Business-Driven Software Engineering (7.Vorlesung)
Message-Driven Beans, Web Services
Thomas Gschwind <thg at zurich.ibm.com>
Agenda

- **Message-Driven Beans**
  - Java Messaging Service (JMS) Introduction
  - Topics and Queues
  - Implementation

- **Web Services**
  - Introduction
  - Web Service Implementation
  - Web Service Client Implementation

- **Demo**
Introduction

- Similar to session beans
- Sometimes more appropriate
  - Asynchrony
  - Reliability
  - Support multiple senders & receivers
  - Decoupling
- Examples: logging, stock ticker, order system, response to events generated by sensors, ...
- May only be called by sending messages
  => Message sinks
RMI vs. Messaging

- RMI

  Application → Application

- Messaging

  Application → Message Middleware → Application
A JMS Primer

- Different incompatible messaging systems exist
- Java Messaging Service provides
  - Standardized API
  - JMS drivers that know how to interact with a given messaging system
  - Abstracts low-level issues such as networking, message format, and server localization
Messaging Domains

- Publish/Subscribe
  (all messages received by all clients)

- Messaging
  (each message received by one client)
1. Retrieve JMS Driver
2. Create Connection
3. Create Session
4. Lookup JMS Destination
5. Create Producer or Consumer
6. Send or Receive Message
public class Client {

    public static void main(String[] args) throws Exception {
        Context ctx=new InitialContext(System.getProperties());
        TopicConnectionFactory f=(TopicConnectionFactory) ctx.lookup("ConnectionFactory");
        TopicConnection c=f.createTopicConnection();
        TopicSession session = c.createTopicSession(false, Session.AUTO_ACKNOWLEDGE);
        Topic t=(Topic)ctx.lookup("topic/quotes");
        TopicPublisher pub=session.createPublisher(t);
        TextMessage msg=session.createTextMessage();
        msg.setText("DIS 9.87");
        msg.setText("DIS 9.87");
        pub.publish(msg);
    }
}

Why a new kind of Bean?

Alternatives

- Use Java wrapper object
  - Write code to register message sink
  - Instantiation
  - No application server services

- Reuse existing bean
  - Different configuration
  - Different lifecycle management
  - Ugly
Characteristics

- Process JMS messages
- Durable or non-durable subscribers
- Only implement onMessage(...) method
- No return value/exceptions
- Stateless
Implementation

- Implemented as “Plain Old Java Object” (POJO)
- Annotate with @MessageDriven attribute
- Implement the javax.jms.MessageListener interface (and optionally the MessageDrivenBean interface)
- Indicate the bean lifecycle methods with annotations
  - @PostConstruct for methods to be invoked after construction
  - @PreDestroy to methods to be invoked before destruction
- As for session beans, use @Resource attribute to get hold of MessageDrivenContext
Lifecycle

Container decided it needs more instances in pool.

1. Class.newInstance()
2. setMessageDrivenContext()
3. ejbCreate()

Bean Instance Does Not Exist

1. ejbRemove

Container decides it does not need so many instances.

Pool of Ready Instances

Any client calls a method on any EJB object.

onMessage(...)
Annotations

```java
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(propertyName="destinationType",
            propertyValue="javax.jms.Topic"),
        @ActivationConfigProperty(propertyName="destination",
            propertyValue="queue/quotes")
    }
)
public class QuoteServiceMDB implements MessageListener {
    @Resource private MessageDrivenContext ctx;
    public QuoteServiceMDB() { ... }
    public void onMessage(Message message) {
        // ... 
    }
    @PreDestroy
    public void remove() { ... }
}
```
Sample onMessage(Message) Implementation

```java
public void onMessage(Message message) {
    if (message instanceof TextMessage) {
        TextMessage textMessage = (TextMessage)message;
        try {
            System.out.println("QuoteServiceMDB.onMessage("+
                             textMessage.getText()+")");
        } catch (JMSException e) {
            e.printStackTrace();
        }
    } else {
        System.err.println("QuoteServiceMDB.onMessage(unknown message type)";)
    }
}
```
Configuration Parameters

- Message Driven Beans are configured through ConfigProperties
  - destinationType and destination indicate Queue or Topic and name of destination
  - messageSelector allows to filter messages received by bean
  - acknowledgeMode indicates how to acknowledge messages with bean-managed transactions
  - subscriptionDurability indicates whether messages should be buffered while the bean is unavailable
Caveats

- **Transactions**
  - MDBs run within their own transactions

- **Security**
  - Needs to be handled by the bean itself

- **@PreDestroy Calls not guaranteed**
  - Message Driven Beans may stay around for a long time
  - Exception within the Message Driven Bean
  - Server Crash

- **Message Ordering**
  - Messages are not guaranteed to arrive in the order sent

- **Poison Messages**
  - If a message is not acknowledged, it will be redelivered
  - Messages causing an exception may be continuously redelivered
Agenda

