

## SOA Suite 12c: First steps with the Coherence Adapter to create cross instance state memory

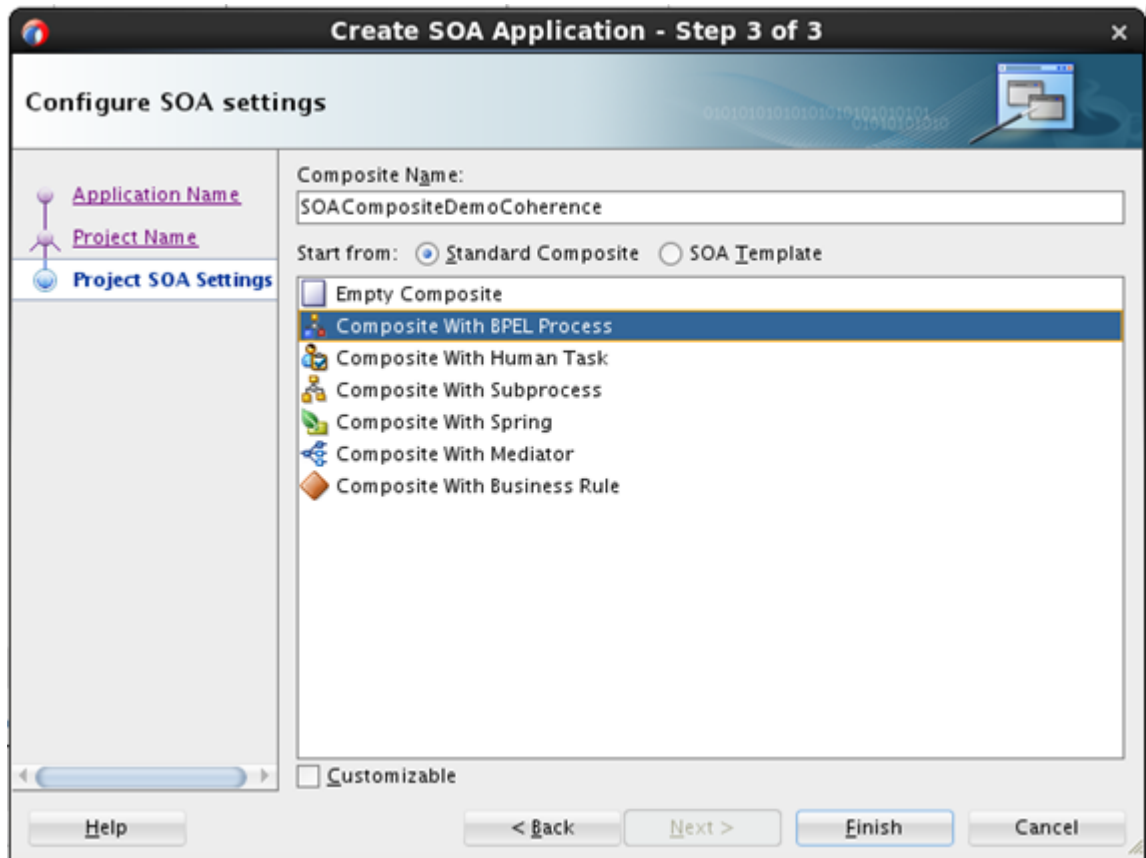
One of the new adapters shipped with SOA Suite 12c is the Coherence Adapter. This JCA adapter makes it easy for a Service Bus business service or a SOA composite application to interact with a Coherence memory grid. Fully declaratively and with very little trouble, data can be put on a Coherence grid (aka cache) and read from that cache. The cache is accessed like a big map: using a key, an object is saved to and retrieved from the cache. The cache is accessible across service executions and process instances, as well as across cluster nodes.

A Coherence Cache is typically used to reuse results: values that have been calculated before at potentially some [considerable] cost to the back end system. By reusing them, the results are obtained much faster and without any load on the enterprise resources that would otherwise have to reproduce them. Using values from the cache may actually mean saving money in the case that calculating the result involves invoking paid for services.

Coherence provides replicated and distributed (partitioned) data management and caching services on top of a reliable, highly scalable peer-to-peer clustering protocol. The Coherence Adapter is a JCA 1.5-compliant resource adapter for Oracle Coherence. Objects in the cache can either be of XML or POJO (Plain Old Java Object) type. The Coherence adapter enables you to perform useful coherence operations such as adding an item to a Coherence cache, obtaining an item from a Coherence cache, removing an item and querying from a Coherence cache.

I will show a simple example of creating a SOA composite application that uses the Coherence Adapter to speed up a process more than 100 times (although admittedly this could only happen because I first made it extremely slow).

Create new SOA Composite:



Create BPEL component with simple synchronous interface, exposed as SOAP Web Service.  
Input is string, output is string:

**Create BPEL Process**

**BPEL Process**

A BPEL process is a service orchestration, based on the BPEL specification, used to describe/execute a business process (or large grained service), which is implemented as a stateful service.

☒ BPEL 2.0 Specification ☐ BPEL 1.1 Specification

**Name:** HelloSlowWorld

**Namespace:** racle.com /SOACompositeDemoCoherence/SOACompositeDemoCoherence/HelloSlowWorld

**Directory:** j01/work/SOACompositeDemoCoherence/SOACompositeDemoCoherence/SOA/BPEL

**Template:** Synchronous BPEL Process

**Service Name:** helloslowworld\_client

☒ Expose as a SOAP service

**Transaction:** required

**Input:** :iteDemoCoherence/SOACompositeDemoCoherence/HelloSlowWorld}process

**Output:** :oherence/SOACompositeDemoCoherence/HelloSlowWorld}processResponse

Help OK Cancel

Create a second BPEL process based on the same WSDL:

### Create BPEL Process

**BPEL Process**

A BPEL process is a service orchestration, based on the BPEL specification, used to describe/execute a business process (or large grained service), which is implemented as a stateful service.

☒ BPEL 2.0 Specification ☐ BPEL 1.1 Specification

Name:

Namespace:

Directory:

Template:

Service Name:

☐ Expose as a SOAP service

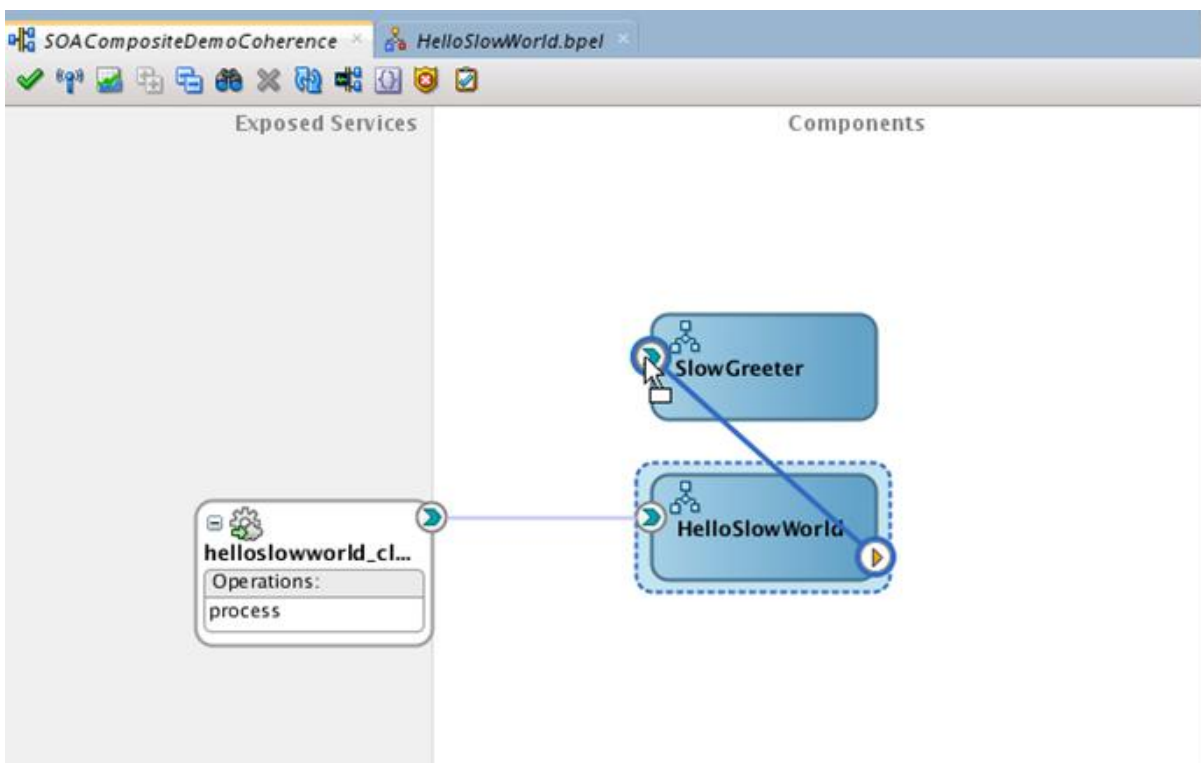
Transaction:

WSDL URL:

