Financial Trading Systems Using JMS and Message-Oriented Middleware

by Yakov Fain

www.weekendwithexperts.com

Philly JUG, October 25, 2005

www.weekendwithexperts.com
About Myself

- J2EE Architect and developer
- Creator of the seminars “Weekend With Experts”
- Enterprise Java editor at Java Developer’s Journal
- Author of Java books and articles
- Part-time professor at New York University
Agenda

- A brief overview of the stock trading process
- Message Oriented Middleware (MOM)
- Basics of Java Messaging Service
- Java JMS clients and Message-Driven Beans (MDB)
- Multithreading
- Introducing LDAP servers
- Potential issues
A Trading System
Wall Street

- Stock Exchanges (NYSE, NASDAQ, American Stock Exchange...)

- Brokerage houses, trading floor, specialists
Stock Purchase Order

- Who can place an order to buy/sell a stock? A broker, a day trader… I’ve done that as well 😊.
- GUI/Web interface
- Market, Limit, GTC orders
- New, Mod, Cancel orders. Slicing
- Ask and Bid prices
- Front/Middle/Back Office systems: trading/profit & loss, risk management/ settlements, record keeping, regulatory compliance
Order’s Life Cycle

- Create a new order in your database
- Send an order to the stock exchange
- Receive execution(s) or rejection(s)
- Process modifications/cancels of some orders
- Report a trade to middle/back office applications
- The order has to go through several applications on written in different languages on different platforms.
Faster, Faster, Faster!

- Round-trip metrics
- A stuck message is a liability
- High Availability
- Scalability
- Integration with other (legacy) systems
MOM Servers
Message-Oriented Middleware

- MOM allows to connect various applications in a loosely coupled fashion with minimum programming required.
- MOM is similar to a post office.
- Guaranteed delivery (is not always needed).
- Asynchronous processing: applications do not rely on availability of other applications.
- Messaging helps in building open ended, service-oriented and loosely coupled systems.
Major MOM Providers

- WebSphere MQ (former MQ Series), IBM
- Tibco Rendezvous, Tibco
- FioranoMQ, Fiorano Software
- Sonic MQ, Sonic Software
- MSMQ, Microsoft

Messaging bridges come handy in cases like mergers of two companies, i.e.: A trading system that uses WebSphere MQ can arrange money transfer from a banking application that uses Tibco.

WebLogic has a concept of foreign JMS servers.

Sonic MQ has a bridge to WebSphere MQ...
Connecting Applications with MOM
More MOM benefits

- You do not need to write a custom software to connect two applications.
- If one of the applications is not available, MOM stores messages.
- MOM provides clustering, load balancing and fail-over without additional programming.
Messaging Buzzwords

- Message producers and consumers, simple clients
- Point-to-Point (P2P), Queues, Senders
- Channels, Queue Managers
- Publish-Subscribe (Pub/Sub), Topics, Publishers, Subscribers, Durable Subscribers
- Persisted messages are stored on disks while non-persisted messages stay in memory.
Point-to-Point Model

- There is only one receiver for any particular message (i.e. a sender places an order and a receiver gets it).
- A sender is called a message **producer** and a receiver is a message **consumer**.
- The message is removed from a queue as soon as the receiver successfully gets it.
Publish-Subscribe Model

- A message **is published** to a topic and multiple receivers can get it (i.e. stock price quotes)
- Receivers **subscribe** to a topic.
- Publishers and subscribers are connected by a message broker.
Java Messaging Service (JMS)

- JMS is an API to MOM. Message producers and consumers can use non-JMS APIs.
- Synchronous message retrieval.
- JMS Listeners.
- J2EE senders: Session Beans.
  *Concurrency, transaction support, easy configuration, they can send messages to any topic or queue*
- J2EE receivers: Message-Driven Beans (MDB).
  *Concurrency, transaction support, easy configuration, they are mapped to a particular topic*
Message-Driven Beans

