IBM® WebSphere® Service Registry and Repository provides the ability to strengthen an enterprise service bus (ESB) that is built into IBM WebSphere ESB to make the ESB more dynamic and adaptable.

Deploying an ESB and identifying core services creates the opportunity but does not fully complete the promise of service-oriented architecture (SOA). Maturing ESB customers are experiencing the following challenges:

- Although the ESB has decoupled the service consumer from the service provider, the ESB still has point-to-point links with the service. This connection makes for operational challenges when service versions change or the service moves locations.
- The properties that are used to configure and drive a mediation change over time. Redeploying mediations can be expensive and risky when a property based on the needs of both the business and IT must change quickly.
- As the number of services grow and as new versions of the services are created, it is increasingly difficult to decommission a service version or even determine who is using that service.
- Advertising the services that are available becomes a large challenge.
- Many ESBs are not implemented truly enterprise wide. Instead, they are implemented as business domain specific ESBs that are built to meet the requirements of a single department or project. Sharing services between these domains in a consistent and secure fashion becomes a challenge.

To address these issues and to capitalize on the opportunity created by the ESB, a service registry is needed to complete the SOA software platform. An ESB and service registry working together are core to building a dynamic SOA, enabling the ESBs to exploit policy dynamically, and transforming the way that customers use ESBs for integration. The service registry also provides the service visibility and governance needed to realize maximum return on investment. With a service registry, you can understand the services that are running in your environment, track service use, and increase service reuse.
An ESB and WebSphere Service Registry and Repository working together means:

- Dynamic endpoint selection becomes possible, which improves operational flexibility.
- Mediation policies, configured by a web-based business space widget, can be enforced.
- New services (delivered by the ESB) can be published to the service registry for lookup and reuse by more consumers.
- Changes to services (created or consumed) can be governed.
- Service Federation Management becomes possible so that you can share select services across connectivity technologies and organizational boundaries.

This IBM Redpaper™ publication demonstrates how to integrate WebSphere ESB and WebSphere Service Registry and Repository to realize the benefits of a more dynamic and flexible ESB by building mediations that show integration points between the products. Through a series of scenarios, this paper shows you how to build mediations that illustrate how WebSphere ESB can interact with WebSphere Service Registry and Repository at runtime to make mediations more dynamic and adaptable. Using a service registry allows you to change configuration data that is stored in WebSphere Service Registry and Repository to change the behavior of the mediation without updating and redeploying mediation code.

Fictional company used in the use case scenarios: The examples that we show in this paper use a fictional company called JKHL Enterprises (JKHLE).

How WebSphere Service Registry and Repository strengthens your ESB

WebSphere Service Registry and Repository strengthens your ESB in the following key ways:

- Take control of services and interfaces
- Provide greater flexibility to the ESB
- Deliver services that the business needs

Take control of services and interfaces

Many SOAs are poorly regulated. Businesses do not know what services, approved or otherwise, are actually running in the network. There is no mechanism to formalize contracts (such as service level agreements) between service providers and consuming applications, and it is impossible to track who is accessing services and business information.

Businesses want real-time visibility of service performance and availability and want to measure and demonstrate return on investment by tracking service use. They want to establish a catalog of existing services that provides visibility into capabilities and that helps reduce redundancy. This catalog can help businesses understand the impact of changes to existing services and enable them to control service consumers and providers through policies.

WebSphere Service Registry and Repository provides service discovery features to help take inventory of services that are deployed in runtime environments. On a scheduled basis,
WebSphere Service Registry and Repository can also be configured to discover services on the runtime environment to check for rogue services.

The Governance Enablement Profile (GEP) of WebSphere Service Registry and Repository gives a prescriptive approach to both service consumer and service provider governance. It manages subscriptions, documents of understanding, and service level agreements between the consumer and provider, enabling the ability to track who is using a given service.

WebSphere Service Registry and Repository also integrates with monitoring systems such as IBM Tivoli® Composite Application Manager for SOA to provide real-time visibility into service performance.

**Provide greater flexibility to the ESB**

An ESB provides a connectivity layer that enables flexible and simple integration of services, decoupling integration logic from applications. However, it can be difficult to change mediation code quickly in response to changing business requirements. Re-development and testing effort is required. It can also be complex to synchronize the mediations with the runtime status of services to invoke.

Also ESB development inherently involves invoking numerous endpoints. When endpoints are not decoupled from mediation code, promotion to other development environments requires significant changes to the ESB configuration.

Enabling dynamic and efficient access to services, policies and metadata information, allows mediations to dynamically choose service providers and endpoints and to adapt quickly to the status of critical services, greatly increasing the flexibility of the ESB.

Storing and defining some application and translation logic outside of the ESB means the logic can be updated without needing to change and redeploy the mediations. Similarly, storing and managing policies outside of the ESB means that you can define and manage the policies separately from the mediations. The ESB can then query WebSphere Service Registry and Repository at run time to determine the policies that are required for the service and can then enforce them.

WebSphere Service Registry and Repository increases the agility of WebSphere ESB through the Endpoint Lookup, SLA Check, and Policy Resolution mediation primitives. These primitives are key to accelerating robust flexible mediations, which you can change rapidly without redeployment of the mediation.

**Deliver services that the business needs**

Many business find that despite adopting SOA principles they are unable to respond quickly to new business requests, to deliver new and updated services quickly, or to address new business opportunities. The service development cycle is too long, with no standardization, and there is a lack of service information and dynamic discovery of existing capabilities. These issues have led to service duplication and an increased disconnect between business analysts and IT.

For a successful SOA, you need to create open, standardized services and reusable assets that can be used by multiple lines of business, and you need to increase productivity by reusing service assets through multiple service implementations. To achieve this level of services, you need to store, catalog, and flexibly organize assets and services and enforce guidelines to prevent uncoordinated or unapproved changes.
WebSphere Service Registry and Repository gives a prescriptive approach to the service life cycle, enabling top-down development with perspectives for business users to propose and approve services. This solution ships with example life cycles and governance policies to enforce standardized services and reusable assets plus policy analytics to help refine the governance processes and to help optimize the service development life cycle.

Integrating WebSphere ESB with WebSphere Service Registry and Repository

All of the scenarios that we describe in this paper assume that a WebSphere Service Registry and Repository definition is defined in WebSphere ESB, allowing the two servers to communicate with each other.

To perform this integration with security enabled on both servers, you need to import the WebSphere Service Registry and Repository signer certificate into the WebSphere ESB trust store.

