2 Scaling Integration Infrastructure with Oracle Service Bus

2.1 Prerequisites

- Chapter One Lab complete, working and deployed.
- SOA and Oracle Service Bus server running.
- Credit Services deployed to OSB.

2.2 Introduction

Note: The solution for this chapter can be found in `soa-osb/solutions/chap-2.` The solution includes both and Lab A and Lab B import files called `chap-2A_solution_sbconfig.jar` and `chap-2B_solution-caching.jar` respectively.

With the new POProcessing system in place, Pega's widget supply business continues to grow rapidly. However, with growth there are now some problems developing from Customer Support. Recently, Pega's Customer Support has alerted the IT department there have been complaints regarding certain aspects of the new POProcessing service.

Many Purchase Orders are being rejected due to credit card transactions processing failures. The failure rate has been thirty percent and deemed unacceptable by Mega Corporation, one of Pega's largest customers. Mega, who should never have a declined credit issue, has escalated to Pega's Executive Mangement. Rapid improvements must occur in the next 30 days or they will consider switching to another supplier.

Upon investigation of the complaints, Pega's IT identified two issues:

1. The current Credit Service Validation Service Provider is unreliable and cannot handle the load required by Pega's growing customer base.
2. Due to the failures, Mega keeps checking status of the PO orders causing unnecessary load and spikes on Pega's infrastructure.

Upon completion of this Lab, you will learn how Oracle Service Bus can help provide solutions to both issues. OSB will help Pega scale their integration infrastructure through **Endpoint Management** and **Service Result Caching**.

2.3 Endpoint Management

Note: The solution for this section can be found in `/home/oracle/soa-osb/solutions/chap-2/chap-2A_solution_sbconfig.jar`

To provide insulation from unreliable Credit Card Vendors and also provide additional capacity for increased load in Credit Card Validation requests, Pega will now contract with multiple Credit Card Payment Vendors to ensure high quality of service for POProcessing thus insulating customers from any disruptions of services caused by backend Service Providers.

Luckily, the Credit Card Validation service is already virtualized by Oracle Service Bus. Pega's IT can make updates quickly and easily without any impact to either the Customer (Service Consumer) or the backend Application Developers of the POProcessing service (Service Provider).

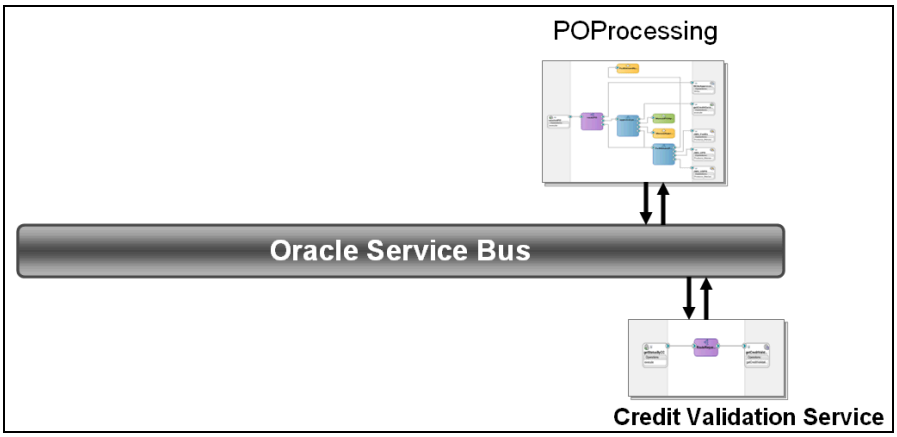
Oracle Bus is the Solution

In this lab, you will bring on-line another credit validation vendor leveraging the advanced Endpoint Management features of Oracle Service Bus.

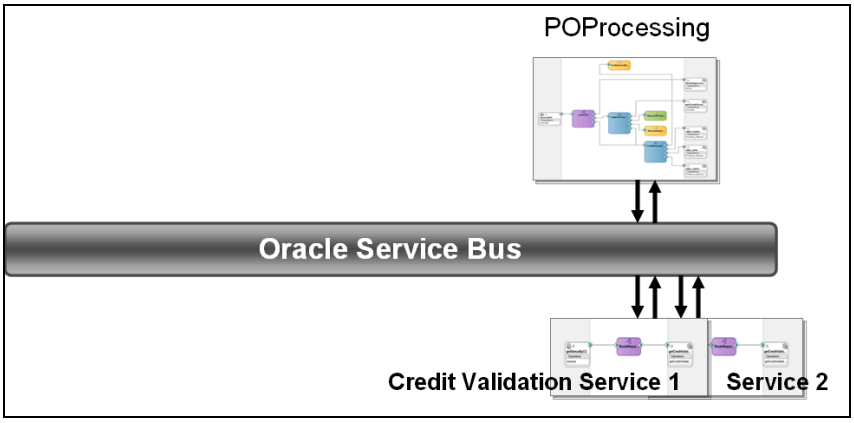
By leveraging multiple endpoints, Oracle Service Bus can guarantee service availability even if one endpoint is offline. This is an infrastructure resiliency proof point for Pega's IT organization, which is under constant pressure and understaffed. Moreover OSB will load balance service requests across endpoints according to a variety of algorithms. For example, if one service endpoint were more expensive to use, a weighted message dispatching algorithm can be configured to favor the less expensive service, while still guaranteeing message routing to all available endpoints.

As if this were not enough business value, OSB raises the bar even further by dynamically adjusting message routing across multiple endpoints in the event that one or more are unavailable. OSB automatically checks offline service endpoints for their availability, and adds them back into the active service endpoint pool when they are back online.

Before Endpoint Management – Single Endpoint



After Endpoint Management – Multiple Endpoints add Agility and Scale



Key Takeaway: Changes are made without any impact to Service Consumers or Providers.

Oracle Service Bus helps insulate service consumers and service providers from these types of changes, allowing Pega's business to be more agile, robust and adaptable. Changes are made without any impact to Service Consumers or Providers.

2.2.4 What is being done?

During this section you will perform the following steps.

- You will start by re-configuring the *validationForCC* **Business Service** to have multiple endpoints. The Business Service will now load-balance across the two service providers: the original endpoint and new provider endpoint just added.
- You will test and Monitor endpoint utilization from the OSB Monitoring console by invoking **ValidateCredit** proxy service several times from OSB Test console.

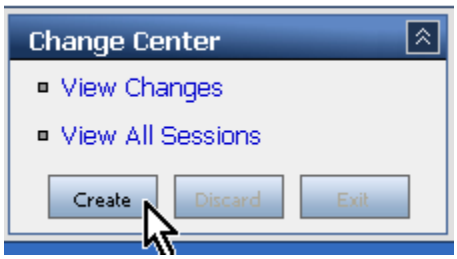
2.3.4 High Level Steps

- Add endpoint http://<IP ADDRESS>:7001/Credit_Services/ProxyServices/ValidateCredit to the *validationForCC* business service.
- Set the load balancing algorithm to round-robin.
- Test the **ValidateCredit** proxy service multiple times to distribute service requests across both service endpoints.
- Validate message distribution to multiple endpoints through the OSB Operations console.

