5.– 8. September 2011
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Muli oder Esel?

Einführung in den Java-ESB Mule

Peter Boxberg

DEVK Versicherungen

Mule and Web Services

Dan Diephouse, MuleSource

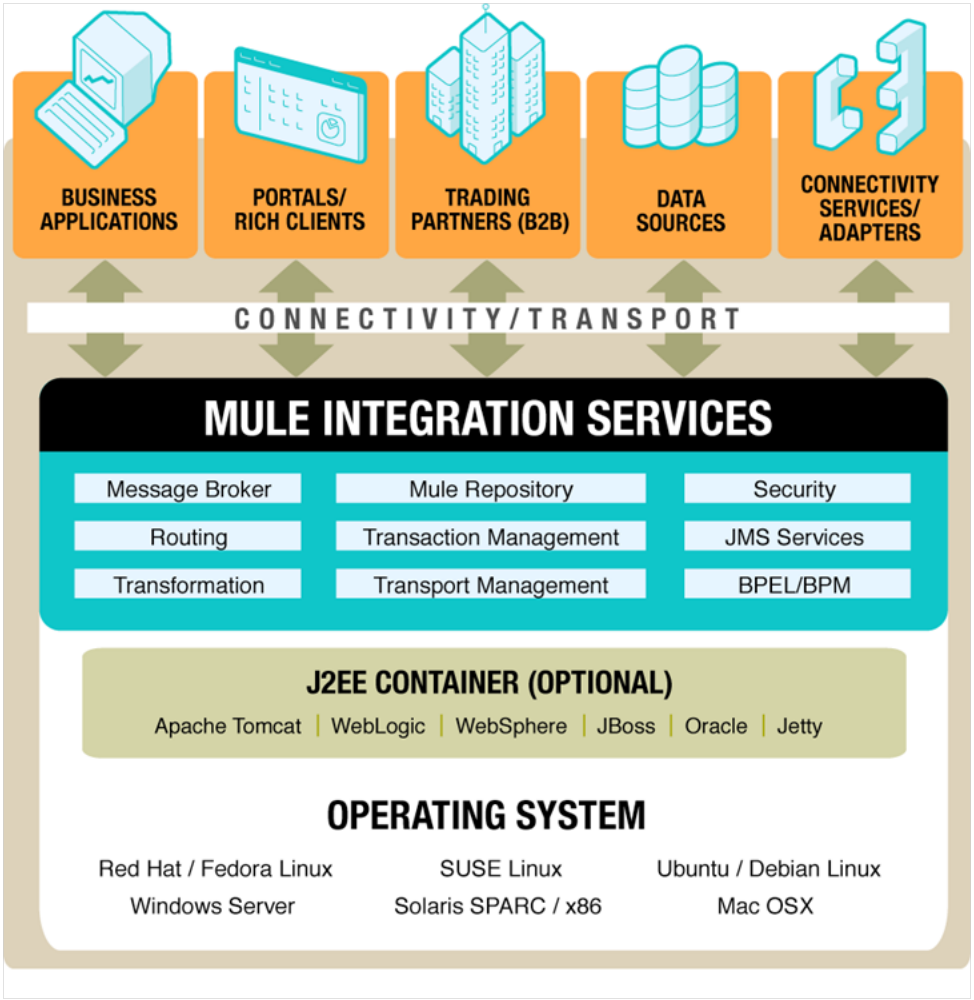
About Me

- Open Source: Mule, CXF/XFire, Abdera, Apache-*
- Exploring how to make building distributed services more powerful/approachable/scalable/etc
- MuleSource http://mulesource.com

Agenda

- Mule Overview
- Mule & web services
- Building web services integration
- The future

Overview


BUSINESS APPLICATIONS
PORTALS/ RICH CLIENTS
TRADING PARTNERS (B2B)
DATA SOURCES
CONNECTIVITY SERVICES/ ADAPTERS
CONNECTIVITY/TRANSPORT
MULE INTEGRATION SERVICES
Message Broker
Mule Repository
Security
Routing
Transaction Management
JMS Services
Transformation
Transport Management
BPEL/BPM
J2EE CONTAINER (OPTIONAL)
Apache Tomcat | WebLogic | WebSphere | JBoss | Oracle | Jetty
OPERATING SYSTEM
Red Hat / Fedora Linux
SUSE Linux
Ubuntu / Debian Linux
Windows Server
Solaris SPARC / x86
Mac OSX

SOA Swiss Army Knife

- Supports a variety of service topologies including ESB
- Highly Scalable; using SEDA event model
- Asynchronous, Synchronous and Request/Response Messaging
- J2EE Support: JBI, JMS, EJB, JCA, JTA, Servlet
- Powerful event routing capabilities (based on EIP book)
- Breadth of connectivity; 60+ technologies
- Transparent Distribution
- Transactions; Local and Distributed (XA)
- Fault tolerance; Exception management
- Secure; Authentication/Authorization

Why do developers choose Mule?

