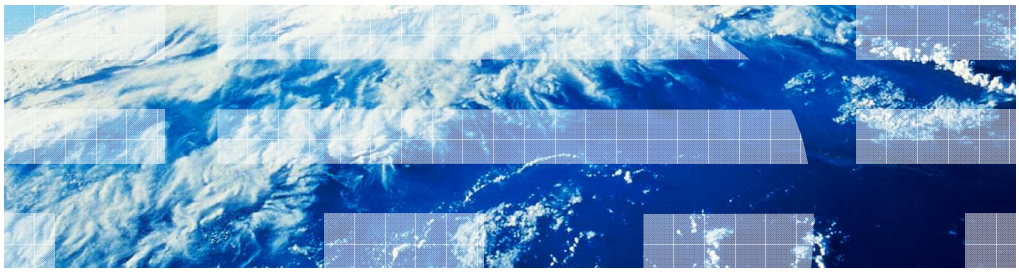
WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Mediation module and flow concepts



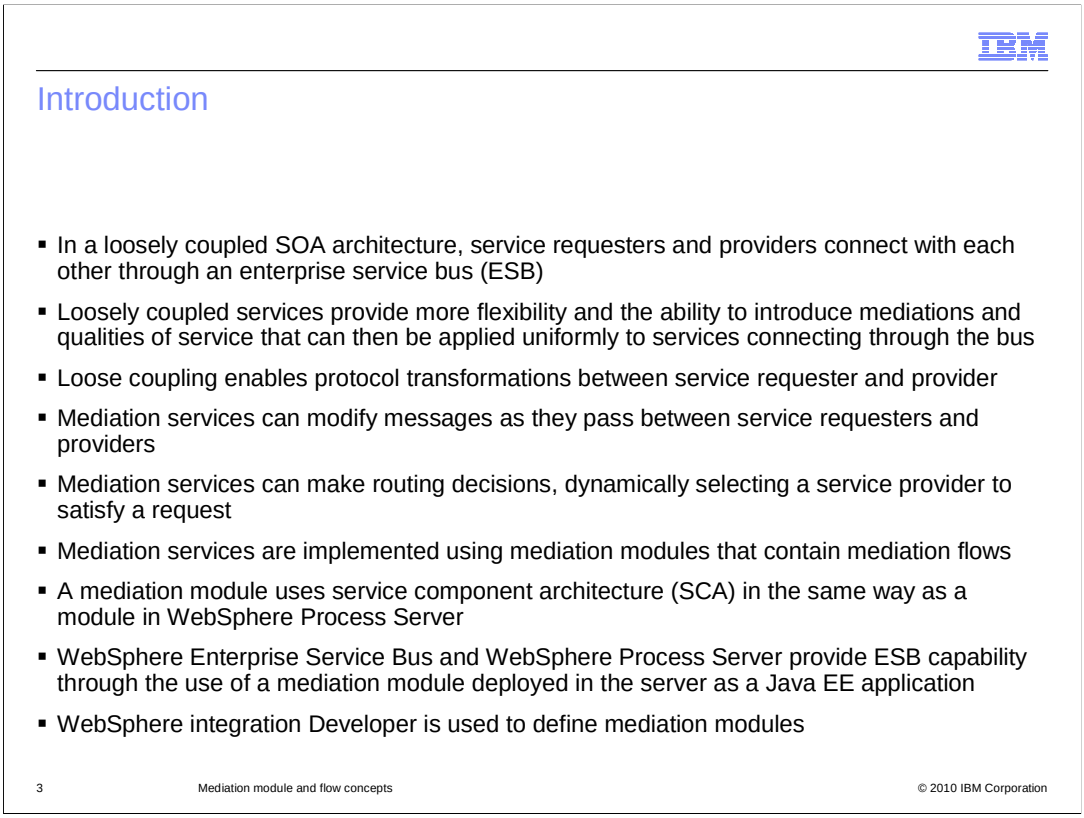
WebSphere. software

This presentation provides an overview of the WebSphere Enterprise Service Bus product, introducing mediation module and mediation flow concepts. It also includes information about WebSphere Integration Developer, which is used to define mediation modules. The material is also applicable to WebSphere Process Sever, which provides the same mediation capabilities as WebSphere Enterprise Service Bus.

Agenda - Introduction

- Introduction
- Mediation module
- Mediation flow component
- Service message object
- Mediation primitives
- Mediation flow component editor

The agenda of the presentation is shown here. The next slide provides 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. The role of service message objects is the presented, followed by an introduction to mediation primitives. The presentation ends with a section that introduces to the mediation flow component editor provided by WebSphere Integration Developer, including links to demonstrations of the editor.



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 requesters 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 requester 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 requesters and providers. The mediation flow component makes use of mediation primitives to define the logic of a mediation flow.

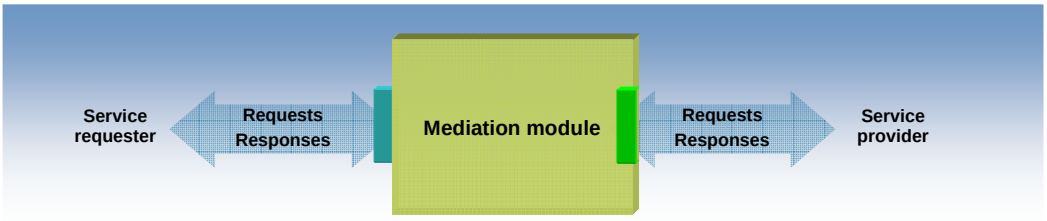
WebSphere Integration Developer is used to define mediation modules, which are deployed to WebSphere Enterprise Service Bus and WebSphere Process Server as Java EE applications.

