**Al Majdouie Middleware Project**

**Solution Architecture Patterns**

Prepared by:

SBM

Khobar, KSA

## Document Control

|  |
| --- |
|  |

**Change Record**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change Reference** |
| 12/4/2015 | Marwan Zakhia | 0.1 | Initial Draft Document |

|  |
| --- |
|  |

**Reviewers**

|  |  |
| --- | --- |
| **Name** | **Position** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
|  |

Library

|  |  |  |
| --- | --- | --- |
| **Copy** | **Name** | **Location** |
| 1 | Library Master | Project Library |
| 2 |  |  |
| 3 |  |  |

**Note to Holders:**

If you receive an electronic copy of this document and print it out, please write your name on the equivalent of the cover page, for document control purposes.

If you receive a hard copy of this document, please write your name on the front cover, for document control purposes.

## Table of Contents

[1. Document Control 2](#_Toc418495369)

[2. Table of Contents 3](#_Toc418495370)

[3. Table of Figures 5](#_Toc418495371)

[4. Introduction 6](#_Toc418495372)

[4.1 Purpose 6](#_Toc418495373)

[4.2 Intended Audience 6](#_Toc418495374)

[4.3 Document Summary 6](#_Toc418495375)

[5. Non-SOA Middleware Integration Patterns 7](#_Toc418495376)

[5.1 Point to Point Integration Pattern 7](#_Toc418495377)

[5.1.1 Overview 7](#_Toc418495378)

[5.1.2 Pattern Breakdown 7](#_Toc418495379)

[6. SOA Middleware Integration Patterns 8](#_Toc418495380)

[6.1 Synchronous Query Integration Pattern 8](#_Toc418495381)

[6.1.1 Overview 8](#_Toc418495382)

[6.1.2 Pattern Breakdown 8](#_Toc418495383)

[6.2 Synchronous Limit Applied Query Integration Pattern 9](#_Toc418495384)

[6.2.1 Overview 9](#_Toc418495385)

[6.2.2 Pattern Breakdown 9](#_Toc418495386)

[6.3 Synchronous Request Acknowledge Integration Pattern 11](#_Toc418495387)

[6.3.1 Overview 11](#_Toc418495388)

[6.3.2 Pattern Breakdown 11](#_Toc418495389)

[6.4 State Messaging Pattern 12](#_Toc418495390)

[6.4.1 Overview 12](#_Toc418495391)

[6.4.2 Pattern Breakdown 12](#_Toc418495392)

[6.5 Asynchronous Request with Call Back Integration Pattern 13](#_Toc418495393)

[6.5.1 Overview 13](#_Toc418495394)

[6.5.2 Pattern Breakdown 13](#_Toc418495395)

[7. SOA Middleware Integration Anti-Patterns 14](#_Toc418495396)

[7.1 State in Middleware 14](#_Toc418495397)

[7.2 Money in Middleware 14](#_Toc418495398)

[8. Abstract Service Patterns 15](#_Toc418495399)

[8.1 SOA Synchronous Service Pattern 15](#_Toc418495400)

[8.1.1 Overview 15](#_Toc418495401)

[8.1.2 Pattern Breakdown 15](#_Toc418495402)

[9. Technology Based Service Implementation Patterns 17](#_Toc418495403)

[9.1 Oracle Service Bus SOA Synchronous Service Pattern 17](#_Toc418495404)

[9.1.1 Overview 17](#_Toc418495405)

[9.1.2 Pattern Breakdown 18](#_Toc418495406)

[9.2 Oracle SOA Suite Composite SOA Synchronous Service Pattern 20](#_Toc418495407)

[9.2.1 Overview 20](#_Toc418495408)

[9.2.2 Pattern Breakdown 21](#_Toc418495409)

[10. Solution Architect Patterns signoff 24](#_Toc418495410)

## Table of Figures

[Figure 1 - Point to Point Integration Pattern 7](#_Toc417989603)

[Figure 2 - Synchronous Query Integration Pattern 8](#_Toc417989604)

[Figure 3 - Synchronous Limit Applied Query Integration Pattern 9](#_Toc417989605)

[Figure 4 - Synchronous Request Acknowledge Integration Pattern 11](#_Toc417989606)

[Figure 5 - State Messaging Pattern 12](#_Toc417989607)

[Figure 6 - Asynchronous Request with Call Back Integration Pattern 13](#_Toc417989608)

[Figure 8 - SOA Synchronous Service Pattern 15](#_Toc417989609)

[Figure 9 - Oracle Service Bus SOA Synchronous Service Pattern 17](#_Toc417989610)

[Figure 10 - Oracle SOA Suite Composite SOA Synchronous Service Pattern 20](#_Toc417989611)

## Introduction

* 1. Purpose

The purpose of this document is to describe the various integration and implementation patterns that will be used to design and implement SOA based integrations.