No prescribed message format

- XML, CSV, Binary, Streams, Record, Java Objects
- Mix and match

Zero code intrusion

- Mule does not impose an API on service objects
- Objects are fully portable

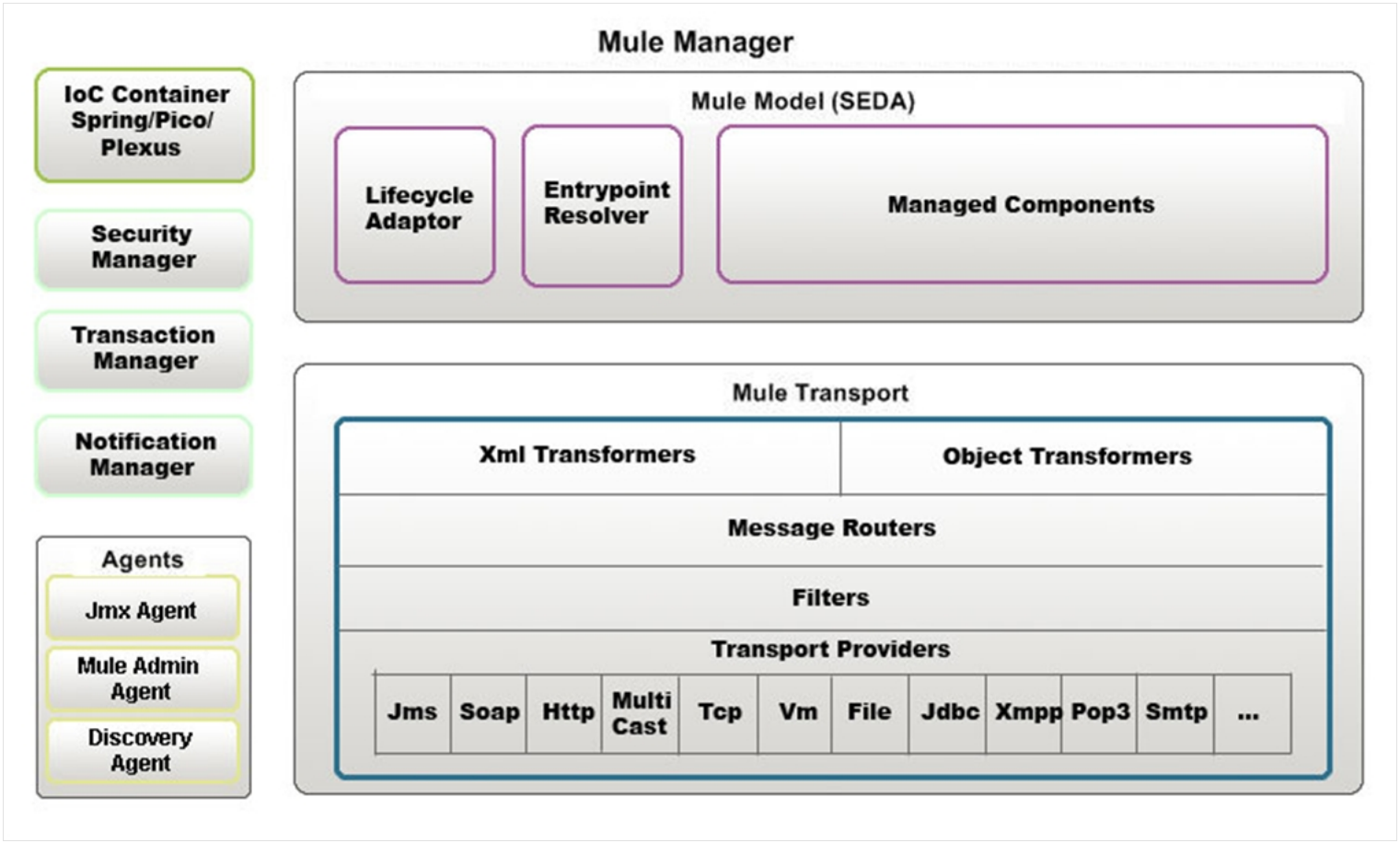
Existing objects can be managed

- POJOs, IoC Objects, EJB Session Beans, Remote Objects
- REST / Web Services