Section

Mediation module

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

Mediation module - Concepts



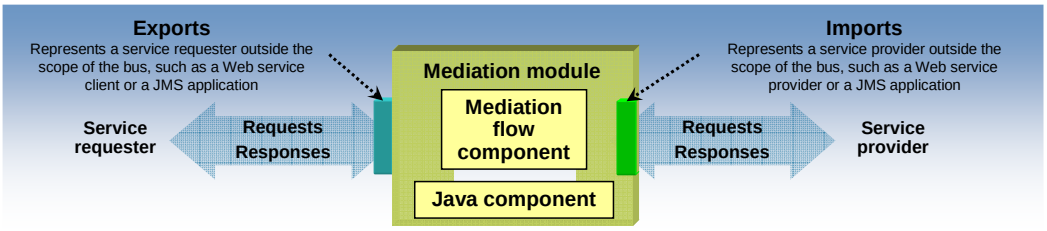
- WebSphere Enterprise Service Bus and WebSphere Process Server provide mediation modules, that:
  - Intercept and modify messages between service requester and the service provider
  - Provide the ESB functions of converting protocols, routing, transformation and other custom processing on the messages
- 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

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 requester 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. The graphic in the slide illustrates the mediation module. The blue box on the left represents SCA exports through which service requests are received. The green box on the right represents SCA imports through which service providers are called.

The mediation module represents the unit of deployment that is generated into a Java EE 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 requesters and providers
  - Interface defined by WSDL only
- Exports
  - Expose the mediation module to external service requesters
  - 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 service invoke primitives or to support the 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.

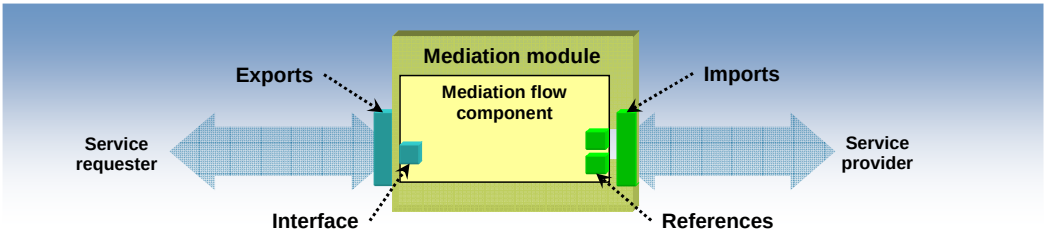
The heart of a mediation module is the mediation flow component, which provides the mediation logic applied to a message as it flows from service requester to a provider. It is an SCA component that is normally used in a mediation module. The interface for input and output of a mediation flow component 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 requesters. They provide the interface and protocol definition used by the requester 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 the service invoke primitive when the Java component in the assembly contains the logic for the service being called. The second reason is for mapping between WSDL and Java interfaces.

Mediation module - Mediation flow component



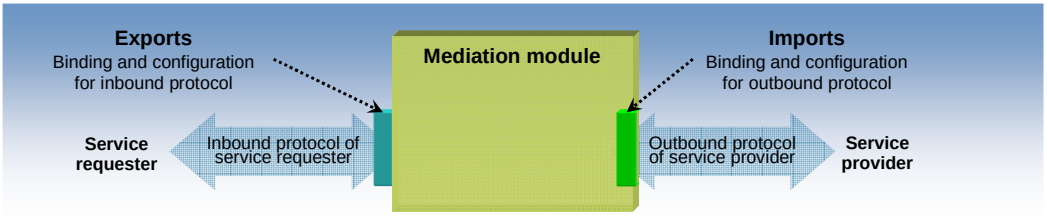
- Mediation flow component is a type of SCA component
  - Normally used in a mediation module, but can also be used in a module
  - A module or mediation module can contain more than one mediation flow component
  - Source defined by an interface
    - Can be more than one interface, but typically is only one
  - 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 requester and provider

The mediation flow component is a type of SCA component that is typically used in a mediation module, but it can also be used in a module. You can have more than one mediation flow components in a module or mediation module.

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. Having more than one interface is possible, but in most cases, a mediation flow component only has one interface. The target references are also described using WSDL and must match the WSDL definitions of the imports or Java components to which they are wired. Having more than one reference is not uncommon.

The mediation flow component 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 requester and provider.

Mediation module - Imports and exports



- Imports and exports are configured with bindings
  - Bindings provide support for varying protocols
    - For example, Web service or JMS
  - Bindings enable the ESB function of protocol transformation between requester and providers
  - Bindings are configured at development time
    - Binding type is fixed for the import or export at this time
  - Bindings can also be administered at runtime to vary some configuration settings

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 requester and mediation module and between mediation module and provider. The ESB functionality of protocol transformation is enabled through the use of an export and import in the case where they each have a different binding type.

Bindings are associated with an import or export at development time using WebSphere Integration Developer, at which time the binding type is set. The configuration data, which varies by binding type, is also set at development time. However, bindings support runtime modification of some of the binding configuration data by

using the Administrative console or the wsadmin commands.

There are several different binding types, or protocols, that are supported. These are listed on the next couple of slides.