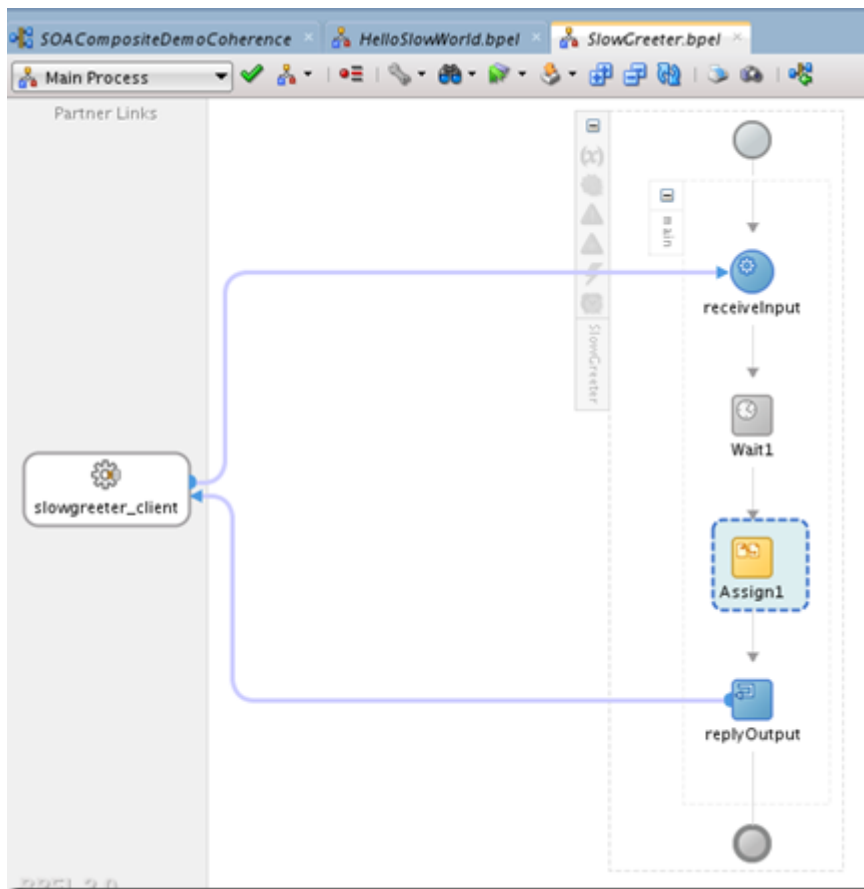
Port Type:

Callback Port Type:

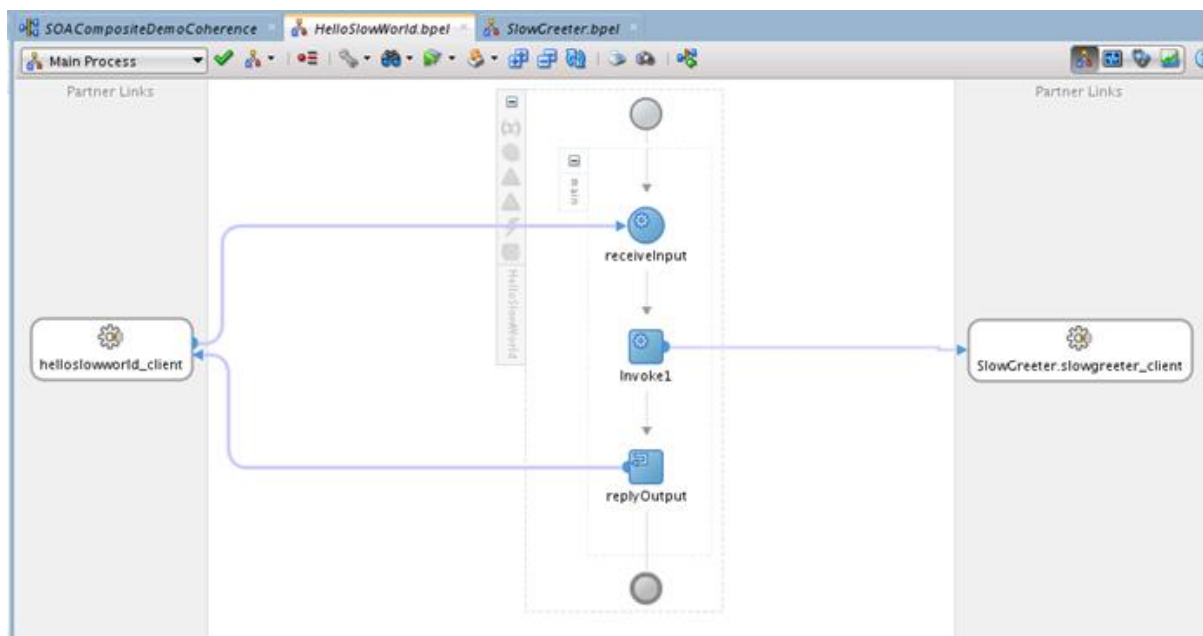
Wire the first BPEL Process to the second. This second process will create the hello world greeting – but it will do so quite slowly.



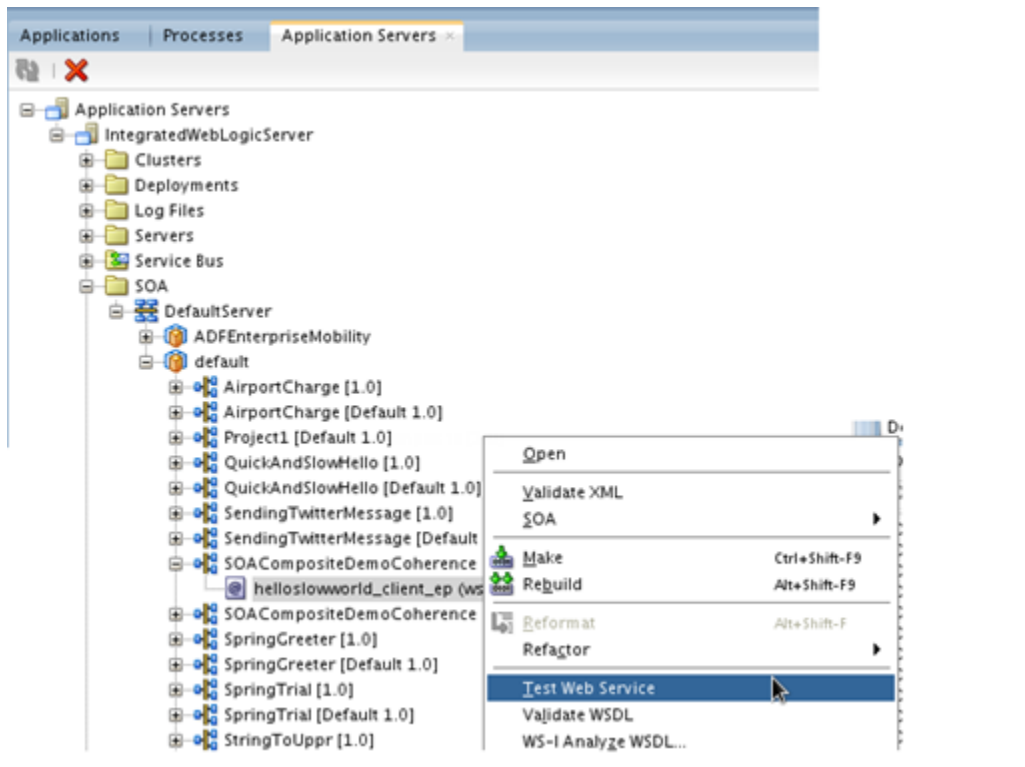
Edit the second, slow process. It waits for 5 seconds, then it assigns a greeting value to the output variable:



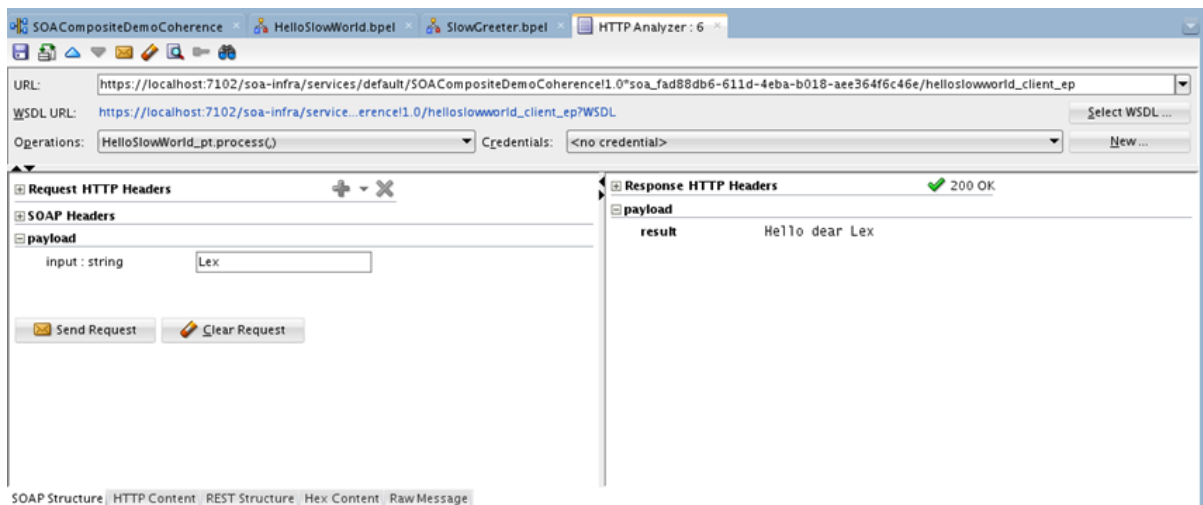
The first process will call this second process. After 5 seconds, it will receive the greeting, which it can then return itself:



Deploy and run:



And run from the automatically opened HTTP Analyzer



With the following call log (just over **5 seconds**):

| ID | Date                          | Call Time | Method | Url                            | Query | Cookie / Key | Status | Set-Cookie | Type | Size |
|----|-------------------------------|-----------|--------|--------------------------------|-------|--------------|--------|------------|------|------|
| 10 | Thu, 12 Jun 2014 05:05:12 GMT | 5183ms    | POST   | https://localhost:7102/soa-in- |       |              | 200 OK |            | SOAP | 1303 |