* 1. Intended Audience

The intended audience for this document are:

* Middleware Solution Architect in production of integration solution architecture
* Developers of the integration solution
  1. Document Summary

This document is captured after the identification of all integrations. It will be updated over time to reflect new integrations, and changes to existing integrations, as time moves forward.

It covers the following areas:

* Integration Patterns
* Abstract Service Patterns
* Technology Based Implementation Patterns

## Non-SOA Middleware Integration Patterns

* 1. Point to Point Integration Pattern
     1. Overview



Figure 1 - Point to Point Integration Pattern

The point to point integration pattern is used only in extreme circumstances where using a service based pattern is not practical or possible. The ICC will need to provide specific approval for the use of this pattern.

Below is a list of possible situations that would lead to the implementation of the point to point integration pattern:

* If the middleware service is not re-usable and will not have any re-use potential in the future
* If the size of the message to be communicated over the integration is too large to be communicated through the SOA middleware. Refer to the Integration Principles for the maximum size a message has to reach before being considered a candidate for the Point to Point Integration Pattern. The SOA Architect should consider using pagination concepts to avoid large messages being communicated through the SOA middleware before considering the Point to Point pattern.
  + 1. Pattern Breakdown

The Consumer calls the Provider directly passing data and possibly receiving response data. The protocol used is specific to the provider e.g. Oracle Net for communication between Oracle databases.

## SOA Middleware Integration Patterns

* 1. Synchronous Query Integration Pattern
     1. Overview



Figure 2 - Synchronous Query Integration Pattern

The Synchronous Query Integration Pattern provides the mechanism by which one system in one domain can perform queries for information that lives in another domain. The consuming system does not need to have knowledge of other systems as this is abstracted by the SOA layer.

* + 1. Pattern Breakdown

#### Client Virtual Service (cvs) or application business connector (abcs)

The Client Virtual Service is an optional component of this pattern. It provides the mechanism where the client is expecting to call a specific interface that relates to the clients information model rather than the canonical model used within the SOA middleware layer.

The Client Virtual Service will perform transformation of the incoming message into a message based on the SOA middleware canonical model and route that message to the External Virtual Service.

#### External Business Service (ebs)

The External Business Service provides the entry point into a domain from other domains. In essence it provides interface decoupling from the implementation. The External Business Service will perform simple transformation and routing to the implementation of the service contained within the application Business Connector Service.

#### Application Business Connector Service (abcs)

The Internal Virtual Service provides the actual implementation of a service. It contains the knowledge and mappings required to talk to the relevant system that is part of the service domain.

The Internal Virtual Service will perform transformations between the canonical model and the system information model enabling the routing of messages to and from the relevant system.

* 1. Synchronous Limit Applied Query Integration Pattern
     1. Overview



Figure 3 - Synchronous Limit Applied Query Integration Pattern

The Synchronous Limit Applied Query Integration Pattern provides the mechanism by which one system in one domain can perform queries for information that lives in another domain. The consuming system does not need to have knowledge of other systems as this is abstracted by the SOA layer.

The pattern will limit how much information is returned by the query to ensure efficient use of resources and proper response times.

* + 1. Pattern Breakdown

#### Client Virtual Service (cvs) or application business connector (abcs)

The Client Virtual Service is an optional component of this pattern. It provides the mechanism where the client is expecting to call a specific interface that relates to the clients information model rather than the canonical model used within the SOA middleware layer.

The Client Virtual Service will perform transformation of the incoming message into a message based on the SOA middleware canonical model and route that message to the External Virtual Service.