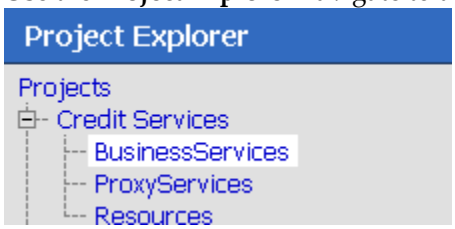
2.4.4 Steps in Detail

Add the New Credit Card Validation Service Endpoint

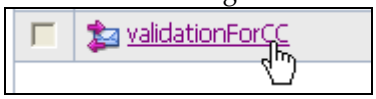
1. Login to the Service Bus console (http://localhost:7001/sbconsole) **username** weblogic, **password** welcome1.
2. Create a session by clicking the ***Create*** button in the **Change Center.**



3. Use the **Project Explorer** navigate to the **BusinessServices** folder of the **Credit Services** project.



4. To access the configuration for business service **validationForCC**, click on it's name.



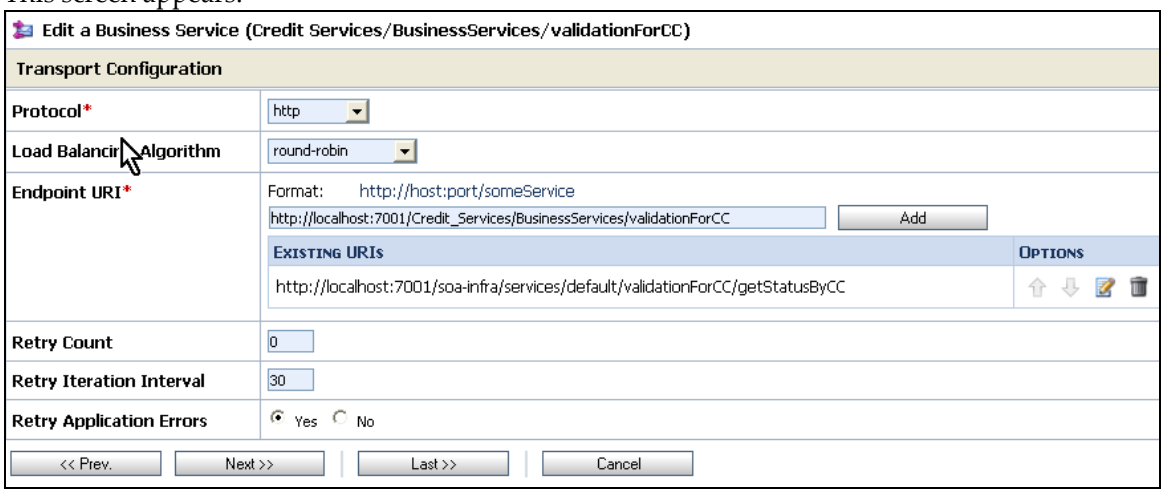
Notice that there is a single service endpoint defined.

Endpoint URI	http://localhost:7001/soa-infra/services/default/validationForCC/getStatusByCC

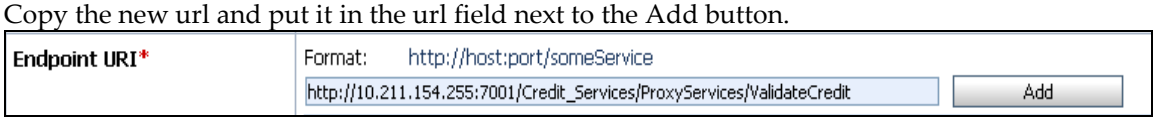
5. We are going to add a 2nd service endpoint and we have been provided the service url. The new service url will be provided by the **INSTRUCTOR. http://<IP-ADDRESS>:7001/Credit_Services/ProxyServices/ValidateCredit**. To start the process of adding a 2nd

service endpoint, click the edit icon in the Transport Configuration header bar.

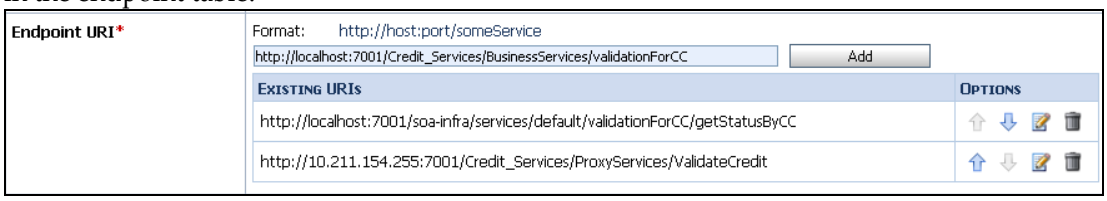
6. This screen appears.



7. Copy the new url and put it in the url field next to the Add button.



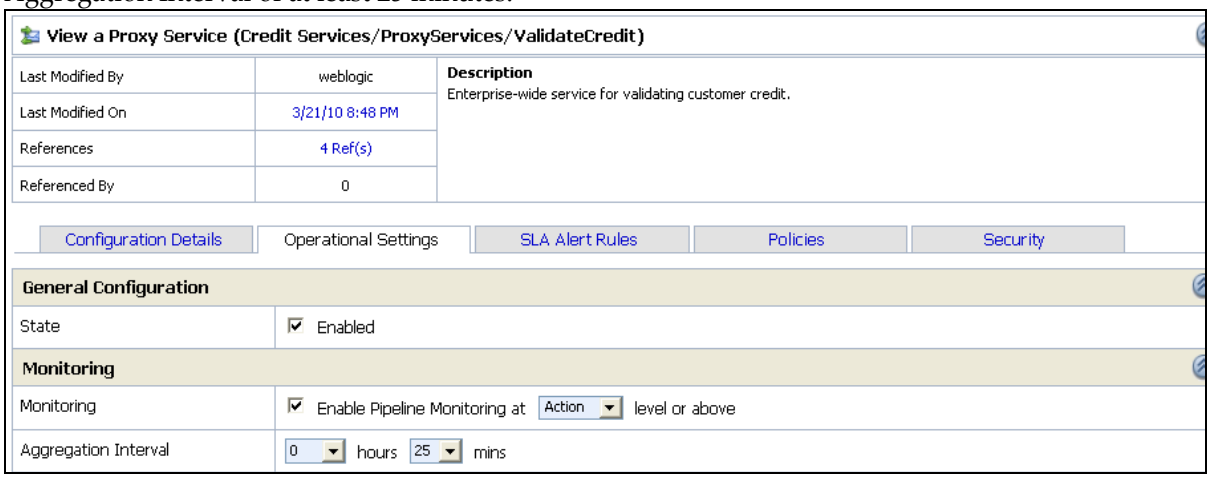
8. Click the **Add** button to add the 2nd url to the list of available endpoints. There should now be 2 urls in the endpoint table.



9. Ensure the endpoint load balancing algorithm is set to round-robin.

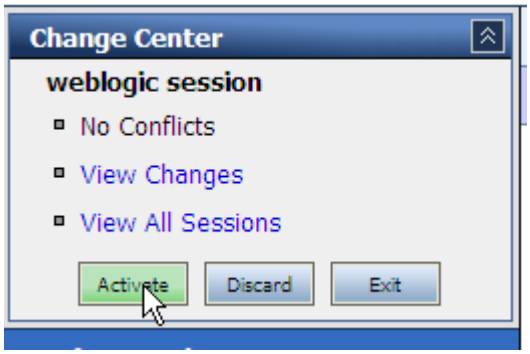
10. There are no more configuration actions to take. Click the **Last** button then scroll to the bottom of the page and click the **Save** button.

11. Ensure that proxy service **ValidateCredit** is configured with Monitoring enabled, with an Aggregation Interval of at least 25 minutes.

