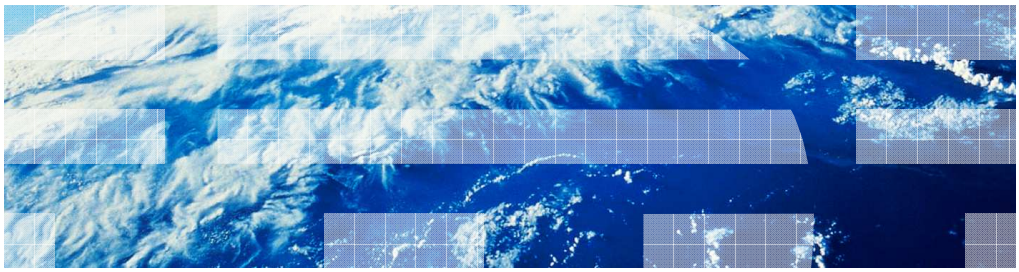
IBM WebSphere Application Server V8

Service Component Architecture (SCA)



WebSphere. software

This presentation describes support for the IBM WebSphere Application Server V8 Service Component Architecture support.

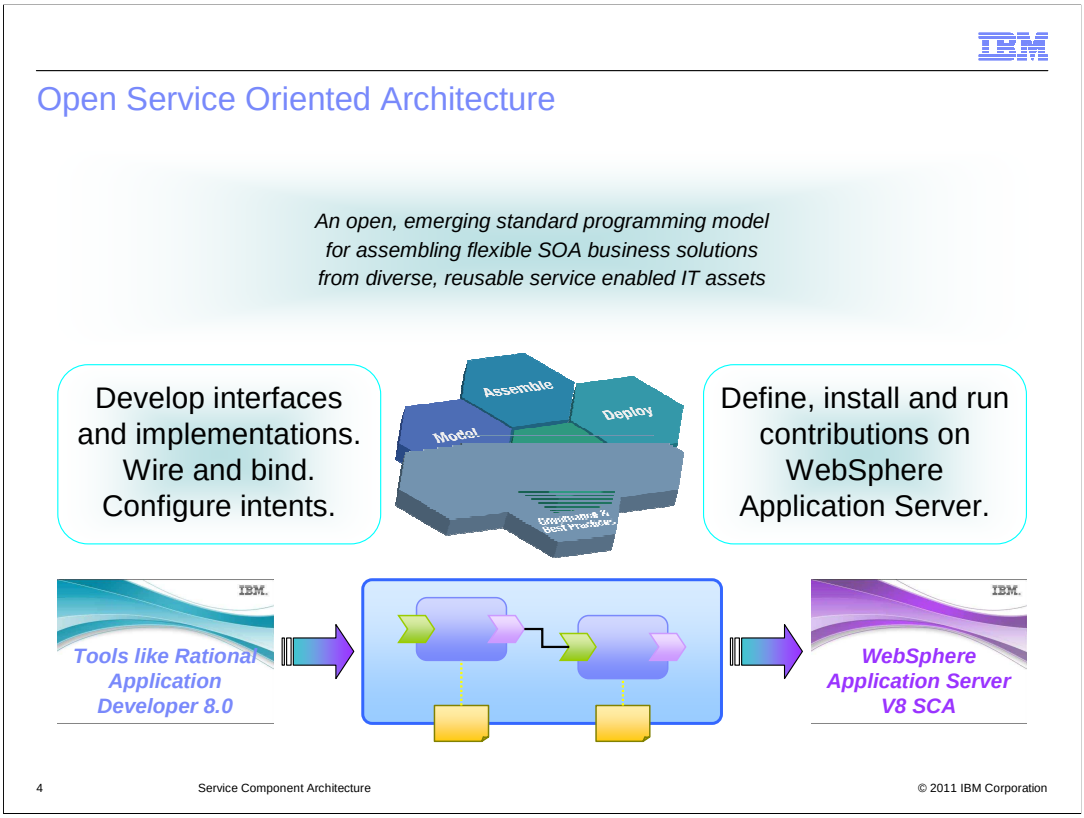
Table of contents

This presentation will start out by briefly going over the introduction to SCA followed by the SCA overview where it will touch on the features of SCA. Additionally, this presentation will look at what the SCA support is at a high level and then finally summarize.

Section

Introduction to SCA

This section will cover an introduction to SCA.



SCA is an open (OSOA), emerging (OASIS) standard programming model.

It is an assembly of flexible business solutions following SOA principles.

It is an exploitation of diverse, reusable IT assets.

Service Component Architecture defines a simple service-based model for construction, assembly and deployment of services (existing and new ones).

SCA is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture (SOA). SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as web services. SCA is the central programming model of the SOA Foundation and an open, leading industry standard for service composition.

What is SCA? (1 of 2)

- A concrete manifestation of an Service Oriented Architecture (SOA) way of thinking
- Designed for building agile service oriented applications
- A framework for implementing, assembling, composing and deploying services
- Supports loose coupling of coarse grained services
- Extends, exploits and complements existing technologies and standards
- Language, application environment, framework and vendor neutral
- Supports Java and web services, and more

SCA stands for “Service Component Architecture”. SCA is a concrete manifestation of a SOA way of thinking. SCA is designed for building agile service oriented applications. SCA is a framework for implementing, assembling, composing and deploying services. SCA supports loose coupling of coarse grained services. SCA extends, exploits and complements existing technologies and standards. SCA is Language, Application Environment, Framework and Vendor neutral. SCA supports Java and web services, and more.

What is SCA? (2 of 2)

- An extensible set of:
  - Protocol bindings (SCA, WS, RMI)
  - Implementation languages (Java)
  - Interface definitions (WSDL, Java)
  - Pluggable Data bindings (PoJo, JAXB, SDO)
  - Policies and Intents (Integrity, Confidentiality).
- "Classic SCA"
  - Service Component Architecture as defined and built by IBM supported in a variety of WebSphere Family products starting with V6
- "Open SCA"
  - Service Component Architecture as defined by the industry at both the OSOA and OASIS collaboration

SCA is also an extensible set of Protocol bindings (SCA, WS, RMI), Implementation languages (Java), Interface definitions (WSDL, Java), Pluggable Data bindings (PoJo, JAXB) and Policies and Intents (Integrity, Confidentiality).

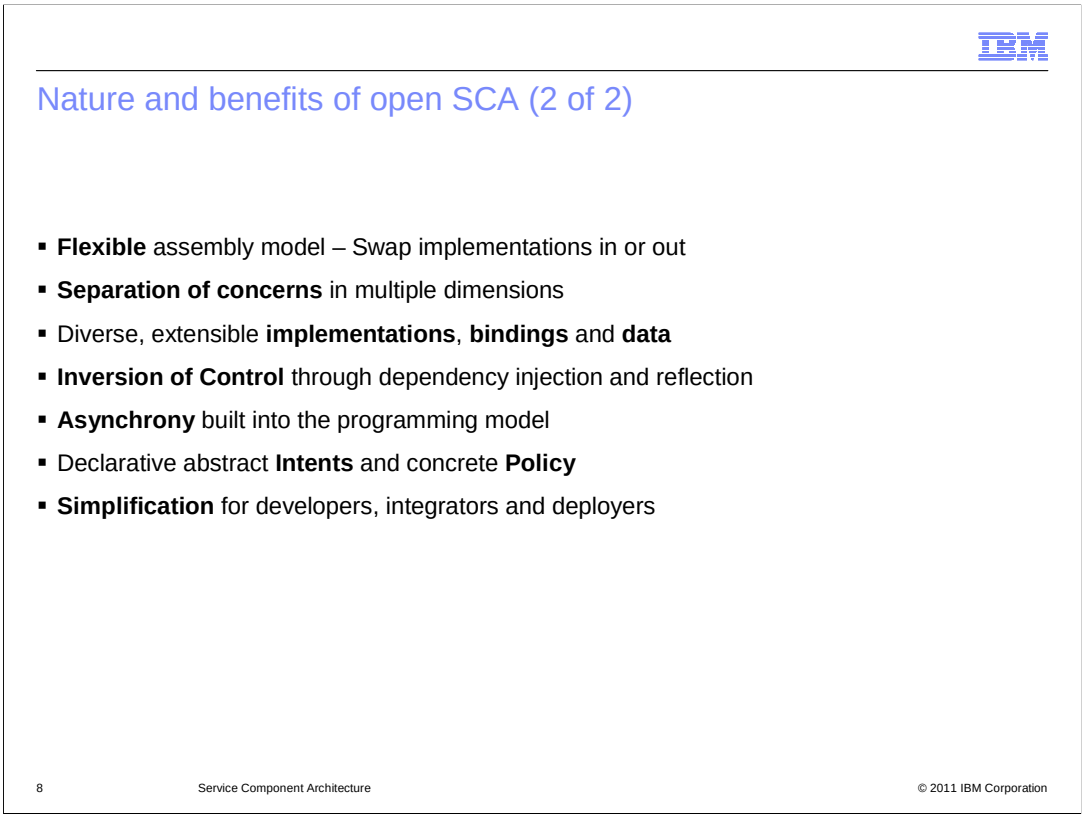
It is important to note the difference between "Classic SCA" and "Open SCA". There is sometimes confusion between the two. "Classic SCA" refers to Service Component Architecture as it is defined and built by IBM. Classic SCA was first introduced by IBM and other vendors in 2005 and it is what is known as SCA 0.5. It is supported in a variety of WebSphere Family products starting with WebSphere V6.