Easy to test

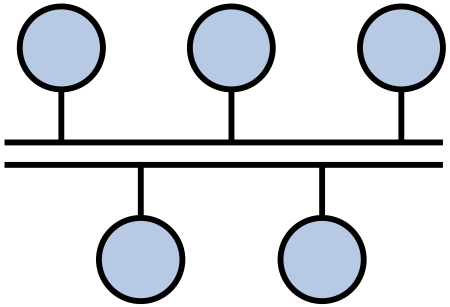
- Mule can be run easily from a JUnit test case
- Framework provides a Test compatibility kit
- Scales down as well as up

Mule Node Architecture

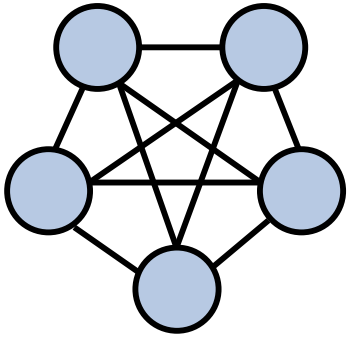


Mule Topologies

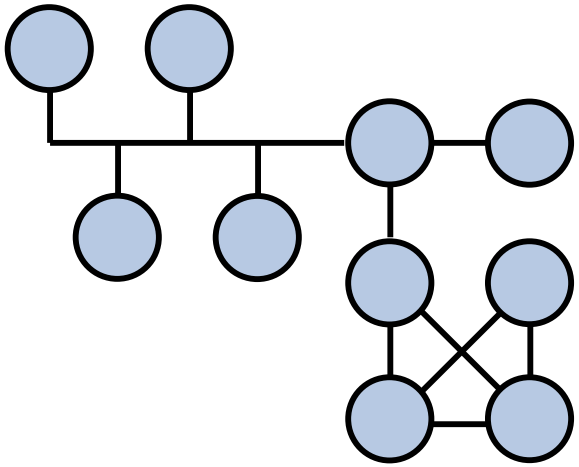
Mule can be configured to implement any topology:



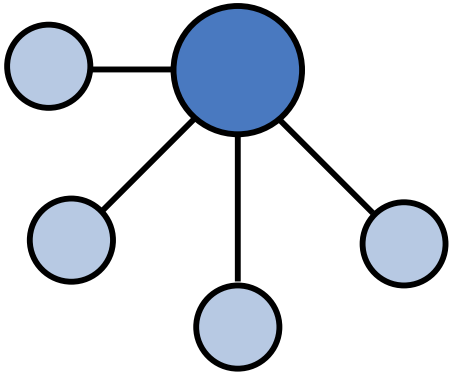
Enterprise Service Bus



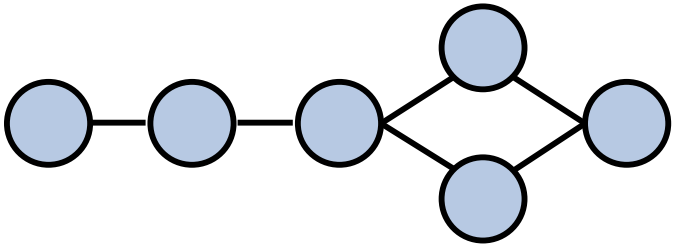
Peer Network



Enterprise Service Network

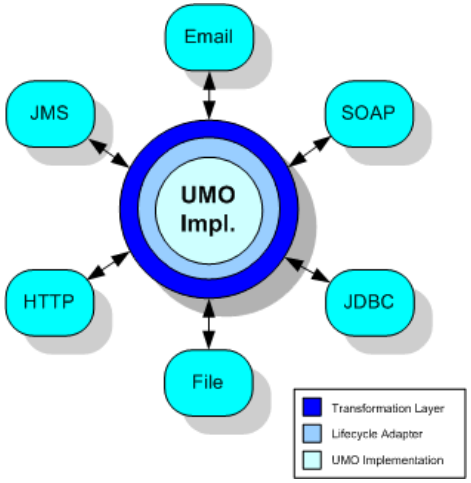


Client/Server and Hub n' Spoke



Pipeline

Core Concepts: UMO Service



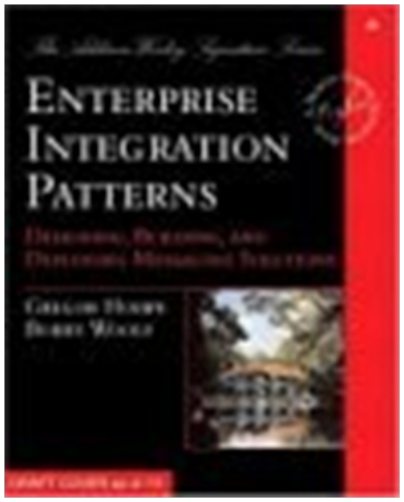
- UMO Services in Mule can be any object -
- POJOs, EJBs, Remote Objects, WS/REST Services
- A service will perform a business function and may rely on other sources to obtain additional information
- Configured in Xml, or programmatically
- Mule Manages Threading, Pooling and resource management
- Managed via JMX at runtime

Core Concepts: Endpoints

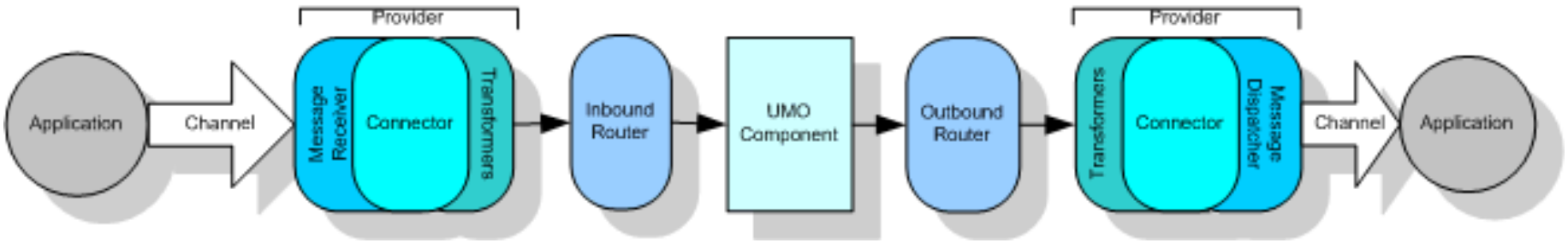
- Used to connect components and external systems together
- Endpoints use a URI for Addressing
- Can have transformer, transaction, filter, security and meta-information associated
- Two types of URI
  - **scheme://[username][:password]@[host][:port]?[params]**
    - **smtp://ross:pass@localhost:25**
  - **scheme://[address]?[params]**
    - **jms://my.queue?persistent=true**

Core Concepts: Routers

- Control how events are sent and received
- Can model all routing patterns defined in the EIP Book
- **Inbound Routers**
  - Idempotency
  - Selective Consumers
  - Re-sequencing
  - Message aggregation
- **Outbound Routers**
  - Message splitting / Chunking
  - Content-based Routing
  - Broadcasting
  - Rules-based routing
  - Load Balancing



Core Concepts: Transformers



Transformers

- Converts data from one format to another
- Can be chained together to form transformation pipelines