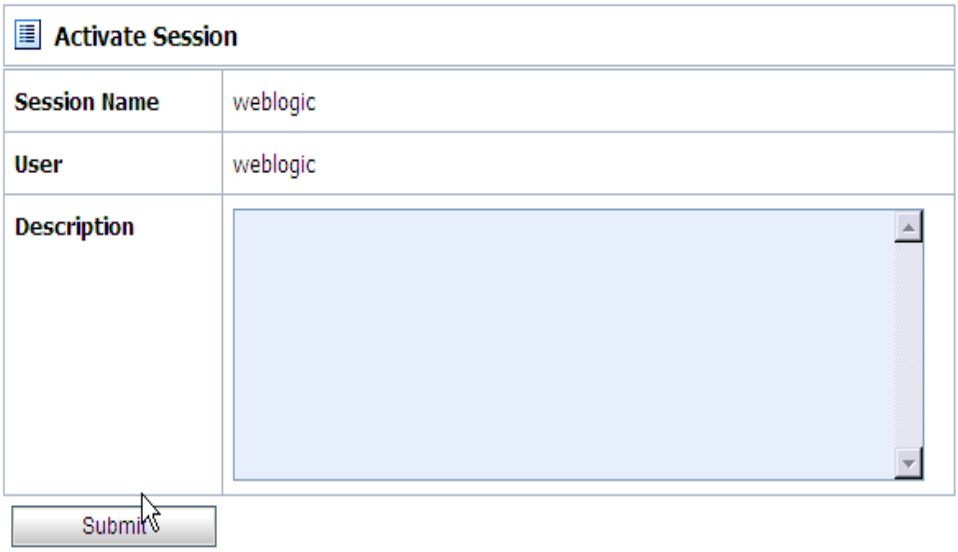


Activate Session

Activate the Session by clicking on **Activate** in **Change Center**



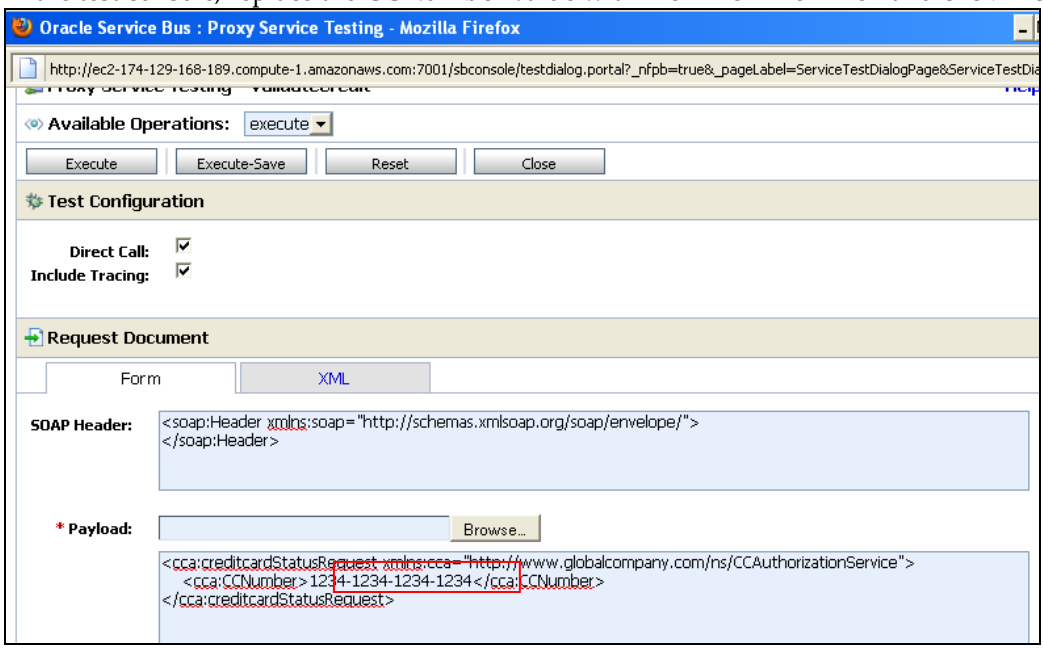
Add optional **Description** and click **Submit**



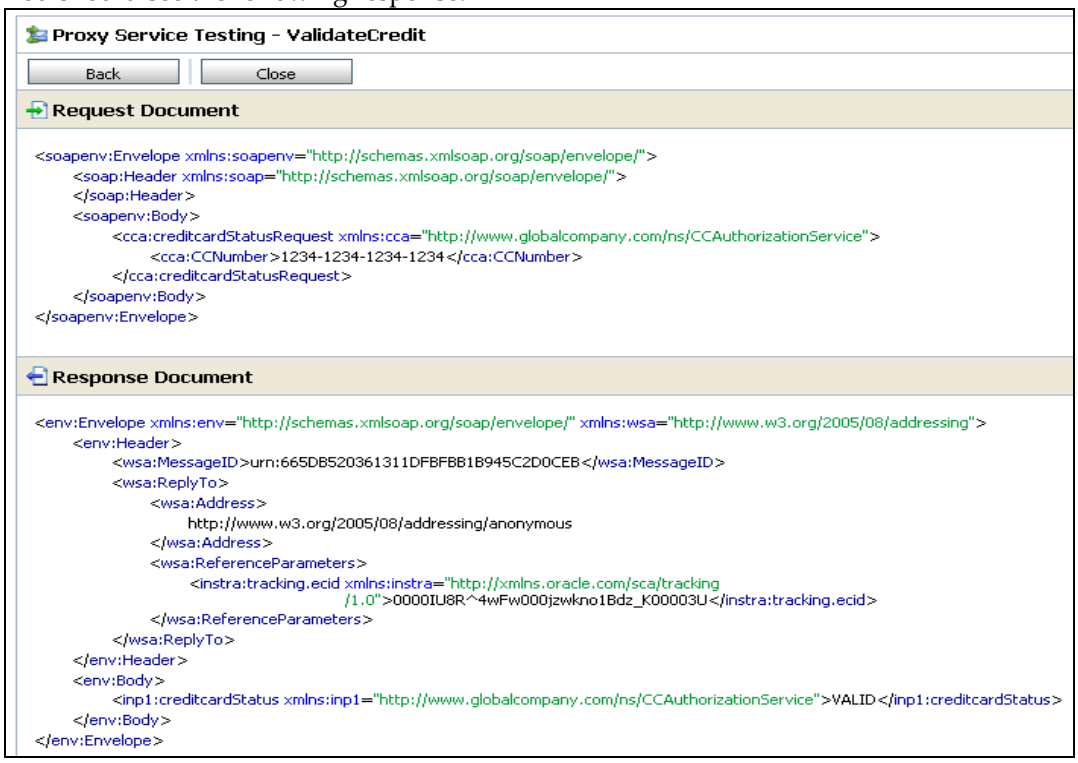
2.5.4 Test

12. In order to test the changes find the **ValidateCredit** proxy service by navigating to **Resource Browser** followed by **Proxy Services**. Next, launch the test console by clicking on the icon under **Actions**.

13. In the test console, replace the **CCNumber** value with **1234-1234-1234-1234** and click **Execute**.



14. You should see the following response.

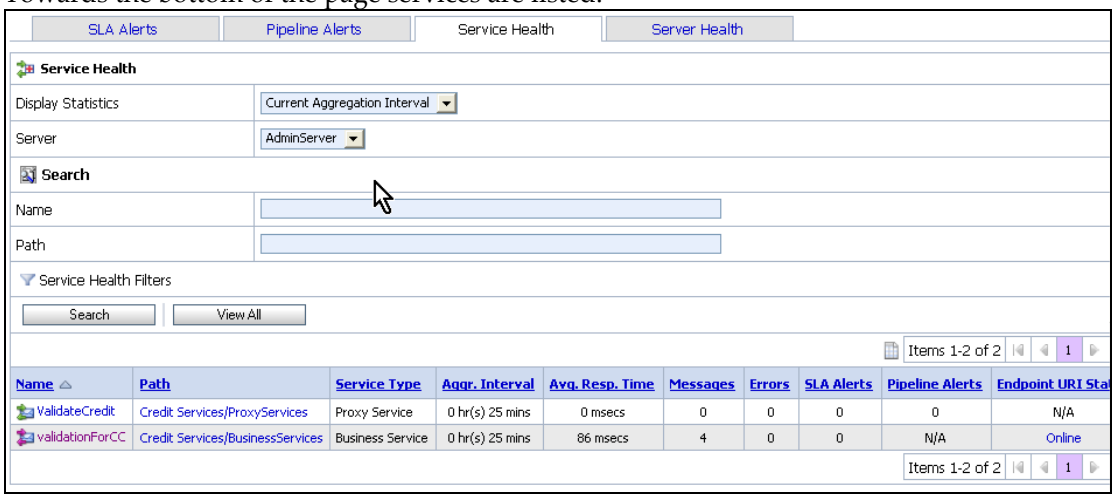


15. Click the **Back** button and run the test a few more times.

16. Navigate to **Operations**.

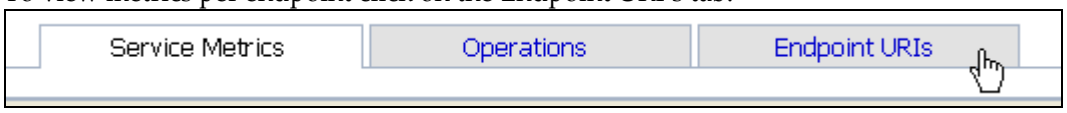
17. Click on the **Service Health** tab.

