Chapter No.9
"BPEL with Oracle Service Bus and Service Registry"
In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter NO.9 "BPEL with Oracle Service Bus and Service Registry"

A synopsis of the book’s content

Information on where to buy this book

About the Authors

Matjaz B. Juric holds a Ph.D. in computer and information science. He is a professor at the university and head of the Cloud Computing and SOA Competence Center. Matjaz is Java Champion and Oracle ACE Director. He has more than 15 years of work experience. He has authored/coauthored Business Process Driven SOA using BPMN and BPEL, Business Process Execution Language for Web Services (English and French editions), BPEL Cookbook: Best Practices for SOA-based integration and composite applications development (award for best SOA book in 2007 by SOA World Journal), SOA Approach to Integration, Professional J2EE EAI, Professional EJB, J2EE Design Patterns Applied, and .NET Serialization Handbook. He has published chapters in More Java Gems (Cambridge University Press) and in Technology Supporting Business Solutions (Nova Science Publishers). He has also published in journals and magazines, such as SOA World Journal, Web Services Journal, Java Developer's Journal, Java Report, Java World, EAI Journal, Theserverside.com, OTN, ACM journals, and presented at conferences such as OOPSLA, Java Development, XML Europe, OOW, SCI, and others. He is a reviewer, program committee member, and conference organizer. Matjaz has been involved in several large-scale projects.

For More Information:

In cooperation with IBM Java Technology Centre, he worked on performance analysis and optimization of RMI-IIOP, an integral part of the Java platform. Matjaz is also a member of the BPEL Advisory Board.

My efforts in this book are dedicated to my family. Special thanks to my dear beautiful Ana. Thanks to my friends at SOA, Cloud Computing Center, and at Packt Publishing. Marcel Krizevnik is a researcher at the University of Maribor, where he is preparing a Ph.D. in computer and information science.

Marcel started his career as a software developer of chemistry information systems. Now, his main research areas are service-oriented architecture and cloud computing. He is also a member of the SOA Competency Center and Cloud Computing Center. In the last three years, he has been involved in several SOA technology projects.

I would like to thank my dear and beautiful Lucija for standing beside me throughout my career and while writing this book.

For More Information:
WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite 11g

Business Process Execution Language for Web Services (BPEL, WS-BPEL, or BPEL4WS) is the commonly accepted standard for defining business processes with composition of services. It is the cornerstone of Service-Oriented Architecture (SOA). With its ability to define executable and abstract business processes, it plays an important role in business process management. BPEL is supported by a majority of software vendors including Oracle, IBM, Microsoft, SAP, and others.

This book explains the role of BPEL when building SOA composite applications with Oracle SOA Suite 11g. It explains the BPEL 2.0 standard, the role of BPEL in SOA, and provides a step-by-step guide to designing and developing BPEL processes. The book also covers several related technologies and products, such as Oracle Business Activity Monitoring (Oracle BAM), Oracle Service Bus (OSB), Oracle Service Registry (OSR), Oracle Business Process Analysis Suite (Oracle BPA Suite), and Oracle Business Process Management Suite (Oracle BPM Suite).

What This Book Covers

Chapter 1, Introduction to BPEL and SOA, provides a detailed introduction to BPEL and Service-Oriented Architecture (SOA). It discusses business processes and their automation, explains the role of BPEL, Web Services, and Enterprise Service Bus (ESB) in SOA, provides insight into business process composition with BPEL, and explains the most important features.

Chapter 2, Service Composition with BPEL, discusses the composition of Web Services with BPEL. The chapter introduces the core concepts of BPEL and explains how to define synchronous and asynchronous business processes with BPEL. The reader gets familiar with the BPEL process structure, partner links, sequential and parallel service invocation, variables, conditions, and so on.

Chapter 3, Advanced BPEL, goes deeper into BPEL specifications and covers advanced features for implementing complex business processes. Advanced activities, scopes, fault handling, compensations, event handling, correlation sets, concurrent activities and links, process lifecycle, dynamic partner links, and other BPEL 2.0 features are covered in detail.

Chapter 4, Using BPEL with Oracle SOA Suite 11g, explains how to develop, deploy, test and manage BPEL processes as part of SOA composite applications in Oracle SOA.

For More Information:
Suite 11g using Oracle SOA Composite Editor, Oracle BPEL Component Designer, and Oracle Enterprise Manager Console.

