination
priorities can be included

Gets messages f
synchronously
asynchronously
infrastructure when

selectors - SQL92 string over properties for message filtering

durable - the named consumer will get the topic messages when it is back "online"

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