Open Message Queue

mq.dev.java.net

Dave Whitla
Technical Architect
Wotif.com
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
Agenda

• Architectural forces

• The Java Message Service

• What is Open Message Queue?

• How do I develop with Open Message Queue?

• The Wotif.com experience
As applications grow they need to evolve

The Monolith

Absence of Discrete Components

Performance Issues

Central Bottleneck

Scalability Problems
Application Evolution

Message Oriented Middleware

• An “event-driven” architecture
• A shared, centralised event notification service
• Reduces unnecessary “polling” communication
• Reduces instance, API, performance and availability coupling
• Components are coupled only to the messaging domain
Agenda

• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
• Architectural forces

• The Java Message Service

• What is Open Message Queue?

• How do I develop with Open Message Queue?

• The Wotif.com experience
The Java Message Service

• Originally developed to allow Java access to existing systems
• Now widely adopted by existing MOM vendors

Core concepts:

• Provision of routing and delivery services
• Support for point-to-point and publish-subscribe patterns
• Synchronous and asynchronous message receipt
• Support for reliability assurance
• Built-in support for common existing message formats
The JMS Provider

JMS Providers must “provide” the following:

• Client libraries that implement the JMS interfaces
• Functionality for routing and delivery of messages
• Administrative tools for management, monitoring and tuning
• Lowest common denominator of MOM features
• Providers typically cannot communicate directly with each other
JMS Messaging Objects
- Connection
- Session
- Producer / Consumer
- Destination
- Message

JMS Messaging Domains
- Point-to-Point (Queues)
- Publish-Subscribe (Topics)
JMS Messaging Objects

• Connection
• Session
• Producer / Consumer
• Destination
• Message

JMS Messaging Domains

• Point-to-Point (Queues)
• Publish-Subscribe (Topics)
Point-to-point Messaging

Senders produce messages to
Queues from which
Receivers consume
Senders produce messages to Queues from which Receivers consume
Point-to-point Messaging

- Multiple Senders per Queue
- Multiple Receivers per Queue (OpenMQ extension)
- No send-receive timing dependency
Publishers produce messages to Topics from which Subscribers may consume after they have subscribed.
Publishers produce messages to Topics from which Subscribers may consume after they have subscribed.
Publish-subscribe Messaging

- More than one producer possible per topic
- More than one subscriber possible per topic
- All subscribers receive all messages
- Send-receive timing dependency
- Durability
- Broadcast
**Common & Unified Domain**

- Messages can be consumed synchronously or asynchronously
- Consumers can filter which messages they receive
- Messages are placed in destinations in sent order
- Message consumption order cannot be guaranteed

<table>
<thead>
<tr>
<th>Base Type (Unified Domain)</th>
<th>Point-to-Point Domain</th>
<th>Publish-Subscribe Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>Queue</td>
<td>Topic</td>
</tr>
<tr>
<td>ConnectionFactory</td>
<td>QueueConnectionFactory</td>
<td>TopicConnectionFactory</td>
</tr>
<tr>
<td>Connection</td>
<td>QueueConnection</td>
<td>TopicConnection</td>
</tr>
<tr>
<td>Session</td>
<td>QueueSession</td>
<td>TopicSession</td>
</tr>
<tr>
<td>MessageProducer</td>
<td>QueueSender</td>
<td>TopicPublisher</td>
</tr>
<tr>
<td>MessageConsumer</td>
<td>QueueReceiver</td>
<td>TopicSubscriber</td>
</tr>
</tbody>
</table>
Administered Objects

- JMS scheme is extensible
- Need for vendor portability of JMS object references
- Two fundamental objects which vary in construction requirements from one vendor to the next:
  - Connections (or rather their factories)
  - Destinations
Administered Objects
Administered Objects provide clear benefits:

- Administrators can tune messaging performance globally by reconfiguring these objects. **No code changes necessary.**

- Administrators can control destination proliferation on the broker.

