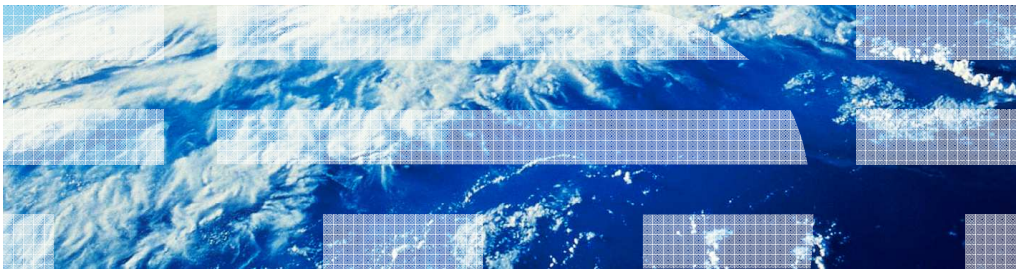
WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

WebSphere Enterprise Service Bus overview



WebSphere software

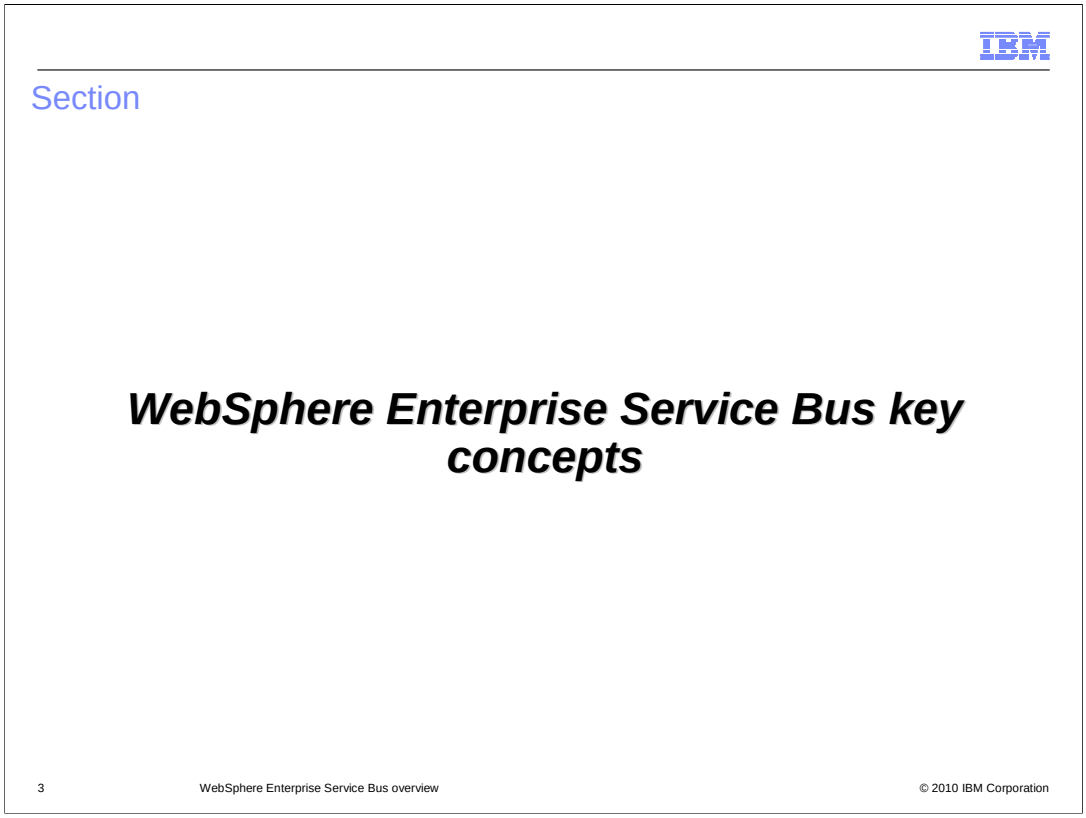
This presentation provides an overview of the WebSphere Enterprise Service Bus product.

Goals

- Introduce the WebSphere Enterprise Service Bus
  - What are the key concepts
  - Relationship to the WebSphere family
  - User roles
- Helpful prior knowledge:
  - Basic understanding of Enterprise Service Bus concepts
  - Basic understanding of Service Component Architecture

The goal of this presentation is to introduce you to the WebSphere Enterprise Service Bus product. You will learn the key concepts needed to understand the approach that is taken to implement mediation functionality. Then you will see how the WebSphere Enterprise Service Bus fits into the product stack of the WebSphere family. Finally, you will learn about the primary user roles that are associated with development and administration for WebSphere Enterprise Service Bus.

This presentation assumes you have at least a cursory knowledge of what an enterprise service bus is and understand the basic concepts of service component architecture, commonly referred to as SCA.



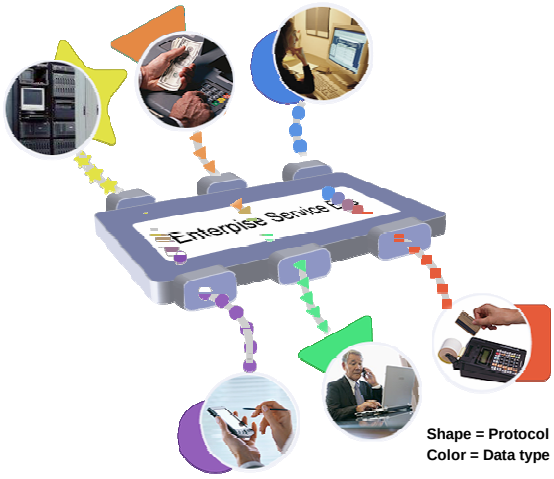
In this section you are given a quick reminder of the concepts associated with an enterprise service bus. With that background, the key concepts of how WebSphere Enterprise Service Bus implements ESB functionality is presented.

Enterprise service bus – Key functions

An enterprise service bus (ESB) is a flexible connectivity infrastructure for integrating applications and services

An ESB performs the following between requestor and service

- **ROUTING messages between services**
- **CONVERTING transport protocols between requestor and service**
- **TRANSFORMING message formats between requestor and service**
- **HANDLING business events from disparate sources**



This slide provides a very good view on the functions of an ESB. An enterprise service bus provides a flexible connectivity infrastructure for integrating applications and services, enabling composite applications to be built as a loose coupling of independent services. It is at the heart of your service oriented architecture, reducing the number, size, and complexity of interfaces and connections that must be defined and maintained.

There are four primary functions provided by an enterprise service bus:

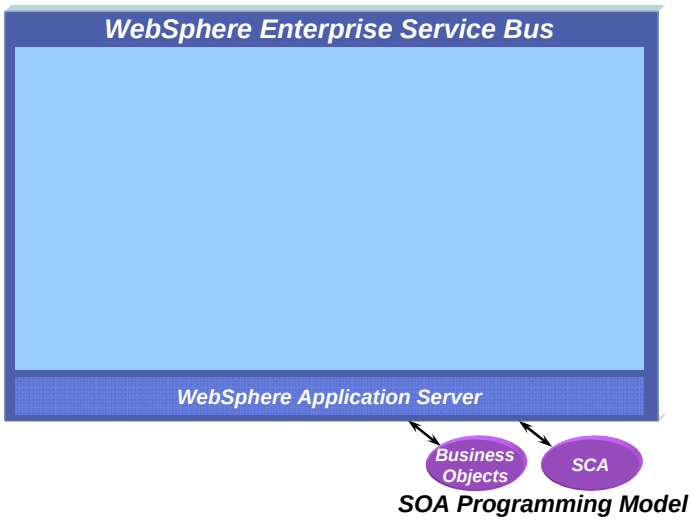
Its first responsibility is the routing of messages. Rather than the service requestor calling directly to the service provider, the requestor sends the request to the ESB, and the ESB then is responsible for making the call on the service provider.