- MDB are stateless, no remote and home interfaces

```java
public class OrderListener implements MessageDrivenBean, MessageListener {

    MessageDrivenContext ctx;
    public OrderListener() {}
    public void onMessage(Message message) {
        // Your business code goes here.
    }

    // Bean Life supporting methods go here
}
```
<message-driven>
  <ejb-name>OrderListener</ejb-name>
  <ejb-class>com.xyz.OrderListener</ejb-class>
  <transaction-type>Container</transaction-type>
  <transaction-scope>Local</transaction-scope>
  <message-driven-destination>
    <jms-destination-type>javax.jms.Topic</jms-destination-type>
    <jms-subscription-durability>nondurable</jms-subscription-durability>
  </message-driven-destination>
...</message-driven>
Mapping of an MDB to a topic:

```xml
<weblogic-enterprise-bean>
    <ejb-name><b>OrderListener</b></ejb-name>
    <message-driven-descriptor>
        <destination-jndi-name><b>OrderTopic</b></destination-jndi-name>
    </message-driven-descriptor>
</weblogic-enterprise-bean>
```
Simplified Order Round Trip
Synch vs. Asynch

Synchronous call

Java Client → J2EE App Server
OrderSessionBean
createOrder()
sendToSE()
Orders table

Asynchronous call

Java/Cobol/VB/C++ Client → MOM Server
OrderQueue
OrderMDB
onMessage()
"osb-ctx.lookup();
osb.createOrder();"
J2EE App Server
OrderSessionBean
createOrder()
sendToSE()

www.weekendwithexperts.com
Synch vs. Asynch (cont.)

Synchronous call

Asynchronous call

www.weekendwithexperts.com
JMS Classes From java.jms

- Message
- Queue, QueueConnection, QueueConnectionFactory, QueueSession, QueueSender, QueueReceiver
- Topic, TopicConnection, TopicConnectionFactory, TopicSession, TopicPublisher, TopicSubscriber
JMS Message Types

- TextMessage – any Java String
- ObjectMessage – any serializable Java Object
- BytesMessage – a stream of bytes
- StreamMessage – a stream of Java primitives
- MapMessage – any key/value pair

An ObjectMessage or a MapMessage are good choices for a trading SOA application.
How to Send a Message

QueueConnectionFactory  factory = new
QueueConnectionFactory();
//QueueConnectionFactory factory=
//(QueueConnectionFactory) ctx.lookup("OrderQCF");

QueueConnection connection =
factory.createQueueConnection();
connection.start();
Session session = connection.createQueueSession(
false, Session.AUTO_ACKNOWLEDGE);
Queue ioQueue = session.createQueue( "OrderQueue" );
QueueSender queueSender =session.createSender(ioQueue);
TextMessage outMsg = session.createTextMessage();
outMsg.setText(“IBM 200 Mkt”); // Buy 200 shares of IBM
queueSender.send(outMsg);
queueSender.close();
How to Retrieve a Message

class MyReceiver implements MessageListener{
    MyReceiver(){
        QueueConnectionFactory  factory = new QueueConnectionFactory();
        QueueConnection connection = factory.createQueueConnection();
        connection.start();
        Session session = connection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
        Queue ioQueue = session.createQueue(”OrderQueue”);
        QueueReceiver queueReceiver = session.createReceiver(ioQueue);
        queueReceiver.setMessageListener(this);
    }
    public void onMessage(Message msg){
        String msgText;
        try{ if (msg instanceof TextMessage){
            msgText = ((TextMessage) msg).getText();
            System.out.println(”Got “ + msgText);
        }else System.out.println(”Got a non-text message”);
        }
    }
}

www.weekendwithexperts.com
How to Publish a Message

```
TopicConnectionFactory conFactory =
    (TopicConnectionFactory) ctx.lookup("ipoTCF");
TopicConnection connection =
    conFactory.createTopicConnection();
TopicSession pubSession =
    connection.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
Topic ipoTopic = (Topic)ctx.lookup("IPO_Alerts");
TopicPublisher publisher = pubSession.createPublisher(ipoTopic);
connection.start();
TextMessage message = pubSession.createTextMessage();
message.setText("Google’s IPO is scheduled for tomorrow");
publisher.publish(message);
```
How to Subscribe for a Topic

