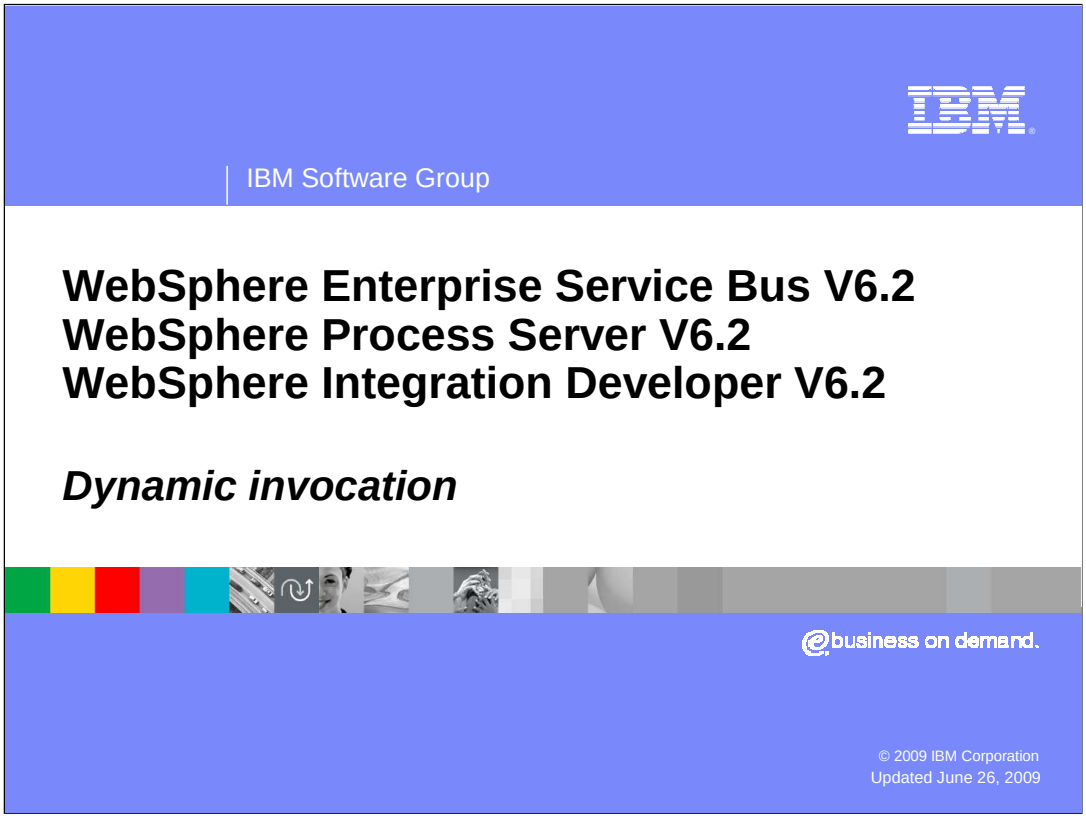


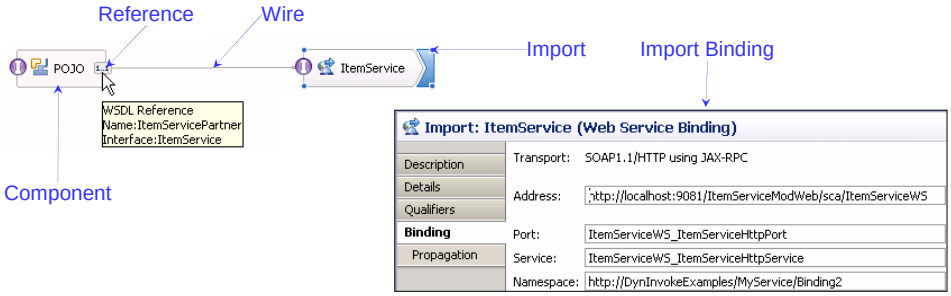
This presentation provides a look at the dynamic invocation capabilities in service component architecture, which enables the dynamic specification of service provider endpoints. It looks at dynamic invocation from both a Java™ code perspective and from a mediation flow perspective.

