J2EE 1.3 Overview

Part 4: Message-Driven Beans

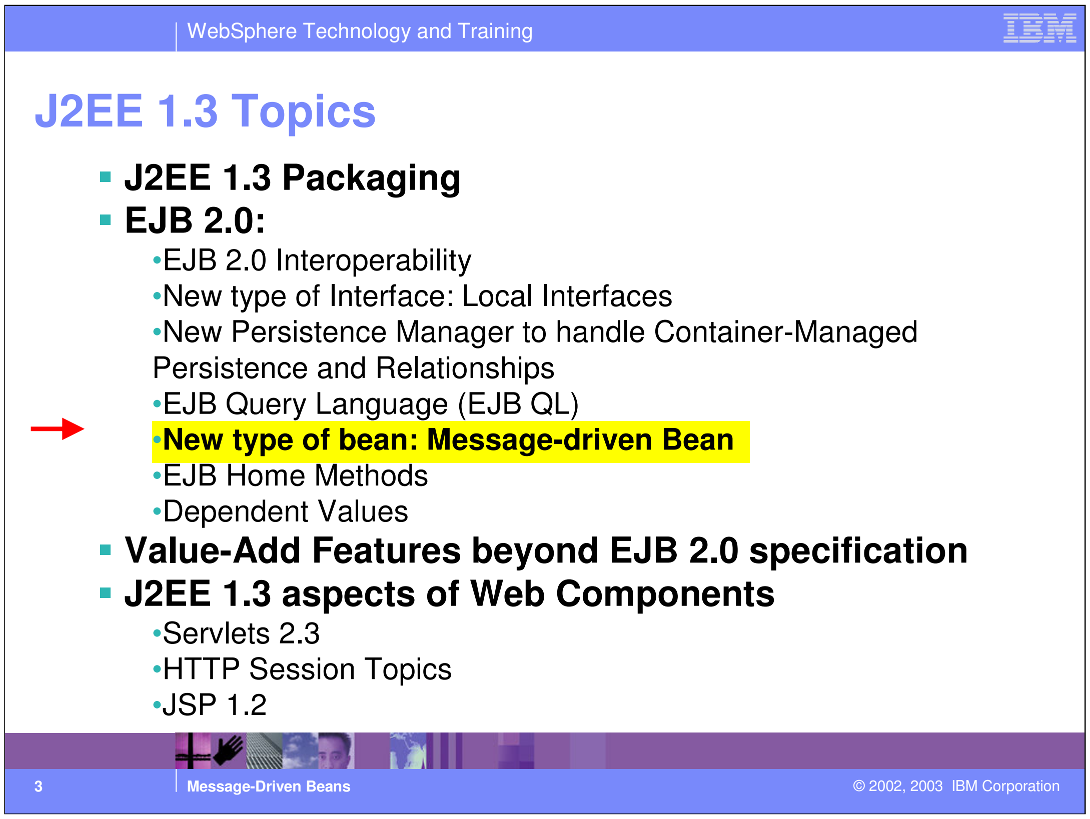
WebSphere software

business software

Updated 5/16/2003

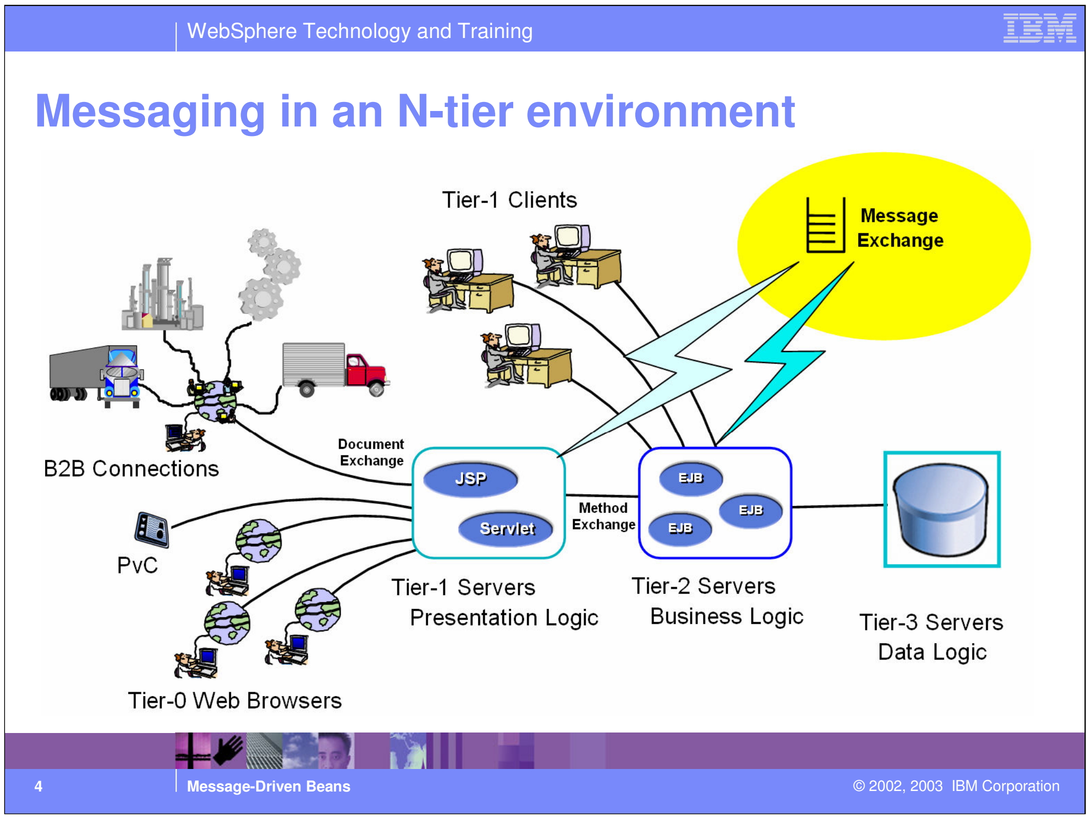
Objectives

- Describe the new type of bean – Message-Driven Bean
- Understand the importance of Messaging in an N-tier environment
- Compare the different Messaging Models (Point to Point and Pub/Sub)

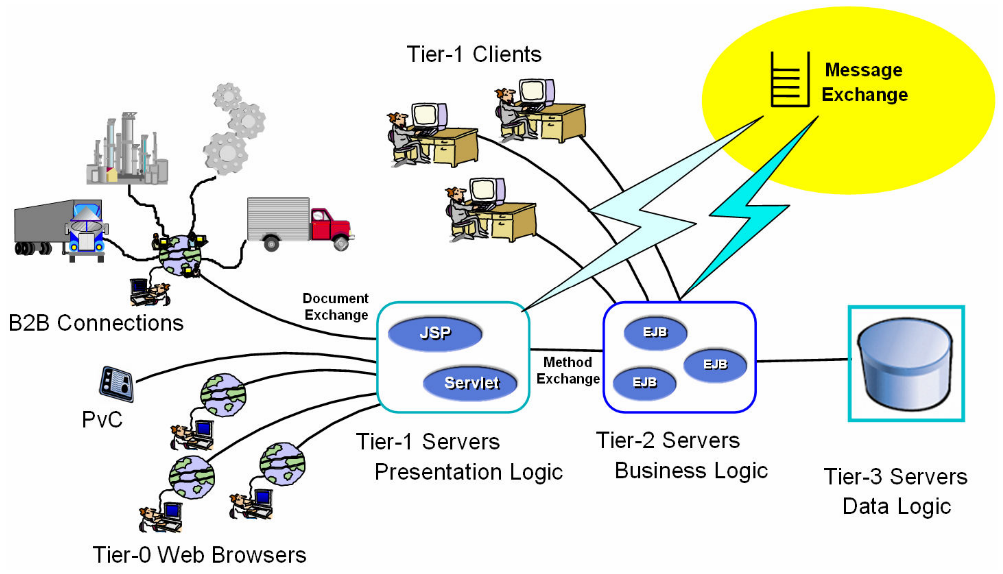


J2EE 1.3 Topics

- **J2EE 1.3 Packaging**
- **EJB 2.0:**
  - EJB 2.0 Interoperability
  - New type of Interface: Local Interfaces
  - New Persistence Manager to handle Container-Managed Persistence and Relationships
  - EJB Query Language (EJB QL)
  - **New type of bean: Message-driven Bean**
  - EJB Home Methods
  - Dependent Values
- **Value-Add Features beyond EJB 2.0 specification**
- **J2EE 1.3 aspects of Web Components**
  - Servlets 2.3
  - HTTP Session Topics
  - JSP 1.2