18. Towards the bottom of the page services are listed.

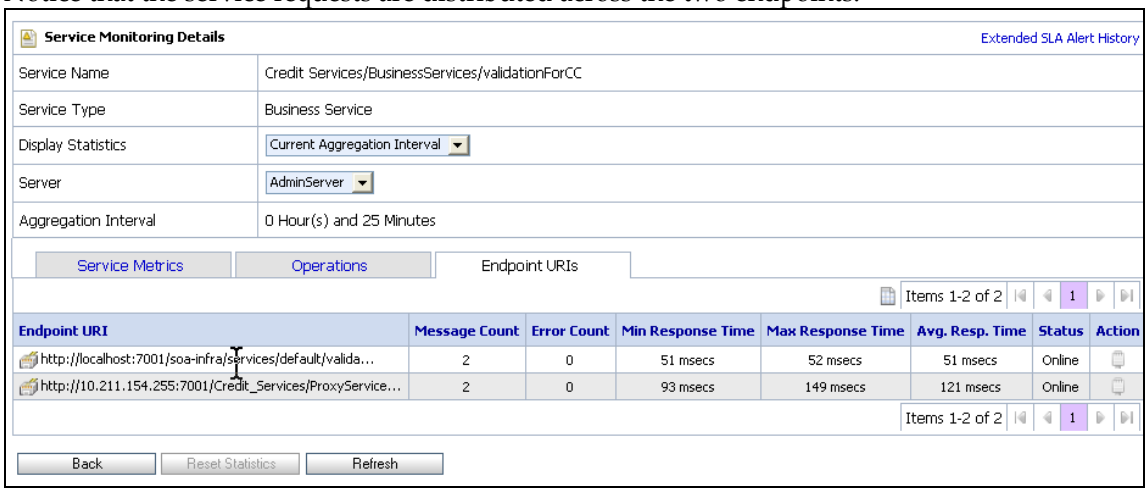


19. Click on the **validationForCC** business service to view the aggregate service metrics. Take a minute to browse the metrics page and get familiar with the available data.

20. To view metrics per endpoint click on the Endpoint URI's tab.



21. Notice that the service requests are distributed across the two endpoints.



Service Monitoring Details Extended SLA Alert History

Service Name	Credit Services/BusinessServices/validationForCC
Service Type	Business Service
Display Statistics	Current Aggregation Interval
Server	AdminServer
Aggregation Interval	0 Hour(s) and 25 Minutes

Service Metrics | Operations | Endpoint URIs

Items 1-2 of 2 1

Endpoint URI	Message Count	Error Count	Min Response Time	Max Response Time	Avg. Resp. Time	Status	Action
http://localhost:7001/soa-infra/services/default/valida...	2	0	51 msecs	52 msecs	51 msecs	Online	
http://10.211.154.255:7001/Credit_Services/ProxyService...	2	0	93 msecs	149 msecs	121 msecs	Online	

Items 1-2 of 2 1

Back | Reset Statistics | Refresh

2.2 Service Result Caching

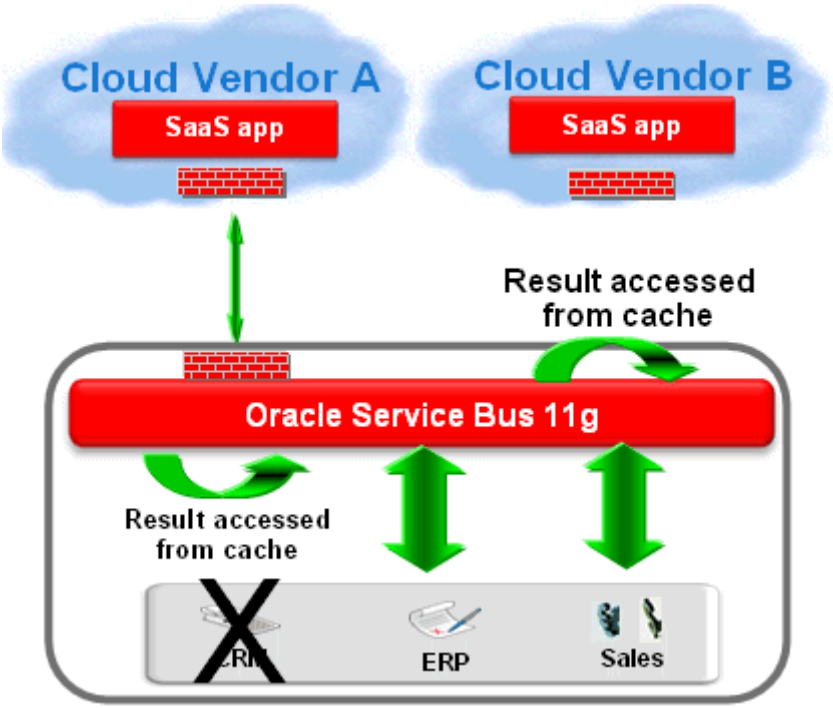
Note: The solution for this section can be found in `/home/oracle/soa-osb/solutions/chap-2/chap-2B_solution-caching.jar`. In order to develop the solution from scratch you will start with `/home/oracle/soa-osb/solutions/chap-2/chap-2B_starting-caching.jar`

LabB

Problem: Because of the CreditCard Validation issues, Mega keeps repetitively checking Purchase Order status causing unnecessary load and spikes on Pega's infrastructure.

Implement the Solution

Pega's IT will leverage OSB's Service Result Cache to normalize the spikes for repetitive service calls. By caching the results of repetitive service calls, Oracle Service Bus can help scale and protect backend systems, avoiding outages caused by spikes enabling predictable scalability. The backend service can even temporarily go down yet cached results can still be returned (within a predefined time window) to ensure no loss in availability.



2.2.4 What is being done?

During this section you will perform the following steps.

- Configure Service Result Caching on the *GetPO* Business Service, which retrieves the Purchase Order given the ID of the Purchase Order. Enabling Result Caching will cache the Purchase Order in the Coherence Cache

- Test Service Result Caching functionality by invoking the *GetPO* Proxy Service, which Routes to the *GetPO* Business Service. First request for a Purchase Order will take 5 seconds, while the following requests (performed before the cache entry is expires from the Coherence Cache) will be much faster as the Purchase Order is served from the Coherence cache.

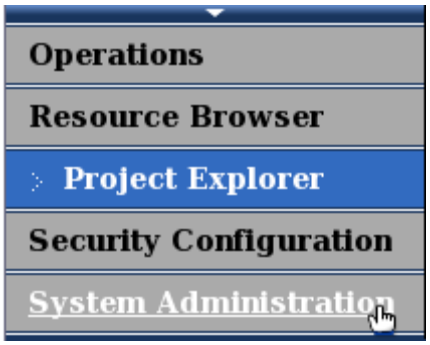
2.3.4 High Level Steps

- Import /home/oracle/soa-osb/solutions/chap-2/chap-2B_starting-caching.jar into OSB
- (Optional) Test to see that every request to the *GetPO* Proxy Service takes about 5 seconds as Service Result caching is not enabled on the *GetPO* Business Service to which the *GetPO* Proxy Service Routes. For simplicity, the *GetPO* Business Service invokes the POStatus/TestServices/POProvider test service on OSB that waits for 5 seconds before responding with a Purchase Order. In reality, this service can exist any where.
- Configure Service Result Caching on *GetPO* Business Service
  - **Cache Token Expression** – *$body/po:PO_ID*
  - **Expiration Time** – 1 minute
- Update *GetPO* Proxy Service to propagate *cache-originated* and *cache-token* response metadata (from the *GetPO* Business Service) as SOAP Headers. *cache-originated* and *cache-token* are available in the Proxy Service at `$outbound/ctx:transport/ctx:response/tp:cache-originated` and `$outbound/ctx:transport/ctx:response/tp:cache-token` respectively.
- Test to see that first request to the *GetPO* Proxy Service takes about 5 seconds. However, following requests for the same Purchase Order (same PO_ID) if made within 1 minute (Expiration Time) of the first request is much faster than 5 seconds as the Purchase Order is retrieved from the Coherence Cache. The request will take 5 seconds once the cache entry expires.