TopicSession subSession = connection.createTopicSession(false,
   Session.AUTO_ACKNOWLEDGE);
Topic topic = (Topic) ctx.lookup(“IPO_Alerts”);
TopicSubscriber subscriber = subSession.createSubscriber(topic);

connection.start();
subscriber.setMessageListener(this);

public void onMessage(Message message) {
    String msgText;
    try{
        if (msg instanceof TextMessage){
            msgText = ((TextMessage) msg).getText();
            System.out.println(“Got “ + msgText);
        }else System.out.println(“Got a non-text message”);
    }
}
Message Structure

- The header describes the message, i.e. 
  `msg id, routing, correlation id` (JMSMessageID, JMSRelyTo, JMSCorrelationID)

- The body contains the application data (up to 100Mb)
  `text, bytes, key-value pairs, etc.`

- Optional properties and selectors.
  Application specific parameters. Support customized message selection. Allows sharing queues for different types of messages.
Message Selectors

If you have to share a queue for different types of messages use selectors (filters) to avoid "stealing" somebody else’s messages:

```java
String selector = "SE=NASDAQ";
session.createReceiver(queue, selector);
```

Such a listener will de-queue only those messages that have a String property SE with the value “NASDAQ”.

Message producers have to set this property:

```java
TextMessage outMsg = session.createTextMessage();
outMsg.setText("100 QQQQ M");
outMsg.setStringProperty("SE", "NASDAQ");
```

For better performance use numeric selectors
Message Acknowledgement

- Use of transactions and a message acknowledgement mode are specified during creation of a JMS session:

```java
ConnectionFactory conFact = new ConnectionFactory();
Connection con = conFact.createConnection();
Session session = con.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

*The other choices are CLIENT_ACKNOWLEDGE and DUPS_OK_ACKNOWLEDGE*

- If the first argument is `true`, the session will use transactions and the second arg is ignored. An explicit `commit()` is required in this case.
Multithreading

- MDB allow you create a pool of message listeners without thread programming. Another choice: instantiate Java listener-classes on the app server startup.
- J2EE spec discourages use of multithreading in containers, but…
- Using threads to send messages from session beans is OK.
- Control threads by using thread pools. As of Java 5.0 they are part of J2SE (see `java.util.concurrent.ThreadPoolExecutor`).
- The Jakarta Commons project has a ThreadPool component
LDAP Servers for Corp Directories

- LDAP directory servers are highly optimized for data retrieval.
- Data in memory are stored in a tree structure:
  
  o=xyz.com
  ou=groups
    cn=accounting
    cn=HR
    cn=RiskTechnology
    cn=Yakov

- The address of the object that’s bound under the node Yakov is:

  cn=Yakov, cn=RiskTechnology, ou=groups, o=xyz.com
LDAP Server for JMS Objects

- Storing Queue Managers and Queues in LDAP directory servers:
  
  o=xyz.com
  
  ou=Trading
  
  cn=OrderQCF
  
  cn=OrderQueue
  
  cn=AckQCF
  
  cn=AckQueue
  
  cn=ComplianceQueue

- Here’s the JNDI lookup parameter:

  cn=ComplianceQueue, ou=Trading, o=xyz.com
LDAP and JNDI lookup

- Set JNDI properties, connect to the LDAP server, and find the object called OrderQCF: a JMS queue connection factory.