Figure 1 shows that we imported signer certificate using the WebSphere ESB Integrated Solutions Console. Note that we set the SSL configuration for outbound connection to NodeDefaultSSLSettings.

![Figure 1 Retrieving a signer certificate](image)

Figure 1   Retrieving a signer certificate
Figure 2 shows that the signer certificate is retrieved from WebSphere Service Registry and Repository.

![Retrieved signer information](image)

Figure 2   Retrieved signer information
Then, you can create a WebSphere Service Registry and Repository definition in the WebSphere ESB Integrated Solutions Console, as shown in Figure 3.
If you click the **Connection properties** link, you can specify the following properties for the WebSphere Service Registry and Repository definition (see Figure 4):

- The Registry URL of the WebSphere Service Registry and Repository server, which includes `https://` for a secure connection and the correct TCP/IP port.
- An authentication alias to use to authenticate to WebSphere Service Registry and Repository. We used the predefined `BPC_Auth_Alias` in this example.
- The SSL Configuration that contains the signer certificate. In this example, we choose set the configuration to `NodeDefaultSSLSettings`.

![Figure 4](image)

**Figure 4** Connection properties for WebSphere Service Registry and Repository

You can click **Test Connection** to confirm that WebSphere ESB and WebSphere Service Registry and Repository are integrated successfully. Figure 5 shows the connection successful message.

![Figure 5](image)

**Figure 5** Successful connection

### Dynamic endpoint selection

SCA Import components in WebSphere ESB contain a hard-coded endpoint URL that points to a service provider. Using a hard-coded endpoint URL has its limitations. For example, if the endpoint URL of the service provider changes, you must update the SCA Import with the new URL. Using a combination of WebSphere Service Registry and Repository and the Endpoint Lookup mediation primitive enables **dynamic endpoint selection**, where the endpoint URL to use for a service provider is retrieved at run time by querying WebSphere Service Registry and Repository. You can make any future changes to the endpoint URL in WebSphere Service Registry and Repository.
Web Services Description Language (WSDL) files for each service provider are loaded into WebSphere Service Registry and Repository. You can add custom properties or classifications added to these WSDL files. The Endpoint Lookup mediation primitive in WebSphere ESB queries WebSphere Service Registry and Repository, specifying the service port type to retrieve. WebSphere Service Registry and Repository returns endpoint URLs for matching services that it finds in the registry.

The Endpoint Lookup mediation primitive can search for and retrieve endpoint information related to SOAP/HTTP, SOAP/JMS, SCA, SCA MQ, MQ JMS, JMS, Generic JMS, HTTP, manual MQ, and manual JMS protocol bindings.

The Endpoint Lookup primitive query can go beyond just looking for a particular port type. Other attributes such as namespace and version are supported, as are custom properties and classifications. The Endpoint Lookup primitive also specifies a match policy, which tells WebSphere Service Registry and Repository how many matching endpoint URLs to return and tells WebSphere ESB where those endpoint URLs should be stored in the SMO message.

The Endpoint Lookup primitive supports the following match policies:

- Return first matching endpoint and set routing target
- Return all matching endpoints
- Return all matching endpoints and set alternate routing targets
- Return endpoint matching latest compatible service version

Here are examples where dynamic endpoint selection is useful:

- It is easier to move code between environments if endpoints are not hard coded in a mediation (moving between development, test, and production endpoints for example). If the endpoint is moved outside of mediation code, you do not need to change the mediation to promote it to a different environment.
- If the endpoint of a service that is being invoked changes (for example it is moved to a different server), then you can update WebSphere Service Registry and Repository, and you do not need to update the mediation code.
- Endpoints can be updated in WebSphere Service Registry and Repository due to service availability. For example if an endpoint is unavailable, then WebSphere Service Registry and Repository can change to point at an available endpoint instead. You can use Tivoli Composite Application Manager to update WebSphere Service Registry and Repository based on monitored events.
- You can select the appropriate service based on input data.
Use case scenario: Return the first matching endpoint

**Additional materials:** You can find the mediation flow that we used in this use case scenario in the additional materials that accompany this paper in the Endpoint\AccountCreationEndpointLookupMediationModule.zip project interchange file.

The details for this scenario are as follows:

- A mediation module contains an SCA Import that points to the endpoint URL of the EligibilityGEN service provider. This endpoint URL is hard coded into the SCA Import component.
- The EligibilityGEN service provider occasionally changes its location. Currently, the mediation module must be modified manually to reflect this new URL, and the mediation module must be redeployed.
- Using dynamic endpoint lookup, the endpoint URL is retrieved at run time from WebSphere Service Registry and Repository. Any changes to the endpoint URL require changes only to the service definition in WebSphere Service Registry and Repository. The mediation module does not need to be modified or redeployed.

Figure 6 shows the solution to this use case scenario.

The following mediation primitives are used in this solution:

1. The Endpoint Lookup mediation primitive queries WebSphere Service Registry and Repository for an endpoint URL for the EligibilityGEN_Service port type. The Endpoint Lookup match policy is set to return the first endpoint URL that matches the EligibilityGEN_Service port type.
2. If an endpoint URL is returned by WebSphere Service Registry and Repository, the SCA Import uses this endpoint to invoke the EligibilityGEN service provider (instead of the endpoint URL that is hard coded in the SCA Import).
3. XSL Transformation mediation primitives are used to transform the different interfaces between the service consumer and service provider.
Building the mediation flow

Figure 7 shows the Endpoint Lookup mediation primitive properties.

![Figure 7](image)

Note the following information that we defined here:

- We entered a Name property of `EligibilityGEN_Service` and specified a Namespace of `http://jkle.com/EligibilityGEN_Service`. These settings tell WebSphere Service Registry and Repository to search for services that implement this port type (`EligibilityGEN_Service`) and namespace.

- We set the Match Policy to **Return first matching endpoint and set routing target**. This setting ensures that WebSphere Service Registry and Repository returns a maximum of one endpoint and that this endpoint is populated in the `headers/SMOHeader/Target/address` element of the SMO message, where it is used for dynamic endpoint selection.

Because the service consumer and service provider use different interfaces, we need a transformation primitive. Figure 8 shows the XSLT mediation primitive that is used in the request flow to perform this mapping. This primitive uses a Mapping file of `xslt/Request_Map_req1.map`.

![Figure 8](image)
Figure 9 shows the transformation map that the XSLT primitive uses.