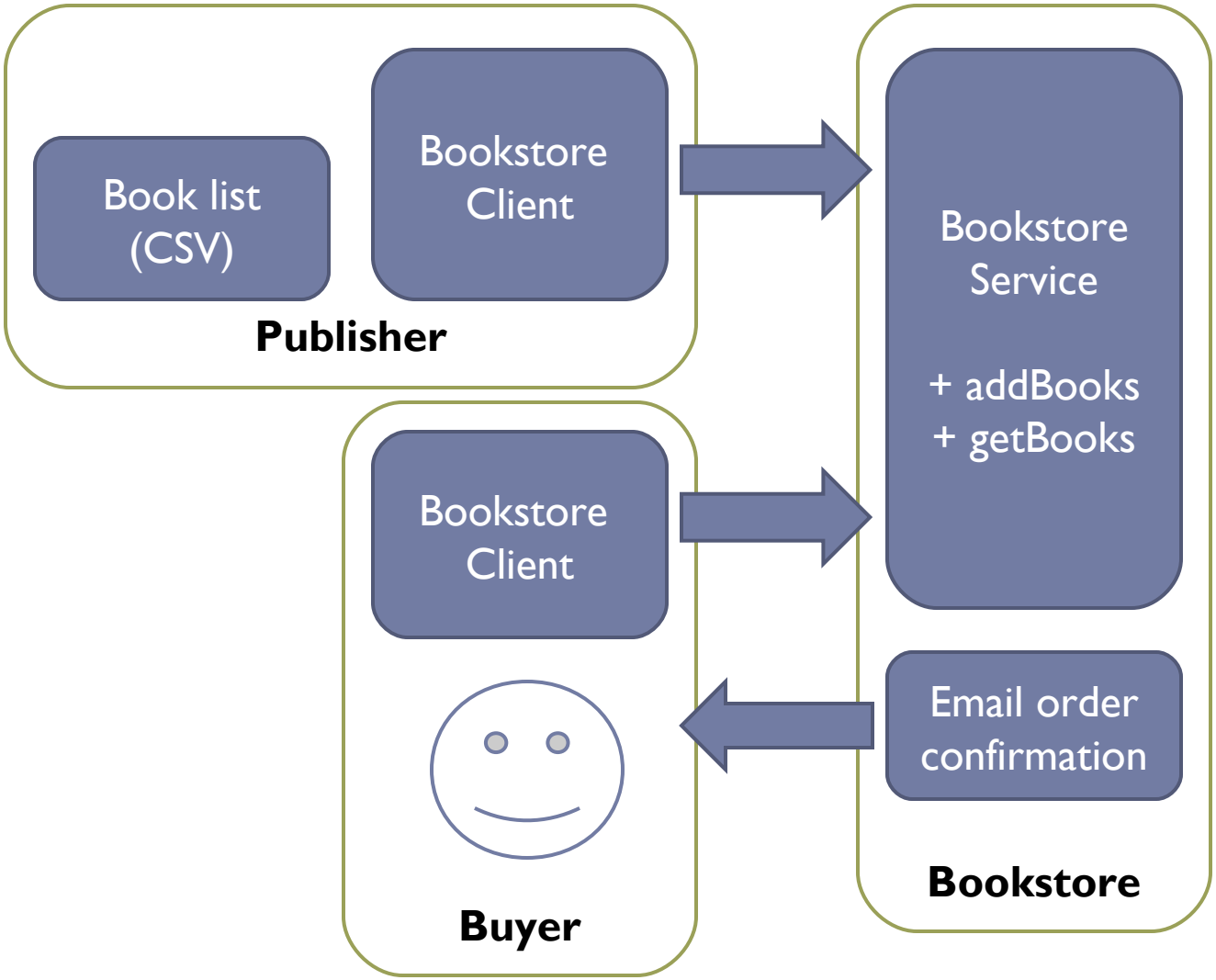
```
<jms:object-to-jms name="XmlToJms"/>

<custom-transformer name="CobolXmlToBusXml"
    class="com.myco.trans.CobolXmlToBusXml"/>

<endpoint address="jms://trades"
            transformers="CobolXmlToBusXml, XmlToJms"/>
```

Mule and Web Services

Our scenario



How?

- Mule provides
  - File connector
  - Email connector
  - CXF connector
- What's CXF?
  - Successor to XFire
  - Apache web services framework
  - JAX-WS compliant
  - SOAP, WSDL, WS-I Basic Profile
  - WS-Addressing, WS-Security, WS-Policy, WS-ReliableMessaging

How?

1. Build web service
2. Configure Mule server
3. Generate web service client
4. Build CSV->Book converter
5. Configure Mule with File inbound router and CXF outbound router
6. Configure Mule with Email endpoint to confirm orders
7. Send messages to email service from bookstore

Quick Intro to JAX-WS

- *The standard way to build SOAP web services*
- Supports code first web service building through annotations
- Supports WSDL first through ant/maven/command line tools

Building the Web Service

```
public interface Bookstore {
  long addBook(Book book);

  Collection<Long> addBooks(Collection<Book> books);

  Collection<Book> getBooks();
}
```

Annotating the service

```
@WebService
public interface Bookstore {
  long addBook(@WebParam(name="book") Book book);

  @WebResult(name="bookIds")
  Collection<Long> addBooks(
      @WebParam(name="books") Collection<Book> books);

  @WebResult(name="books")
  Collection<Book> getBooks();

  void placeOrder(@WebParam(name="bookId") long bookId,
                  @WebParam(name="address")String address,
                  @WebParam(name="email")String email)
}
```