Mediation module – Import and export bindings

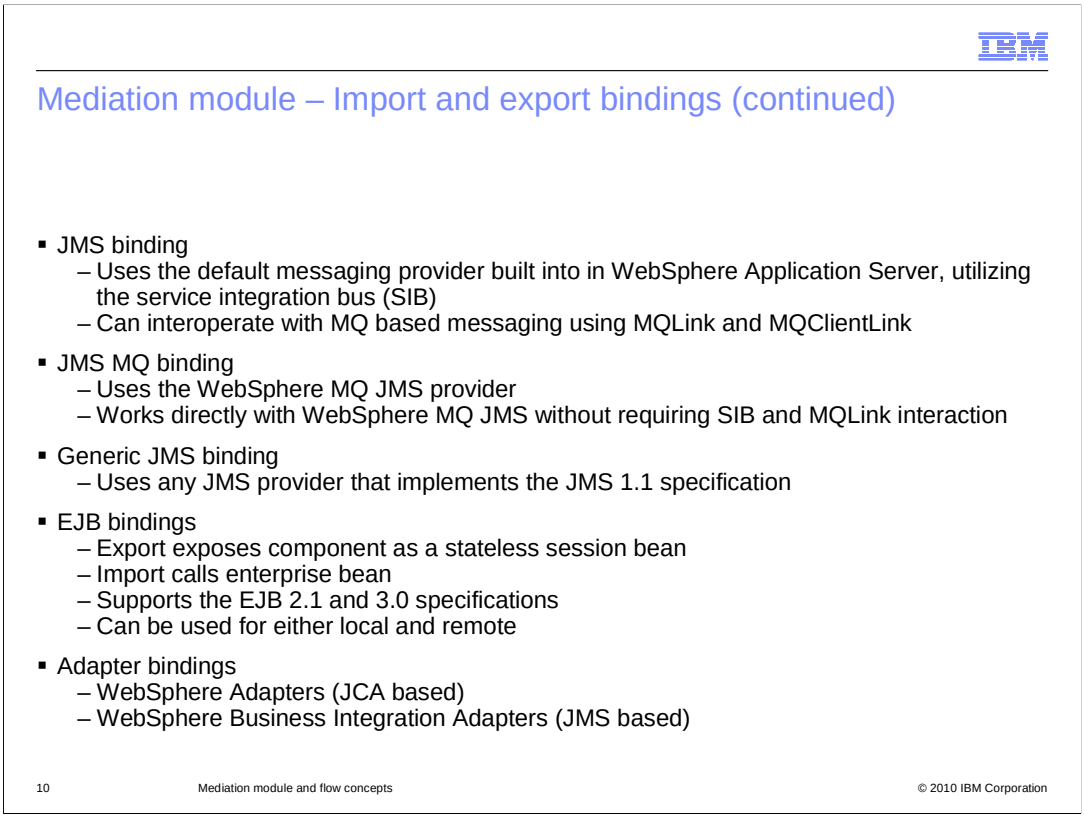
- SCA (default) binding
  - Used for communication between SCA modules
- Web services binding
  - JAX-RPC with SOAP 1.1 over HTTP or JMS
  - JAX-WS with SOAP 1.1 or 1.2 over HTTP
- HTTP binding
  - HTTP/HTTPS
  - Payload does not have to be SOAP
  - Access to headers
- MQ binding
  - Enables interaction 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

The first binding type is the SCA binding, which allows imports to communicate with exports in a different SCA module or mediation module using SCA as the transport. These are often referred to as the SCA default bindings.

The Web services binding supports either JAX-RPC or JAX-WS. The JAX-RPC bindings use SOAP 1.1 over either HTTP or JMS, while the JAX-WS bindings use either SOAP 1.1 or 1.2 over HTTP.

The HTTP binding supports any HTTP or HTTPS interaction. They differ from the Web services binding in that the payload of the HTTP message does not have to be SOAP and the binding provides access to HTTP headers.

There are a few different variations of bindings that support messaging protocols, the first of which is shown on this slide. The MQ binding enables interaction with MQ based applications that are using native MQ protocols rather than JMS protocols. These applications typically make use of data in the headers and follow MQ header handling conventions, all of which is exposed in this binding type. The various request and reply correlation schemes used in MQ are also supported.



The next three bindings are also messaging bindings and all support JMS protocols.

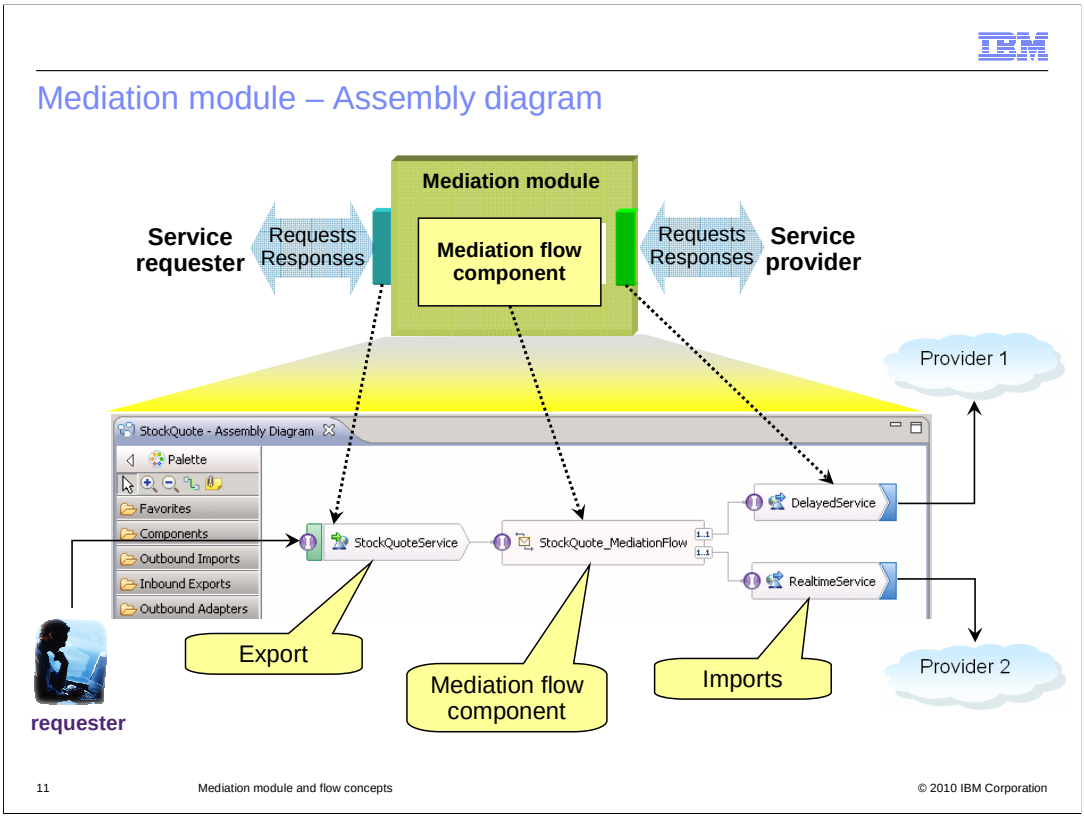
The JMS binding makes use of the default messaging provider that is built-in to WebSphere Application Server and uses the service integration bus (SIB). These are used for JMS communication with other applications using the default messaging provider. They can also communicate with MQ applications through configuration of an MQLink or MQClientLink in the SIB.