![XSL Transformation primitive mapping](image)

**Figure 9**  XSL Transformation primitive mapping

Figure 10 shows the completed request flow.

![Completed request flow](image)

**Figure 10**  Completed request flow

The response flow contains an XSLT mediation primitive to map the response message from the service provider to the response message format expected by the service consumer. Figure 11 shows the response flow.

![Response Map](image)

**Figure 11**  Completed response flow

**Populating metadata into WebSphere Service Registry and Repository**

Figure 12 shows that the EligibilityGEN service definition is loaded into WebSphere Service Registry and Repository. This service definition includes an endpoint URL.

**The WSDL file:** You can find this WSDL file in the WebSphere Integration Developer workspace where you imported the mediation flow. You can find it at:

```
[workspace directory]\EligibilityGEN_Service\WebContent\WEB-INF\wsdl\EligibilityGEN_ServiceHttpPort.wsdl
```

You also need the `EligibilityGEN_Service.wsdl` and `EligibilityServiceSchema.xsd` files from the same directory.
Figure 12  Loading WSDL documents into WebSphere Service Registry and Repository

Figure 13 shows the endpoint URL defined in EligibilityGEN_ServiceHttpPort.wsdl.

Figure 13  Endpoint URL of the EligibilityGEN service
Testing the mediation

Figure 14 shows the AccountCreationEndpointLookupMediation mediation flow being tested in the Integration Test Client. A firstName value starting with JKHLE results in the web service returning a successful account activation.

![Detailed Properties](image)

Figure 14 Invocation of the AccountCreationEndpointLookupMediation mediation flow

The results from the Integration Test Client indicate that the EligibilityGEN service was invoked (Figure 15), and the Server Logs shows a response message from the EligibilityGEN service indicating that the account is eligible (Figure 16).

![Events](image)

Figure 15 Results of the mediation flow invocation
Figure 16  Server Logs showing a successful invocation of the EligibilityGEN service

Figure 17 shows the results of the SMO message after the Endpoint Lookup mediation is invoked. The \headers\SMOHeader\Target\address element of the SMO message is populated with the endpoint URL of the EligibilityGEN service, retrieved from WebSphere Service Registry and Repository.

Figure 17  Endpoint URL retrieved from WebSphere Service Registry and Repository

This test shows that dynamic endpoint lookup is functioning correctly. If the endpoint URL of the EligibilityGEN service changes in future, then you need to update only the EligibilityGEN_ServiceHttpPort.wsdl file with the new URL and load it into WebSphere Service Registry and Repository. You do not need to change the WebSphere ESB mediation.
Use case scenario: Return all matching endpoints

Additional materials: You can find the mediation flow that we used in this use case scenario in the additional materials that accompany this paper in the Endpoint\AccountCreationCustomEndpointLookupMediationModule.zip project interchange file.

The details for this scenario are as follows:

- In this mediation flow, there are three potential implementations of the Eligibility service:
  - EligibilityCIVService
  - EligibilityDoDService
  - EligibilityGenericService
- To achieve dynamic endpoint selection, all endpoint URLs for the Eligibility service are retrieved at run time from WebSphere Service Registry and Repository, and one endpoint URL must be selected based on the customer name that is supplied by the service consumer.

Figure 18 shows the solution to this use case scenario.
The following mediation primitives are used in the solution shown in Figure 18:

- The Endpoint Lookup mediation primitive queries WebSphere Service Registry and Repository for an endpoint URL for the Eligibility service. The Endpoint Lookup match policy is set to return all endpoint URLs that match the Eligibility service port type.
- The Custom mediation primitive examines all endpoint URLs returned by WebSphere Service Registry and Repository and selects a service based on a custom property in WebSphere Service Registry and Repository.
- XSLT mediation primitives are used to transform service interfaces.
- A Fail primitive is used if the Eligibility service provider responds with an error.

**Building the mediation flow**

Figure 19 shows the Endpoint Lookup mediation primitive properties.

![Endpoint Lookup: Eligibility Lookup](image)

**Figure 19  Endpoint Lookup primitive properties**

Note the following information that we defined here:

- We set the Name property to `EligibilityService` and specified a Namespace of `http://jkle.com/EligibilityService`. This information tells WebSphere Service Registry and Repository to search for services that implement this port type (EligibilityService) and namespace.
- We set the Match Policy to **Return all matching endpoints**. This setting ensures that WebSphere Service Registry and Repository returns all endpoints that match this query. The matching endpoints are populated in the `context/primitiveContext/EndpointLookupContext` element of the SMO message.

Example 1 shows the Custom mediation primitive logic. This Custom primitive performs the following steps:

1. Compares the CustomerSubType custom property from all services that are returned from WebSphere Service Registry and Repository with the first name specified in the body of the SMO message.
2. If the CustomerSubType and first name match, the `headers/SMOHeader/Target` element of the SMO message is updated with the matching service endpoint URL. This endpoint URL is used for dynamic endpoint lookup. The regular output terminal of the Custom primitive is fired.
3. If no first names match the CustomerSubType custom property in WebSphere Service Registry and Repository, then a failure message is written, and a second output terminal of the Custom primitive is fired.
Example 1  Custom mediation primitive logic

```java
String requiredPropName = "CustomerSubType";
String requiredPropValue = smo.getString("/body/account/customer/firstName");
System.out.println("Select Endpoint:: Customer First Name is: " + requiredPropValue);
boolean finished = false;

if(smo.get("/context/primitiveContext")!=null){
    java.util.List services = smo.getList("/context/primitiveContext/EndpointLookupContext");
    System.out.println("Select Endpoint:: Endpoint count: " + services.size());
    /* create an iterator to loop through the service data */
    java.util.Iterator serviceIterator = services.iterator();

    while (!finished && serviceIterator.hasNext()) {
        /* get the next service */
        EndpointLookupContextType currentService = (EndpointLookupContextType) serviceIterator.next();
        String address = currentService.getEndpointReference().getAddressElement().getValue();
        System.out.println("Select Endpoint:: Endpoint is: " + address);

        /* retrieve all of the properties for the current service */
        java.util.List properties = currentService.getRegistryAnnotations().getProperty();
        System.out.println("Select Endpoint:: Properties count: " + properties.size());
        /* create an iterator to loop through the service properties */
        java.util.Iterator propsIterator = properties.iterator();

        while (!finished && propsIterator.hasNext()) {
            /* get the next property */
            RegistryPropertyType currentProperty = (RegistryPropertyType) propsIterator.next();
            System.out.println("Select Endpoint:: Property Type read is : " + currentProperty.getName());

            /* check if the current property is the one we want */
            if (requiredPropName.equals(currentProperty.getName())) {
                /* check if this property has the correct value */
                if (requiredPropValue.equals(currentProperty.getValue())) {
                    System.out.println("Select Endpoint:: Property Value read is : " + currentProperty.getValue());
                    /* create a new message target address element */
                    TargetAddressType targetAddress =
                        com.ibm.websphere.sibx.smobo.ServiceMessageObjectFactory.eINSTANCE.createTargetAddressType();

                    /* set this element to contain the current endpoint */
                    System.out.println("Select Endpoint:: Selected endpoint is: " + address);
                    targetAddress.setAddress(address);

                    /* update the dynamic callout header */
                    smo.set("/headers/SMOHeader/Target", targetAddress);
                }
            }
        }
    }
}
```

