



IBM Software Group

WebSphere® Enterprise Service Bus V6.0.2
WebSphere Process Server V6.0.2
WebSphere Integration Developer V6.0.2

Mediation module and flow concepts



@business on demand.

© 2007 IBM Corporation
Updated April 17, 2007

This presentation provides an overview of mediation module and flow concepts. It examines the role a mediation module plays in enabling enterprise service bus functionality and then looks at the mediation flow component and mediation primitives that are used to define mediation logic.

Agenda

- **Introduction**
- Mediation module
- Mediation flow component
- Mediation primitives
- Development life cycle & deployment

The agenda of the presentation is shown here. This first section is a short introduction to the concepts of an enterprise service bus and the role of mediations. The presentation then takes a close look at mediation module concepts and the building blocks used to define a mediation module. An introduction to the mediation flow component and mediation primitives is provided. The presentation ends with looking at the development life cycle and deployment of a mediation module.

Note that many of the screen captures in this presentation were taken before version 6.0.2 and will have slight differences from what you see in the latest version of WebSphere Integration Developer.

Introduction

- In a loosely coupled SOA architecture, service requestors and providers connect with each other through an enterprise service bus (ESB).
- Loosely coupled services provide more flexibility and the ability to introduce mediations and qualities of service that can then be applied uniformly to the services connecting through the bus.
- Loose coupling enables protocol transformations between service requestor and provider.
- Mediation services can modify messages as they pass between service requestors and providers.
- Mediation services can make routing decisions, dynamically selecting a service provider to satisfy a request.
- Mediation services are implemented using mediation modules that contain mediation flows.
- A mediation module uses service component architecture (SCA) in the same way as a module in WebSphere Process Server.
- WebSphere Enterprise Service Bus and WebSphere Process Server provide ESB capability through the use of a mediation module deployed in the server as a J2EE application.

3

Mediation module and flow concepts

© 2007 IBM Corporation

In a service oriented architecture, services represent business functions that can be reused and combined to create flexible and responsive business systems. These services can have loosely coupled connections through an enterprise service bus (ESB) rather than being connected directly to each other.

When services are loosely coupled through an ESB the overall system has more flexibility and can be easily modified and enhanced to meet changing business requirements. The ESB can also be used to apply qualities of service uniformly to all the services connecting through the bus.

The use of an ESB provides several functional capabilities which together result in the loose coupling nature of the system. The ESB enables protocol transformations, allowing service requestors and service providers to use different protocols for communication. The ESB also provides mediation services which can inspect, modify, augment and transform a message as it flows from requestor to provider. The mediation services of the ESB can also be used to dynamically select a service provider to satisfy the request.

In WebSphere Enterprise Service Bus and WebSphere Process Server there are mediation modules which are used to provide the enterprise service bus functionality. These mediation modules are a part of service component architecture (SCA) and are very similar to SCA modules. They contain an SCA component called the mediation flow component and use exports and imports to connect to external service requestors and providers. The mediation flow component makes use of mediation primitives to define the logic of a mediation flow.

Mediation modules are deployed to WebSphere Enterprise Service Bus and WebSphere Process Server as J2EE applications.

Agenda

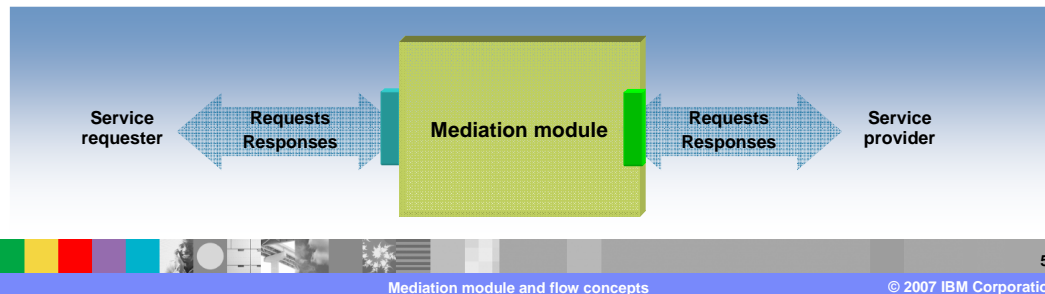
- Introduction
- **Mediation module**
- Mediation flow component
- Mediation primitives
- Development life cycle & deployment



This section of the presentation looks at the concepts of mediation modules and the building blocks used to define them.

Mediation module - Concepts

- WebSphere Enterprise Service Bus and WebSphere Process Server provide mediation modules, that:
 - ▶ Intercept and modify messages between service requester and the service provider
 - ▶ Provides the ESB functions of converting protocols, routing, transformation and other custom processing on the messages
- A mediation module is special type of service component architecture (SCA) module, containing:
 - ▶ Exports and imports
 - ▶ SCA components (only certain component types are allowed)
- A mediation module is the unit of deployment that runs in WebSphere Enterprise Service Bus or WebSphere Process Server



5

Mediation module and flow concepts

© 2007 IBM Corporation

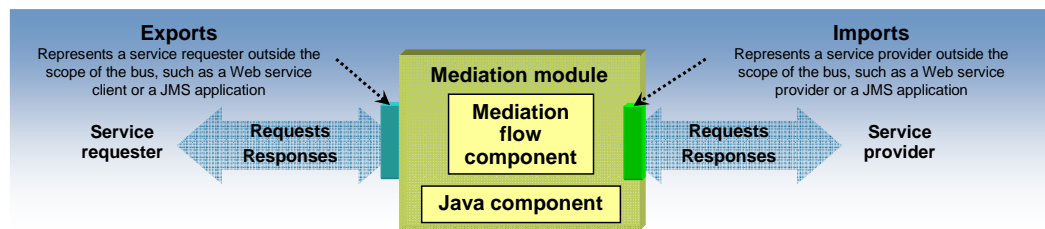
Enterprise service bus functionality is provided by WebSphere Enterprise Service Bus and WebSphere Process Server through the use of mediation modules. They intercept and modify messages as they flow between a service requestor and service provider. ESB capabilities that are provided include protocol transformation, dynamic routing decisions and message modification and transformation.

The mediation module is a special type of SCA module used specifically for ESB mediation capabilities. It contains SCA imports, exports and a limited set of SCA components. Looking at the graphic in the slide, the mediation module is shown with the blue box on the left representing SCA exports through which service requests are received. The green box on the right of the mediation module represents SCA imports through which service providers are called.

The mediation module represents the unit of deployment that is generated into a J2EE application that is installed on a WebSphere Enterprise Service Bus or WebSphere Process Server.