The JMS MQ binding makes 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 generic JMS binding allows interaction with other JMS providers. It is based on the JMS 1.1 specification, and therefore the provider used must be compliant with JMS 1.1. An example of such a provider is SonicMQ, but there are many others as well.

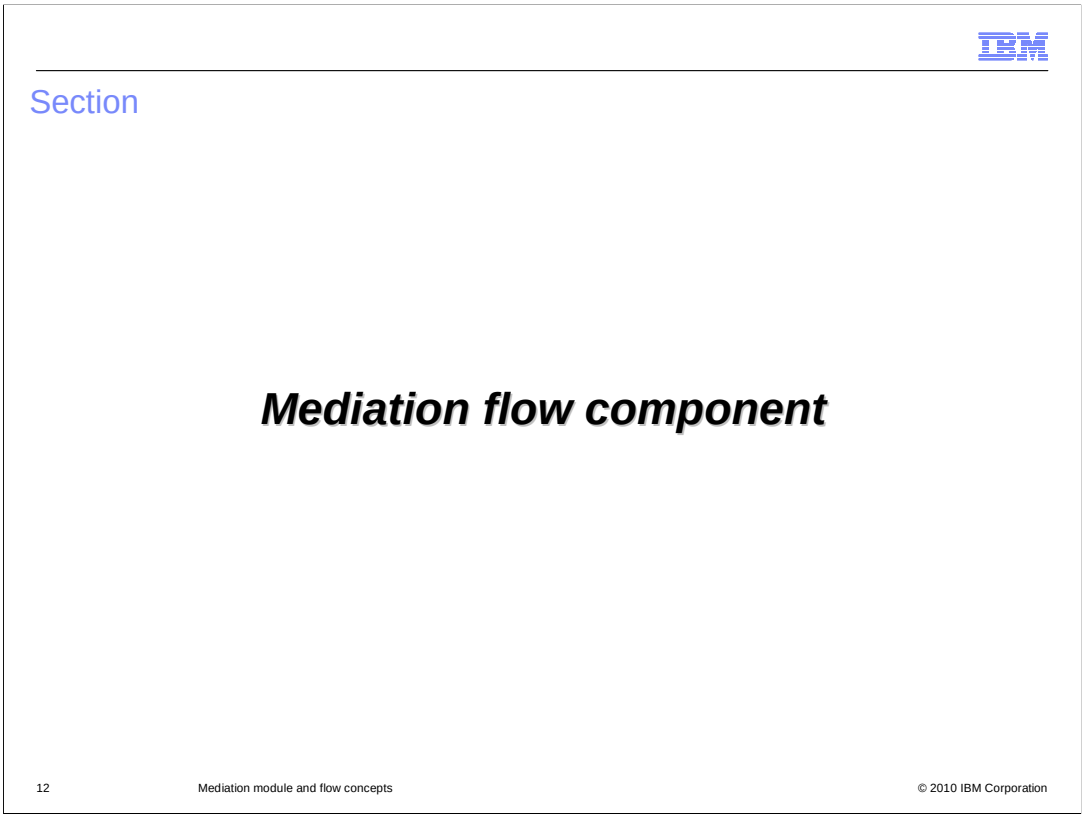
EJB bindings provide a way to expose an SCA component as a stateless session bean and to call an enterprise bean from an SCA component. There is support for both the EJB 2.1 and EJB 3.0 specifications. The support works using either local or remote EJB interfaces.

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.



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.

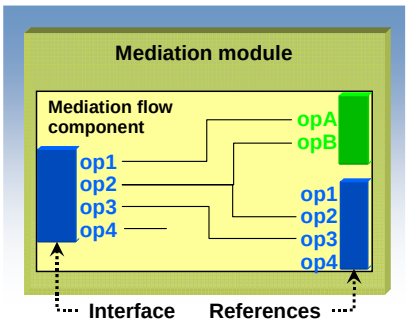
In addition to introducing the assembly diagram, the graphic on this slide shows you how to relate the diagrams that are used in this presentation to what you will see in WebSphere Integration Developer.



This section provides the next level of details about the mediation flow component. It discusses the request and response flows and basics of flow logic.

