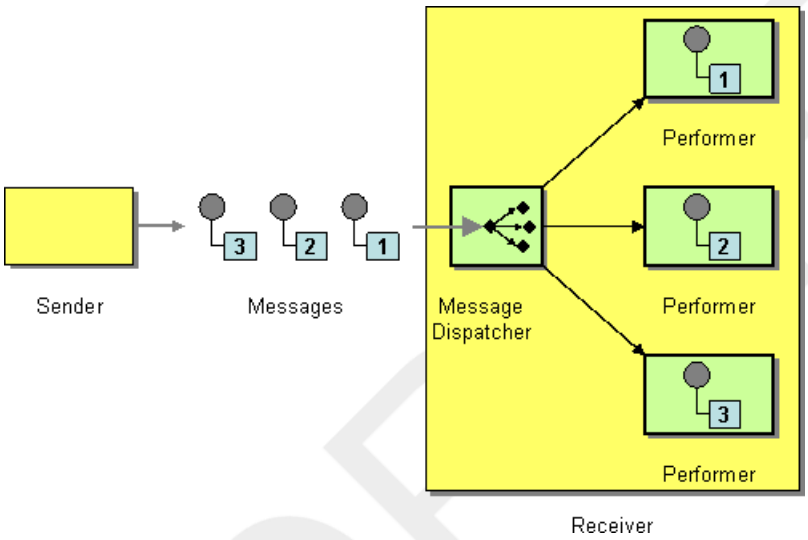
Message Dispatcher

Overview

The *message dispatcher* pattern is used to consume messages from a channel and distribute them locally to *performers*, which are responsible for processing the messages. In a Mediation Router application, performers are usually represented by in-process endpoints, which are used to transfer messages to another section of the route.

Figure 29. Message Dispatcher Pattern



You can implement the message dispatcher pattern in Mediation Router using one of the following approaches:

- JMS selectors on page 93.
- JMS selectors in ActiveMQ on page 95.
- Content-based router on page 95.

JMS selectors

If your application consumes messages from a JMS queue, you can implement the message dispatcher pattern using *JMS selectors*. A JMS selector is a predicate expression involving JMS headers and JMS properties: if the selector

evaluates to `true`, the JMS message is allowed to reach the consumer; if the selector evaluates to `false`, the JMS message is blocked. In many respects, a JMS selector is like a filter processor on page 54, but it has the added advantage that the filtering is implemented inside the JMS provider. This means that a JMS selector can block messages before they are transmitted to the Mediation Router application, giving a significant efficiency advantage.

In Mediation Router, you can define a JMS selector on a consumer endpoint by setting the `selector` query option on a JMS endpoint URI. For example:

```
from("jms:dispatcher?selector=Country
Code%3d%27US%27").to("cxf:bean:replica01");
from("jms:dispatcher?selector=Country
Code%3d%27IE%27").to("cxf:bean:replica02");
from("jms:dispatcher?selector=Country
Code%3d%27DE%27").to("cxf:bean:replica03");
```

Where the selector strings are URL encoded according to the `application/x-www-form-urlencoded` MIME format (see the HTML specification [http://www.w3.org/TR/html4/]). The unencoded selector strings in the preceding routes are, respectively: `CountryCode='US'`, `CountryCode='IE'`, and `CountryCode='DE'`. The predicates that appear in a selector string are based on a subset of the SQL92 conditional expression syntax (for full details, see the JMS specification [http://java.sun.com/products/jms/docs.html]). The identifiers appearing in a selector string can refer either to JMS headers or to JMS properties. For example, in the preceding routes, we presume that the sender has set a JMS property called `CountryCode`.

If you want to add a JMS property to a message from within your Mediation Router application, you can do so by setting a message header (either on *In* message or on *Out* messages). When reading or writing to JMS endpoints, Mediation Router maps JMS headers and JMS properties to and from its native message headers.

For more complex selector strings, it is probably more convenient to encode the strings directly using the `java.net.URLEncoder` utility class. For example:

```
from("jms:dispatcher?selector=" + java.net.URLEncoder.en
code("CountryCode='US'","UTF-8")).
    to("cxf:bean:replica01");
```

Where the UTF-8 encoding must be used.

JMS selectors in ActiveMQ

You can also define JMS selectors on ActiveMQ endpoints. For example:

```
from("activemq:dispatcher?selector=Country
Code%3d%27US%27").to("cxf:bean:replica01");
from("activemq:dispatcher?selector=Country
Code%3d%27IE%27").to("cxf:bean:replica02");
from("activemq:dispatcher?selector=Country
Code%3d%27DE%27").to("cxf:bean:replica03");
```

For more details, see ActiveMQ: JMS Selectors [http://activemq.apache.org/selectors.html] and ActiveMQ Message Properties [http://activemq.apache.org/activemq-message-properties.html].

Content-based router

The essential difference between the content-based router pattern and the message dispatcher pattern is that a content-based router dispatches messages to physically separate destinations (remote endpoints), whereas a message dispatcher dispatches messages locally, within the same process space. In Mediation Router, the distinction between these two patterns is not very great, because the same router logic can be used to implement both of them. The only distinction is whether the target endpoints are remote (content-based router) or in-process (message dispatcher).

For details and examples of how to use the content-based router pattern see Content-Based Router on page 52.