Mediation module - Contents

- Mediation flow component
 - ▶ Provides mediation function on messages between service requestors and providers
 - ▶ Interface defined by WSDL only
- Exports
 - ▶ Expose the mediation module to external service requestors
 - ▶ Interface defined by WSDL or Java™
- Imports
 - ▶ Identify external service providers and their interfaces
 - ▶ Interface defined by WSDL or Java
- SCA Java components
 - ▶ Used for custom mediation function or to support use of Java interfaces
 - ▶ Interface defined by WSDL or Java



This slide takes a little closer look at a mediation module, examining its contents as shown in the graphic. It is composed of SCA exports and imports, an SCA mediation flow component and optionally SCA Java components.

The heart of a mediation module is the mediation flow component which provides the mediation logic applied to a message as it flows from service requestor to a provider. It is an SCA component that can only be used in a mediation module and cannot be used in an SCA module. The interface for input and output of a mediation flow component must be defined by a Web services definition language (WSDL) document. The use of Java defined interfaces is not allowed for the mediation flow component. This component type is examined in more detail on the next slide.

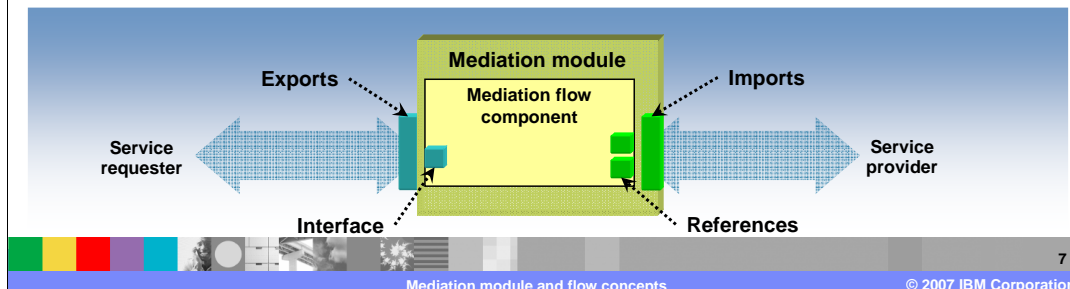
SCA exports are used to expose the mediation module to service requestors. They provide the interface and protocol definition used by the requestor to make the call. Their interfaces can be defined using either Java or WSDL interfaces.

SCA imports are used so that the mediation module can make calls out to service providers. They define the interface and protocol needed to make the call. Similar to exports, their interfaces can also be defined with WSDL or Java.

Finally, a mediation module can optionally contain SCA Java components. There are two reasons why you might need to do this. The first is to support the use of a custom mediation primitive using the invoke implementation type where the Java component in the assembly contains the custom logic defined for the primitive. The second reason would be for mapping between WSDL and Java interfaces. An example of when this would be required is where the service provider is implemented as a stateless session bean which requires the import to have a Java interface. A Java component is used to map the WSDL interface of the mediation flow component to the Java interface of the import.

Mediation module - Mediation flow component

- Mediation flow component is a type of SCA component
 - ▶ Can only be used in a mediation module, not in a module
 - ▶ Can only be one per mediation module
 - ▶ Source defined by an interface
 - ▶ Target defined by one or more references
 - ▶ Interface and references are described as WSDL interfaces, Java interfaces are not supported
- Enables ESB functionality
 - ▶ Routing decisions, selection of service provider
 - ▶ Message modification and augmentation
 - ▶ Covert messages to support different interfaces between requestor and provider



As previously mentioned, the mediation flow component is a type of SCA component that can only be used in a mediation module and not in a module. Further, a mediation module can contain only a single mediation flow component.

Looking at the graphic, you see that the mediation flow component contains a source interface and target references similar to other SCA components. The source interface is described using WSDL and must match the WSDL definition of the export to which it is wired. The target references are described using WSDL and must match the WSDL definitions of the imports or Java components to which they are wired.

The mediation module handles most of the ESB functions. These include dynamic routing and selection of service provider, message modification and augmentation and mapping of message formats between differing interfaces used by the requestor and provider.

Mediation module - Imports and exports

- Imports and Exports are configured with bindings
 - ▶ Support varying protocols
 - ▶ Enables the ESB function of protocol transformation between requester and providers
- Supported binding types
 - ▶ **Built-in SCA (default) binding**
 - Used for communication between SCA modules
 - Can be administratively modified using the administrative console or wsadmin
 - ▶ **Web services bindings**
 - SOAP/HTTP or SOAP/JMS
 - Can be administratively modified using the administrative console or wsadmin

SCA exports and imports play a major role in supporting ESB functionality in a mediation module. They are configured with bindings that specify and configure the protocol used for communication between requestor and mediation module and between mediation module and provider. The ESB functionality of protocol transformation is enabled through the use of exports and imports.

There are several different binding types, or protocols, that are supported. The type of binding and its configuration are defined at development time and some bindings also allow modifications to their configuration at runtime.

The first binding type is the built-in default binding provided by SCA which allows imports to communicate with exports in a different SCA module or mediation module. These can be configured be administratively at runtime.

There are also Web services bindings which support both SOAP over HTTP and SOAP over JMS protocols. Similar to the default SCA bindings, these can also be administratively configured at runtime.

Supported binding types (cont.)

▶ JMS binding

- Uses the default messaging provider built into in WebSphere Application Server, utilizing the service integration bus (SIB)
- Can interoperate with MQ based messaging using MQLink and MQClientLink

▶ JMS MQ binding

- Uses the WebSphere MQ JMS provider
- Works directly with WebSphere MQ JMS applications without requiring SIB and MQLink interaction

▶ MQ binding

- Enables interaction with MQ applications that are not based on JMS
- Exposes MQ header handling conventions and provides access to all header data
- Support for a variety of different request/reply correlation techniques common to MQ

▶ Adapter bindings

- WebSphere Adapters (JCA based)
- WebSphere Business Integration Adapters (JMS based)

▶ Session bean binding

- Imports only, used to call session bean as a service



There are a few different variations of bindings that support messaging protocols. JMS bindings make use of the default messaging provider that is a built-in component of WebSphere Application Server and uses the service integration bus (SIB). These are used for JMS communication with other applications using the default messaging provider or to MQ applications through configuration of an MQLink or MQClientLink in the SIB.