Secondly, it is responsible for converting transport protocols. When a service requestor called directly to the service provider, they need to use the same transport protocol. The ESB enables the service requestor to use one transport protocol while the service provider uses another.

Thirdly, it is responsible for transforming message formats. By eliminating the direct call from the service requestor to the service provider, the ESB is capable of modifying the message so that the interfaces used by the requestor and provider do not have to be identical. In these cases of message transformation, the interfaces are semantically the same yet syntactically different.

Finally, the ESB is capable of handling business events from disparate sources. Therefore, the same service provider responsible for performing some particular business function can be indirectly invoked from a variety of application contexts.

WebSphere Enterprise Service Bus (1 of 8)



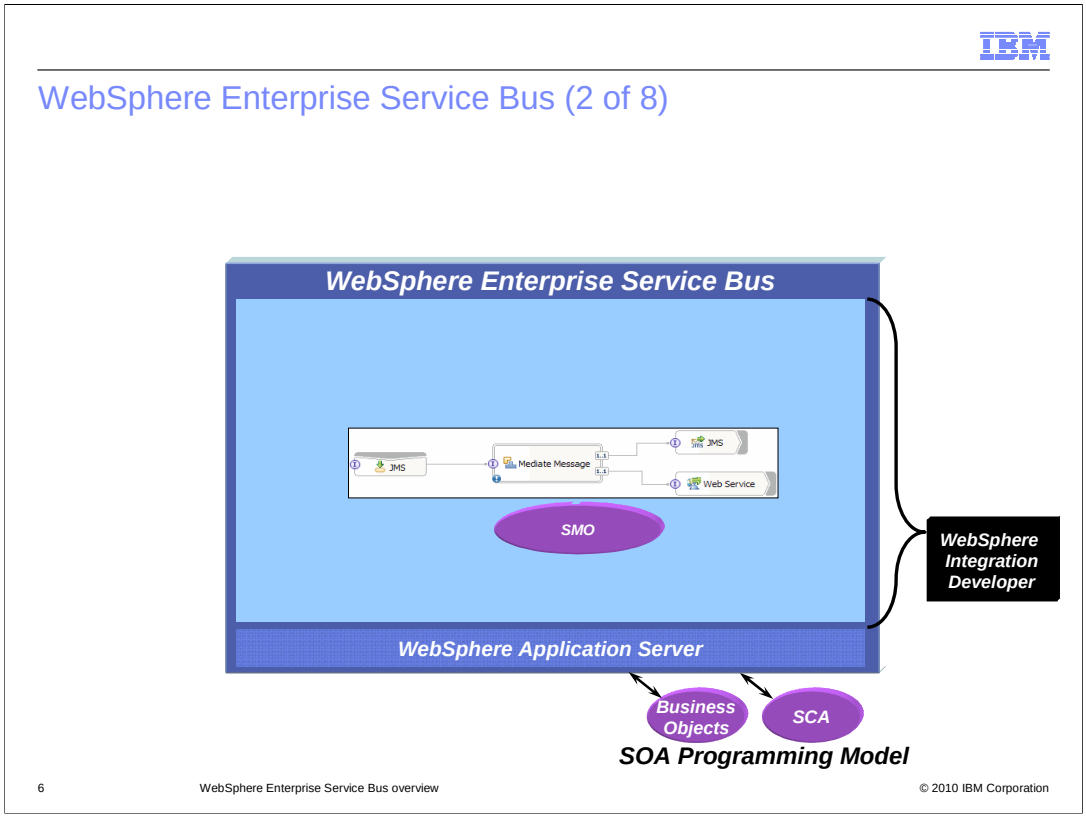
This is the first of a series of diagrams which provides a fairly complete overview of WebSphere Enterprise Service Bus. The diagrams reveal the basic building blocks used by WebSphere Enterprise Service Bus to implement ESB functionality.

First, notice that WebSphere Enterprise Service Bus is built on top of WebSphere Application Server. More detail on this aspect is provided in a subsequent section of this presentation.

The next thing to notice is that WebSphere Enterprise Service Bus is built around a service oriented architecture (SOA) programming model. The key components of the SOA programming model are service component architecture (SCA) and business objects.

SCA specifications have been evolving over the past several years. The SCA implementation in WebSphere Enterprise Service Bus is based on the original SCA specification, published in 2005, with extensions. The implementation is not fully compliant with any of the SCA specifications currently published. However, it is interoperable with the Open SCA implementation provided by WebSphere Application Server V7.0 Feature Pack for Service Component Architecture.

The business object programming model is similar to the programming model defined by the service data object (SDO) specifications. The business object programming model implemented in WebSphere Enterprise Service Bus is based on an early SDO specification with extensions. Extensions to the early specification were needed to enable fundamental requirements that were not fully defined in the specification. Over time, the SDO specifications have improved to address some of these requirements. However, the WebSphere Enterprise Service Bus implementation is not fully compliant with any of the published specifications.

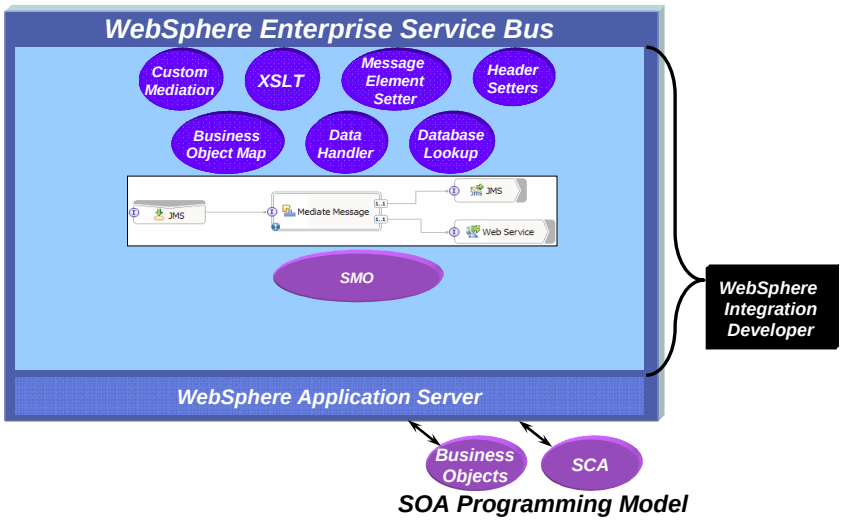


In the center of the diagram you will see an SCA assembly diagram which is used to define an SCA module. It contains an SCA export, an SCA component and two SCA imports. An SCA export is used to receive service requests, an SCA import is used to make requests to service providers and an SCA component contains processing logic. In this case, it is a mediation flow component, the primary SCA component type for providing ESB functionality. It encapsulates the logic needed for message transformation and routing decisions. Protocol conversion happens based on the bindings that are configured for the SCA exports and imports.

You can also see WebSphere Integration Developer on the right. It is the tool used to develop SCA based mediation applications that run in the WebSphere Enterprise Service Bus. It is through the assembly editor in WebSphere Integration Developer that you define the assembly containing the SCA imports, exports and components.

Below the assembly diagram is shown the service message object, commonly referred to as the SMO. The SMO is a representation of the message passing through the bus. It contains a message body, which is the application data associated with the service request. It also contains headers with information relevant to the transport protocol, for example, JMS properties associated with an inbound JMS message. It also contains context information which is data required by the internals of the flow logic.

WebSphere Enterprise Service Bus (3 of 8)



7 WebSphere Enterprise Service Bus overview

Added to the diagram you can now see a few of the mediation primitives. Mediation primitives are the primary entities involved in the definition of mediation logic within a mediation flow component. The processing performed by the mediation primitives is centered on the SMO. Primitives can access and update the data in the SMO and can modify the format of the SMO. The resulting content and format of the SMO defines what the outgoing message will be.

Mediation primitives come in several types, each performing some specific predefined function. An instance of a mediation primitive in a flow is customized through the use of configuration properties. This defines exactly what the predefined function will do in this instance.