2.4.4 Steps in Detail

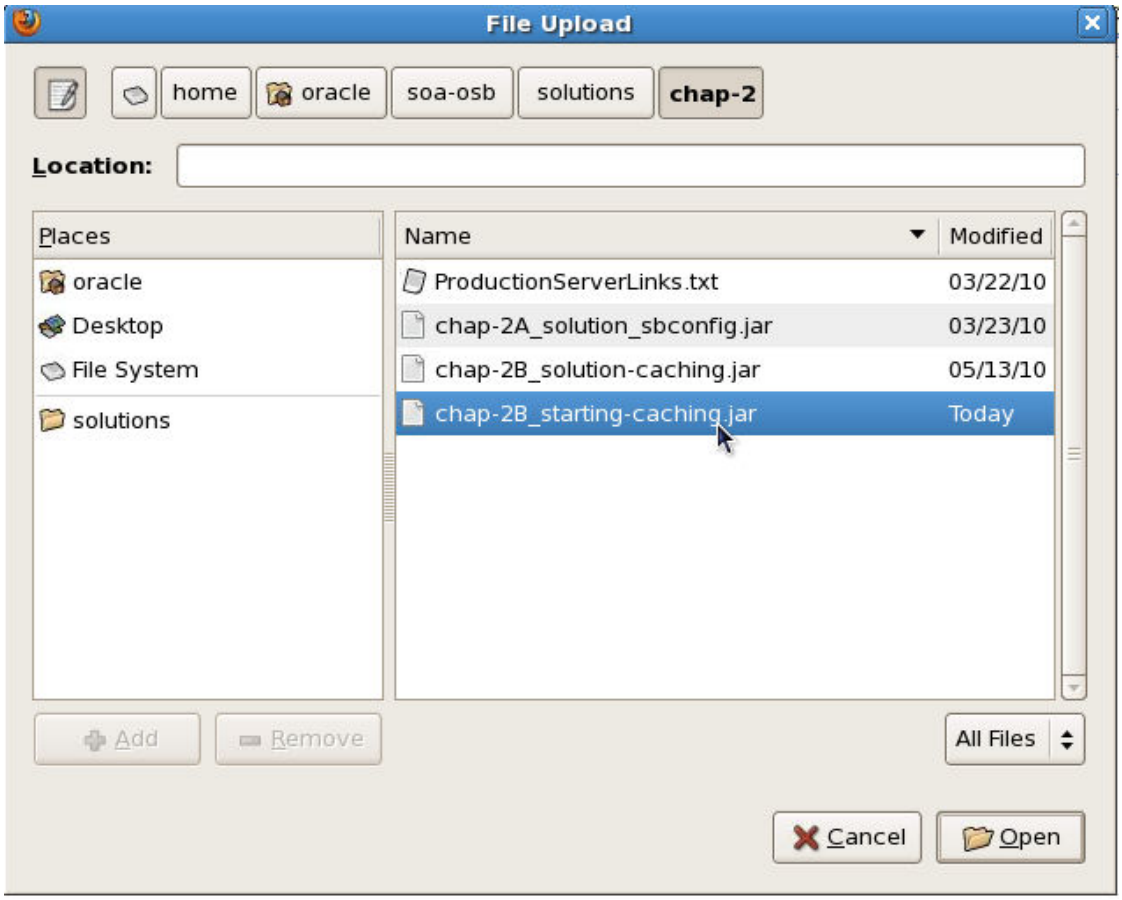
22. Create Session

23. Navigate to **System Administration**



24. In **Import Resources**, Click **Browse**

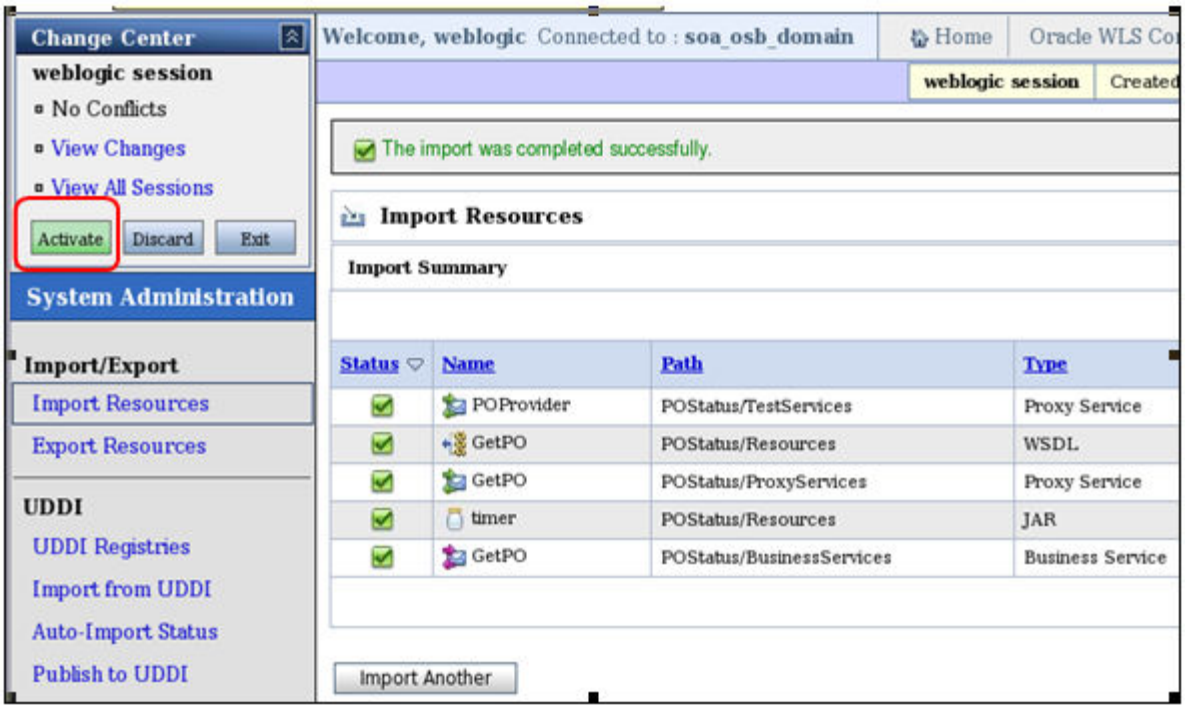
25. Select /home/oracle/soa-osb/solutions/chap-2/chap-2B_starting-caching.jar