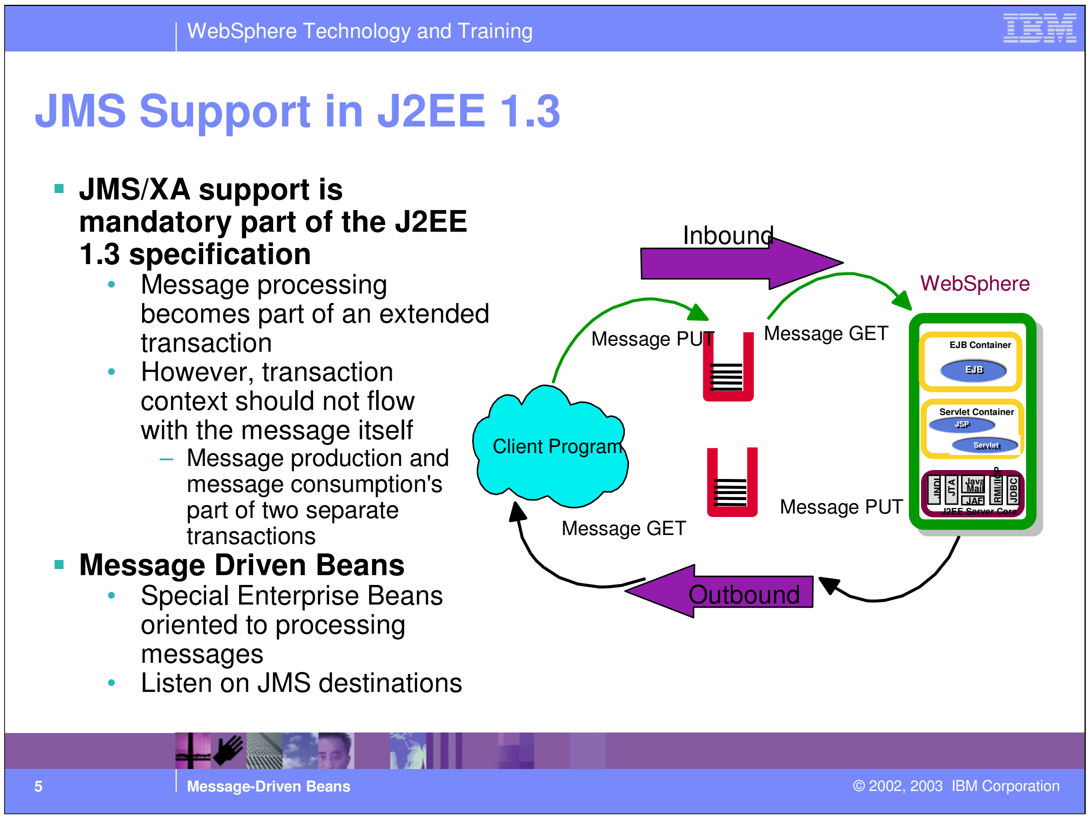


Messaging in an N-tier environment



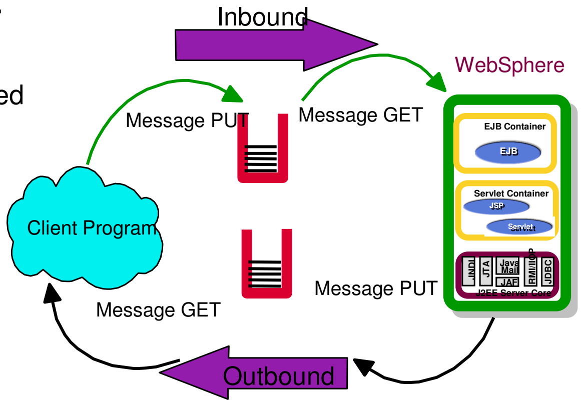
Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: a messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging agent that provides facilities for creating, sending, and receiving messages. Servlets and EJBs can send and receive messages to/from a messaging vendor. Messaging technologies provide an additional means of communicating with Server Side Components. Messaging enables distributed communication that is loosely coupled - You just need to know the Message format and the destination to use.

JMS (Java Message Service) is a Java API that allows applications to create, send, receive and read messages.



JMS Support in J2EE 1.3

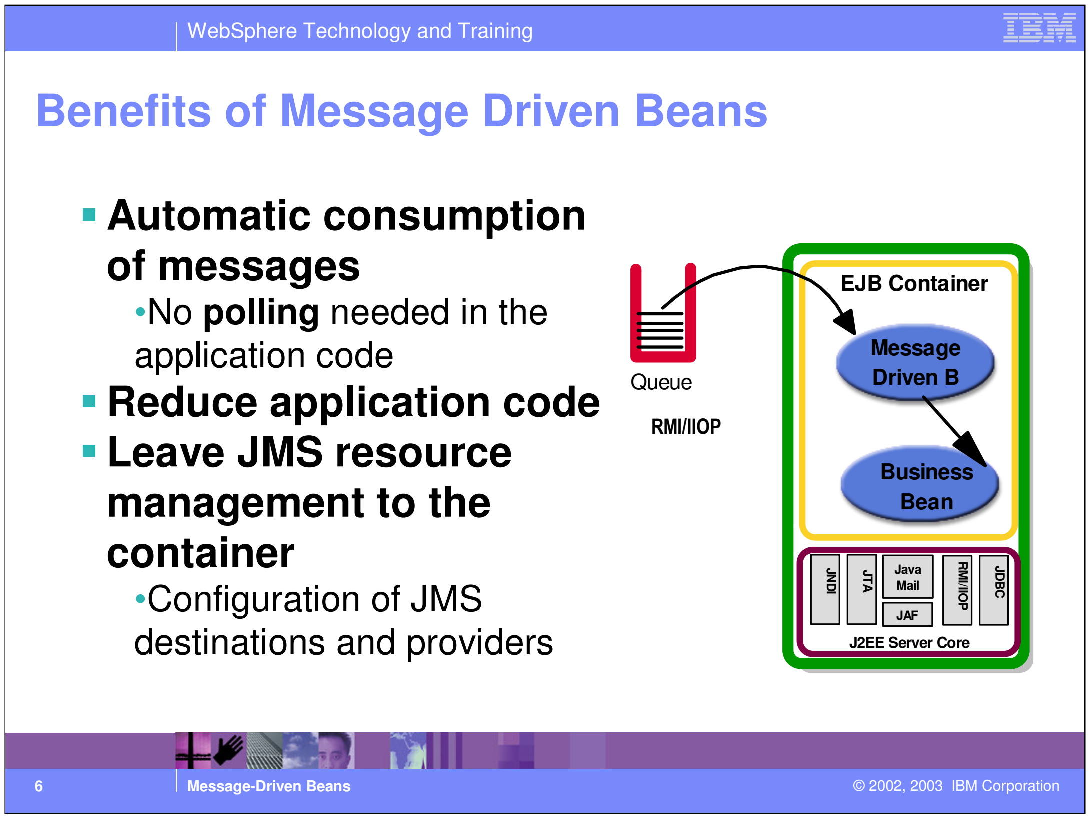
- **JMS/XA support is mandatory part of the J2EE 1.3 specification**
  - Message processing becomes part of an extended transaction
  - However, transaction context should not flow with the message itself
    - Message production and message consumption's part of two separate transactions
- **Message Driven Beans**
  - Special Enterprise Beans oriented to processing messages
  - Listen on JMS destinations



For compliance of J2EE 1.2 JMS support was mandatory, but JMS/XA was not. However, WebSphere 4.0 Advanced Edition had supported JMS/XA as a value-add feature.