Introduced on this slide are those mediation primitives whose primary purpose is to access and modify the content of the SMO. Some can also change the SMO format.

The custom mediation primitive enables you to drop into Java code to access and manipulate the SMO. It is the most flexible of the primitives, enabling the implementation of functionality which is not provided by the other mediation primitives.

The XSL transformation primitive performs an extensible stylesheet language transformation on the SMO. This capability enables the message to be modified so that the service requestor and service provider do not have to support the identical interface.

The business object map primitive enables the use of business object maps within a mediation flow. It is similar to the XSL transformation in that it enables the message to be modified so that the service requestor and service provider do not have to support the identical interface.

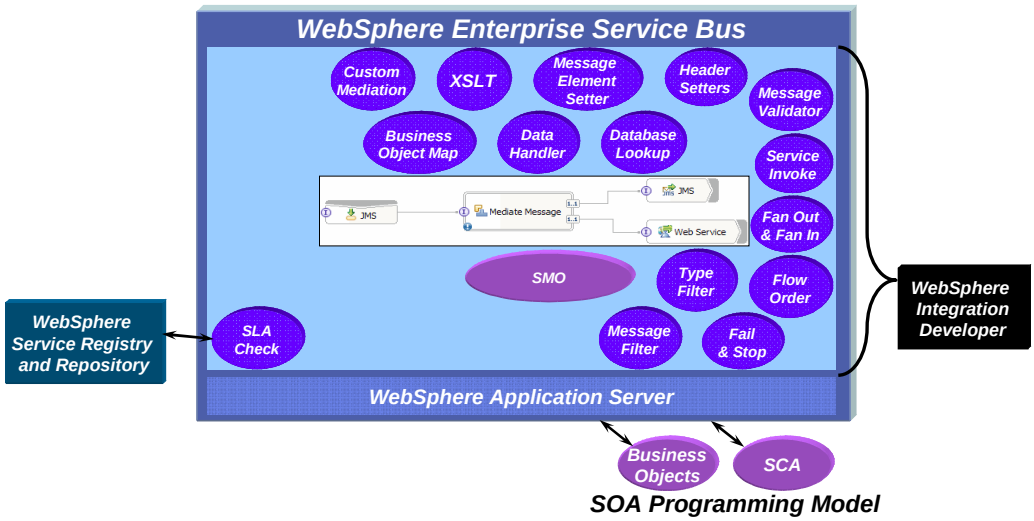
The data handler primitive allows you to configure a data handler, similar to one used with an SCA import or export. Data handlers convert between business object format and native data formats used by the transport protocols.

The message element setter primitive allows elements in the SMO to be set to a fixed value or to values copied from another part of the SMO.

The database lookup primitive enables you to use an element value from the SMO as a key for a database lookup. Values from the database which are returned from the lookup can then be used to update the SMO.

The header setter primitives are four protocol specific primitives used for accessing and manipulating the headers within the SMO. There are header setter primitives for JMS, HTTP, MQ and SOAP.

WebSphere Enterprise Service Bus (4 of 8)



Several more primitives are now added to the diagram. Generally speaking, these primitives influence the path taken through a mediation flow by providing some form of flow of control logic.

The message filter and type filter primitives enable you to control the flow through the mediation based on what is contained in the SMO. The message filter controls the flow based on element values and the type filter controls the flow based on element types.

The fail and stop primitives provide a mechanism to terminate a path of execution through the mediation flow. The fail primitive raises an exception and terminates the entire flow. The stop primitive terminates the current path, allowing other paths to continue.

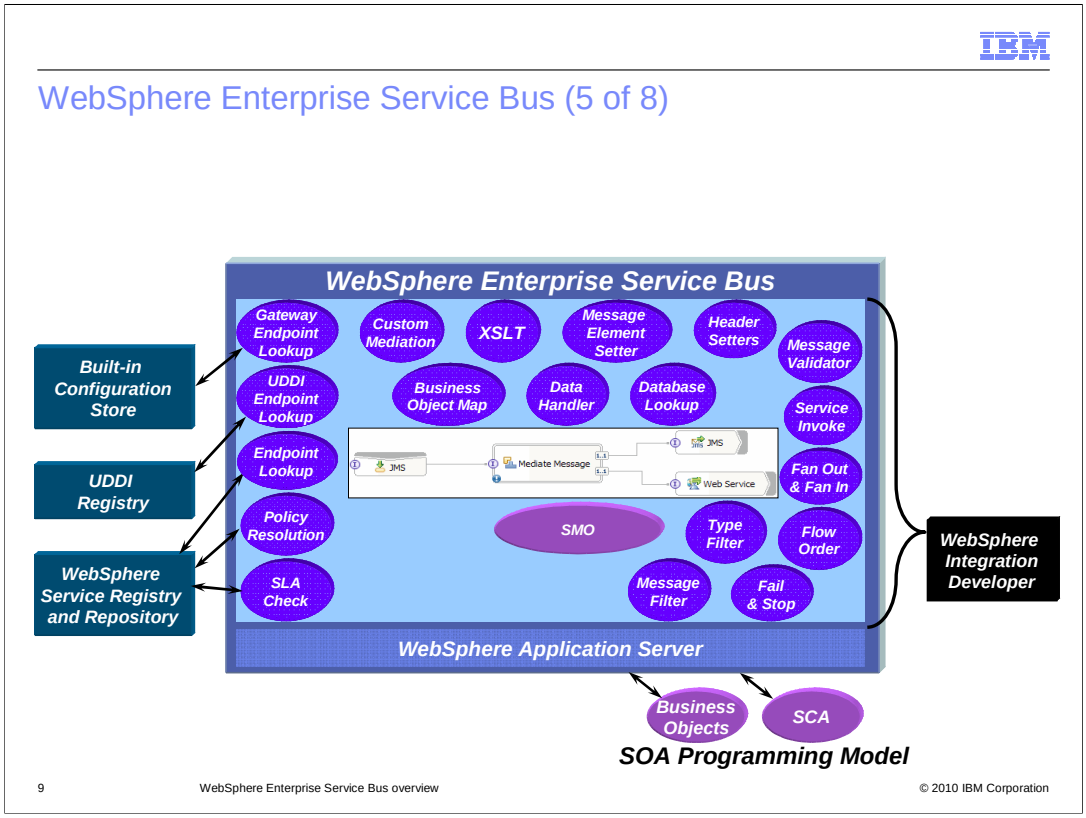
Flow order provides a mechanism to control the order in which multiple paths are given control when those paths start from the same point in the flow.

The fan out and fan in primitives are used in conjunction with each other to provide aggregation support within mediation flows. The fan out enables iterating through the individual elements of an array within the message. The fan in is used to aggregate the results of each iteration. If you consider the fan out primitive to be the beginning of a loop, the fan in primitive effectively is the end of the loop.

Service invoke is a primitive which enables the calling of an external service from within a mediation flow. Following the invoke, different paths are subsequently taken through the flow, depending upon whether the service returns normally or returns a fault.

The message validator provides a way to check the actual SMO contents against the schema definition for the SMO. Different paths are taken based on whether the SMO content conforms to the schema definition or not.

The SLA check primitive provides a mechanism to incorporate service level agreements into mediation flows. The primitive makes a call to the WebSphere Service Registry and Repository which contains configuration information for service level agreements. Different paths are then taken through the flow based on the response from the registry which indicates whether the request conforms to the configured service level agreements or not.



The diagram now contains mediation primitives that update the SMO based on lookups from registries. The data in the updated SMO then influences subsequent behavior of the mediation.

