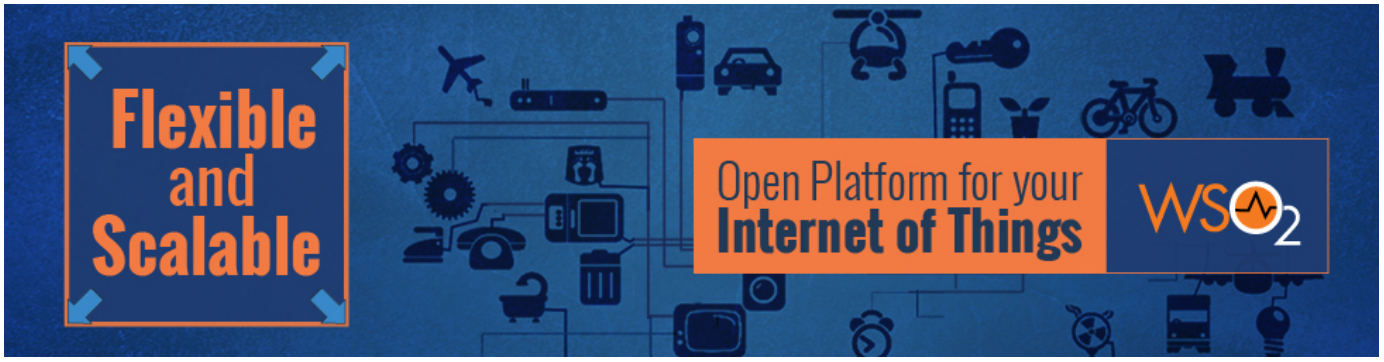


IVAN KRIZSAN Bio Website

How to Create a Dynamic HTTP Proxy with Mule

The Enterprise Integration Zone is brought to you in partnership with WSO2. Learn more about WSO2's API Management.

In this post I will show a proof-of-concept for a dynamic HTTP proxy implemeted using Mule.

Principle

The proxy forwards HTTP request using the context and relative path parts of the request URL to determine the server and port to which the request is to be forwarded.
In the example in this article a SOAP web service will be deployed to listen to the following URL:

```
1. <a
   href="http://localhost:8182/services/GreetingService">http://locall
```

In the above URL, the server and port is localhost:8182, the context and relative path parts of the URL is “services/GreetingService”.
The example program will be deployed to listen to requests at the following URL:

```
1. <a
   href="http://localhost:8981/dynamicHttpProxy/">http://localhost:898
```

In order to invoke the GreetingService via the HTTP proxy, the endpoint URL will look like this:

```
1. <a
   href="http://localhost:8981/dynamicHttpProxy/services/GreetingS
```

Motivation

The main motivation for the dynamic HTTP proxy is the ability to be able to add new HTTP proxies with a minimum of effort and without having to restart the proxy.

Limitations of the Example Program

Lacking from the example program to make it useable in a production environment are:

- Error handling.
- Retrieval of configuration from database.
  In the example, a simple map is used to store mapping between the HTTP relative path and the destination server. This does of course not allow for dynamically modifying the proxy configuration.
- Support for additional HTTP verbs.

In the example program only support for the HTTP verbs GET and POST have been implemented. It is trivial to add support for additional HTTP verbs as needed.

- Handling of HTTP parameters.
  The example program does not consider HTTP parameters but these are considered to be part of the HTTP relative path.
- Support for HTTPS.

There are probably additional things that one would consider lacking – feel free to add suggestions in the comments!

A Service to Proxy

The example program will be implemented in a Mule Project in SpringSource Tool Suite with the MuleStudio plug-in installed. Any Eclipse-based IDE with the MuleStudio plug-in installed.

In order to be have a service to proxy, a simple SOAP greeting-service is implemented using one Mule configuration file and one Java class.
The Mule configuration contains the following configuration:

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <mule
03.   xmlns:cxf="http://www.mulesoft.org/schema/mule/cxf"
04.   xmlns="http://www.mulesoft.org/schema/mule/core"
05.   xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
06.   xmlns:spring="http://www.springframework.org/schema/beans"
07.   xmlns:test="http://www.mulesoft.org/schema/mule/test"
08.   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
09.   xsi:schemaLocation="
10. <a
      href="http://www.mulesoft.org/schema/mule/cxf">http://www.mulesoft.org/schema/mule/
      <a href="http://www.mulesoft.org/schema/mule/cxf/current/mule-
      cxf.xsd">http://www.mulesoft.org/schema/mule/cxf/current/mule-cxf.xsd<;/a>
11. <a
      href="http://www.springframework.org/schema/beans">http://www.springframework.org/
      <a href="http://www.springframework.org/schema/beans/spring-beans-
      current.xsd">http://www.springframework.org/schema/beans/spring-beans-
      current.xsd<;/a>
12. <a
      href="http://www.mulesoft.org/schema/mule/core">http://www.mulesoft.org/schema/mul
      <a
      href="http://www.mulesoft.org/schema/mule/core/current/mule.xsd">http://www.mulesof
13. <a
      href="http://www.mulesoft.org/schema/mule/test">http://www.mulesoft.org/schema/mule
      <a href="http://www.mulesoft.org/schema/mule/test/current/mule-
      test.xsd">http://www.mulesoft.org/schema/mule/test/current/mule-test.xsd"</a>>
14.
15.   <spring:beans>
16.     <spring:bean id="helloService"
          class="com.ivan.mule.dynamichttpproxy.HelloService"/>
17.   </spring:beans>
18.
19.   <flow name="GreetingFlow">
20.     <inbound-endpoint address="http://localhost:8182/services/GreetingService"
21.       exchange-pattern="request-response"/>
22.
23.     <cxf:jaxws-service
          serviceClass="com.ivan.mule.dynamichttpproxy.HelloService"/>
24.     <component>
25.       <spring-object bean="helloService"/>
26.     </component>
27.   </flow>
28. </mule>
```

The Java class implementing the service looks like this:

```
01. package com.ivan.mule.dynamichttpproxy;
02.
03. import java.util.Date;
04. import javax.jws.WebParam;
05. import javax.jws.WebResult;
06. import javax.jws.WebService;
07.
08. /**
09. * SOAP web service endpoint implementation class that implements
10. * a service that extends greetings.
11. *
12. * @author Ivan Krizsan
13. */
14. @WebService
15. public class HelloService {
16.   /**
17.   * Greets the person with the supplied name.
18.   *
19.   * @param inName Name of person to greet.
20.   * @return Greeting.
21.   */
22.   @WebResult(name = "greeting")
23.   public String greet(@WebParam(name = "inName") final String inName) {
24.     return "Hello " + inName + ", the time is now " + new Date();
25.   }
26. }
```

Server Information Bean Class

Instances of the server information bean class holds information about a server which to forward requests to.

```
01. package com.ivan.mule.dynamichttpproxy;
02.
03. /**
04. * Holds information about a server which to forward requests to.
05. *
06. * @author Ivan Krizsan
07. */
08. public class ServerInformationBean {
09.   private String serverAddress;
10.   private String serverPort;
11.   private String serverName;
12.
13.   /**
14.   * Creates an instance holding information about a server with supplied
15.   * address, port and name.
16.   *
17.   * @param inServerAddress
18.   * @param inServerPort
19.   * @param inServerName
20.   */
21.   public ServerInformationBean(final String inServerAddress,
22.     final String inServerPort, final String inServerName) {
23.     serverAddress = inServerAddress;
24.     serverPort = inServerPort;
25.     serverName = inServerName;
26.   }
27.
28.   public String getServerAddress() {
29.     return serverAddress;
30.   }
31.
32.   public String getServerPort() {
33.     return serverPort;
34.   }
35.
36.   public String getServerName() {
37.     return serverName;
38.   }
39. }
```

The reasons for storing this information in a dedicated bean class is to make it easy to extend the class with additional information, to facilitate migration of storage to a database and to keep the different kinds of data stored in the Mule context to a minimum.

Dynamic HTTP Proxy Mule Configuration

The dynamic HTTP proxy Mule configuration is implemeted as follows:

```
001. <?xml version="1.0" encoding="UTF-8"?>
002. <!--
```

```
  The dynamic HTTP proxy Mule configuration file.

  Author: Ivan Krizsan
-->
<mule xmlns:scripting="http://www.mulesoft.org/schema/mule/scripting"
  xmlns:http="http://www.mulesoft.org/schema/mule/http"
  xmlns="http://www.mulesoft.org/schema/mule/core"
  xmlns:doc="http://www.mulesoft.org/schema/mule/documentation"
  xmlns:spring="http://www.springframework.org/schema/beans"
  xmlns:util="http://www.springframework.org/schema/util"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:test="http://www.mulesoft.org/schema/mule/test"
  version="CE-3.4.0"
  xsi:schemaLocation="http://www.springframework.org/schema/beans <a
     href="http://www.springframework.org/schema/beans/spring-beans-
     current.xsd">http://www.springframework.org/schema/beans/spring-beans-
     current.xsd<;/a>
<a
    href="http://www.springframework.org/schema/util">http://www.springframework.org/sch
    <a href="http://www.springframework.org/schema/util/spring-util-
    current.xsd">http://www.springframework.org/schema/util/spring-util-current.xsd<;/a>
<a
    href="http://www.mulesoft.org/schema/mule/core">http://www.mulesoft.org/schema/mul
    <a
    href="http://www.mulesoft.org/schema/mule/core/current/mule.xsd">http://www.mulesof
<a
    href="http://www.mulesoft.org/schema/mule/http">http://www.mulesoft.org/schema/mule
    <a href="http://www.mulesoft.org/schema/mule/http/current/mule-
    http.xsd">http://www.mulesoft.org/schema/mule/http/current/mule-http.xsd<;/a>
<a
    href="http://www.mulesoft.org/schema/mule/test">http://www.mulesoft.org/schema/mule
    <a href="http://www.mulesoft.org/schema/mule/test/current/mule-
    test.xsd">http://www.mulesoft.org/schema/mule/test/current/mule-test.xsd<;/a>
<a
    href="http://www.mulesoft.org/schema/mule/scripting">http://www.mulesoft.org/schema/
    <a href="http://www.mulesoft.org/schema/mule/scripting/current/mule-
    scripting.xsd">http://www.mulesoft.org/schema/mule/scripting/current/mule-
    scripting.xsd"</a>>

  <spring:beans>
    <!--
      Mappings from path to server represented by a hash map.
      A map has been choosen to limit the scope of this example.
      Storing data about mappings between path to server in a database
      will enable runtime modifications to the mapping data without
      having to stop and restart the general proxy Mule application.
    -->
    <util:map id="pathToServerAndPortMapping" map-
       class="java.util.HashMap">
      <!-- Entry for MyServer. -->
      <spring:entry key="services/GreetingService">
        <spring:bean
           class="com.ivan.mule.dynamichttpproxy.ServerInformationBean">
          <spring:constructor-arg value="localhost"/>
          <spring:constructor-arg value="8182"/>
          <spring:constructor-arg value="MyServer"/>
        </spring:bean>
      </spring:entry>
      <!-- Entry for SomeOtherServer. -->
      <spring:entry key="services/GreetingService?wsdl">
        <spring:bean
           class="com.ivan.mule.dynamichttpproxy.ServerInformationBean">
          <spring:constructor-arg value="127.0.0.1"/>
          <spring:constructor-arg value="8182"/>
          <spring:constructor-arg value="SomeOtherServer"/>
        </spring:bean>
      </spring:entry>
    </util:map>
  </spring:beans>

  <flow name="HTTPGeneralProxyFlow">
    <!--
      Note that if you increase the length of the path to, for instance
      generalProxy/additionalPath, then the expression determining
      the outgoing path need to be modified accordingly.
      Changing the path, without changing its length, require no
      modification to outgoing path.
    -->
    <http:inbound-endpoint
      exchange-pattern="request-response"
      host="localhost"
      port="8981"
```

```
063.     path="dynamicHttpProxy" doc:name="HTTP Receiver"/>
064.
065.   <!-- Extract outgoing path from received HTTP request. -->
066.   <set-property
067.     value="#
          [org.mule.util.StringUtils.substringAfter(org.mule.util.StringUtils
          '/'), '/')]"
068.     propertyName="outboundPath"
069.     doc:name="Extract Outbound Path From Request" />
070.
071.   <logger message="#[string:Outbound path = #
          [message.outboundProperties['outboundPath']]]" level="DEBUG"/>
072.
073.   <!--
074.     Using the HTTP request path, select which server to forward the
          request to.
075.     Note that there should be some kind of error handling in case there
          is no server for the current path.
076.     Error handling has been omitted in this example.
077.   -->
078.   <enricher target="#[variable:outboundServer]">
079.     <scripting:component doc:name="Groovy">
080.       <!--
081.         If storing mapping data in a database, this Groovy script
082.         should be replaced with a database query.
083.       -->
084.       <scripting:script engine="Groovy">
085.         <![CDATA[
086.           def theMap =
                muleContext.getRegistry().lookupObject("pathToServerAndPortMapp
087.           def String theOutboundPath =
                message.getOutboundProperty("outboundPath")
088.           def theServerBean = theMap[theOutboundPath]
089.           theServerBean
090.         ]]>
091.       </scripting:script>
092.     </scripting:component>
093.   </enricher>
094.
095.   <logger
096.     message="#[string:Server address = #
          [groovy:message.getInvocationProperty('outboundServer').serverAddre
097.     level="DEBUG"/>
098.   <logger
099.     message="#[string:Server port = #
          [groovy:message.getInvocationProperty('outboundServer').serverPort]
100.     level="DEBUG"/>
101.   <logger
102.     message="#[string:Server name = #
          [groovy:message.getInvocationProperty('outboundServer').serverName]
103.     level="DEBUG"/>
104.
105.   <!-- Log the request and its metadata for development purposes, -->
106.   <test:component logMessageDetails="true"/>
107.
108.   <!--
109.     Cannot use a MEL expression in the value of the method attribute
110.     on the HTTP outbound endpoints so have to revert to this way of
111.     selecting HTTP method in the outgoing request.
112.     In this example, only support for GET and POST has been implemented.
113.     This can of course easily be extended to support additional HTTP
114.     verbs as desired.
115.   -->
116.   <choice doc:name="Choice">
117.     <!-- Forward HTTP GET requests. -->
118.     <when expression="#
          [message.inboundProperties['http.method']=='GET']">
119.       <http:outbound-endpoint
120.         exchange-pattern="request-response"
121.         host="#
              [groovy:message.getInvocationProperty('outboundServer').serverAd
122.         port="#
              [groovy:message.getInvocationProperty('outboundServer').serverPo
123.         method="GET"
124.         path="#[message.outboundProperties['outboundPath']]"
125.         doc:name="Send HTTP GET"/>
126.     </when>
127.     <!-- Forward HTTP POST requests. -->
128.     <when expression="#
          [message.inboundProperties['http.method']=='POST']">
129.       <http:outbound-endpoint
130.         exchange-pattern="request-response"
131.         host="#
```

```
            [groovy:message.getInvocationProperty('outboundServer').serverAd
        port="#
            [groovy:message.getInvocationProperty('outboundServer').serverPo
        method="POST"
        path="#[message.outboundProperties['outboundPath']]"
        doc:name="Send HTTP POST"/>
    </when>
    <!-- If HTTP method not recognized, use GET. -->
    <otherwise>
     <http:outbound-endpoint
       exchange-pattern="request-response"
       host="#
            [groovy:message.getInvocationProperty('outboundServer').serverAd
       port="#
            [groovy:message.getInvocationProperty('outboundServer').serverPo
       method="GET"
       path="#[message.outboundProperties['outboundPath']]"
       doc:name="Default: Send HTTP GET"/>
    </otherwise>
   </choice>
  </flow>
</mule>
```

Note that:

- A map named “pathToServerAndPortMapping” is configured using Spring XML.
  This map contains the mapping between context and relative path of an URL to the server to which requests are to be forwarded, as discussed above.
- The map contains an entry for “services/GreetingService?wsdl”.
  As discussed in the section on limitations of the example program, it currently does not handle HTTP parameters. I also wanted more than one single mapping in order to make the example more interesting.
- There is a <set-property> element setting the property “outboundPath” immediately after the HTTP inbound endpoint.
  The slightly complicated expression in the value attribute is used to remove the context part of incoming HTTP requests. The context part of the dynamic HTTP proxy can be changed without requiring modifications of the expression. However, if you want to add another part to the URL which should not be regarded when determining which server to forward a request to, this expression need to be modified.
- An <enricher> is used to retrieve the correct instance of the *ServerInformationBean* class.
  Instead of using a Groovy script, the enricher should perform a database query.
  In addition, there is no error handling for the case where there is no server information available for a particular key.
- There is a <choice> element containing multiple outbound HTTP endpoints.
  The outbound HTTP endpoints only differ as far as the *method* attribute is concerned. The reason for having to use the <choice> element and multiple HTTP outbound endpoints is that Mule does not allow for expressions to be entered in the *method* attribute.

Test the Example Program

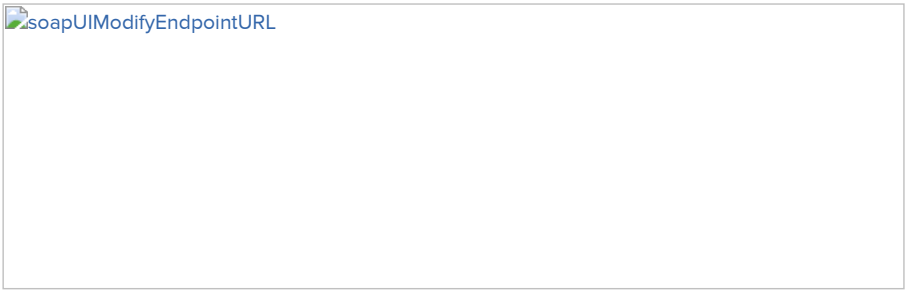
The example program is now complete and can be started by right-clicking the project in the IDE and selecting Run As -> Mule Application.
When the Mule instance has started up, try issuing a reques to the following URL in a browser of your choice:

```
1. http://localhost:8182/services/GreetingService?wsdl
```

You should see the WSDL of the greeting service.
Using soapUI, try sending a request to the greeting service. You should receive a greeting containing the current date and time.
Next, add a new endpoint to the request in soapUI and enter the following URL:

```
1. http://localhost:8981/dynamicHttpProxy/services/GreetingService
```

Then send the request again from soapUI. You should receive the same kind of response as when communicating directly with the greeting service:



If examining the console log in the IDE, output similar to the following four lines should be present (if not, try changing the log level to ERROR and repeat send a request again):

```
... Outbound path = services/GreetingService
... Server address = localhost
... Server port = 8182
... Server name = MyServer
```

In a browser, issue a request for the greeting service's WSDL using the following URL:

```
http://localhost:8981/dynamicHttpProxy/services/GreetingService
```

The four lines of console output sawn earlier now changes to:

```
... Outbound path = services/GreetingService?wsdl
... Server address = localhost
... Server port = 8182
... Server name = SomeOtherServer
```

From this we can see that different mappings come into effect depending on the outbound part of the URL.

Tags: enterprise integration esb integration Mule Mule ESB Tutorial Integration

The Enterprise Integration Zone is brought to you in partnership with WSO2. Learn more about WSO2's API Management.

Code Golf: Smile!

We're Going to Learn A TON About How We Monitor Performance

POPULAR ON JAVALOBBY

· Spring Batch - Hello World
· Is Hibernate the best choice?
· How to Create Visual Applications in Java?
· 9 Programming Languages To Watch In 2011
· Introduction to Oracle's ADF Faces Rich Client Framework
· Interview: John De Goes Introduces a Newly Free Source Code Editor
· Lucene's FuzzyQuery is 100 times faster in 4.0
· Time Slider: OpenSolaris 2008.11 Killer Feature

LATEST ARTICLES

· Platform thinking
· Paper explores how technology will underpin change
· Meet Rebecca Lieb of Altimeter Group
· Hack up a Simple JDBC ResultSet Cache Using jOOQ's MockDataProvider
· The Cloudcast #184 - Streaming Analytics for Distributed Applications
· Excerpts From the RavenDB Performance Team Report: JSON & Structs in Voron
· The Dinovator Movement is Better Than the Bitpipe Accelerator Den...
· Will Apple Watch Transform Enterprise IT?

SPOTLIGHT RESOURCES

Practical DNS: Managing Domains for Safety, Reliability, and Speed

Essential Couchbase APIs: Open Source NoSQL Data Access from Java, Ruby, and .NET

Camel Essential Components

DZone's 170th Refcard is an essential reference to Camel, an open-source, lightweight, integration library. This Refcard is authored by...

Search

DZone

Refcardz
Tech Library
Snippets
About DZone
Tools & Buttons
Book Reviews
IT Questions
My Profile
Advertise
Send Feedback

Topics

HTML5
Cloud
.NET
PHP
Performance
Agile
Windows Phone
Mobile
Java
Eclipse
Big Data
DevOps

Follow Us

Google +
Facebook
LinkedIn
Twitter

"Starting from scratch" is seductive but disease ridden
-Pithy Advice for Programmers