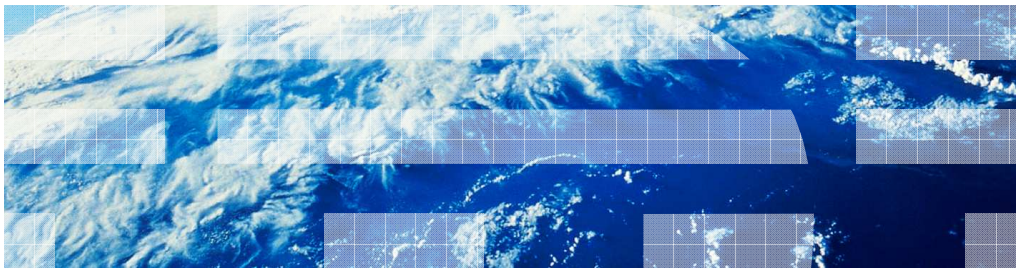
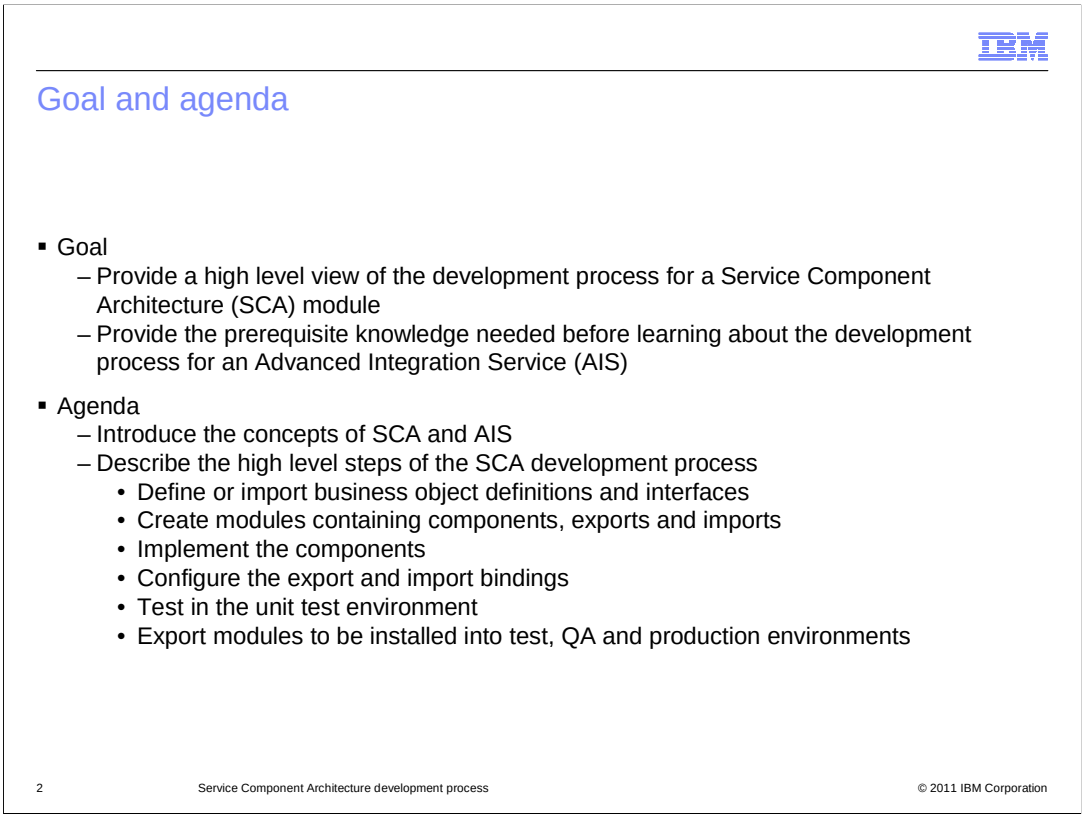
Business Process Management
IBM Business Process Manager V7.5

Service Component Architecture development process

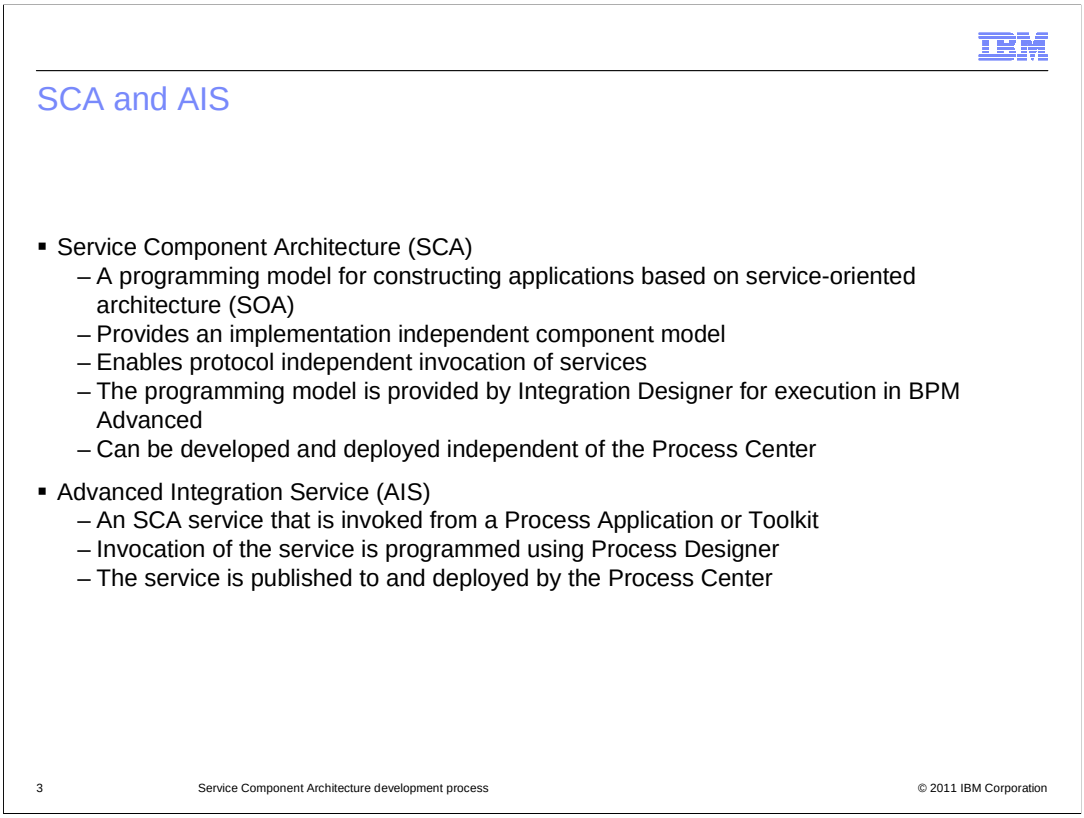


This presentation provides an overview of the process for developing Service Component Architecture services using IBM Integration Designer.