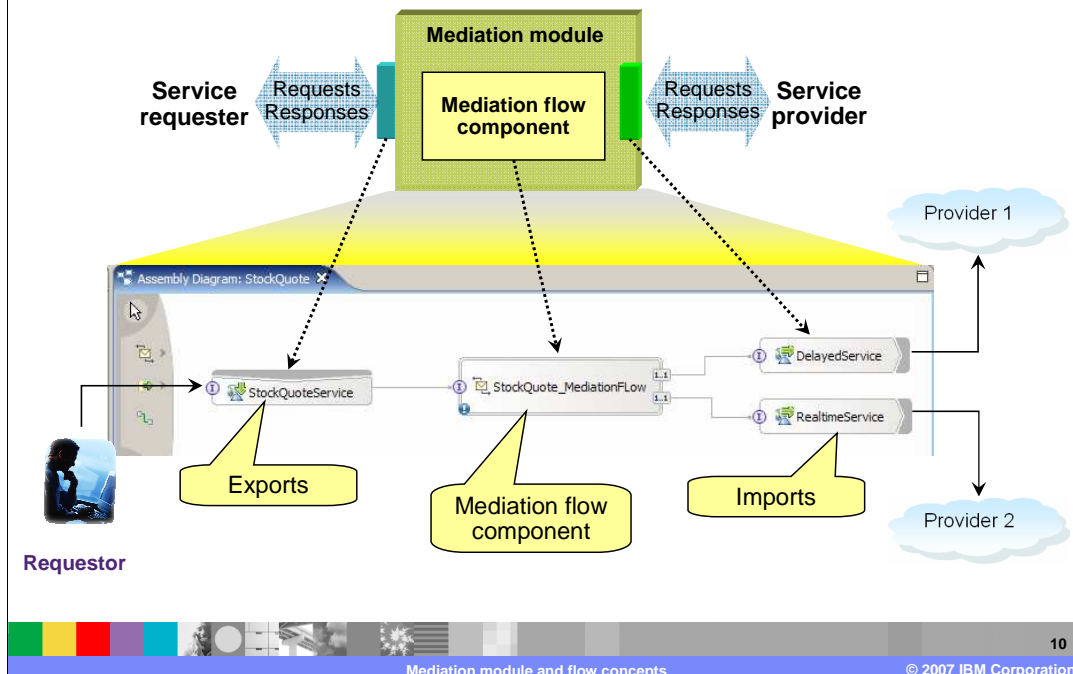
The JMS MQ bindings make use of JMS with the MQ JMS provider. This enables direct interaction with MQ JMS applications without having to go through the SIB with MQLink or MQClientLink. This enables better performance and more flexibility in configuring the use of JMS.

The MQ bindings provide support for direct communication with MQ applications that are not based on JMS. These bindings are probably the most familiar to MQ knowledgeable users, since they expose the entire chain of MQ headers and support various request/reply correlation techniques.

There are two basic forms of adapters, the WebSphere Adapters based on JCA and the WebSphere Business Integration Adapters which use a specific protocol over JMS. There are adapter specific import and export binding types that can be generated for both of these adapter varieties.

Finally, there is the session bean binding. These are supported only for imports to enable a stateless session bean to be called as a service. These import bindings only support Java interfaces and thus an intervening Java component is required when wiring a mediation module to one of these imports.

Mediation module – Assembly diagram



An assembly diagram is used to represent the contents of a mediation module. WebSphere Integration Developer provides an assembly editor which enables you to define and configure the imports, exports and mediation flow component that make up the mediation module. It is only the interface, references and associated wiring of the mediation flow component that is defined in the assembly diagram. Other editors are used to define the contents of the mediation flow component.

In addition to introducing the assembly diagram, the graphic on this slide shows you how to relate the diagrams that are used in this presentation to what you will see in WebSphere Integration Developer.

Agenda

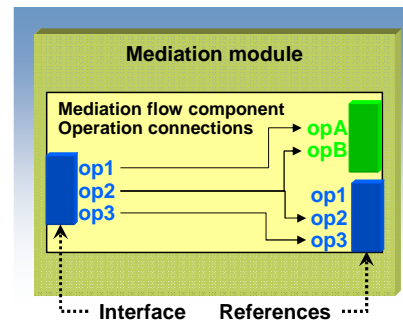
- Introduction
- Mediation module
- **Mediation flow component**
- Mediation primitives
- Development life cycle and deployment



This section provides the next level of details about the mediation flow component. It discusses operation connections, request and response flows, flow logic, service message objects and the tools in WebSphere Integration Developer.

Mediation flow component – Operation connections

- Operation connections
 - ▶ Establish relationship between source operations and target operations
 - ▶ A source operation can be related to one or more target operations
 - ▶ There must be an operation connection for each source operation
 - ▶ Not all target operations need to be connected
 - ▶ Define source and target of mediation flows
- Source operation can connect to
 - ▶ Same operation
 - Reference with the same WSDL
 - ▶ Different operation
 - Reference with a different WSDL
 - Reference with same WSDL
 - ▶ More than one operation
 - Enables flow to make routing decisions



12

Mediation module and flow concepts

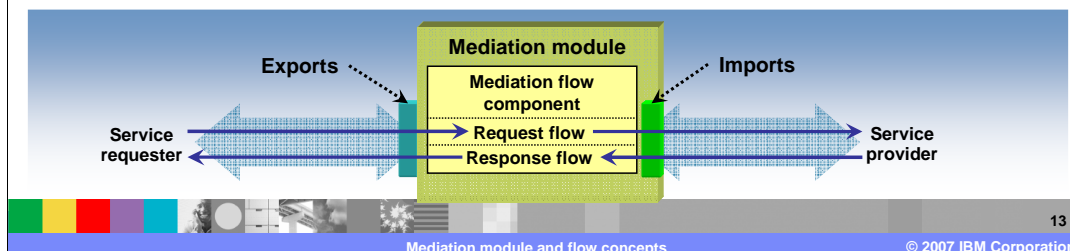
© 2007 IBM Corporation

Once a mediation flow component has been defined on an assembly diagram it has both a source interface and one or more target references. These are each associated with a WSDL interface which defines the operations and their associated inputs, outputs and faults. The first part of defining the implementation of a mediation flow component is to define the operation connections that establish the relationships between source and target operations. There must be an operation connection defined for every operation on the source interface and each source operation can be connected to one or more target operations. Not all target operations need to be part of a connection.

Looking at the graphic, you see that the interface on the left is defined by a WSDL with operations op1, op2 and op3. On the right, the bottom reference is defined by the same WSDL and the upper reference is defined by a WSDL with opA and opB. Using this as an example, you can see an operation connection between op3 and op3 showing a connection between the same operation defined on the same WSDL. There is also an operation connection between op1 and opA showing a connection between different operations of different WSDLs. This connection will require the message be transformed by the mediation. Looking at op2, it is connected to both op2 and opB. This connection will require some type of routing decision and when the target is opB it will also require a message transformation.