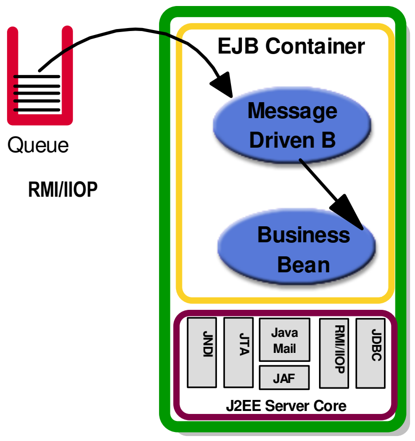
J2EE 1.3 introduced mandatory support for JMS/XA - the XA support makes it possible to protect both the message processing (put or get) and the business logic under the "umbrella" of an individual XA transaction. It has to be clearly understood that the transaction that produces an inbound message and the transaction that consumes that same message are two separate transactions.

J2EE 1.3 introduced a new type of bean called Message driven bean. A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. An MDB consumes messages from queues and topics that are sent by a JMS Client.

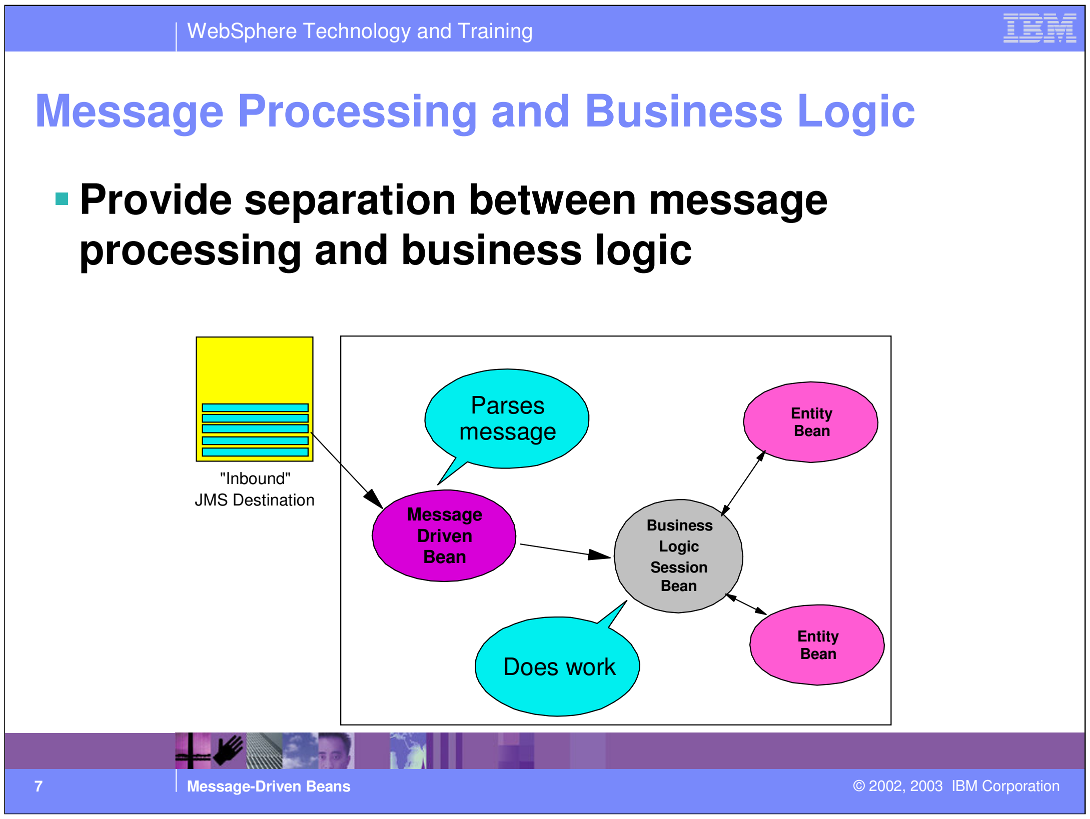


Benefits of Message Driven Beans

- **Automatic consumption of messages**
  - No **polling** needed in the application code
- **Reduce application code**
- **Leave JMS resource management to the container**
  - Configuration of JMS destinations and providers

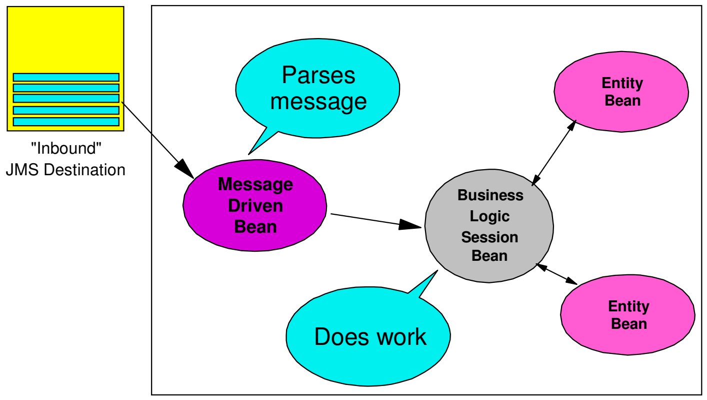


Message-Driven Beans offer a standard way to create a message consumer that is fully managed by the container. The bean provider only needs to concentrate on writing the logic that performs the parsing and processing of the message. Typically, the MDB will delegate the execution of business logic to some other EJB - a Session EJB in most cases. However, no coding needs to be done to retrieve the message or poll the JMS destination - no specific coding is needed to provide quality of service (failover, parallel sessions, etc.) - all this is up to the container to implement and provide.