Mediation flow component – Basic structure

- The interface and references define the source and target operations of the mediation
- There should be a mediation flow defined for every source operation
- A mediation flow can have zero, one or more target operations
- A target operation can be used in more than one mediation flow
- A target operation might not be used by any mediation flow
- The mediation flow for a source operation can:
  - Have target with the same operation
    - •Reference with the same WSDL
  - Have target with a different operation
    - •Reference with a different WSDL
    - •Reference with same WSDL
  - Have more than one target operation
    - •Enables flow to make routing decisions
  - Have no target operation
    - •Flow directly responds to request

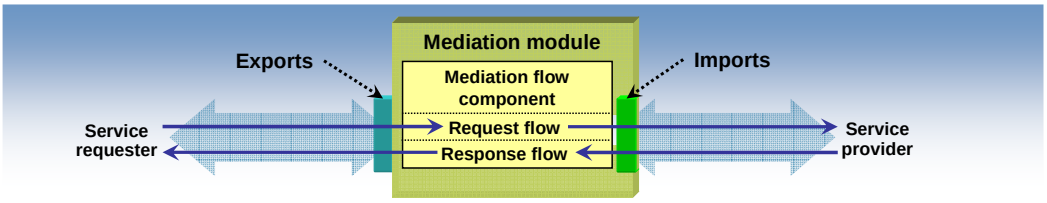


13 Mediation module and flow concepts

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. For the mediation flow component implementation to be complete, there should be a mediation flow implemented for every source operation. The mediation flow for a source operation can include the use of one or more target operations, and can also be implemented without including the use of a target operation. A target operation might be included in the mediation flows for more than one source operation or might not be included in any mediation flows.

Looking at the graphic, you see that the interface on the left is defined by a WSDL with operations op1, op2, op3 and op4. 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 a mediation flow between op3 and op3 showing a connection between the same operation defined on the same WSDL. There is also an mediation flow between op1 and opA showing a connection between different operations of different WSDLs. This connection will require the message be transformed by the mediation to the schema required by opA. Not shown on the graphic is the possibility that a mediation flow can be between two different operations defined in the same WSDL. Looking at op2, its mediation flow includes both op2 and opB. This flow will require some type of routing decision to determine which to use, and when the target is opB, it will also require a message transformation. Finally, op4 shows the case where the mediation flow contains logic to respond to the message without including the use of any target operation.

Mediation flow component – Request and response flows

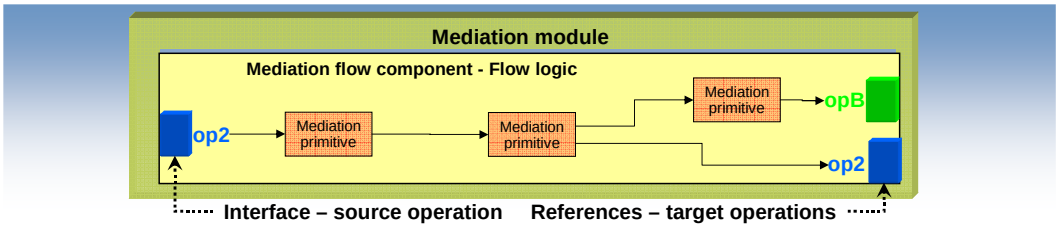


- For each source operation there is a flow
  - Separated into request flow and response flow
    - One-way operations only have a request flow
  - Defines the mediation logic for that specific source operation
- The flow logic enables the ESB functions of:
  - Routing decisions, selection of service provider
  - Message modification and augmentation
  - Convert 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

Every source operation of the mediation flow component has a mediation 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. A one way operation will only have a request 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 redundant 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 logic path between source operations, primitives and target operations
  - Source operation can be wired directly to the target operation
    - Flow is a pass through with no logic

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 such as 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 can be involved in making routing decisions based on the message content, registry lookup or other criteria.

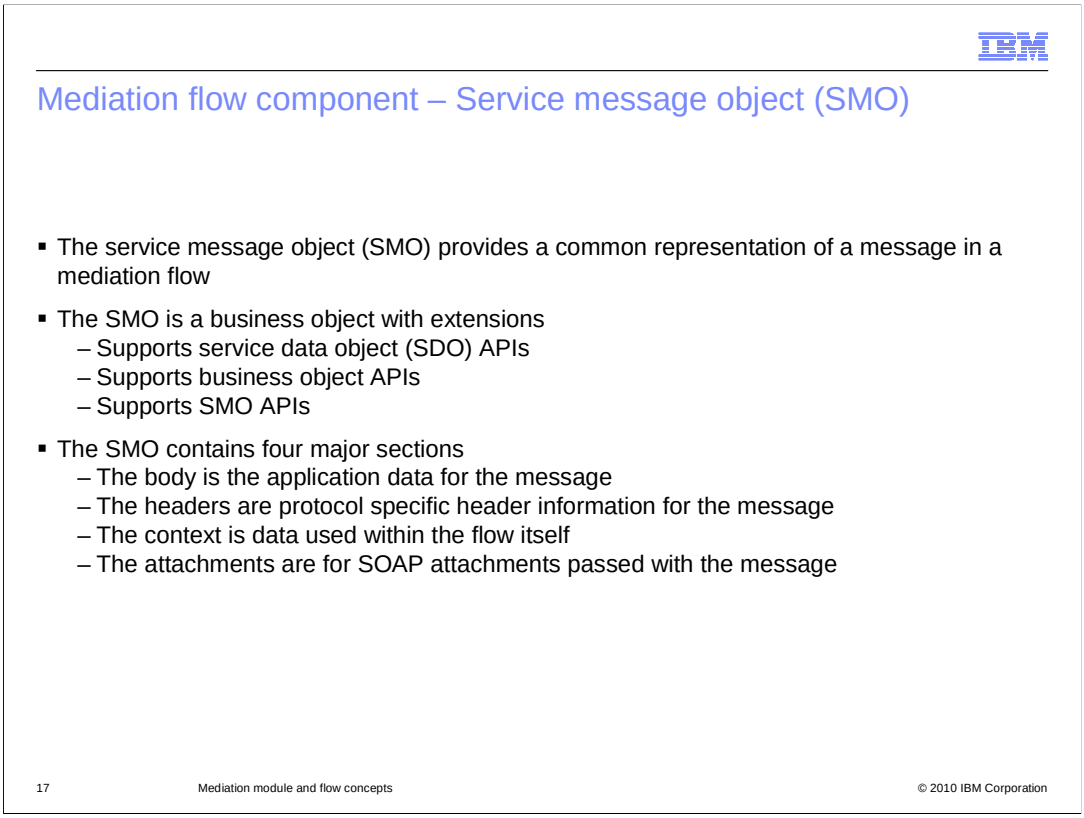
In addition to the primitives, the overall logic for the flow is defined by wiring the source operation to primitives, primitives to other primitives and finally primitives to a target operation. The message flows through the wires and is acted upon by the primitives.