The goal of this presentation is to provide you with an understanding of the Service Component Architecture (SCA) development process. SCA services can be developed and deployed to a Process Server independent from the Process Center repository and Process Center administration. The independent use of SCA services is the focus of this presentation. It is prerequisite knowledge you will need before learning about how an SCA service is used as an Advanced Integration Service (AIS) with a process application or toolkit managed by the Process Center. The AIS development process is covered in another presentation.

The presentation starts by giving you an introduction to the concepts of SCA and the use of SCA services as an AIS. A walkthrough of the high level steps for developing an SCA service are then presented. These include definition of business objects and interfaces and the creating of modules that contain components, imports and exports. The components then need to be implemented and the imports and exports need to have bindings configured, after which the module can be unit tested. Following unit test, the module can be exported from the development environment to be installed into test, quality assurance and production environments as fully functioning SCA services.

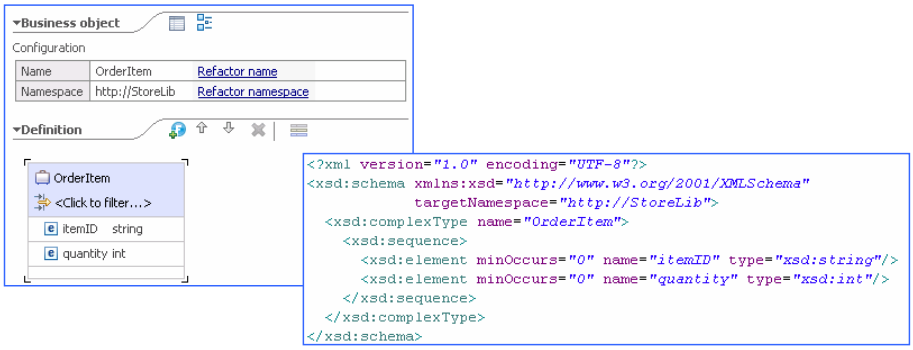


Service Component Architecture is a programming model that implements the basic principles of a service-oriented architecture (SOA). A key characteristic of the SCA architecture is that it provides an implementation independent component model combined with a protocol independent approach to invocation of services. Development of applications using the SCA programming model are done using the Integration Designer and SCA services are deployed to a BPM Advanced Process Server. SCA applications can be deployed to Process Servers independent from a Process Center.

An Advanced Integration Service, or AIS, is an SCA service that is invoked from a process application or toolkit. The Process Designer provides an invocation mechanism for the process application or toolkit to invoke the SCA service as an AIS. In order for the SCA service to be used as an AIS, it must be included as part of the process application or toolkit. Therefore, it is maintained in the Process Center repository and is also deployed by the Process Center.

As was previously mentioned, the development process for SCA is addressed in this presentation and a subsequent presentation addresses how to use the SCA application as an AIS.

Business objects define the data



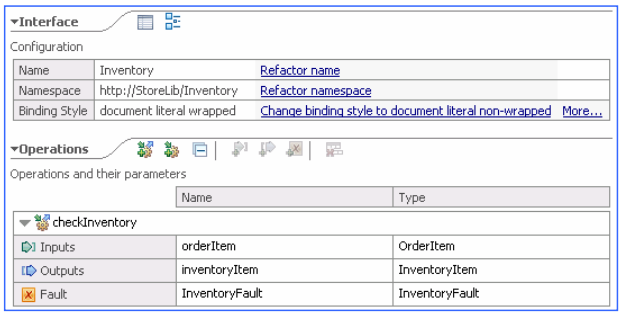
- Business objects define the data used with SCA
- Business objects are defined using XML Schema Definition (XSD)
- The business object editor provides a visual representation of the XSD
- Business objects can be defined using the editor or through import of an XSD

4 Service Component Architecture development process

In SCA, the data is defined by business objects, which are encoded using XML Schema Definitions, or XSD. Integration Designer provides a business object editor which provides a visual representation of the business object definition. This is shown as a screen capture in the upper left of the slide. Next to it is a screen capture of the actual business object definition in the form of an XSD. The business objects can be defined using the editor or they can be imported from existing XSD files.

Key to any SCA application are the business objects that define the data. Defining business objects using XSD provides an implementation independent structure for passing data between SCA components. Defining the business objects is normally the first step in the SCA development process.

Interfaces



- Interfaces define the interactions within SCA
- Interfaces are defined using Web Service Description Language (WSDL)
- They define operations and the data for inputs, outputs and faults
- Operations can be one-way or request/response
- The interface editor provides a visual representation of the WSDL
- Interfaces can be defined using the editor or through import of a WSDL

5 Service Component Architecture development process

Interactions within SCA are defined by interfaces. The Web Service Definition Language, or WSDL, is used to encode these interfaces. WSDL enables the definition of an interface that contains operations which have inputs, outputs and faults. The inputs, outputs and faults can take simple types or business objects as parameters. The operations can be either one-way or request/response. Integration Designer provides an interface editor that represents the WSDL in a visual form for easy editing and an easier way to see the definition. Interfaces used with SCA can be imported from existing WSDL files or created from scratch using the editor.

Similar to the use of XSD for the business objects, the use of WSDL for the interfaces provides an implementation independent mechanism for interaction between SCA components. After defining the business objects, defining the interfaces in normally the next step is the SCA development process.