#### External Business Service (ebs)

The External Virtual Service provides the entry point into a domain from other domains. In essence it provides interface decoupling from the implementation. The External Virtual Service will perform simple transformation and routing to the implementation of the service contained within the Internal Virtual Service.

#### Application Business Connector Service (abcs)

The Internal Virtual Service provides the actual implementation of a service. It contains the knowledge and mappings required to talk to the relevant system that is part of the service domain.

The Internal Virtual Service will perform transformations between the canonical model and the system information model enabling the routing of messages to and from the relevant system.

#### Environment Properties Store

The environment properties store contains the value the represents the maximum records that can be returned by a query implementation based on the Limit Applied Synchronous Query Integration Pattern. This limit can be updated at run-time enabling tuning of the query in real time. The IVS will read this value and ensure that the number of records that can be returned by the query does not exceed this value.

* 1. Synchronous Request Acknowledge Integration Pattern
     1. Overview



Figure 4 - Synchronous Request Acknowledge Integration Pattern

The Synchronous Request Acknowledge Integration Pattern provides the mechanism by which one system in one domain can perform queries for information that lives in another domain. The consuming system does not need to have knowledge of other systems as this is abstracted by the SOA layer.

* + 1. Pattern Breakdown

#### Client Virtual Service (cvs)

The Client Virtual Service is an optional component of this pattern. It provides the mechanism where the client is expecting to call a specific interface that relates to the clients information model rather than the canonical model used within the SOA middleware layer.

The Client Virtual Service will perform transformation of the incoming message into a message based on the SOA middleware canonical model and route that message to the External Virtual Service.

#### External Business Service (ebs)

The External Virtual Service provides the entry point into a domain from other domains. In essence it provides interface decoupling from the implementation. The External Virtual Service will perform simple transformation and routing to the implementation of the service contained within the Internal Virtual Service.

As noted in Figure 4, the response from the EVS will contain an acknowledgement that the provided request has been successfully received and or processed. The acknowledgement may take many forms, including that of a reference number or simply a Boolean value.

#### Application Business Connector Service (abcs)

The Internal Virtual Service provides the actual implementation of a service. It contains the knowledge and mappings required to talk to the relevant system that is part of the service domain.

The Internal Virtual Service will perform transformations between the canonical model and the system information model enabling the routing of messages to and from the relevant system.

* 1. State Messaging Pattern
     1. Overview

This pattern is defined at <http://soapatterns.org/design_patterns/state_messaging> and is used as a reference in the depiction and description of this pattern.



Figure 5 - State Messaging Pattern

The State Messaging pattern ensures that the SOA Middleware providing integration services does not manage state. State is rather managed by both the consumer and the provider and passed through the SOA middleware between each.

* + 1. Pattern Breakdown

The Consumer issues a Request Message to the provider (1) via the SOA Middleware. The provider creates the necessary data structures to maintain the necessary state and updates the data structures after processing this message. The provider then adds the state data to the Response Message, which it then returns back to the consumer (2). The Consumer processes the response and may then create a new request message with the same or updated state data and the process repeats itself.

* 1. Asynchronous Request with Call Back Integration Pattern
     1. Overview



Figure 6 - Asynchronous Request with Call Back Integration Pattern

The Asynchronous Request with Call Back Integration Pattern provides the mechanism by which a consumer system in one domain can perform long running queries for information, or execute long running functions, that live in another domain. The consuming system does not need to have knowledge of other systems as this is abstracted by the SOA layer. The long running nature of the query or function execution precludes the use of a synchronous integration pattern as the long running nature of the execution will very likely lead to time out issues. However, the initial request is in fact synchronous to allow for the provider to indicate acceptance of the request.

Once the query or function execution has completed the provider will make a call back to return the response. This call back places a requirement on the consumer to provide some sort of interface to enable the call back (e.g. web service, JMS queue, etc…).

Asynchronous services require a correlation identifier to be used to enable the correlation between request and response. This may be achieved by either the consumer providing a correlation identifier for the provider to return with the response, or the provider may return a correlation identifier in the initial response indicating that the request had been accepted for processing allowing the consumer to subsequently correlate the response.

* + 1. Pattern Breakdown

#### External Business Service (ebs)

The External Virtual Service provides the entry point into a domain from other domains. In essence it provides interface decoupling from the implementation. The External Virtual Service will perform simple transformation and routing to the implementation of the service contained within the Internal Virtual Service. The request into the EVS is in fact synchronous to enable it to respond to the consumer that the request has been accepted. This response may also contain a correlation identifier if one is provided by the provider.