26. Click **Next**

27. Click **Import**

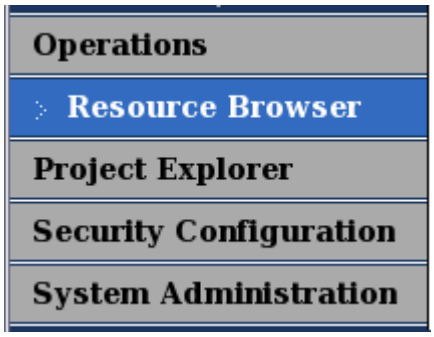
28. Click **Activate**



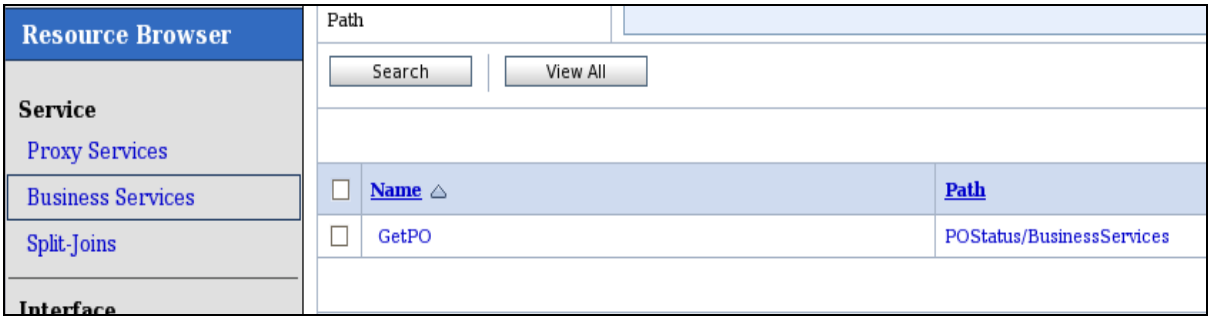
29. Click **Submit**

30. Create Session

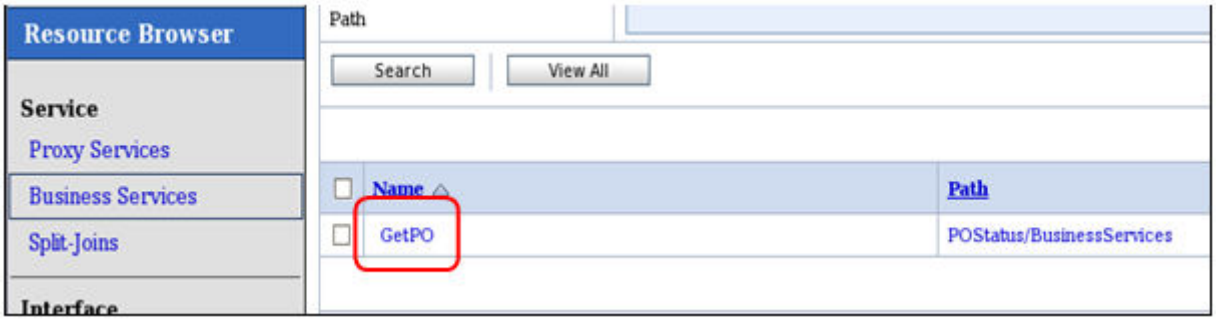
31. Click **Resource Browser**



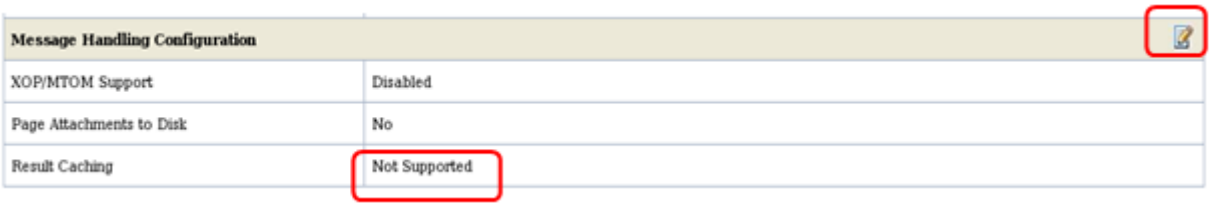
32. Select **Business Services**



33. Edit *GetPO* Business Service by clicking the service name



34. Click  on the **Message Handling Configuration** section to edit Result Caching. Note that currently **Result Caching** is set to *Not Supported*.



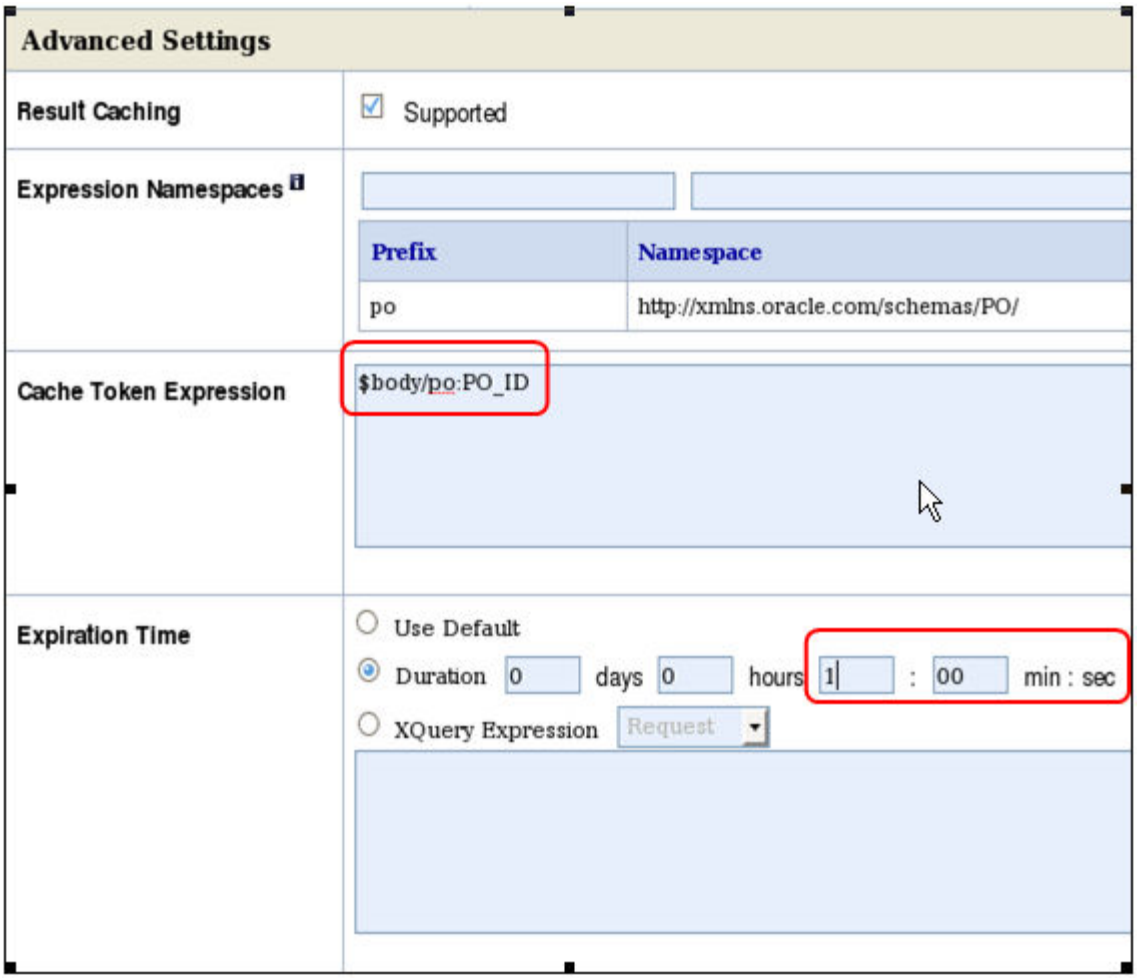
35. Expand **Advanced Settings** by clicking 

36. Check **Result Caching** to *Supported* to enable rest of the Result Caching configuration

37. Setup Result Caching configuration. The **Cache Token Expression** evaluates to the cache token part of the cache key used to cache the entry in Coherence Cache. The **Expiration Time** evaluates to the Time To Live for the cache entry.

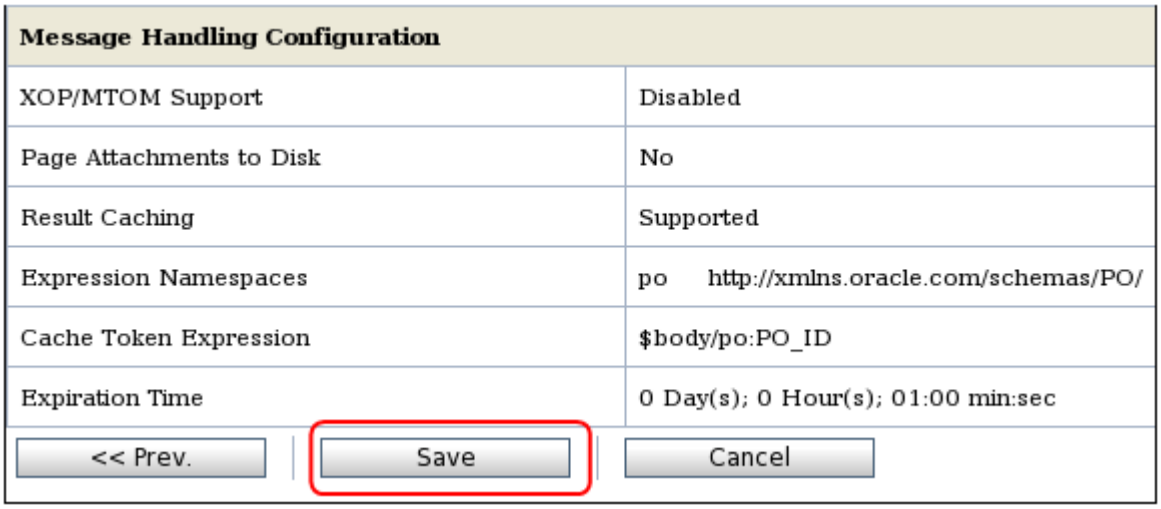
    Cache Token Expression: $body/po:PO_ID