Strengthening Your ESB with WebSphere Service Registry and Repository
if (!finished) {
    System.out.println("Select Endpoint:: No valid endpoint found for customer type: " + requiredPropValue);
    out1.fire(smo);
} else {
    out1.fire(smo);
}

Figure 20 shows the completed request flow. Note that if the Endpoint Lookup or Custom primitives do not find a matching service, the NoMatchResponse XSLT primitive is navigated to, and a response is sent back immediately to the service consumer (no service provider is invoked).

![Figure 20 Completed request flow](image1)

Figure 21 shows the completed response flow. An XSLT primitive transforms the response message into a format expected by the service consumer. Note that if an unmodelled fault is received by the mediation, the Fail mediation primitive is navigated and the service consumer receives no response.

![Figure 21 Completed response flow](image2)
Populating metadata into WebSphere Service Registry and Repository

Figure 22 shows that the Eligibility service definitions are loaded into WebSphere Service Registry and Repository. These service definition include endpoint URLs.

**Importing the WSDL files:** You can find these WSDL files in the WebSphere Integration Developer workspace where you imported the mediation flow. Import the following files, which are in the WebSphere Integration Developer workspace directory:

- `\EligibilityCIVService\WebContent\WEB-INF\wsdl\EligibilityServiceSchema.xsd`
- `\EligibilityCIVService\WebContent\WEB-INF\wsdl\EligibilityService.wsdl`
- `\EligibilityCIVService\WebContent\WEB-INF\wsdl\EligibilityCIVService.wsdl`
- `\EligibilityDoDService\WebContent\WEB-INF\wsdl\EligibilityDoDService.wsdl`
- `\EligibilityGenericService\WebContent\WEB-INF\wsdl\EligibilityGenericService.wsdl`

**Figure 22   Loading WSDL documents into WebSphere Service Registry and Repository**

![Load Documents](image)
You can add custom properties for each of the three Eligibility service port definitions, as described in Table 1. Figure 23 shows the custom properties for EligibilityGenericService.

**Table 1 Custom properties for the Eligibility services**

<table>
<thead>
<tr>
<th>Port</th>
<th>Custom property name</th>
<th>Custom property value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EligibilityGenericService_EligibilityServiceHttpPort</td>
<td>CustomerType</td>
<td>JKHLE</td>
</tr>
<tr>
<td></td>
<td>CustomerSubType</td>
<td>JKHLE-GEN</td>
</tr>
<tr>
<td>EligibilityCIVService_EligibilityServiceHttpPort</td>
<td>CustomerType</td>
<td>JKHLE</td>
</tr>
<tr>
<td></td>
<td>CustomerSubType</td>
<td>JKHLE-CIV</td>
</tr>
<tr>
<td>EligibilityDoDService_EligibilityServiceHttpPort</td>
<td>CustomerType</td>
<td>JKHLE</td>
</tr>
<tr>
<td></td>
<td>CustomerSubType</td>
<td>JKHLE-DoD</td>
</tr>
</tbody>
</table>

*Figure 23 Custom properties for the EligibilityGenericService*
Testing the mediation

Figure 24 shows the AccountCreationCustomEndpointLookupMediation mediation flow being tested in the Integration Test Client. A firstName value starting with JKHLE-CIV results in the Eligibility CIV service returning a successful account activation.

The Server Logs views (Figure 25) shows that three endpoints were returned from WebSphere Service Registry and Repository by the Endpoint Lookup mediation primitive. The Custom mediation primitive found that one of the endpoints contained a CustomerSubType property that matched the value JKHLE-CIV. Therefore, this endpoint URL (EligibilityCIVService) is used as the service provider.
If you specify a firstName of JKHLE-GEN or JKHLE-DoD, the JKHLEGenericService or JKHLEDoDService endpoint URLs are invoked instead.

If you enter any other firstName, for example ABCDE, none of the endpoint URLs that are returned by WebSphere Service Registry and Repository are used, and no service provider is invoked (Figure 26).

![Console (filtered): WebSphere Process Server v7.0 at localhost](image)

Figure 26  Server Logs view showing no valid endpoint URL is found

**Service gateway**

A service gateway is a common ESB pattern where a single endpoint is exposed to clients. This single endpoint acts as a proxy to a variety of different services that are called by the ESB but that are not exposed directly to the service consumers. In this pattern, all service consumers use only the endpoint address that is exposed by the gateway. The gateway then routes the request to the appropriate service implementation. The gateway can also add additional value by including transformations, routing, and common processing that needs to be applied to all service requests, for example by carrying out common logging requirements.

You can use the Gateway Endpoint Lookup mediation primitive in WebSphere ESB to implement a service gateway that routes web service requests to the required service based on the request's SOAPAction or the WS-Addressing Action field. The Gateway Endpoint Lookup mediation primitive uses the action field value to retrieve the endpoint for the required service from WebSphere Service Registry and Repository.

Additional materials: You can find the mediation flow that we used in this use case scenario in the additional materials that accompany this paper in the ServiceGateway\AccountCreationPolicyGateway.zip project interchange file.
Figure 27 shows an example of this mediation flow.

The service gateway receives requests for multiple services (in this example general or civilian or defense account creation requests) and based on the SOAPAction setting in the request, routes it to appropriate service provider.

