Service Component Architecture Java EE Integration Specification Version 1.1

Working Draft 06

14 September 2009

Specification URIs:

This Version:

http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.html
http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.doc
http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.pdf

Previous Version:

Latest Version:

http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.html
http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.doc
http://docs.oasis-open.org/sca-j/sca-jee-1.1-spec-wd06.pdf

Latest Approved Version:

Technical Committee:

OASIS Service Component Architecture / J (SCA-J) TC

Chair(s):

David Booz, IBM
Mark Combellack, Avaya

Editor(s):

Anish Karmarkar, Oracle
David Booz, IBM
Mike Edwards IBM
Plamen Pavlov SAP

Related work:

This specification replaces or supercedes:

- Service Component Architecture Java EE Integration Specification Version 1.00, May 13, 2008

This specification is related to:

- Service Component Architecture Assembly Model Specification Version 1.1
- Service Component Architecture Policy Framework Specification Version 1.1
- Service Component Architecture Java Common Annotations and API Specification Version 1.1

Declared XML Namespace(s):

http://docs.oasis-open.org/ns/opencsa/sca/200903

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 5
Field Code Changed
Deleted: 5
Deleted: 5
Deleted: 4
Deleted: 4
Deleted: 4
Field Code Changed
Deleted: 0
Deleted: 712
Deleted: http://docs.oasis-open.org/ns/opencsa/sca-j/200808
Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

Abstract:

This document specifies the use of Service Component Architecture (SCA) within and over the scope of applications and modules developed, assembled, and packaged according to the Java Platform Enterprise Edition (Java EE) specification.

Status:

This document was last revised or approved by the OASIS Service Component Architecture / J (SCA-J) TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/sca-j/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/sca-j/ipr.php.

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/sca-j/.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS" is a trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Deleted: , **DisplayText cannot span mo** are

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Table of Contents

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

1 Introduction

This document specifies the use of Service Component Architecture (SCA) in relation to applications and modules developed, assembled, and packaged according to the Java Platform Enterprise Edition (Java EE) specification.

Java EE is a standard for Java-based enterprise applications. While it offers a rich set of technologies, it does not define important concepts that are inherently required in service oriented architectures such as

- Extensibility of component implementation technologies
- Extensibility of transport and protocol abstractions
- A concept of cross-application assembly and configuration

Service Component Architecture provides a standardized and extensible assembly language and methodology that can be layered on top of existing component models and runtimes.

The Java EE client and implementation specification focuses on the relationship of SCA's concepts of assembly, implementation type, and deployment to Java EE structures, it is also expected that SCA application assemblies will combine Java EE components with other technologies. Examples of technologies for which SCA integration specifications have been completed include BPEL and the Spring framework. It is expected that an **SCA enabled Java EE runtime** will offer a palette of technologies for integration in an SCA assembly.

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 a level of integration between SCA components and Java EE applications. These bindings are addressed in separate specifications:

- The EJB Session Bean Binding Specification [2] describes the exposure and consumption session beans
- The JMS Binding Specification [9] describes the exposure and consumption of Java Message System (JMS) destinations
- The specification for Java Connectivity Architecture (JCA) adaptors describes connectivity to applications using the JCA specification [12].

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in **[RFC2119]**.

1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

[1] Java ™ Platform, Enterprise Edition Specification Version 5 http://jcp.org/en/jsr/detail?id=244 http://java.sun.com/javaee/5

[2] SCA EJB Session Bean Binding V1.00 http://www.oasis-open.org/apps/org/workgroup/sca-j/download.php/25552/sca-ejbbinding-draft-20071004.pdf

[3] SCA Assembly Model V1.00 http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-CD-01.pdf

[4] SCA Java Common Annotations and APIs V1.00

Comment: This might need more work.

Formatted: Italian Italy

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

http://www.oasis-open.org/committees/download.php/29078/sca-javacaa-1.1-spec-WD04.pdf

[5] SCA Java Component Implementation V1.00 http://www.oasis-open.org/apps/org/workgroup/sca-j/download.php/25527/sca-javaci-draft-20070926.doc

[6] SCA Policy Framework V1.00
http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf

[7] Java Servlet Specification Version 2.5
http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index.html

[8] Enterprise JavaBeans 3.0
http://jcp.org/en/jsr/detail?id=220

[9] SCA JMS Binding specification
http://www.oasis-open.org/committees/download.php/29083/sca-jmsbinding-1.1-spec-WD-04.pdf

[10] SCA Transaction Policy specification
http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf

[11] Norm Walsh. XML Catalogs 1.1. OASIS Committee Specification, OASIS, July 2005
http://www.oasis-open.org/committees/download.php/14041/xml-catalogs.html

[12] SCA JCA Binding specification
http://www.oasis-open.org/committees/download.php/29071/sca-jcabinding-1.1-spec-WD-02.pdf

[13] **JAX-WS Specification (JSR-224)**
http://www.jcp.org/en/jsr/detail?id=224

Formatted: English U.S.

Formatted: English U.S.

Formatted: English U.S.

1.3 Non-Normative References

[TBD] TBD

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

2 Scenarios

In this document, the term **SCA-enabled Java EE runtime** is used to refer to a Java EE runtime that supports deployment and execution of SCA-enhanced Java EE applications as well as SCA-enhanced Java EE modules (see also section 5).

An SCA-enabled Java EE runtime that fully implements this specification supports the use cases defined in appendix A. These demonstrate the following scenarios:

2.1 Consume SCA-exposed services from Java EE components

For example, a Java EE web component should be able to consume a service implemented by an SCA service component, either by using SCA constructs in the implementation of the web component implementation or via an EJB reference in combination with an EJB binding on the SCA service component as defined in the EJB Session Bean Binding [2].

2.2 Deploy SCA Components as a Part of a Java EE application

SCA applications will typically combine Java EE components with components using other implementation technologies, such as BPEL. This specification enables the deployment of components implemented in these non-Java EE technologies as part of a Java EE application, taking advantage of whatever tooling and infrastructure support exists for the deployment and lifecycle management of Java EE applications. Such components are treated as running in an unmanaged environment and cannot not rely on Java EE features (access to java:comp/env, etc.)

2.2.1 Use Recursive SCA Assembly in Java EE Applications

SCA Assembly provides the means to define sophisticated application assemblies for Java EE applications.

2.3 Use Java EE Archives as Service Component Implementation

This specification enables the creation of SCA applications where one or more components are implemented by Java Java EE archives, so that they can be wired to each other and to components implemented using other technologies. This use-case takes a high-level view of the Java EE application as a single SCA component implementation, providing services and consuming.

2.4 Use Java EE components as Service Component Implementations

The recursive assembly model of SCA provides rich means of configuration and re-use of service components that can be implemented by a wide variety of implementation types. Session beans and other Java EE components are the Java EE component implementation model and these can also serve as SCA service component implementations.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

3 Overview of SCA Assembly in a Java Enterprise Edition Environment

Comment: DAB: This section needs to be cleaned up, but that cleanup is not central to this restructure. There may be too many concepts introduced here. I think this section should simply expand on the use cases in section 2.

This specification defines a model for using SCA assembly in the context of a Java EE runtime that enables integration of SCA artifacts with Java EE technologies on a fine-grained component level as well as use of complete Java EE applications and modules within a larger SCA assembly in a coarse-grained large system approach.

The Java EE specifications define programming models for a number of application component types, such as Enterprise Java Beans (EJB) and Web applications, that are packaged in modules and that are assembled into Java EE applications using a Java Naming and Directory Interface (JNDI) based system of component level references and component naming.

Names of Java EE components are scoped to the application package (including single module application packages), while references, such as EJB references and resource references, are scoped to the component and bound in the Environment Naming Context (ENC).

In order to reflect and extend this model with SCA assembly, this specification introduces the concept of the Application Composite (see section 5.1.3) and a number of SCA implementation types, such as the EJB implementation type and the Web implementation type, that represent the most common Java EE component types (see section 6).

Implementation types for Java EE components associate those component implementations with SCA service components and their configuration, consisting of SCA wiring and component properties as well as an assembly scope (i.e. a composite). Note that the use of these implementation types does not create new component instances as far as Java EE is concerned. Section 6.1 explains this in more detail.

In terms of packaging and deployment this specification supports the use of a Java EE application package as an SCA contribution, adding SCA's ***domain*** metaphor to regular Java EE packaging and deployment.

In addition, the Java EE implementation type provides a means for larger scale assembly of systems in which a Java EE application forms an integrated part of a larger assembly context and where it is viewed as an implementation artifact that may be deployed several times with different component configurations. See section 7 for more details.

Through the extended semantics of the application composite and by virtue of the component type definition for the Java EE implementation type, both approaches, local assembly within the Java EE package as well as a coarse-grained use of a Java EE application, can be combined in a straightforward way.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

4 Scope and Limitations of the Specification

Various parts of this specification are limited with respect to what version of Java EE specifications they refer and apply to.

- <implementation.ejb/> is only defined for EJB version 3 and higher.
- <implementation.web/> is only defined for Servlet JSP specification version 2.5 and higher.
- <implementation.jee/> is only defined for Java EE archives that are compliant to Java EE 5 and higher

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

5 SCA-enhanced Java EE Archives

The following sections provide a detailed description of how to make use of SCA concepts within and over the scope of Java EE applications and Java EE modules.

We will use the term **SCA-enhanced Java EE application** when referring to Java EE applications that are composed from a mix of Java EE artifacts as well as SCA artifacts and additional implementation artifacts. Similarly we will use the term **SCA-enhanced Java EE module** to refer to Java EE modules that have been extended with SCA artifacts and implementations, and we will use the term **SCA-enhanced Java EE archive** when referring to either construct.

5.1 Assembly and Deployment of SCA-enhanced Java EE Archives

In this section we will see how to apply SCA assembly concepts when assembling and deploying SCA-enhanced Java EE applications. The SCA assembly specification [3] defines a language and model to make effective use of the implementation types and bindings described in this specification and other specifications (as far as supported by the target runtime environment).

The reader should be familiar with the concepts and terms of the SCA assembly specification [3].

In order to provide a visual representation of assembly and deployment related examples, we use the following graphical notation:

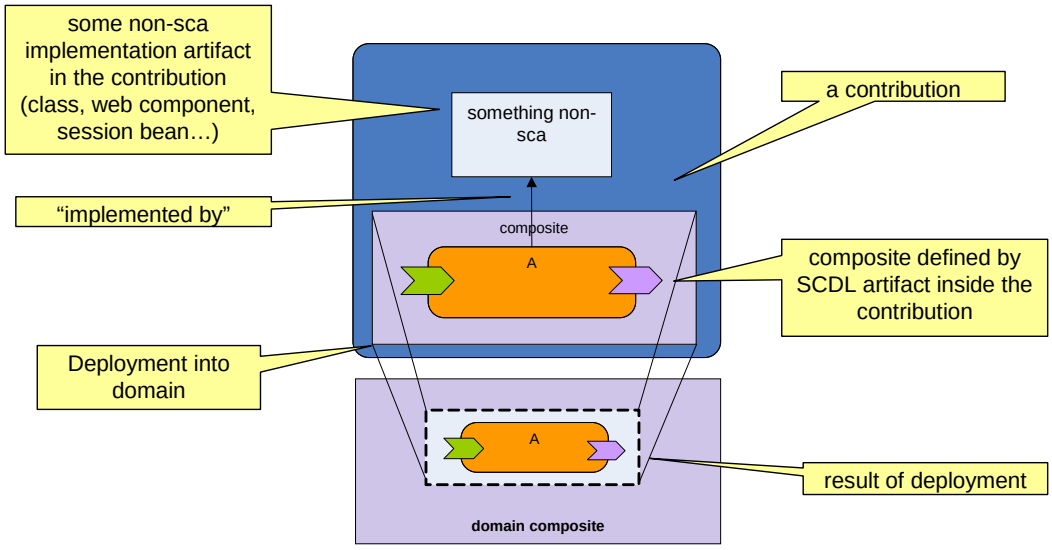


Figure 1: Graphical notation for SCA enhanced Java EE

Note: Java EE archives, SCA-enhanced or not, may also be used as service component implementations via the Java EE implementation type. See section 7 for details.

5.1.1 Java EE Archives as SCA Contributions

A Java EE archive, for example a Java EE application or a Java EE module (a Web application, an EJB module), can be used as an SCA contribution (see [3]).

We will use the term **Java EE contribution** for a Java EE archive that is used as an SCA contribution.

Comment: This section really should come later in chap 5 but then the flow of the example is messed up.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

A Java EE archive that is being used as an SCA contribution MUST still be valid according to Java EE requirements, containing all required Java EE artifacts (e.g., META-INF/application.xml in an .ear file) - if it is not valid, then the SCA runtime MUST throw an error and MUST NOT run the archive..

Many Java EE implementations place some additional requirements on deployable archives, for instance, requiring vendor specific deployment descriptors. A Java EE archive that is an SCA contribution should also fulfill these additional implementation specific constraints when necessary.

As with any regular SCA contribution, a Java EE contribution may be associated with a set deployment composites that can be deployed to the SCA domain. A Java EE archive that is being used as an SCA contribution indicates its deployment composites, as well as any imported or exported SCA artifacts, by providing an SCA Contribution Metadata Document at

META-INF/sca-contribution.xml

The SCA Assembly Specification [3] describes the format and content of the contribution metadata.

A ***META-INF/sca-contribution-generated.xml*** file may also be present. An SCA-enabled Java EE runtime MUST process these documents, if present, and deploy the declared composites.

Implementations that support an install step separate from a deployment step SHOULD provide the Add Deployment Composite function to allow composites to be added to an installed SCA-enhanced Java EE archive without modifying the archive itself. In this case, the composites are passed in by value. Such a feature is useful because it allows the deployer to complete the SCA wiring by adding in the composite.

The deployment of a set of deployment composites from a Java EE contribution, including the exposure of components in the domain and the addition of external bindings, takes place in addition to Java EE deployment: every Java EE component in the application's deployment descriptors (including EJB3 implied deployment descriptors) is deployed, whether it is mentioned in an SCA composite or not. See also section 6.5.

Irrespective of how many SCA deployment composites are deployed from a Java EE contribution, only one Java EE deployment will occur.

For example, deployment of the composite below and the following contribution metadata document would lead to the deployment of a service component named ***org.sample.Accounting*** into the domain composite. See section 6 for a description of how to use EJB implementations as SCA component implementations. This component exposes a single service AccountReporting that is implemented by the EJB session bean ***module.jar#RemotableBean***, assuming that the session bean ***RemotableBean*** has one business interface by the name ***services.accouting.AccountReporting*** (see also 5.1.2):

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="AccountingToDomain"
            targetNamespace="http://www.sample.org"
            xmlns:sample="http://www.sample.org"
            xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <component name="org.sample.Accounting">
         <implementation.ejb ejb-link="module.jar#RemotableBean"/>
   </component>
</composite>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<contribution xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   xmlns:sample="http://www.sample.org">

   <deployable composite="sample:AccountingToDomain"/>
</contribution>
```

Deleted: 712

Formatted: French France

Deleted: 712

Formatted: French France

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Using the diagram notation introduced above we get

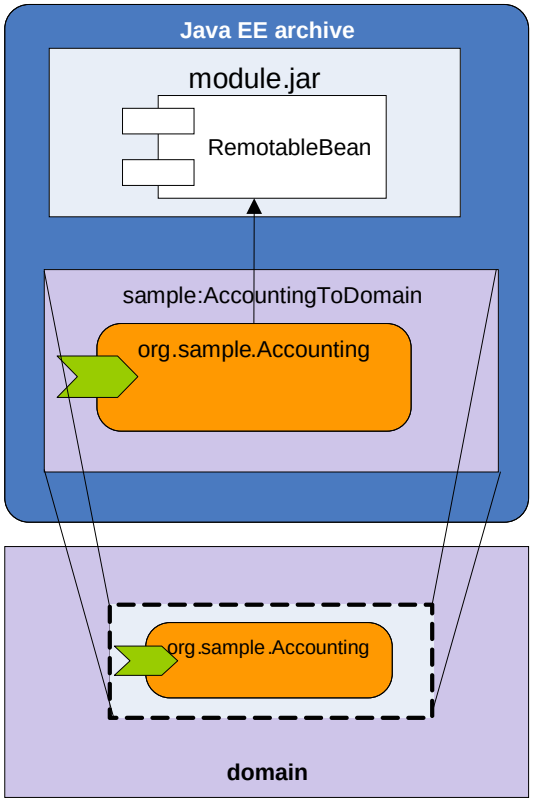


Figure 2: Example of contributing a Java EE EJB as a component in the SCA domain

This kind of assembly is very practical for rapidly achieving domain exposure of service components implemented by Java EE contributions.

5.1.2 Local Assembly of SCA-enhanced Java EE Applications

On an SCA-enabled Java EE runtime SCA assembly extends Java EE assembly by providing a framework for introducing additional implementation types, bindings, and wiring capabilities into a Java EE application. For instance, SCA makes it possible to wire an EJB component to a BPEL process. Such application internal wiring, between standard Java EE components and SCA components whose implementations may not be Java classes (supported implementation and binding types will, of course, vary from implementation to implementation) is a major benefit of SCA.

Users should take advantage of this benefit, which allows a separation of the application's internal wiring from the components that the application wishes to expose in the domain, in particular, to encapsulate the internal construction of service assemblies. The recursive composition model in SCA enables this pattern, as shown in the following diagram:

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

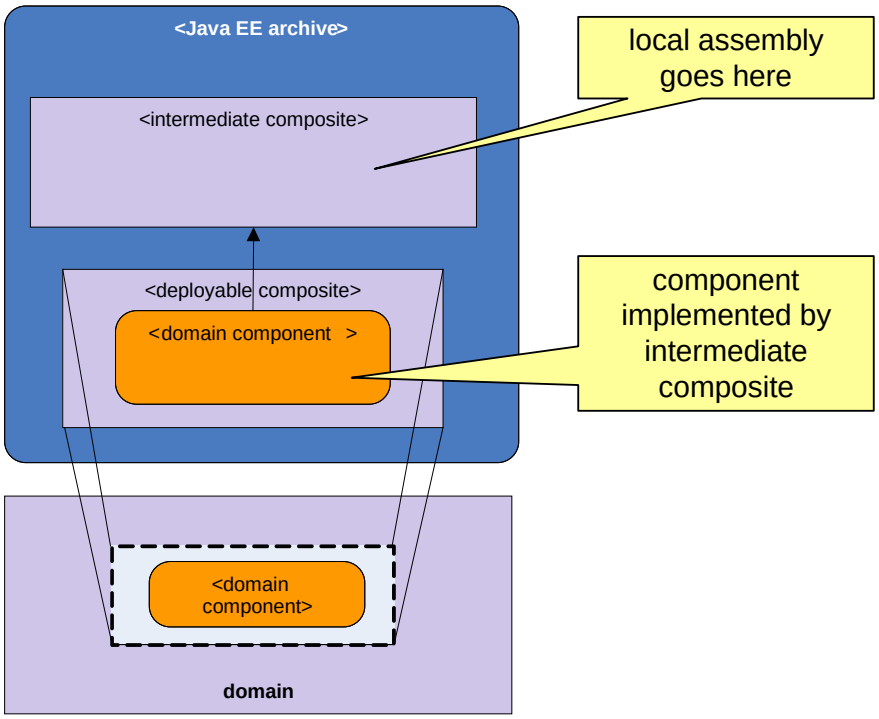


Figure 3: Recursive Composition using Composites

In order to simplify the implementation of this pattern and in order to provide a developer-friendly implementation model, SCA enabled Java EE runtimes support an ***application composite*** as described in the next section.

5.1.3 The Application Composite

A Java EE contribution may contain an SCA composite, the ***application.composite*** file, that supports the use of SCA programming model within the scope of the Java EE archive.

The application composite has two characteristics:

1. The application composite may be directly or indirectly used as a composite implementation or by inclusion into some deployment composite.

   However, if that is not the case, the SCA implementation MUST logically add a deployment composite into the Java EEarchive. The deployment composite contains a single component, named after the application composite, that uses the application composite as its implementation. In addition this deployment composite MUST be deployed into the domain. Consequently the services and references of the application composite are exposed into the domain.

2. The application composite supports automatic (logical) inclusion of SCDL definitions that reproduce the component type of the Java EE implementation type into the composite's component type. See section 7.2 7.1.3 for a detailed description of the includeDefaults feature.

Application archives (.ear files) that are used as SCA contributions define the application composite by a composite definition at

META-INF/application.composite

in the Java EE application package.

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

The Java EE specification also supports deployment of single application modules. This method of deployment is particularly popular for web application modules but also used for EJB modules and resource adapter modules. Single modules are treated as a simplified application package. The application composite for these archives is defined at

WEB-INF/web.composite

for Web Modules, and in

META-INF/ejb-jar.composite

for EJB modules.

For example the following ***application.composite*** file configures a property of a session bean ***RemotableBean*** and exposes its remote interface service to the domain using a default web service binding.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="accounting_application"
   targetNamespace="http://www.sample.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <service name="AccountReporting"
         promote="beancomponent/AccountServiceRemote">
         <binding.ws/>
   </service>

   <component name="beancomponent">
         <implementation.ejb ejb-link="module.jar#RemotableBean"/>
         <property name="currency">EUR</property>
   </component>
</composite>
```

By definition the application composite implies the generation of a deployment composite that deploys a single component to the domain as shown in the following figure:

Deleted: 712

Comment: There is normative text missing here - what is the name of the component in the domain obtained when this procedure applies?

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

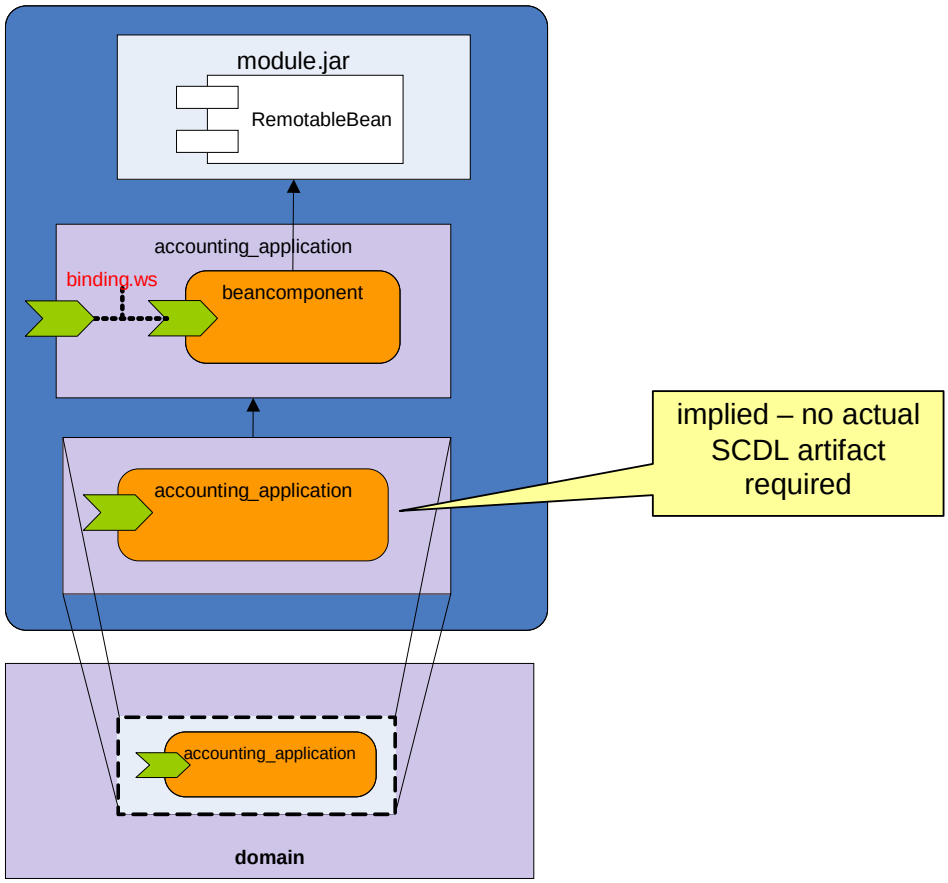


Figure 4: Automatic generation of Deployment composite for application composite

The EJB-implemented service component ***beancomponent*** may be modified in a later version so that it makes use of another service component ***othercomponent*** (whose implementation technology we ignore for the sake of the example). It can do so by modifying the application composite but this modification is not seen in the SCA Domain. Now the example looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="accounting_application"
   targetNamespace="http://www.sample.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <service name="AccountReporting"
         promote="beancomponent/AccountServiceRemote">
         <binding.ws/>
   </service>

   <component name="beancomponent">
         <implementation.ejb ejb-link="module.jar#RemotableBean"/>
         <property name="currency">EUR</property>
         <reference name="other" target="othercomponent"/>
   </component>

   <component name="othercomponent">
```

Deleted: *712*

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
            <implementation.ejb ejb-link="..."/>
        </component>

    </composite>
```

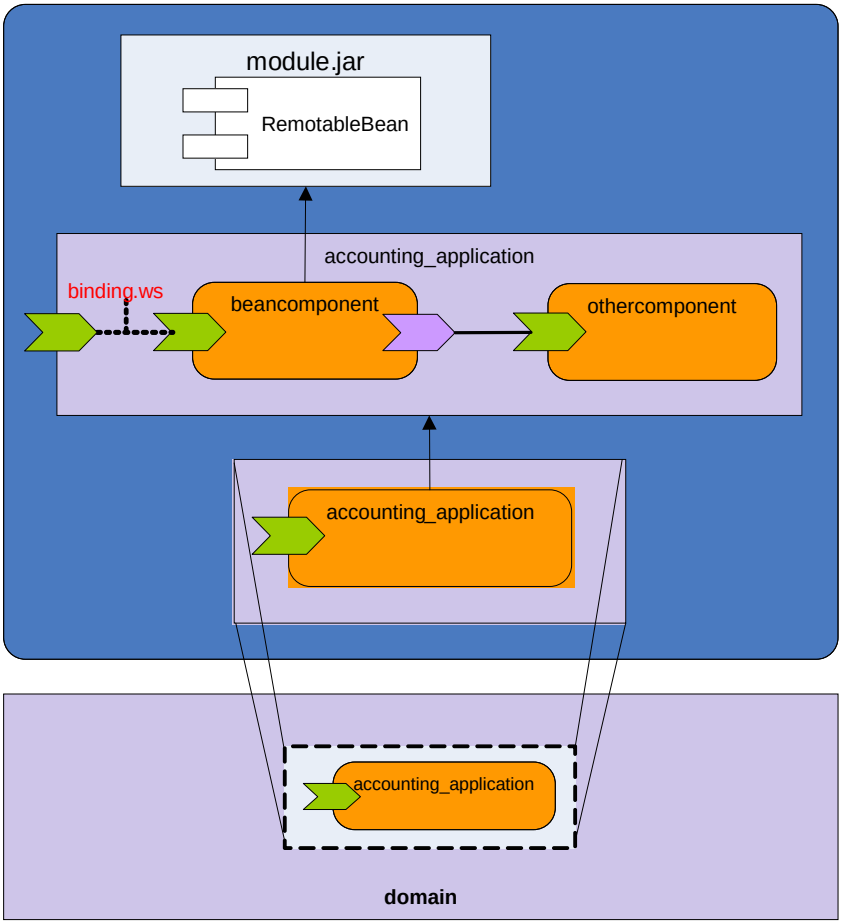


Figure 5: Effect of updating the Application composite

5.1.4 Domain Level Assembly of SCA-enhanced Java EE Applications

As applications are deployed into the SCA domain as components, they make themselves available for SCA wiring. In this way, SCA allows Java EE applications to do cross application wiring. To illustrate this, we extend the previous example by introducing a TicketSystem component into the SCA Domain. The TicketSystem component wants to make use of the accounting application that was deployed in the previous section. The TicketSystem is a web facing application component that is implemented by the SCA application composite in a Web Module, call web.war. The Web Module contains a reference to a web service which provides accounting functions for the Ticket System. By providing a suitable deployment composite, the TicketSystem can be wired to the accounting application. In the example below assume the following SCA contribution metadata document, META-INF/sca-contribution.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
<contribution xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   xmlns:here="http://www.acme.com">

   <deployable composite="here:LinkToAccounting"/>
</contribution>
```

Where the LinkToAccounting composite, contained in the file LinkToAccounting.composite (which may be in the root of the Java EE application, or in a nested subdirectory) is defined as:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="LinkToAccounting"
            targetNamespace="http://www.acme.com"
            xmlns:here="http://www.acme.com"
            xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <component name="com.acme.TicketSystem">
         <implementation.composite name="here:ticketing_application"/>
         <reference name="AccountReporting"
                     target="org.sample.Accounting/AccountReporting"/>
   </component>
</composite>
```

And the META-INF/application.composite is defined as:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="ticketing_application"
   targetNamespace="http://www.acme.com"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">


   <component name="web">
         <implementation.web web-uri="web.war"/>
   </component>

   <reference name="AccountReporting" promote="web/AccountReporting"/>

</composite>
```

Note that in this example, the application composite is explicitly used as a component implementation of a composite that is included into the domain. The example above results in the wiring of a reference called ***AccountReporting*** in the ***web.war*** Web Module to the AccountReporting service offered by beancomponent. Following deployment of this example, the domainhas the following graphical representation:

Formatted: French France

Deleted: 712

Deleted: 712

Deleted: 712

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

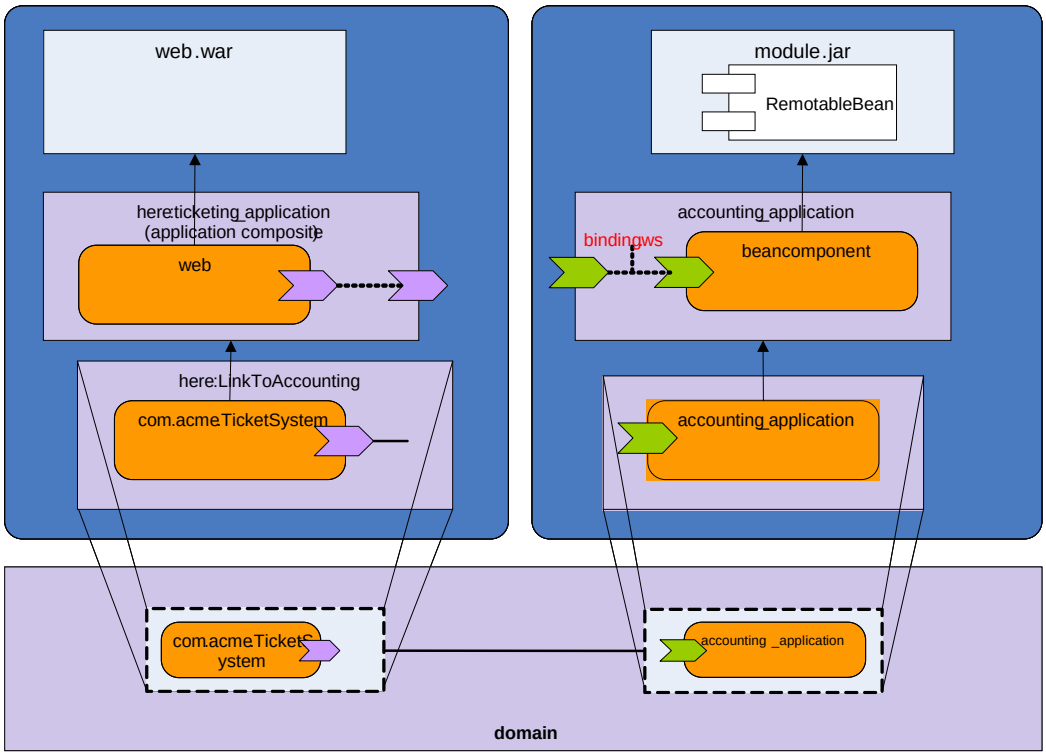


Figure 6: Two Java EE modules wired via SCA

5.1.5 Import and Export of SCA Artifacts

The import and export of SCA artifacts across contributions for example to be used as composite definitions is described in the assembly specification.

For the specific case of the location attribute of the import element of the ***META-INF/sca-contribution.xml*** document a vendor specific resolution mechanism should be provided.

5.1.6 Resolution of WSDL and XSD artifacts

Comment: Need to re-visit this in light of Issue 8 in Assembly TC.

Composite files and other SCA artifacts may reference, directly or indirectly WSDL and XML Schema documents that are not hosted locally, or which cannot be modified to suit the local environment. The OASIS XML Catalogs 1.1 specification [11] defines an entity catalog that can be used to avoid costly remote calls, or to provide a mechanism through which customized versions of documents can be provided without changing application code. Specifically, the XML Catalogs specification provides a mechanism through which

- an external entity's public identifier and/or system identifier can be mapped to a URI reference.
- the URI reference of a resource can be mapped to another URI reference.

Support for the OASIS XML Catalogs 1.1 specification is mandated by JAX-WS 2.0 [13], and an SCA-enabled Java EE runtime MUST resolve WSDL and XML Schema artifacts in a manner consistent with JAX-WS.

Specifically, when an SCA-enabled Java EE archive is deployed, the process of resolving any URIs that point to WSDL or XML schema documents MUST take into account the catalog that is constructed from

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

all META-INF/jax-ws-catalog.xml found in the archive, and resolve the reference as prescribed in the XML Catalogs 1.1 specification.

Comment: Spec needs to say what happens if this catalog is not present.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

6 Java EE Component Based Implementation Types

The basic building block of SCA assembly is the Service Component. In order to provide first class capabilities for exposure of services or consumption of services, the Java EE specification defines implementation types that represent the most prominent application components in Java EE applications: Enterprise JavaBeans (EJB) and Web application components.

The intention is to define a convenient implementation model for Java EE developers to integrate their components with SCA components. For example, a web component developer can use SCA annotations such as ***@Reference*** to declare service component references in a web component implementation.

6.1 Using Session Beans as Implementation Types

Session beans are the Java EE means to encapsulate business logic in an environment that manages remoting, security, and transaction boundaries. Service components play a similar role in SCA and so session beans are candidates for service component implementations in a combined Java EE/SCA environment.

The SCA service programming model described in the SCA Java Component Implementation specification [5] resembles the EJB 3.0 programming model, for instance in its use of dependency injection. As in EJB 3.0, and unlike EJB 2.x, service interfaces do not need to extend any framework defined interfaces. An SCA-enabled Java EE runtime MUST support EJB 3.0 session beans as SCA implementation types. An SCA-enabled Java EE runtime is not required to support EJB 2.1 session beans as SCA implementation types. Handling of other Java EE components, such as Message Driven Beans, is discussed in later sections.

Services and references of service components are associated with interfaces that define the set of operations offered by a service or required by a reference when wiring them. Interface definitions are an important part of the assembly metadata and the interfaces derived from Java EE components are described in the following sections.

6.1.1 Mapping EJB business Interfaces to SCA Service Interfaces

The service interface derived from the business interface of an EJB 3 session bean is comprised of all the methods of the EJB business interface. Furthermore:

- The service interface is remotable if and only if it is derived from a remote business interface. Otherwise it is a local interface. The EJB semantics for remote and local invocations (and thus the by-reference and by-value calls) as defined in the EJB 3.0 specification [8] MUST be honored .

In the case of a business interface of a stateful session bean:

- The service interface is treated as conversational - the interface is treated as if @required=conversational is applied.
- Methods of the interface that are implemented by @Remove methods are treated as @EndsConversation methods of the interface.

6.1.2 The Introspected Component Type of a Session Bean

The component type of a session bean that does not use any SCA annotation and is not accompanied by a component type side file is defined as follows:

1. One <service/> element is present for each EJB 3 business interface of the session bean. The name of the service is the unqualified name of the interface as specified in section 6.1.1. The services have the "ejb" intent applied (i.e. they are treated as if there is @requires="ejb" declared on the service).
   EJB 2.x component interfaces are ignored.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

2. Remote EJB 3 references MAY translate into an SCA references according to section 6.6.
3. Each Simple-Typed Environment Entry of the session bean MAY translate into an SCA property according to section 6.6.

For example, with a business interface as follows:

```
package services.accountdata;

import javax.ejb.Local;

@Remote
public interface AccountService {
   AccountReport getAccountReport(String customerId);
}
```

and with a session bean implementation as follows:

```
package services.accountdata;

import javax.ejb.Stateless;

@Stateless
public class AccountServiceImpl implements AccountService {

   public AccountReport getAccountReport(String customerId) {
          // ...
          return null;
   }
}
```

the following component type of the session bean is:

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
  <service name="AccountService" requires="ejb">
    <interface.java interface="services.accountdata.AccountService"/>
  </service>
</componentType>
```

Deleted: 712

Comment: I've added this as it's clearly required by the defintion above - I am treating this as an editorial slip, but I will raise an issue if required.

6.1.3 Dependency Injection

Any session bean (or other Java EE construct) that is serving as the implementation type of an SCA service component may use dependency injection to acquire proxies for the service references wired to the component by the SCA assembly. Dependency injection may also be used to obtain:

- the value of a property
- a handle to the SCA ComponentContext
- a reference to a callback service reference
- attributes of the current conversation.

Comment: TODO: This may go away - need to revisit

The following table shows the annotations that may be used to indicate the entities to be injected.

Annotation	Purpose
@Callback	Injection of a callback reference (Session beans only)
@ComponentName	Injection of the component name

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

@Context	Injection of SCA ComponentContext
@Property	Injection of a property value
@Reference	Injection of a service reference.
@ConversationID	Injection of a conversation id (Stateful Session beans only)

A complete description of these annotations, and the values associated with them, is given in the Java Common Annotations and APIs specification [4].

When a session bean uses dependency injection, the container MUST inject these dependencies after the bean instance is created, and before any business methods are invoked on the bean instance. If the bean has a PostConstruct interceptor registered, dependency injection MUST occur before the interceptor is called.

EJB's dependency injection occurs as part of construction, before the instance processes the first service request. For consistency, SCA's dependency injection also occurs during this phase.

Instances of stateless session beans are typically pooled by the container. This has some consequences for the programming model for SCA. In general, the values returned from the injected ComponentContext must reflect the current state in which the SCA component is being called. In particular, the value of getRequestContext() MUST return the request context of the current service call, not the request context for which the bean was initially created.

Comment: I think this material relating to pooling of instances is actually NOT part of the model. It should be removed. What it says is true, but it is a note to implementers of runtimes that does not form part of the specification.

"Stateless" COULD be implemented by newing instances each time - and indeed stateless has to work AS IF this is occurring. Pooliing can only be used if this can be guaranteed.

See also section 6.5 for an overview over the life cycle handling of SCA-enhanced Java EE components.

6.1.4 Providing additional Component Type data for a Session Bean

Several of the annotations described in the SCA Java Annotations & APIs specification [4] influence the component type of a session bean (or other Java EE construct). The following table shows the annotations that are relevant in a SCA-enabled Java EE runtime.

Annotation	Purpose
@Property	Declares a property in the component type. The type of the property is obtained through introspection.
@Reference	Declares a reference in the component type. The interface associated with this wire source is obtained through introspection. In the case a field is annotated with both @EJB and @Reference, SCA wiring overrides the EJB target identified by the configuration metadata within the Java EE application by a new target according to SCA wiring rules. If the SCA reference is not wired, the value of the field is the target EJB as determined by Java EE semantics.
@Service	Session beans only: Allows the specification of which of the bean's EJB business interfaces should be exposed as SCA services. The business interface indicated in this annotation MUST be an EJB 3 compliant business interface. The service name is the unqualified name of the interface. A Java EE remote interface is considered a remotable SCA interface. If the @Service annotation is not used, SCA services exist for each business interface exposed by the bean, as described in the section on the component type of unannotated Session Beans.

An SCA-enabled Java EE runtime MUST observe the specified annotations and use them when generating an effective component type.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

6.1.4.1 Example of the use of annotations:

This example shows the use of annotations within an EJB. Continuing the example from section 6.1.2, properties and references are added to the EJB, that are then injected based on SCA assembly metadata:

```
package services.accountdata;

import javax.ejb.Stateless;
import org.osoa.sca.annotations.*;

import services.backend.BackendService;

@Stateless
public class AccountServiceImpl implements AccountService {
   @Reference protected BackendService backend;
   @Property protected String currency;

   public AccountReport getAccountReport(String customerId) {
         // ...
         return backend.getCustomerReportforCurrency(customerId, currency);
   }
}
```

Comment: Java example was simply wrong - it would never compile

has the following component type:

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
  <service name="AccountService">
    <interface.java interface="services.accountdata.AccountService"/>
  </service>
  <property name="currency"/>
  <reference name=”backend”>
    <interface.java interface="services.backend.BackendService"/>
  </reference>
</componentType>
```

Deleted: 712

6.1.5 Using a ComponentType Side-File

Using SCA annotations, a service implementation developer can create session beans that imply an extended component type. If further tuning of the component type is necessary than can be achieved with annotations, a component type side file may be included in the contribution. The component type side file follows the naming pattern

META-INF/<bean name>.componentType

and is located in the EJB module containing the bean. The rules on how a component type side file adds to the component type information reflected from the component implementation are described as part of the SCA assembly model specification [3]. If the component type information is in conflict with the implementation, it is an error as defined in [3].

If the component type side file specifies a service interface using a WSDL interface, then the bean interface MUST be compliant with the specified WSDL, according to the rules given in section 'WSDL 2 Java and Java 2 WSDL' in the Java Annotations and APIs Specification [4].

.

Comment: I really feel that this statement is redundant - it is dealt with in the assembly spec.

Comment: Deleted this paragraph as a) it isn't normative and b) its wrong - it misunderstands the CAA spec.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

6.1.6 Creating SCA components that use Session Beans as Implementation Types

In order to declare a service component that is implemented as a session bean, an implementation.ejb declaration is used. It has the following pseudo schema:

```
<implementation.ejb ejb-link="<ejb-link-name>”/>
```

The ejb-link-name attribute uniquely identifies the EJB that serves as the component implementation. The format of the value is identical to the format of the ***ejb-link*** tag in a Java EE deployment descriptor. In the case that the SCA contribution containing the composite file is an application EAR file, it is possible that several session beans have the same name. In that case the value of the ejb-link element must be composed of a path name specifying the ejb-jar containing the referenced enterprise bean with the ejb-name of the referenced enterprise bean appended and separated from the path name with a '#'. The path name is relative to the root of the EAR. In the case that SCA contribution is an EJB module's JAR file, the path name may generally be omitted.

The following example declares a service component named ***beancomponent*** in the composite ***beancomposite*** of the namespace ***http://www.sample.org***. ***Beancomponent*** is implemented by the bean ***SimpleBean*** in the ejb-module ***module.jar***. ***Beancomponent*** exposes a service, named after the bean's business interface name, that is promoted to the composite level:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="beancomposite" targetNamespace="http://www.sample.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <service name="AccountReporting" promote="beancomponent/AccountService"/>

   <component name="beancomponent">
         <implementation.ejb ejb-link="module.jar#SimpleBean"/>
   </component>
</composite>
```

Deleted: 712

6.1.7 Limitations on the use of Session Beans as Component Implementations

Session beans that serve as SCA implementations remain session beans from a Java EE perspective, and may be found and used just like any other session bean, for instance, through dependency injection via an @EJB annotation, or though JNDI lookup.

An enterprise bean accessed through normal Java EE methods can contain SCA annotations such as @Reference or @Property, or may look up its configuration through the SCA API, and therefore, require configuration from the SCA runtime.

Therefore, within the assembly of the contribution package, a session bean MUST NOT be used as service component implementation more than once once. Whether the enterprise bean is accessed through Java EE means, or through an SCA reference, the same service component configuration is used (see also section 6.5).

The EJB Specification defines a container contract that defines what behavior implementations may expect from the container, and what behavior the container can expect from the implementation. For instance, implementations are forbidden from managing class loaders and threads, but on the other hand, implementations need not be programmed for thread safety, since the container guarantees that no bean instance will be accessed concurrently. In an SCA-enabled Java EE runtime, both parties are expected to continue to abide by this contract. That is, a session bean that is serving as an SCA implementation type must continue to be a well-behaving EJB, abstaining from thread and class loader management, and the SCA-enabled Java EE runtime must also continue to behave as in accordance with the EJB container contract.

6.1.8 Use of Implementation Scopes with Session Beans

The lifecycle of a stateless session bean is not impacted by its use in an SCA context. In the terminology provided in the Java Common Annotation and APIs specification [4], a stateless session bean always has a STATELESS implementation scope. An SCA-enabled Java EE runtime is not required to provide means for tuning or customizing this behavior.

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

Similarly, the lifecycle of a stateful bean is, by default, not impacted by its use in an SCA context. The bean instance remains (modulo passivation/activation cycles) until it times out or one of its @Remove methods are called. In SCA Java terms, a stateful session bean has CONVERSATIONAL implementation scope.

6.1.9 SCA Conversational Behavior with Session Beans

The SCA Assembly Specification [3] introduces the concept of conversational interfaces for describing service contracts in which the client can rely on conversational state being maintained between calls, and where the conversational identifier is communicated separately from application data (possibly in headers). Note that a conversational contract assumes association with a conversationally scoped implementation instance such as stateful bean. Section 6.1.1 defines how business interfaces are mapped to SCA service. An SCA conversational interface MUST NOT be used with a stateless bean.

6.1.10 Non-Blocking Service Operations

Service operations defined by a Session Bean's business interface may use the @OneWay annotation to declare that when a client invokes the service operation, the SCA runtime must honor non-blocking semantics as defined by the SCA assembly Specification [3].

6.1.11 Accessing a Callback Service

Session Beans that provide the implementation of SCA components which implement bidirectional services and require a callback reference may use the @Callback annotation to have a reference to the callback service associated with the current invocation injected on a field or setter method.

6.2 Using Message Driven Beans as Implementation Types

Message Driven Beans are the Java EE construct for consuming asynchronous messages. Message Driven beans may participate in SCA assembly as the implementation type of a component that does not offer any services, but may be configured or wired from. Message-driven beans cannot be instantiated arbitrarily often due to their association with non SCA-controlled endpoints (typically JMS). Therefore, within the assembly of the application package, a message-driven bean may be used as service component implementation at most once (see also section 6.5).

6.2.1 Dependency Injection

A message driven bean that is the implementation type of an SCA component may use dependency injection to acquire references to the services wired to the component by the SCA assembly. Dependency injection may also be used to obtain the value of properties or a handle to the component's component context. The following table shows the annotations that may be used to indicate the fields or properties to be injected.

Annotation	Purpose
@ComponentName	Injection of component name
@Context	Injection of SCA context into member variable of service component instance
@Property	Injection of configuration properties from SCA configuration
@Reference	Injection of Service references

A complete description of these annotations, and the values associated with them, is given in the Java Common Annotations and APIs specification [4].

When a message driven bean uses dependency injection, the container MUST inject these references after the bean instance is created, and before any business methods are invoked on the bean instance. If

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

650 the bean has a PostConstruct interceptor registered, dependency injection MUST occur before the
651 interceptor is called.

652 See also section 6.5 for an overview over the life cycle handling of SCA-enhanced Java EE components.

653 6.2.2 The Component Type of an Unaltered Message Driven Bean

654 Unlike Session Beans, Message Driven Beans do not have business interfaces. Therefore, the
655 component type implied from a message driven bean does not offer any SCA services. The bean may, of
656 course, be accessed indirectly over a binding.jms call to its associated queue, but this is not transparent
657 to the SCA assembly.

658 The component type of a message driven bean that does not use any SCA annotation and is not
659 accompanied by a component type side file is constructed according to the following algorithm:

660 1. Remote EJB 3 references MAY translate into an SCA references according to section 6.6.
661 2. Each Simple-Typed Environment Entry of the message driven bean MAY translate into an SCA
662 property according to section 6.6.

663 6.2.3 Providing additional Component Type data for a Message Driven Bean

664 Several of the annotations described in the Java Common Annotations and APIs specification [4] can be
665 used to extend Message Driven Beans and influence the component type of the Message Driven Bean.
666 The following table shows the annotations that are relevant:.

Annotation	Purpose
@Property	Declares a property in the component type
@Reference	Declares a reference in the component type.

667

668 An SCA-enable Java EE runtime MUST observe the specified annotations and use them when
669 generating an implied component type.

670 6.2.4 Creating SCA Components that use Message Driven Beans as
671 Implementation Types

672 Since both Message Driven Beans and Session Beans are Enterprise Java Beans, both can be uniquely
673 referenced in an ejb-link. Therefore an ***implementation.ejb*** (described in section 6.1.6 above) is used for
674 a message driven bean.

675 6.2.5 Limitations on the Use of Message Driven Beans as Component
676 Implementation

677 A few limitations with respect to use as service component implementation apply to Message Driven
678 Beans:

679 - A Message-Driven Bean MAY NOT be given an implementation scope.

680 6.3 Mapping of EJB Transaction Demarcation to SCA Transaction
681 Policies

682 The EJB programming model supports a concept of container managed transaction handling in which the
683 bean provides class-level or method-level information on transaction demarcation that is observed by the
684 EJB runtime implementation. SCA's policy framework [6] defines an extended transaction demarcation
685 model using SCA policy intents.

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

However, since EJB transaction attributes can be defined on the class as well as on the method-level, the EJB model more fine-granular than SCA's transaction model and a simple mapping to SCA policies is not possible.

Comment: We need to revisit this - it may not be correct now

For class-level transaction demarcation, the following table illustrates the mapping of EJB transaction attributes to SCA transaction implementation policies:

EJB Transaction Attribute	SCA Transaction Policy, required intents on services	SCA Transaction Policy, required intents on implementations
NOT_SUPPORTED	suspendsTransaction	
REQUIRED	propagatesTransaction	managedTransaction.global
SUPPORTS	propagatesTransaction	managedTransaction.global
REQUIRES_NEW	suspendsTransaction	managedTransaction.global
MANDATORY	propagatesTransaction	managedTransaction.global
NEVER	suspendsTransaction	

Note: in the case of MANDATORY and NEVER demarcations, policy mapping is not completely accurate as these attributes express responsibilities of the EJB container as well as the EJB implementer rather then expressing a requirement on the service consumer (see [8]).

It is required that EJB's transaction model stays unchanged by SCA, and an SCA-enabled Java EE runtime MUST adhere to the rules laid out in the EJB 3.0 specification [8].

Comment: Something should be said here about the SCA Policy annotations - are they legal?

6.4 Using Web Modules as Implementation Types

Web modules can participate in SCA assembly as the implementation type of a component that does not offer services, but may be configured or wired from.

6.4.1 Dependency Injection

A Web Module can use dependency injection to acquire references to the services wired to the component by the SCA assembly. Dependency injection can also be used to obtain the value of properties or to obtain the component context. The following table shows the annotations that can be used to indicate the entities to be injected.

Annotation	Purpose
@ComponentName	Injection of component name
@Context	Injection of SCA ComponentContext
@Property	Injection of a property value
@Reference	Injection of a Service reference

A complete description of these annotations, and the values associated with them, is given in the Java Common Annotations and APIs specification [4].

Due to the multi-threaded nature of web artifacts, in the case where a Reference Proxy targeted to a conversational interface (such as stateful session beans) might not behave as expected. SCA-Java EE Runtimes MAY treat this case as an error. The recommended approach to obtain such reference proxy is via usage of ComponentContext.

Comment: FIXME - requires a clearer explanation

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Dependency injection of values from SCA configuration occurs in exactly those artifacts that the web container can inject values based on the Java EE configuration. An SCA-enabled Java EE server MUST be able to perform dependency injection on the following artifacts.

Name	Interface or Class
Servlets	javax.servlet.Servlet
Servlet filters	javax.servlet.ServletFilter
Event listeners	javax.servlet.ServletContextListener javax.servlet.ServletContextAttributeListener javax.servlet.ServletRequestListener javax.servlet.ServletRequestAttributeListener javax.servlet.http.HttpSessionListener javax.servlet.http.HttpSessionAttributeListener
Taglib tag handlers	javax.servlet.jsp.tagext.JspTag
JavaServer Faces technology-managed beans	Plain Old Java Objects (POJOs)

Deleted: javax.servlet.http.HttpSessionBindingListener

See also section 6.5 for an overview over the life cycle handling of SCA-enhanced Java EE components.

6.4.2 The Component Type of an Unaltered Web Module

Since it does not offer SCA services the component type of a Web Module does not contain any SCA services. However, it can contain references and properties.

The component type of a web application that does not use any SCA annotation and is not accompanied by a component type side file is constructed according to the following algorithm:

1. Remote EJB 3 references MAY translate into an SCA references according to section 6.6.
2. Each Simple-Typed Environment Entry of the Web Module MAY translate into an SCA property according to section 6.6.

6.4.3 Providing additional Component Type Data for a Web Application

Several of the annotations described in the Java Common Annotations and APIs apecification [4] can be used in a Web application and affect its component type. The following table shows the annotations that are relevant in a SCA-enabled Java EE runtime.

Annotation	Purpose
@Property	Declares a property
@Reference	Declares a service reference

An SCA-enable Java EE runtime MUST observe the specified annotations their effect on the component type. All artifacts where dependency injection may occur (see the table in section 6.4.1) MUST be inspected when determining the component type.

A web component can provide additional component type data in a side file

WEB-INF/web.componentType

in the Web Module archive.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

6.4.4 Using SCA References from JSPs

JavaServer Pages (JSP) tag libraries define declarative, modular functionality that can be reused by any JSP page. Tag libraries reduce the necessity to embed large amounts of Java code in JSP pages by moving the functionality of the tags into tag implementation classes ([7]).