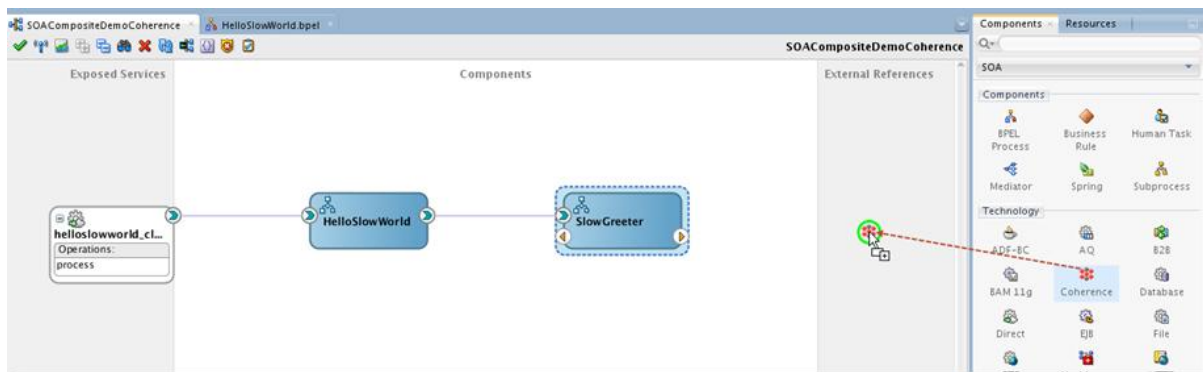
With the following flow trace:

| Trace                    |         |  |             |  |
|--------------------------|---------|--|-------------|--|
| Actions ▾                | View ▾  | Show Instance IDs <input type="checkbox"/> |             |  |
| Instance                 | Type    | Usage                                      | State       |  |
| helloslowworld_client_ep | Service | Service                                    | ✔ Completed |  |
| HelloSlowWorld           | BPEL    |  | ✔ Completed |  |
| SlowGreeter              | BPEL    |  | ✔ Completed |  |

No matter how often we make the call, the response time stays over 5 seconds, even if the response is the same each time. Enter: the cross instance memory, implemented using Coherence and accessed using the Coherence Adapter.

Coherence Adapter as shared memory

Open the composite editor. Drag the Coherence Adapter from the component palette to the References swimlane.



The Adapter Configuration wizard appears.

The screenshot shows the 'Coherence Adapter Configuration Wizard - Step 1 of 3'. The wizard is titled 'Coherence Adapter Reference' and has a welcome message: 'Welcome to the Coherence Adapter Configuration Wizard. This wizard helps you create a Coherence Adapter. You will be asked to specify configuration parameters and define an operation for the adapter.' Below this, there is a section 'Enter a Reference Name' with a text input field containing 'WriteResultToCache'. At the bottom, there are buttons for 'Back', 'Next', 'Cancel', and 'OK'.

Set the name of the adapter binding to WriteToCache. Press Next.



Set or select the JNDI name of the Coherence connection to eis/Coherence/Local. This name corresponds with one of the connections configured in the Coherence Adapter deployment in the Integrated WebLogic Server. Press Next.



Select Put as the operation type.

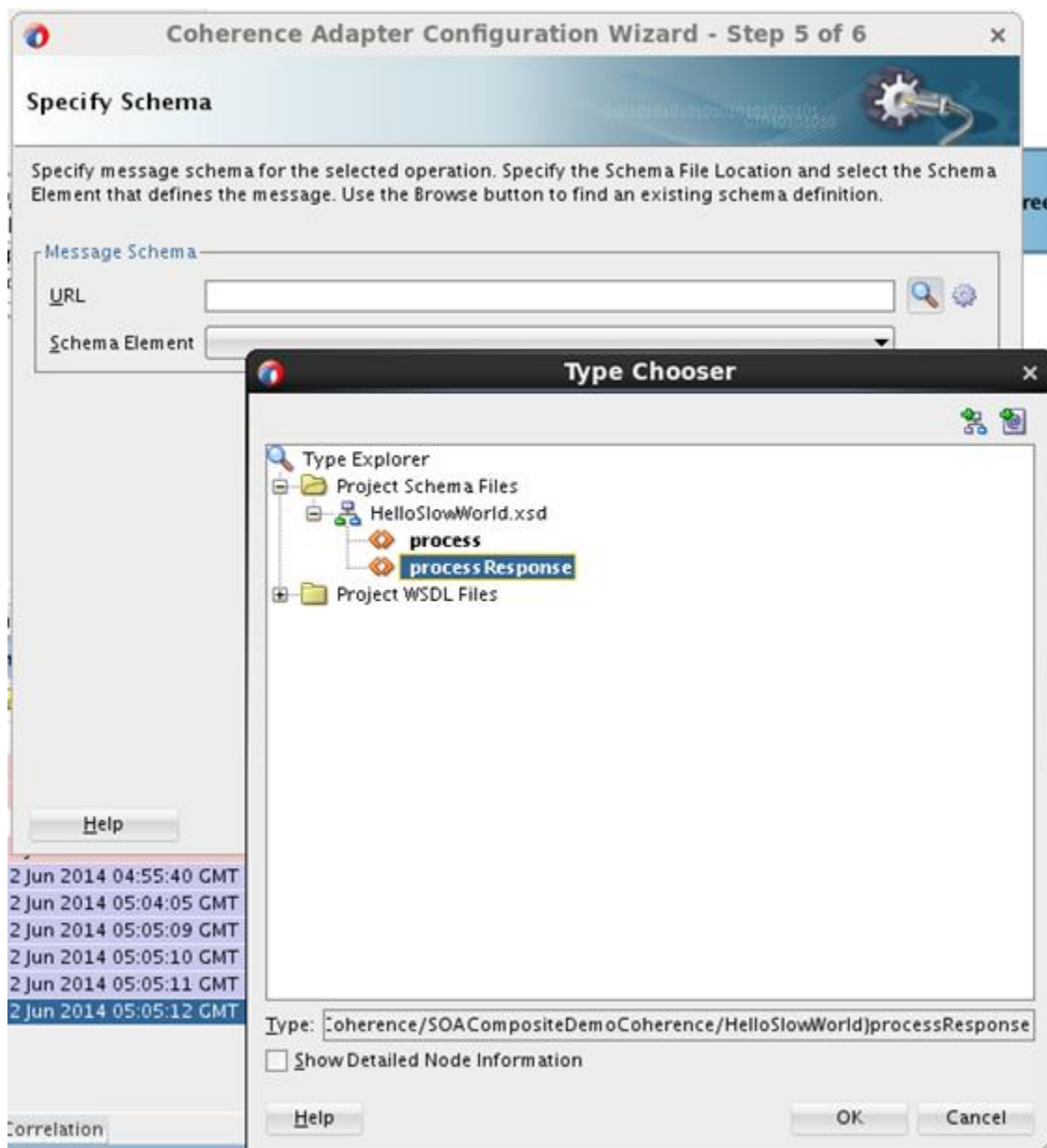
In step 4 of the wizard, set *Cache Type* to XML. Set the *Cache Name* to adapter-local. This cache is preconfigured – ready to use for straightforward situations where no special configuration is required.



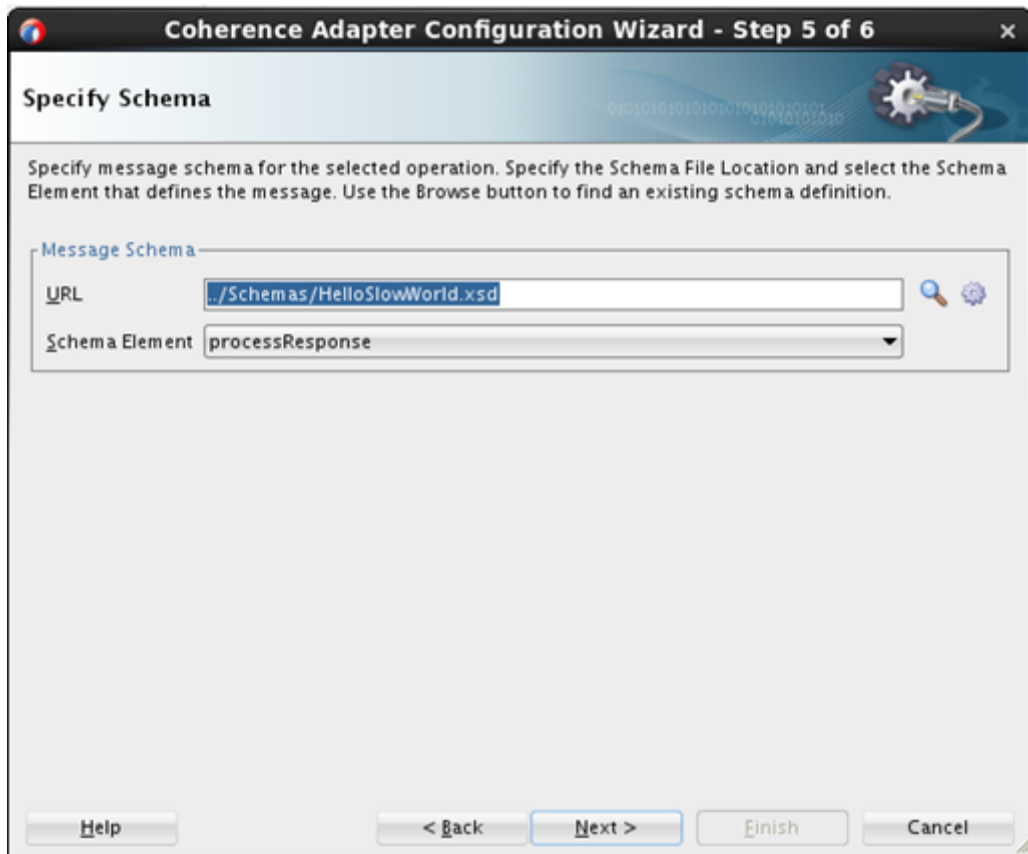


Uncheck the checkbox for auto generating the key. Set the Key Type to String and leave the Key field empty. Accept the default for *time to live*. Press Next.

A popup appears with a message, instructing us on setting the value for the key at run time using the JCA header property `jca.coherence.Key`. Acknowledge the message and continue.