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 through operation.

Section

Service message object (SMO)

This section is a short introduction to the service message object (SMO). Other presentations provide more detailed information about the SMO.



In a mediation flow, the message is represented as a service message object (SMO). The SMO provides a common representation of the message, enabling mediation primitives to use a defined schema for accessing the message within the flow. The SMO is a business object with extensions. As a business object, it is built using service data object (SDO) technology and can be accessed using the generic DataObject APIs and the business object APIs. In addition, there are SMO specific APIs which are aware of the SMO schema definition.

The SMO is divided into four major sections, the context, headers, body and attachments. 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 attachments section contains any SOAP attachments that are associated with the body of the message.

Mediation flow component – Service message object (SMO)

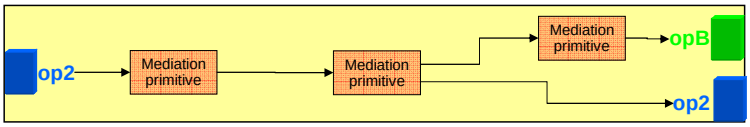
- An SMO has a schema definition
  - The headers schema is predefined and the same for all flows
  - The context schema
    - Mostly predefined and the same for all flows
    - Correlation, transient and shared contexts specified on a per flow basis
  - The body schema varies by flow and within flow, based on:
    - Interface and operation the message represents
    - Whether application data is the operation's input, output or fault data
  - The attachments are an array with a predefined schema definition
- The body schema defines the SMOs "message type"

The SMO has a schema definition for all four 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. For most of the context, the schema is the same for all flows except for the correlation, transient and shared context sections. The correlation, transient and shared contexts 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 of the message 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. Finally, the attachments section is an array with a fixed schema definition.

The schema definition for the body of the message defines the SMO message type. This is an important factor when developing the logic of a mediation flow.

This section introduces mediation primitives and how they are classified into groups.

Mediation primitives

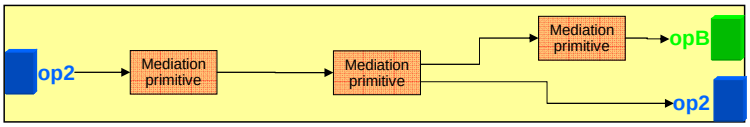


- Mediation primitives perform functions within a flow
  - Updating or adding data elements to the SMO
  - Transforming the SMO to a different schema
  - Flow control for iteration or routing decisions
  - Perform logging or event generation
- Built-in primitives provide a predefined configurable function
- Custom mediation primitives allow you to implement the function in Java
- Subflows allow you to create reusable logic that is used similar to a primitive=

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 control the flow for iteration, to make routing decisions, to perform logging and event generation.

Most of the primitives are built-in primitives which perform some predefined function that is configurable through the use of properties. There is also a custom mediation primitive which allows you to implement the function in Java. You can also define a subflow similar to how you define a flow, and then use that subflow as if it were a primitive.