The EVS also provides the capability to call back to the Consumer once the call back response is received from the IVS.

#### Application Business Connector Service (abcs)

The Internal Virtual Service provides the actual implementation enabling communication with the provider. It contains the knowledge and mappings required to talk to the relevant provider. The IVS will perform transformations between the canonical model and the system information model enabling the routing of messages to and from the relevant system. It provides this service for both the initial request/response as well as the call back.

## SOA Middleware Integration Anti-Patterns

The purpose of this section to clearly identify anti-patterns, or patterns which must never be used within middleware.

* 1. State in Middleware

As detailed in the State Messaging Pattern, the SOA middleware should only pass state *through* the middleware. State should never be kept in the middleware. There are some notable exceptions to this rule and they are (in these cases the state is technical:

1. Oracle BPEL
2. Dynamic Cross references
   1. Money in Middleware

This is a special class of the State in Middleware anti-pattern detailed above but is worth highlighting given the banking context. This means the middleware should NOT hold, in state, any “money” for which the Sadad interface or the bank core would not be able to balance book (therefore meaning the bank would need to audit the middleware to balance books in addition to the core). Some examples might be:

* Delayed payments where money is debited, and the credit is queued on the middleware. This would mean SADAD would not technically balance and the core would need to be accounted for money to balance.
* Delayed transfers would be as above

## Abstract Service Patterns

* 1. SOA Synchronous Service Pattern
     1. Overview

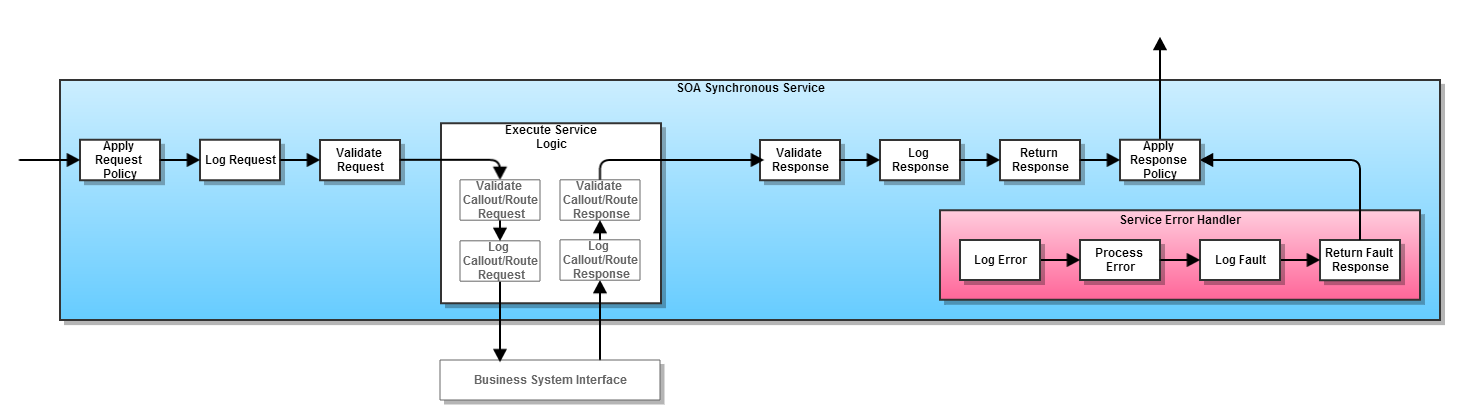


Figure 8 - SOA Synchronous Service Pattern

The SOA Synchronous Service Pattern guides the generic design for all synchronous services. It does not define technology choice, but rather provides an abstract view of the parts of a synchronous service.

* + 1. Pattern Breakdown

#### Apply Request Policy

Applies any necessary policy to the service request as required by the service. This could be security measures, encryption, etc...

#### Log Request

The request as received by the service is logged for support purposes. This logged record may also be used for auditing purposes.

#### Validate Request

Validating the request may involve two types of validation.