- Message-Driven Beans
  - Java Messaging Service (JMS) Introduction
  - Topics and Queues
  - Implementation

- Web Services
  - Introduction
  - Web Service Implementation
  - Web Service Client Implementation

- Demo
Introduction

Clients and Servers that communicate via HTTP

- **Web Services**
  - Use the Web Service Description Language (WSDL) as Interface Description Language (IDL)
  - Inputs and outputs well-defined
  - Similar to statically typed languages
  - Code can be checked whether it uses the service correctly

- **RESTful (Web) Services**
  - No formal description of input and output
  - Similar to dynamically typed languages
  - Hard to check statically

- In the following, we will focus on Web Services
Motivation

- Typical middleware systems support a single RPC/RMI mechanism
  - Communication between CORBA and Microsoft COM systems?
  - Historically, requires bridging technologies for inter-operability

- Web Services have been developed to simplify this inter-operability

Why?
- Only need to build an adapter from “foo” to SOAP
- Use Web Services as standard protocol
- Loose coupling between client & servers

Are Web Services really so great?
- Technically, probably not
  (significant overhead but that can be dealt with by today’s systems)
- Politically, yes as long as everybody adheres to it
Web Service Basics

- Build and integrate large-scale systems (Service Oriented Architectures)
- Web Service support introduced in EJB2.1

Service Registry

Find

Publish

WSDL & UDDI

Service Requestor

Bind

SOAP

Service Provider

SOAP … Simple Object Access Protocol
WSDL … Web Services Description Language
UDDI … Universal Description, Discovery and Integration
Web Service Basics

- **Web Services Interaction Model**
  - Stateless, server retains no information regarding the client
  - There is no object with which the clients interact
  - Uncorrelated “one-way” messages

- **Web Service Description Language**
  - Describes the interface and operations of a web service
  - Describes the data (documents) exchanged with the service
Web Service Development

JEE offers seamless web service integration

- **Top-Down Development (WSDL First)**
  - We implement the WSDL file
  - Generate the Java Interface
  - Implement the Web Service

- **Bottom-Up Development (Java First)**
  - We start from a Java Interface
  - Implement the Web Service
  - Generate WSDL from the Java Interface

- Implementation Based on EJB
A Simple Hello World WSDL File

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wSDL:definitions
  name="QuoteService"
  targetNamespace="http://bdse.ifi.uzh.ch/QuoteService/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/">
  <wSDL:types>
    <xsd:schema
      targetNamespace="http://bdse.ifi.uzh.ch/QuoteService/"
    >
      <xsd:element
        name="NewOperation">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element
              name="in"
              type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element
        name="NewOperationResponse"/>
    </xsd:schema>
  </wSDL:types>
  <wSDL:message
    name="NewOperationRequest">
    <wSDL:part
      element="tns:NewOperation"
      name="symbol"/>
  </wSDL:message>
  <wSDL:message
    name="NewOperationResponse">
  </wSDL:message>
  <wSDL:portType
    name="QuoteService">
    <wSDL:operation
      name="getQuote">
      <wSDL:input
        message="tns:NewOperationRequest"/>
      <wSDL:output
        message="tns:NewOperationResponse"/>
    </wSDL:operation>
  </wSDL:portType>
  <wSDL:binding
    name="QuoteServiceSOAP"
    type=""/>
  <wSDL:service
    name="QuoteService">
    <wSDL:port
      binding="tns:QuoteServiceSOAP">
      <soap:address
        location="http://bdse.ifi.uzh.ch/"/>
    </wSDL:port>
  </wSDL:service>
</wSDL:definitions>
```

Looks hard to read? Don’t worry there are myriads of tools to help you…
The Same as Java Interface

```java
interface QuoteService {
    public double getQuote(String symbol);
}
```
Simple Object Access Protocol (SOAP)

- Uses HTTP for transport
- Not high performance, only interoperability

POST /.../QuoteService HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: ...
SOAPAction: ...
Host: ...

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="..."
   <soapenv:Body>
      <q0:getQuote>
         <in>xxxx</in>
      </q0:getQuote>
   </soapenv:Body>
</soapenv:Envelope>
Simple Object Access Protocol (SOAP)

- Response

```xml
<soapenv:Envelope xmlns:soapenv="..."
  xmlns:p193="http://bdse.ifi.uzh.ch/QuoteService/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
>
  <soapenv:Body>
    <NewOperationResponse>
      <out>-3.0</out>
    </NewOperationResponse>
  </soapenv:Body>
</soapenv:Envelope>
```
WSDL First Development

- Implement the WSDL
- Generate the Java code
- Package & Deploy the Web Service
Web Service Development

JEE offers seamless web service integration

- Top-Down Development (WSDL First)
  - We implement the WSDL file
  - Generate the Java Interface
  - Implement the Web Service

- Bottom-Up Development (Java First)
  - We start from a Java Interface
  - Implement the Web Service
  - Generate WSDL from the Java Interface

- Implementation Based on EJB
Java First Development

- Implement the Web Service
- Use Java Annotations in order to express Web Service artifacts that cannot be expressed in Java
A Sample Web Service (Implementation)

- Using wsgen, the WSDL can be generated from the Java implementation