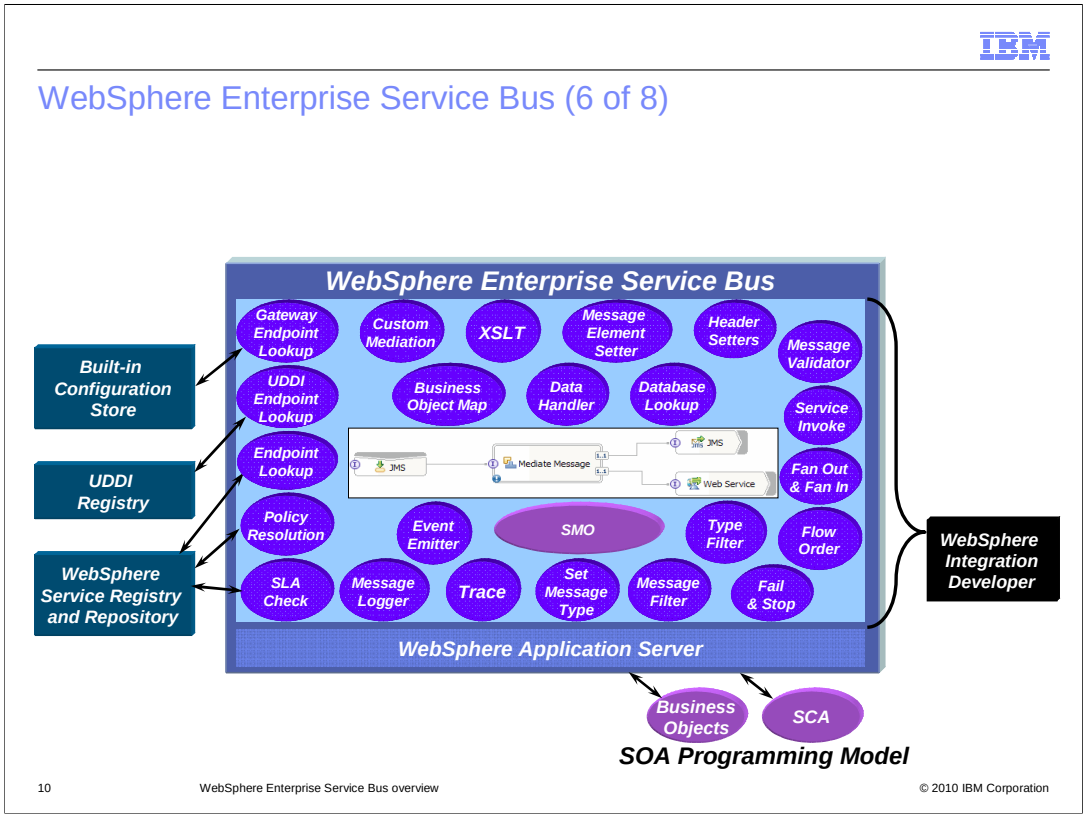
Three of the primitives have to do with resolution of endpoint addresses which are then used to control dynamic invocation capabilities. Dynamic invocation applies to calling services from within a flow using the service invoke primitive or calling the target provider of the mediation using a callout. The primitives update the target address fields in the SMO based on information returned from the registry. Those target address fields are then used by a service invoke or a callout to invoke an operation on a designated endpoint.

The first of these primitives is the endpoint lookup. It interacts with the WebSphere Service Registry and Repository to find appropriate endpoints.

The UDDI endpoint lookup primitive interacts with a UDDI registry to discover endpoints.

The gateway endpoint lookup is used in conjunction with a configuration store that is built into WebSphere Enterprise Service Bus. This primitive is used as part of a proxy gateway pattern. The proxy gateway is a specialized type of mediation flow that can be constructed using a wizard provided by the patterns explorer in WebSphere Integration Developer. The built-in configuration store is administered through business space widgets provided with the WebSphere Enterprise Service Bus.

Finally, the policy resolution primitive interacts with WebSphere Service Registry and Repository to discover policy information. Policies can be associated with the mediation module or with an endpoint that is contained in the target address field of the SMO. The primitive places the returned policy data into the SMO. Subsequent mediation primitives in the flow can have properties designated to have their values overridden by the policy data. In this way, policies can be used to modify values used in the flow or can be used to influence the flow of control.



The set of primitives shown is now complete, with the addition of primitives that do not update the SMO or affect the flow of control.

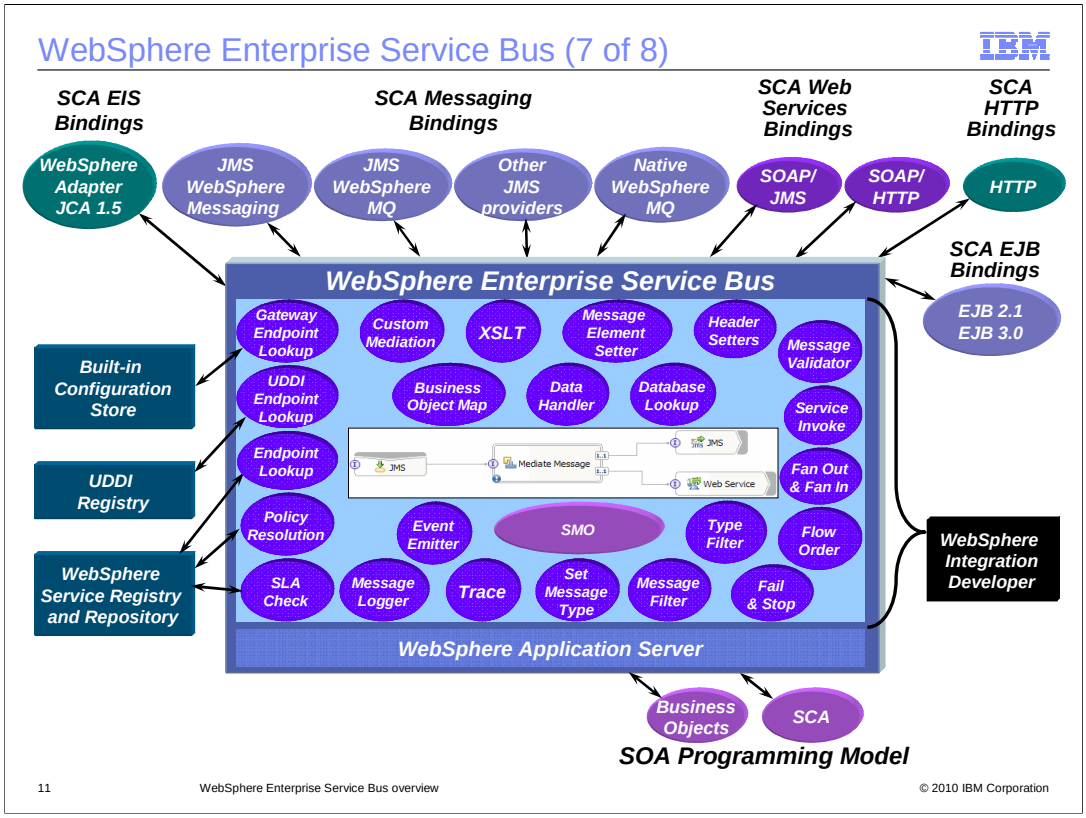
The first of these is the set message type. This primitive is used to further qualify the schema of the SMO by setting more specific type information for loosely typed fields in the SMO. This enables subsequent primitives in the flow to access the field based on the more specific type. This can be compared with the function of a cast operation in a programming language.

The last three primitives provide ways to write some or all of the contents of the SMO to an external location.

The trace primitive enables the writing of SMO data to the system log or to a designated log file. The intended usage of the trace primitive is to help during development of mediation flows.

The message logger primitive enables the writing of SMO data to a message log database. It can also be configured to provide the SMO data to a custom Java logging implementation, which allows the data to be used in whatever way the custom logging implementation chooses. This primitive can be useful in either development or production environments.

Finally, the event emitter primitive writes the SMO data as part of a common base event that is passed to the common event infrastructure. The common event infrastructure enables the event to be written to an event database or forwarded to a message queue. It also makes the event potentially available to the WebSphere Business Monitor. The use of this primitive is primarily intended for production environments.



The SCA import and export binding types are now shown in the diagram. These define the possible transport protocols that can be used with the WebSphere Enterprise Service Bus, for receiving service requests and for making calls to service providers.