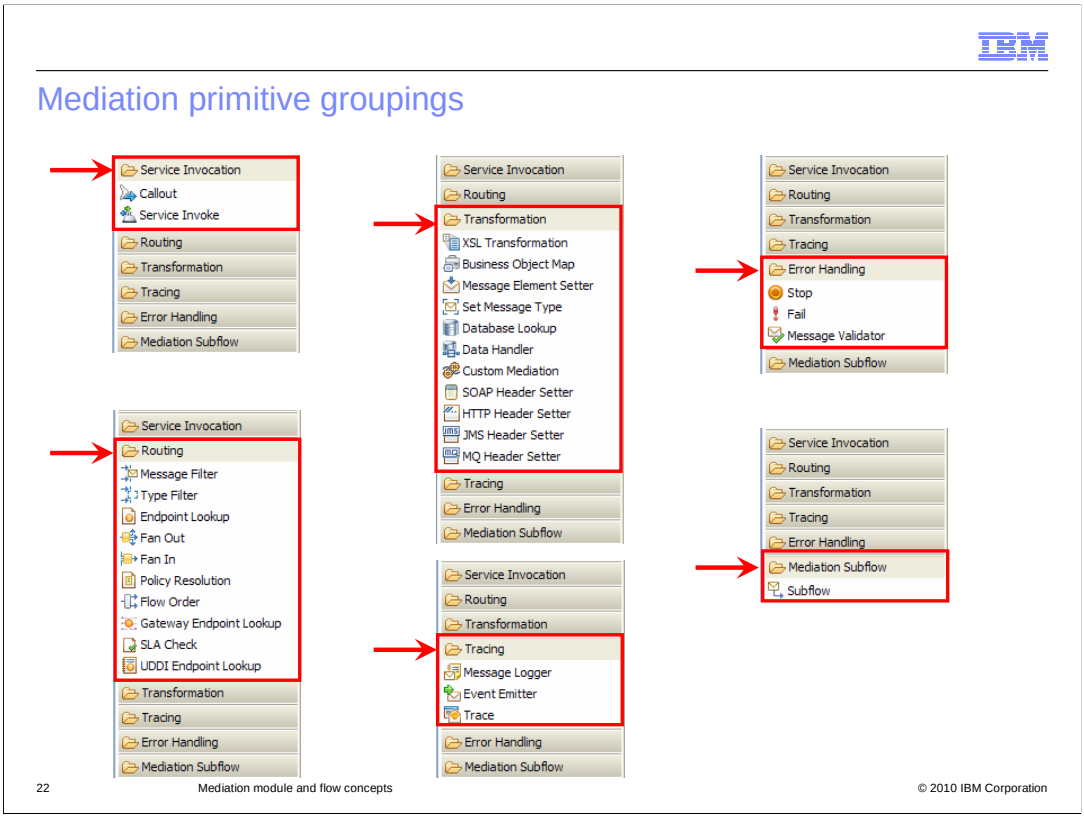
Mediation primitives



- Mediation primitives have input and output terminals
  - Terminals are the endpoints for wires in a mediation flow
  - Primitives have one or more input terminals and zero, one or more output terminals
  - A fail terminal is an output terminal taken when the mediation primitive has an exception
- Most terminals have a specific message type
  - Message type defines the schema of the SMO at that point in the flow
  - A few primitives can change the message type between input and output
  - Primitives not dependent upon message type can have a terminal defined as “any” type

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. Almost all primitives have only one input terminal, but multiple input terminals are possible for some primitive types. Primitives can have zero, one or more output terminals. There is also a special terminal called the fail terminal through which the message is propagated when the processing of a primitive results in an exception.

Most terminals on primitives and nodes have a specific message type associated with it. The message type defines the exact schema for the SMO at that point in the flow. Some primitives are able to change the message schema, therefore changing the message type between the input and output terminals. However, most primitives do not change the message type. For some primitives, the terminal type can be defined to be an “any” type. This is allowed for primitives that do not have dependencies on the message type.



This slide introduces the various mediation primitive groupings. There are various ways that mediation primitives can be organized or classified. In WebSphere Integration Developer, they are categorized into broad groupings based on the primary function of the primitive.

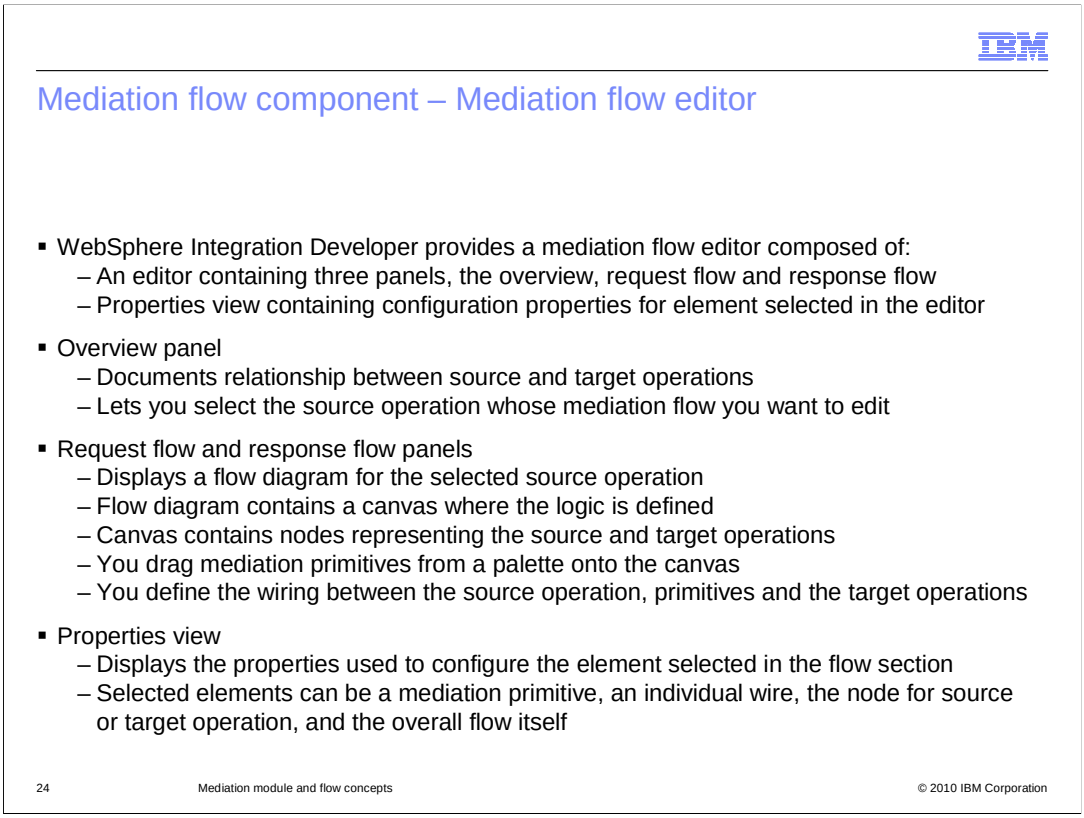
These groupings are shown using screen captures from the mediation flow editor palette. In the upper left, you see the service invocation primitives which are used to interface with external services. Below them are the routing primitives. They are used to either control the path within a flow or identify endpoints for external services. In the center are the transformation primitives which are used to access and modify data in the SMO. Some can also manipulate the schema of the SMO, thus changing the message type. Below them are the tracing primitives, which write data extracted from the SMO to an external source such as a log or event queue. In the upper right are the error handling primitives that are used to check for errors or control the flow in error situations. Finally, below them is the mediation subflow. A subflow can be thought of as a user defined primitive. They are used to encapsulate the logic of a flow, enabling it to be reused in other flows as if it were a primitive.