Fundamental elements of SCA

- Components
  - Container for an implementation
  - Provide a uniform invocation mechanism for various implementation types
- Exports
  - Enable the invocation of an SCA service from an external requestor
  - Provide protocol conversion
- Imports
  - Enable SCA services to invoke external services
  - Provide protocol conversion
- Modules
  - Package components, exports and imports into deployable units

In order to understand SCA, it is important for you to know about the fundamental elements that make up SCA. At the highest level, there are SCA components, exports, imports and modules.

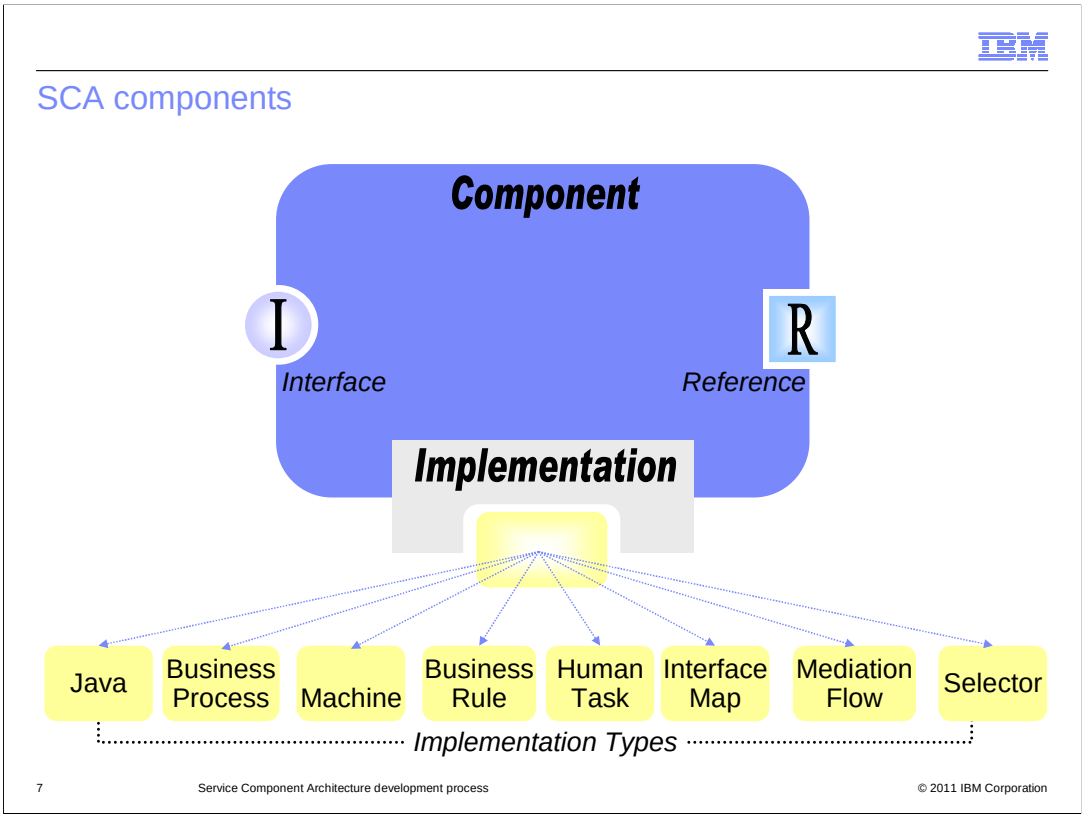
A component provides a container for an implementation. There is a uniform invocation mechanism for an SCA component that is based on the use of WSDL and XSD, but the component itself can be implemented using various implementation types.

Exports enable an SCA service, implemented as an SCA component, to be invoked from outside of SCA. The export provides a protocol conversion mechanism allowing the SCA service to be invoked using a variety of different protocols.

SCA imports provide a mechanism to an SCA component to make a call to a service external to SCA. The import provides the protocol conversion needed to invoke the external service using the appropriate protocol for that service.

Finally, the SCA module is the packaging mechanism for combining components, exports and imports into deployable units.

These are all presented in more detail in the subsequent slides.