"Open SCA" refers to Service Component Architecture as defined by the industry at both the OSOA and OASIS collaboration which was released to OASIS for standardization in 2007. It is what is referred to as SCA 1.0. Current SCA support in WebSphere Application Server V8 is based on "Open SCA". SCA support was shipped as an add on "feature pack" in WebSphere Application Server V7, but is now shipped as part of WebSphere Application Server V8.

Nature and benefits of open SCA (1 of 2)

- **Open**, multi-vendor programming model designed for SOA
- **Emerging standard** through OASIS
- Extends, exploits and complements **existing standards**
- **Unifying framework** for assembling services from diverse assets
- **Formal representation** of services and their interdependencies
- Embraces **heterogeneity** – Strength and agility through diversity
- **Rapid integration** of components into applications
- **Loose Coupling** of coarse or fine grained components

Here are the benefits of open SCA. SCA is open, multi-vendor programming model designed for SOA with emerging standard through OASIS. Open SCA extends, exploits, and complements existing standards. It unifies frameworks for assembling services from diverse assets, It provides a formal representation of services and their interdependencies. It also embraces strength and agility through diversity. It allows for rapid integration of components into applications and loose coupling or fine grained components.

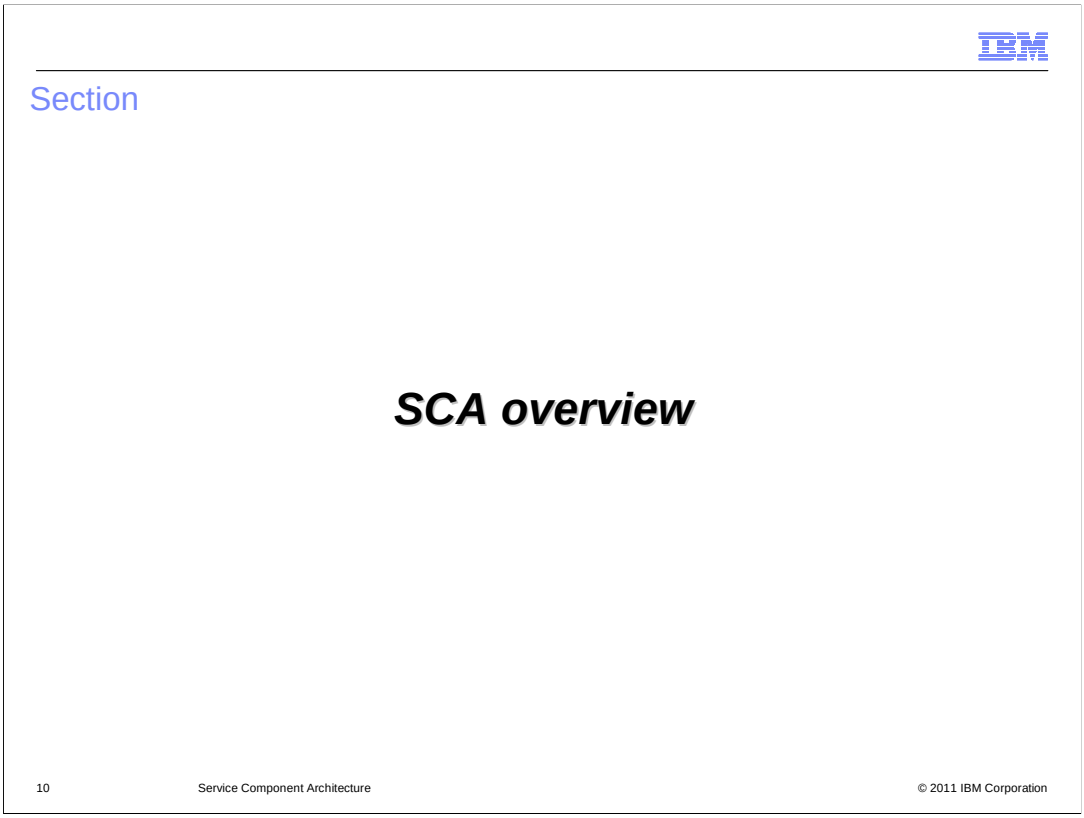


Other open SCA benefits include flexibility in the assembly model and *separation of concerns* in multiple dimensions. Open SCA also offers diverse, extensible *implementations*, *bindings* and *data, Inversion of Control* through dependency injection and reflection and *Asynchrony* built into the programming model. Lastly, it provides declarative abstract *Intents* and concrete *Policy Simplification* for developers, integrators and deployers.

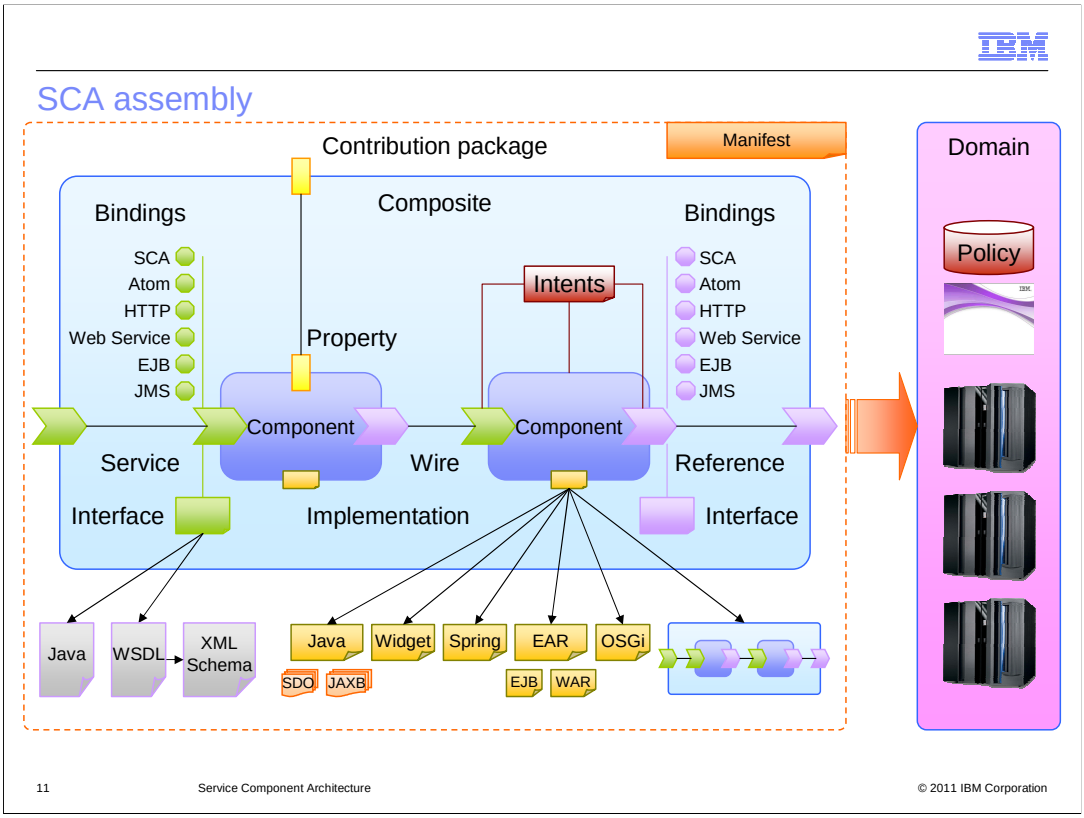
What SCA is NOT

- SCA does not choreograph services or model workflows
- SCA is not web services
- SCA is not tied to a specific runtime environment
- SCA does not force specific languages / technologies

SCA does not choreograph services or model workflows. To do so you can use BPEL or other workflow languages. SCA is not “web services”, web services are a binding technology used by SCA. WSDL is an interface description language to SCA. SCA is not tied to a specific runtime environment, it can be applied to any kind of environment – Distributed, Heterogeneous, Large or Small. SCA does not force specific languages or technologies, it is open to many languages, frameworks, and technologies. SCA embraces, not replaces, assets and other frameworks and it engages new technology.



Now that you have a little bit of background of what SCA is, the next section will provide a more detailed description of the various aspects of SCA.



An essential characteristic of SOA is the ability to assemble new and existing services to create brand new applications that may consist of different technologies.

Service Component Architecture defines a simple, service-based model for construction, assembly and deployment of a network of services (both existing and new ones) These services are defined in a language-neutral way. Tuscany implements the SCA Version 1.0 specification.

