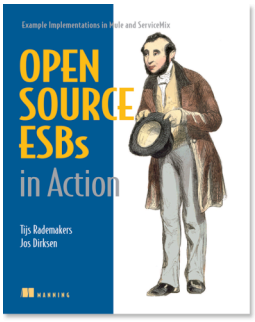
MANNING PUBLICATIONS

Routing and transformations in ServiceMix

Excerpted from



[Open-Source ESBs in Action](#)
Example Implementations in Mule and ServiceMix
EARLY ACCESS EDITION

Tijs Rademakers and Jos Dirksen
MEAP Release: September 2007
Softbound print: August 2008 (est.) | 480 pages
ISBN: 1933988215

*This article is taken from the book **Open Source ESBs in Action: Implementation Examples in Mule and ServiceMix**. As part of the book's introductory overview of the architecture of these two ESBs, this segment covers how you can create your own routing and transformation service in ServiceMix.*

An important part of any ESB is routing and transformation. Because ServiceMix itself is based on JBI, and JBI doesn't specify anything about complex routing and transformations, it isn't part of the architecture of ServiceMix.

Luckily the developers of ServiceMix have provided us with a couple of options that fill in this gap. In this article, we'll show you a high level overview of what the options are and how you can use them.

ROUTING USING THE EIP SERVICE ENGINE

The EIP service engine provides an implementation of a number of enterprise integration patterns that can be used for routing. These components are used just like any of the other service engines used in ServiceMix, using an XML based configuration. The code from Listing 1 shows an XML fragment from such a configuration.

Listing 1 Content based routing using the EIP ServiceEngine

```
<eip:content-based-router service="esb:simplerouter"          #1
   endpoint="routerEndpoint">                                 #1
  <eip:rules>
   <eip:routing-rule>                                        #2
    <eip:predicate>
     <eip:xpath-predicate
         xpath="/esb:order/esb:type=1"                       #3
         namespaceContext="#nsContext" />
    </eip:predicate>
    <eip:target>
```

```
        <eip:exchange-target                                    #4
            service="esb:orderService1" />                          #4
        </eip:target>
      </eip:routing-rule>
      <eip:routing-rule>                                        #5
        <eip:predicate>
          <eip:xpath-predicate
              xpath="count(/esb:order/esb:type)=2"
              namespaceContext="#nsContext" />
        </eip:predicate>
        <eip:target>
          <eip:exchange-target
              service="esb:orderService2" />
        </eip:target>
      </eip:routing-rule>
      <eip:routing-rule>                                        #6
        <eip:target>
          <eip:exchange-target service="esb:orderService3" />
        </eip:target>
      </eip:routing-rule>
    </eip:rules>
  </eip:content-based-router>

  <eip:namespace-context id="nsContext">                            #7
    <eip:namespaces>
      <eip:namespace
          prefix="esb">http://opensourceesb/architecture
      </eip:namespace>
    </eip:namespaces>
  </eip:namespace-context>
```

Annotation 1: Define the type of router
Annotation 2: Add a routing rule
Annotation 3: Evaluates the incoming message
Annotation 4: Target service for routing rule
Annotation 5: Another routing rule
Annotation 6: Target service for default rule
Annotation 7: Namespace definition for XPath expression

You can see that we defined a content-based router service (#1). If a message is sent to this service, this service uses routing rules (#2 and #5) to determine what to do with the message. In a routing rule a number of predicates (#3) are defined. If all the predicates match the message is sent to the target that is specified (#4). If no routing rules match the message is sent to the routing rule with no predicates (#6).

This is just one of the many routing patterns available in ServiceMix and the EIP Service Engine. Table 1 provides an overview of the routing patterns provided by the EIP Service Engine.

Table 1 An overview of the routing patterns supported by the EIP service engine provided with ServiceMix.