First, there are the EJB bindings. These allow mediations to be exposed to clients as EJB stateless session beans. The also enable mediations to call enterprise beans as services. The EJB 2.1 and 3.0 specifications are supported and both local and remote beans are supported.

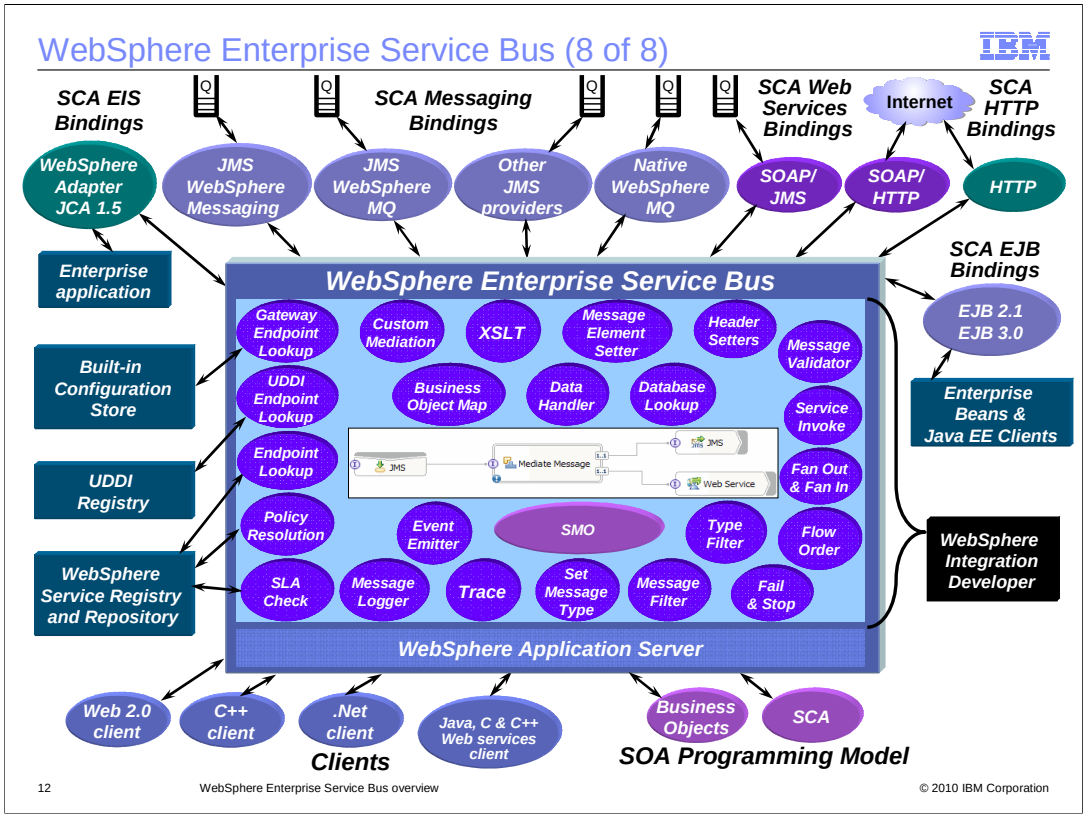
Next, there is the HTTP binding which is used with HTTP protocols. Using this binding, the HTTP message can contain data in any format.

Next there are the Web services bindings. These are used with messages conforming to SOAP and can be configured in several ways. You can use JAX-WS with SOAP 1.1 or SOAP 1.2 over HTTP. Alternatively, you can have JAX-RPC with SOAP 1.1 over either JMS or HTTP.

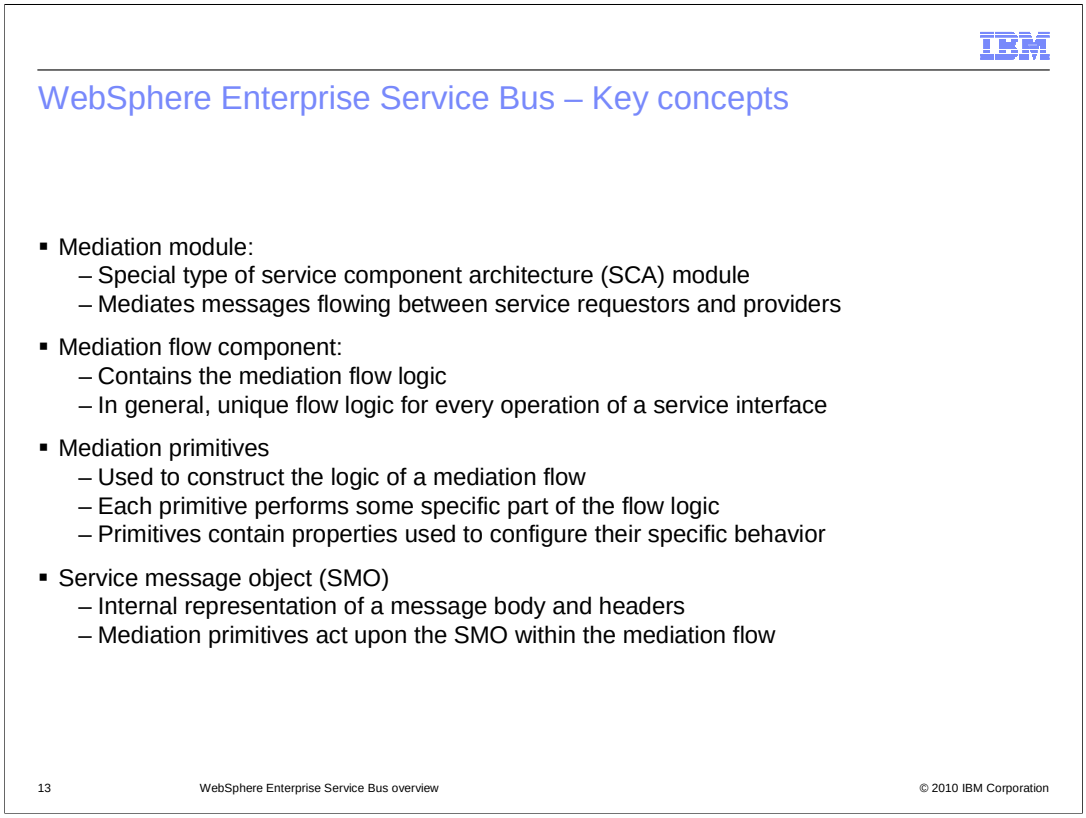
The next set of bindings are the messaging bindings. There are JMS bindings that can be used with the messaging support that is built directly into WebSphere Enterprise Service Bus and is also available with WebSphere Application Server and WebSphere Process Server. There are also JMS bindings that can be used with a WebSphere MQ JMS provider. Additionally, there are generic JMS bindings, which can be used with other JMS providers that conform to the JMS 1.1 specification. Finally, there are messaging bindings, which can be used directly with native WebSphere MQ without JMS.

Another kind of binding is the SCA EIS binding, which is used in conjunction with WebSphere Adapters. These are built following the JCA 1.5 resource adapter specification. They include both technology adapters, such as the e-mail adapter, and application adapters, such as the SAP adapter.

Not shown in the diagram, but also supported, are the SCA bindings, sometimes referred to as the SCA default binding. This binding type is based on the SCA messaging model and can be used between SCA imports and exports.



In this completed picture you will now see those things that are external to the WebSphere Enterprise Service Bus. There are queues associated with the various messaging bindings. The information defining queues, topic, queue managers and so forth, are part of the configuration information for each of the bindings. You can see that the HTTP and SOAP over HTTP bindings are connected to the internet or an intranet. The SCA EIS bindings with their associated WebSphere Adapters talk to whatever backend system is supported by the adapter configured for that binding. The EJB bindings can be called from Java EE clients or call out to enterprise beans. Finally, you can see that calls can be made to the WebSphere Enterprise Service Bus using various types of clients.