Data Transfer Objects

```
public class Book {
   get/setId
   get/setTitle
   get/setAuthor
}
```

Implementation

```
@WebService(serviceName="BookstoreService",
  portName="BookstorePort",

  endpointInterface="org.mule.example.bookstore.Bookstore"
  )
public class BookstoreImpl implements Bookstore {
…
}
```

Configuring Mule

```
<mule-descriptor name="echoService"

  implementation="org.mule.example.bookstore.BookstoreImpl
  ">

 <inbound-router>
  <endpoint

  address="cxf:http://localhost:8080/services/bookstore"/>
 </inbound-router>

</mule-descriptor>
```

Starting Mule

- Web applications
- Embedded
- Spring
- Anywhere!

Main.main() – simplicity

- MuleXmlConfigurationBuilder builder = new MuleXmlConfigurationBuilder();
- UMOManager manager = builder.configure("bookstore.xml");

What happened?

Some notes about web services

- Be careful about your contracts!!!
- Don't couple your web service too tightly (like this example), allow a degree of separation
- This allows
  - Maitainability
  - Evolvability
  - Versioning
- WSDL first helps this

The Publisher

- Wants to read in CSV and publish to web service
- Use a CXF generated client as the "outbound router"
- CsvBookPublisher converts File to List<Book>
- Books then get sent to outbound router

Our implementation

```
public class CsvBookPublisher {

  public Collection<Book> publish(File file)
    throws IOException {
    …
  }
}
```

Domain objects are the Messages

- File
- Book
  - Title
  - Author
- Access can be gained to raw UMOMessage as well

Generating a client

```
<plugin>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-codegen-plugin</artifactId>
  <version>2.0.2-incubator</version>
  <executions>
    <execution>
      <id>generate-sources</id>
      <phase>generate-sources</phase>
      <configuration>
        <sourceRoot>
          ${basedir}/target/generated/src/main/java
        </sourceRoot>
        <wsdlOptions>
          <wsdlOption>
            <wsdl>
              http://localhost:8080/services/bookstore?wsdl
            </wsdl>
          </wsdlOption>
        </wsdlOptions>
      </configuration>
      <goals>
        <goal>wsdl2java</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Our configuration

```
<mule-descriptor name="csvPublisher"
  implementation="org.mule.example.bookstore.publisher.CsvBookPublisher"
  singleton="true">

 <inbound-router>
   <endpoint address="file://./books?
  pollingFrequency=100000&amp;autoDelete=false"/>
 </inbound-router>
 <outbound-router>
   <router className="org.mule.routing.outbound.OutboundPassThroughRouter">
…
   </router>
 </outbound-router>