Content-Based Router	Route a message to a certain service based on its content.
Message Filter	Drop a message if it doesn't match the certain criteria.
Pipeline	Serves as a bridge between an In-Only MEP and an In-Out MEP.
Static Recipient List	Sends a message to a number of different services. (multicast)
Static Routing Slip	Route a message to a number of services in sequence.
Wire Tap	Listen in on the messages being sent on the line
XPath Splitter	Splits a message based on an XPath expression and routes the resulting messages to the specified service.
SplitAggregator	Combines the messages from the XPath Splitter back into a single message.
Content Enricher	Enrich the message with information from an additional service.
Resequencer	Resequence the order of the messages before sending them on to the target.

ROUTING USING CAMEL

Besides routing using the EIP Service Engine, ServiceMix can also use the Apache Camel project to handle it's routing. Apache Camel is a sub-project of Apache ActiveMQ that implements a full set of Enterprise Integration Patterns that can be configured in Java or in XML. We won't go too deep into Apache Camel here, but just to give you a taste of the functionality of Apache Camel, the following is a quick example of how this works in combination with ServiceMix.

Apache Camel has two different configuration types. You can either write the routing rules in Java using a Java Domain Specific Language (DSL), or you configure them in XML. We will first look at how the most basic routing rule looks in Java in code Listing 2.

Listing 2 Camel route using Java

```
public class SimpleRoute extends RouteBuilder {

  private final static String SERVICE_IN =                    #1
    "jbi:service:http://dummy.org/camelReceiver";               #1
  private final static String ENDPOINT_OUT_1 =                  #1
    "jbi:endpoint:http://dummy.org/fileSender/endpoint";        #1
  private final static String ENDPOINT_OUT_2 =                  #1
    "jbi:endpoint:http://dummy.org/fileSender2/endpoint";       #1

  public void configure() throws Exception {                    #2
    from(SERVICE_IN).to(ENDPOINT_OUT_1,ENDPOINT_OUT_2);         #2
  }                                                             #2
}
```

Annotation #1: Define the endpoints and services
Annotation #2: Configure the route in camel

What you can see is that we've used plain Java to configure a route. We've first defined a number of fully qualified names (names with namespaces) (#1), and in the configure method, we tell Apache Camel how it should route a message. The route described in (#2) is a very basic route. It listens for messages that are sent to the service specified as SERVICE_IN and sends those messages to the endpoints defined as ENDPOINT_OUT_1 and ENDPOINT_OUT_2. So with this very simple configuration we've implemented the recipient list pattern, where each incoming message is sent to multiple targets.

Doing this in Spring is pretty much the same as in Java, only we'll be using a different notation as shown in Listing 3.

Listing 3 Camel route using Spring

```
<route>
  <from uri="jbi:service:http://dummy.org/camelReceiver" />
  <to>
    <uri>jbi:endpoint:http://dummy.org/fileSender/endpoint</uri>
    <uri>jbi:endpoint:http://dummy.org/fileSender2/endpoint</uri>
  </to>
</route>
```

As you can see in this listing, it's very readable and easy to understand. You once again specify where the message is coming from, and where you want it to be sent to.

With the EIP and Camel options for routing, ServiceMix provides good support for routing messages from one service (or endpoint) to the other. Let's quickly look at how ServiceMix deals with applying transformations to messages.

APPLYING TRANSFORMATIONS

Transforming messages from one format to the other is less important in ServiceMix (or in any other JBI container) then it is in Mule. Since the internal format used has to be XML, all the messages that are sent between the various components are guaranteed to be XML. However it can still be the case that you want to transform an XML to a different format before you want to send it to a certain service. To implement transformations in ServiceMix, we can use the Saxon Service Engine that uses XSLT style sheets as shown in the following code snippet.

```
<saxon:xslt service="esb:xslt-transformation"
        endpoint="trans-endpoint"
        resource="classpath:OrderTypeAToOrderTypeB.xsl">
```

Besides the Saxon Service Engine, ServiceMix also provides an XQuery based transformation component.