Chapter 5, BPEL Extensions, Dynamic Parallel Flow, Dynamic Partner Links, Notification Service, Java Embedding, and Fault Management Framework and

Chapter 6, Entity Variables, Master and Detail Processes, Security, and Business Events in BPEL, take a detailed look at several advanced topics of using BPEL with Oracle SOA Suite 11g, such as dynamic parallel flows, dynamic partner links, Java embedding, fault management framework, entity variables, master and detail processes, security policies, business events, and more.

Chapter 7, Human Interactions in BPEL, explains how to enable human interaction in BPEL processes using the Human Task service component. The chapter discusses Oracle Human Workflow architecture and features. The reader gets familiar with how to design a human task, how to create ADF-based human task web forms, and how to access and act on tasks using the Oracle BPM Worklist Application.

Chapter 8, Monitoring BPEL Processes with BAM, discusses how to capture and monitor real-time information about the execution of business activities to improve business process effectiveness. It explains the Oracle BAM architecture and features. The chapter also discusses how to use data objects, sensors, sensor actions, monitoring objects, and how to build BAM dashboard.

For More Information:
Chapter 9, BPEL with Oracle Service Bus and Service Registry, explains how to ensure loose coupling between different components of the SOA architecture by using the Oracle Service Bus (OSB) and Oracle Service Registry (OSR). The chapter explains the OSB and OSR architecture and features. The reader gets familiar with how to publish business entities and services to OSR using the Registry Control console, how to import and export resources between OSB and OSR using the Oracle Service Bus Console, how to create OSB projects, business and proxy services. The chapter also covers some advanced features of OSB, such as service-result caching and monitoring.

Chapter 10, BPMN to BPEL Round-tripping with BPA Suite and SOA Suite, explains how to eliminate the semantic gap between IT and process models through automated translation between BPMN and BPEL using Oracle Business Process Analysis Suite (Oracle BPA Suite) and Oracle SOA Suite. The chapter discusses Oracle BPA Suite architecture and features and how various constructs map between BPMN and BPEL. The reader gets familiar with how to model a BPMN business process, how to convert a BPMN model into a BPEL Blueprint, how to import the generated BPEL code in JDeveloper, and how to propagate changes from BPEL code back to the BPMN model.

Chapter 11, Integrating BPEL with BPMN using BPM Suite, presents an interesting new feature provided by Oracle Business Process Management Suite 11g (Oracle BPM Suite 11g) - the BPMN 2.0 service engine. It allows direct execution of BPMN 2.0 processes, without the need to transform them to BPEL. The chapter discusses Oracle BPM Suite architecture and features, and demonstrates how both, BPMN and BPEL processes can be used inside a single SOA composite application.

For More Information:
If we want our SOA architecture to be highly flexible and agile, we have to ensure loose coupling between different components. As service interfaces and endpoint addresses change over time, we have to remove all point-to-point connections between service providers and service consumers by introducing an intermediate layer—Enterprise Service Bus (ESB). ESB is a key component of every mature SOA architecture and provides several important functionalities, such as message routing, transformation between message types and protocols, the use of adapters, and so on. Another important requirement for providing flexibility is service reuse. This can be achieved through the use of a UDDI (Universal Description, Discovery and Integration) compliant service registry, which enables us to publish and discover services. Using a service registry we can also implement dynamic endpoint lookup, so that service consumers retrieve actual service endpoint addresses from the registry at runtime.

In this chapter, we will get familiar with Oracle Service Bus (OSB), the Oracle strategic ESB. First, we will look at the OSB architecture and features. Then we will demonstrate the use of OSB on our Travel business process. We will show the combined use of OSB and Oracle Service Registry (OSR) to provide very high flexibility. Therefore, we will first publish the EmployeeTravelStatus service to the OSR. Then, we will open the Oracle Service Bus Console, create a new project, and import the service and all related artifacts from OSR. Next, we will create a proxy service for the EmployeeTravelStatus service. We will show how to define the proxy service message flow. We will also demonstrate some advanced features of OSB, such as service result caching. Then, we will deploy the new version of the EmployeeTravelStatus service with a slightly different interface and show how OSB can absorb these changes by simply adding an XSL transformation. Finally, we will publish the new proxy service to OSR and modify the corresponding reference binding component in our SOA composite, so that it will retrieve the proxy endpoint address from OSR at runtime.
In this chapter, we will discuss the following:

- Oracle Service Bus architecture and features
- Publishing services to OSR
- The use of Oracle Service Bus Console
- Importing resources from OSR to OSB
- Service Result Caching
- Creating a proxy service
- Adding XSL Transformations to the request pipeline
- Publishing a proxy service to OSR
- Dynamic endpoint lookup

**Oracle Service Bus architecture and features**

In Chapter 1, we discussed the importance of the ESB in SOA. We have identified that the ESB eliminates point-to-point connections between services and BPEL processes. ESB also provides a means to manage connections, control the communication between services, supervise the services and their SLAs (Service Level Agreements), and much more. The importance of the ESB often becomes visible after the first development iteration of an SOA composite application has taken place. For example, when a service requires a change in its interface or payload, the ESB can provide the transformation capabilities to mask the differences to existing service consumers. ESB can also mask the location of services, making it easy to migrate services to different servers. There are plenty other scenarios where ESB is important.

In this chapter, we will look at the Oracle Service Bus (OSB). OSB presents a communication backbone for transport and routing of messages across an enterprise. It is designed for high-throughput and reliable message delivery to a variety of service providers and consumers. It supports XML as a native data type; however, other data types are also supported. As an intermediary, it processes incoming service request messages, executes the routing logic, and transforms these messages if needed. It can also transform between different transport protocols (HTTP, JMS, File, FTP, and so on). Service response messages follow the inverse path. The message processing is specified in the message flow definition of a proxy service.
OSB provides some functionalities that are similar to the functionalities of the Mediator component within the SOA Composite, such as routing, validation, filtering, and transformation. The major difference is that the Mediator is a mediation component that is meant to work within the SOA Composite and is deployed within an SOA composite application. The OSB, on the other hand, is a standalone service bus. In addition to providing the communication backbone for all SOA (and non-SOA) applications, OSB's mission is to shield application developers from changes in the service endpoints and to prevent those systems from being overloaded with requests from upstream applications.

In addition to the Oracle Service Bus, we can also use the Mediator service component, which also provides mediation capabilities, but only within SOA composite applications. On the other hand, OSB is used for inter-application communication.

The following figure shows the functional architecture of Oracle Service Bus (OSB). We can see that OSB can be categorized into four functional layers:

- Messaging layer: Provides support to reliably connect any service by leveraging standards, such as HTTP/SOAP, WS-I, WS-Security, WS-Policy, WS-Addressing, SOAP v1.1, SOAP v1.2, EJB, RMI, and so on. It even supports the creation of custom transports using the Custom Transport Software Development Kit (SDK).
- Composition layer: Provides a configuration-driven composition environment. We can use either the Eclipse plug-in environment or web-based Oracle Service Bus Console. We can model message flows that contain content-based routing, message validation, and exception handling. We can also use message transformations (XSLT, XQuery), service callouts (POJO, Web Services), and a test browser. Automatic synchronization with UDDI registries is also supported.
Management layer: Provides a unified dashboard for service monitoring and management. We can define and monitor **Service Level Agreements (SLAs)**, alerts on operation metrics, and message pipelines, and view reports.

Proxy services and business services

OSB uses a specific terminology of proxy and business services. The objective of OSB is to route messages between business services and service consumers through proxy services.

Proxy services are generic intermediary web services that implement the mediation logic and are hosted locally on OSB. Proxy services route messages to business services and are exposed to service consumers. A proxy service is configured by specifying its interface, type of transport, and its associated message processing logic. Message flow definitions are used to define the proxy service message-handling capabilities.

Business services describe the enterprise services that exchange messages with business processes and which we want to virtualize using the OSB. The definition of a business service is similar to that of a proxy service; however, the business service does not have a message flow definition.
Message flow modeling

Message flows are used to define the message processing logic of proxy services. Message flow modeling includes defining a sequence of activities, where activities are individual actions, such as transformations, reporting, publishing, and exception management. Message flow modeling can be performed using a visual development environment (Eclipse or Oracle Service Bus Console).

Message flow definitions are defined using components, such as pipelines, branch nodes, and route nodes, as shown in the following figure:
A pipeline is a sequence of stages, representing a one-way processing path. It is used to specify message flow for service requests and responses. If a service defines more operations, a pipeline might optionally branch into operational pipelines. There are three types of pipelines:

- Request pipelines are used to specify the request path of the message flow
- Response pipelines are used to specify the response path of a message flow
- Error pipelines are used as error handlers

Request and response pipelines are paired together as pipeline pairs.