Building the mediation flow

The requestResponse operation solution uses the following mediation primitives:

- The RouteMessage Gateway Endpoint Lookup primitive queries WebSphere Service Registry and Repository for an endpoint URL for a service that matches the SOAPAction property in the request message.
- The LogRequest and LogEndpointRetrievedFromWSRR custom mediation primitives are used to write messages to the system console to record processing through the mediation. They contain simple `System.out.println()` statements.
Figure 28 shows the RouteMessage mediation properties. Note that we set the Lookup Method property to **Action** and the Registry Name to `<use default registry>`.

![Gateway Endpoint Lookup : RouteMessage](image)

**Figure 28**  Gateway Endpoint Lookup primitive properties

Figure 29 shows the completed request flow.

![Completed request flow](image)

**Figure 29**  Completed request flow

Figure 30 shows the completed request flow. Note that it contains no primitives that need customizing.

![Completed response flow](image)

**Figure 30**  Completed response flow
Populating metadata into WebSphere Service Registry and Repository

**Additional materials:** You can find the WSDL files and supporting XSDs that we used in this use case scenario in the additional materials that accompany this paper in the ServiceGateway/WSDLs directory. You might need to update the endpoint addresses to match where you have deployed the services in your environment.

Figure 31 shows that the service definitions for the services that the gateway invokes are loaded into WebSphere Service Registry and Repository. These service definition include endpoint URLs.

![Figure 31](image)

**Figure 31**  Verification of endpoint artifacts

You need to export the SCA mediation module from WebSphere Integration Developer and load it as an SCA integration module into WebSphere Service Registry and Repository. Figure 32 shows the SCA mediation module loaded into WebSphere Service Registry and Repository.

![Figure 32](image)

**Figure 32**  Verification of SCA Export artifact
Testing the mediation

To test the service gateway mediation, we used a mediation module that contained SCA import components for each of the backend services that we want to invoke. Figure 33 shows the SCA import components on the assembly diagram. The SCA Import module sets the SOAPAction field of the request to the name of the service to invoke.

![Figure 33 Test client SCA Import components on assembly diagram](image)

Each SCA import is configured to invoke the service gateway rather than directly invoke the service. Figure 34 shows the import binding that is configured to point to the service gateway address, meaning that the service gateway mediation is invoked by the import.

![Figure 34 Test client import binding configuration](image)
Figure 35 shows the EligibilityGEN_ServiceImport_JAX_WSSCA import being tested in the Integration Test Client. In this test we specify a customer of JKHL.

Figure 35   Integration Test Client

Figure 36 shows result of the test.

Figure 36   Consolidated output log

Figure 37 shows the server console log for the unit test case run that shows the Service Gateway receiving the request, retrieving the endpoint, and invoking the correct service.

Figure 37   Message log output for EligibilityGEN_Service unit test case

If we repeat the test for the EligibilityDoD_Service SCA Import and EligibilityCIV_Service SCA import, respectively, the appropriate service is also invoked by the Service Gateway mediation.

**Dynamic properties**

**Additional materials:** You can find the mediation flow that we used in this use case scenario in the additional materials that accompany this paper in the Policy\AccountCreationModule.zip project interchange file.

Dynamic properties give you the power to control the properties of mediation primitives on a message-by-message basis. Using dynamic properties in WebSphere ESB can simplify a mediation flow so that it is controlled dynamically. By using the Policy Resolution mediation primitive and promoted properties, you can avoid complexity in a mediation flows where branching and multiple mediation primitives are used.
Figure 38 shows an example of this mediation flow.

The request flow of this mediation primitive should behave differently, depending on the content of the message sent by the service consumer:

- If the customerNumber sent by the service consumer is equal to 999, the message must be logged in the request flow. To achieve this, a Message Filter mediation primitive checks the value of the customerNumber property, and if that value equals 999, the Message Logger mediation primitive is invoked.

- The address element of the input message must be formatted depending on the country the request is intended for. There are three XSL Transformation mediation primitives in the request flow, to transform U.K., German, and U.S. addresses. A second Mediation Filter primitive examines the value of the country code and invokes the relevant XSLT Transformation primitive.

This request mediation flow contains six mediation primitives to perform these tasks. As the complexity of the mediation grows (more country addresses need to be supported for example), the number of mediation primitives needed also grows.

By using dynamic properties, this request mediation flow can be simplified to use just three mediation primitives (Figure 39).
In this mediation flow, the Enabled property of the Message Logger primitive and the Mapping file property of the XSL Transformation primitive are promoted. Promoting a property makes it possible to change its value dynamically at runtime. A Policy Resolution mediation primitive is used to query policies in WebSphere Service Registry and Repository. The policy definitions in WebSphere Service Registry and Repository are queried on a message-by-message basis and appropriate policy assertions are returned to the mediation. These assertions specify values to set the promoted properties to.

For example, the assertion might determine that the Enabled property of the Message Logger needs to be set to false (meaning no message is logged) and the Mapping file of the XSLT Transformation mediation primitive needs to be set to RequestTransformation_UK (to transform a U.K. address).

In addition to using policies to override promotable properties based on message content, as shown in this use case scenario, policies can override promoted properties with values that are relevant to a dynamically selected target service.

Building the mediation flow

Figure 40 shows the Policy Resolution properties.

![Policy Resolution: Account Creation Policy](image)

**Figure 40** Policy Resolution properties

Note the following information that we defined here:

- The default WebSphere Service Registry and Repository registry is used to query for policies, and the policy scope is Module, meaning these policies are associated with an SCA mediation module (not a target service).
- The following policy condition names are defined:
  - customerNumber
  - country
  
  Both of these values are retrieved from the body of the SMO message.
- The mediation policies returned by the Policy Resolution primitive are propagated to the response flow of the mediation (where they are used to determine the Enabled property of the Message Logger mediation primitive in the response flow).
Figure 41 shows the Message Logger promotable properties. Note that the Enabled property is promoted in the Promotable Properties tab. This property is promoted in the Message Logger primitive in the request and response flow.

Figure 42 shows XSL Transformation primitive properties. This primitive specifies a NoFormat.xsl mapping file. You can use policies to override this value.

Figure 43 shows that the Mapping file property of the XSL Transformation primitive is promoted.

Figure 44 shows the completed request flow.
Figure 45 shows the completed response flow.

![Completed response flow](image)

**Defining policies in WebSphere Service Registry and Repository**

You need to export the SCA mediation module from WebSphere Integration Developer and load it as an SCA integration module into WebSphere Service Registry and Repository. Figure 46 shows the SCA mediation module loaded into WebSphere Service Registry and Repository.