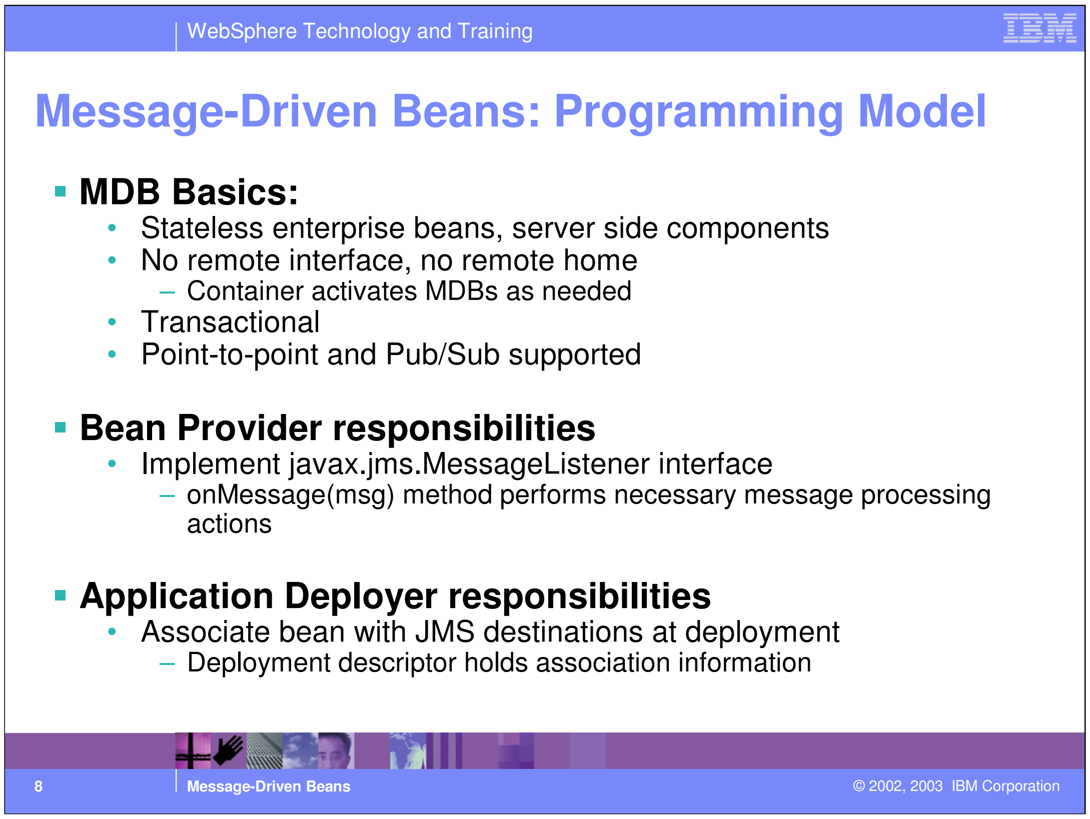


Message Processing and Business Logic

- **Provide separation between message processing and business logic**



By providing a clear separation between message and business processing, it is easier to implement the Message-Driven Beans. Ideally, the Message-driven bean parses the message and then delegates the work to be done to a business logic session bean. This design pattern promotes components reusability because the business logic session bean can be used by a variety of other clients. This programming model also reinforces the concepts that the Message-driven bean acts as an interface to the application.



Message-Driven Beans: Programming Model

- **MDB Basics:**
  - Stateless enterprise beans, server side components
  - No remote interface, no remote home
    - Container activates MDBs as needed
  - Transactional
  - Point-to-point and Pub/Sub supported
- **Bean Provider responsibilities**
  - Implement javax.jms.MessageListener interface
    - onMessage(msg) method performs necessary message processing actions
- **Application Deployer responsibilities**
  - Associate bean with JMS destinations at deployment
    - Deployment descriptor holds association information

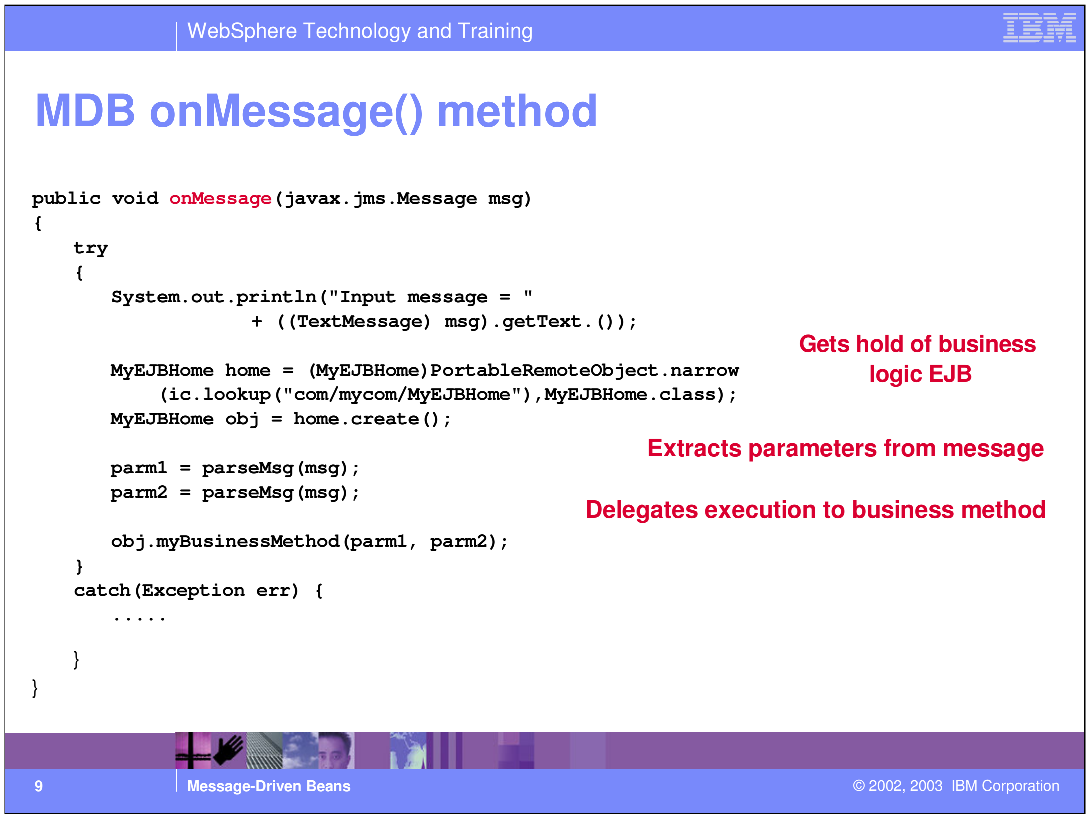
Message-driven beans (MDBs) are stateless, server-side, transaction-aware components for processing asynchronous JMS messages. It supports the two messaging models namely, Point-to-Point and Publish/Subscribe.

A message-driven bean is a complete enterprise bean, just like a session or entity bean, but there are some important differences. While a message-driven bean has a bean class and XML deployment descriptor, it does not have component interfaces. The component interfaces are absent because the message-driven bean is not accessible via the Java RMI API; it responds only to asynchronous messages.

It's important to understand that MDBs are not for "client use". Therefore they have no interface and no home, they cannot be "looked up" by a client.

The Bean Provider writes the application code for the Message Driven Bean. All Message driven beans must implement the javax.jms.MessageListener interface. The onMessage method contains the business logic that handles the processing of the messages. This method is called by the container when a message has arrived for the bean to service.

It is the Application Deployer's responsibility to associate the Message driven bean with the appropriate JMS destinations. This association is done at deployment time.



MDB onMessage() method

```
public void onMessage(javax.jms.Message msg)
{
    try
    {
       System.out.println("Input message = "
                   + ((TextMessage) msg).getText());

       MyEJBHome home = (MyEJBHome)PortableRemoteObject.narrow
           (ic.lookup("com/mycom/MyEJBHome"),MyEJBHome.class);
       MyEJBHome obj = home.create();

       parm1 = parseMsg(msg);
       parm2 = parseMsg(msg);

       obj.myBusinessMethod(parm1, parm2);
    }
    catch(Exception err) {
       .....

    }
}
```

Gets hold of business logic EJB

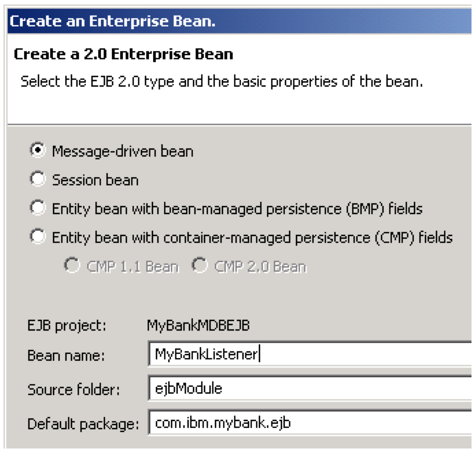
Extracts parameters from message

Delegates execution to business method

This is an example of a Message-Driven Bean's onMessage method. The message driven bean parses the message and then delegates the work to a business logic Enterprise Bean.