Branch nodes are used as exclusive switches, where the processing can follow one of the branches. A variable in the message context is used as a lookup variable to determine which branch to follow.

Route nodes are used to communicate with another service (in most cases a business service). They cannot have any descendants in the message flow. When the route node sends the request message, the request processing is finished. On the other side, when it receives a response message, the response processing begins.

Each pipeline is a sequence of stages that contain user-defined message processing actions. We can choose between a variety of supported actions, such as Publish, Service Callout, For Each, If... Then..., Raise Error, Reply, Resume, Skip, Delete, Insert, Replace, Validate, Alert, Log, Report, and more. Later in this chapter, we will show you how to use a pipeline on the TravelApproval process. However, let us first look at the Oracle Service Registry, which we will use together with the OSB.

**Oracle Service Registry**

*Oracle Service Registry (OSR)* is a fully V3-compliant implementation of UDDI (Universal Description, Discovery and Integration), and one of the key components of Oracle SOA Suite 11g. It allows us to publish and discover services and service providers, and manage metadata about services (security, transport, or quality service) using taxonomies. Therefore, it plays an important role when trying to improve visibility and promote service reuse. It is also important in the scope of SOA governance.

A service registry is very important for various reasons. It provides a central place where all service definitions are stored. This becomes important when the number of services (including BPEL processes) grows. It helps to maintain an overview of services. A service registry also provides a central place where developers can search for existing services. This improves service reuse, which is one of the most important aspects of SOA. Of course, a service registry also provides means to publish a services for other developers to discover and reuse.
In addition to reuse, a service registry can also be helpful when we need to migrate services from one server to the other. This can happen because of various reasons, but one of the most common reasons is the migration between the development, test, and production environments. A service registry is also helpful when we need to version services and manage changes. With a service registry, we can also develop more loosely coupled composite applications, because we do not need to hard-code the service URLs. Rather, the application will resolve URLs at run time. In all cases, the service registry is often used together with the ESB.

We will not discuss all OSR details in this chapter. We will demonstrate how to publish a service, how to export/import resources between OSB and OSR, how to browse the OSR using JDeveloper, and how to enable dynamic endpoint lookup in an SOA composite application.

## Logging into Oracle Service Registry

To log into Oracle Registry Control, we have to open a web browser and access the following URL: [http://host_name:port/application_name/uddi/web](http://host_name:port/application_name/uddi/web), where `host_name` is the name of the host on which OSR is installed, `application_name` is name of the application (default name is `registry`), and `port` is a number that is set during the installation process.

OSR provides two web consoles: **Registry Control** and **Business service Control**. Registry Control provides an interface that is based on the UDDI specification and is useful for developers familiar with business entities and tModels. Business service Control, on the other hand, provides a simpler interface for less-technical users unfamiliar with tModels and other UDDI stuff. We can publish and discover services using both. In this chapter, we will show how to use Registry Control Console.
To log into the console, we click on the **Login** link in the upper-right corner and enter username and password. The Registry Control home page opens, as shown in the following screenshot:

![Oracle Service Registry](image)

**Publishing a business entity**

Before publishing the *EmployeeTravelStatus* service, we have to publish a business entity (service provider). This is because the UDDI data model requires a business entity for each service, which is represented by a `tModel`. We click on the **Publish** link and then press the **Add business** button. We name the business entity *Packt Publishing*. We can also add a description and enter a custom business key.

![Add business](image)

We click on the **Add business** button and then **Save changes**.
Publishing a business service

Back on the Publish page, we expand the Businesses folder, right-click the Packt Publishing business entity, and select Publish WSDL.

The Publish WSDL document page opens. We enter the WSDL location (URI) of the EmployeeTravelStatus service and click Publish.

The Publish summary page opens. We can review the entities that have been published to the OSR. We click OK.
Using Oracle Service Bus Console

To log into Oracle Service Bus Console, we have to open a web browser and access the following URL: http://host_name:port/sbconsole, where host_name is the name of the host on which OSB is installed and port is a number that is set during the installation process. We log in as user weblogic. The Oracle Service Bus Console opens, as shown in the following screenshot:

The Dashboard page is opened by default, displaying information about alerts. We will show how to define and monitor alerts later in this chapter. In the upper-left corner, we can see the Change Center.