The previous slides showed and explained to you the functionality of the WebSphere Enterprise Service Bus. This slide summarizes the key concepts already presented that are needed to understand mediations as implemented in WebSphere Enterprise Service Bus.

Starting at the highest level of abstraction, there are mediation modules, which are a special type of SCA module. They make use of SCA exports and imports to communicate with service requestors and service providers. These provide the key to handling protocol conversions within the bus. The mediation module also contains a mediation flow component, which is a type of SCA component implementation.

The mediation flow component is where the overall logic for the mediation is defined. In general, for every operation defined on an input interface there is unique mediation flow logic defined for the operation's request and response. It is in this mediation flow logic that message transformation and dynamic routing decisions take place.

The flow logic is defined within the mediation flow component using mediation primitives. Each mediation primitive provides some specific portion of the logic. The overall logic is defined by wiring these mediation primitives together into a logical flow.

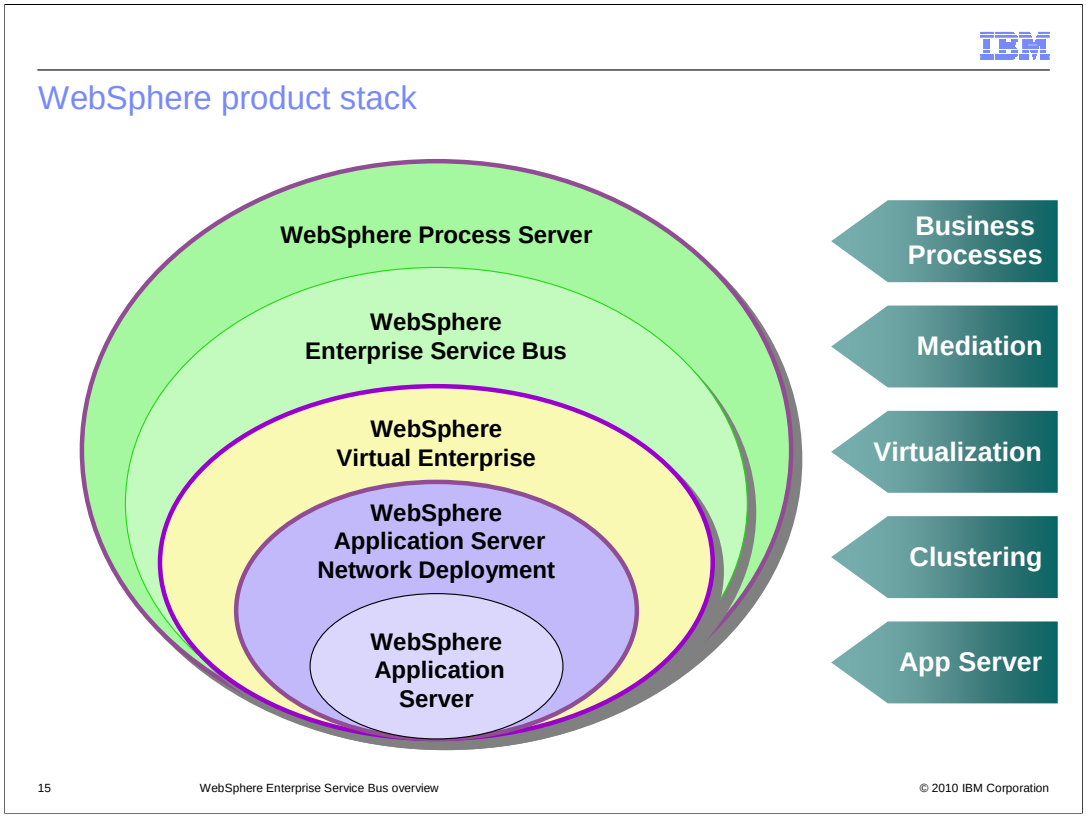
The data that is acted upon by the mediation primitives is contained in a representation of the message called a service message object. The SMO contains a body, which is the application data and headers, which provide protocol and control data.

To summarize, the highest level of a mediation is the mediation module, which contains a mediation flow component, which contains mediation flows defined using mediation primitives which act upon service message objects.

Section

WebSphere Enterprise Service Bus product family

In this section you will learn about the relationship between WebSphere Enterprise Service Bus and other products that are part of the WebSphere server stack.



This slide shows the product stack of the WebSphere servers.

The core of the stack is the WebSphere Application Server which provides the base Java EE application hosting environment.

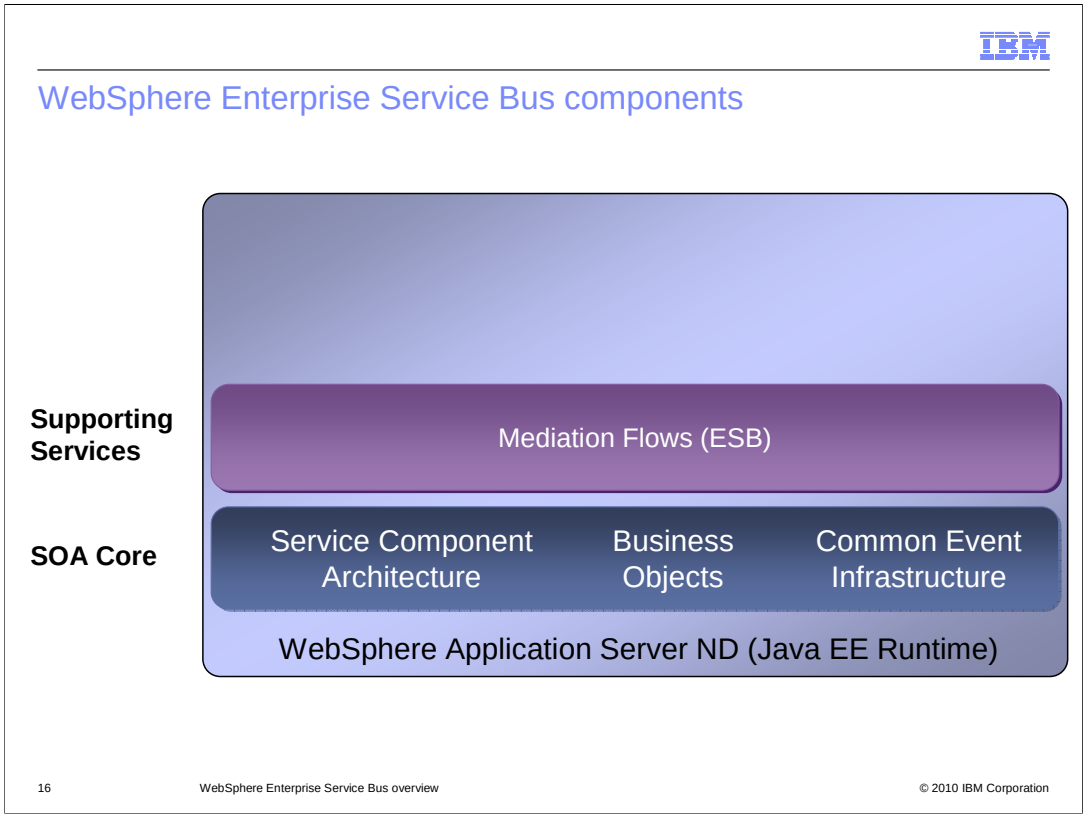
The WebSphere Application Server Network Deployment product then adds the ability to cluster application servers for scalability and high availability and centralizes server administration.

The next part of the stack is WebSphere Virtual Enterprise, which is optional. It provides for virtualization that helps to balance hardware resources and reduce the hardware requirements needed at times of peak demand.

WebSphere Enterprise Service bus is built on top of WebSphere Application Server Network Deployment with or without WebSphere Virtual Enterprise being present. As you have seen in this presentation, WebSphere Enterprise Service Bus provides a service oriented architecture and enterprise service bus functionality to be able to mediate message flows between service requestors and providers.