* XML schema validation ensures that the message received by service is understood by the service. This validation is a mandatory requirement for all services.
* Business data validation ensures that the information received by the service can actually be used by the service to successfully serve the desired function offered by the service. This validation may or may not be required and is therefore optional.

#### Execute Service Logic

The service logic is executed here. This may involve calls out to other services as well as the formation of the service response.

#### Validate Callout/Route Request

XML schema validation is performed on the request being sent to the Business System Interface to ensure the request complies with the Business System Interface.

#### Log Callout/Route Request

The request as created by the service for the service callout/route is logged for support purposes. This logged record may also be used for auditing purposes.

#### Log Callout/Route Response

The response as received by the service from the service callout/route is logged for support purposes. This logged record may also be used for auditing purposes.

#### Validate Callout/Route Response

XML schema validation is performed on the response being received from the Business System Interface. Data validation could also occur ensuring that the data required for the service response/further processing is present/represented correctly (e.g. is present, correct data type, etc...).

#### Validate Response

The response as created by the service is XML schema validated to ensure it complies with the service interface prior to returning to the service consumer.

#### Log Response

The response as created by the service is logged for support purposes. This logged record may also be used for auditing purposes.

#### Return Response

The response is returned to the service consumer.

#### Apply Response Policy

Applies any necessary policy to the service response as required by the service. This could be security measures, encryption, etc...

#### Service Error Handler

If an error occurs within the service, or via the call out to another service, the error is processed and a fault is produced for the return to the service consumer.

#### Log Error

The error caught by the service is logged for support purposes. This logged record may also be used for auditing purposes.

#### Process Error

The error is processed and a fault in compliance with the service interface is produced to return to the service consumer.

#### Log Fault

The fault as created by the error processing is logged for support purposes. This logged record may also be used for auditing purposes.

#### Return Fault Response

The fault is returned to the service consumer.