![SCA Integration Modules](image)

**Additional materials:** The SCA mediation module is exported from WebSphere Integration Developer in the additional materials that we supply with this paper. You can find this file at `Policy\AccountCreationModule.ear`.

When an SCA integration module is loaded into WebSphere Service Registry and Repository, a default policy is defined (see Figure 47). Note that there are assertions for `Log_Request.enabled`, `Transform_Address.XSLTransform`, and `Log_Response.enabled`. The value for these assertions is taken from the default value that is assigned to the mediation primitive properties in WebSphere Integration Developer.

By default, this policy is not attached to an SCA mediation module. If you want to use the default policy, you must attach it manually. We do not attach the default policy in this scenario.
In Business Space, you can define additional policies. For our scenario, we created a new space in Business Space containing the Module Browser and Mediation Policy Administration widgets (Figure 48).

We created three policy attachments with assertions and defined a gate condition for each. The gate condition determines whether the policy attachment is applicable.
Figure 49 shows the policy attachment AccountCreationLoggingQoSPolicy, which sets the Enabled property of both Message Logger mediation primitives to true with the gate condition that the customerNumber equals a value of 999.

![Figure 49 AccountCreationLoggingQoSPolicy policy attachment](image)

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountCreationModule.AccountCreationModule</td>
<td>Log_Request enabled</td>
<td>true</td>
</tr>
<tr>
<td>AccountCreationModule.AccountCreationModule</td>
<td>Log_Response enabled</td>
<td>true</td>
</tr>
</tbody>
</table>

Gate Conditions (Optional)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>medGate_customerNumber</td>
<td>customerNumber = 999</td>
</tr>
</tbody>
</table>

Figure 50 shows the policy attachment AddressTransformation_UK, which sets the Mapping file property of the XSL Transformation mediation primitive to xslt/RequestTransform_UK.xsl with the gate condition that the country equals a value of UK.

![Figure 50 AddressTransformation_UK policy attachment](image)

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountCreationModule.AccountCreationModule</td>
<td>Transform_Address XSL Transform</td>
<td>xslt/RequestTransform_UK.xsl</td>
</tr>
</tbody>
</table>

Gate Conditions (Optional)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>medGate_CountryUK</td>
<td>country=UK</td>
</tr>
</tbody>
</table>
Figure 51 shows the policy attachment AddressTransformation_DE which sets the Mapping file property of the XSL Transformation mediation primitive to xslt/RequestTransform_DE.xsl with the gate condition that the country equals a value of DE.

![Mediation Policy Administration](image1)

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountCreationModule</td>
<td>Transform_Address XSLTransform</td>
<td>xslt/RequestTransform_DE.xsl</td>
</tr>
</tbody>
</table>

Date Conditions (Optional)

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>modGate_CountryDE</td>
<td>country=DE</td>
</tr>
</tbody>
</table>

Figure 51  AddressTransformation_DE policy attachment

If the Policy Resolution mediation primitive queries WebSphere Service Registry and Repository for policies and no matching assertion is found for a given promoted property, the value for a promoted property specified in the Integrated Solutions Console of WebSphere ESB is used instead. If that country does not equal DE or UK, the default value of xslt/RequestTransform_USA.xsl is applied to the Mapping file property (see Figure 52).

![SCA modules](image2)

<table>
<thead>
<tr>
<th>SCA modules</th>
<th>AccountCreationModule</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The properties that are set for this module.

**General Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccountCreationModule.AccountCreationModule</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Log_Request.enabled</td>
<td>BOOLEAN</td>
<td>false</td>
</tr>
<tr>
<td>Log_Response.enabled</td>
<td>BOOLEAN</td>
<td>false</td>
</tr>
<tr>
<td>Transform_address XSLTransform</td>
<td>STRING</td>
<td>xslt/RequestTransform_USA.xsl</td>
</tr>
</tbody>
</table>

Figure 52  Integrated Solutions Console overriding the XSLTransform promoted property
Testing the mediation

Figure 53 shows the AccountCreationModule mediation flow being tested in the Integration Test Client. In this test we specify a customerNumber of 999, a firstName of JKHLE, and a country of DE.

Figure 53   Integrated Solutions Console to test policy resolution
When this test runs, the Policy Resolution mediation primitive queries WebSphere Service Registry and Repository for matching policies and returns the policy assertions where the gate condition is true. In this case, a customerNumber of 999 returns an assertion for each Message Logger, sets the Enabled promoted property to true, and sets the Mapping file promoted property to xslt/RequestTransform_DE.xsl. The SMO message from the Transform Address mediation primitive (shown in Figure 54) indicates this action.

![Figure 54](image)

Note the following information:

- The following property sets are returned from the Policy Resolution mediation primitive:
  - LogRequest.enabled = true
  - LogResponse.enabled = true
  - Transform_Address.XSLTransform = xslt/RequestTransform_DE.xsl

  These settings are used to set the value of the promoted properties in the mediation flow.

- The result of using xslt/RequestTransform_DE.xsl is shown in the body customer element, which contains the value [German Customer - DE] --> JKHLE, indicating the German mapping file was used.
Changing the input data to provide a customerNumber of 100 and a country of UK, no policy assertions are returned for LogRequest and LogResponse (meaning that the default value for the Message Logger Enabled property, false, is used). One policy assertion is returned that indicates the use of the xslt/RequestTransform_UK.xsl Mapping file (see Figure 55).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>FailInfo</td>
<td>FailInfoType</td>
<td></td>
</tr>
<tr>
<td>shared</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>userContext</td>
<td>UserContextType</td>
<td></td>
</tr>
<tr>
<td>transient</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>primitiveContext</td>
<td>PrimitiveContext</td>
<td></td>
</tr>
<tr>
<td>dynamicProperty</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>isPropagated</td>
<td>boolean</td>
<td>true</td>
</tr>
<tr>
<td>propertySets</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>propertySets[0]</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>headers</td>
<td>HeaderType</td>
<td></td>
</tr>
<tr>
<td>body</td>
<td></td>
<td></td>
</tr>
<tr>
<td>validadeGEN</td>
<td>validateGENType</td>
<td></td>
</tr>
<tr>
<td>policy</td>
<td>InsurancePolicy</td>
<td></td>
</tr>
<tr>
<td>coverValue</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>renewRate</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>customer</td>
<td>String</td>
<td>[British Customer - UK] -&gt; 3MLE</td>
</tr>
</tbody>
</table>

Figure 55   Results indicating use of the U.K. mapping file (and no Message Logger primitives)

Specifying a customerNumber of 100 and a country of USA, no policies are returned. The values specified in the Integrated Solutions Console are used instead. Therefore, the U.S. mapping file is used (as shown in Figure 56).