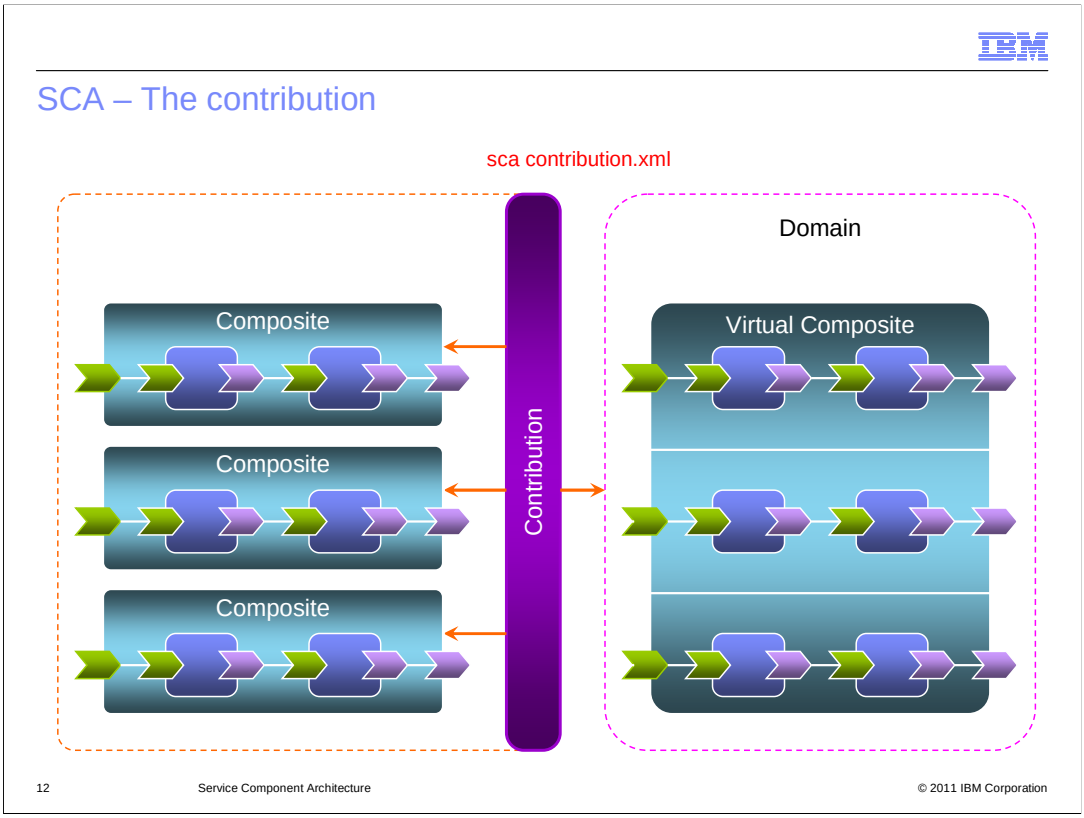
This diagram provides an overview of an SCA composition. It demonstrates some basic concepts of an SCA application.

As the name implies, Service Component Architecture is a component model. Service-oriented coarse-grained building blocks are represented as descriptions of ***components (in blue)***. **Components** describe the ***services (green chevrons)*** they provide and the services they depend on or ***reference (purple chevrons)***. Components also point at the chunk of code which provides the ***implementation (in yellow)*** of the service it provides. Components are then connected together through *wires*, also using metadata. Components can tailor implementations through the usage of ***properties***. Policy and quality of service ***intents*** can decorate services or references (called ***interaction intents***) and can decorate components (called *implementation intents*.)

Assemblies of components are formally composed into ***composites***. A composite is the set of components and wires – the assembly of services. The composite provides a scoping mechanism which defines a local boundary for components, but further can hide services provided in components which are not intended for other SOA applications. Once defined, a composite can be reused to provide the implementation for other components in a nested fashion. The components, assemblies, internal wires and service and reference definitions are written in an open xml language called Service Component Definition Language (SCDL). Services and references in a composite are bound to specific protocols (such as web services) through the usage of ***bindings(in green)***. The bindings are part of the SCDL definition and the business logic (implementation) does not need to be polluted with this detail.

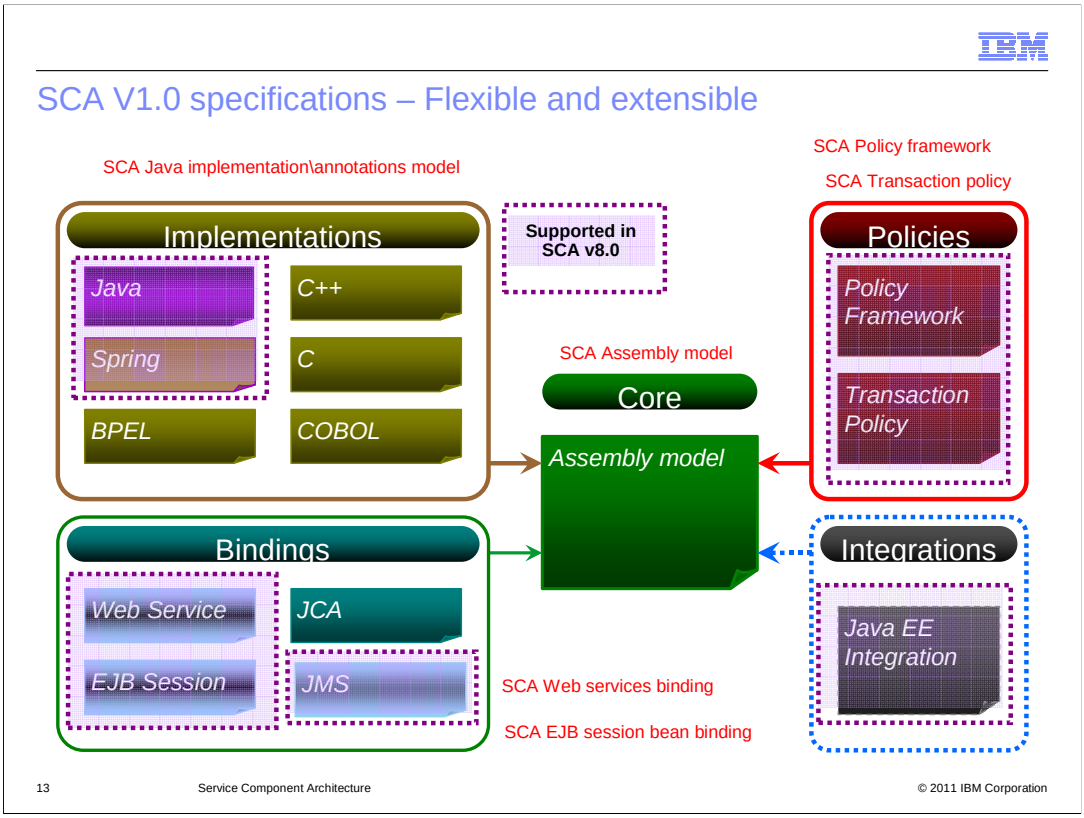
Note that SCA is a language-neutral set of specifications which enable its usage in a variety of application environments. Each application environment will offer the set of implementation, quality of service, and policy features that make sense in that environment. For example, the SCA support in WebSphere Application Server V8 offers support for components whose implementations are Java, other SCA composites, or other technologies such as Spring, Widgets, OSGi and JEE; and specifically does not support the deployment of C or C++ implementations

There is also the concept of **promote.** Just as components expose services, a composite can also expose one or more services. These services are actually implemented by components within the composite. To make them visible to the outside world, the composite's creator can *promote* those services. Lastly, an **SCA Domain** represents a complete runtime configuration, potentially distributed over a series of interconnected runtime nodes. Components sit inside Composites which sit inside a SCA domain (The SCA domain in the WebSphere Application Server maps to a Cell)



Once the composites are defined, they need to be deployed into a runtime. A special xml document called a *contribution* document (sca contribution.xml) describes how composites should be deployed. It contains information regarding which composites in the deployable jar are executable, what their namespaces are, and which can be used by other contributions.

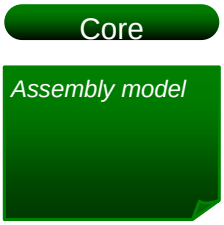
Contributions are deployed into an SCA *Domain*. The SCA Domain is an administration scoping mechanism, but it also provides a service catalog which can be used by the SCA runtime to simplify the wiring of SCA composites. Services deployed in the SCA domain are available over the *SCA default binding.* This allows wires to be specified by its logical domain name without having to specify the particulars of the endpoint configuration as is the case with other bindings. For the purposes of the SCA support in the WebSphere Application Server, the SCA domain and cell have the same scope, and services deployed in the SCA domain are available throughout the WebSphere Application Server cell over the default binding.



SCA V1.0 has several specifications that allow for flexibility and extensibility when developing SCA applications. There are specifications in each group shown, and each of these groups is covered in an upcoming slide.

SCA assembly model

- A declarative model for the assembly of services
- For composition of tightly or loosely coupled services
- Supports asynchrony, callbacks and conversations
- Extensible:
  - Implementation languages
  - Interface type languages
  - Protocol bindings
  - Data bindings
  - Qualities of Service (intents and policy)

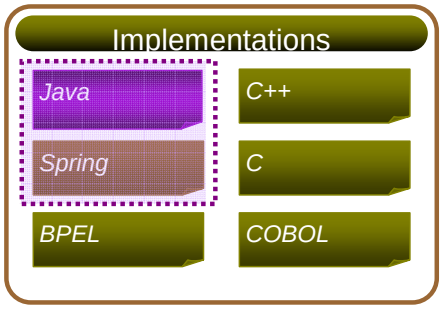


The SCA assembly model provides specifications on how to define structure of composite applications. It is a model for the assembly of services, both tightly coupled and loosely coupled. It is also a model for applying infrastructure capabilities to services and to service interactions, including security and transactions.

The SCA assembly model consists of a series of artifacts which define the configuration of an SCA domain in terms of composites. These contain assemblies of service components and the connections and related artifacts which describe how they are linked together.