Mediation flow component – Request and response flows

- For each operation connection there is a flow
 - ▶ Separated into request flow and response flow
 - One-way operations only have a request flow
 - ▶ Defines the mediation logic for that specific source operation
- The flow logic enables the ESB functions of:
 - ▶ Routing decisions, selection of service provider
 - ▶ Message modification and augmentation
 - ▶ Covert messages to support different interfaces and operations
- Request flow capable of building a response without calling a provider
 - ▶ For example, if the mediation module implemented a cache



13

Mediation module and flow concepts

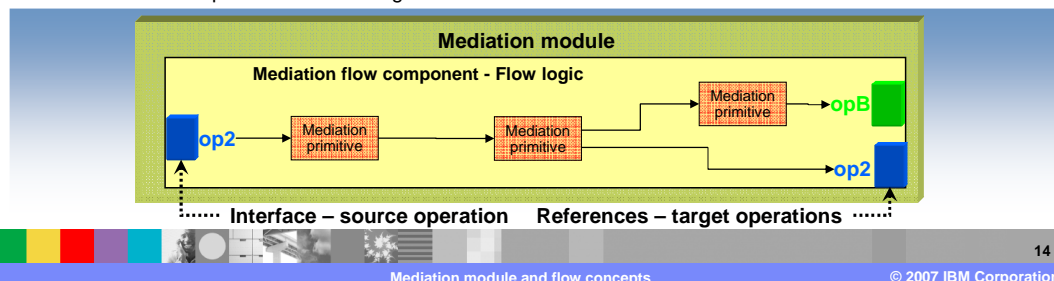
© 2007 IBM Corporation

After defining the operation connections, the next step in building a mediation flow component implementation is to define the flows. Every source operation and its associated operation connections has a flow associated with it. When the operation is a request response operation there are actually two parts of the flow to define, the request flow and the response flow. The logic of the mediation is defined in the flow, providing the ESB functions of dynamic routing and service selection, message modification and augmentation and message transformation.

One of the interesting capabilities of the request flow is to enable the flow to respond directly to the service requester without making a call to any service provider. This might be used for various reasons, such as the mediation module implementing a caching scheme to optimize response time by eliminating certain calls to service providers.

Mediation flow component – Flow logic

- Each request and response flow has its own logic
- Mediation primitives are used to define the processing
 - ▶ Each primitive performs some specific function
 - Interrogate the message
 - Update the message
 - Transform the format of the message
 - Make routing decisions
- Flow of logic defined by wiring
 - ▶ Defines connection between source operations, primitives and target operations
 - ▶ Source operation can be wired directly to the target operation
 - Flow is a pass thru with no logic



14

Mediation module and flow concepts

© 2007 IBM Corporation

Each request flow and response flow has its own logic. The logic is defined using mediation primitives where each primitive performs some function within the flow. A primitive might interrogate the values in the message to do something like write a log message or raise an event. It might update elements within a message, such as filling in input parameter values from a database lookup or the primitive might need to transform the message format because the source and target operations are different. Or the primitive could be involved in making routing decisions based the message content.

In addition to the primitives which each perform a function, the overall logic for the flow is defined by wiring the source operation to primitives, primitives to other primitives and finally primitives to a target operation. The message flows through the wires and is acted upon by the primitives.

When the source and target operation are the same, they can be wired directly together without any primitives being in the flow, defining a mediation flow that is essentially a pass thru operation.

Mediation flow component – Mediation flow editor

- WebSphere Integration Developer provides a mediation flow editor
- The editor is divided into the 3 sections:
 - ▶ Operation connections – top section
 - You define the mapping of each source operation to one or more target operations
 - Once the connection is defined, you select a source operation to display the flow
 - ▶ Message flow – middle section
 - Displays a flow diagram for the selected source operation
 - Tabs let you select between seeing the request flow or the response flow
 - Flow diagram contains a canvas where the logic is defined
 - Canvas contains nodes representing the source and target operations
 - You drag mediation primitives from a palette onto the canvas
 - You define the wiring between the source operation, primitives and the target operations
 - ▶ Properties view – bottom section
 - Displays the properties used to define the element selected in the message flow section
 - Selected elements can be for a mediation primitive, an individual wire, the source and target operations and the overall flow itself

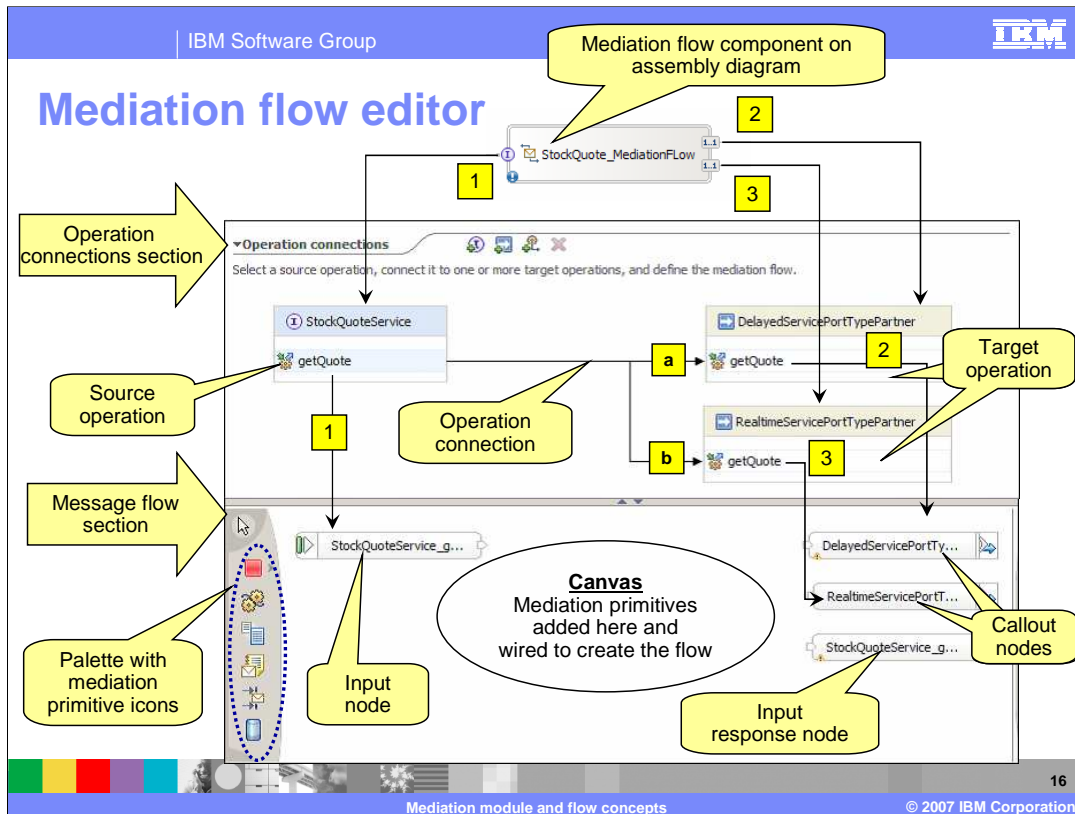
Mediation flow components, which were described on the last several slides, are implemented in WebSphere Integration Developer using the mediation flow editor. There are three major sections in the mediation flow editor, the operation connections section, the message flow section and the properties view.