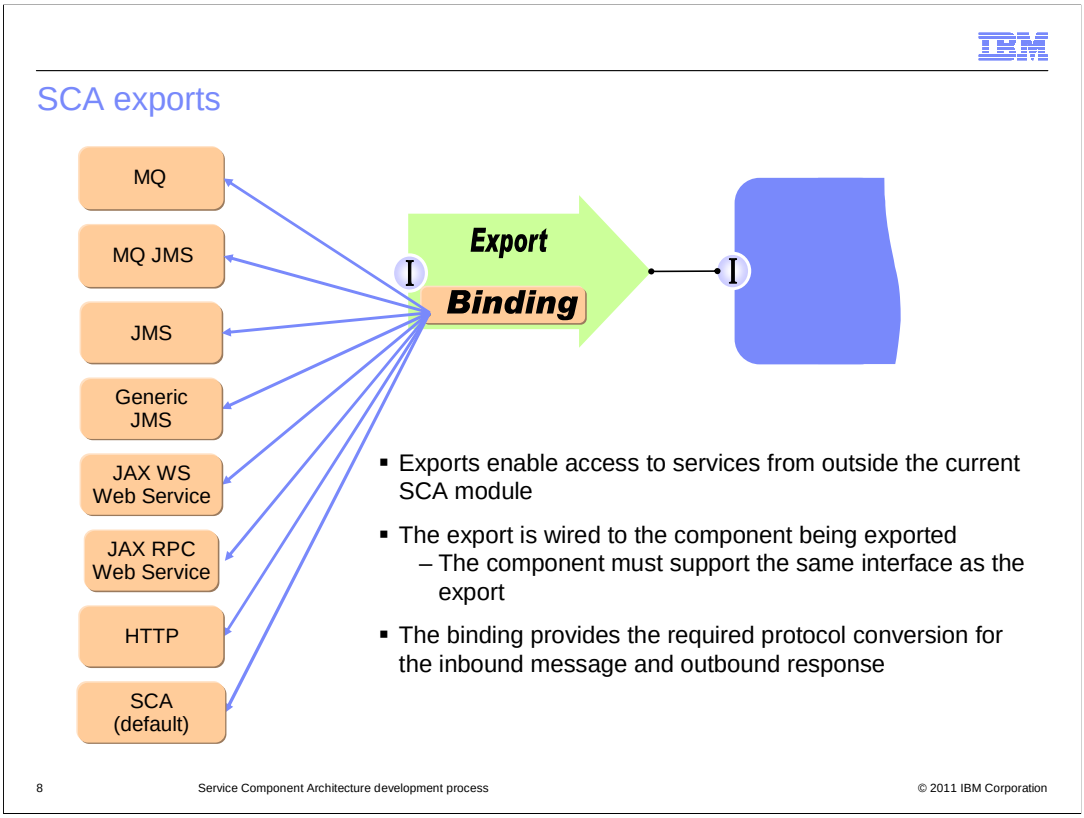


The basic building block in SCA is the component. The component represents a service that publishes or operates on data. The diagram on this slide introduces the essential pieces of a component definition, specifically the interface, implementation, and reference.

A component has one or more interfaces with which it is associated. The interfaces associated with a component advertise the operations provided with this service. These interfaces are defined by the WSDL interfaces previously discussed. The arguments and return types for these interfaces are specified as simple types or as business objects defined by XSD.

Also associated with a component definition is an implementation. As the diagram indicates, there are multiple types available for implementing a component. The implementation types include Java, business processes defined using business process execution language (BPEL), and business state machines that present a state machine representation of BPEL. Human tasks enable a workflow model for human interaction within a business process and business rules abstract elements of business logic enabling modification without changing the implementation. An interface map enables the transformation of data between different interfaces. A mediation flow provides an enterprise service bus (ESB) capability and a selector is a specialized mechanism for controlling flow between SCA components.

Finally, components might need to invoke other components or imports that are within the SCA module. When this is required, the component contains one or more references which are used by the component to do the invocation. A reference is defined by a WSDL interface with data defined by XSD.



SCA exports provide access to components defined in an SCA module for use by requestors outside of the current SCA module. Exports include an interface and a binding. The export is wired to a component that uses the same interface and the binding defines the protocol used by the external requestor to invoke the SCA service. Therefore, you can think of the export as providing the glue between the external requestor and the component providing the service.

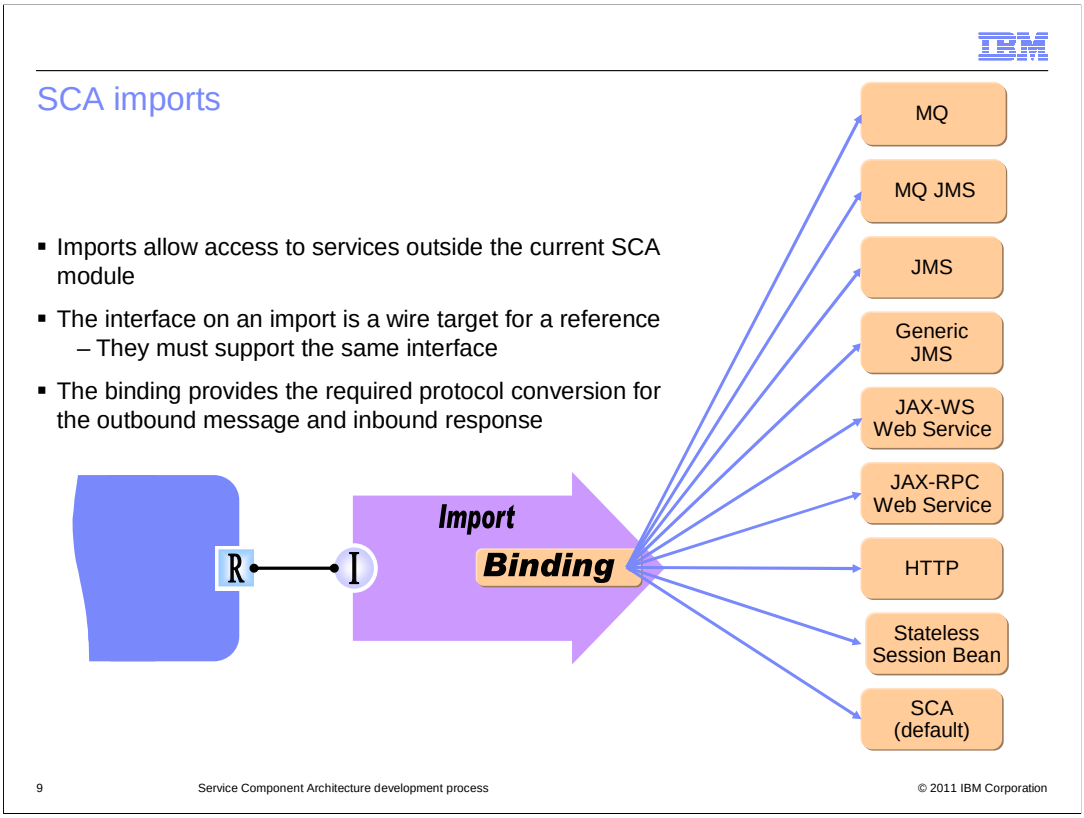
The binding types supported by SCA exports are indicated on this slide.

There are a few different binding types to support messaging protocols. The MQ binding enables requestors using WebSphere MQ native protocols to invoke SCA services, with the binding providing access to the MQ headers and message payloads. The MQ JMS binding enables requestors using the WebSphere MQ JMS provider to invoke SCA services. The JMS binding enables requestors to invoke SCA services using the JMS default messaging provider, which uses the systems integration bus that is built into WebSphere Application Server. Requestors using other JMS 1.1 compliant JMS providers, such as Sonic MQ, make use of the generic JMS binding in order to access SCA services.

The web service bindings type allows SCA services to be exported and made available to external requestors as a web service. The external requestors can use JAX WS with SOAP 1.1 or 1.2 over HTTP to invoke the SCA service. They can also use JAX RPC with SOAP 1.1 over HTTP or JMS.

The HTTP bindings enable HTTP based applications to access SCA services. The payload does not have to be SOAP but can be any format. Access to the HTTP headers is provided.