SCA Java implementation

- Java Component Implementation Specification
  - defines how Java components can be used in the assembly
  - Defines how POJOs can be used as components in a composition
- Java Common Annotations and APIs Specification
  - Defines Java annotations for SCA Assembly Model concepts
  - Defines the rules for Java to WSDL and WSDL to Java



15 Service Component Architecture

SCA adopts the ease of use improvements provided by JEE5 annotations and modern container design supporting inversion of control or dependency injection. SCA Java implementations have the benefit of declaring service definitions using annotations; having SCA service references, policy and properties injected into their code by the SCA container. Implementations that use this pattern are not only insulated from the specifics of endpoint configuration, but they are also insulated from specific SCA framework APIs. The SCA architecture provides for enterprises to override annotations provided in the code with formal metadata, allowing you to keep things simple during development but giving the enterprise deployment the stringent control over service policy and configuration.

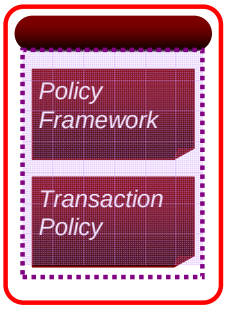
There are two Java specifications: Java Component Implementation Specification which extends the SCA Assembly Model and Java Common Annotations and APIs Specification.

Note that you can incorporate Java components into assembly model without adding any annotation.

Together, these models provide the specifications for writing business services in particular languages like Java, C++, BPEL, PHP, and Spring.

SCA policy framework and transaction policy

- A framework for specifying qualities of service
  - Constraints
  - Capabilities
  - Expectations
- Based on existing standards
  - WS-Policy
  - WS-PolicyAttachment
- Intents, policies and policy sets

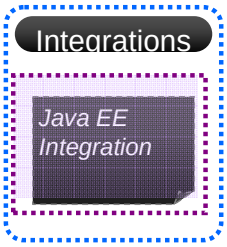


SCA provides a framework to support specification of constraints, capabilities and QoS expectations from component design through to concrete deployment. This specification describes the framework and its usage. It covers policies and policy languages, to be associated with SCA components.

The Policy specification provides the requirements on how to add infrastructure services to handle Security, Transactions, Reliable messaging, and so on.

SCA Java EE integration

- This specification focuses on:
  - The projection of SCA's concepts of assembly, implementation type, and deployment onto Java EE structures
  - Specifying the use of Java EE components as service component implementations
  - Deployment of Java EE archives either within or as SCA contributions

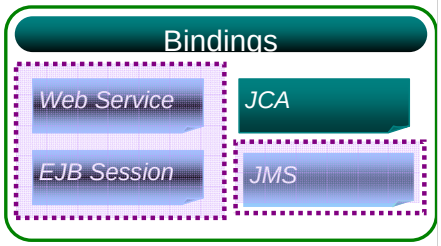


The Java EE client and implementation specification focuses on the projection of SCA's concepts of assembly, implementation type, and deployment onto Java EE structures.

This specification defines the integration of SCA and Java EE within the context of a Java EE application, the use of Java EE components as service component implementations, and the deployment of Java EE archives either within or as SCA contributions. It is also possible to use bindings to achieve some level of integration between SCA and Java EE. These bindings are addressed in separate specifications.

SCA bindings specifications

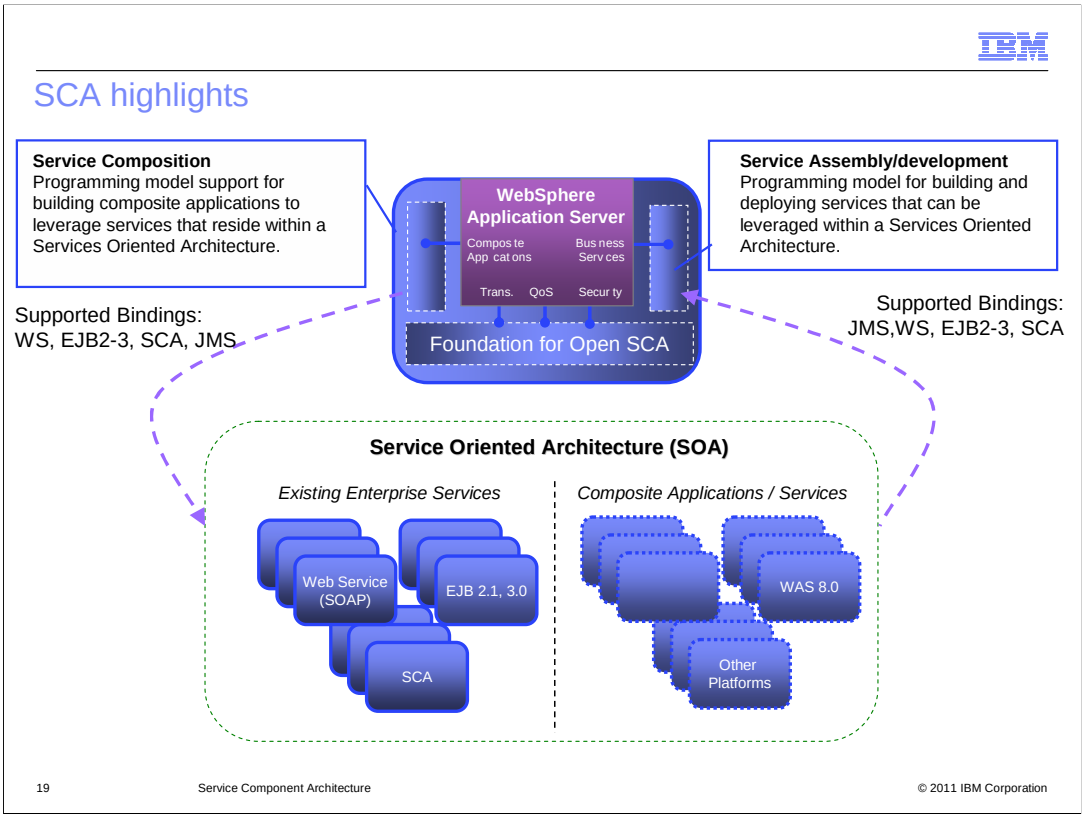
- There are SCA specifications for each binding
  Examples:
  - Web services binding specification
  - JMS binding specification
  - EJB session bean binding specification



There are specifications for the different bindings involved namely web service binding, EJB binding and JCA binding and JMS binding. Each of these provide specifications for each of these bindings.

As an example, the web service binding specification defines the manner in which a service can be made available as a Web service, and in which a reference can invoke a Web service.

Web service binding is important because it is INTEROPERABLE (WS-I).



WebSphere Application Server has chosen to embrace the open source Apache project Tuscany to provide the core of the SCA runtime and then support with the world-class enterprise application server functions inherented in the WebSphere Application Server V8.

This picture shows the specific highlights of SCA, which can be classified into three groups:

Group one, Features under service composition on the application server.

Group two, Features under Simple Service development

Group three, Features under WebSphere Application Server SCA foundation – agility and flexibility

For SCA feature pack highlights you have the following scenarios.

First, Service composition: Businesses today are challenged not only by competitors, but by social and economic pressures that directly affect their information technology systems. As businesses adopt SCA and build a growing inventory of business services, there is a real need to be able to compose, reuse, and otherwise assemble new services from those existing business services. One of the primary objective of this release is to highlight usage of SCA as a coarse-grained composition model that can be used to assemble and compose existing services in your enterprise. The key principle of SCA demonstrated by this support could be described as "Use what you have got and run it where it lives," or "Use your existing services to create new ones.”

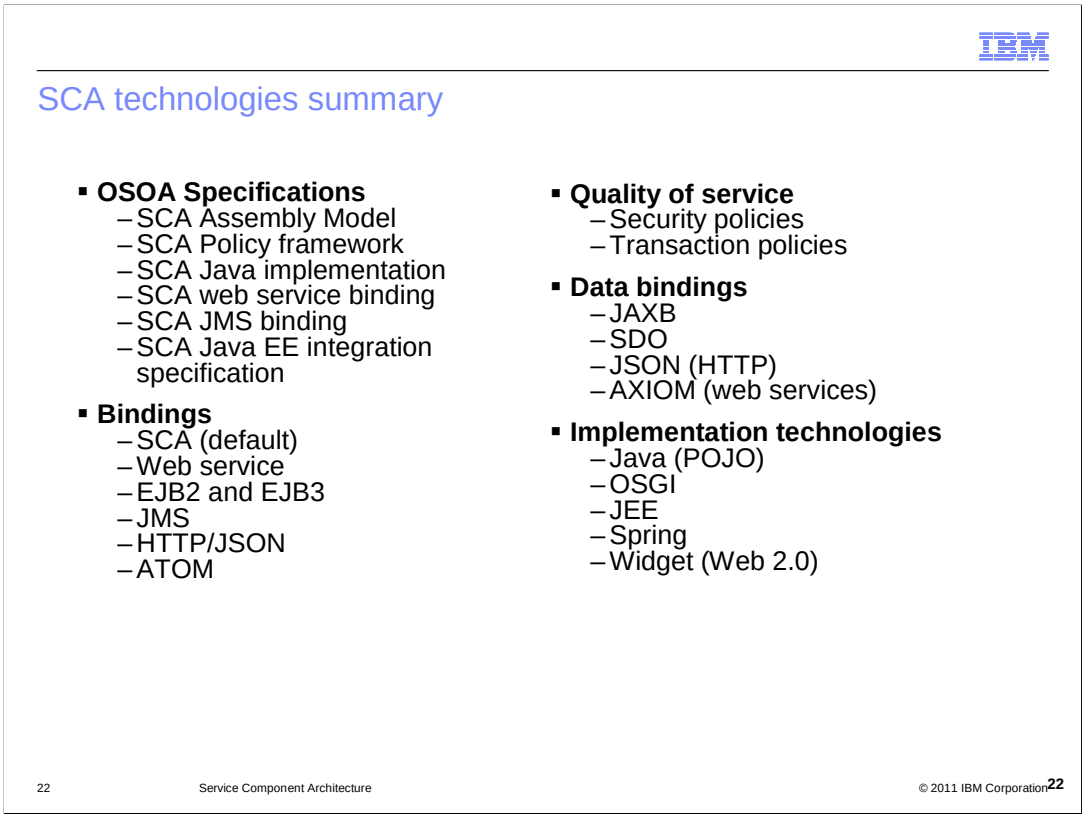
Second, Service development : SCA has a language-neutral programming model for which there are multiple language-specific specifications defined at OSOA. The concepts of SCA apply broadly across both Java and non-Java application environments. The SCA component model has at its heart a strong focus on a proper separation of concerns. The service consumer business logic author should not need to know the details of the service implementation. For instance, a Java service consumer should not be burdened with having to know that a target service is implemented using C++ or COBOL. Therefore another key objective of this delivery of SCA is to highlight the ease-of-use characteristics of SCA service development in Java. This is accomplished by demonstrating annotated plain-old java-object (POJO) components deployed using simple JAR packaging schemes, an easy to use assembly model, and powerful wiring abstractions that enable service definition over different transports and protocols. The key principle of SCA demonstrated by this support might be described as “Know only what you need to know to get your job done,” or “Maintain proper separation of concerns.”