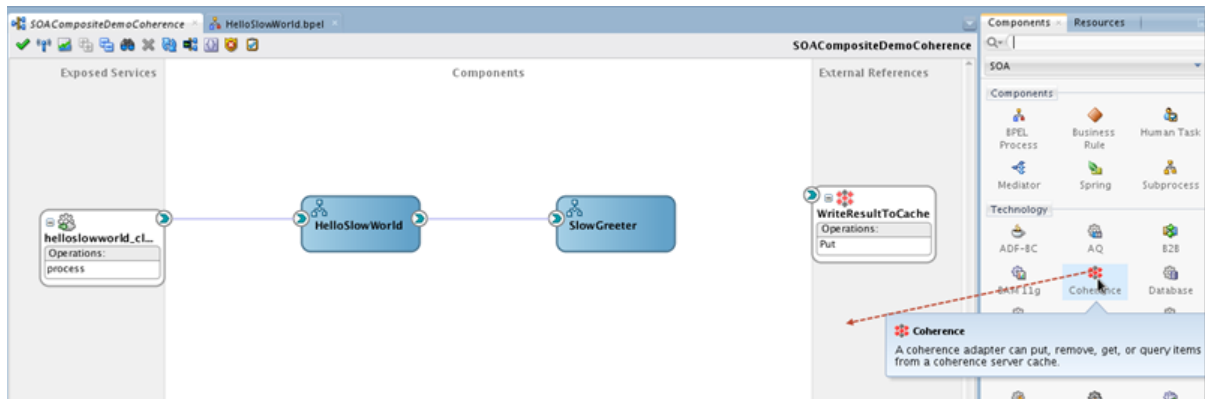
Select element processResponse from the HelloSlowWorld.xsd as the schema element [for objects to retrieve from the cache].



Click Next and click Finish.



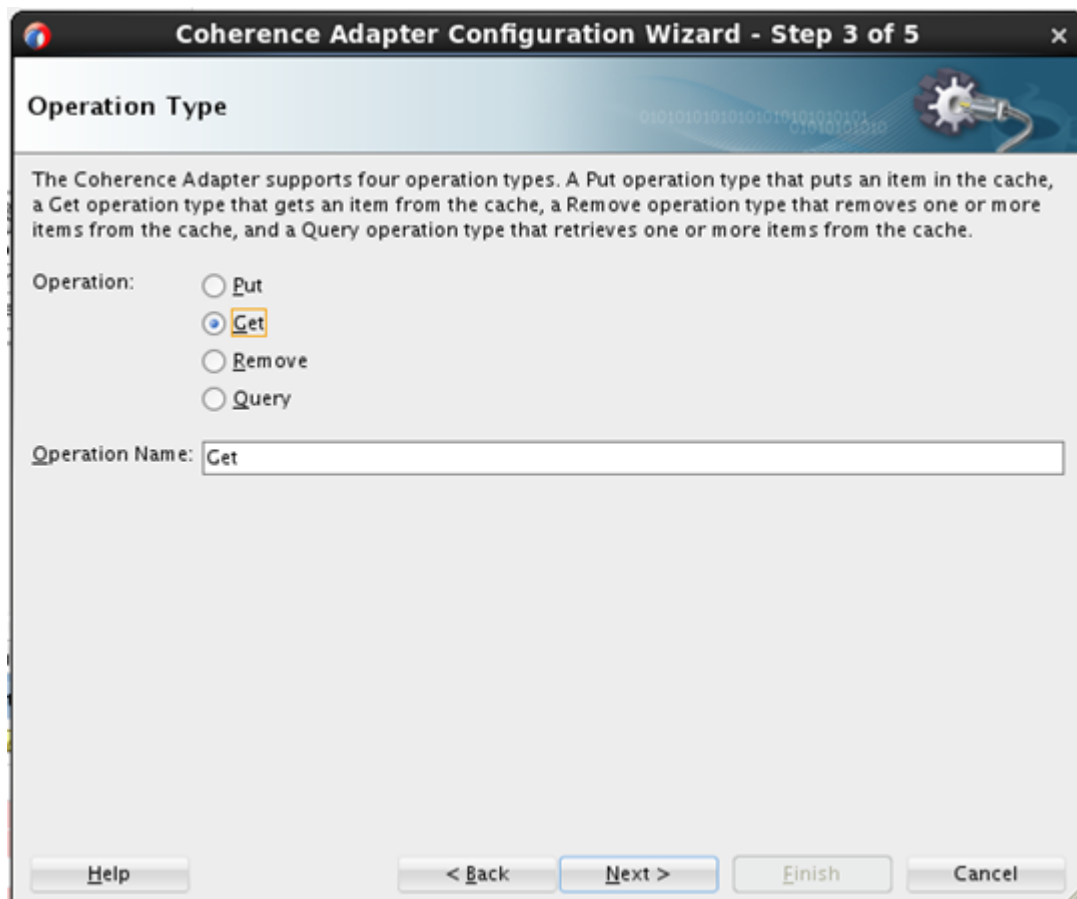
The Coherence Adapter reference (for reading stuff from the cache) is created.



Drag a second Coherence Adapter to the References lane, to create the Adapter binding to write data to the cache.

The screenshot shows the 'Coherence Adapter Configuration Wizard - Step 1 of 3'. The title bar indicates the step. The main window has a header 'Coherence Adapter Reference' and a welcome message: 'Welcome to the Coherence Adapter Configuration Wizard. This wizard helps you create a Coherence Adapter. You will be asked to specify configuration parameters and define an operation for the adapter.' Below this, there is a prompt 'Enter a Reference Name.' and a text input field labeled 'Name:' containing the text 'RetreiveResultFromCache'. At the bottom, there are four buttons: 'Help', '< Back', 'Next >', and 'Finish'. The 'Next >' button is highlighted.

Press Next.



Select Put as the operation type.

In step 4 of the wizard, set *Cache Type* to XML. Set the *Cache Name* to adapter-local. This cache is preconfigured – ready to use for straightforward situations where no special configuration is required.

Uncheck the checkbox for auto generating the key. Set the Key Type to String and leave the Key field empty. Accept the default for *time to live*. Press Next.

**Coherence Adapter Configuration Wizard - Step 4 of 5**

### Configure Get Operation

Enter configuration parameters to retrieve an item from the Coherence cache.

Cache Type: XML

Cache Name: adapter-local


Key Type: string

Key:


Help < Back Next > Finish Cancel

A popup appears with a message, instructing us on setting the value for the key at run time using the JCA header property `jca.coherence.Key`. Acknowledge the message and continue.

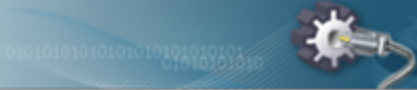




Coherence Adapter Configuration Wizard - Step 5 of 6





Specify Schema




Specify message schema for the selected operation. Specify the Schema File Location and select the Schema Element that defines the message. Use the Browse button to find an existing schema definition.