For More Information:
Change Center is key to making configuration changes in OSB. Before making any changes, we have to create a new session by clicking the Create button. Then, we are able to make different changes without disrupting existing services. When finished, we activate all changes by clicking Activate. If we want to roll back the changes, we can click the Discard button. We can also view all changes before activating them and write a comment.

Creating a project and importing resources from OSR

First, we have to create a new session, by clicking the Create button in the Change Center. Next, we will create a new project. OSB uses projects to allow logical grouping of resources and to better organize related parts of large development projects. We click on the Project Explorer link in the main menu. In the Projects page, we enter the name of the project (TravelApproval) and click Add Project. The new project is now shown in the projects list on the left side in the Project Explorer. We click on the project.
Next, we add folders to the project, as we want to group resources by type. To create a folder, we enter the folder name in the Enter New Folder Name field and click Add folder. We add six folders: BusinessServices, ProxyServices, WSDL, XSD, XSLT, and AlertDestinations.

Next, we have to create resources. We will show how to import service and all related resources from the UDDI registry. Before creating a connection to the UDDI registry, we activate the current session. First, we review all changes. We click the View Changes link in the Change Center. We can see the list of all changes in the current session. We can also undo changes by clicking the undo link in the last column.

Now, we activate the session by clicking on the Activate button. The Activate Session page opens. We can add a description to the session and click Submit.
Now, all changes made are activated.

**Creating a connection to Oracle Service Registry**

First, we start a new session in the Change Center. Then we click on the System Administration link in the main menu. We click on the UDDI Registries and then Add registry on the right side of the page. We enter connection parameters and click Save.
Now, the registry is listed in the **UDDI Registries** list, as shown next:

![UDDI Registries list](image)

We can optionally activate a current session. In that case, we have to create a new session before importing resources from UDDI.

**Importing resources from Oracle Service Registry**

We click on the **Import from UDDI** link on the left-hand side. As there is only one connection to the registry, this connection is selected by default. First, we have to select the **Business Entity**. We select **Packt Publishing**. Then we click on the **Search** button to display all services of the selected business entity. In the next screenshot, we can see that currently there is only one service published. We select the service and click **Next**.

![Import from UDDI - Search Business Services and Import](image)

In the second step, we select the project and folder, where we want to save the resources. We select the **TravelApproval** project and the folder **BusinessServices** and click **Next**.

For More Information:

On the final screen, we just click the **Import** button. Now we can see that a business service, a WSDL, and three XSD resources have been created.

All resources have been created automatically, as we imported a service from the UDDI registry. If we create resources by hand, we first have to create an XML Schema in WSDL resources, and then the Business service.

As all resources have been saved to the **BusinessServices** folder, we have to move them to appropriate folders based on their type. We go back to the **Project Explorer** and click on the **BusinessServices** folder in the **TravelApproval** project. We can see all imported resources in the **Resources** list at the bottom of the page.

We can move resources by clicking on the **Move Resource** icon and then selecting the target folder. We move the WSDL resource to the **WSDL** folder and the XML Schemas to the **XSD** folder.
Configuring a business service

If we want to monitor service metrics, such as average response time, number of messages, and number of errors, we have to enable monitoring of the business service. We will also show how to improve performances by enabling service result caching, which is a new feature in OSB 11g PS2.

Enabling service result caching

OSB supports service result caching through the use of Oracle Coherence, which is an in-memory data grid solution. In this way, we can dramatically improve performances if the response of the business service is relatively static. To enable the use of service result caching globally, we have to open the Operations | Global Settings and set Enable Result Caching to true.

In the Project Explorer, we click on our Business service. On the Configuration Details tab, we will enable service result caching. We scroll-down and edit the Message Handling Configuration. Then we expand the Advanced Settings. We select the Result Caching checkbox. Next, we have to specify the cache token, which uniquely identifies a single cache result. This is usually an ID field. In our simplified example, we do not have an ID field; therefore, we will use the employee last name for testing purposes. We enter the following cache token expression: $body/emp:employee/LastName. Then we set the expiration time to 20 minutes.