Third, Service agility and flexibility: SCA highlights the flexibility and agility of metadata bindings, and the appropriate separation of concerns. The ability to rewire, compose, and assemble business logic without impacting the business logic itself is key.

Benefits of SCA

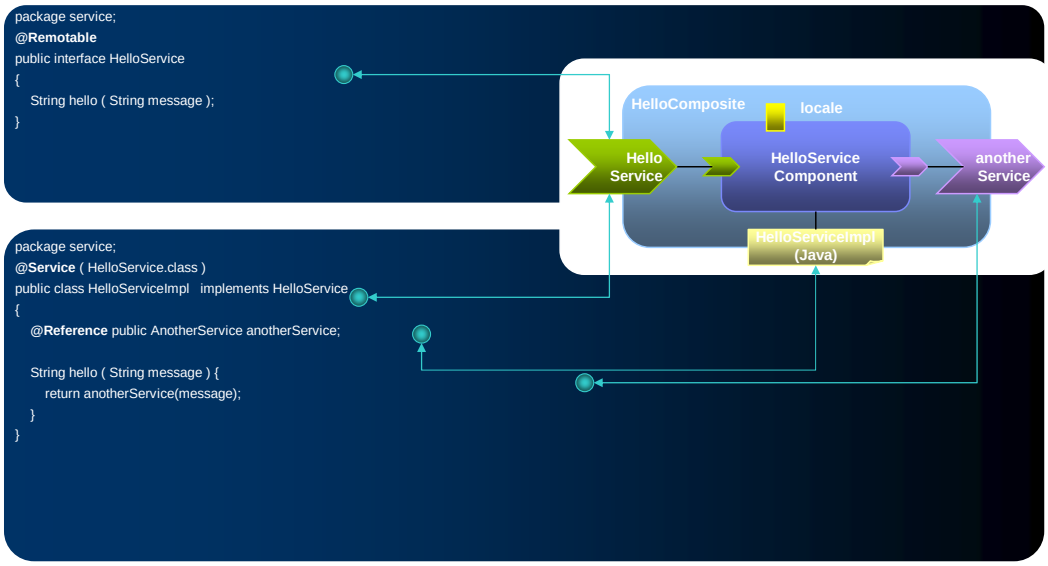
- Perform basic service composition in new POJO-based SCA applications. (Reuse)
- Develop and deploy services that use SCA natively to simplify access from different application types. (Reuse, Connectivity)
- Key technology components supported
  - Java (POJO) language support
  - Wiring Support: SCA, web services (SOAP), EJB2-3, JMS
  - SCA Data Support: JAXB and SDO data bindings
- Quality of Service (QoS) and full WebSphere Application Server platform support

The benefits of SCA include the ability to perform basic service composition in new POJO-based SCA applications. Another benefit is to develop and deploy services that use SCA natively to simplify access from different application types. The key technology components supported include the Java language, wiring support, and SCA data support.



To summarize, SCA supports and helps integrate a wide array of technologies. Some of the bindings supported by SCA in WebSphere Application Server include SCA, web service, JMS, EJB, and Web 2.0 bindings. The component implementation types supported include Java, OSGi, JEE, Spring, and Widget. SCA also supports security and transaction policies and offers a few options for data binding like JAXB and SDO.

Example of a service in Java



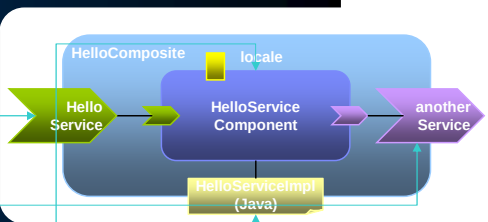
23 Service Component Architecture

Infrastructure logic is all but removed from the business logic. Java annotations, introduced in Java 5, make service implementation, declaration and use easy.

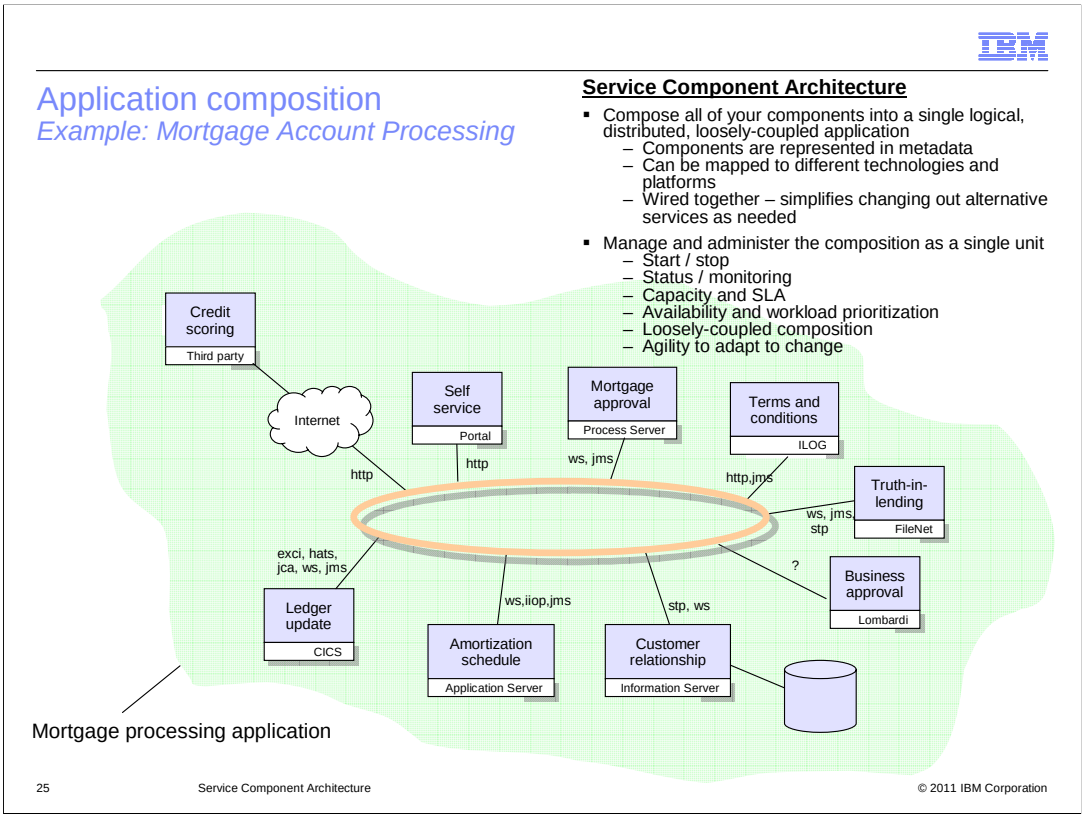
Java 5+ support with annotations makes SCA easier to use programmatically, but support JDK 1.4 is supported through explicit APIs that provide annotation equivalence.