</mule-descriptor>
```

Outbound router configuration

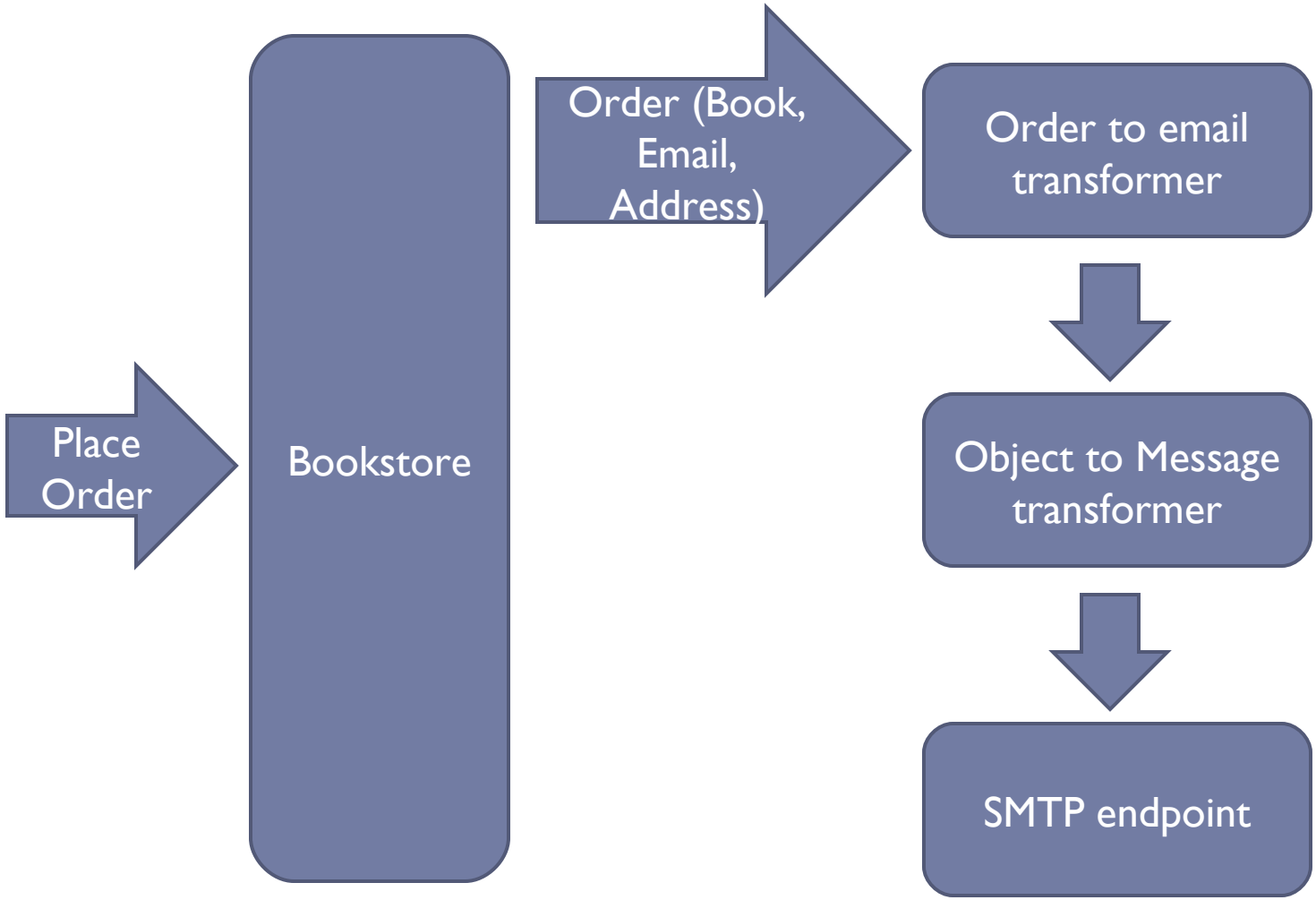
```
<router className="org.mule.routing.outbound.OutboundPassThroughRouter">

 <endpoint address="cxf:http://localhost:8080/services/bookstore">
   <properties>
     <property name="clientClass"
               value="org.mule.example.bookstore.BookstoreService"/>
     <property name="wsdlLocation"
               value="http://localhost:8080/services/bookstore?wsdl"/>
     <property name="port" value="BookstorePort"/>
     <property name="operation" value="addBooks"/>
   </properties>
 </endpoint>

</router>
```

Start it up!

Email Integration



Endpoint Configuration

```
<global-endpoints>
  <endpoint name="orderEmailService"
     address="smtps://username:password@hostname"
     transformers="OrderToEmail ObjectToMimeMessage">
    <properties>
      <property name="fromAddress"
                value="bookstore@mulesource.com" />
      <property name="subject"
                value="Your order has been placed!" />
    </properties>
  </endpoint>
</global-endpoints>
```

Transformers

```
<transformers>
  <transformer name="OrderToEmail"
    className="org.mule.example.bookstore.OrderToEmailTransformer"/>
  <transformer name="ObjectToMimeMessage"
    className="org.mule.providers.email.transformers.ObjectToMimeMessage"/>
</transformers>
```

Sending the Message

```
public void orderBook(long bookId, String address, String email) {
    // In the real world we'd want this hidden behind
    // an OrderService interface
    try {
        Book book = books.get(bookId);
        MuleMessage msg = new MuleMessage(
            new Object[] { book, address, email} );

        RequestContext.getEventContext().dispatchEvent(
             msg, "orderEmailService");
        System.out.println("Dispatched message to orderService.");
    } catch (UMOException e) {
        // If this was real, we'd want better error handling
        throw new RuntimeException(e);
    }
}
```

Questions?

- My Blog: http://netzooid.com
- Project Site: http://mule.mulesource.org
- MuleForge: http://muleforge.org
- MuleSource: http://mulesource.com