Goals

- The goal is to provide
  - An introduction to dynamic invocation in SCA
  - Usage from Java components and mediation flow components
- Agenda
  - Review normal service invocation in SCA
  - Overview of dynamic invocation
    - Introduce elements affecting dynamic invocation behavior
  - Describe behavior based on differing combinations of elements
  - A look at the Java API used in a Java component
  - A look at usage in a mediation flow component

The goal of this presentation is to provide you with an introduction to dynamic invocation in service component architecture (SCA). It is looked at it from both the perspective of the Java API and from the use in a mediation flow. To accomplish this goal, the presentation starts out with a review of how service invocation normally occurs in SCA, looking at both the Java API and mediation flow capabilities. An overview of dynamic invocation is then presented. This includes describing the various elements that affect the behavior of dynamic invocation. Having understood the contributing elements, you are then provided with the exact behavior for each of several variations. The discussion of the variations is applicable to both the Java API and mediation flow components. Sample code for the Java API is provided, showing the code required to do a dynamic invocation from a Java component. Finally, the usage of dynamic invocation within a mediation flow is described.

Review SCA invocation basics



- Static endpoint information contained in an import binding
- Component contains a reference (partner reference)
- Reference is wired to the import
- Component uses reference to invoke an operation on the service
- Service at static endpoint called

The basics of SCA invocation are addressed in the next couple of slides. The illustration at the top shows an SCA component and an import as they appear in an SCA assembly diagram. The binding properties for the import are also shown. The basic requirements for SCA service invocation are these. First, you need to have an import, whose binding contains static endpoint information for the target service, and defines the protocol that is used to invoke it. You also need to have an SCA component that contains a reference, sometimes referred to as a partner reference. In the assembly, the reference on the component is wired to the import. The fact that they are wired implies that they also support the same WSDL interface definition. Within the component, there is an invocation of an operation defined on the interface supported by the reference. The result is a call to the external service at the endpoint defined in the import, using the protocol defined by the import's binding.

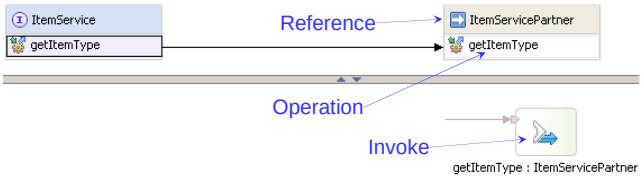
Review SCA invocation basics (continue)

- Java component Service API

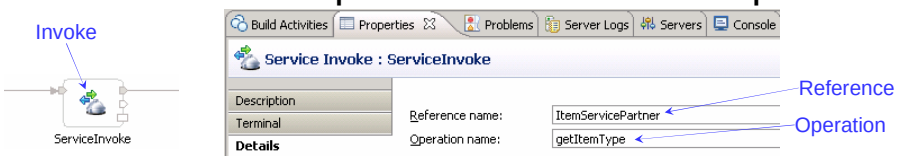
```
ServiceManager serviceManager = new ServiceManager();
Service itemService = (Service) serviceManager.locateService("ItemServicePartner");
DataObject wrappedResponse = (DataObject) itemService.invoke("getItemType", "12345");
String itemType = (String) wrappedResponse.getString("itemType");
```

Reference
Invoke
Operation

- Mediation flow component callout node



- Mediation flow component service invoke primitive



How the component does the invocation is shown here. At the top is an example of using the Java APIs from a Java component. The locateService operation on the ServiceManager is passed the name of the reference. The ServiceManager returns a Service object that represents the external service defined by the import the reference is wired to. The invoke operation on the Service object is then used to invoke a specific operation on the external service, passing it any input parameters it is expecting. The response is returned in a DataObject from which the output can be extracted.

In the middle of the slide is a screen capture, illustrating the use of a callout node in a mediation flow component. There is an operations connections section, which is the top portion of the screen capture. In there, the source and target of the flow are defined by drawing a wire between a source operation and a target operation. The target operation is associated with a reference as you can see in the screen capture. It is the import wired to the specified reference that defines the external service. In the lower portion of the screen capture is shown the callout node for the flow. When the flow runs, the message arriving at the callout node contains the input parameters which are needed to invoke the operation on the external service.