The top of the stack is WebSphere Process Server which focuses on enabling business processes. It provides the facilities for business process automation and integration to enable business process management applications.

Built as a stackable architecture, each product makes use of the functionality of the products on which they are built, easily allowing the extension of capabilities as needed.



In this slide you can see the components that make up the WebSphere Enterprise Service Bus. As has been previously mentioned, it is built on WebSphere Application Server Network Deployment. This provides a robust Java EE application server runtime with capabilities that the ESB server implementation can exploit, such as JMS messaging, Web services support and enterprise Java beans. It can also make use of the application server qualities of service such as transactions, security and clustering. Overall, this provides a well proven and scaleable runtime environment for WebSphere Enterprise Service Bus.

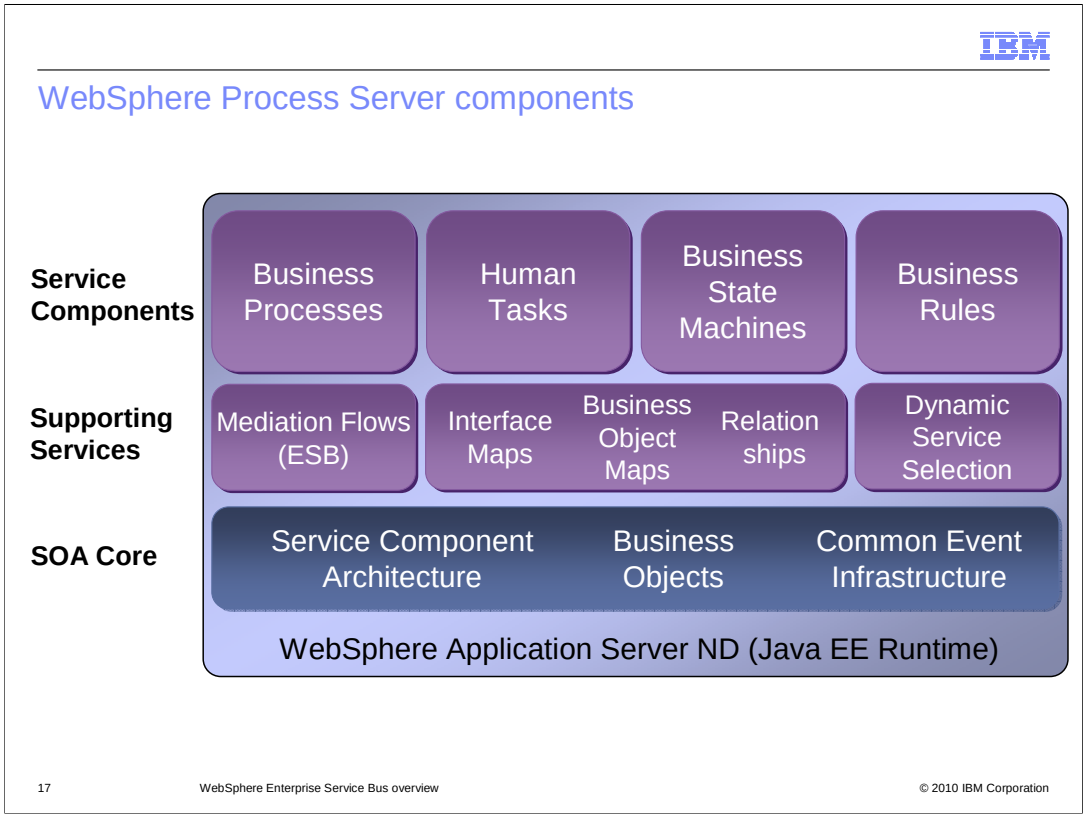
The service oriented architecture core is the foundation for the WebSphere Enterprise Service Bus. The main components of the SOA core are the service component architecture, business objects and the common event infrastructure.

SCA is the uniform programming and invocation model for business services that publish or operate on business data. Business objects represent the data that is passed within that framework. Business objects are extensions to service data objects, carrying additional information needed for some integration scenarios.

The common event infrastructure, also known as CEI, provides the basis for the management and handling of events. It provides facilities for the generation, propagation, persistence and consumption of events.

On top of the SOA core, the heart of the WebSphere Enterprise Service Bus is provided by mediation flows. These are defined within the mediation flow component which has been fully described in the previous slides of this presentation.

With SOA core and mediation flows, the WebSphere Enterprise Service Bus includes all the functions and services necessary to provide the ESB services in any given integration solution.



This slide expands on the previous slide to show the additional components that WebSphere Process Server introduces. As you can see, all the components of WebSphere Enterprise Service Bus are included in the WebSphere Process Server.

WebSphere Process Server introduces some additional supporting services to the mediation flows provided by WebSphere Enterprise Service Bus.

Interface maps, business object maps and relationships combine to handle transformation essential to enabling the integration and synchronization of disparate backend systems within an enterprise.

Dynamic service selection provides the capability to invoke different component implementations of the same interface based on a date and time criteria.

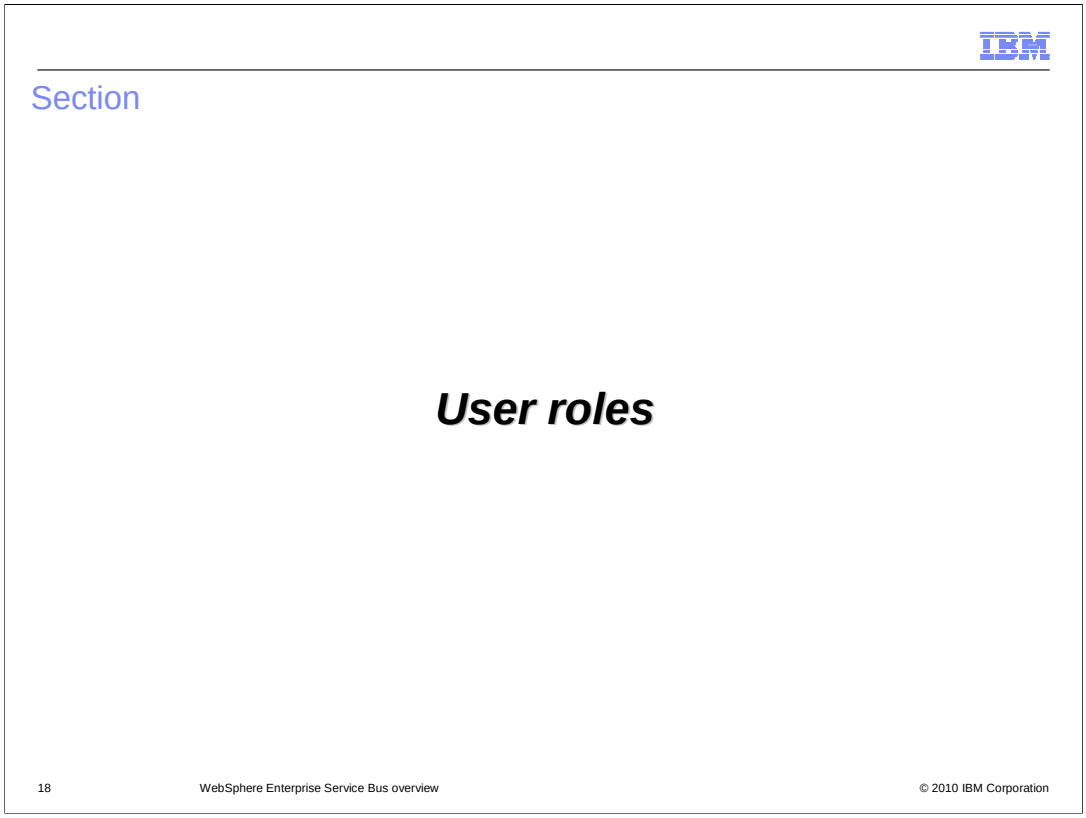
On top of the supporting services are the service components, which are the primary SCA components for enabling business process application function.