## Technology Based Service Implementation Patterns

* 1. Oracle Service Bus SOA Synchronous Service Pattern
     1. Overview

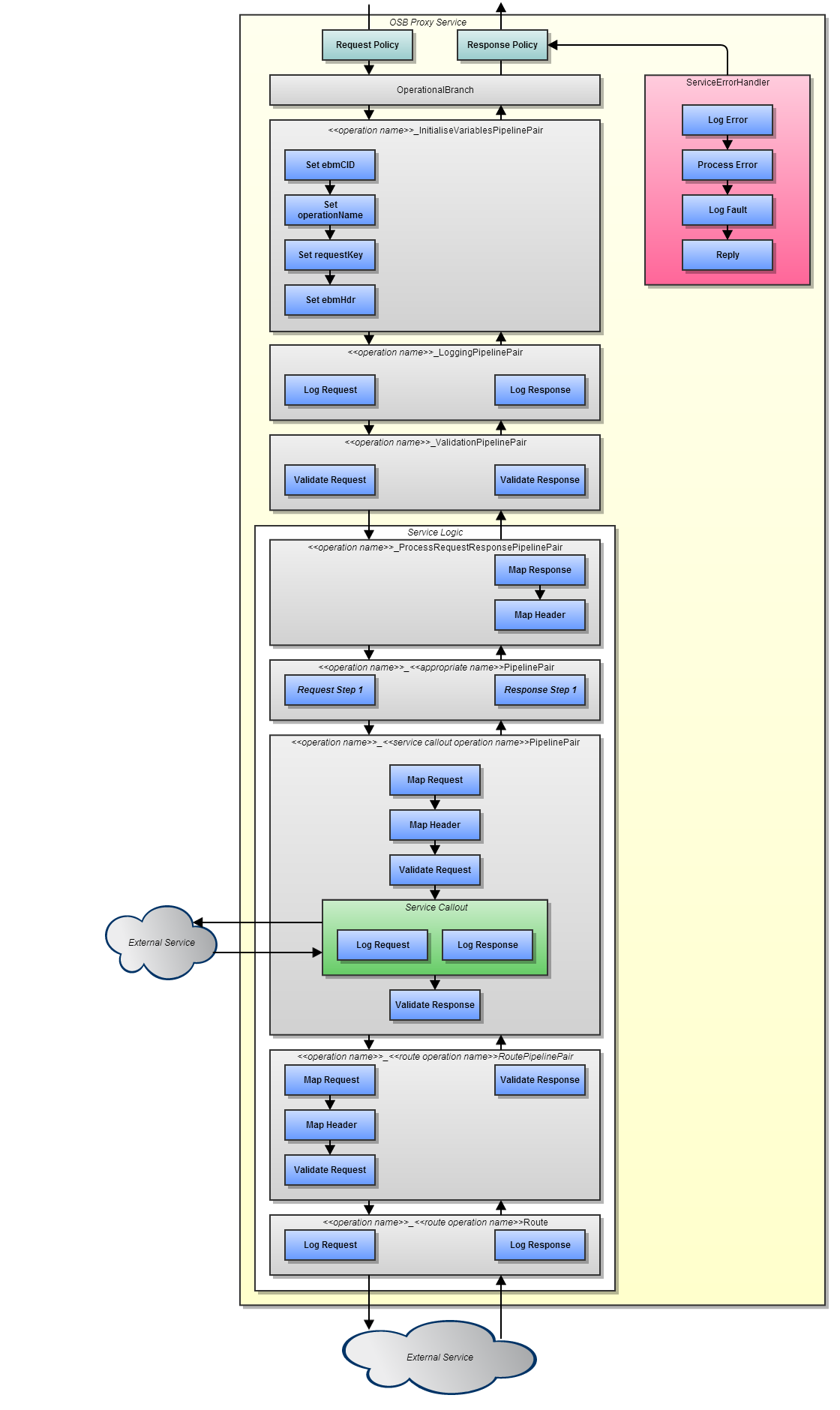


Figure 9 - Oracle Service Bus SOA Synchronous Service Pattern

The Oracle Service Bus SOA Synchronous Service Pattern defines how a synchronous service based on the SOA Synchronous Service Pattern is implemented in Oracle Service Bus

* + 1. Pattern Breakdown

The pattern break down will only address components in this pattern that are not addressed in the required detail in the SOA Synchronous Service Pattern. For components not addressed here please refer to the SOA Synchronous Service Pattern.

#### Operational Branch

The operational branch component directs a request to the correct logic within the OSB proxy component.

#### <<operation name>>\_InitialiseVariablesPipelinePair

This pipeline pair initiliases variables used within the operation processing. The variables listed in the pattern are the minimum required for use in the pattern. Others may be added as required.

#### Set ebmCID

This stage stores the ebmCID from the request header into a variable. This variable is used within the logging steps to form part of report key.

#### Set operationName

This stage stores the operation name into a variable. This variable is used within the logging steps to form part of report key.

#### Set requestKey

This stage stores the request key into a variable. The request key will vary from operation to operation. This variable is used within the logging steps to form part of report key.

#### Set ebmHdr

This stage stores the ebmHdr from the request header into a variable. This variable is used when processing errors into faults for return to the consumer.

#### <<operation name>>\_LoggingPipelinePair

This pipeline pair logs the request message received by the service and the response message about to be returned by the service to the consumer. Each logged message will include the request key, operation name, and the ebmCID from the service request header. Each step inside this pipeline pair is contained within an OSB stage.

#### <<operation name>>\_ValidationPipelinePair

This pipeline pair validates the request message. This validation will include XML schema validation as well as optional data validation. Each step inside this pipeline pair is contained within an OSB stage.

#### Operational Branch

The operational branch component directs a request to the correct logic within the OSB proxy component.

#### <<operation name>>\_InitialiseVariablesPipelinPair

This pipeline pair initiliases variables used within the operation processing. The variables listed in the pattern are the minimum required for use in the pattern. Others may be added as required.

#### Set ebmCID

This stage stores the ebmCID from the request header into a variable. This variable is used within the logging steps to form part of report key.

#### Set operationName

This stage stores the operation name into a variable. This variable is used within the logging steps to form part of report key.

#### Set requestKey

This stage stores the request key into a variable. The request key will vary from operation to operation. This variable is used within the logging steps to form part of report key.

#### Set ebmHdr

This stage stores the ebmHdr from the request header into a variable. This variable is used when processing errors into faults for return to the consumer.

#### <<operation name>>\_LoggingPipelinePair

This pipeline pair logs the request message received by the service and the response message about to be returned by the service to the consumer. Each logged message will include the request key, operation name, and the ebmCID from the service request header. Each step inside this pipeline pair is contained within an OSB stage.