Other ways that can be used to organize or classify primitives and details about the individual primitives are addressed in the mediation primitive presentations.

Section

Mediation flow component editor

This section introduces the mediation flow component editor provided by WebSphere Integration Developer.



Mediation flow components are implemented in WebSphere Integration Developer using the mediation flow editor. The editor is contained in a window with three panels, the overview, the request flow and the response flow. In another window is the properties view. This is an integral part of the editor, containing configuration information for whatever element is selected in the request flow or response flow.

The overview panel in the editor shows all the source operations on the left and target operations on the right. There are wires connecting a source operation to each of the target operations that are used in the source operations mediation flow. These are provided for documentation and are not editable on this panel. The other thing you use this panel for is to select a source operation whose mediation flow you want to edit.

The mediation flow for the selected source operation is found in the request and response panels. Each panel contains a canvas starting with a node representing the source operation, a palette for selecting target nodes and primitives, and a section containing some overall configuration information for the flow. Nodes for targets are dropped onto the canvas and associated with specific target operations. You also drag mediation primitives from a palette, dropping them unto the canvas. You wire together the source operation node to primitives, primitives to other primitives, and primitives to the target operation node. This provides the flow logic.

Below the window for the editor is a window which contains the properties view. which is used to display and edit configuration properties. Whatever element is selected in the mediation 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 operation nodes or clicking on the canvas will select the flow itself.

Mediation flow editor demonstrations

- Introduction to the mediation flow editor

- Use of nodes and callouts in a mediation flow

This slide contains links to demonstrations of the mediation flow editor. The first one presents an introduction to the editor. It shows the editor's layout, basic navigation within the editor and describes the relationships between the various parts of the editor. The second demonstration presents an overview of the nodes that are the starting and ending points of the flow. It describes the steps needed to add callout nodes to a flow and the relationships between the nodes and the inbound and outbound messages of the flow.

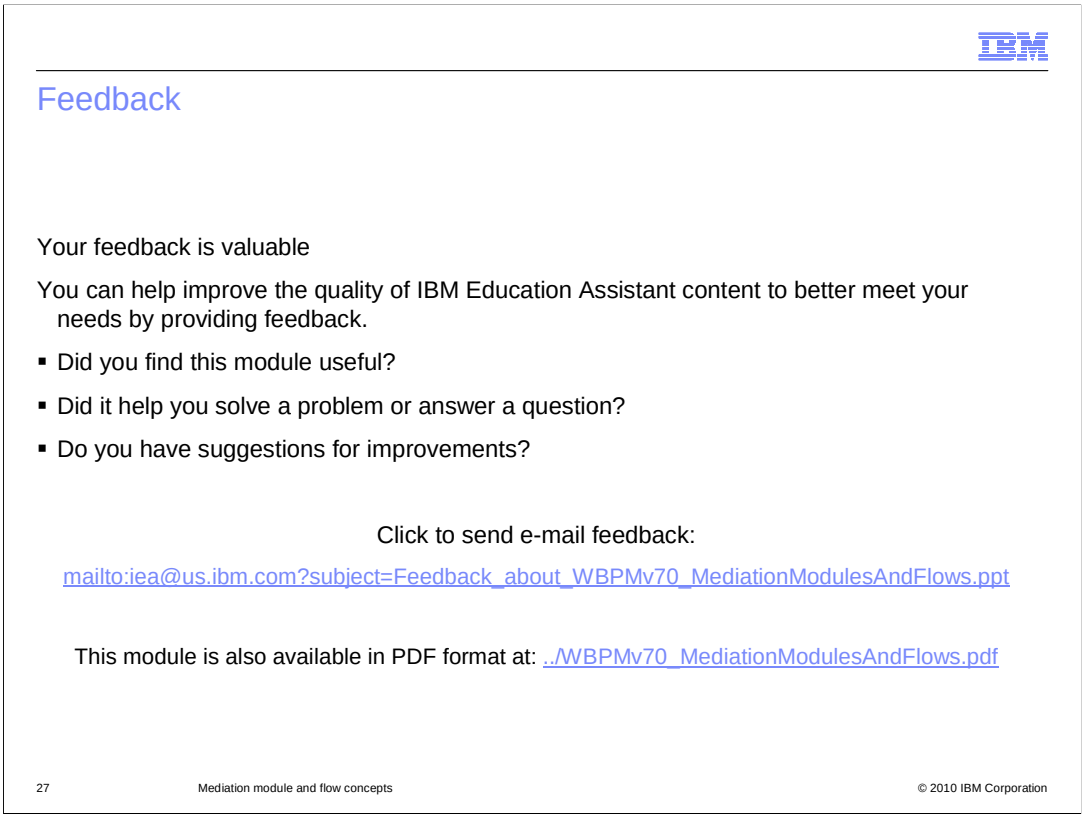
Summary

- ESB mediation capabilities are provided by both:
  - WebSphere Enterprise Service Bus
  - WebSphere Process Server
- ESB capabilities are enabled by mediation modules
  - Similar to SCA modules with imports, exports and components
  - Provide loosely coupled connectivity between service requester 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 requesters 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 business object called a service message object.



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

Trademarks, disclaimer, and copyright information