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>FailInfo</td>
<td>FailInfoType</td>
<td></td>
</tr>
<tr>
<td>shared</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>userContext</td>
<td>UserContextType</td>
<td></td>
</tr>
<tr>
<td>transient</td>
<td>EDBObject</td>
<td></td>
</tr>
<tr>
<td>primitiveContext</td>
<td>PrimitiveContext</td>
<td></td>
</tr>
<tr>
<td>dynamicProperty</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>isPropagated</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>propertySets</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>propertySets[0]</td>
<td>DynamicProperty</td>
<td></td>
</tr>
<tr>
<td>headers</td>
<td>HeaderType</td>
<td></td>
</tr>
<tr>
<td>body</td>
<td></td>
<td></td>
</tr>
<tr>
<td>validadeGEN</td>
<td>validateGENType</td>
<td></td>
</tr>
<tr>
<td>policy</td>
<td>InsurancePolicy</td>
<td></td>
</tr>
<tr>
<td>coverValue</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>renewRate</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>customer</td>
<td>String</td>
<td>[American Customer - USA] -&gt; 3MLE</td>
</tr>
</tbody>
</table>

Figure 56   Results indicating no policies returned
Service level agreements

You can use service level agreements (SLAs) to ensure that limited resources of business enterprises are used efficiently, by facilitating service consumers of the highest priority to consume the more of finite resources.

In the governance enablement profile of WebSphere Service Registry and Repository, an SLA represents a formal or an informal agreement between a service consumer and a service provider. An SLA can specify the agreed upon levels of functionality, performance, availability, priorities and guarantees. An example of agreed upon parameters between a consuming and providing services might include average number of messages per period, peak message rates, and availability levels.

WebSphere Service Registry and Repository stores consumer and provider service representations. An SLA can be associated with a consumer service representation in WebSphere Service Registry and Repository so that contract terms can be specified to a provider service definition contained in the same WebSphere Service Registry and Repository.

WebSphere ESB provides the SLA Check mediation primitive to check whether a particular service requester has an SLA with a particular service provider. Additionally, the SLA Check mediation primitive can using additional metadata that is stored in the WebSphere Service Registry and Repository SLA to facilitate consumer and provider interactions.

A capability version defines a specific version of a capability (service version, process version, or application version) and specifies the following main characteristics:

- A set of service level definitions (SLDs)
- A set of service level agreements (SLAs)
- A definition of the web services that deliver the capability in this particular version. Each capability version can have a consumerIdentifier attribute to identify it.

WebSphere ESB supports the following primary SLA mappings:

- SLAs between a service consumer and a mediation
- SLAs between a mediation and service provider
- SLAs between a service consumer and service provider
Figure 57 shows a use case scenario that makes use of the SLA Check mediation primitive.

The request flow of the mediation contains the following primitives:

- The SLA Check mediation primitive is used to determine if a service consumer has the appropriate SLA in place to access the service provider, EligibilityGENService. The SLA Check primitive will query WebSphere Service Registry and Repository for this information. We will define a capability version, SLA, and SLD in WebSphere Service Registry and Repository to enable this query.

- If the service consumer meets the SLA, the accept terminal of the SLA Check mediation primitive is fired. The message sent by the service consumer is transformed into the format expected by the service provider (using an XSLT mediation primitive) and sent to EligibilityGENService.

- If the service consumer does not meet the SLA, the reject terminal of the SLA Check mediation primitive is fired. The Event Emitter mediation primitive logs a Common Base Event message to the Common Event Infrastructure (CEI) to log that the request to the service provider was terminated, for audit purposes. The Stop mediation primitive then terminates the mediation flow.

Additional materials: We provide the mediation flow that we used in this use case scenario in the additional materials that accompany this paper in the SLA\AccountCreationSLAModule.zip project interchange file.
Building the mediation flow

Figure 58 shows the properties of the SLA Check mediation primitive.

<table>
<thead>
<tr>
<th>Description</th>
<th>Registry Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoint</td>
<td><a href="http://localhost:9080/EligibilityGEN_Service/EligibilityGEN_ServiceHttpPort_EligibilityGEN_ServiceHttpService">http://localhost:9080/EligibilityGEN_Service/EligibilityGEN_ServiceHttpPort_EligibilityGEN_ServiceHttpService</a></td>
</tr>
<tr>
<td>Consumer ID</td>
<td>/body/account/customer/firstName</td>
</tr>
<tr>
<td>Context ID</td>
<td>AccountCreationSLA</td>
</tr>
</tbody>
</table>

Note the following information that we defined here:

- The Registry Name specifies the WebSphere Service Registry and Repository definition that is used by the SLA Check mediation primitive. We used the default WebSphere Service Registry and Repository registry.
- The Endpoint field identifies the endpoint that is invoked by the SLA Check mediation primitive when appropriate SLAs are found in WebSphere Service Registry and Repository. This value can be a literal string value or an XPath 1.0 expression to an SMO object.
- Consumer ID and Context ID parameters are optional. The SLA Check mediation primitive uses these specified values to determine whether the consumer has the appropriate SLAs to invoke the endpoint that is defined in the endpoint parameter. We specified a Context ID with an XPath expression that points to the firstName element in the message body. The Context ID is set to a literal value of AccountCreationSLA.

Figure 59 shows the Event Emitter mediation primitive properties, which logs an event that is labeled AccountCreationModule_AccountCreationSLAModule_SLACheckReject with the body of the SMO message to a CEI server. This primitive is executed when the SLA check results in a rejection.

<table>
<thead>
<tr>
<th>Description</th>
<th>Terminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Details</td>
<td>Label: AccountCreationModule_AccountCreationSLAModule_SLACheckReject</td>
</tr>
<tr>
<td></td>
<td>Root: /body</td>
</tr>
<tr>
<td></td>
<td>Transaction mode: Default</td>
</tr>
</tbody>
</table>

Both the request and response flows contain an XSLT primitive to transform the body of the message sent between the service consumer and service provider.
Defining SLAs in WebSphere Service Registry and Repository

We loaded the Governance Enablement Profile into WebSphere Service Registry and Repository for this scenario.