#### <<operation name>>\_ValidationPipelinePair

This pipeline pair validates the request message. This validation will include XML schema validation as well as optional data validation. Each step inside this pipeline pair is contained within an OSB stage.

* 1. Oracle SOA Suite Composite SOA Synchronous Service Pattern
     1. Overview



Figure 10 - Oracle SOA Suite Composite SOA Synchronous Service Pattern

* + 1. Pattern Breakdown

The pattern break down will only address components in this pattern that are not addressed in the required detail in the SOA Synchronous Service Pattern. For components not addressed here please refer to the SOA Synchronous Service Pattern.

#### *<<component name>>*Mediator

The mediator provides a decoupling of the composite interface from its implementation. This enables the implementation to have an interface different to the formal interface for such reasons as tighter interface controls enabling tighter validation using XML schema. An example of the mediator name is *ValidateAccountMediator*.

#### *Receive Input (receive)*

Receives the input from the mediator using a receive component.

#### *Log Request*

This step logs the request message. This is automatically performed by the composite infrastructure and is not a discreet step requiring implementation by the developer.

#### *Validate Request (scope)*

This *Scope* component contains the request validation step/s enabling discreet error handling for request validation failure.

#### *Validate Request (validate)*

This step performs the actual schema validation of the request using a *Validation* component.

#### *Service Logic (scope)*

This *Scope* component contains all of the logic for this component.

#### *Step 1*

This represents the first service logic step.

#### *Step 2*

This represents the second and subsequent service logic steps.

#### *Call <<reference name>> (scope)*

This is a *Scope* component that contains all of the steps required to call a reference component, usually an external service component.

#### *Map Request (transform)*

This creates the request for the reference call using a *Transform* component. This will result in the creation of an XSL file.

#### *Map Header (assign)*

This step uses an *Assign* component to insert the header into the reference request.

#### *Validate Request (validate)*

This step performs schema validation on the reference request using a *Validate* component to ensure that the request is valid prior to sending.

#### *Log Request*

This step logs the reference request message. This is automatically performed by the composite infrastructure and is not a discreet step requiring implementation by the developer.

#### *Perform Call (invoke)*

This step executes the call to the reference using the pre-defined request using an *Invoke* component.

#### *Log Response*

This step logs the reference response message. This is automatically performed by the composite infrastructure and is not a discreet step requiring implementation by the developer.

#### *Validate Response (validate)*

This step performs schema validation on the reference response using a *Validate* component to ensure that the response is valid prior to further logic execution. If the response is invalid it will likely lead to further logic execution failing so it is better to catch an issue early for faster resolution.

#### *Create Response (scope)*

This Scope component contains the steps to create the response that will be returned to the consumer.

#### *Map Response (transform)*

This creates the response to return to the consumer using a *Transform* component. This will result in the creation of an XSL file.

#### *Map Header (assign)*

This step uses an *Assign* component to insert the header into the response.

#### *Validate Response (scope)*

This *Scope* component contains the response validation step/s enabling discreet error handling for response validation failure.

#### *Validate Response (validate)*

This step performs the actual schema validation of the response using a *Validation* component.

#### *Log Response*

This step logs the response message. This is automatically performed by the composite infrastructure and is not a discreet step requiring implementation by the developer.

#### *Reply Response (reply)*

This step returns the response message to the consumer via the mediator using a *Reply* component.

#### *Initialise Fault (transform)*

This initialises the fault to be returned with the elements that are common to all faults using a *Transform* component. This will result in the creation of an XSL file.

#### *Set Fault Details (assign)*

This populates the initialised fault message with fault specific information (e.g. error code, error message, etc…) using an Assign component.

#### *Log Fault*

This step logs the fault message. This is automatically performed by the composite infrastructure and is not a discreet step requiring implementation by the developer.

#### *Reply Fault (reply)*

This step returns the fault message to the consumer via the mediator using a *Reply* component.

#### *MDS*

The MetaData Store (MDS) centrally houses the following reusable components:

* Service WSDL’s
* Reference WSDL’s
* Fault policies
* Fault bindings
* Cross reference (XREF) definitions

## Solution Architect Patterns signoff

| **Name** | **Position** | **Date** | **Signature** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |