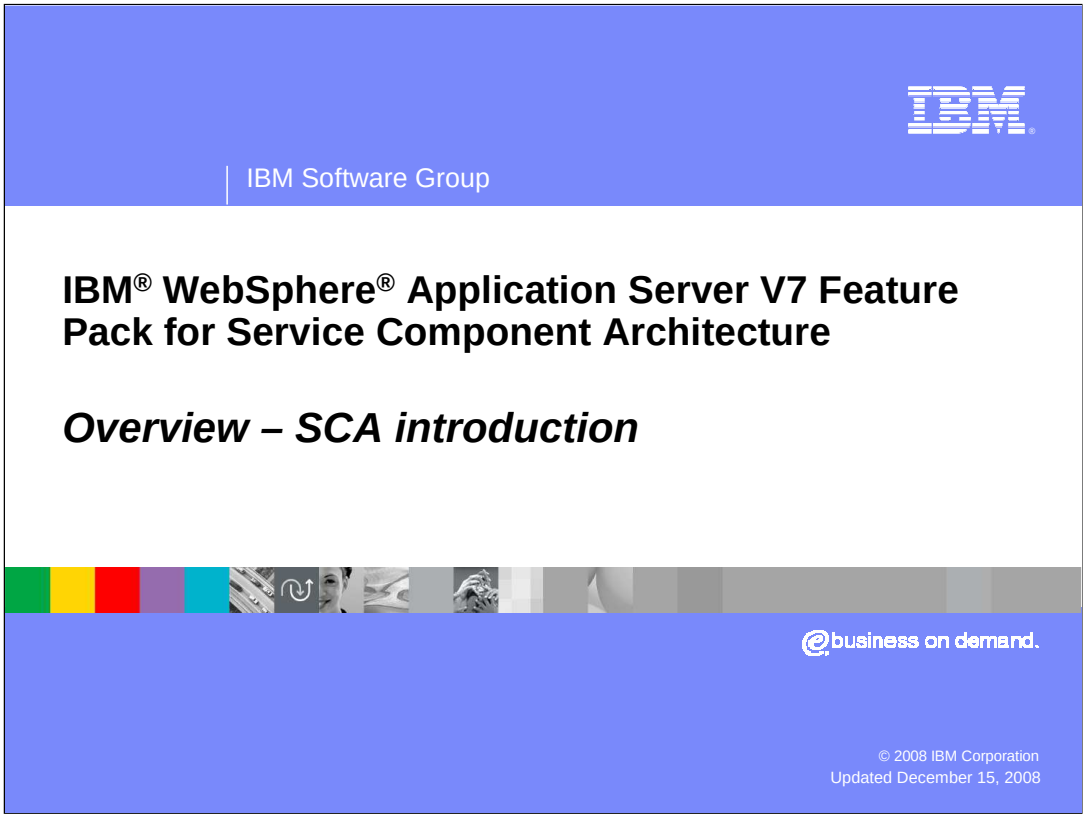


This presentation will cover an overview of the WebSphere Application Server V7.0 feature pack for SCA release.

Agenda

- Introduction to SCA
- SCA overview
- Summary and references

This presentation will start out by briefly going over the introduction to SCA followed by the SCA overview.

Section

Introduction to SCA

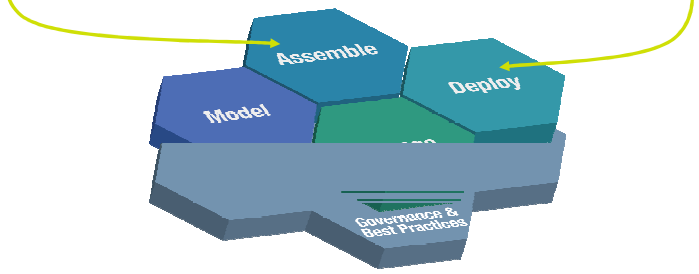
This section will cover an introduction of SCA.

SCA terms and relations to SOA foundation

Service Component Architecture (SCA) addresses the complexity of developing an SOA solution

SCA is the open standard model for service assembly.

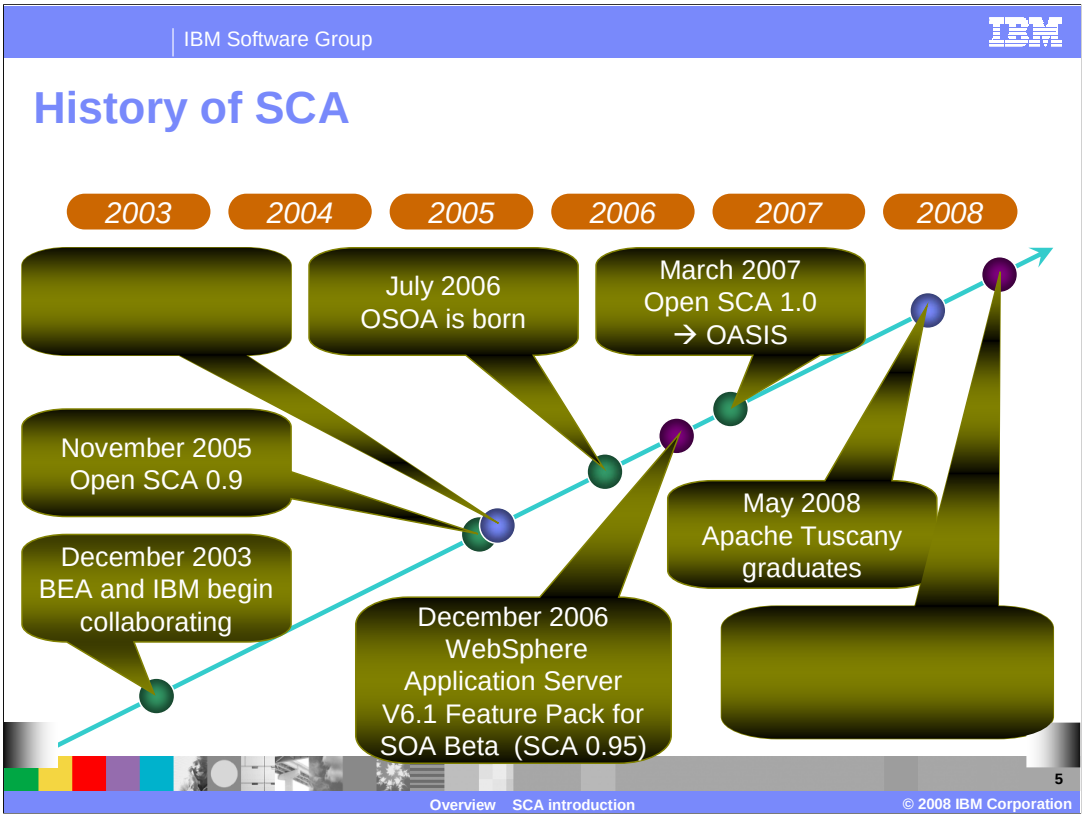
Assemble = develop interfaces, implementations, composites.
Deploy = define, install and run contributions on WebSphere Application Server



An essential characteristic of SOA is the ability to assemble new and existing services to create brand new application that can consist of different technologies. Service Component Architecture (SCA) defines a 'simple' service-based model for construction, assembly, and deployment of services - both existing and new services.

Service component architecture (SCA) is a set of specifications which describe a model for building applications and systems using a Service-Oriented Architecture approach. SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services.

The diagram above depicts the SOA life cycle. The cycle repeats as the processes change due to external business pressures and internal improvements made by observing the process in action.



Service Component Architecture is not really new – WebSphere Process Server pioneered SCA delivering a precursor to the open version in its V6.0 release of runtime and tools. This “Classic SCA,” was used by IBM to seed open specifications endorsed by several vendors at the Open Service Oriented Architecture organization. As seen in the previous slide, SCA represents the SOA programming model of the IBM SOA Foundation and is a strategic technology of the IBM application server and BPM software portfolio.

The initial SCA code contribution and collaboration started back in December 2003 with contribution from BEA and IBM.

The real SCA Project was created in December 2005 in Apache incubator and this version was (SCA 0.9). Next came SCA version 0.95 OSOA, which was a collaboration and an interim delivery implemented by Tuscany and WebSphere Application Server version 6.1 SOA feature pack beta1. The next release SCA version 1.0 came in March 2007, which was an OSOA (Open Service Component Architecture) collaboration and implemented by Tuscany. This is the version implemented in this SCA feature pack.

SCA V1.0 features – What is new?

- Open, multi-vendor support
- Separation of concerns – Protocols moved outside of business logic
- Deployment and packaging model
- Declarative policy
- Inversion of control (injection/annotations)
- Recursive “composite” definition
- Multi-data binding capability
- Heterogeneous - Multi language and container support.
- Extendible – implementation and bindings can be added through extensions

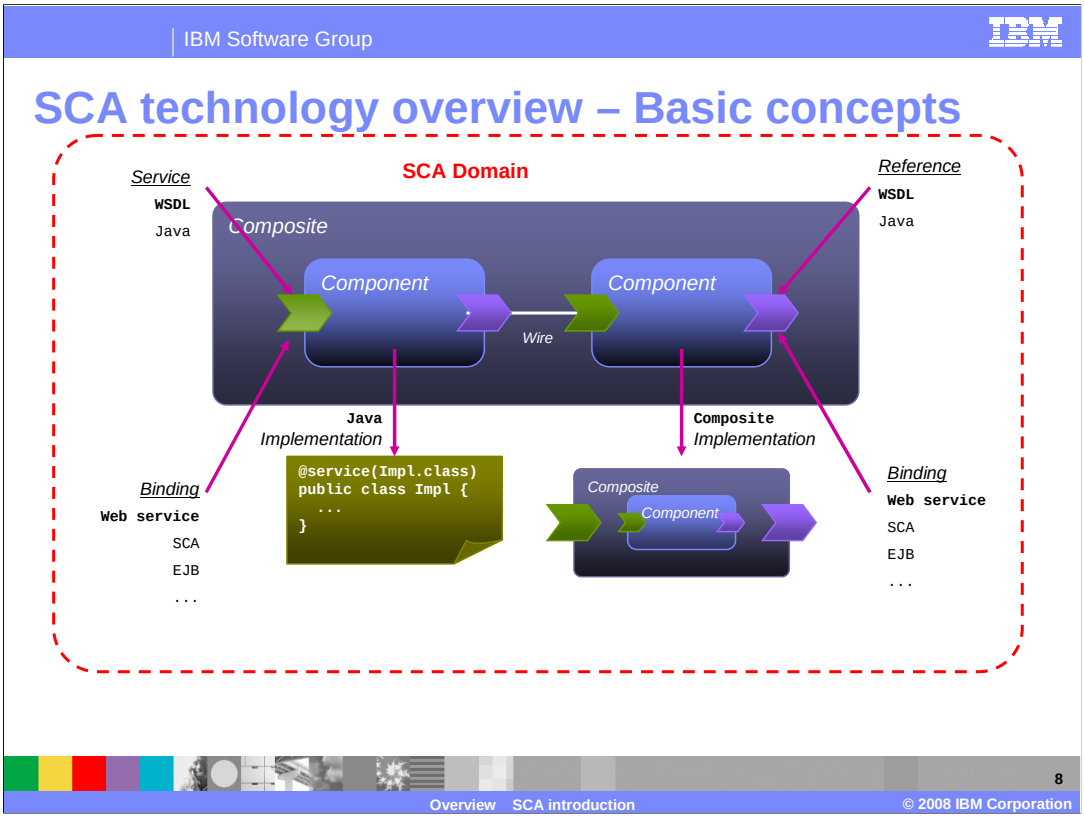
Here are the latest updates to the SCA V1.0 since WebSphere Process Server. Currently, there is an open, multi-vendor support.

Second, there is the concept of separation of concerns where protocols are moved outside of business logic. Then there is Deployment and packaging model which implies WebSphere Application Server Runtime will support SCA natively and not dependent on JEE packaging. Declarative policy which implies policy configuration is native to SCA and exploiting policySets is supported. Inversion of control (injection/annotations) is yet another feature that exploits the new ease of use techniques in Java™ 5. Another feature is recursive composite definition. Multi-data binding capability has also been tested. Heterogeneous - multi language and container support has been added. A composition can have Java, C++, BPEL, and Spring components. Finally the concept of being extendible – You can add implementation and bindings through extensions.

Section

SCA overview

The next section will highlight an overview of SCA

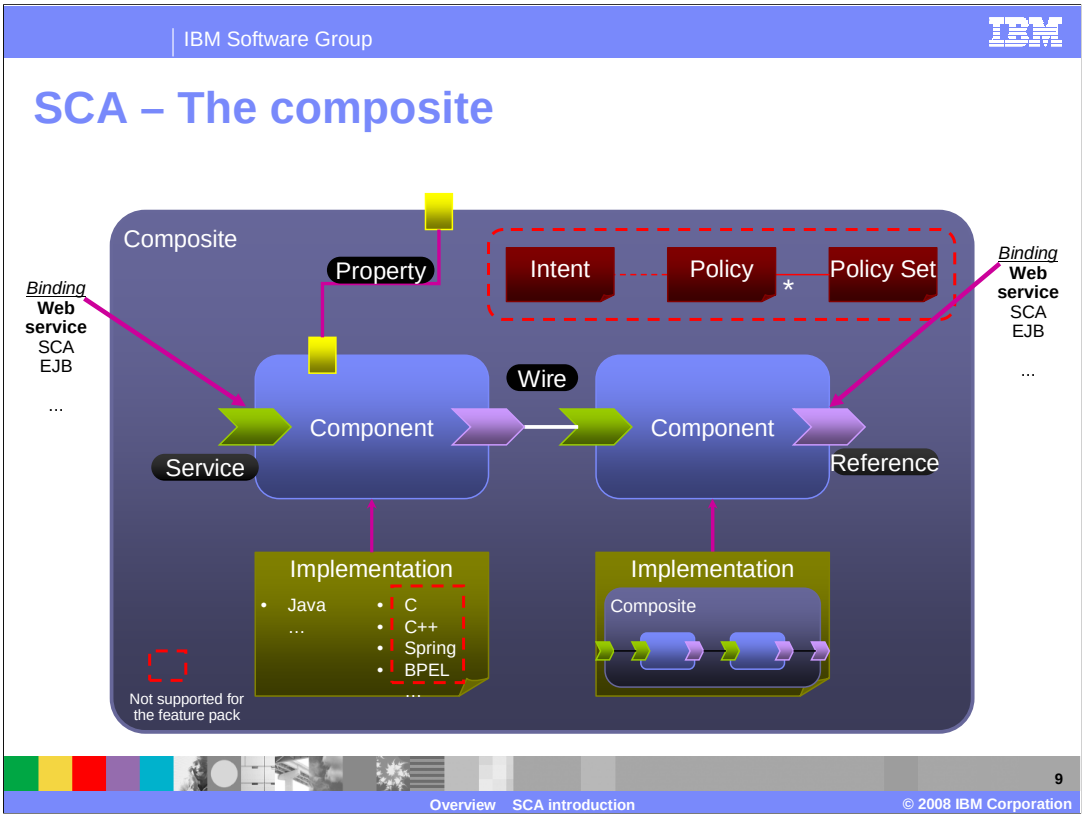


This picture covers the overview of SCA. It shows the basic concepts of what SCA is made up of.

As the name implies, Service Component Architecture is a component model. Service-oriented coarse-grained building blocks are represented as descriptions of ***components***. **Components** describe the ***services*** they provide and the services they depend on or ***reference***. Components also point at the chunk of code which provides the ***implementation*** 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***. The binding are part of the SCDL definition and the business logic (implementation) does not need to be polluted with this detail.

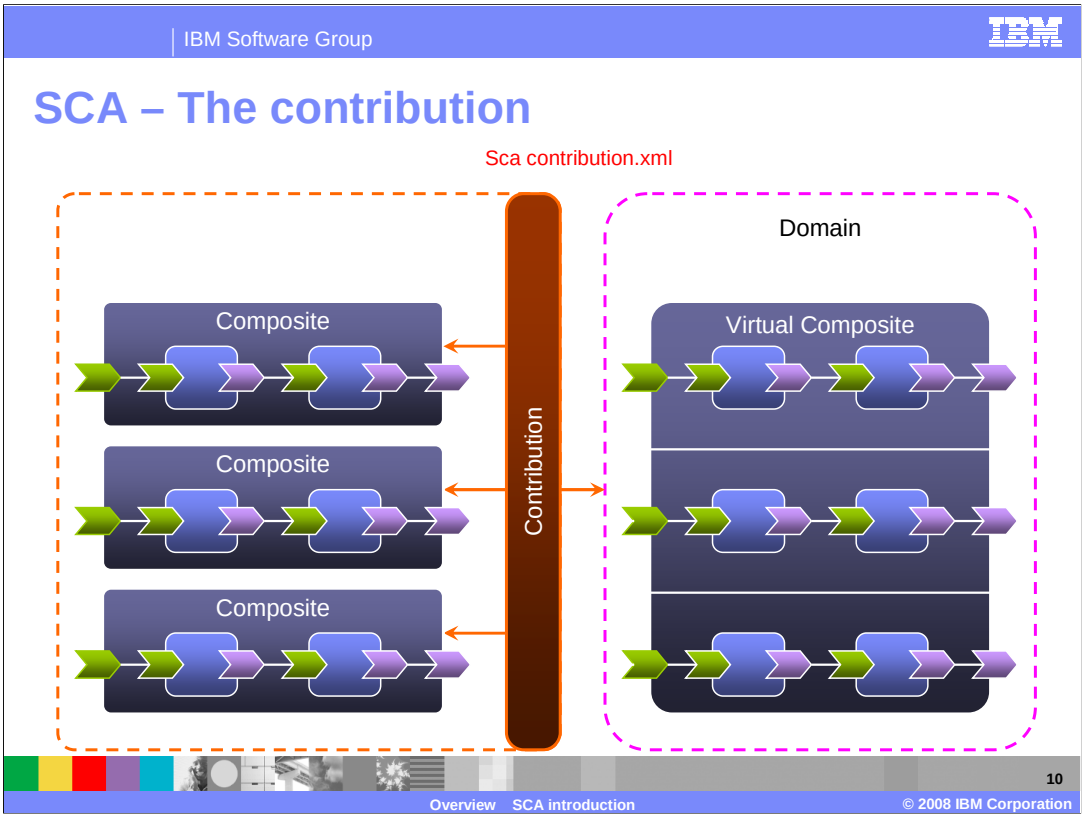
Note that SCA is a language-neutral set of specifications which enable its usage in a variety of application environments. Each application environment will offer the set of implementation, quality of service, and policy features that make sense in that environment. For example, the SCA feature pack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



Since the composite contains components and a component is the basic element of a business function in an SCA application, this picture is re-iterating more of what you have learned in the previous slide.

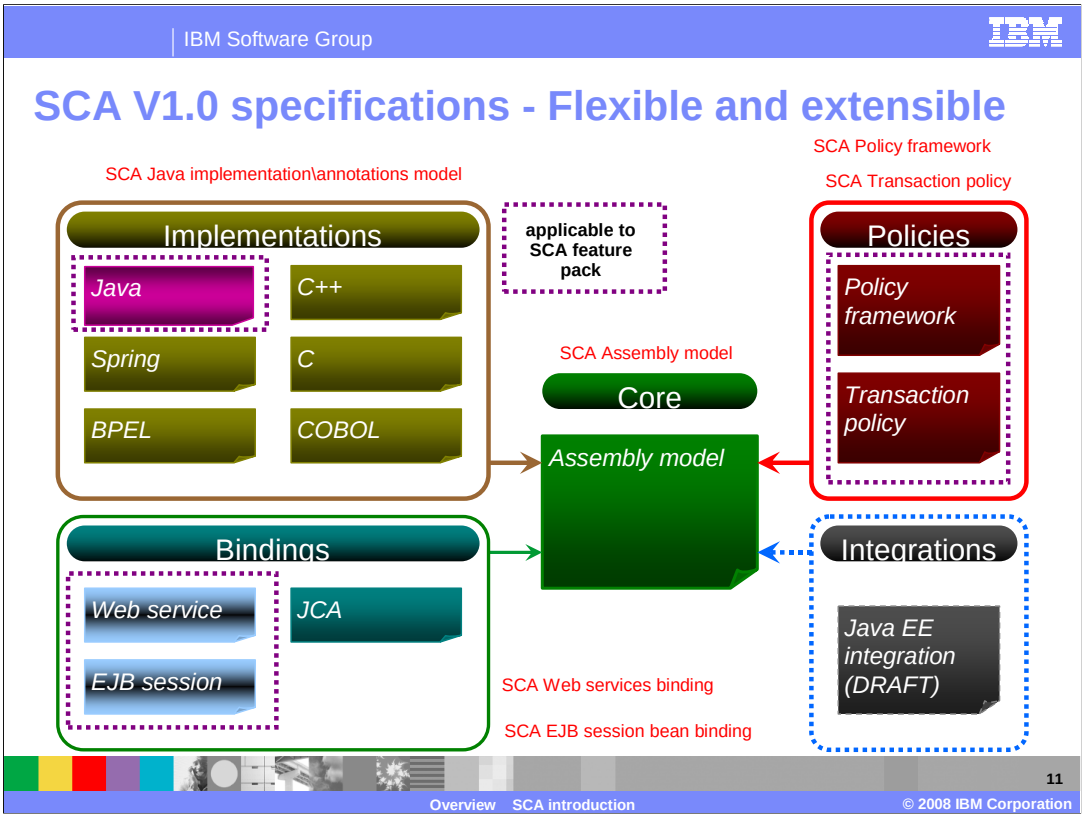
An SCA composite is used to assemble SCA elements in logical groupings and is the basic unit of composition within an SCA System.

An SCA composite contains a set of components, services, and references. Wires interconnect services and references. The SCA composite also contains a set of properties that can be used to configure components. Note that SCA feature pack implementation is only in Java.



Once the composites are defined, they need to be deployed into a runtime environment. 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 composites 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 that 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 their 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 feature pack for WebSphere, the SCA domain and cell have the same scope, and services deployed in the SCA domain are available throughout the WebSphere cell over the default binding.



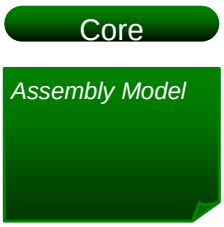
SCA V1.0 has several specifications that allow for flexibility and extensibility when developing SCA applications. There are specifications in each group shown.

The assembly model has an assembly model specification. The policies group has two specifications, namely policy framework and transaction policy.

Implementations have the two Java specifications, and under Bindings there are specifications for each binding.

SCA assembly model

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

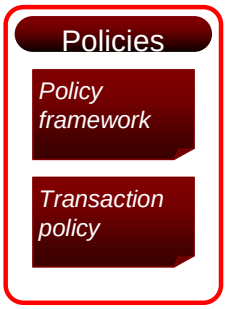


The SCA assembly model specification defines the structure of composite applications. It is a model for the assembly of services, both tightly coupled and loosely coupled. It is also a model for applying infrastructure capabilities to services and to service interactions, including security and transactions.

The SCA assembly model consists of a series of artifacts that 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 policy framework

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

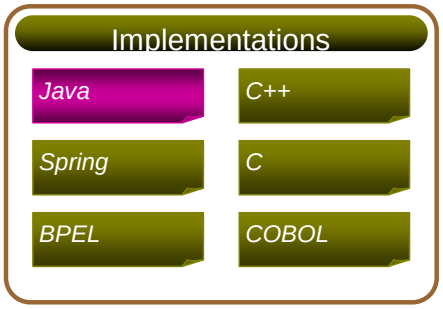


SCA provides a framework to support specification of constraints, capabilities, and QoS expectations from component design through to concrete deployment. This specification describes the framework and its usage. It covers policies and policy languages, to be associated with SCA components.

Basically it is a specification that provides the requirements on how to add infrastructure services to solutions security, transactions, reliable messaging, and so on

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



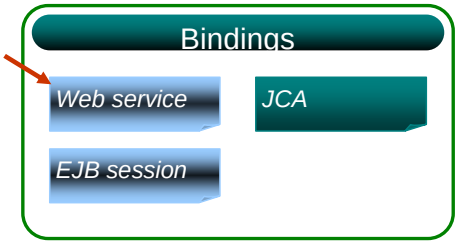
SCA adopts the ease of use improvements provided by JEE 5 annotations and modern container design supporting inversion of control or dependency injection. SCA Java implementations have the benefit of declaring service definitions using annotations, having SCA service references, policy, and properties 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; the Java component implementation specification, which extends the SCA assembly model, and the Java common annotations and APIs specification. Together, these models provide the specifications for writing business services in particular languages like Java, C++, BPEL, PHP, and spring.

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

SCA Web service binding

- Applies to the services and references of components
- Defines how:
  - a service can be exposed as a Web service
  - a reference can access a Web service
- Designed for SOAP based, WS-I compliant Web services



There are specifications for Web service binding, EJB binding, and JCA binding. 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).

Section

Summary and references

This next section will provide a summary of this presentation.

Summary

- SCA is not a competitor to the other fine-grained service implementation languages and frameworks, but rather a complementary and inclusive technology

In summary, SCA is not a competitor to the other fine-grained service implementation languages and frameworks, but rather a complementary and inclusive technology.

References

- Open Service Oriented Architecture Web site
  - http://www.osoa.org/
- Apache Tuscany Web site
  - http://incubator.apache.org/tuscany/

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_WASv7SCA_Overview_sca_Introduction.ppt

This module is also available in PDF format at:
../WASv7SCA_Overview_sca_Introduction.pdf

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

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

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

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.