To provide effective governance, ownership and accountability, you need to identify clearly who is responsible for services, such as the Account Creation service and EligibilityGEN_Service service. For our fictional JKHLE parent organization, we defined two child organizations named Common Services and Commercial.

For this scenario, we loaded metadata artifacts into WebSphere Service Registry and Repository to populate the Account Creation service consumer and EligibilityGEN_Service service provider. We established metadata relationships between the consumer and provider services and performed life cycle state transitions so that the EligibilityGEN_Service service provider is online.
We defined a Context Identifier for an SLA (Figure 62). The SLA details include the SLA Context Identifier (AccountCreationSLA), which is matched to the Context ID property on the SLA Check mediation primitive. The SLA is linked to one or more SLDs.

**Figure 62  Context Identifier for an SLA**
We specify a Consumer Identifier for the Account Creation service version (Figure 63), which includes the Consumer Identifier that is associated with this service (AccountCreationConsumerService123). The Consumer Identifier is matched to the Consumer ID property in the SLA Check mediation primitive.
Figure 64 shows the SLD, which includes the Endpoint that is associated with the backend service. The Endpoint is matched to the Endpoint property in the SLA Check mediation primitive. The backend service that provides this SLD is also linked.
Testing the mediation

Figure 65 shows the AccountCreationSLAModule mediation flow being tested in the Integration Test Client. In this test, we specify a firstName of JKHELEAccountCreationServiceConsumer123, which results in the SLA Check accepting the request to invoke the service provider.

<table>
<thead>
<tr>
<th>account</th>
<th><a href="http://www.jkhle.com/Account">http://www.jkhle.com/Account</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlns:ns0</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
</tr>
<tr>
<td>xsi:type</td>
<td>AccountBO</td>
</tr>
<tr>
<td>creditLimit</td>
<td>5000</td>
</tr>
<tr>
<td>active</td>
<td>false</td>
</tr>
<tr>
<td>customer</td>
<td></td>
</tr>
<tr>
<td>customerNumber</td>
<td>100</td>
</tr>
<tr>
<td>dateProfileCreated</td>
<td>07/24/2010</td>
</tr>
<tr>
<td>firstName</td>
<td>JKHELEAccountCreationServiceConsumer123</td>
</tr>
<tr>
<td>lastName</td>
<td>Smith</td>
</tr>
<tr>
<td>homePhoneNumber</td>
<td>111-222-3333</td>
</tr>
<tr>
<td>workPhoneNumber</td>
<td>444-444-4444</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:jkhlrinc@yahoo.com">jkhlrinc@yahoo.com</a></td>
</tr>
<tr>
<td>dataValid</td>
<td>false</td>
</tr>
<tr>
<td>Address</td>
<td>High Lane</td>
</tr>
<tr>
<td>city</td>
<td>Winchester</td>
</tr>
<tr>
<td>state</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>UK</td>
</tr>
<tr>
<td>zip</td>
<td>90222 123</td>
</tr>
</tbody>
</table>

Figure 65  Invoking a test of the SLA Check mediation primitive

Figure 66 shows that the EligibilityGEN_Service service provider was invoked. Therefore, the SLA Check primitive accepted the request.

Figure 66  SLA Check primitive accepts the request and the service provider is invoked
If we specify a different firstName, the SLA Check primitive will reject the request to invoke the service provider (Figure 67). The Event Emitter and Stop mediation primitives are invoked. Note that the EligibilityGEN_Service service provider is not invoked.

![Integration Test Client: AccountCreationSLAModule_Test](image)

*Figure 67*  SLA Check primitive rejects the request and the Event Emitter primitive is invoked

### Additional material

You can download the additional material from the Internet as we describe in this section. The additional material that is associated with this paper is available in softcopy on the Internet from the IBM Redbooks® Web server. Point your Web browser at:

*ftp://www.redbooks.ibm.com/redbooks/REDP4686*

Alternatively, you can go to the IBM Redbooks website at:

*ibm.com/redbooks*

Select **Additional materials**, and open the directory that corresponds with the IBM Redpaper form number, REDP4686.

### References

For more information, refer to the following resources:

- *Building WebSphere Enterprise Service Bus V7 Solutions* IBM Redbooks Publications Workshop (course code ZTS82 and VTS82)
  
  *http://www.redbooks.ibm.com/workshops/GR2779*

- *What's new in WebSphere Enterprise Service Bus V7*
  
What's new in WebSphere Enterprise Service Bus V6.2, Part 1: Overview

What's new in WebSphere Enterprise Service Bus V6.2, Part 2: Service gateway patterns

The team who wrote this paper

This paper was produced at the International Technical Support Organization, Raleigh Center.

Bhargav Perepa is a WebSphere IT Specialist in IBM Federal Software Group in the Washington, D.C. area, U.S. He has 15 years of experience in application and system software development at IBM. Bhargav holds a Masters degree in Computer Sciences from IIT, Chicago, and an MBA degree from the University of Texas. His areas of expertise include solution building, development, implementation, and skill transfer of WebSphere Application Server, Business Activity Monitoring (BAM), SOA, Web Services, and Service Governance technologies. Bhargav is an IBM developerWorks® contributing author and has contributed to multiple IBM Redbooks publications efforts.

Andrew Humphreys is a Consulting IT Specialist at the IBM Hursley Laboratory in the U.K. He is a Chartered Engineer and holds a Masters degree in Information Systems from The University of Huddersfield and bachelor's degree in Economics from City University, London. He has extensive experience in architecting ESB style solutions working with customers in a variety of industries using a range of IBM and other technologies.

Martin Keen is a Consulting IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products and service-oriented architecture (SOA). He also teaches IBM classes worldwide about WebSphere, SOA, and ESB. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, U.K. Martin holds a bachelor’s degree in Computer Studies from Southampton Institute of Higher Education.

Thanks to the following people for their contributions to this paper:

- Callum Jackson
- Philip Rowley
- Ian Heritage
- Brian Hulse
- David Bell
- Jim Ramaker
- Kurt Baumann
- Ray Daniel

Now you can become a published author, too!

Here’s an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks, and you can participate either in person or as a remote resident working from your home base.
Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks publications

- Find us on Facebook:
  http://www.facebook.com/IBMRedbooks
- Follow us on Twitter:
  http://twitter.com/ibmredbooks
- Look for us on LinkedIn:
  http://www.linkedin.com/groups?home=&gid=2130806
- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
- Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html
Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.