The SCA default binding type allows SCA services to be exported to other SCA requestors in modules external to the current SCA module. This binding type is used in conjunction with a corresponding import in another SCA module that has an SCA default binding.



SCA imports allow components in an SCA module to access services that are outside the current SCA module. Imports include an interface and a binding. The import is wired to a reference on a component that uses the same interface and the binding defines the protocol used to invoke the external provider. Similar to the export, you can think of the import as providing the glue between the component requesting the service and the external provider of that service. The binding types supported by SCA imports are indicated on this slide.

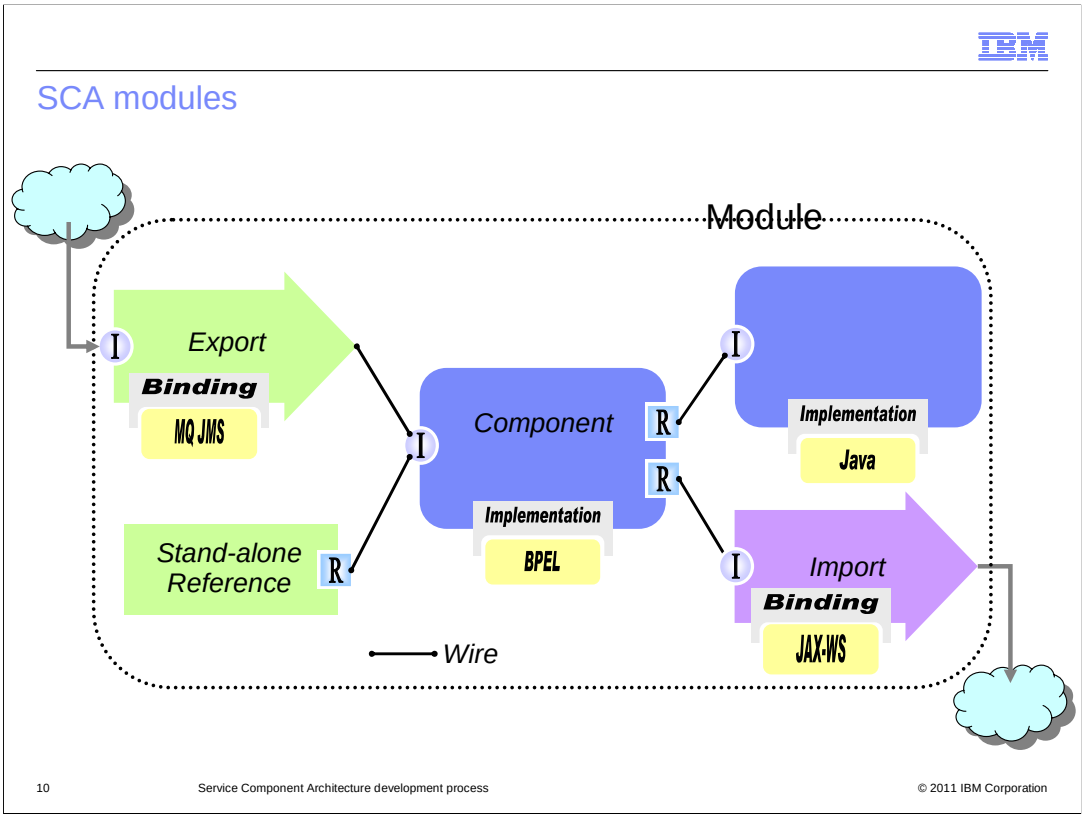
As with the export, there are multiple binding types that handle messaging protocols. The MQ binding enables invocation of the external service using WebSphere MQ native protocols, including access to MQ headers and message payloads. The remaining messaging bindings are all based on JMS. The MQ JMS binding provides access to services using the WebSphere MQ JMS provider. The JMS binding makes use of the JMS default messaging provider that is built into WebSphere Application Server. The last of the messaging bindings is the generic JMS binding which can be used with any JMS provider that is compliant with the JMS 1.1 specification.

Web service bindings allow requestors to access external web services using the SCA programming model. Support is included for JAX WS using SOAP 1.1 or 1.2 over HTTP. Support is also included for JAX RPC using SOAP 1.1 over either HTTP or JMS.

The HTTP bindings provide access to HTTP based applications. These are different from web service bindings in that the payload does not have to be SOAP, but can be any format. Also, unlike the web service bindings, the HTTP bindings provide access to the HTTP headers.

Stateless session bean bindings enable requestors, using the SCA programming model, to access services that have been exposed using a stateless session bean.

Finally, there is the SCA binding. Imports with SCA bindings allow requestors to access SCA services that reside in another SCA module and have an export which also has an SCA binding type. Because all types of bindings are a part of SCA, this binding is often referred to as the SCA default binding.



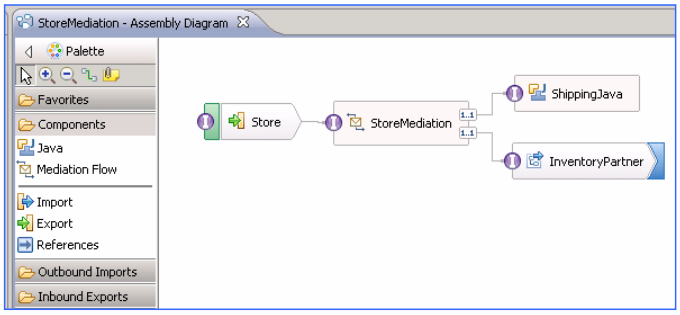
The previous slides introduced the service component as the basic building block in SCA and exports and imports as the glue between SCA and requestors and providers outside of SCA. This slide takes these and puts them into the context of an SCA module. In addition to the components, imports and exports, a module can also contain stand-alone references. These are for a special case of requestor, a non SCA implementation such as a JSP, that is contained in the same Java EE EAR as the SCA module.

Within the SCA module, wires are used to define the connections between the various elements. Wires connect a reference on one element to an interface on another element, indicating a requestor/provider relationship. These must be defined by the same WSDL interface. The exception to this is the export, which has an interface but not a reference. However, it also must be wired to an element supporting the same WSDL interface. Therefore, wires always connect elements which have the same WSDL interface.