On the bottom of the slide is a service invoke primitive that is used within a mediation flow. Its properties are also shown. The properties for the service invoke primitive define the reference and operation. It is the import wired to the reference that defines the external service. When the message arrives at the service invoke primitive it contains the input parameters needed to invoke the service. The message flowing out from the service invoke primitive contains the output from the service.

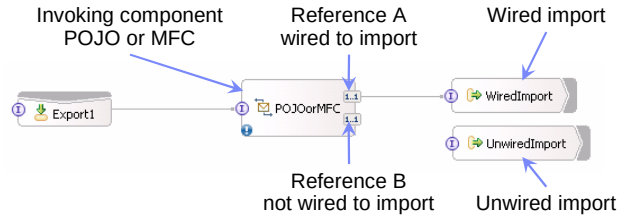
Overview of dynamic invocation

- Dynamic invocation requires:
  - A reference on the invoking component
  - An endpoint reference (EPR) defining the endpoint
- The EPR is composed of:
  - Address – URL defining the endpoint
  - Import – name of import containing configuration information
  - Binding type – defines the binding type used with the URL
- The EPR is defined by:
  - Java API - an EndpointReference object
  - Mediation flow – a TargetAddressType element in the SMO

The next few slides provide an overview of dynamic invocation and the elements that participate in enabling this capability. For there to be a dynamic invocation there are two things required, a reference on the invoking component and an endpoint reference defining the endpoint. The endpoint reference is composed of an address, import name and binding type. The address is a URL defining the endpoint of the target service. The import is the name of an import, in the same module, that contains configuration information used when invoking the service. Finally, the binding type defines the type of SCA binding applicable for invoking the service. This endpoint reference is represented by an EndpointReference object when using the Java API. In a mediation flow, the endpoint reference is represented by an element in the service message object defined as a TargetAddressType.

Overview of dynamic invocation (continue)

- Variations in dynamic invocation settings
  - The wiring of the reference to an import is optional
  - Not all values in the EPR need to be set
    - At a minimum, the EPR must have either the address or import specified
    - Import and address can both be specified
    - Binding type can be specified if address is also specified
- Dynamic invocation behavior varies based on:
  - The combination of settings in the EPR
  - Whether the reference is wired



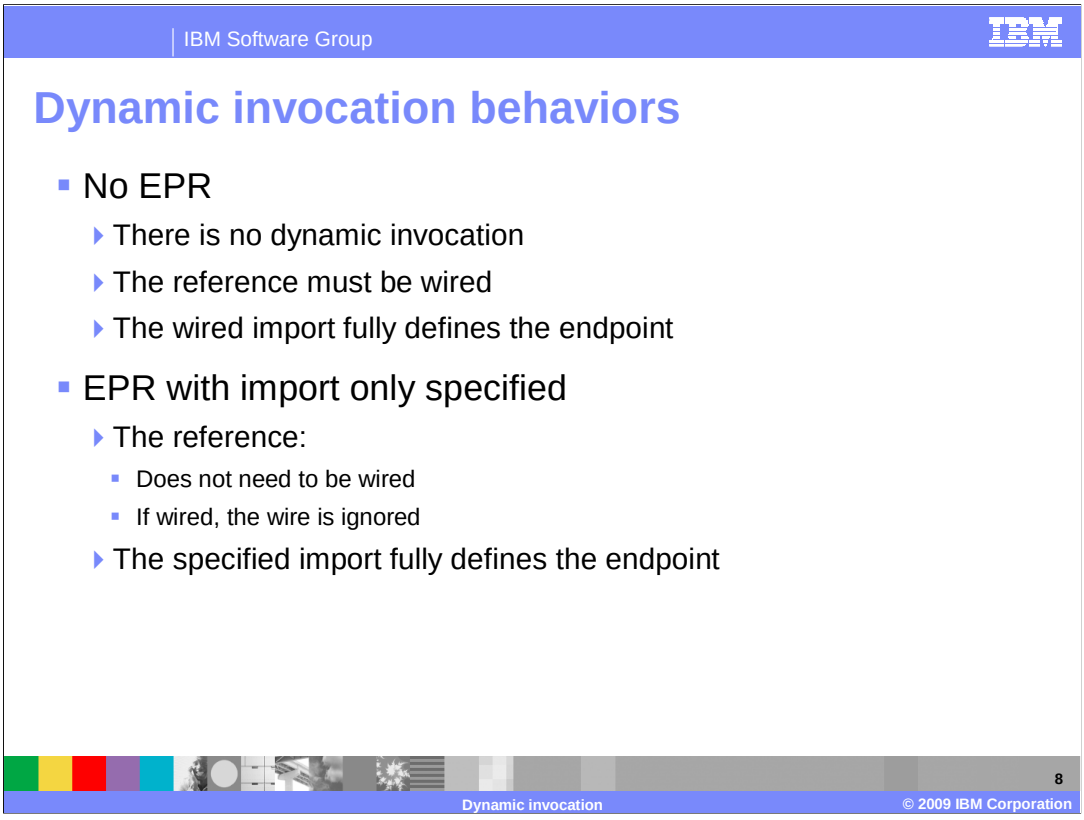
When considering the reference and endpoint reference used for dynamic invocation, there are variations in what can be specified. First of all, the reference used for dynamic invocation might be wired to an import, as it is in the static invocation case. However, it can also be left unwired so that it is not associated with an import. Then, the endpoint reference does not have to have all three elements specified. At a minimum, the endpoint reference needs to have either the address set to a URL or the import set to an import name. It is also acceptable to have both the address and import set. As for the binding type, it can be set when the address is also set. The overall behavior of dynamic invocation is dependent upon the particular combination of these various possible settings.

The illustration at the bottom of the slide is showing that when using dynamic invocation, your assembly diagram can have wired or unwired references and that imports also might be wired or unwired.

Overview of dynamic invocation (continue)

- The invocation can take these forms:
  - Uses an import containing a static endpoint URL
    - Import is the named import in the EPR
  - Uses an import and a dynamically specified URL
    - May be the wired import or the named import in the EPR
    - URL contained in the EPR
  - No import, uses a dynamically specified URL
    - URL contained in the EPR
- Invocation with URL only can be ambiguous
  - For example, an HTTP URL might be for any of these:
    - JAX-RPC SOAP 1.1
    - JAX-WS SOAP 1.1 or 1.2
    - HTTP w/o SOAP

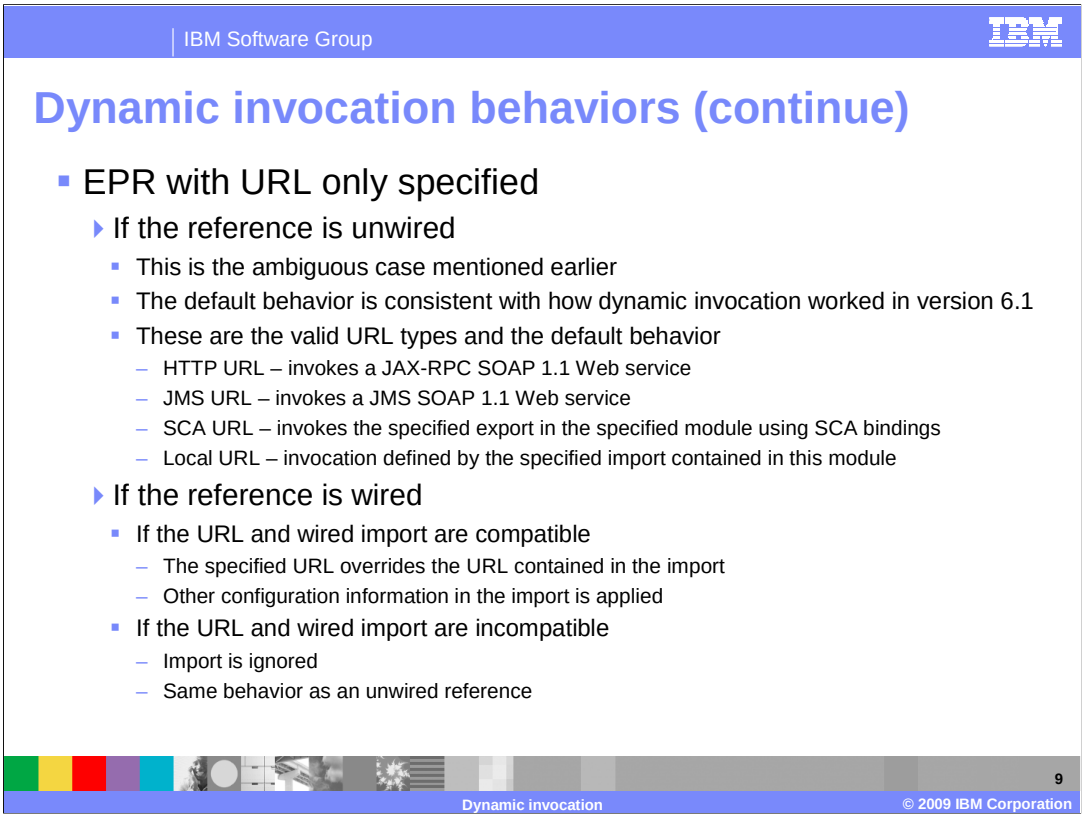
The basic variations of dynamic invocation are listed here. One possibility is to have an import name, which is dynamically specified in the endpoint reference. All endpoint information, including the URL, is obtained from the specified import. Another basic variation is to have a dynamically specified URL in the endpoint reference, that is combined with either a dynamically specified import or with an import wired to the reference. The last basic variation does not use an import, but just uses a dynamically specified URL from the endpoint reference. In this last case, the URL used for the invocation might in fact be ambiguous, as is illustrated in the example. Given an HTTP URL, the intention might be for the invocation to be a Web service call using JAX-RPC with SOAP 1.1. However, it might also be intended for a JAX-WS Web service call using SOAP 1.1 or SOAP 1.2. And finally, it might represent an HTTP call that is not a Web service at all, and does not contain a SOAP payload. Because of these ambiguities, there are rules regarding the URL only case, that are explained in an upcoming slide.



The next several slides look at specific variations of dynamic invocation and describe the behavior that you will see for each. It is important to understand these behaviors to ensure that you obtain the expected result when defining dynamic invocation to be used by your Java or mediation flow components.

The first case is when there is no endpoint reference specified. Since the endpoint reference is required, there is no dynamic invocation in this case. Therefore, it is a static invocation, and the reference must be wired to an import that fully defines the endpoint.

The first real case of dynamic invocation is when the endpoint reference only contains a named import. For this case, the reference on your component does not need to be wired, but if it is, the wired import is ignored. The import specified in the endpoint reference fully defines the endpoint for the target service.



The next case to consider is when the endpoint reference only contains the address with a URL. In this case, the behavior is different, depending upon whether the reference is wired to an import. When the reference is unwired, there exists the ambiguous situation that was mentioned earlier, and therefore there are rules defining the behavior. These rules are consistent with how dynamic invocation worked in version 6.1 and earlier releases, and eliminate the ambiguity. Only a subset of URL types are allowed. When the URL is for HTTP, the behavior is to invoke a JAX-RPC Web service using SOAP 1.1. When the URL is for JMS it results in a JMS Web service call using SOAP 1.1. SCA URLs are also allowed, which specify a module and export to be invoked using SCA default bindings. Finally, a local URL is allowed, which identifies the name of an import within the same module. The invocation is based on the binding type and configuration of the specified import.

The other situation is that the reference on the assembly is wired to an import. Assuming that the URL type and the binding type of the import are compatible, the URL from the endpoint reference overrides the URL from the import, and all other information is obtained from the import. However, if the URL type and the binding type of the import are not compatible, the wired import is ignored and the behavior is the same as that described for an unwired reference.

Dynamic invocation behaviors (continue)

- EPR with both URL and import specified
  - The reference:
    - Does not need to be wired
    - If wired, the wire is ignored
  - If the URL and specified import are compatible
    - The specified URL overrides the URL contained in the import
    - Other configuration information in the import is applied
  - If the URL and specified import are incompatible
    - An exception is raised

The next situation is when the endpoint reference contains both a URL address and a named import. In this case, it doesn't matter if the reference on the assembly is wired or not. If it is wired, the wired import is ignored. If the URL type and the binding type of the specified import are compatible, the URL from the endpoint reference overrides the URL from the import, and all other information is obtained from the import. However, if the URL type and the binding type of the import are not compatible, an exception is raised.

Dynamic invocation behaviors (continue)

- EPR with both URL and binding type specified
  - If the reference is unwired
    - Only certain combinations of URL and binding type are valid
    - These are the valid URL type and binding type combinations
      - HTTP URL and Web service binding type – invokes a JAX-RPC SOAP 1.1 Web service
      - JMS URL and Web service binding type – invokes a JMS SOAP 1.1 Web service
      - SCA URL and SCA binding type – invokes the specified export in the specified module using SCA bindings
    - Other combinations result in an exception being raised
  - If the reference is wired
    - If the URL, binding type and wired import are compatible
      - The specified URL overrides the URL contained in the import
      - Other configuration information in the import is applied
    - If the URL and binding type are compatible, but the wired import is incompatible
      - Import is ignored
      - Same behavior as an unwired reference

Another variation is when the endpoint reference contains both a URL address and a binding type specification. There is really little difference between this case and the previous one described, when only the URL is specified. However, in this case, there is a check that the binding type specified is compatible with the URL type.

For an unwired reference, these are the possibilities. In the case of an HTTP URL, the binding type must be specified as Web service and the result is a JAX-RPC Web service call using SOAP 1.1. For a JMS URL, the binding type also has to be Web service, and the result is a JMS Web service call using SOAP 1.1. When an SCA URL is specified, the binding type must be SCA, and the specified export in the specified module is invoked using the SCA default bindings. For any other combination of URL type and binding type, an exception is raised.
If the reference is wired, and the URL, binding type, and wired import are all compatible, the specified URL overrides the URL contained in the wired import. All the other configuration information is obtained from the import. However, if the wired import is not compatible, it is ignored. The effective behavior is the same as that described for the unwired reference.

Dynamic invocation behaviors (continue)

- EPR with URL, binding type and import specified
  - The reference:
    - Does not need to be wired
    - If wired, the wire is ignored
  - If the URL, binding type and specified import are compatible
    - The specified URL overrides the URL contained in the import
    - Other configuration information in the import is applied
  - If the URL, binding type or specified import are incompatible
    - An exception is raised

The last case is when the endpoint reference contains all three, the URL address, the named import, and the binding type. In this case, it doesn't matter if the reference on the assembly is wired or not. If it is wired, the wired import is ignored. If the URL type, the binding type, and the specified import are compatible, the URL from the endpoint reference overrides the URL from the specified import. All other information is obtained from the import. However, if the URL type, the binding type or the type of the specified import are not compatible, an exception is raised.

Dynamic invocation using Java APIs

- Dynamic invocation from a POJO component
  - Uses EndPointReference, ServiceManager and Service APIs

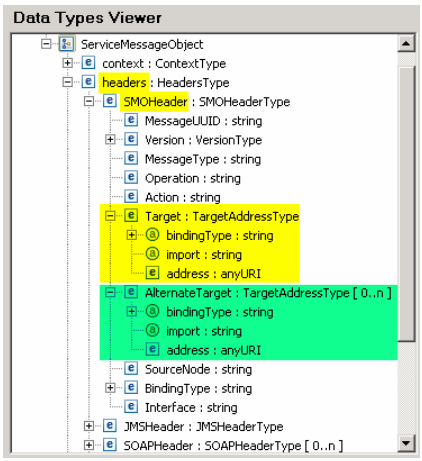
```
import com.ibm.websphere.sca.addressing.EndpointReference;
import com.ibm.websphere.sca.addressing.EndpointReferenceFactory;
import com.ibm.websphere.sca.Service;
import com.ibm.websphere.sca.ServiceManager;
import commonj.sdo.DataObject;

// Initialize name of reference to use
     String refName = "myReference";

// Get an EPR and initialize it
     EndpointReference  epr = EndpointReferenceFactory.INSTANCE.createEndpointReference();
     epr.setAddress("http://myHost:9080/MyDynamicService");
     epr.setBindingType(EndpointReference.BINDING_TYPE_HTTP);
     epr.setImport("myUnwiredHttpImport");

// Get the service passing in reference name and initialized EPR
     Service service = (Service) ServiceManager.INSTANCE.getService(refName, epr);

// Invoke the operation
     DataObject resultDO = (DataObject) service.invoke("myOperation", "inputParm");
```