Example of a composite document

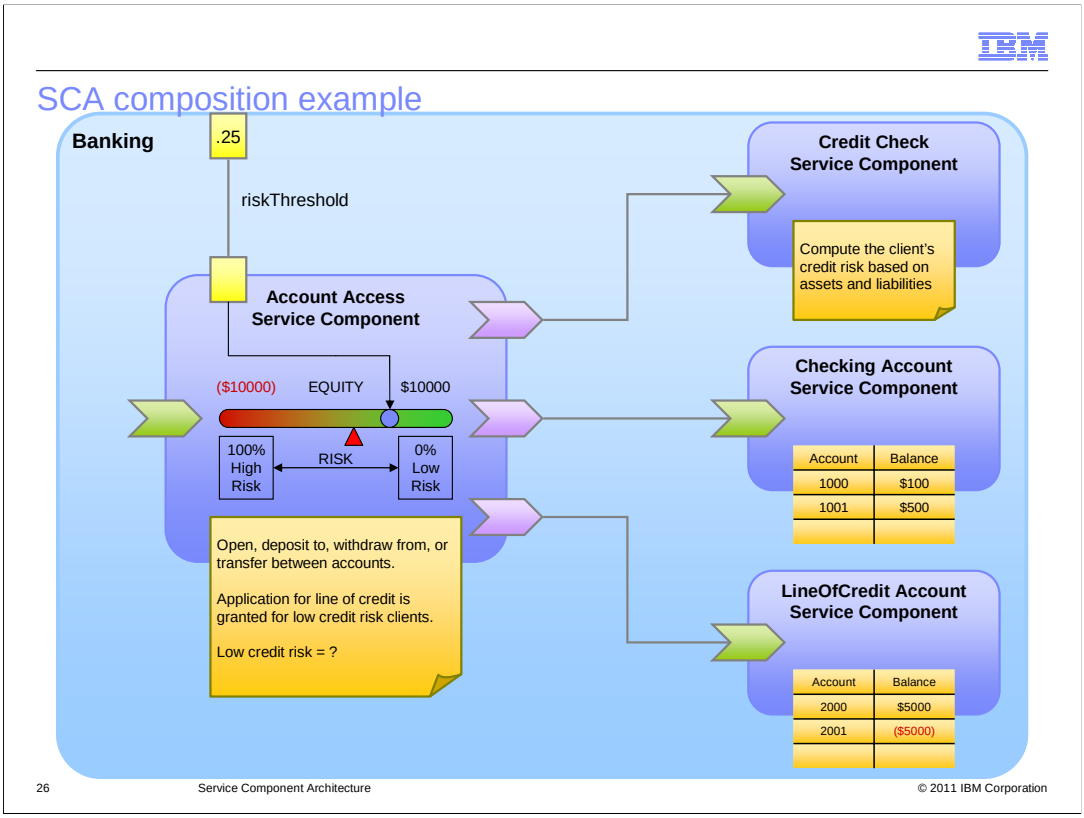
```
<?xml version  1.0  encoding  ASCII ?>
<composite name  HelloComposite
 xmlns  http://www.osoa.org/xmlns/sca/1.0
 targetNamespace  http://too.com >
   <service name  HelloService
    promote  HelloServiceComponent />
   <reference name  anotherService
    promote  HelloServiceComponent />
   <component name  HelloServiceComponent >
      <implementation.java class  services.HelloServiceImpl />
      <property name  locale >en  us</property>
      <reference name  anotherService />
   </component>
</composite>
```



This diagram is an example of a Composite XML, which is the central XML document in the SCA Assembly specification. This is **NOT** a deployment descriptor, deployment is the job of the container, not the developer.



The diagram shows the different ways to expose and consume services using the SCA assembly model. It demonstrates how different technologies can be used in an SCA composition.

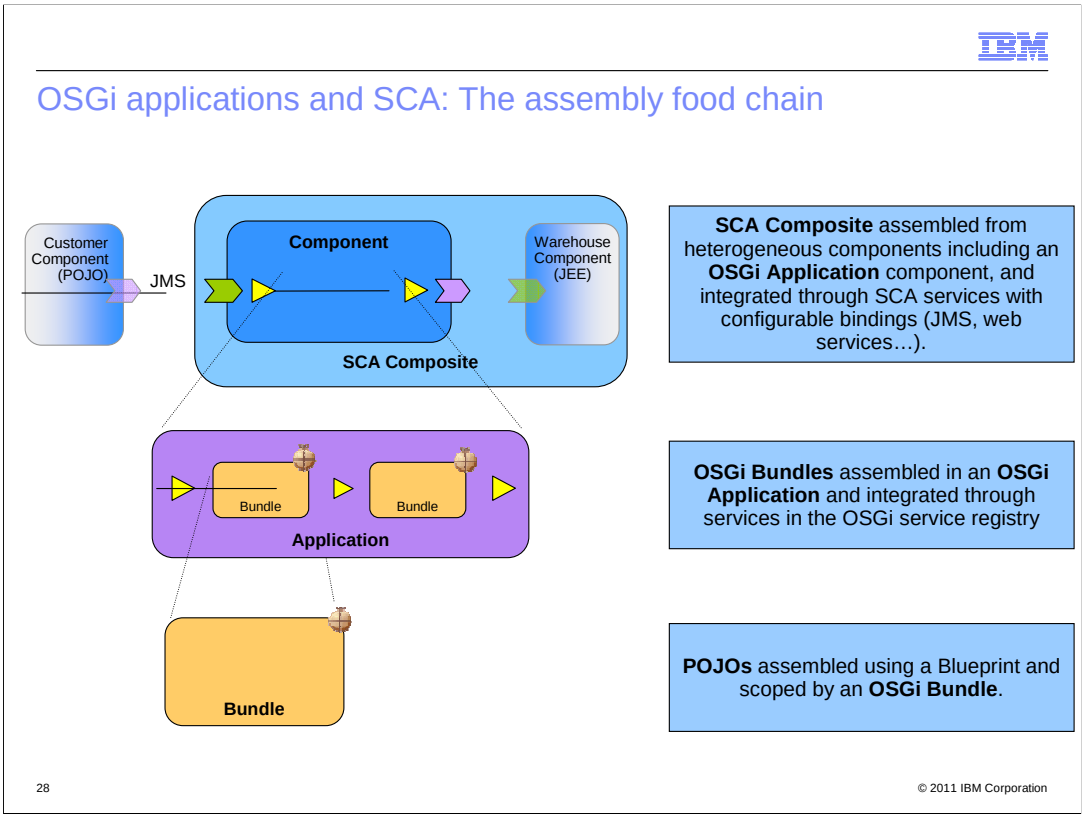


The diagram shows a banking application example, which is composed of several SCA services. The “Account Access” component provides the main service through which all the banking operations can be performed. “Account Access” is wired to several other SCA components (Credit Check, Line Of Credit and Checking Account) which provide specific types of operations on an account.

Section

OSGi integration overview

This section will address the integration of SCA and OSGi.



Earlier talks have shown how Plain Old Java Beans can be assembled and configured in a blueprint bundle and how multiple bundles including web and persistence bundles can be assembled into an isolated OSGi Application.

There is a further level of assembly available. OSGi applications may be represented as an SCA component. Within this composite the OSGi Application is a component that can be wired to other components with different implementation types. For example, an SCA composite could contain an OSGi-application component, a JEE component containing EJBs, a BPEL component and so on. Each component within an SCA composite declares abstract services and references to which concrete bindings can be applied. It is through these services and references that the components of an SCA composites are wired together. The OSGi Application architecture was designed with this form of assembly in mind so that the services and references declared in a Blueprint XML configuration can be exposed through the Application manifest to be visible outside the application. Such exposed services and references can then be mapped to SCA services and references with the full range of available SCA bindings applied to them.

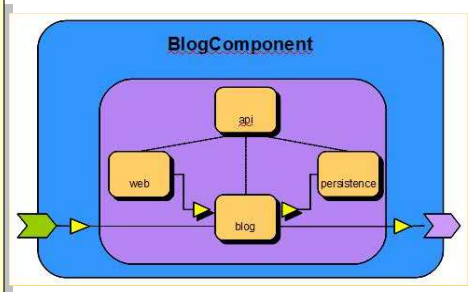
This enables OSGi applications to participate in two new scenarios:

Scenario one is assembly into heterogeneous composites of OSGi and non-OSGi components.

Scenario two is remoting of OSGi application services through SCA services with a variety of bindings including JMS, SOAP/HTTP, IIOP and JSON-RPC.