The operation connections section is located in the top section of the editor. It enables you to define the connections between source operations and target operations. In the editor, selecting the source operation will display the flow for that operation.

The middle section of the editor contains the message flow for the selected operation, with tabs enabling you to switch between the request flow and the response flow. It contains a canvas with nodes representing the source and target operations. You drag mediation primitives from a palette, drop them onto the canvas and wire the nodes and primitives together to define the flow.

The bottom section of the editor has the properties view which is used to display and edit configuration properties. Whatever element is selected in the message flow section will have its properties displayed in the properties view. The elements which can be selected include mediation primitives, a wire, the source or target operations or clicking on the canvas will select the flow itself.

The next few slides provide diagrams of the different sections of the editor.

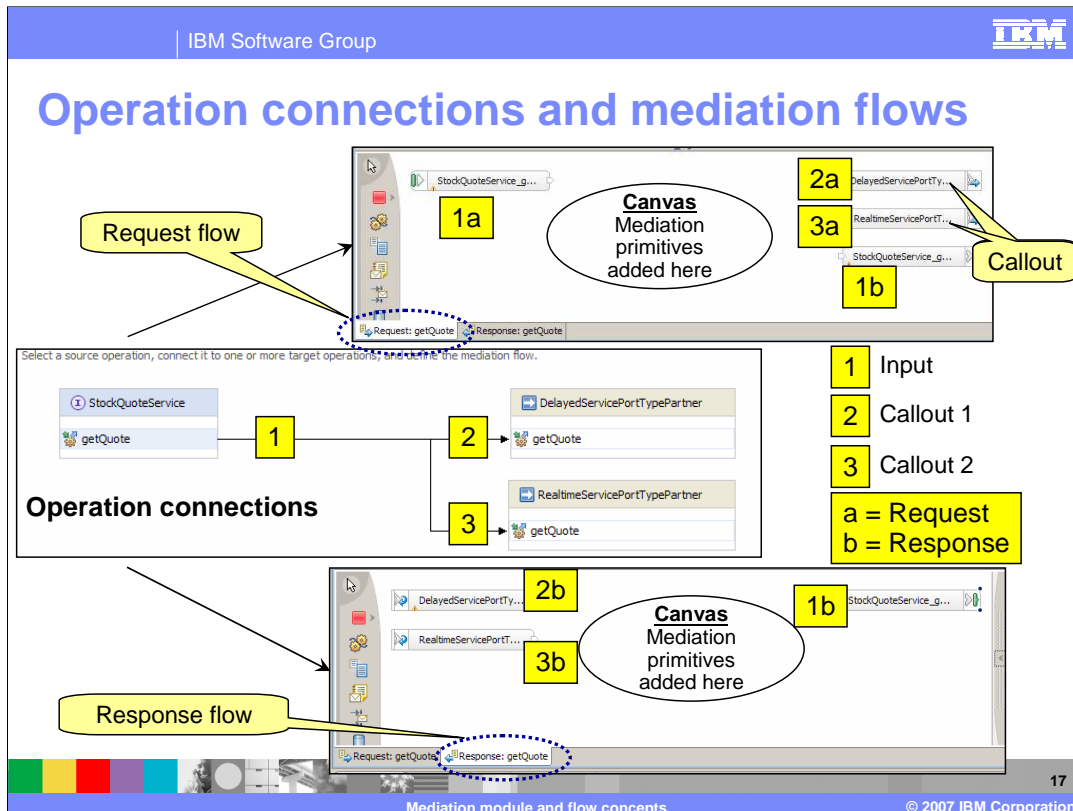


This slide shows how a mediation flow component on an assembly diagram is represented in the mediation flow editor. At the top of the graphic you see a mediation flow component as it would appear in the assembly diagram. It has an interface labeled 1 and two references labeled 2 and 3.

The operations connections section of the mediation flow editor shows the source interface on the left and the two target references on the right. The name of the interface and references are shown in the title bar for each. All the operations defined for each are listed below the title bar, but in this case they all contain only one operation, called getQuote. The lines labeled a and b define the operation connections for the getQuote source operation, showing that its flow might possibly call either of the two getQuote target operations.

The message flow section contains the input node on the left side, representing the message as it is received from the requestor. On the right side are the two callout nodes, representing the message as it is sent to a target service. At the bottom right of the message flow section is the input response node which can be used to respond to the requestor without calling a service provider.

The nodes that were just described are on the canvas. Mediation primitives are dragged from the palette on the left and dropped onto the canvas. Wires are then used to create the flow by connecting the nodes and primitives to perform the mediation function required for the flow.



This page shows more details of the mediation flow editor, relating the operation connections to both the request and response in the message flow section of the editor. The middle of the slide shows the operation connection, with the source input operation being labeled 1 and the target callout operations being labeled 2 and 3.

The top of the slide shows the request flow. Notice that in the lower left corner is a tab used to display the request flow portion of the message flow panel. The label 1a is by the input node where a request flow will start with an incoming request. The labels 2a and 3a are by the callout nodes where the request flow ends with a call to a service provider. Label 1b is by the input response node which can be used to return to the caller without making a call to a provider. The mediation flow logic is built on the canvas by wiring together primitives.

The bottom of the slide shows the response flow. On the lower left is the response flow tab used to display the response portion of the message flow panel. The labels 2b and 3b are by the callout response nodes, where the response flow will begin upon return from the service provider. Label 1b is by the input response node, which is where the response flow ends as it returns to the original requestor. The mediation flow logic for the response is built on the canvas in the same way it is done for the request flow.

Mediation flow component – Service message object (SMO)

- Service message object (SMO) provides a common representation of a message in a mediation flow
- The SMO is a service data object (SDO)
- The SMO contains three major sections
 - ▶ The body is the application data for the message
 - ▶ The headers is protocol specific header information for the message
 - ▶ The context is data used within the flow itself
- An SMO has a schema definition
 - ▶ The headers schema is predefined and the same for all flows
 - ▶ The context schema
 - Mostly predefined and the same for all flows
 - Correlation and transient contexts specified on a per flow basis
 - ▶ The body schema varies by flow and within flow, based on:
 - Interface and operation the message represents
 - Whether application data is the operations input, output or fault data
- The body schema defines the SMOs “message type”

In a mediation flow, the message is represented as a service message object (SMO). It provides a common representation of the message so that mediation primitives are able to access it within a flow. The SMO is built using service data object (SDO) technology and can be accessed using the generic DataObject APIs, type specific SMO APIs and XPath expressions.