The Java API for dynamic invocation is illustrated here. The import statements seen in the code identify the Java classes that are used. Specifically, these are the EndpointReference, the EndpointReferenceFactory, the Service, and the ServiceManager. The DataObject API is also needed to handle the returned value from the invoked operation.

The EndpointReferenceFactory is used to obtain an EndpointReference object. You then initialize it with the appropriate settings for address, binding type, and import to obtain the required behavior for your scenario. The ServiceManager is passed the name of the reference, and the EndpointReference object. It returns a Service object, which is initialized according to the appropriate behavior as described in the previous slides. The Service object can then be used to invoke an operation on the service, passing it the name of the operation and the input parameters. The output values from the operation are returned in a DataObject wrapper from which the output can be extracted.

Dynamic invocation using mediation flows

- Dynamic invocation from a mediation flow
  - Can be used from either a callout node or a service invoke primitive
    - Must be configured to indicate dynamic invocation is to be used
  - Uses SMO target address or alternate target address
    - Alternate target address can be used when retry is enabled
  - SMO can be set using Endpoint lookup
    - WebSphere® Service Registry and Repository only allows Web service or SCA endpoints
  - SMO can be set using any other primitive based on flow logic, such as:
    - Message element setter
    - XSL transformation
    - Database lookup



Just as with static invocation, as was described at the beginning of this presentation, dynamic invocation from a mediation flow can happen through either a callout node or service invoke primitive. Both of these contain configuration switches that are used to indicate if dynamic invocation is allowed.

The service message object (SMO) contains elements in the SMOHeader section that are used with dynamic invocation. The first is the Target, which is shown in yellow. It is a TargetAddressType, containing the three values of an endpoint reference, namely the address, import, and binding type. Then shown in green is the AlternateTarget, which is a sequence of TargetAddressType. The endpoint reference information contained in this sequence can be used by the callout, or service invoke, to perform retry processing when a service invocation fails.

The values for Target and AlternateTarget in the SMO header can be set through the use of the endpoint lookup primitive, which interacts with WebSphere Service Registry and Repository. Endpoints obtained from WebSphere Service Registry and Repository can only be for Web service and SCA endpoints.

In addition to the option of using the endpoint lookup primitive, the Target and AlternateTarget elements can be set by any other mechanism within the mediation flow. This might include, but is not limited to, the use of a message element setter, XSL transformation, or database lookup primitive.

Summary

- Provided an introduction to dynamic invocation in SCA
  - Reviewed normal service invocation in SCA
  - Presented an overview of dynamic invocation
  - Described behavior based on differing combinations
  - Looked at the Java API used in a Java component
  - Looked at usage in a mediation flow component

In this presentation you were introduced to dynamic invocation in service component architecture from both a Java API and mediation flow perspective. The presentation started out with a review of how service invocation normally occurs in SCA, looking at both the Java API and mediation flow capabilities. An overview of dynamic invocation was then presented. This included describing the various elements that affect the behavior of dynamic invocation. Having understood the contributing elements, you were provided with the exact behavior for each of several variations. Sample code for the Java API was provided, showing the code required to do a dynamic invocation from a Java component. Finally, the usage of dynamic invocation within a mediation flow was described.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_DynamicInvocation.ppt

This module is also available in PDF format at: ../WBPMv62_DynamicInvocation.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. 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 in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

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

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.