In the example shown on the slide, there is an SCA service implemented in a BPEL component. It can be invoked from either an external requestor by sending an MQ JMS message or by an internal non SCA requestor such as a JSP. Within the BPEL component, the logic requires the possible invocation of two other services. One of those services is implemented in Java in an SCA component in the same module. The other service is an external JAX WS web service.

In an SCA module, none of these elements are required to be present. You can have zero, one or many instances of each element type. For example, you might not have an export, rather just a component with a stand-alone reference. It is common not to have imports as they are only needed when the implementation makes an external call. Although uncommon, you can have a module without a component, such as a stand-alone reference wired to an import, enabling a JSP to make an external call through SCA. However, it is most common to have at least one export and one component within an SCA module, possibly containing imports, additional components and additional exports.

Assembly editor

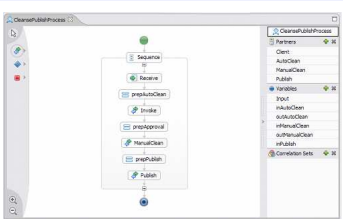


- SCA modules are built using the assembly editor
- Drag and drop components, imports and exports from the palette to the canvas
- Configure the interfaces
- Use a wiring tool to establish the connections

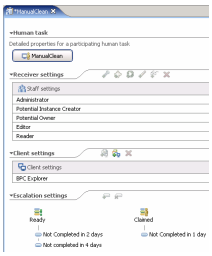
Integration Designer provides an assembly editor that is used to define your SCA modules. It uses a drag-and-drop paradigm to take the elements from a palette and drop them on the canvas. After adding an element, you can associate WSDL interfaces with the interfaces and references needed by that element. There is a wiring tool used to establish the wires between the various elements. You end up with a diagram that is similar to the one shown on the previous slide. Generally, building the SCA assembly is the next step in the development process after defining your business objects and interfaces.

Implement the component

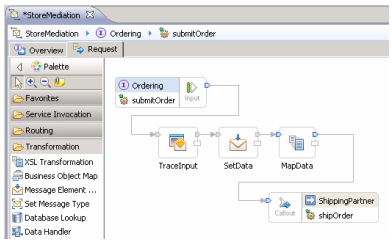
- Each implementation type has its own editor



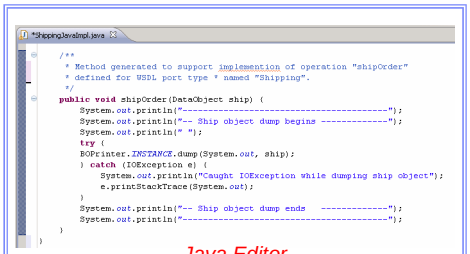
Business Process Editor



Human Task Editor



Mediation Flow Editor

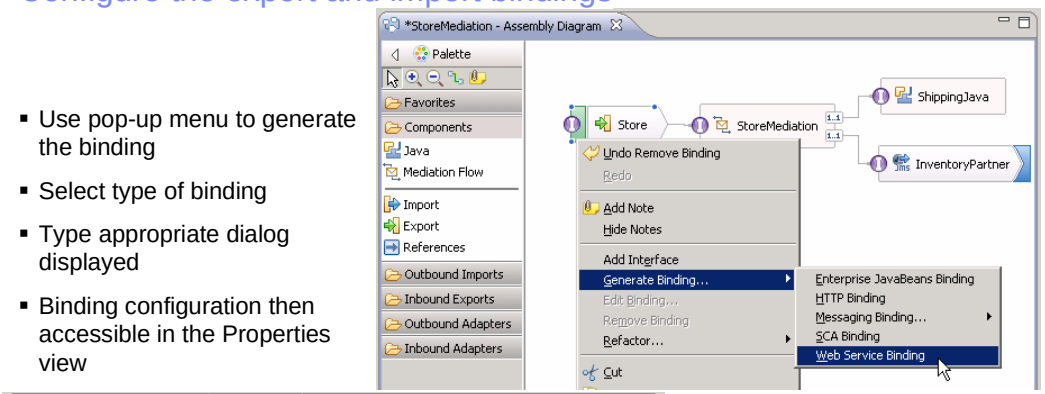


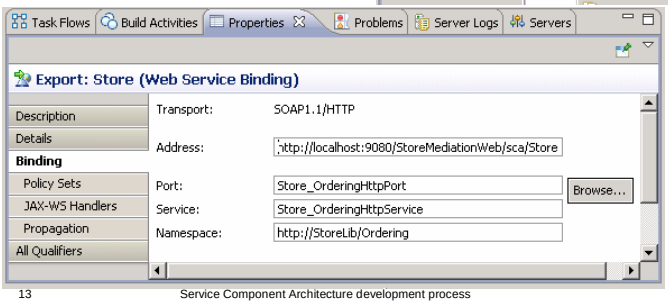
Java Editor

The next step in the development process is the implementation of the components contained within your module. This is when you actually define the logic for your service. The type of the component's implementation determines the tool used to define it. Integration Designer contains individual tools for each of the component types. This slide shows example screen captures for a few of these editors. The business process editor and the mediation flow editor both use drag-and-drop paradigms similar to the assembly editor. Elements are dragged from a palette and dropped on a canvas and properties are used to configure the elements, essentially defining how they will operate. The human task editor allows you to define the characteristics of human tasks in a table form, The Java editor enables you to code your component implementation using a Java aware text editor.