The SMO is divided into three major sections, the context, headers and body. The body is the application specific data which is defined by the operation and the input, output or fault data that is being passed for that operation. The headers contain protocol specific header information that is related to the protocol used by the export or import associated with this flow. The context contains flow specific data that is useful within the flow but is not passed outside of the flow.

The SMO has a schema definition for all three sections. The schema for the headers is the same for all flows, but which of the specific headers contain data will depend upon the protocol used by the export or import. The schema for most of the context is the same for all flows except for the correlation and transient sections. The correlation context and transient context are flow specific extensions to the context that are used to pass data within the flow that is required by the flow logic. The body varies for every flow and can also vary within a flow. This is based first on the interface and operation the message represents at that point in the flow and whether the data for the operation is input, output or fault data. These elements of the body of the message define an SMO message type which becomes an important factor when developing flow logic.

Agenda

- Introduction
- Mediation module
- Mediation flow component
- **Mediation primitives**
- Development life cycle and deployment

This section provides more details related to mediation primitives and lists the primitives available for use. There is also an example mediation flow explained.

Mediation primitives

- Mediation primitives perform functions within a flow
 - ▶ Updating or adding data elements to the SMO
 - ▶ Transforming the SMO to a different schema
 - ▶ Making routing decisions
 - ▶ Perform logging or event generation
- Built-in primitives provide a predefined configurable function
- Custom mediation primitives allow you to implement the function in Java
- Mediation primitives have input and output terminals
 - ▶ Terminals are the endpoints for wires in a mediation flow
 - ▶ Primitives have one input terminal and zero, one or more output terminals
 - ▶ A fail terminal is an output terminal taken with the mediation primitive has an exception
 - ▶ Each terminal has a specific message type defining the schema of the SMO at that point in the flow














Mediation primitives are the core building blocks used to process the request and response messages in a flow. They are used to update, add to and transform the SMO, to make routing decisions and perform logging and event generation.

There are built in primitives which perform some predefined function that is configurable through the use of properties. There are also custom mediation primitives which allow you to implement the function in Java.

Mediation primitives have input and output terminals, through which the message flows. The terminals are the endpoints for the wires used to connect primitives and nodes into a flow. Primitives have one input terminal and can have zero, one or more output terminals. There is also a special terminal called the fail terminal through which the message will be propagated when the processing of a primitive results in an exception.

Every terminal on primitives and nodes has a specific message type associated with it. It defines the exact schema for the SMO at that point in the flow. Some primitives are able to change the message type between the input and output terminals, but most do not change the message type.

Mediation primitives – Types

Primitives which read but do not update the SMO elements		
		Write a log message to the configured log database
		Raise a common base event to CEI
		Selectively forward message based on condition
Primitives which can update SMO elements		
		Message elements are set, copied or deleted
		Set elements from contents of a database row
		Set potential endpoints from registry query
Primitives which can modify the SMO message type		
		Update, modify message using XSL transformation
		Read, update, modify message using Java code
Primitives that ignore the SMO		
		Stop single path in flow without an exception
		Stop entire flow and throw an exception

21

Mediation module and flow concepts

© 2007 IBM Corporation

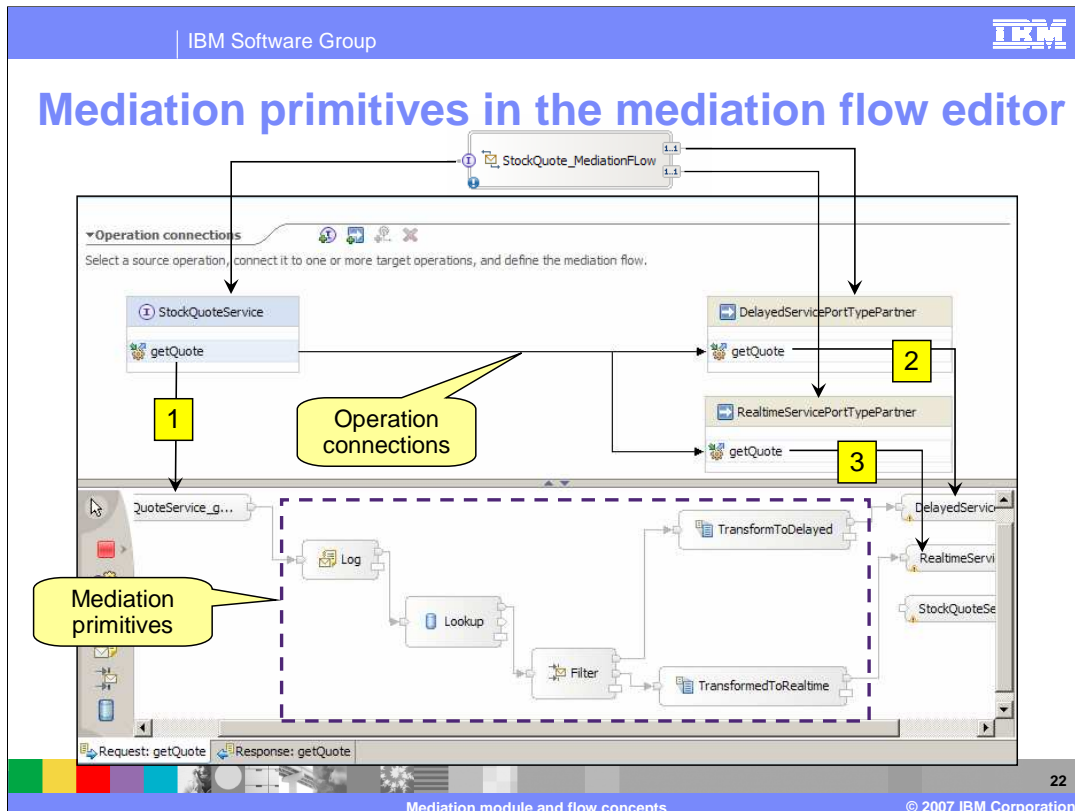
This slide introduces the various mediation primitive types. They are organized according to their behavior and abilities for updating the service message object as it flows through the mediation.

The first group includes those primitives that read from but do not update the SMO. The message logger primitive is used to log all or part of the contents of the message to a message log database which is identified through configuration of the primitive. The event emitter primitive is used to raise an event containing all or part of the contents of the message. The event is a common base event which will be handled by the common event infrastructure. The message filter primitive is used to modify the path through a flow by selectively forwarding the message based on the evaluation of simple XPath expressions. Each expression is associated with an output terminal defining where the message will be forwarded.

The next group contains the primitives that have the ability to update elements of the SMO but do not modify the type of the message. The message element setter primitive can be configured to update elements of the SMO. Individual elements can be set to a specific value or can have their value deleted. Individual elements or sub-trees in the SMO can be set by copying the values from another location in the SMO. The database lookup primitive is used to access information from a database and insert it into the message. A field in the message is used as a key for the database access and selected fields from the resulting database row can be placed into the message. The endpoint lookup primitive is used to perform a query of the WebSphere Service Registry and Repository. The SMO is updated with potential endpoints that can be used for the callout to a service.

The next group includes those primitives that have the ability to modify the message type. The XSLT primitive is used to update or transform messages using XSL transformation. This can be used to change the format of the message. An example of when the format needs to change is when the target provider has a different interface than the incoming message. The custom mediation primitive is used to do any message processing not covered by the other mediation primitives. This is done through Java code that can be written as a visual snippet, a Java snippet or a Java SCA component.

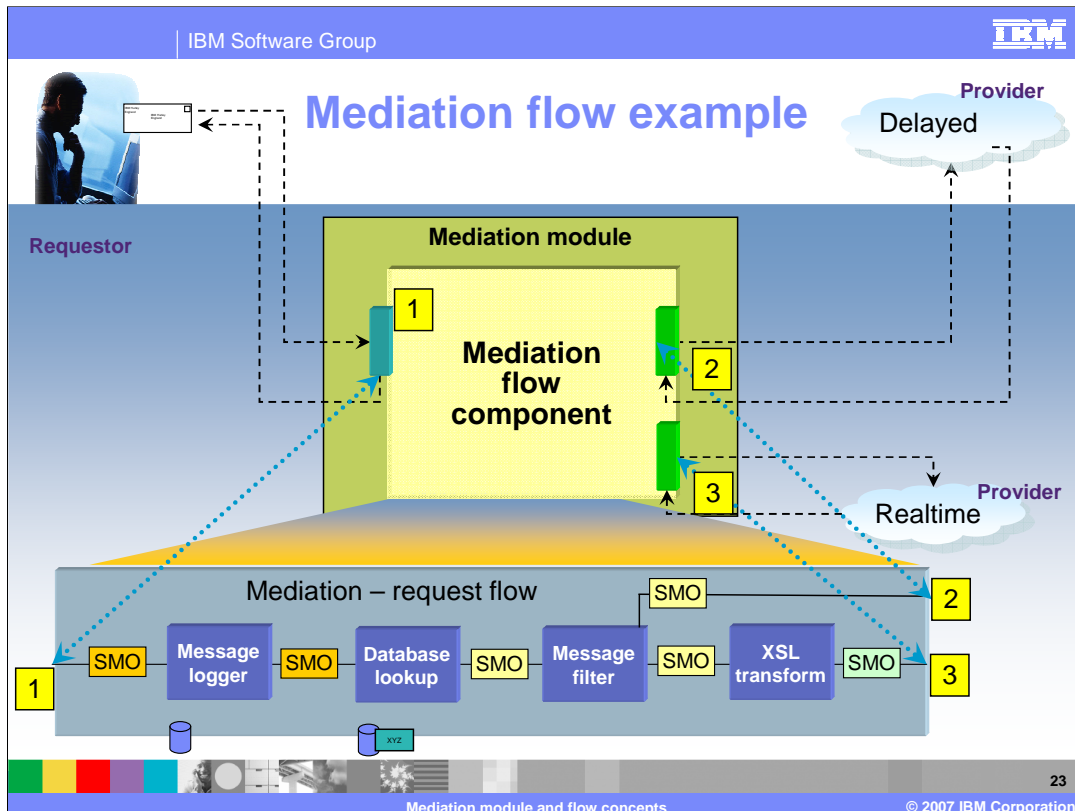
The remaining primitives do not access the SMO. The stop primitive is used to stop an individual path through the mediation flow without raising an exception or affecting other paths through the flow. The fail primitive is used for error conditions and will stop the entire mediation flow and cause an exception to be thrown.



This slide shows a mediation flow component in the assembly editor and in the mediation flow editor, similar to a previous slide. However, in this slide mediation primitives have been added to the request flow. The request flow shown here will be used on the next slide to illustrate the flow of a message. The flow is designed to provide a stock quote, with quotes being supplied by a service returning quotes that are delayed 20 minutes, except for premium customers who receive real time quotes.

The request flow starts at the input node and goes to a message logger primitive that will log the message to the message database. Then a database lookup primitive that determines if the request is for a premium customer, making an update to the transient context to indicate their status. The next primitive is a message filter which looks at the transient context and directs the flow based on the type of customer. Depending upon the result of the message filter, the message request is then transformed to the interface of the delayed quote service or the real time quote service by an XSLT primitive. The appropriate service is then called through one of the callout nodes.

The response flow is not shown in this example, but it would need to contain XSLT primitives to translate the responses to the interface used by the requestor.



This is an illustration of the request flow shown on the previous slide. There is one difference: the delayed quote service uses the same interface as the requestor, so there is one less XSLT primitive in the flow. The color of the boxes labeled SMO is intended to indicate when changes have occurred to the SMO content or message type.

Starting in the upper left, a request for a stock quote is made. It is received by the mediation module through an export, the SMO is constructed and the request flow begins in the mediation flow component as shown on the bottom at the label 1. The SMO is passed from the input node to a message logger which writes the message to a log database. The unchanged SMO is then passed to the database lookup primitive which performs a lookup of the customer in a database to determine if they are a premium customer. This information is placed into the transient context of the SMO and the modified SMO is passed to the message filter primitive. The message filter primitive contains an XPath expression that checks to see if the transient context says this is a premium customer. If not, the unchanged SMO is propagated from the message filter to the callout for the delayed service at label 2. The call to the delayed service is then made through the import labeled 2. If it was a premium customer, the unchanged SMO is propagated from the message filter to the XSLT primitive that modifies the message type of the SMO to be compatible with the interface of the real time service. The transformed SMO is then passed to the callout for the real time service as shown at label 3. The real time service is called using the import labeled 3.

When the service provider returns, the result will be processed through a response flow that is not shown here, and the result is then passed back to the requestor. For the delayed quote service, the response flow would not need to do any processing. However, for the real time quote service there would need to be an XSLT primitive to change the message type of the response message to be compatible with the interface used by the requestor.