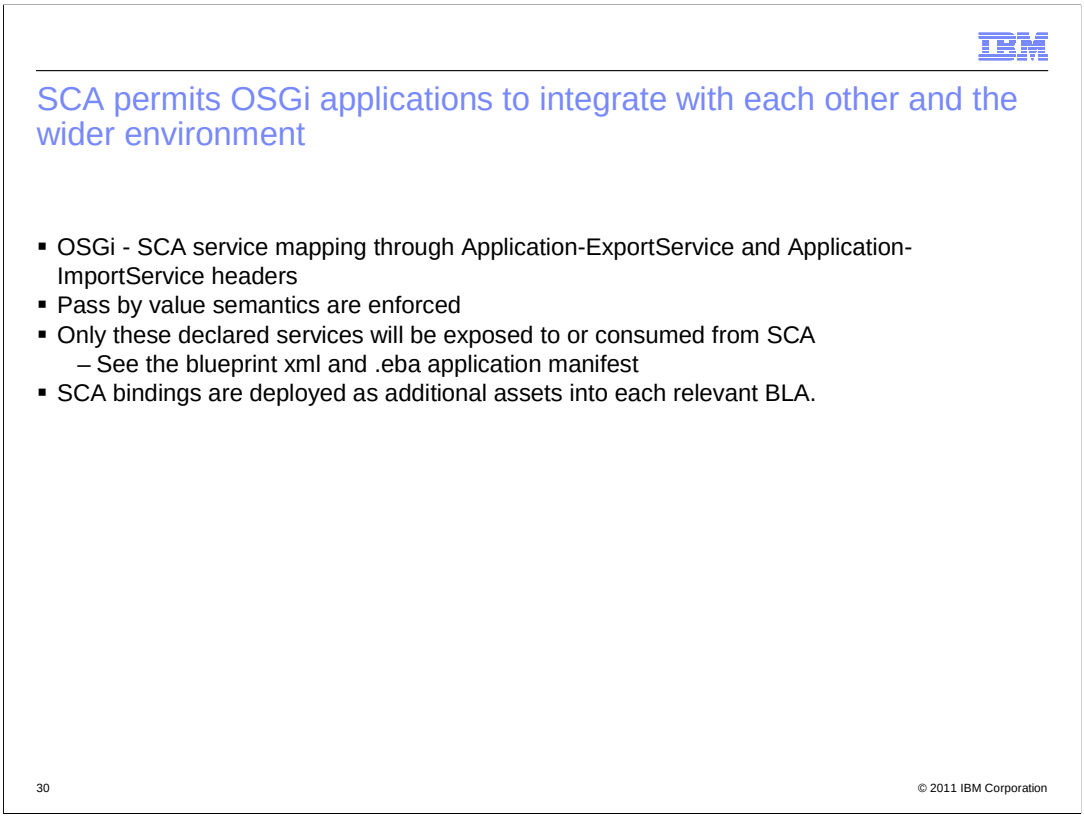
SCA integration: implementation.osgiapp

```
Manifest-Version: 1.0
Application-ManifestVersion: 1.0
Application-Name: Aries Blog
Application-SymbolicName: com.ibm.ws.eba.example.blog.app
Application-Version: 1.0
Application-Content:
 com.ibm.ws.eba.example.blog.api;version="1.0.0",
 com.ibm.ws.eba.example.blog.persistence;version="1.0.0",
 com.ibm.ws.eba.example.blog.web;version="1.0.0",
 com.ibm.ws.eba.example.blog;version="1.0.0"
Use-Bundle: com.ibm.json.java;version="[1.0.0,2.0.0)"
Application-ExportService:
 com.ibm.ws.eba.example.blog.Blog
Application-ImportService:
 com.ibm.ws.eba.example.blog.UserAuthorization
```




```
<component name="com.ibm.ws.aries.example.BlogComponent">
  <service name="bloggingService">
    <interface.java interface="com.ibm.ws.eba.example.blog.Blog" />
    <binding.ws
      port="http://www.blogging.org/BlogService#wsdl.endpoint(BlogService/BlogServiceSOAP)" />
  </service>
  <reference name="userAuthorization">
    <interface.java interface="com.ibm.ws.eba.example.blog.UserAuthorization" />
  </reference>
  <scafp:implementation.osgiapp applicationSymbolicName="com.ibm.ws.aries.example.blog.app"
    applicationVersion="1.0.0" />
</component>
```


This slide shows an OSGi application packaged as an SCA component. Its application manifest in the top left exports a Blog service and imports a UserAuthorization service. The xml shown at the bottom represents the SCA component declaration. Note how the application symbolic name and version appears in both, as well as the imported and exported services.



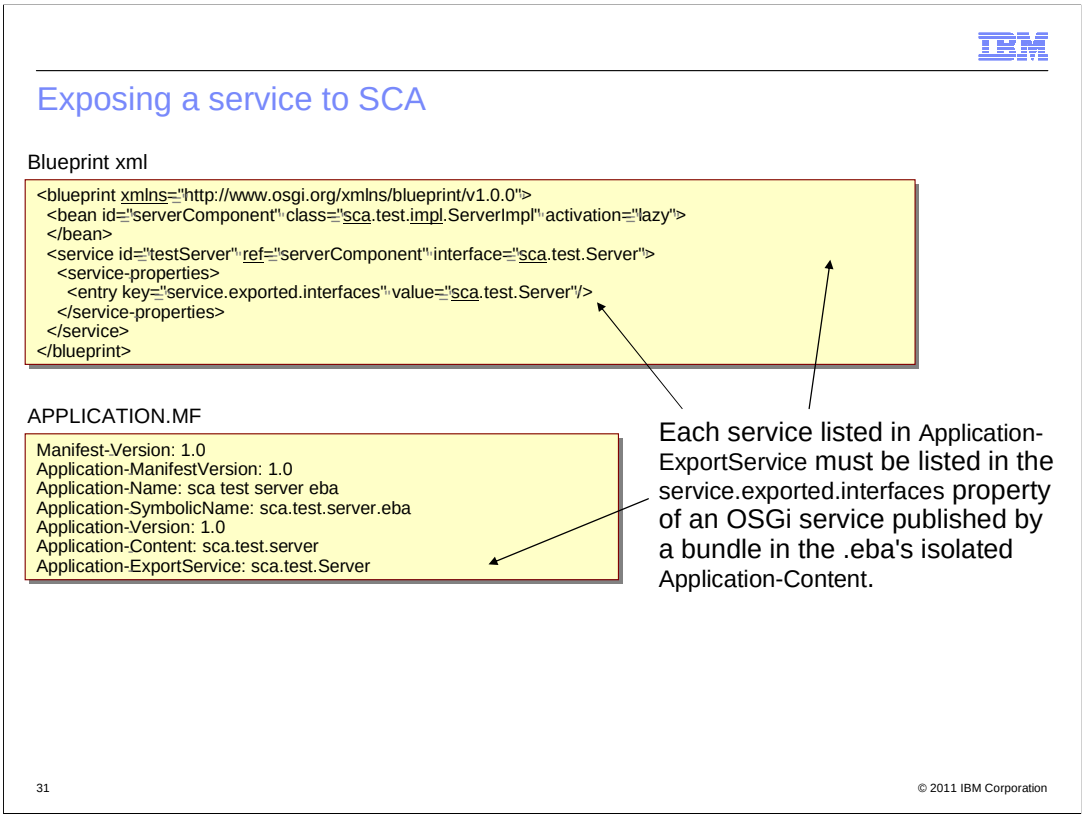
OSGi services are mapped to SCA services through the Application-ExportService and Application-ImportService headers in the .eba application manifest, META-INF/APPLICATION.MF.

OSGi applications can communicate with each other, or any other SCA composites using this mechanism.

SCA can integrate distributed components, and so enforces pass by value semantics.

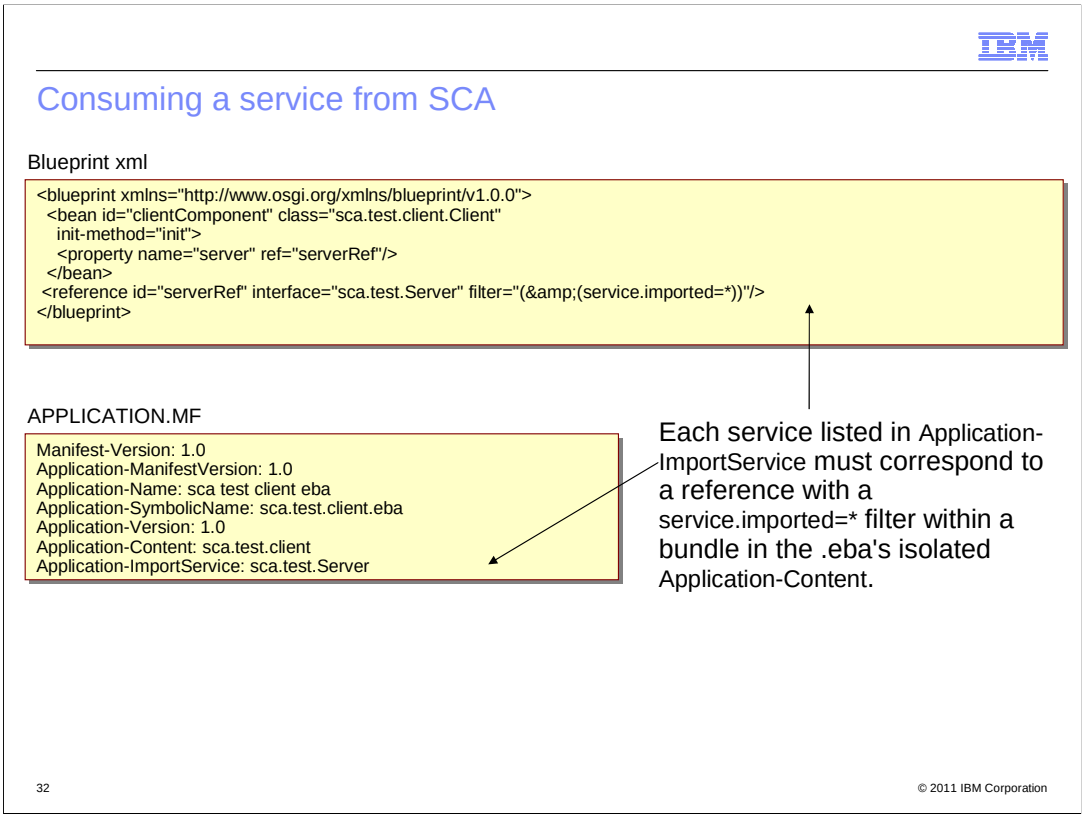
The OSGi application developer needs to know up front which services and references will be exposed to SCA. This knowledge is reflected in the blueprint xml and in the .eba application manifest.

SCA bindings are written as .composite files, packaged into .jar files and deployed as additional assets into each relevant BLA.



This slide shows a service declared in blueprint xml and exported to SCA through the application manifest. Note that each exported service must have a service.exported.interfaces property with at least one interface as its value.

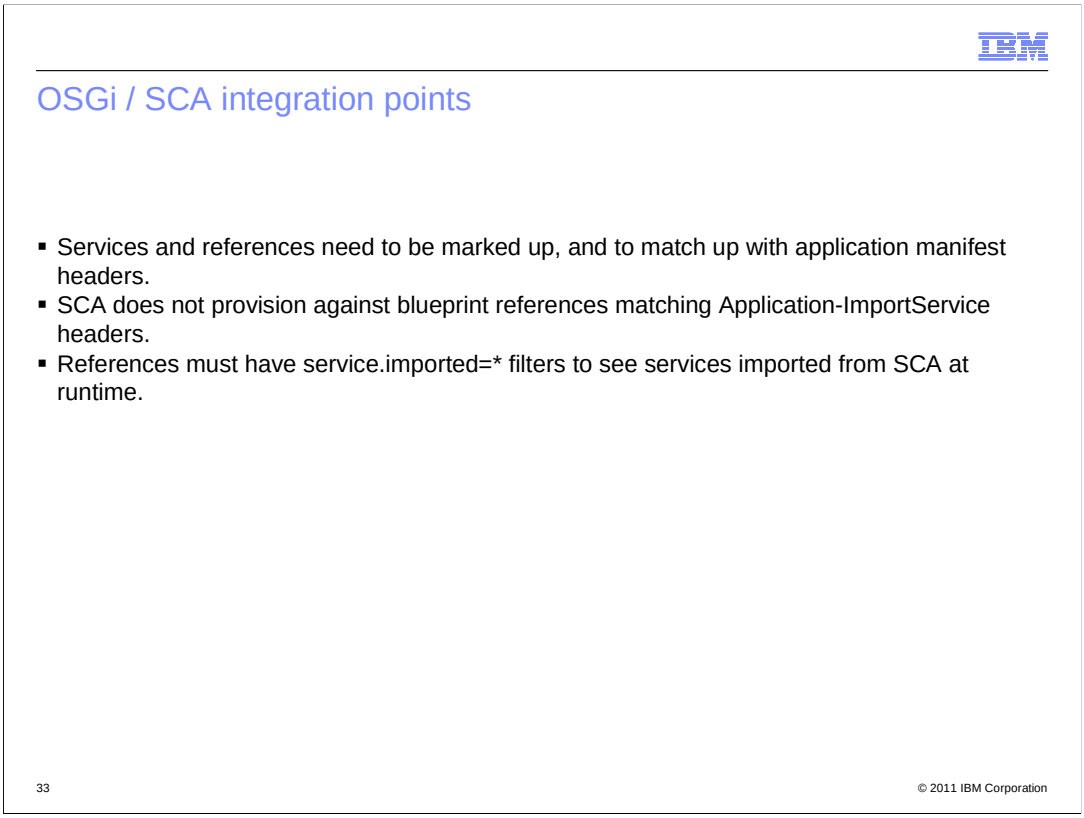
By default, OSGi services use pass-by-reference semantics. By adding the service-exported-interfaces property, the developer indicates that these interfaces are intended to be exposed remotely, and that they follow pass-by-value semantics instead.



Here is the corresponding consumption of a service from SCA.

The service proxy representing an SCA reference advertises itself as a remote service by using the service property service.imported=true. This allows consumers to distinguish local services, which use pass-by-reference semantics, from remote services, which use pass-by-value semantics.

By default, a blueprint reference will not find remote services. It must request them using a filter, as shown above. In this way the developer of the consuming application indicates their awareness that these services are remote and use pass-by-value semantics.



Services and references need to be marked up, and to match up with application manifest headers. Improperly configured applications will frequently fail to resolve, preventing .eba asset import.

When an .eba application is resolved, SCA does not provision services against blueprint references matching Application-ImportService headers.

SCA proxies representing imported services are hidden at runtime from references lacking service.imported=* filters.

Section

ITCAM monitoring support overview

This section will address monitoring support for SCA.

ITCAM for SOA - Features

- Track service-to-service relationship within SCA flow and other SOAP web services
- Measure service response time and availability
- Report aggregated metrics per-operation
- Report aggregated metrics per-authenticated user
- Identify SCA faults

35

The SCA ITCAM for SOA feature lets the user track service to service relationship for SCA and other SOAP web services. It also supports measurement of service availability and response time. It also reports aggregated metrics per operation and per authenticated user and helps to identify SCA Faults.

Section

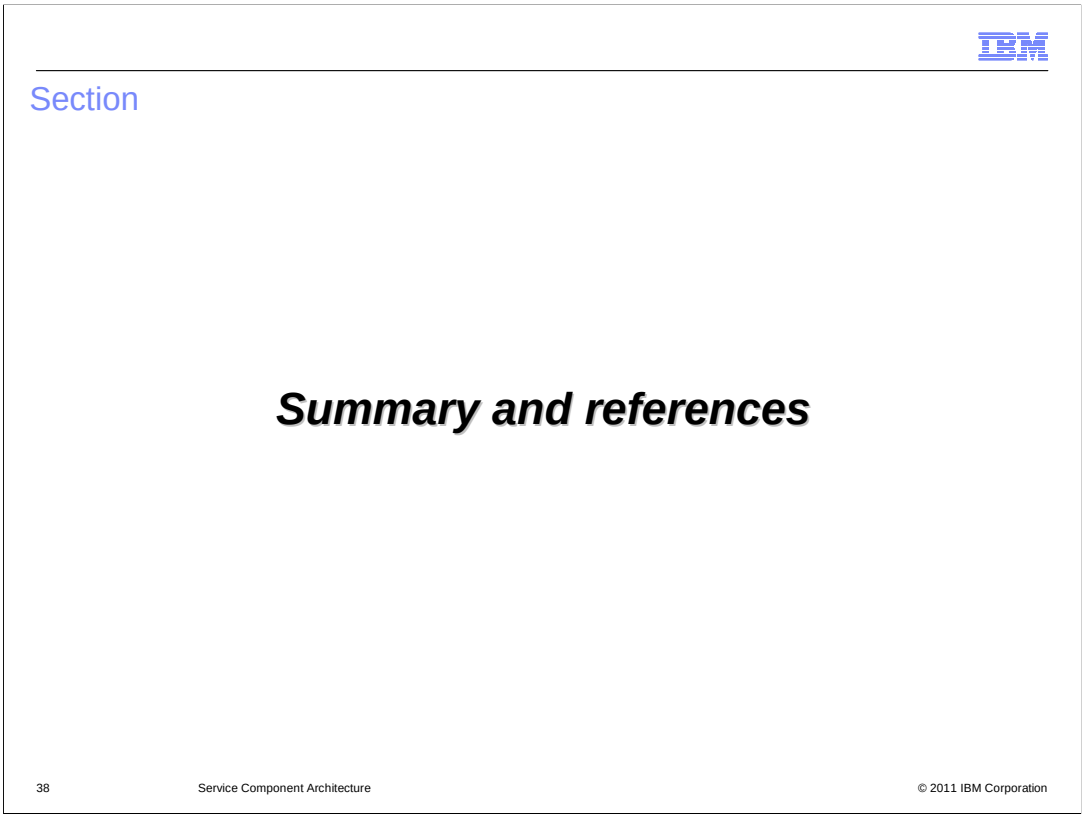
SCA V8.0 release contents

This section will address new SCA features contained in WebSphere Application Server V8.0.

SCA V8 : New function

- SCA Domain can host services from WebSphere Application Server V7 and V8.
- SCA Domain also provides interoperability between services from WebSphere Application Server V7 and V8 over all supported bindings.
- Migration support for SCA applications between WebSphere Application Server V7 and V8.
  - SCA applications will be migrated just like JEE applications.
  - No options or changes in migration tools for SCA.

The main change for SCA in WebSphere Application Server V8 is that the SCA function is now part of the core application server instead of as an add-on feature pack. The SCA domain provides interoperability between services from V7 to V8 over all the supported bindings. SCA migration support was also added in V8. All SCA artifacts will be migrated if you choose to migrate applications during the migration process. There are no new options added in the migration tools that's specific to SCA.



This section will include a summary and references.

SCA makes SOA simpler for developers in that applications are structured as services and components, business logic is not coupled to deployment infrastructure, and there is a wide choice of component kinds allowing you to use the right tool for the job.

As a result, this makes developers stay focused on solving business problems, rather than getting bogged down in the individual complexities of the technologies that connect service consumers and service providers. The SCA support preserves and enhances the value of existing assets which can be exposed as SCA services. SCA can model and integrate assets from a mix of new and existing runtime environments, and provide a consistent system-wide abstraction. Service abstraction captures the SOA design and its mapping to underlying implementations. SCA can apply QoS policies at the service abstraction level across different runtime implementations.

Resources

- Articles on developerWorks
  - http://www.ibm.com/developerworks/websphere

This slide contains some useful resources about SCA provided by IBM.

More resources

- Open Service Oriented Architecture website for SCA V1.0 Specifications
  - http://www.osoa.org/
- OASIS Open CSA website for SCA V1
  - http://www.oasis-opencsa.org/sca
- Apache Tuscany Web site
  - http://tuscany.apache.org/

This slide contains links that point to the SCA specifications and to the Apache Tuscany site.

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 email feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASV8_SCA.ppt

This module is also available in PDF format at: ../WASV8_SCA.pdf

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

Trademarks, disclaimer, and copyright information