Configure the export and import bindings

- Use pop-up menu to generate the binding
- Select type of binding
- Type appropriate dialog displayed
- Binding configuration then accessible in the Properties view



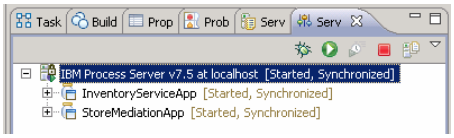


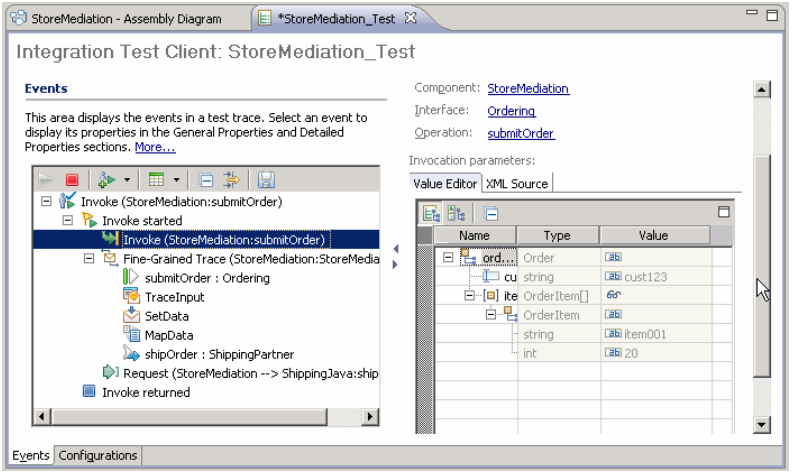
13 Service Component Architecture development process

Once your components are implemented, the next step in the development process is to define the bindings for your imports and exports. The screen capture in the upper right shows a pop-up used to select the protocol type of the binding. At this point, a dialog is displayed where you provide the basic configuration information needed for a binding of this type. After completing the dialog, the properties for the binding are available in the properties view, enabling you to examine and do further configuration of the binding. This is shown in the screen capture at the bottom of the slide.

Test in the unit test environment

- Publish modules to unit test server
- Test using appropriate tools, for example:
  - Integration Test Client
  - Web Service Explorer

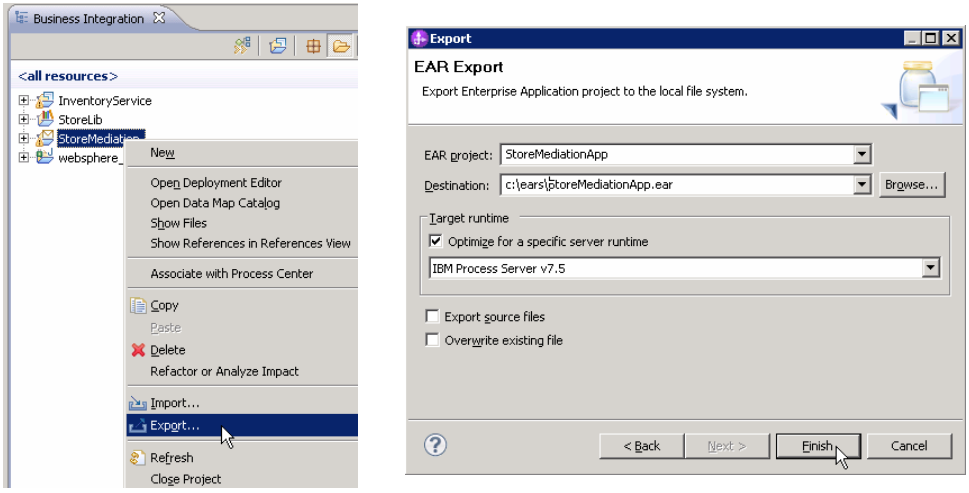




Now that your implementation is complete, you need to unit test it. You deploy the module to the unit test server configured with Integration Designer. The unit test server is a BPM Advanced Process Server independent of a Process Center. The upper right screen capture shows the servers view with the unit test process server and a couple of SCA modules deployed on it. The testing can be done using the integration test client or with the web service explorer if using web service bindings. The main screen capture on this slide shows an SCA module being tested in the integration test client.

Export executables

- Export each module as a Java EE EAR file
- Use administrative console or wsadmin commands to install in test, QA or production servers



The final step in the development process is to export your SCA module as a Java EE EAR file that can be deployed to process servers that are part of your testing, quality assurance or production environments. The screen capture on the left shows the pop-up menu from which you can select to export your SCA module. The dialog on the right is where you provide the particulars about what is exported and where it is exported to.

Flexibility in SCA development process

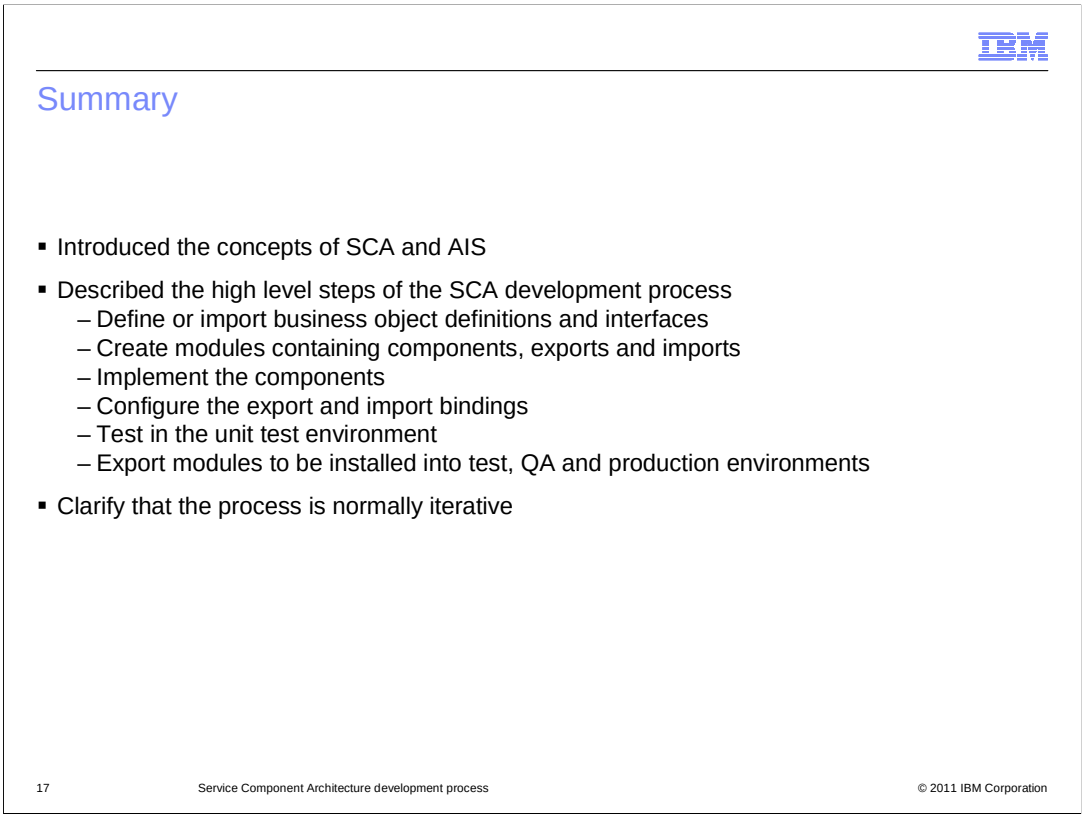
- This presentation provided the basic steps in the SCA development process
  - Presented as if development were a waterfall process
  - Done this way to step through the major items that need to be accomplished
- The actual development process is more iterative
  - Generally you do need to define you business objects and interfaces first
    - These can be modified and expanded during the development process
  - Implementation of a component can be done:
    - Without SCA assembly being complete with imports and exports
    - Iteratively developed and tested using the integration test client
    - Component tested in isolation, with emulation of calls to services not yet existing
  - Assembly diagram updated and expanded as components are tested
  - Import and export bindings often added after component unit testing is complete