Advanced Settings

<table>
<thead>
<tr>
<th>Result Caching</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression Namespaces</td>
<td></td>
</tr>
<tr>
<td>Prefix</td>
<td>Namespace</td>
</tr>
<tr>
<td>emp</td>
<td><a href="http://packtpub.com/service/employee/">http://packtpub.com/service/employee/</a></td>
</tr>
<tr>
<td>Cache Token Expression</td>
<td>$body/emp:employee/LastName</td>
</tr>
<tr>
<td>Expiration Time</td>
<td>Use Default</td>
</tr>
</tbody>
</table>

Then, we click Next and Save.
Now, if the business service locates cached results through a cache key, it returns those cached results to the client instead of invoking the external service. If the result is not cached, the business service invokes the external service, returns the result to the client, and stores the result in cache.

Service result caching works only when the business service is invoked from a proxy service.

Enabling service monitoring

Again, we click on our Business service and then click on the Operational Settings tab. We select the Enabled checkbox next to the Monitoring and set the Aggregation Interval to 20 minutes. The aggregation interval is the sliding window of time over which metrics are computed. We can also define SLA alerts which are based on these metrics.

| Monitoring | |
|------------|--
| Monitoring | ☑ Enabled |
| Aggregation Interval | 0 hours 20 mins |
| SLA Alerts | ☑ Enable Alerting at Normal level or above |

We click Update to save the changes. Then, we activate the changes by clicking on the Activate button in the Change Center.

Testing a business service

After activating the changes, we can test the business service using the Test Console. To open the console, we select the BusinessServices folder and then click on the bug icon next to the Business service.
The Test Console opens. We set the XML payload and click the **Execute** button.

After executing the Business service, we can see the response message as shown in the next screenshot:

![Response Document](image-url)

For More Information:
Creating an Alert destination

Before creating a proxy service, we will create an Alert Destination resource, which will be later used for sending e-mail alerts to the administrator. Remember, that we have already created the AlertDestinations folder.

To be able to send e-mail alerts, we have to first configure the SMTP server on the System Administration page.

To create an Alert destination, we navigate to the AlertDestinations folder and then select the Alert Destination from the Create Resource drop-down. We set the name to Administrator and add an e-mail recipient by clicking the Add button. We enter the recipient e-mail address (we can add more recipients) and select the SMTP server.

Then we click Save twice.
Creating a proxy service

Although at the first sight it might seem redundant, using a proxy service instead of calling the original business service directly has several advantages. If we add a proxy service between the service consumer and the original service, we gain transparency. Through OSB, we can monitor and supervise the service and control the inbound and outbound messages. This becomes important when changes happen. For example, when a service interface or the payload changes, the proxy service can mask the changes to all service consumers that have not yet been upgraded to use the new version. This is, however, not the only benefit. A proxy service can enable authentication and authorization when accessing a service. It can provide a means to monitor service SLAs, and much more. Therefore, it often makes sense to consider using proxy services.

We will show an example to demonstrate the capabilities of proxy services. We will create a proxy service, which will contain the message processing logic and will be used to decouple service clients from the service provider. Our proxy service will validate the request against the corresponding XML schema. It will also perform error handling and alert the service administrator of any problems with the service execution.

First, we start a new session (if there is no active session) by clicking the Create button in the Change Center. Then we navigate to the ProxyServices folder in the Project Explorer. We click on the Create Resources drop-down and select Proxy Service.
The **General Configuration** page opens. We set the name of the proxy service to `EmployeeTravelStatusServiceProxy`. We also have to define the interface of the service. We select the **Business service**, as we want the proxy service to use the same interface as the business service. We click the **Browse** button and select the `EmployeeTravelStatusService` business service.

![Create a Proxy Service (TravelApproval/ProxyServices/)](image)

Then we click **Next**. On the **Transport Configuration** screen, we can change the transport **Protocol** and **Endpoint URI**.

![Create a Proxy Service (TravelApproval/ProxyServices/EmployeeTravelStatusServiceProxy)](image)

We use the defaults values and click **Next**. The **HTTP Transport Configuration** screen opens. We click **Next** on the remaining configuration screens. On the **Summary** page, we click the **Save** button at the bottom of the page.

---

For More Information:  
Configuring Message Flow

We select the new proxy service. We then click on the **Edit Message Flow** icon in the **Resources** list to open the **Message Flow** editor.

In the **Message Flow** editor, we can see that a route node for the **EmployeeTravelStatusService** business service already exists.

Now, we will modify the message flow by adding a pipeline pair. Then, we will edit the request pipeline and add a stage for validating the request against an XML Schema. Finally, we will add an error handler to the message flow and define a stage for sending an e-mail alert to an administrator every time the error occurs.

