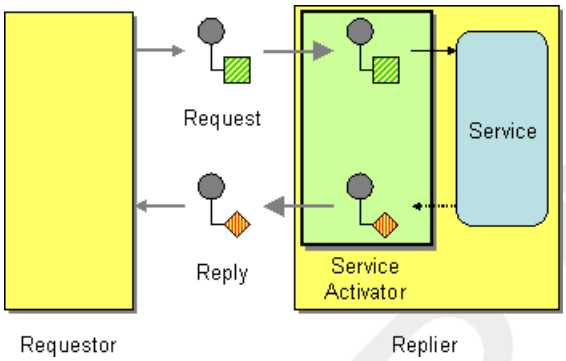
Service Activator

Overview

The *service activator* pattern describes the scenario where a service's operations are invoked in response to an incoming request message. The service activator is responsible for identifying which operation to call and for extracting the data to use as the operation's parameters. Finally, the service activator invokes an operation using the data extracted from the message. The operation invocation can either be oneway (request only) or two-way (request/reply).

Figure 34. Service Activator Pattern



In many respects, a service activator resembles a conventional remote procedure call (RPC), where operation invocations are encoded as messages. The main difference is that a service activator needs to be more flexible. Whereas an RPC framework standardises the request and reply message encodings (for example, Web service operations are encoded as SOAP messages), a service activator typically needs to improvise the mapping between the messaging system and the service's operations.

Bean integration

The main mechanism that FUSE Mediation Router provides to support the service activator pattern is *bean integration*. Bean integration [http://activemq.apache.org/camel/bean-integration.html] provides a general framework for mapping incoming messages to method invocations on Java objects. For example, the Java fluent DSL provides the processors, `bean()` and `beanRef()`, that you can insert into a route in order to invoke methods on a registered Java bean. The detailed mapping of message data to Java

method parameters is determined by the *bean binding*, which can be implemented by adding annotations to the bean class.

For example, consider the following route which calls the Java method, `BankBean.getUserAccBalance()`, in order to service requests incoming on a JMS/ActiveMQ queue:

```
from("activemq:BalanceQueries")
  .setProperty("userid", xpath("/Account/Bal
anceQuery/UserID").stringResult())
  .beanRef("bankBean", "getUserAccBalance")
  .to("velocity:file:src/scripts/acc_balance.vm")
  .to("activemq:BalanceResults");
```

The messages pulled from the ActiveMQ endpoint, `activemq:BalanceQueries`, have a simple XML format that provides the user ID of a bank account—for example:

```
<?xml version='1.0' encoding='UTF-8'?>
<Account>
  <BalanceQuery>
    <UserID>James.Strachan</UserID>
  </BalanceQuery>
</Account>
```

The first processor in the route, `setProperty()`, extracts the user ID from the *In* message and stores it in the `userid` exchange property, (this is preferable to storing it in a header, because the *In* headers cease to be available after invoking the bean).

The service activation step is performed by the `beanRef()` processor, which binds the incoming message to the `getUserAccBalance()` method on the Java object identified by the `bankBean` bean ID. The following code shows a sample implementation of the `BankBean` class:

```
// Java
package tutorial;

import org.apache.camel.language.XPath;

public class BankBean {
    public int getUserAccBalance(@XPath("/Account/Bal
anceQuery/UserID") String user) {
        if (user.equals("James.Strachan")) {
            return 1200;
```

```
        }
        else {
            return 0;
        }
      }
}
```

Where the binding of message data to method parameter is enabled by the `@XPath` annotation, which injects the content of the `UserID` XML element into the `user` method parameter. On completion of the call, the return value is inserted into the body of the *Out* message (which is then copied into the *In* message for the next step in the route). In order for the bean to be accessible to the `beanRef()` processor, you must instantiate an instance in Spring XML. For example, you can add the following lines to `META-INF/spring/camel-context.xml` configuration file to instantiate the bean:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ... >
  ...
  <bean id="bankBean" class="tutorial.BankBean"/>
</beans>
```

Where the bean ID, `bankBean`, identifes this bean instance in the registry.

The output of the bean invocation is fed into a Velocity template, in order to produce a properly formatted result message. The Velocity endpoint, `velocity:file:src/scripts/acc_balance.vm`, specifies the location of a velocity script, which has the following contents:

```
<?xml version='1.0' encoding='UTF-8'?>
<Account>
  <BalanceResult>
    <UserID>${exchange.getProperty("userid")}</UserID>
    <Balance>${body}</Balance>
  </BalanceResult>
</Account>
```

The exchange instance is available as the Velocity variable, `exchange`, which enables you to retrieve the `userid` exchange property, using `${exchange.getProperty("userid")}`. The body of the current In message, `${body}`, contains the result of the `getUserAccBalance()` method invocation.