Now that you have been introduced to the SCA development process, a clarification seems appropriate. This presentation provided the basic steps in the process as if the waterfall approach to development was being used. This was done as it is the easiest way to convey the tasks that need to be accomplished. However, the actual development process of an SCA module typically follows a more iterative approach.

In general, you do need to start by defining your business objects and your interfaces. These can be modified and expanded during the iterative development process, but it is a good idea to get these thought out ahead of time to limit the possible rework that might occur from changing them.

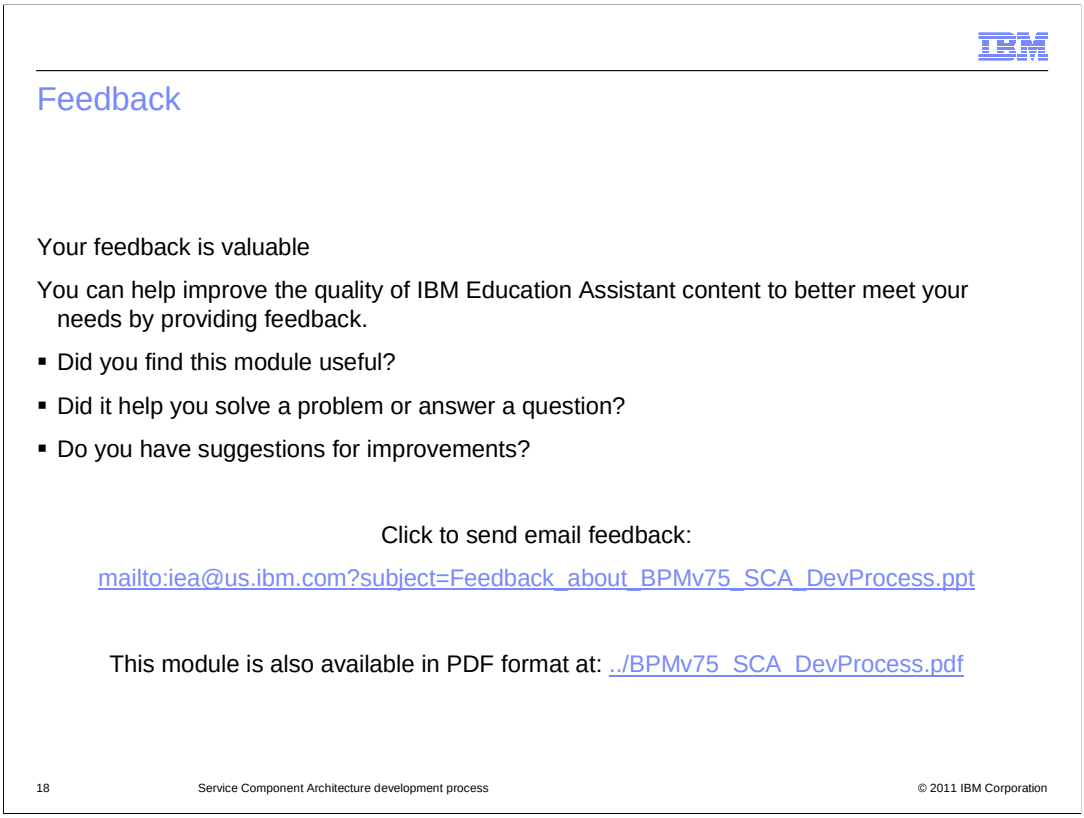
Next, you need to create your assembly diagram and add a component, but you do not need to add all the components, imports and exports that will eventually be included. As long as you have a component in the assembly that is configured with an interface, you can begin the implementation of that component. As you work on the implementation, it is possible to use the integration test client to test it even when it is only partially completed. Typically, an iterative process is used for developing the implementation, combined with unit testing until the component is complete. If your component makes calls to other services, these can be emulated in the integration test client so that the other services do not need to exist for you to proceed.

While iteratively developing, or when the component is complete, you can add other components, imports and exports to the assembly. This happens if you want to expand your testing beyond the current state of the component being worked on. For example, an external call that was being emulated in the integration test client might now be made live by adding an import to the actual service. The import and export bindings are also added as needed. For example, if the component is eventually going to be called as a web service, you can add the export with a web service binding and use the web service explorer to continue testing.



In summary, you were given an introduction to the concepts of SCA and the use of SCA services as an AIS. A walkthrough of the high level steps for developing an SCA service were presented. These included definition of business objects and interfaces and the creating of modules that contain components, imports and exports. The implementation of components was discussed as was configuring imports and exports with their appropriate bindings. Unit testing the module was then presented along with the export for deployment of the completed module.

After presenting the SCA development process as consisting of these steps, a clarification was made to bring out the fact that the process is not a waterfall process but is actually much more of an iterative process.



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

Trademarks, disclaimer, and copyright information