Business processes provide an implementation of a process model that describes the logical order in which the different activities of the process take place, making calls out to the individual SCA services that implement the specific activities.

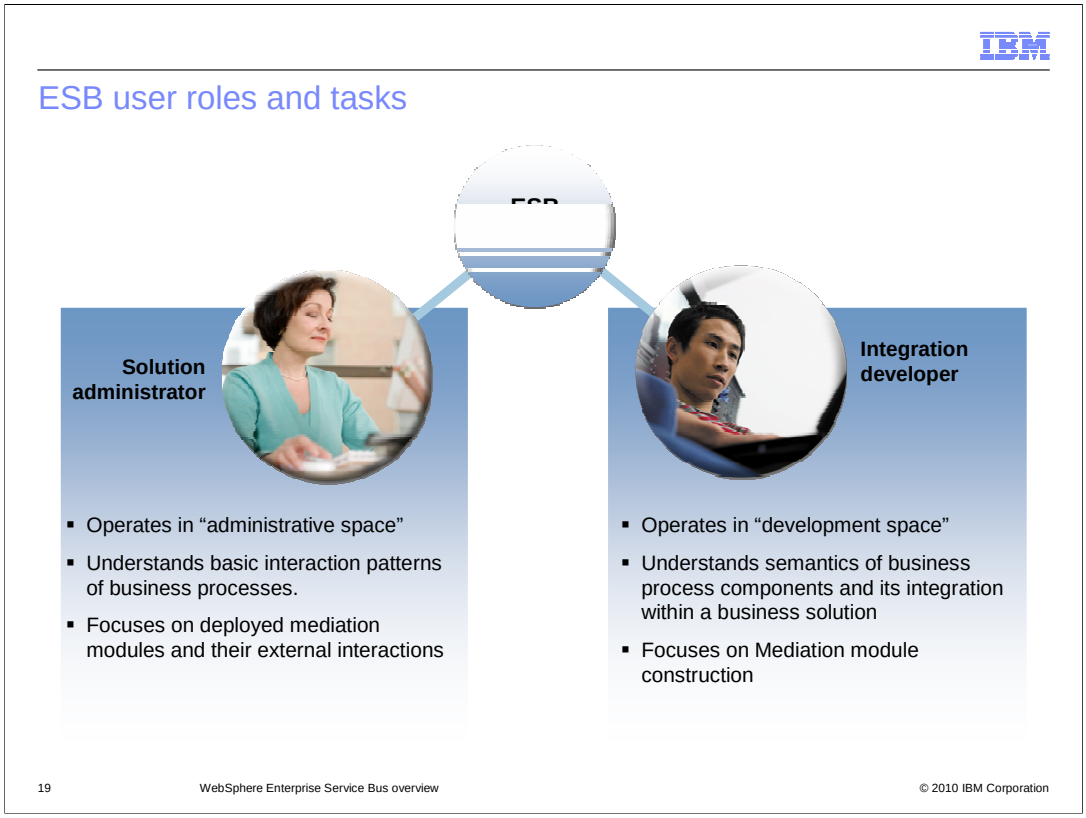
Human tasks allow people to participate in a business process. A human task can integrate into the overall business process in a machine-to-human scenario, a human-to-machine scenario and in a human-to-human scenario.

The business state machine component provides another way of modeling a business process. It is used when the business process can be easily thought of in terms of a state transition diagram.

Business rules provide a means for implementing and enforcing business policy through externalization of business function. Externalization enables the business rules to be managed independently from other aspects of an application.



This section takes a look at user roles that are typically associated with the WebSphere Enterprise Service Bus.



There are two primary user roles that are associated with the WebSphere Enterprise Service Bus.

The first of these is the integration developer. They use the WebSphere Integration Developer tool and they focus on creating the mediation module and defining the mediation flows. Integration developers need to understand the semantics of the various services available and how they can be integrated together within a business solution.

The solution administrator uses WebSphere Enterprise Service Bus or WebSphere Process Server to manage the mediation module within the server runtime environment. In some situations, the solution administrator might need to understand the basic interaction patterns of the business processes to change the routing of the business process if needed.

This slide takes a closer look at some of the tasks performed by the integration developer for developing a mediation flow using the WebSphere Integration Developer tool.

First they need to identify the service requestors and providers that are interacting with the bus. They need to understand the interface and the protocols used by the requestors and providers. The mediation module is then started by defining the appropriate SCA exports and imports needed to communicate with the requestors and providers.

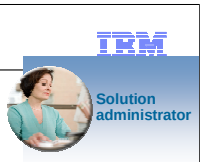
For each operation on the requestors interface, flow logic must be defined for the request and the response. If the service provider supports a different interface, the first thing the integration developer must do is determine which operations on the provider interface can be used to satisfy incoming requests. Then each of the flows must be constructed by selecting the appropriate mediation primitives, wiring them together and customizing their properties to perform the required functions. This might also include the need to write Java code for a custom mediation primitive.

Another decision the integration developer must make is to determine which of the configuration properties of the mediation flow should be exposed at runtime, this allowing a level of runtime configuration of the flow’s behavior.

Finally, the integration developer can use the WebSphere Integration Developer unit test environment servers to test and debug the message flow through the mediation module.

When it has been fully tested, it can be exported from the WebSphere Integration Developer as a Java EE application and provided to the solution administrator.

Solution administrator tasks



- Uses the administrative console or wsadmin commands
- Installs mediation applications into the ESB
  - Normal installation procedures for Java EE applications
  - Provides custom values for mediation flow configuration options exposed by the integration developer
- Manages the installed mediation application
  - Ensures SCA imports are correctly configured for target endpoints
  - Modifies SCA imports as required to setup or reconfigure the environment
  - Modifies mediation flow configuration options as appropriate

The solution administrator operates in the runtime environment for WebSphere Enterprise Service Bus, using the administrative console or the wsadmin commands to administer the ESB applications.

The first task is to install the mediation application. Mediation applications are installed in the same way any other Java EE application is installed into WebSphere Application Server. However, the mediation application might contain mediation flow configuration properties which the integration developer choose to expose to the solution administrator. If so, the default values specified at development time might need to be changed during the installation of the application.

Once the application is installed, the solution administrator needs to make sure that the SCA imports are correctly configured for the service provider endpoints they should be associated with. If not, these can be administratively changed at this time.

Finally, while the mediation application is running, there might be a need to modify configuration options if the behavior of the flow needs to be changed.

Section

Summary and references

The last section of the presentation provides a summary and some references.

Summary

- You learned about:
  - The key concepts of WebSphere Enterprise Service Bus
    - Mediation modules
    - Mediation flow components
    - Mediation primitives
    - Service message objects
  - The place of WebSphere Enterprise Service Bus in the WebSphere family
  - Key user roles associated with WebSphere Enterprise Service Bus
    - Integration developer
    - Solution administrator

In this presentation you learned about the key concepts of the WebSphere Enterprise service bus, specifically the mediation module, mediation flow component, mediation primitives and service message objects. Then the place of the WebSphere Enterprise Service Bus in the WebSphere application server stack was examined. Finally, you learned that the integration developer and the solution administrator are the primary user roles that are associated with the WebSphere Enterprise Service Bus.

References

- IBM WebSphere Business Process Management Version 7.0 Information Center
  - http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp
- Supported platforms and system requirements:
  - WebSphere Integration Developer
    - http://www.ibm.com/software/integration/wid/sysreqs/
  - WebSphere Enterprise Service Bus
    - http://www.ibm.com/software/integration/wsesb/sysreqs

On this slide you will find a link to the IBM WebSphere Business Process Management Information Center, which includes the documentation for the WebSphere Enterprise Service Bus and the WebSphere Integration Developer.

There are also links to the IBM software pages defining the platforms supported and system requirements for WebSphere Integration Developer and WebSphere Enterprise Service Bus.

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_WBPMv70_WESBOverview.ppt

This module is also available in PDF format at: ../WBPMv70_WESBOverview.pdf

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

Trademarks, disclaimer, and copyright information