Message Driven Beans - Tooling Support

- **New EJB type in EJB Creation Wizard:**



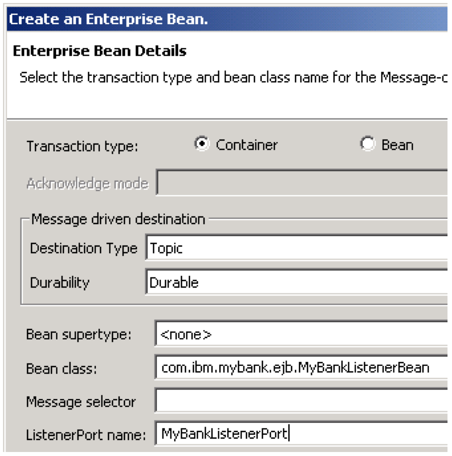
Create an Enterprise Bean.

Create a 2.0 Enterprise Bean

Select the EJB 2.0 type and the basic properties of the bean.

[✓ Message-driven bean]
Session bean
Entity bean with bean-managed persistence (BMP) fields
Entity bean with container-managed persistence (CMP) fields
CMP 1.1 Bean CMP 2.0 Bean

EJB project:	MyBankMDBEJB
Bean name:	MyBankListener
Source folder:	ejbModule
Default package:	com.ibm.mybank.ejb



Create an Enterprise Bean.

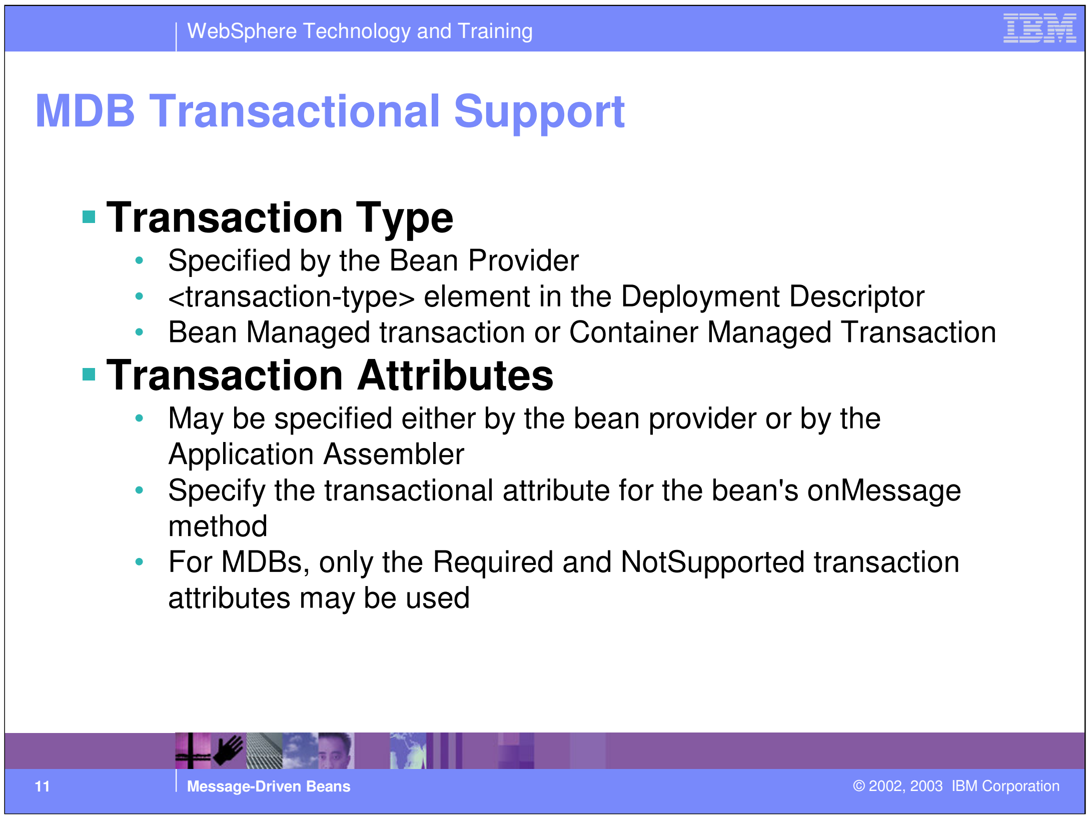
Enterprise Bean Details

Select the transaction type and bean class name for the Message-c

Transaction type: [✓ Container] Bean

Acknowledge mode	
Message driven destination	
Destination Type	Topic
Durability	Durable
Bean supertype:	<none>
Bean class:	com.ibm.mybank.ejb.MyBankListenerBean
Message selector	
ListenerPort name:	MyBankListenerPort

WebSphere Studio Application Developer provides tooling support for the creation of the code and deployment descriptors for Message-Driven Beans. To create a new Message driven bean, click File->New->Enterprise Bean. To create a Message Driven bean select Message-driven bean in the Create a 2.0 Enterprise bean wizard. Select the appropriate destination and other configuration options. Most of these fields are used as information for the deployer to correctly find the Message driven bean to the appropriate JMS destination. If the values are not specified at creation time, they can be specified in the deployment descriptors. The ListenerPort name is the most important value and it can be specified at creation time or later in the deployment descriptor. The ListenerPort name is the resource on the EJB Container which monitors the JMS destinations for messages and passes these messages onto any Message-driven beans which have been bound to the listener.



MDB Transactional Support

- **Transaction Type**
  - Specified by the Bean Provider
  - <transaction-type> element in the Deployment Descriptor
  - Bean Managed transaction or Container Managed Transaction
- **Transaction Attributes**
  - May be specified either by the bean provider or by the Application Assembler
  - Specify the transactional attribute for the bean's onMessage method
  - For MDBs, only the Required and NotSupported transaction attributes may be used

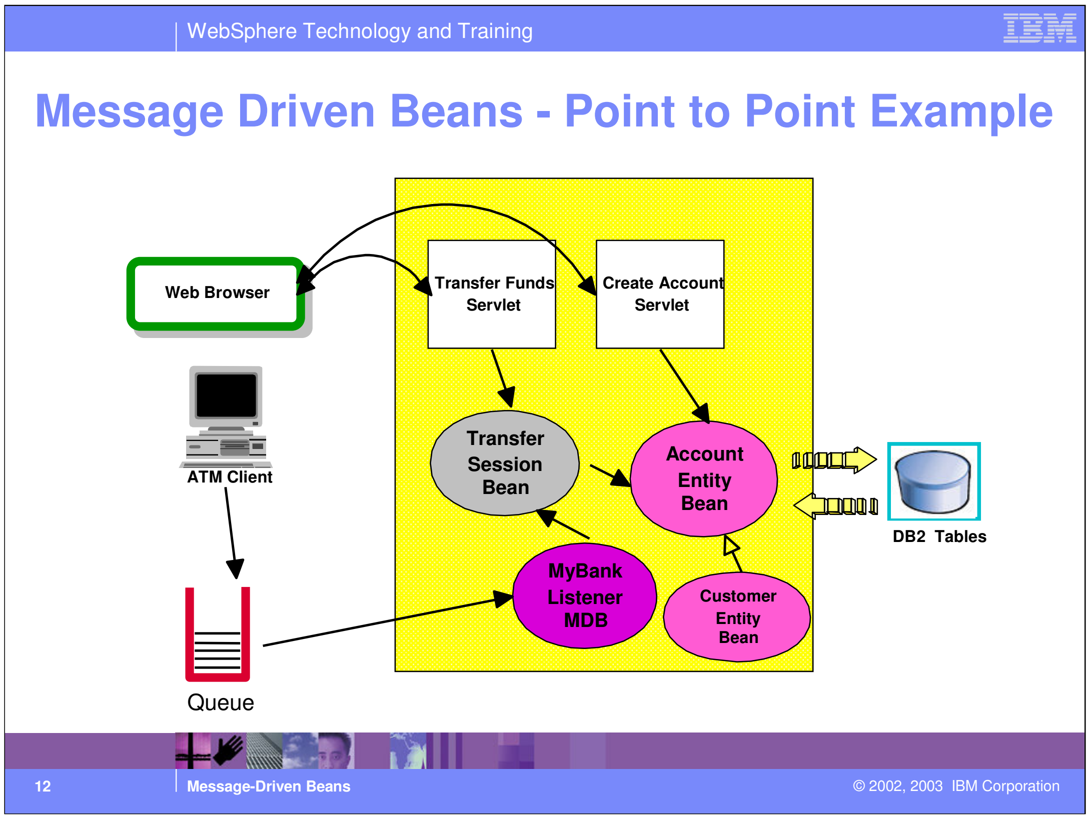
The bean provider of an MDB must use the transaction-type element to declare whether the MDB is of the bean-managed or container-managed transaction demarcation type. There is no mechanism for an Application Assembler to affect enterprise beans with bean-managed transaction demarcation. The Application Assembler must not define transaction attributes for an enterprise bean with bean-managed transaction demarcation.

The Application Assembler can use the Transaction Attribute to manage transaction demarcation for enterprise beans using container-manager transaction demarcation.

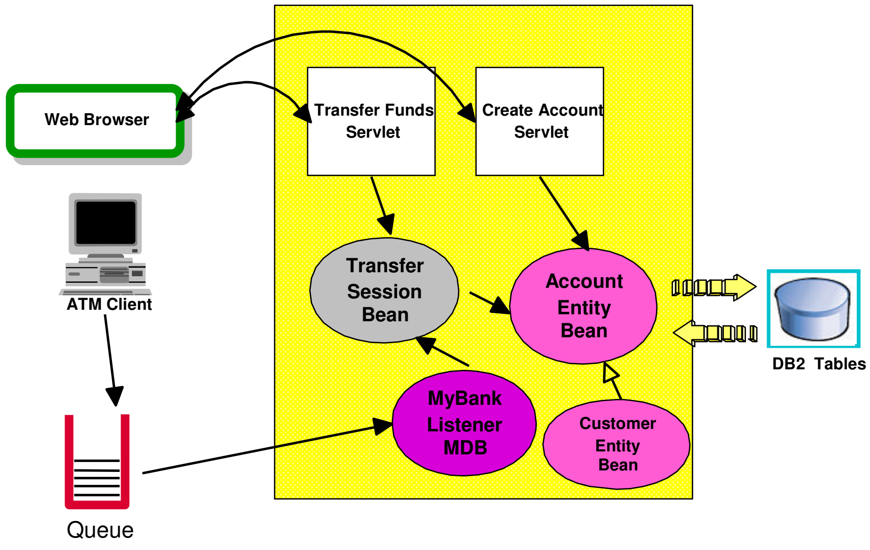
The transaction attribute for the MDB's onMessage method specifies how the container must manage transactions for the method when the method is invoked as a result of the arrival of a JMS message.

Only the NotSupported and Required transaction attributes may be specified for the MDB because there can be no preexisting transaction context (RequiresNew, Supports) and no client to handle exceptions (Mandatory, Never). The Container invokes a message-driven Bean method whose transaction attribute is set to NotSupported with an unspecified transaction context. If the onMessage method invokes other enterprise beans, the Container passes no transaction context

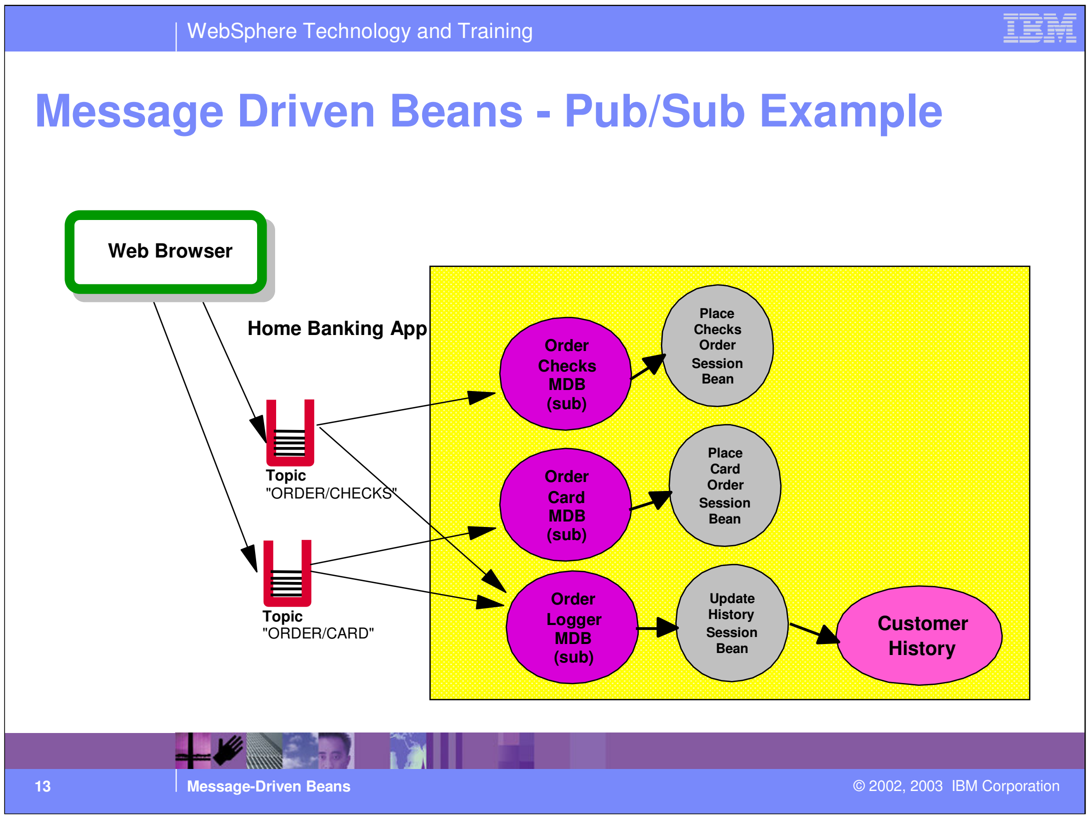
with the invocation. The Container must invoke a message-driven bean method whose transaction attribute is set to Required with a valid transaction context.



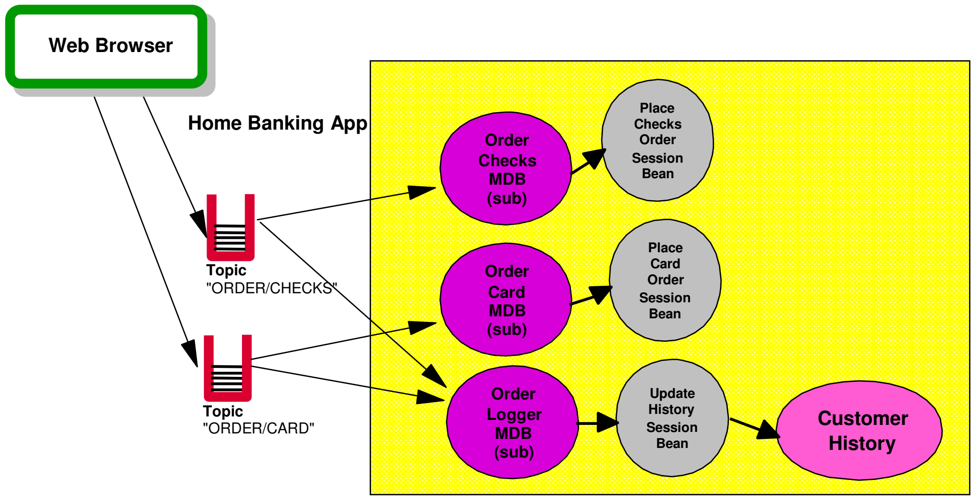
Message Driven Beans - Point to Point Example



Here is a point-to-point example built on top of the banking sample. The transfer may occur via the Web, or be initiated by a Java client, or be initiated by a message. The message is processed and taken apart by an MDB which calls the same Transfer Session EJB as the two other clients. In effect, the MDB access as another interface into the application just as the servlet and Java client are interfaces into the business logic of the application.

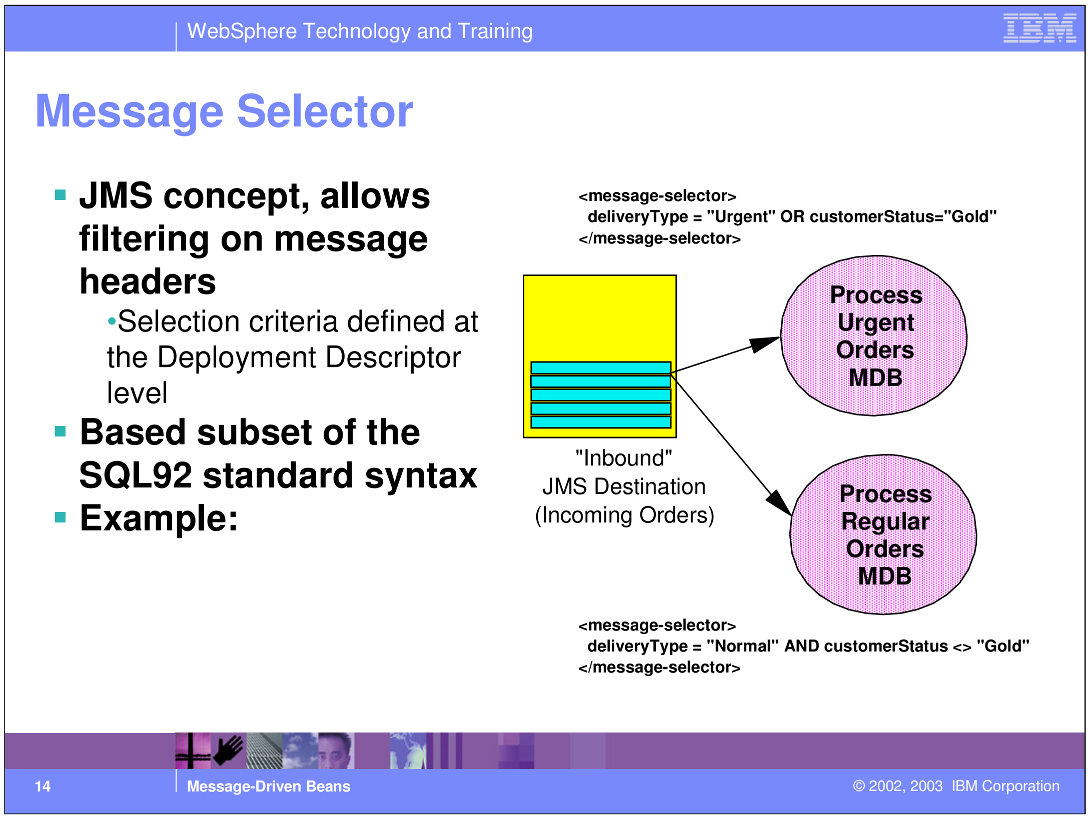


Message Driven Beans - Pub/Sub Example



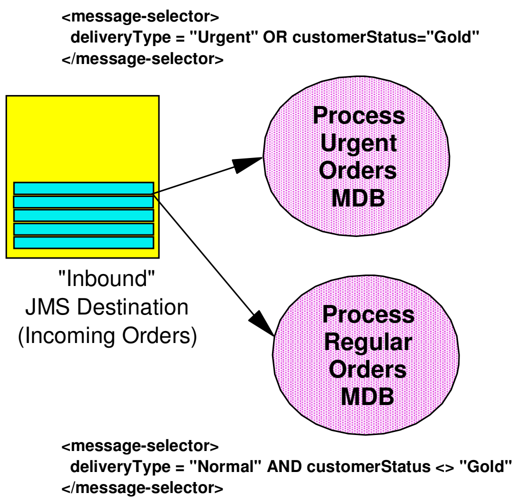
We can also accommodate pub/sub technology with MDBs.

In this case, we have one topic with two subtopics (one for ordering checks, one for ordering ATM cards) - and we have two different MDBs listening to the two different subtopics (one to initiate the processing of a card order, the other for the processing of the checks order) But, we also have a third MDB which has subscribed for all the subtopics of the ORDER topic and will log all of the ordering activity.



Message Selector

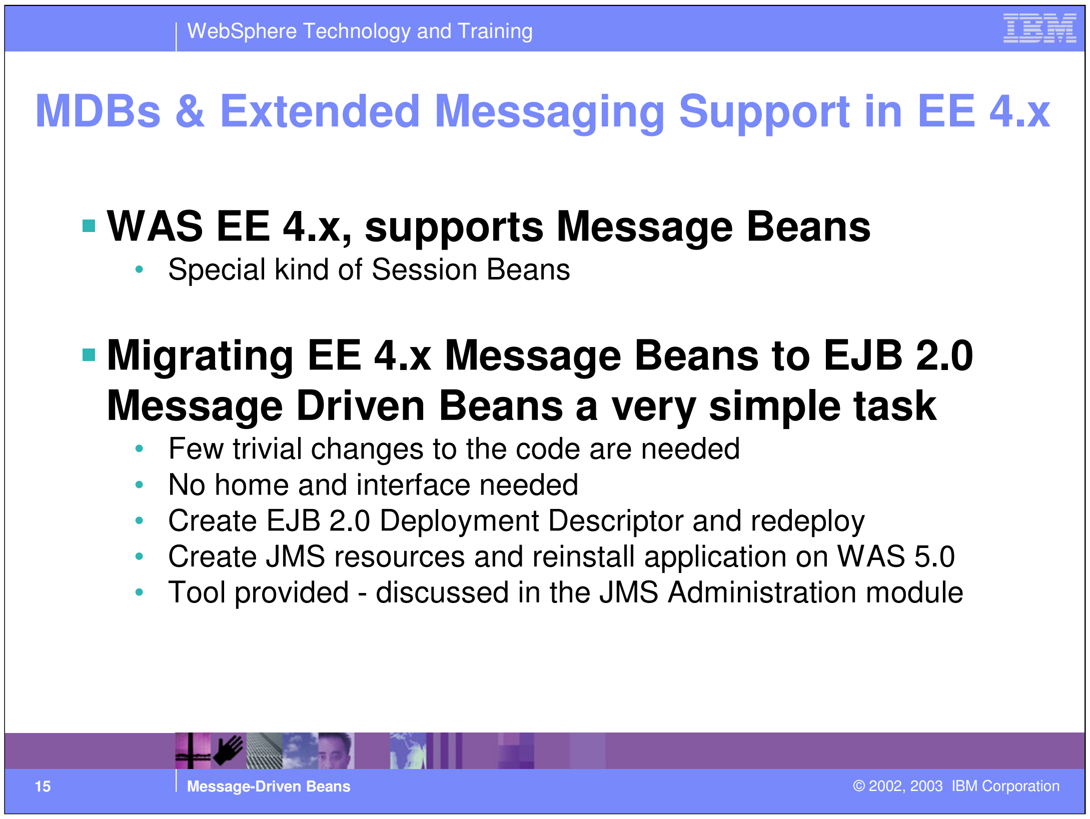
- **JMS concept, allows filtering on message headers**
  - Selection criteria defined at the Deployment Descriptor level
- **Based subset of the SQL92 standard syntax**
- **Example:**



<message-selector>
deliveryType = "Urgent" OR customerStatus="Gold"
</message-selector>
Process Urgent Orders MDB
"Inbound" JMS Destination (Incoming Orders)
Process Regular Orders MDB
<message-selector>
deliveryType = "Normal" AND customerStatus <> "Gold"
</message-selector>

An MDB can also declare a <message-selector> element, which is unique to message-driven beans. Message selectors allow an MDB to be more selective about the messages it receives from a particular topic or queue. Message selectors use Message properties as criteria in conditional expressions. Message properties, upon which message selectors are based, are additional headers that can be assigned to a message. They give the application developer or JMS vendor the ability to attach more information to a message.

The message selector is similar to a WHERE clause that you can specify in the Deployment Descriptor. It will define which messages will be passed down to the MDB. In our example, one of the two MDBs only gets messages for urgent orders, the other one will get all the other messages.

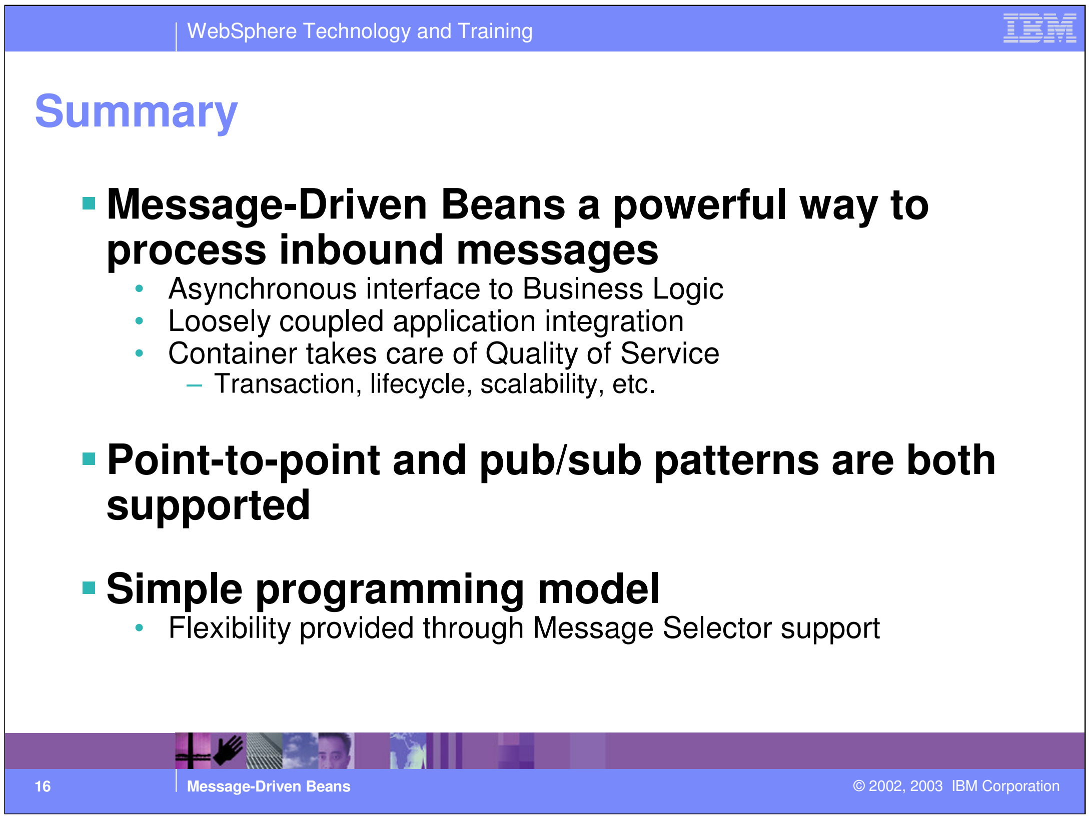


MDBs & Extended Messaging Support in EE 4.x

- **WAS EE 4.x, supports Message Beans**
  - Special kind of Session Beans
- **Migrating EE 4.x Message Beans to EJB 2.0 Message Driven Beans a very simple task**
  - Few trivial changes to the code are needed
  - No home and interface needed
  - Create EJB 2.0 Deployment Descriptor and redeploy
  - Create JMS resources and reinstall application on WAS 5.0
  - Tool provided - discussed in the JMS Administration module

In Enterprise Edition 4.x we had Extended Messaging support, which is very similar to Message-Driven Beans

Migration is not a big deal, although some code changes are needed (primarily because the Extended Messaging was implemented through session EJBs). A migration tool called mb2mdb has been provided with WebSphere 5.0 that can be used to do the migration from EE 4.x Message-Beans to Message-Driven Beans.

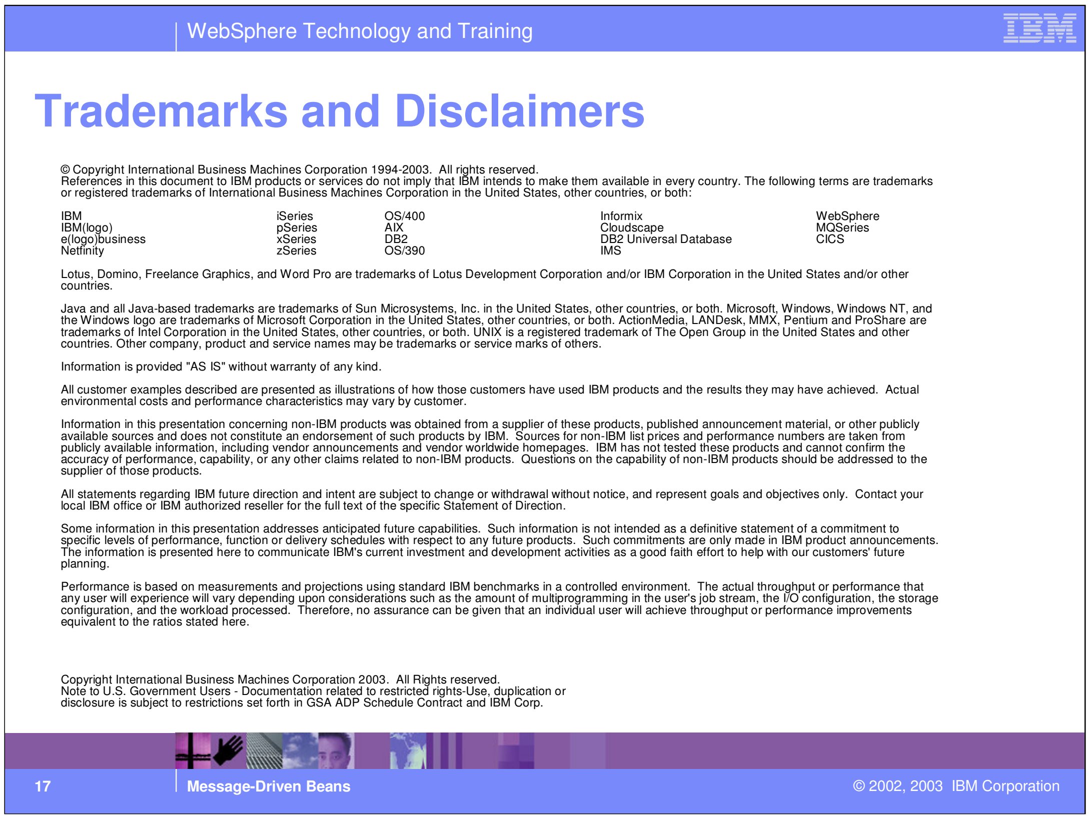


Summary

- **Message-Driven Beans a powerful way to process inbound messages**
  - Asynchronous interface to Business Logic
  - Loosely coupled application integration
  - Container takes care of Quality of Service
    - Transaction, lifecycle, scalability, etc.

- **Point-to-point and pub/sub patterns are both supported**

- **Simple programming model**
  - Flexibility provided through Message Selector support



Trademarks and Disclaimers