Agenda

- Introduction
- Mediation module
- Mediation flow component
- Mediation primitives
- **Development life cycle and deployment**

This section looks at the development, test and deployment life cycle for a mediation module developed in WebSphere Integration Developer and deployed to a WebSphere Process Server or WebSphere Enterprise Service Bus server.

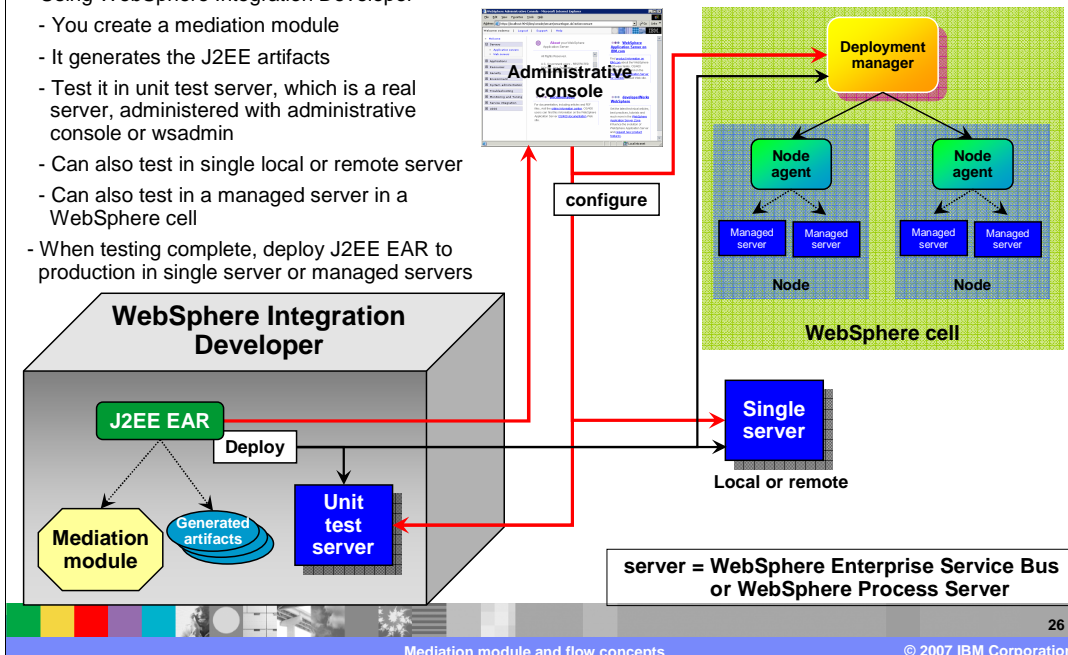
Deployment

- A mediation module is deployed in the same way as an SCA module
- WebSphere Integration Developer generates J2EE artifacts
- A J2EE EAR is generated from the J2EE artifacts using:
 - ▶ WebSphere Integration Developer
 - ▶ serviceDeploy
- Install into the server using standard J2EE EAR installation
 - ▶ Administrative console application installation panels
 - ▶ Using *wsadmin \$AdminApp install* from the command line
- Can be installed into either:
 - ▶ WebSphere Enterprise Service Bus
 - ▶ WebSphere Process Server

The packaging and deployment for a mediation module is the same as that of an SCA module. There are J2EE artifacts generated by WebSphere Integration Developer that are used to build a J2EE EAR that can be deployed to a server. The building of the J2EE EAR can be done by WebSphere Integration Developer or using a command line tool named *serviceDeploy*. The J2EE EAR can then be installed into a WebSphere Enterprise Service Bus or WebSphere Process Server using the normal installation capabilities of the administrative console or the *wsadmin \$AdminApp install* command line tool.

Development, test and deployment life cycle

- Using WebSphere Integration Developer
- You create a mediation module
- It generates the J2EE artifacts
- Test it in unit test server, which is a real server, administered with administrative console or wsadmin
- Can also test in single local or remote server
- Can also test in a managed server in a WebSphere cell
- When testing complete, deploy J2EE EAR to production in single server or managed servers



This diagram shows the end to end development, testing and deployment life cycle of a mediation module. Using WebSphere Integration Developer, you construct your mediation module with SCA imports and exports, a mediation flow component and possibly some Java components.

As you are developing your mediation module, WebSphere Integration Developer will be generating J2EE artifacts.

When you are ready to unit test your module, WebSphere Integration Developer generates a J2EE EAR and installs it into the unit test server that is installed with WebSphere Integration Developer. This can be either a WebSphere Enterprise Service Bus or WebSphere Process Server. The unit test server is a fully functional server and is administered using the administrative console or wsadmin commands. However, while unit testing the mediation module you interact with WebSphere Integration Developer to deploy the mediation application and to refresh it in the server as you make iterative changes.

In addition to the unit test servers, WebSphere Integration Developer enables you to configure a stand-alone WebSphere Enterprise Service Bus or WebSphere Process Server to use for unit testing. A possible reason for doing this would be to do your unit testing on a server running on a different operating system than the development tools, such as an AIX system.

You can also do your testing using a managed server which belongs to a WebSphere cell that is managed by a deployment manager.

Once you have finished with your development and unit testing you may want to continue your testing without the WebSphere Integration Developer. Using WebSphere Integration Developer you can export a J2EE EAR containing your mediation application.

The EAR file can then be installed into a stand-alone or managed server using the administrative console or wsadmin command line tools. You would do this in your quality assurance test environment and your production environment.

Summary

- ESB mediation capabilities are provided by both:
 - ▶ WebSphere Enterprise Service Bus
 - ▶ WebSphere Process Server
- ESB capabilities are enabled by mediation modules
 - ▶ Follow the SCA module of imports, exports and components
 - ▶ Provide loosely coupled connectivity between service requestor and provider
 - Protocol transformation
 - Dynamic message routing
 - Message augmentation and transformation
- Mediation modules contain a mediation flow component
 - ▶ Contains the flow logic for service operation requests and responses
 - ▶ Built using mediation primitives that act on the message in the flow
 - ▶ The message is represented as a service message object (SMO)

In summary, enterprise service bus functionality can be deployed into either WebSphere Enterprise Service Bus or WebSphere Process Server.

The ESB capabilities are provided using a special type of SCA module called the mediation module. Using the mediation module, a loose coupling of service requestors and providers can be obtained. This includes enabling protocol transformations, dynamic routing of messages and the augmentation and transformation of messages.

The mediation module contains a mediation flow component. The mediation flow logic is defined in this component through the use of mediation primitives. The message that flows through the mediation flow component is a type of SDO called a service message object.

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](#)



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

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

J2EE, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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.

Information is provided "AS IS" without warranty of any kind. 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 NON-INFRINGEMENT. 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.

© Copyright International Business Machines Corporation 2007. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.