```java
@WebService(name="Hello", portName="Hello",
    serviceName="HelloService",
    targetNamespace="http://sample")
public class Hello implements IHello {
    @WebMethod
    public String hello(String name) {
        System.out.println("Hello "+name);
        return "Hello "+name;
    }

    @WebMethod @Oneway
    public void helloOneWay(String name) {
        System.out.println("Hello "+name);
    }
}
```
A Sample Web Service (Eclipse Notes)

- Eclipse provides a nice wizard for top-down and bottom-up web service development
- The bottom-up wizard, however, directly uses the Java sources
  - Picks up arguments from source but
  - Ignores the annotations
The Web Service Client

Can invoke web service using

- **Static Stubs**
  - Uses a Java Stub that has been generated from the WSDL
  - More readable, service needs to be known during development

- **Dynamic Proxy**
  - XML Message is generated on the fly
  - Service need not be known during development
The Static Web Service Client

```java
public class QuoteServiceClient {
    public static void main(String[] args) throws Exception {
        URL url = new URL("http://localhost:8080/Quote_Service/
                              services/QuoteServiceSOAP?wsdl");
        String ns = "http://sample/QuoteService/";
        QName qname = new QName(ns, "QuoteService");
        ServiceFactory factory = ServiceFactory.newInstance();
        Service service = factory.createService(url, qname);

        QuoteService_PortType qs = (QuoteService_PortType) service.getPort(QuoteService_PortType.class);
        System.out.println(qs.getQuote("IBM"));
    }
}
```
public class QuoteDIIClient {
    public static void main(String[] args)
    throws Exception {
        URL url=new URL(args[0]);
        String ns="http://sample/QuoteService/";
        QName qname=new QName(ns, "QuoteService");
        ServiceFactory factory=ServiceFactory.newInstance();
        Service service=factory.createService(url, qname);

        QName port=new QName(ns,"QuoteServicePort");
        QName operation=new QName(ns, "getQuote");
        Call call=service.createCall(port, operation);
        System.out.println(call.invoke(new Object[]{})�
    }
}
Implementing a Web Service

- Web Service managed by EJB Container
- Create port components as session beans (can reuse existing session bean)
- Create the service endpoint interface (a Java interface)
- Write the webservices deployment descriptor
- In EJB3.0, one can also mark the methods to be exported
- Use the wizard of WebSphere Integration Developer
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.jws.WebMethod;
import javax.jws.WebService;

@Stateless
@Remote(CookieServer.class)
@WebService(serviceName="CookieWebService",
    portName="CookieWebServicePort")
public class CookieServerBean implements CookieServer
{
    @WebMethod
    public String getCookie() {
        return "Get it up, keep it up, ... LINUX: Viagra for the PC.";
    }
}
Where’s the WSDL File?

- **JBoss**
  - Need JBossWS (see www.jboss.org/jbossws for details)
  - Automatically generated by the container
  - Can be retrieved by clients from
    [http://localhost:8080/projectName/SessionBeanName?wsdl](http://localhost:8080/projectName/SessionBeanName?wsdl)
  - WebServiceName and PortName as specified by the annotations
Agenda

- Message-Driven Beans
  - Java Messaging Service (JMS) Introduction
  - Topics and Queues
  - Implementation

- Web Services
  - Introduction
  - Web Service Implementation
  - Web Service Client Implementation

- Demo
JBoss and Message Driven Beans

- JBoss 5 does not automatically create queues and topics
- Needs to be specified in a destination-service.xml file
- Needs to be copied into the deploy directory
- For the QuoteService, copy etc/quotes-destination-service.xml to $JBOSS_HOME/server/$PROFILE/deploy ($PROFILE typically is default)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <mbean code="org.jboss.mq.server.jmx.Topic"
         name="jboss.mq.destination:service=Topic,name=quotes">
    <depends optional-attribute-name="DestinationManager">
      jboss.mq:service=DestinationManager</depends>
  </mbean>
</server>
```
Summary

- **Message-Driven Beans**
  - Java Messaging Service (JMS) Introduction
  - Topics and Queues
  - Implementation

- **Web Services**
  - Introduction
  - Web Service Implementation
  - Web Service Client Implementation

- **Demo**
Review Questions

- What is the difference between Session Beans and Message Driven Beans
- What is the difference between a message queue and a message topic
- What are the different approaches to develop web services
Tasks

1. Download and run the example
2. Add another Message Driven Bean as client to the same topic
3. Change the topic to a queue and vice-versa
Outlook

- Petri Nets