We click on the **EmployeeTravelStatusServiceProxy** envelope and select **Add Pipeline Pair**. A pipeline pair node is added to the message flow.
Now, we will edit the request pipeline. Remember that a pipeline contains a sequence of stages. We click on the Request Pipeline node and select Add Stage.
We click on the new stage and select **Edit Name and Comments**. We rename the stage to **Validate**. Next, we click on the stage and select **Edit Stage**. The **Edit Stage Configuration** screen appears. We click on **Add an Action** and select **Message Processing | Validate**.

We specify that we want to validate `/emp:employee` in the variable `body` against the `EmployeeType` defined in the `XMLSchema_1488776269` XML Schema, as shown in the following screenshot:

Then we click **Save** to save the changes and return to the **Message Flow** editor.

Now, we click on the envelope and select **Add Service Error Handler**. Then, we click on the **Error Handler** node and select **Add Stage**.
We rename the new stage to **Send Alert**. Next, we click on the stage and select **Edit Stage**. We add an **Alert** action as shown in the following screenshot:

![Alert Action](image)

We configure the alert as shown in the following screenshot and click **Save All**.

![Alert Configuration](image)

Then we activate the current session in **Change Center**.
Testing a proxy service

After activating the session, we can test the proxy service by selecting it and clicking on the bug icon in the Resources list. The Test Console window opens. We enter the XML payload and click Execute. We can see the service response and the invocation trace.

Now, we shut down the EmployeeTravelStatus service using the Oracle Enterprise Manager Console to test the service result caching and the error handler.

Then, we initiate the proxy service using the same input payload as before. We can see that the proxy service returns the same response although the EmployeeTravelStatus service is not running. This means that the service result caching works. Next, we initiate the proxy service using a different input payload. This time the invocation fails, as the response for this input is not cached.

For More Information:
If we click on the **Operations** link on the left-hand side and then select the **Pipeline Alerts** tab, we can see the pipeline alerts history and the details of alerts. As the proxy service execution failed, the error handler triggered an alert, as can be seen in the following screenshot:
If we click on the Service Health tab, we can monitor metrics, such as average response time, number of messages, and number of errors for the current aggregation interval.

If we check an e-mail specified when configuring the Alert destination, we can make sure that an e-mail with the alert has been automatically sent. As we can see in the next screenshot, the alert contains data about the service, error severity, alert timestamp, and so on.

If we check an e-mail specified when configuring the Alert destination, we can make sure that an e-mail with the alert has been automatically sent. As we can see in the next screenshot, the alert contains data about the service, error severity, alert timestamp, and so on.

Now we can start up the EmployeeTravelStatus service again.
Publishing a proxy service to the Oracle Service Registry

In a similar way as we imported the EmployeeTravelStatus service from the Oracle Service Registry, we can also publish our EmployeeTravelStatusServiceProxy proxy service in the opposite direction. This service will then be used from service consumers.

We click on the System Administration link in the main menu and then select Publish to UDDI. The Publish to UDDI screen appears, where we select the service we want to publish (EmployeeTravelStatusServiceProxy) and the Business entity (Packt Publishing).

We publish the service by clicking on the Publish button.

Re-wiring an SOA composite application

Now we will re-wire the TravelApproval composite application to remove the direct connection to the service and replace it with the connection to the proxy service. We open our TravelApproval SOA composite application in JDeveloper again, and double-click the composite.xml to open the SOA Composite Editor. We will modify the EmployeeTravelStatus reference binding component, as we want to remove the direct connection to the service. Instead, we will use the proxy service we just created. We will find the Proxy service by browsing the Oracle Service Registry, therefore, we have to create a connection to the OSR on the Resource Palette first.

To ensure even greater flexibility, we will enable dynamic endpoint lookup, so that the actual address of the proxy service will be retrieved from the OSR at runtime.

For More Information:
We double-click the **EmployeeTravelStatus** reference binding component. Then we click on the **Find existing WSDLs** icon next to the **WSDL URL** input field. The **SOA Resource Browser** dialog opens. We select **Resource Palette** as a source. Then we expand our UDDI connection and navigate to the **EmployeeTravelStatusServiceProxy** service, as shown in the following screenshot:

We click **OK**. The **UDDI Deployment Options** dialog opens.
We select both checkboxes and click OK. Then we click OK again to close the Update Reference dialog and return to the SOA composite editor. We save the project by clicking Save All.