- Developers can catch programming errors early which might otherwise silently create an incorrect destination.

- They shield developers from vendor-specific provider details maintaining vendor portability **without code changes.**
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
What is Open Message Queue?

• A Java Message Oriented Middleware server
• A complete JMS provider implementation
• Provides a reference implementation of the JMS APIs
• OpenMQ IS Sun Java System Message Queue
• A member of the GlassFish community
• “Stand-alone” or embedded within GlassFish
• Completely open source software with a strong community
• Project home at mq.dev.java.net
Licencing & Availability

- Complete source code
  - Dual licence CDDL and GPLv2 like GlassFish
- Stable binaries and source are available for each release
  - Since version 4.0 / GlassFish V1
  - Today 4.1 / GlassFish V2
  - Promoted builds (unstable) of next generation 4.2
OpenMQ is composed of 3 major elements

- Messaging Server
  - known as a “broker”

- Client Libraries
  - Java language runtime
  - C language runtime
  - JCA 1.5 adaptor for JEE containers

- Administration Tools
  - Command-line tools
  - GUI tool
  - JMX API
Broker features:
• The full JMS specification
• Clustering / load-balanced and failover (HA service)
• Dead Message Queue
• SOAP over HTTP, SOAP over JMS, SSLJMS and TLS transport
• Multiple Queue Receiver extension
• No-Acknowledge extension
• Message body compression and encryption
• Message store - file or JDBC - for guaranteed delivery
Client features:

• Java and C client libraries
• JCA 1.5 resource adaptor
• Client runtime logging
• Connection event notification
Administration features:

- Destination consumer limits
- Destination message count limits
- Quiesce destinations and/or broker for managed hot upgrades
- JMX API - remote programmatic management and monitoring
- Pure Java GUI (uses JMX)
- Comprehensive command-line tools
New in version 4.1:

- New installer built from OpenInstaller
- HA service AND data with HA storage
- Pluggable JAAS authentication
- Improved deployment and performance in GlassFish
Scaling & Redundancy

- Vertical Scaling
- Stateless Horizontal Scaling
  - Stateless Redundancy (service failover)
- Stateful Horizontal Scaling
  - Conventional Clustering (service availability)
  - HA Clustering (service + data availability)
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
Agenda

• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
### Producing and Consuming

<table>
<thead>
<tr>
<th>Producing a Message</th>
<th>Consuming a Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Administrator creates a ConnectionFactory administered object</td>
<td></td>
</tr>
<tr>
<td>2. Administrator creates a physical destination and the administered object reference to it</td>
<td></td>
</tr>
<tr>
<td>3. Client obtains a ConnectionFactory instance through a JNDI lookup</td>
<td></td>
</tr>
<tr>
<td>4. Client obtains a Destination instance through a JNDI lookup</td>
<td></td>
</tr>
<tr>
<td>5. Client uses the ConnectionFactory to create a Connection to the broker (sets properties)</td>
<td></td>
</tr>
<tr>
<td>6. Client uses the Connection to create a Session and sets properties for message reliability</td>
<td></td>
</tr>
<tr>
<td>7. Client uses the Session to create a MessageProducer</td>
<td>Client creates a MessageConsumer</td>
</tr>
<tr>
<td>8. Client uses the Session to create a Message</td>
<td>Client starts the Connection</td>
</tr>
<tr>
<td>9. Client uses the Session to send the Message</td>
<td>Client receives the Message</td>
</tr>
</tbody>
</table>
Message Consumption

3 factors affect how the broker delivers messages to a consumer

• Synchronous / Asynchronous

• Property Selector Filtering (eg colour = “red” or size > 10)

• Subscription Durability (Topics only)
Reliable Messaging

- Message delivery occurs in two separate steps
- Messages have 3 opportunities for loss
- Reliable delivery only applies to persistent messages
- Two mechanisms for ensuring reliable delivery
  - Persistent message storage
  - Acknowledgements / Transactions
Acknowledgements

- Sent between client and broker

Production
- Message received, placed in destination and persisted
- send() blocks

Consumption
- AUTO_ACKNOWLEDGE
- CLIENT_ACKNOWLEDGE
- DUPS_OK_ACKNOWLEDGE
- NO_ACKNOWLEDGE (OpenMQ extension)
Transactions

- Groups production and/or consumption of one or more messages into an atomic unit
- Applies to a single Session
- Therefore not end-to-end
- End-to-end requires distributed transactions (JTA)
  - Requires a distributed transaction manager (GlassFish)
  - Cover multiple XA resources using a two-phase commit
Message Lifecycle
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
• Architectural forces
• The Java Message Service
• What is Open Message Queue?
• How do I develop with Open Message Queue?
• The Wotif.com experience
Application evolving - messaging is at the core of our architecture

ESB patterns rather than products

Features we were looking for:

- Active product support and development
- High performance and robustness
- JMX monitoring and administration
- High availability - migration path from HA service to HA data
- Standards support - JAAS authentication

Features we have made good use of:

- Scripting of the CLI tools for simple deployment
- Flexibility in redundancy implementation
Dive into OpenMQ now at

mq.dev.java.net
package somepackage;

import java.util.logging.Level;
import java.util.logging.Logger;
import java.annotation.PostConstruct;
import java.annotation.Resource;
import javax.ejb.MessageDriven;
import javax.ejb.MessageDrivenContext;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.ejb.ActivationConfigProperty;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

@MessageDriven(
    name = "SomeMDB", mappedName = "jms/SomeTopic",
    activationConfig = {
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Topic"),
        @ActivationConfigProperty(propertyName = "acknowledgeMode", propertyValue = "Auto-acknowledge"),
        @ActivationConfigProperty(propertyName = "messageSelector", propertyValue = "Colour = 'red'"),
        @ActivationConfigProperty(propertyName = "subscriptionDurability", propertyValue = "Durable")
    }
)
public class SomeMDB implements MessageListener {

    private static final Logger LOGGER = Logger.getLogger(SomeMDB.class.getName());

    @Resource
    private MessageDrivenContext ejbContext;

    @EJB
    private SomeService someService;

    private MessageHandler handler;

    @PostConstruct
    public void postConstruct() {
        handler = new MessageHandler(someService);
    }

    @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
    public void onMessage(Message jmsMessage) {
        try {
            handler.handleMessage((TextMessage) jmsMessage);
        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Message could not be processed, message has been swallowed, please resend the message", e);
            // Need to explicitly acknowledge the message, as the transaction rollback seems to stop it.
            try {
                jmsMessage.acknowledge();
            } catch (JMSException jmse) {
                // ignore
            }
        }
    }
}

Message Consumer Example

```xml
<enterprise-beans>
  <message-driven>
    <ejb-name>RelayMDB</ejb-name>
    <mapped-name>jms/InputTopic</mapped-name>
    <resource-ref>
      <res-ref-name>jms/RemoteConnectionFactory</res-ref-name>
      <res-type>javax.jms.ConnectionFactory</res-type>
    </resource-ref>
    <mapped-name>jms/RemoteConnectionFactory</mapped-name>
    <injection-target>
      <injection-target-class>com.wotif.bogus.RelayMDB</injection-target-class>
      <injection-target-name>jmsConnectionFactory</injection-target-name>
    </injection-target>
    <message-destination-ref>
      <message-destination-ref-name>jms/OutputTopic</message-destination-ref-name>
      <message-destination-type>javax.jms.Topic</message-destination-type>
      <message-destination-usage>Produces</message-destination-usage>
    </message-destination-ref>
    <mapped-name>jms/SomeTopic</mapped-name>
    <injection-target>
      <injection-target-class>com.wotif.bogus.RelayMDB</injection-target-class>
      <injection-target-name>outputTopic</injection-target-name>
    </injection-target>
  </message-driven>
</enterprise-beans>
```