    Expiration Time: 1 min



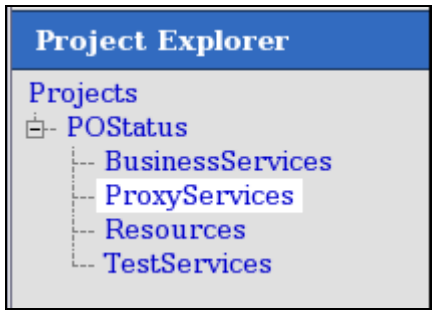
38. Click **Next**

39. Click **Save** (Do not miss this step)



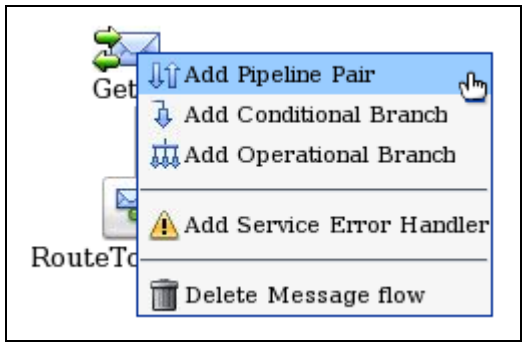
Next, Edit the Message Flow of *POStatus/ProxyServices/GetPO* to propagate *cache-originated* and *cache-token* response metadata as SOAP Headers

40. From the **Project Explorer**, select POStatus/ProxyServices

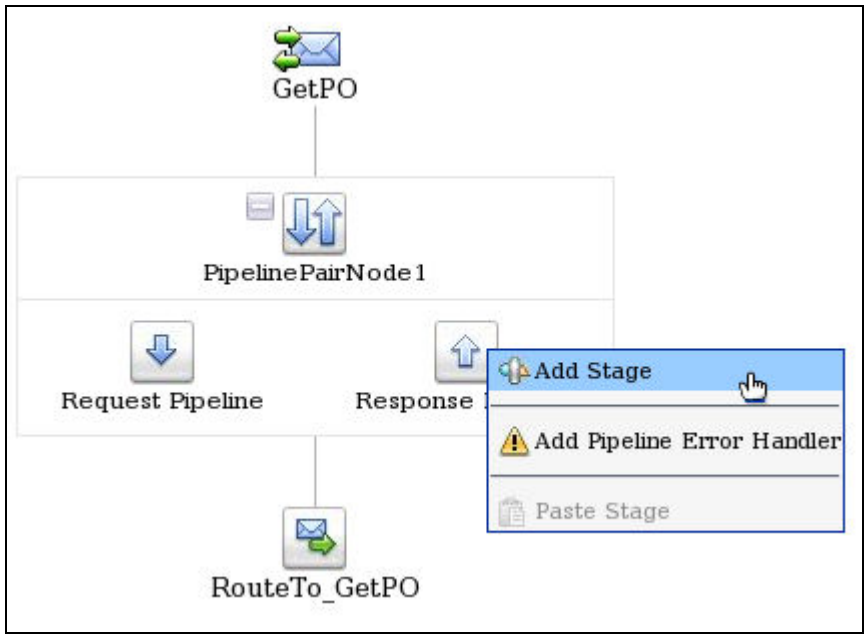


41. Click  for *GetPO* Proxy Service to edit the Message Flow

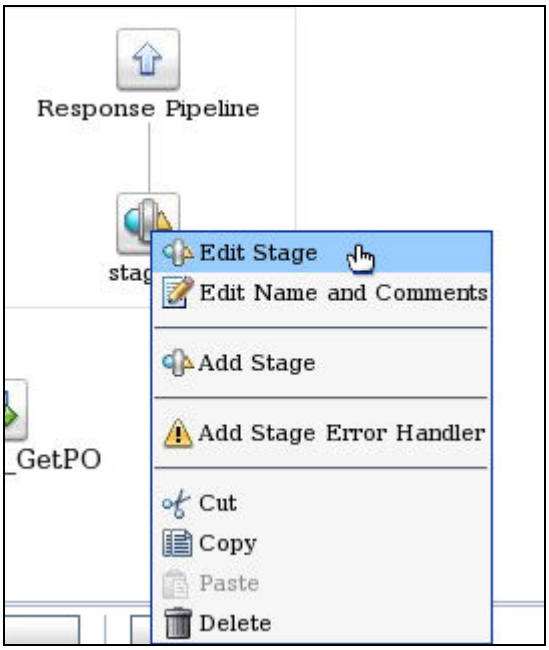
42. Add **Pipeline Pair** by clicking *GetPO* and selecting **Add Pipeline Pair**



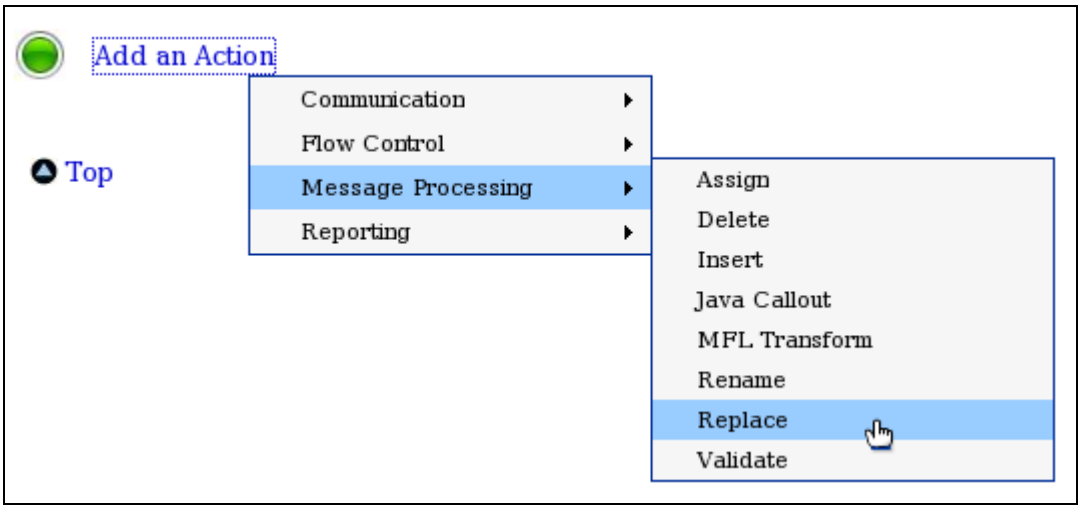
43. Add **Stage** to **Response Pipeline**



44. **Edit Stage**



45. Add a Replace action – Click **Add an Action** and Select **Message Processing->Replace**



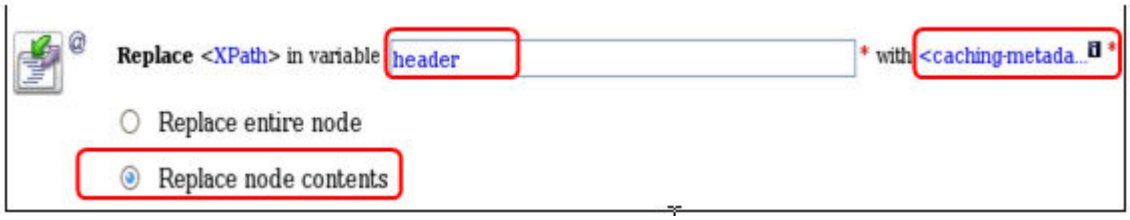
46. Edit Replace action

Variable: header

Expression: Click **<Expression>,** Add the following XML in the text area and Click **Save**

```
<caching-metadata>
  <cache-originated>{$outbound/ctx:transport/ctx:response/tp:cache-originated/text()}</cache-originated>
    <cache-token>{$outbound/ctx:transport/ctx:response/tp:cache-token/text()}</cache-token>
</caching-metadata>
```