Notice that we were able to simply replace the EmployeeTravelStatus service with the proxy service, as the proxy service uses the same interface. If that was not the case, we would have to modify our TravelApproval BPEL process.

Now, we can re-deploy and test our composite application.

**Oracle Service Bus use case**

By decoupling the service provider from service consumers, we have achieved high flexibility. Now, let's have a look at what happens if we deploy a new version of EmployeeTravelStatus service which has a slightly different interface and different endpoint address. If we did not use mediation, we would have to modify and re-deploy all service consumers. In our case, there is no need to do that. Instead, we can simply modify the Proxy message flow and service consumers do not need to be aware of the change.
Now, we will demonstrate this scenario. Let's say that the new version of service uses the same namespace, but different input. The old version of service accepts the employee of the following type:

```xml
<xsd:complexType name="EmployeeType">
  <xsd:sequence>
    <xsd:element name="FirstName" type="xsd:string" minOccurs="0"/>
    <xsd:element name="LastName" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Department" type="xsd:string" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The new service version uses an extended version of the EmployeeType complex type.

```xml
<xsd:complexType name="EmployeeType">
  <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Surname" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Department" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Position" type="xsd:string" minOccurs="0"/>
    <xsd:element name="Age" type="xsd:int" minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

The output of the service remains the same.

After deploying the new version, we publish it to the Oracle Service Registry. We also have to create an XSLT transformation, which will be used to transform the request message to the new type.

### Modifying the Proxy message flow

We log into the Oracle Service Bus Console again and start a new session. Then we import the new version of the EmployeeTravelStatus service (EmployeeTravelStatus-v2) from the Oracle Service Registry, as we did before. We also have to import the XSLT transformation and save it to the XSLT folder.

Next, we have to edit the proxy service message flow. First, we will modify the route node, so that it will use the new version of the service. We click on the route node and select Edit Route. Then we click on the EmployeeTravelStatusService link. We select the new version of the service.
We click **Submit** and then **Save**. Then we click on the Request Pipeline and select **Add Stage**. We rename the stage to **TransformEmployee**. Then we edit the stage. We add a replace action.

We specify that we want to replace `/emp:employee` in the variable **body**.
Now we have to select the XSLT transformation. We click on the Expression link. Then we click on XSLT Resources and select the XSLT transformation we want to use. In the Input document field we enter $body/emp:employee.

We click Save. The Edit Stage Configuration page now looks like this:

We click Save. Finally, we modify the Validate stage and specify that the input has to be validated against the new version of the XML Schema. We click Save to return to the message flow editor.

The following screenshot shows the modified Proxy message flow:

For More Information:
We click save and then activate the session by clicking the Activate button in the Change Center.

Testing an SOA composite application
We open the Oracle Enterprise Manager Console. First, we shut down the old version of the EmployeeTravelStatus service. Then, we initiate a new instance of the TravelApproval composite application to test if the deployed composite application still works and uses the new version of the service. If we open the instance flow trace, we can make sure that the new version of the EmployeeTravelStatus service has been successfully invoked.

As we can see, when using Oracle Service Bus as an intermediate layer between service providers and consumers, there is no need to modify and redeploy our existing applications when changes occur. This gives us the much-needed flexibility, so that we are able to quickly adopt to business changes.
BPEL with Oracle Service Bus and Service Registry

Summary

In this chapter, we have learned the use of Oracle Service Registry and Oracle Service Bus. Both products play an important role towards achieving a flexible SOA architecture. We have explained the architecture of Oracle Service Bus, including proxy services, business services, and message flow modeling. We have explained how to publish business entities and services to Oracle Service Registry.

We have shown how to use the service bus console, how to create a project, and import resources from the service registry. We have demonstrated how to configure business services, such as caching and monitoring. We have discussed alert destinations and shown how to create a proxy service, configure it, and publish it to the service registry. We have also shown how to re-wire the SOA composite to include the OSB proxy services. As these services are most usable when changes occur, we have demonstrated such a situation and concluded that the combined use of OSB and OSR can considerably improve the flexibility of SOA composite applications and their architecture.

In the next chapter, we will show how to implement the BPMN to BPEL round-tripping using Oracle BPA Suite and SOA Suite.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.