Message Schema

URL



Schema Element



Help

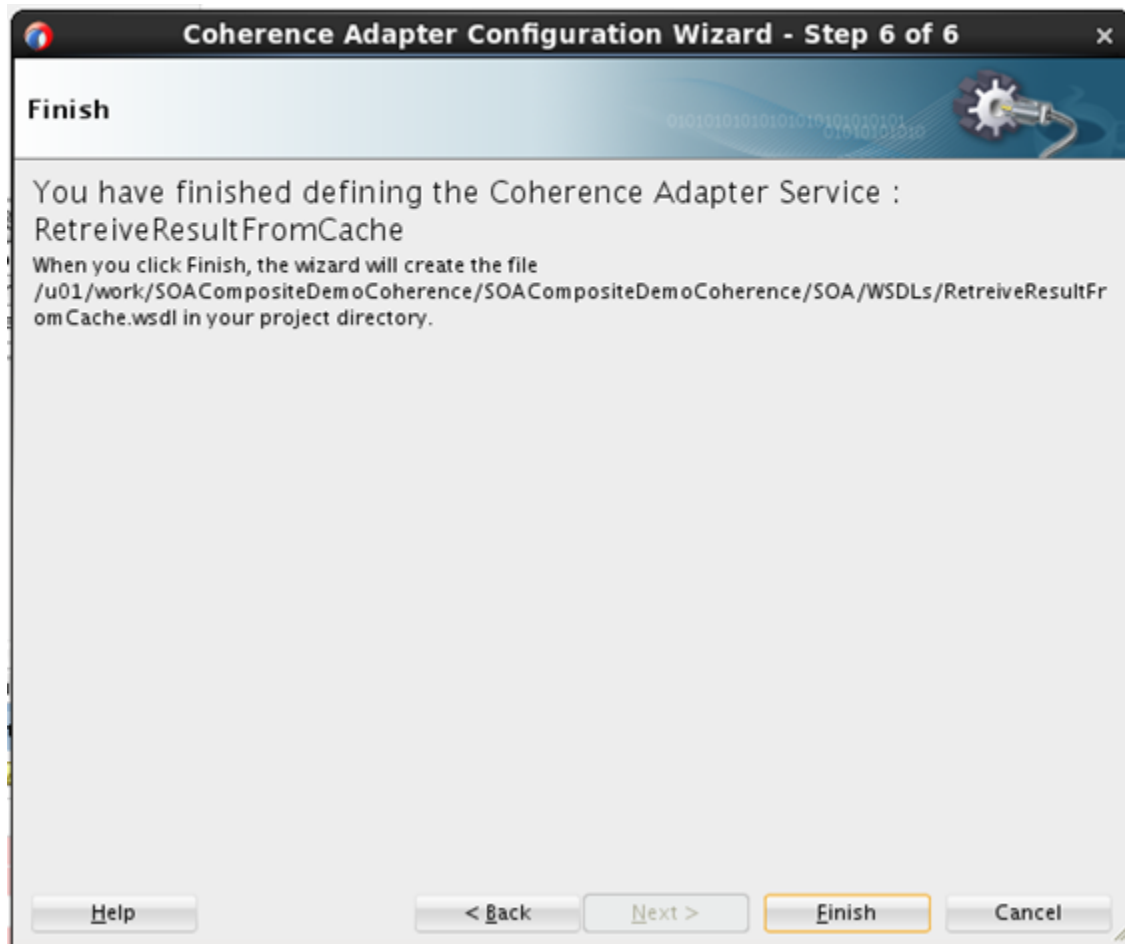
< Back

Next >

Finish

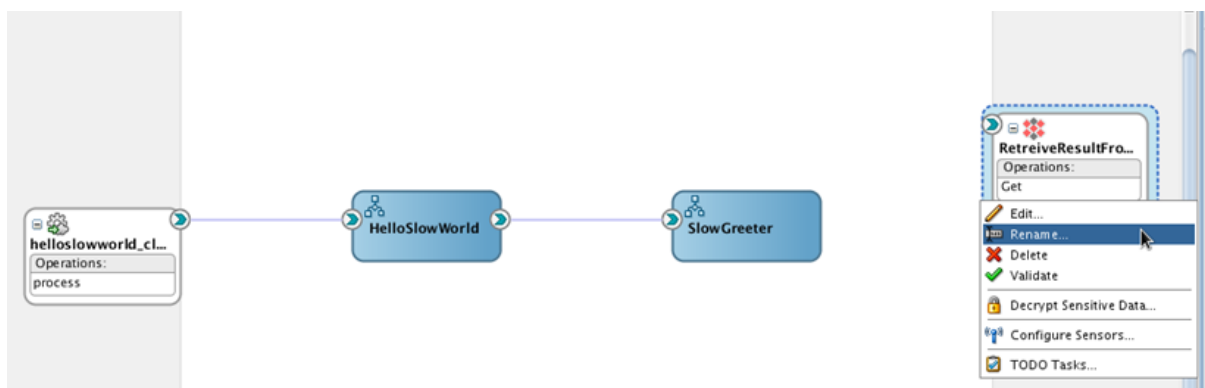
Cancel



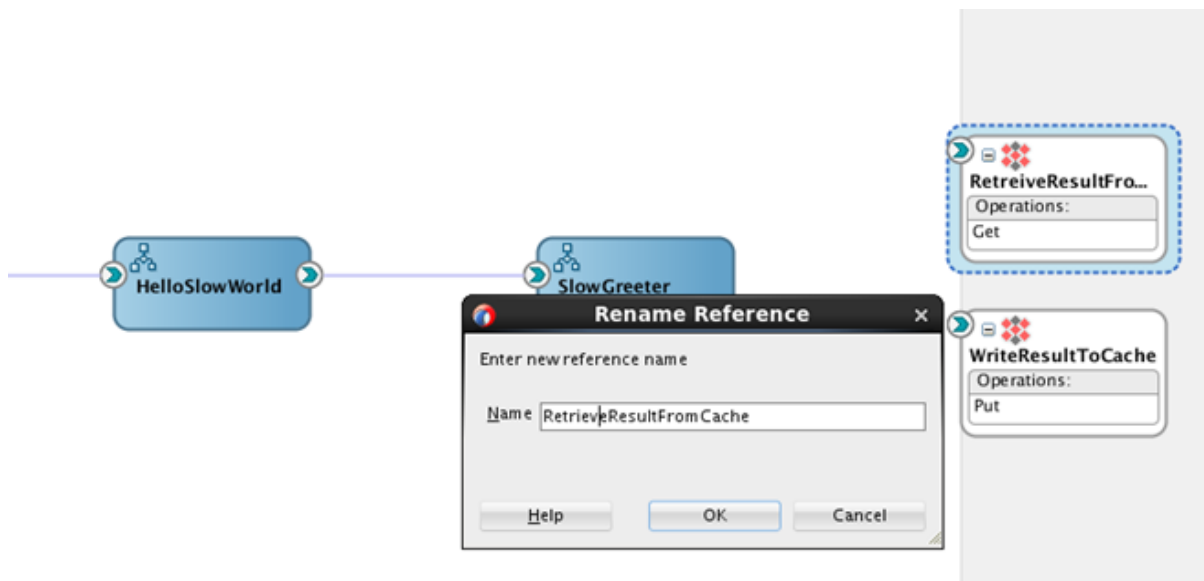


### Refactoring: Rename Adapter Binding

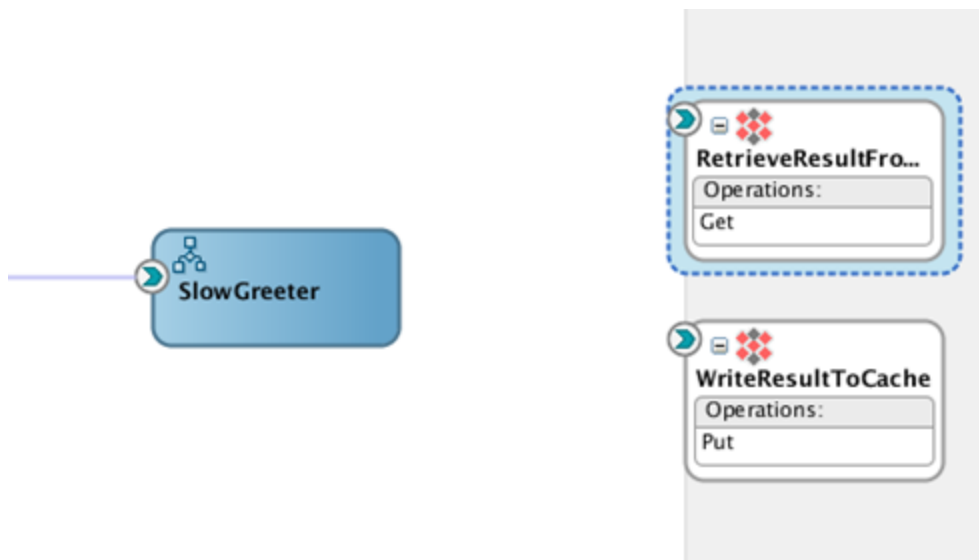
I have just realized that I have mistyped the name of the second adapter binding: Retreive instead of Retrieve. In 11g that would have given me a headache (or I would probably have recreated the adapter binding, afraid to not be able to correctly manually edit all occurrences of the string Retreive). In 12c, we can easily refactor many things, including the names of Reference Bindings. Simply right click the adapter binding:



select Rename from the dropdown menu and specify the new name:

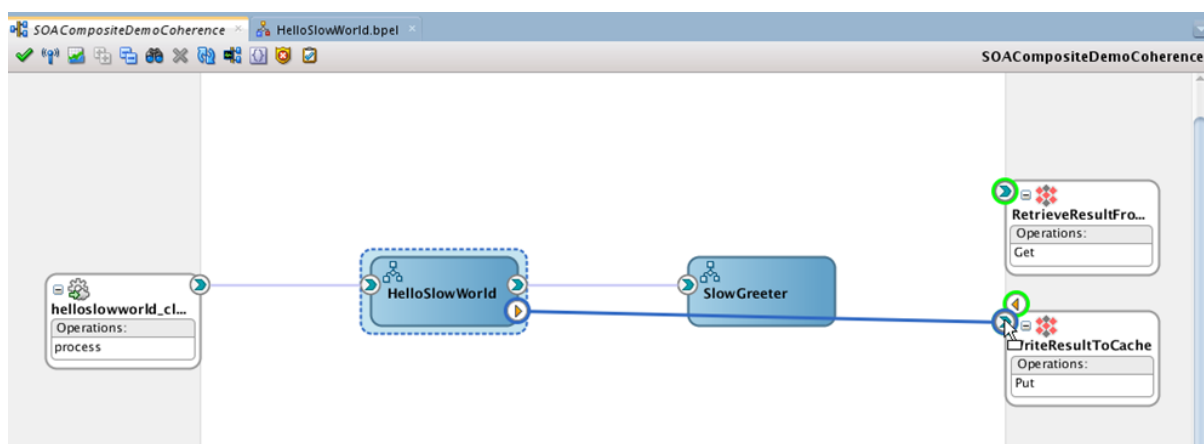


Press OK, and you're done:

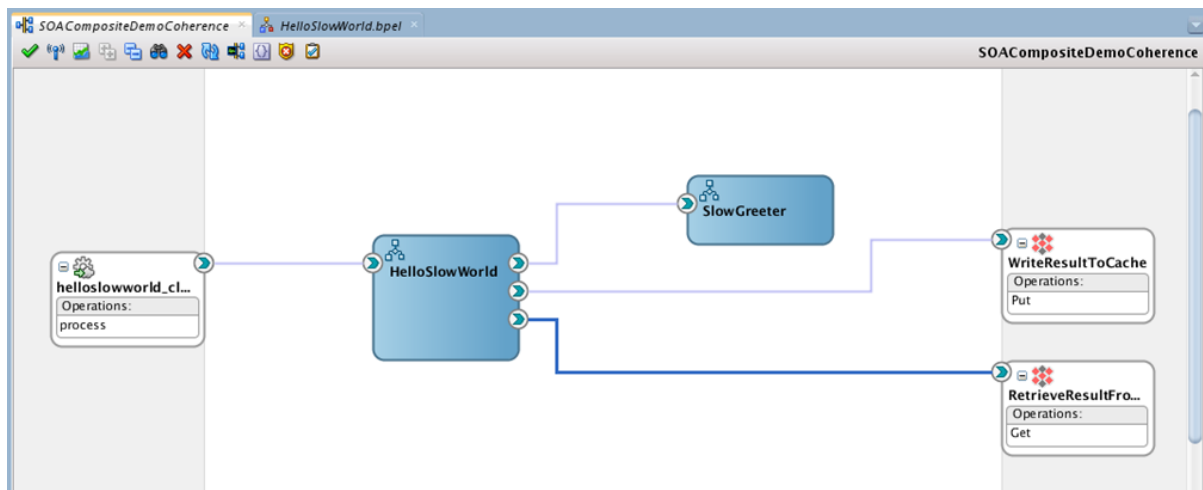


Leveraging the Coherence cache from the BPEL process

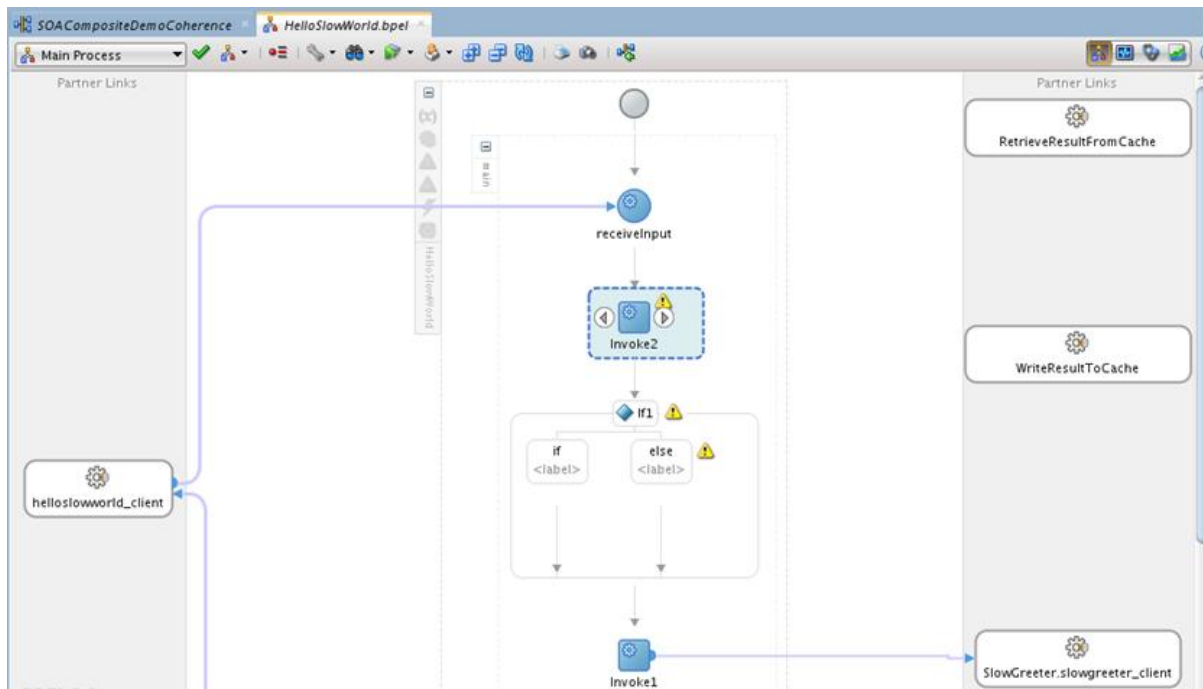
Wire the first BPEL process – HelloSlowWorld – to the WriteResultToCache adapter binding.



and wire the BPEL process to the other adapter binding:



Open BPEL Process HelloSlowWorld. Add Invoke activities: one immediately after the Receive (to invoke the RetrieveFromCache partner binding) and one after the invoke to the SlowGreeter



Wire the first invoke to the RetrieveResultFromCache partner link. Configure the activity – create local input and output variables:

**Edit Invoke**

Headers Documentation Skip Condition Targets Sources

**General** Correlations Properties Assertions Annotations

Name: RetrieveResultFromCache

Conversation ID:

Detail Label:

☐ Invoke as Detail

Interaction Type: Partner Link

Partner Link: RetrieveResultFromCache

Port Type: Get\_ptt

Operation: Get

Input Output

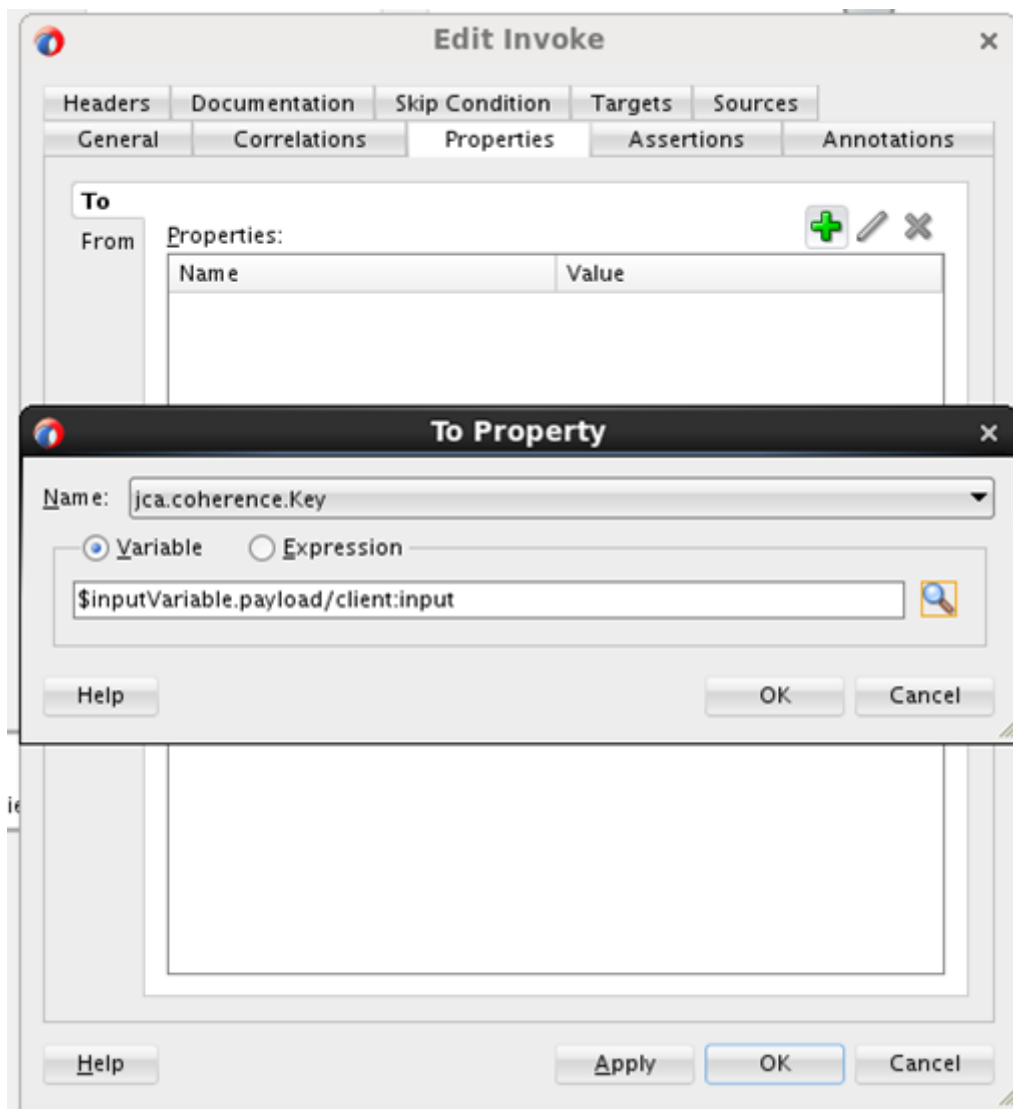
☐ Arguments Mapping ☒ Input Variable

Input: RetrieveResultFromCache\_Get\_InputVariable

Help Apply OK Cancel

Note that we will not actually use the input variable – because the key value we need to pass to the partner link and indirectly the Coherence Adapter binding is passed using a header property.

Open the Properties tab in the Invoke editor. Click on the green plus icon to add a property to pass. Select property `jca.coherence.Key` from the dropdown list. Select the input element in the payload part of the `$inputVariable` as the value for the property; the name passed into the service is used as the cache key:



Click OK (twice) to complete the configuration of the Invoke activity.


Wire the second Invoke activity to the SaveResultToCache partner link. Configure this activity – also creating two local variables.

**Edit Invoke**

Headers Documentation Skip Condition Targets Sources


General Correlations Properties Assertions Annotations


Name: SaveResultToCache

Conversation ID: 

Detail Label:

☐ Invoke as Detail


Interaction Type:  Partner Link



Partner Link: WriteResultToCache 

Port Type: Put\_ptt

Operation: Put

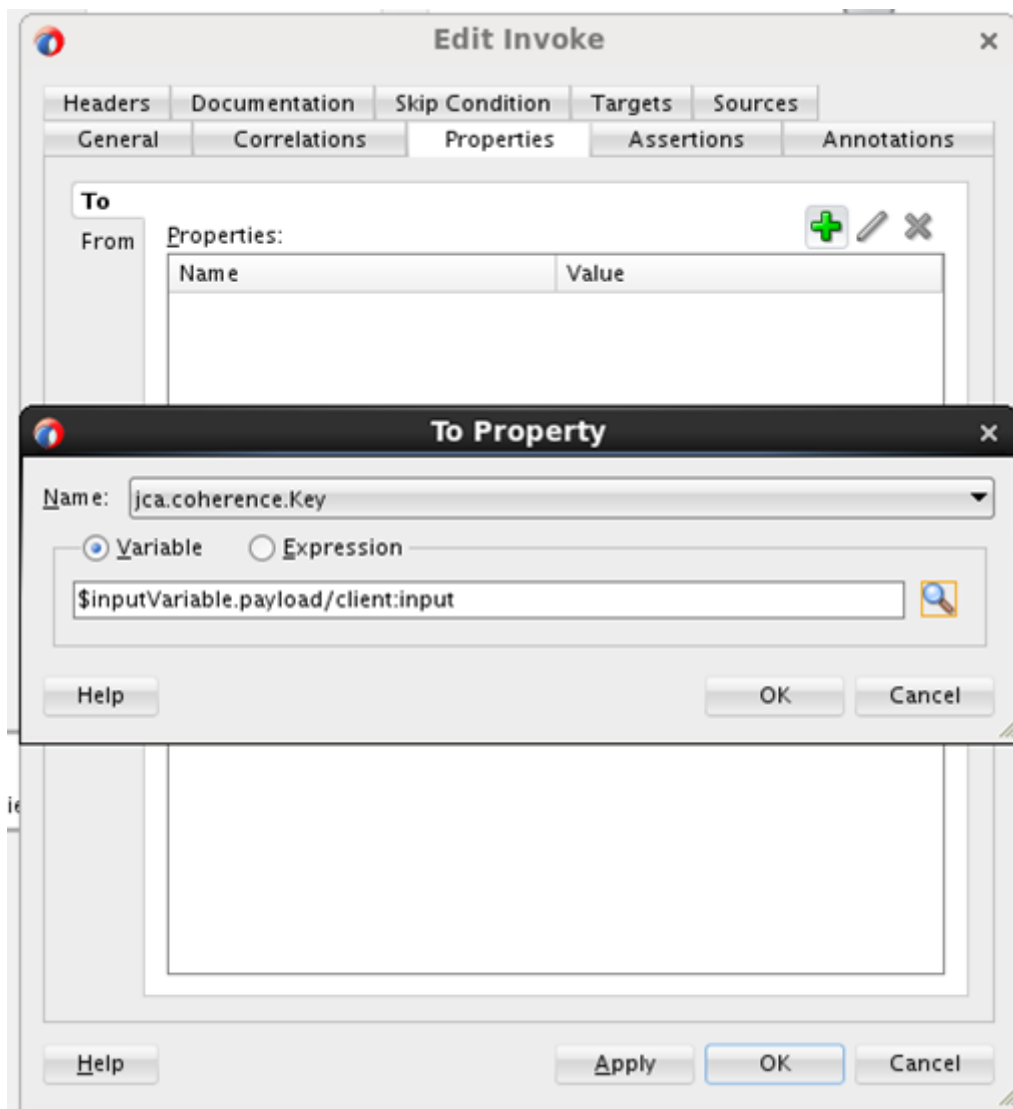
Input Output

☐ Arguments Mapping ☒ Input Variable 

Input: SaveResultToCache\_Put\_InputVariable  

Help Apply OK Cancel

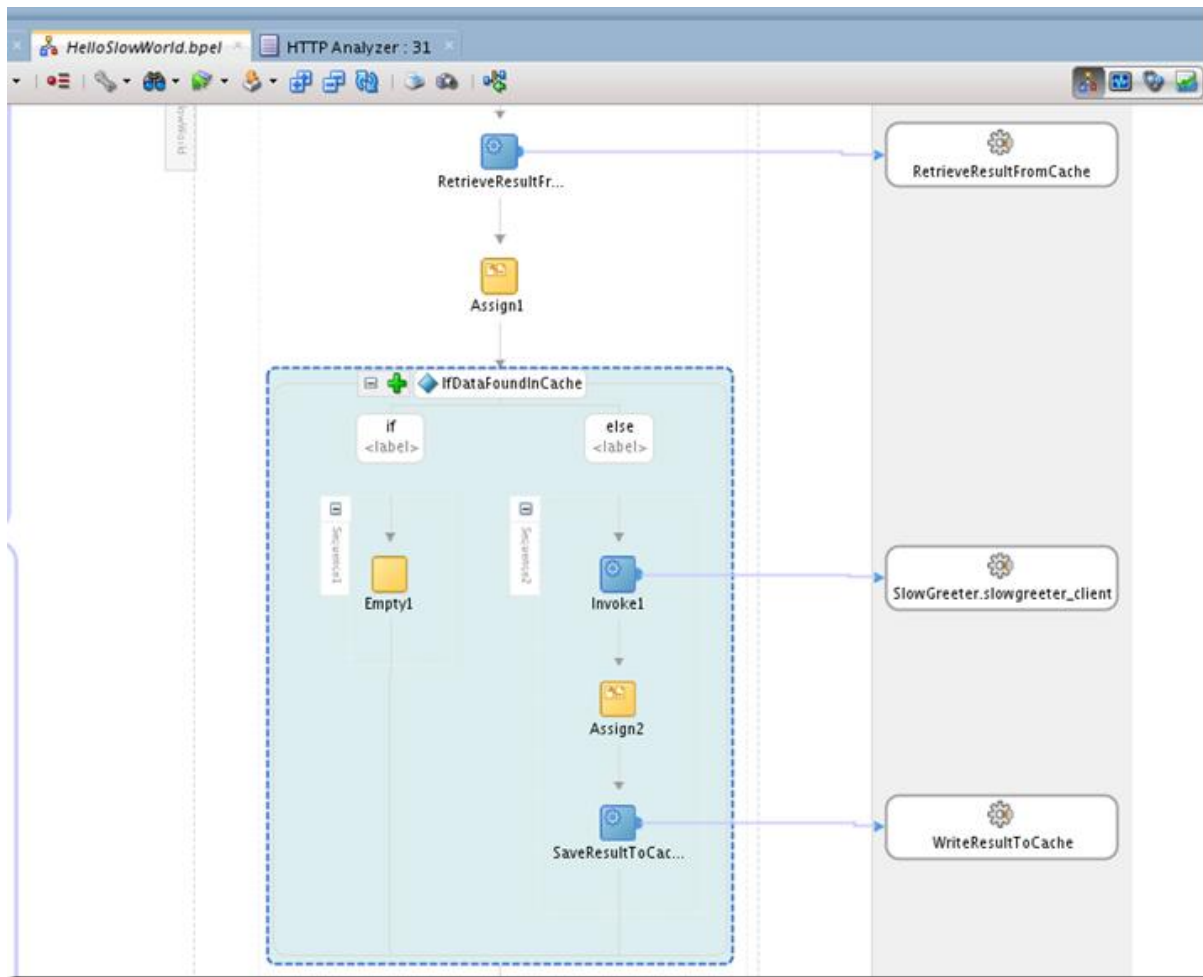
Open the Properties tab in the Invoke editor. Click on the green plus icon to add a property to pass. Select property `jca.coherence.Key` from the dropdown list. Specify the following expression to derive the value for the Key property – the same as for the previous invoke activity:



Click on OK (twice) to complete the definition of the invoke activity.

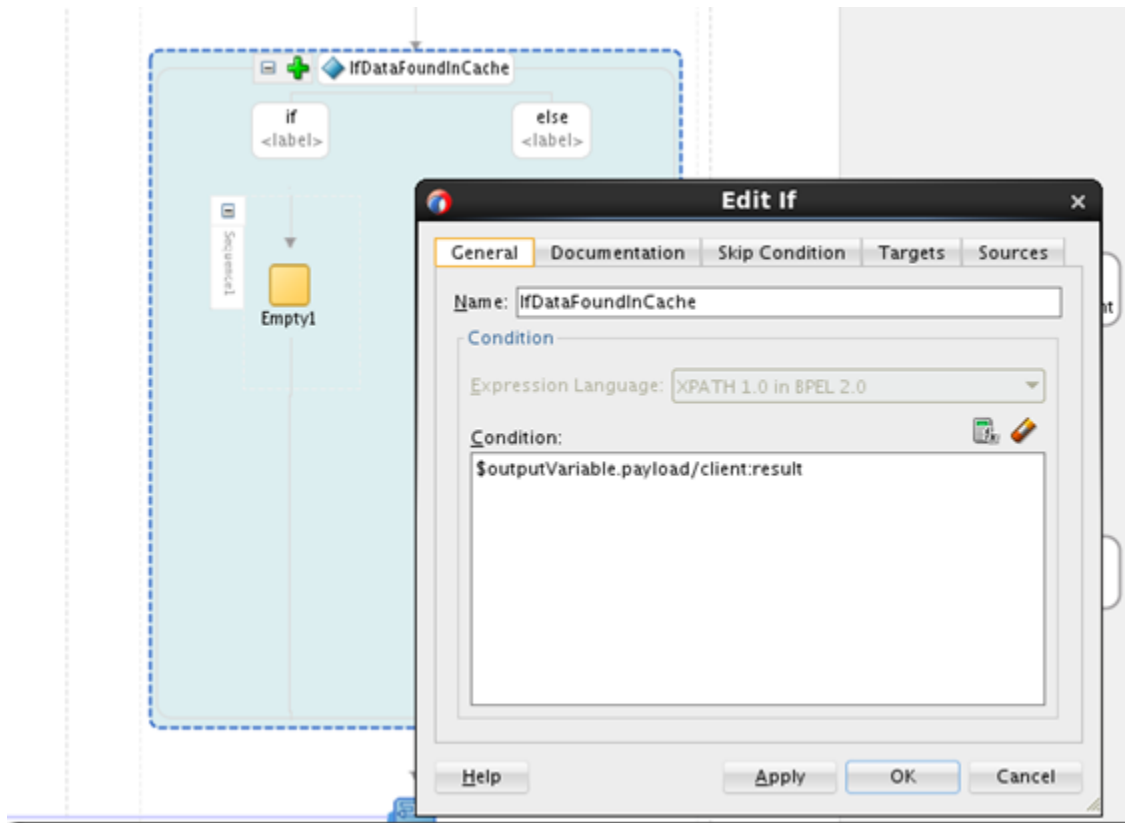
Now we add some logic to the BPEL process.

Add an IF activity. It will test whether the cache contained the object under the given key. If it does, then we are done: the output variable has been set. If it does not, we have to execute the invoke to the SlowGreeter followed by the invoke to the WriteResultToCache partnerlink.