Select **Replace node contents**



47. Click **Save All**

48. Click **Activate**

49. Click **Submit**

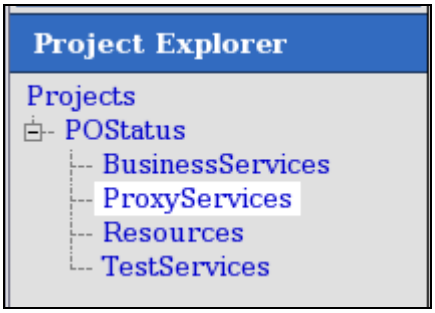
50. You are done with

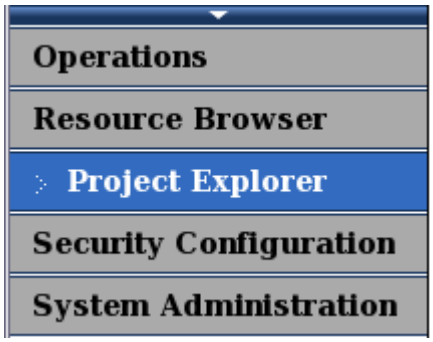
    Configuring the *GetPO* Business Service for Result Caching and

    Updating the *GetPO* Proxy Service to insert the **cache-originated** and **cache-token** response metadata (from the outbound) into the SOAP Headers (on the inbound)

2.5.4 Test

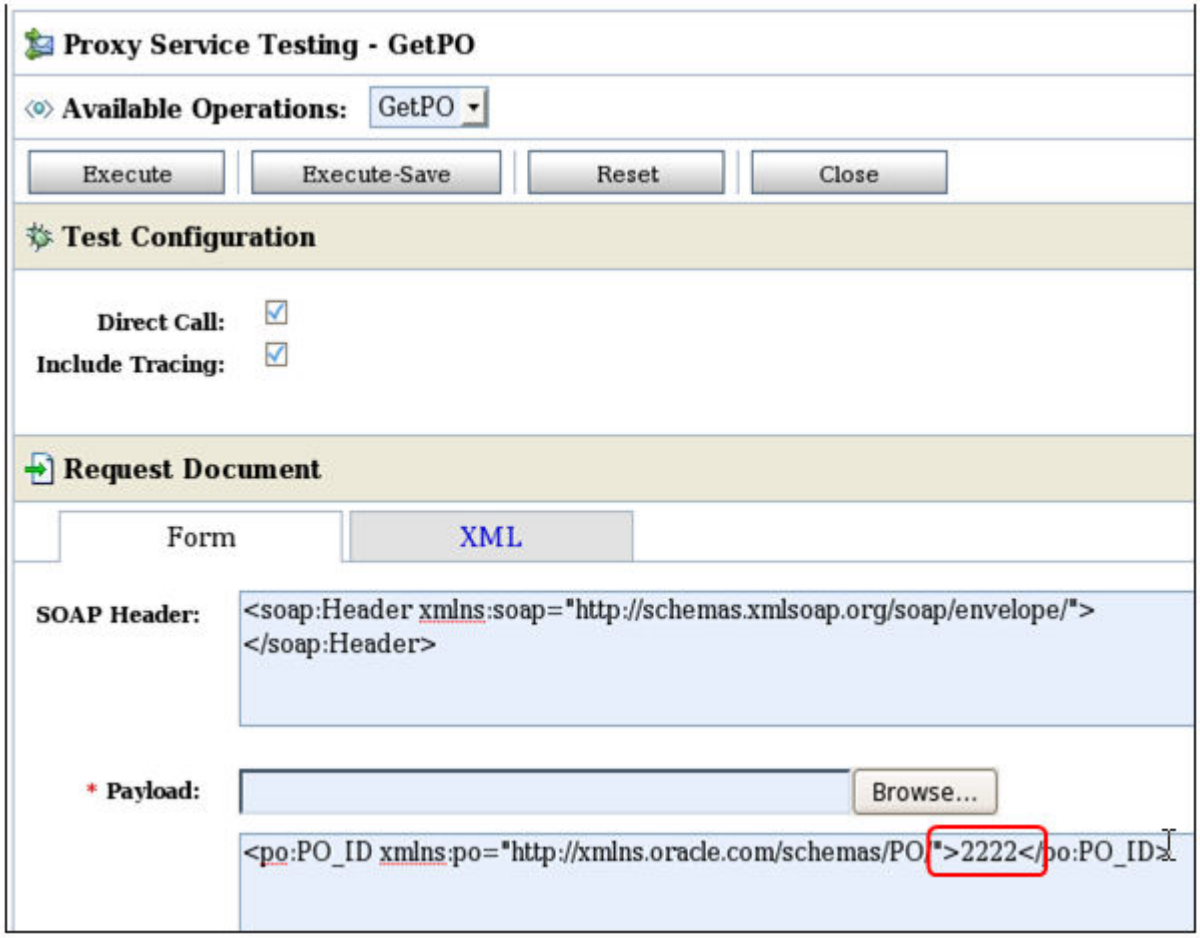
51. Select **Project Explorer** and expand POStatus/ProxyServices





52. Click  for *GetPO* Proxy Service to test

53. Specify PO_ID as 2222 and click Execute



54. You will notice that it will take about 5 seconds to return with the PurchaseOrder response. This is because the PO service, invoked by the *GetPO* Business Service, is mimicked by POStatus/TestServices/POProvider proxy service which has a Java callout that waits for 5 seconds before returning with the Purchase Order. You will see the following response which indicates the response was not retrieved from the cache (**cache-originated=false**). You will also see the cache-token that was used to add the entry to the Coherence cache (**cache-token=2222**)

Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <po:PO_ID xmlns:po="http://xmlns.oracle.com/schemas/PO/">2222</po:PO_ID>
    </soapenv:Body>
</soapenv:Envelope>
```

Response Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <caching-metadata>
            <cache-originated>false</cache-originated>
            <cache-token>2222</cache-token>
        </caching-metadata>
    </soap:Header>
    <soapenv:Body>
        <ns1:PurchaseOrder xmlns:ns1="http://xmlns.oracle.com/schemas/PO/">
            <ns1:CustID>1111</ns1:CustID>
            <ns1:ID>2222</ns1:ID>
            <ns1:productName>Bluetooth Headset</ns1:productName>
            <ns1:itemType>Electronics</ns1:itemType>
            <ns1:price>49.99</ns1:price>
            <ns1:quantity>1</ns1:quantity>
            <ns1:status>APPROVED</ns1:status>
            <ns1:ccType>Mastercard</ns1:ccType>
            <ns1:ccNumber>8765-8765-8765-8765</ns1:ccNumber>
        </ns1:PurchaseOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

55. Immediately **within 1 minute of the first request** make another request with the same, **2222**, PO_ID. Click **Back** and Click **Execute** to perform this step

Proxy Service Testing - GetPO

Back | Close

Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.
    <soap:Header xmlns:soap="http://schemas.xmls
    </soap:Header>
    <soapenv:Body>
        <po:PO_ID xmlns:po="http://xmlns.oracle.c
    </soapenv:Body>
</soapenv:Envelope>
```

56. You will now see the following response. This response has **cache-originated=true**, which implies the Purchase Order came from the Coherence Cache. The header also has **cache-token=2222**, which is the PO_ID.

Request Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    </soap:Header>
    <soapenv:Body>
        <po:PO_ID xmlns:po="http://xmlns.oracle.com/schemas/PO/">2222</po:PO_ID>
    </soapenv:Body>
</soapenv:Envelope>
```

Response Document

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <caching-metadata>
            <cache-originated>true</cache-originated>
            <cache-token>2222</cache-token>
        </caching-metadata>
    </soap:Header>
    <soapenv:Body>
        <ns1:PurchaseOrder xmlns:ns1="http://xmlns.oracle.com/schemas/PO/">
            <ns1:CustID>1111</ns1:CustID>
            <ns1:ID>2222</ns1:ID>
            <ns1:productName>Bluetooth Headset</ns1:productName>
            <ns1:itemType>Electronics</ns1:itemType>
            <ns1:price>49.99</ns1:price>
            <ns1:quantity>1</ns1:quantity>
            <ns1:status>APPROVED</ns1:status>
            <ns1:ccType>Mastercard</ns1:ccType>
            <ns1:ccNumber>8765-8765-8765-8765</ns1:ccNumber>
        </ns1:PurchaseOrder>
    </soapenv:Body>
</soapenv:Envelope>
```

57. If you repeat the test again after the cache entry expires (**more than 1 minute after the first request**) you will see the same response as in Step 54. The response comes from the backend (instead of the cache) and the response is cached using the specified cache-token.

Success – You are done with Chapter 2 !