```java
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, "ldap://myldapserver.xyz.com:389");
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=Directory Manager");
env.put(Context.SECURITY_CREDENTIALS, "myPassword");

DirContext ctx = new InitialDirContext(env);
QueueConnectionFactory factory = (QueueConnectionFactory) ctx.lookup("cn=OrderQCF, ou=Trading, o=xyz.com");

Queue orderQueue = (Queue) ctx.lookup("cn=OrderQueue, ou=Trading, o=xyz.com");
```
An example of binding a WebSphere MQ queue object. A queue connection factory can be bound to LDAP similarly.

DirContext ctx = new InitialDirContext(env); // see previous slide
...
MQQueue queue = new MQQueue();
queue.setPersistence(JMSC.MQJMS_PER_NON);
// If on the other end of the queue is a non-jms reader:
// queue.setTargetClient(JMSC.MQJMS_CLIENT_NONJMS_MQ);
String ldapName = "cn=OrderQueue, ou=Trading, o=xyz.com";
ctx.rebind(ldapName, queue);

An app does not know if you’re using WebSphere MQ or other provider:
Queue orderQueue = (Queue)
ctx.lookup("cn=OrderQueue, ou=Trading, o=xyz.com");
A quiz

The data center with the MOM server is down. Your trading application uses several queue managers, topics and dozens of queues.

How do you quickly re-configure your trading system to use a different MOM server?
Quiz Solution

Run an LDAP binder program that will rebind the Disaster Recovery site’s MOM servers, queue managers, queues and topics.

Your trading system may stay alive even without bouncing J2EE servers.
Reporting and Compliance
Trade Reporting and Compliance

- NASDAQ 90-seconds rule
- Trading Pre-Clearance
- Insider Trading
- Use of Business Intelligence Engines
- Anti-Money Laundering

Using real-time messaging helps in preventing violations: reject a trade if it violates any regulations.
SOA, EDA and Messaging

Services:
- Orders, Accounts
- Credit Rating (CR). Instead of checking for every order, subscribe for modifications of CR (Pub/Sub).
- Trades. Check for and publish violations. Compliance department subscribes for violations (Point-to-Point).
- Positions
- Market data. Subscribe for particular stock info and initiate trades when conditions are met (Pub/Sub).
- Tibco BusinessWorks offers an easy way to integrate services by connecting them to the same message bus and creating service adapters.

Watch for WS-Notifications spec (OASIS) that won’t require every service to be written in Java.
Potential Issues

- Sequencing. Messages may arrive out of order (i.e. mod comes before create).
- If you know in advance how many slices are being sent, just add a sequence number to the header property.
- Create a re-sequencer that will put a message with high sequence numbers aside until the previous message arrives.
- Receiver can also not accept a message if a previous one does not exist in the database (modification of a non-existing order).
- JMSPriority header field (0-10) might help in delivering expedited messages (5-9) ahead of normal ones.
Potential Issues (cont.)

- Poisoned Messages (any type of corruption). They cause an application to rollback and retry.
- The number of re-tries should be configurable.
- Poisoned messages should be placed into a separate queue and require examination by people.
Potential Issues (cont.)

- The connection to the MOM provider is lost
  If you set a `javax.jms.ExceptionListener` to your Connection object, it’ll be notified of connectivity problems.

- This interface has the only method:
  `onException(JMSException e) {
    // try to reconnect};`

- If your MOM provider supports logical IPs, arrange some wait in this method to allow failover to a DR site.
  `conn.setExceptionListener(ExceptionListener listener)`
Scalability in Messaging

- Systems need to accommodate more concurrent users or increased volume.
- MOM Clusters with the messages persisted on the Storage Area Networks (SAN) devices based on RAID disks.
- If you can not afford SAN, you’ll have to arrange a replication of the persisted messages.
- Is the price of persisting messages too high? You can easily increase the message throughput 10 times or more by switching to a non-persistent mode.
www.eaipatterns.com has a good and growing collection of EI patterns that are not tied to a specific implementation, for example:

- **Message Routing:** Dynamic Router, Aggregator, Resequencer, Splitter

- **Messaging Endpoints:** Event-Driven Consumer, Polling Consumer, Transactional Client
Thank you!

Q & A

class: yakov@weekendwithexperts.com