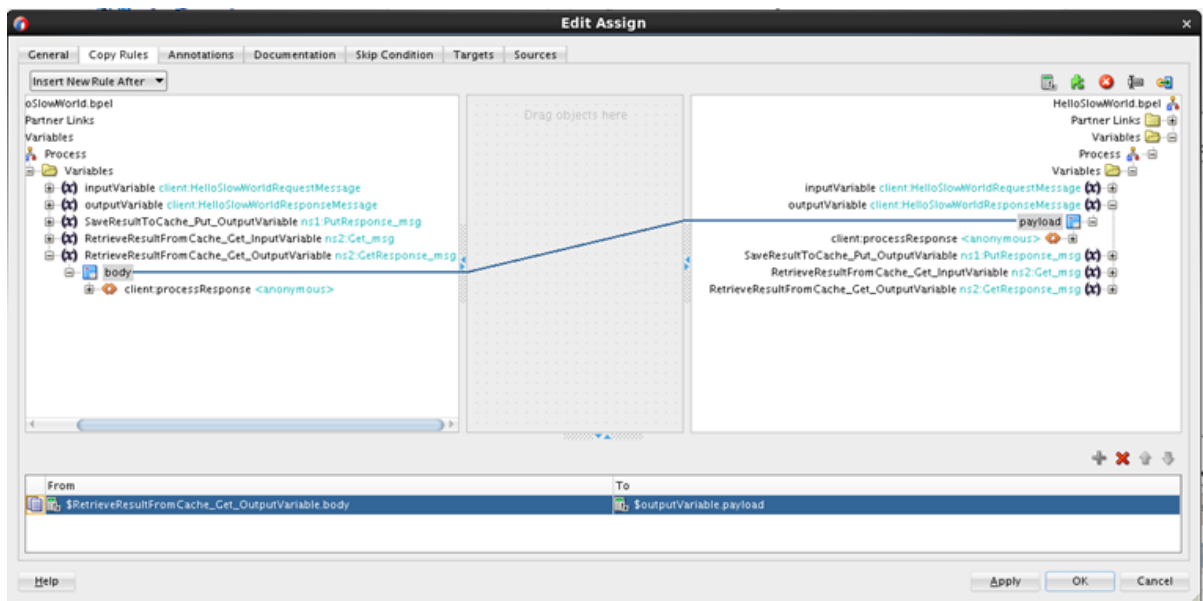


The IF condition is defined as follows:

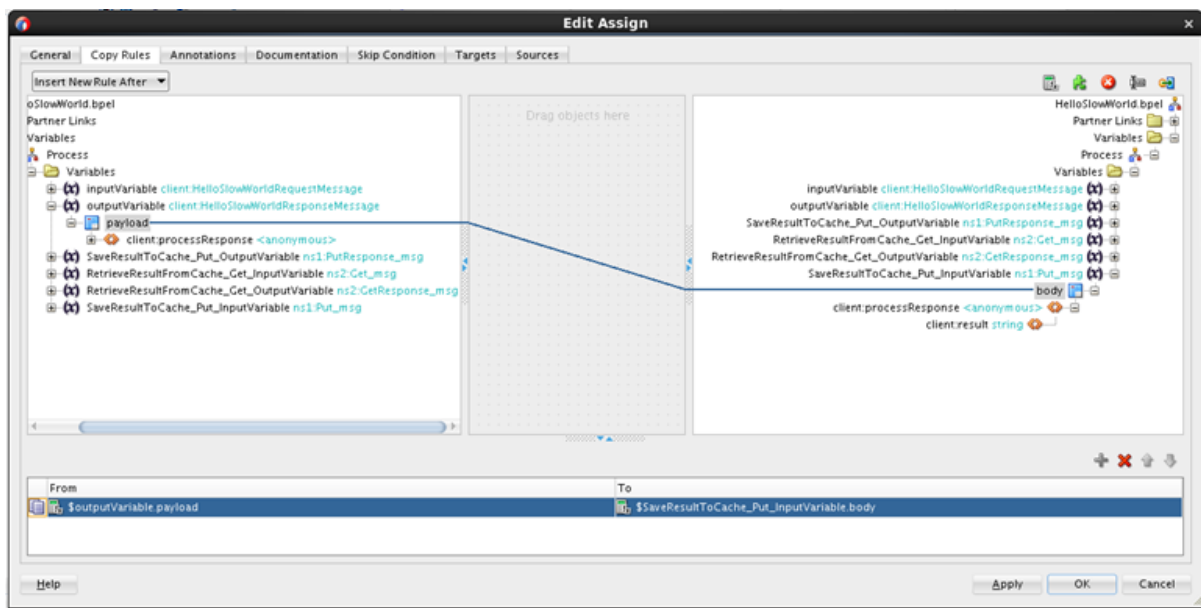




We also need to add assign variables, to assign the result from retrieve from cache to the outputVariable

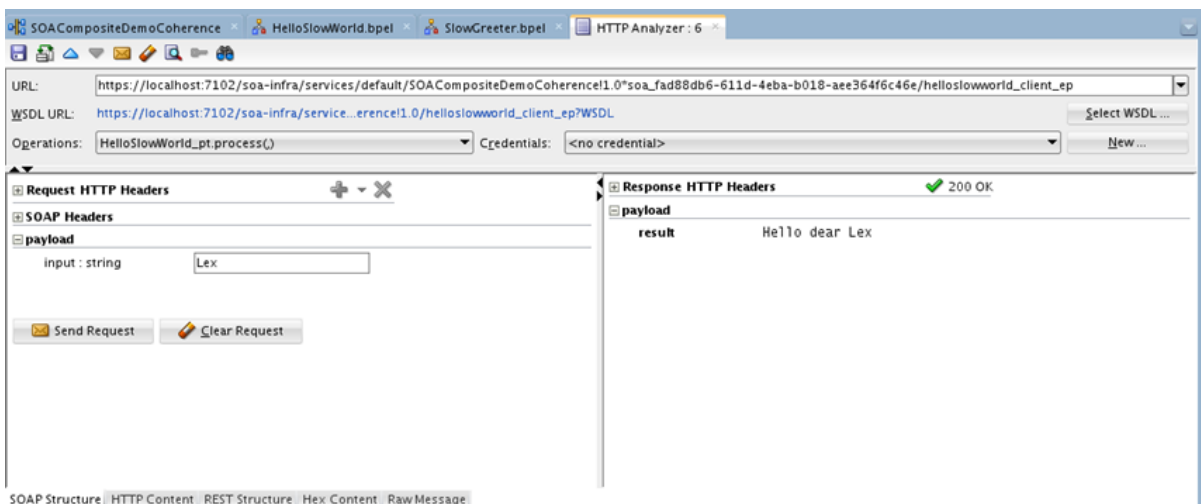


and from the invoke to SlowGreeter to the input variable used in the invoke to WriteResultToCache:



Deploy the SOA composite and Test it

After deploying the SOA composite to the Integrated WLS, we can again make a test call from the HTTP Analyzer tool.

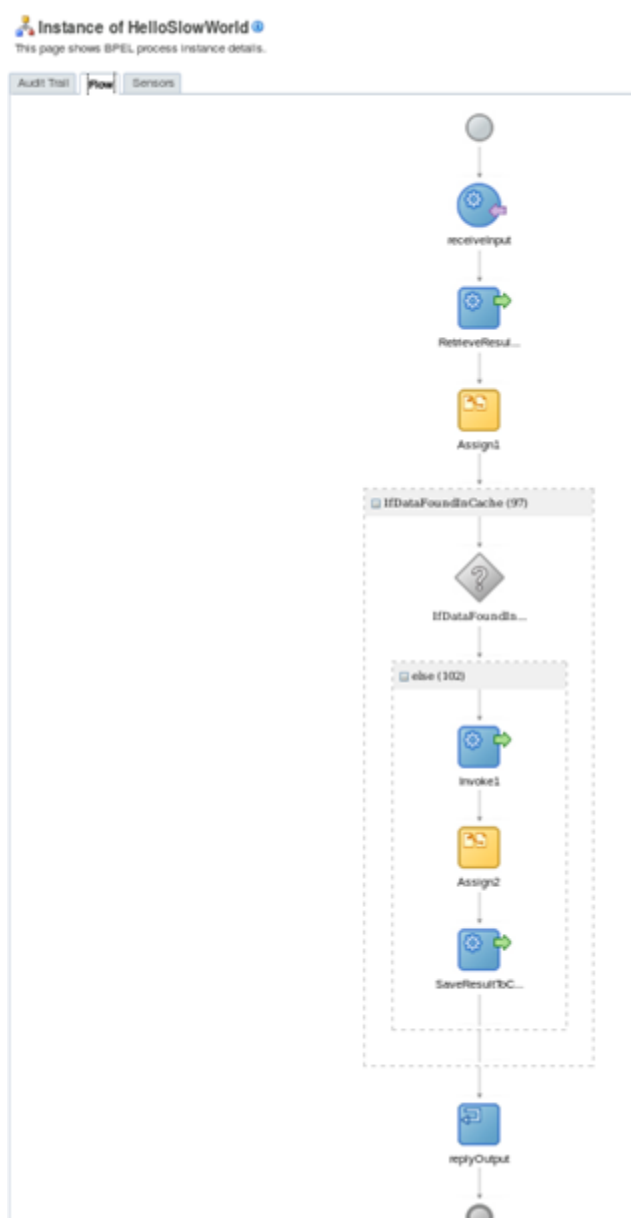


It looks the same, the result is the same and the time it took to produce the result is the same as well. However, when we check the flow trace, we can see that this time a little bit more went on behind the scenes:

| Trace                   |           |  |             |  |
|-------------------------|-----------|--|-------------|--|
| Actions ▾               | View ▾    | Show Instance IDs <input type="checkbox"/> |             |  |
| Instance                | Type      | Usage                                      | State       |  |
| heloslowworld_client_ep | Service   | Service                                    | ✓ Completed |  |
| HelloSlowWorld          | BPEL      |  | ✓ Completed |  |
| RetrieveResultFromCache | Reference | Reference                                  | ✓ Completed |  |
| SlowGreeter             | BPEL      |  | ✓ Completed |  |
| WriteResultToCache      | Reference | Reference                                  | ✓ Completed |  |

The composite did not just use the SlowGreeter to create the response. It also first checked the cache – in vain as it turned out – and then after obtaining the result from the SlowGreeter, it also put the result onto the cache. That last action does not benefit this instance of the composite of course, but it will help speed up future instances that have to deal with the same input and the same SlowGreeter.

The BPEL Audit Flow reveals exactly what happened – no surprises there: invoke the get-from-cache. If not found, then invoke SlowGreeter and subsequently pass result to put-on-cache.



The next call to the service for the same input has a much faster response and a different flow trace:

| Message                  | Type      | Source    | State       |
|--------------------------|-----------|-----------|-------------|
| helloslowworld_client_ep | Service   | Service   | ✓ Completed |
| HelloSlowWorld           | BPEL      |           | ✓ Completed |
| RetrieveResultFromCache  | Reference | Reference | ✓ Completed |

This time, the result could be found in the cache, so no additional processing is required and SlowGreeter does not have to be invoked.