Following this philosophy, a JSP tag library is defined to expose SCA capabilities in JSP pages. The following snippet illustrates the use of an SCA reference using the tag library:

```
<%@ taglib uri="http://docs.oasis-open.org/ns/opencsa/sca/200903/sca.tld"
prefix="sca" %>

......

<sca:reference name="service" type="test.MyService"  />

<% service.sayHello(); %>
```

Deleted: http://www.osog.org/sca

An SCA-enabled Java EE runtime MUST support the SCA JSP tag library by providing implementations of the tag-class and tei-class. The servlet container hosting the webapp instantiates new instances of the tag-class whenever it comes across the SCA specific tag in a JSP page. The tag-class is responsible for doing dependency injection into the JSP page based on the properties provided to the JSP page. The scope of the object injected is PageContext. APPLICATION_SCOPE in case the the interface is not conversational and PageContext. SESSION_SCOPE in case the interface is stateful. The SCA JSP tag also makes available the given reference with a newly declared scripting variable of the same id.

In order to access SCA configuration from JSP pages, JSP page authors MUST import the SCA tag library provided by the SCA runtime and provide all the properties necessary for dependency injection. The required properties are the name of the reference to be injected, and the type of the field (Service interface class name).

All tag libraries are required to provide a TagLibrary Descriptor (TLD). The information provided by via the tag library descriptors is used by the web application container to handle processing of tags in the JSP page. The TLD of the SCA tag library is shown here:

```
<?xml version = '1.0' encoding = 'ISO-8859-1'?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN" "http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd">
<taglib version="2.1">

  <tlib-version>1.0</tlib-version>
  <short-name>SCA-JSP</short-name>
  <uri>http://docs.oasis-open.org/ns/opencsa/sca/200903/sca_jsp.tld</uri>
  <description>A tag library for integrating sca components with jsp
  </description>

  <tag>
    <name>reference</name>
    <tag-class><!—To be provided by the SCA runtime implementation --></tag-
class>
    <tei-class><!—To be provided by the SCA runtime implementation --></tei-
class>

    <attribute>
      <name>name</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
      <type>java.lang.String</type>
    </attribute>

    <attribute>
      <name>type</name>
      <required>true</required>
```

Formatted: English U.S.

Deleted: http://www.osoa.org/sca

Formatted: English U.S.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
      <rtexprvalue>false</rtexprvalue>
      <type>java.lang.String</type>
    </attribute>


    <body-content>empty</body-content>

  </tag>

</taglib>
```

Formatted: Norwegian Bokmål

6.4.5 Creating SCA Components that Use Web Modules as Implementation Types

The ***implementation.web*** element is used to declare a service component that is implemented by the web component. It has the following pseudo-schema.

```
<implementation.web web-uri="<module name>"/>
```

A web component MUST NOT be configured more than once per assembly of the contribution package.

6.4.6 Limitations on the Use of Web Modules as Component Implementations

Because each module is associated with a unique context root, Web Modules MUST NOT be used more than once as a service component implementation (see also section 6.5).

Furthermore, a Web Module MUST NOT be given an implementation scope.

6.5 Life-Cycle Model for Service Components implemented by Java EE Components

The EJB implementation type and the Web implementation type refer to components whose life cycle is not completely controlled by the SCA runtime implementation but rather in a shared responsibility with a Java EE runtime.

This model is motivated by several considerations:

- EJB and Web components MAY be invoked out-of-band from an SCA perspective: for example via a JNDI lookup and invocation in the case of a session bean, by receiving a JMS message in the case of a Message-Driven bean, or by an HTTP request in the case of a web application.
- Prior to invocation of an SCA enhanced component, the runtime MUST provide the Java EE context for the Java EE components as well as the SCA context (e.g. by injecting references)..

This specification defines the following rules that eliminate potential ambiguities:

- A Java EE component MUST NOT be used more than once as implementation of an SCA service component within the assembly of a Java EE application package (an EAR archive, or a standalone web application module, or a standalone EJB module).
- If a Java EE component that has a component type side file and/or is enhanced by SCA annotations is not used as a component implementation by an explicit service component declaration within the assembly of a Java EE application package, then it will not be associated with a component context and any SCA annotation MAY cause an error or may be ignored.

Comment: MUST?

Furthermore the following life cycle handling rules apply:

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

- The component life cycle of an SCA enhanced Java EE component is nested within its Java EE component life cycle. More specifically:

- Java EE initialization of an SCA enhanced Java EE component happens before any SCA component initialization. Both occur before any business method invocation (or HTTP request in the case of a web application).
- If an EJB has a PostConstruct interceptor registered, component initialization happens before the interceptor is called.
- No business method invocation (or HTTP request in the case of a web application) on the service component occurs after scope destruction (i.e. while and after @Destroy life cycle methods are called) and before the component implementation instance is finalized.

- The point in time of deployment of an SCA enhanced Java EE component is exactly the point in time it is deployed as a Java EE component.

6.6 Mapping a Java EE Component's Environment to Component Type Data

In the absence of optional extensions, the component type of a Java EE component (such as a Servlet or Enterprise Bean) does not contain SCA references. However, as an optional extension, an SCA runtime can choose to provide the capability of re-wiring EJB references using SCA. If an SCA runtime provides this optional extension, then the following rule is applied:

Each EJB 3 remote reference of each session bean within a Java EE application is exposed as an SCA reference. Each EJB reference has a target (within the Java EE application) that is the EJB identified by the configuration metadata within the Java EE application - it is this target which may be overridden by a new target identified in the SCA metadata of the component using the Java EE application. The multiplicity of the generated reference is 0..1. The generated reference has the the "ejb" intent applied to it:

```
<intent name="ejb" constrains="sca:binding">
   <description> The EJB intent requires that all of the semantics required by
the Java EE specification for a communication to or from an EJB must be
honored </description>
</intent>
```

Note that SCA names for references are of the XML Schema type NCName, while Java EE names for EJB references are of a type that allows a larger character set than what is supported in NCNames. The following escape algorithm defines how to translate names of EJB references and into names of SCA references:

1. Replace all "/" characters by "_" (underscore) characters
2. All remaining characters that are not supported in NCName are escaped as XML entities or character references.

As an additional vendor extension, each environment entry with a simple type may be translated into an SCA property. The name of the property is derived from the name of the resource, according to the algorithm given below. The XML simple type of the SCA property is derived from the Java type of the environment entry according to the following type mapping:

Environment Entry Type	XSD Type
String	String
Character	String
Byte	Byte
Short	Short
Integer	Int
Long	Long

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Boolean	Boolean
Double	Double
Float	Float

These optional extensions are not required to be provided by any given SCA runtime and that, as a result, it is unadvisable to rely on the capability of rewiring EJB references when porting applications between different runtimes.

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

7 Java EE Archives as Service Component Implementations

The previous sections described how Java EE archives can be represented in SCA where each of the Java EE components in the archive can be mapped to separate SCA components. It is also possible in SCA for an entire archive to be represented as a single coarse-grained component.

The ***JEE implementation type*** supports this usecase. It has the following pseudo schema:

```
<implementation.jee archive="..."?>
   <xs:any/>*
</implementation.jee>
```

The ***archive*** attribute specifies a relative path to the Java EE archive that serves as implementation artifact. The context of that relative path (the value ".") is the location of the artifact that contains the ***implementation.jee*** element. All Java EE components contained in the archive are deployed, regardless of any SCA enhancements present (see also section 6.5).

Every deployed SCA component using the Java EE implementation type represents a deployment of the referenced Java EE archive. Implementers are encouraged to make use of the extensibility of the Java EE implementation type declaration to provide deployment plan meta-data as to support vendor-specific deployment features as well as multiple deployments of one Java EE archive.

The archive that is referred to by <implementation.jee> may be an artifact within a larger contribution (i.e. an EAR inside a larger ZIP file), or the archive may itself be a contribution. In the latter case, the @archive attribute can be left unspecified, and the archive defaults to be the archive of the contribution itself.

Comment: What about referring to the archive as a stand-alone, non-SCA contribution?

The component type derived from a Java EE archive depends on whether it has been enhanced with SCA artifacts and contains an application composite or not – as described in following sections.

7.1 The Component Type of a non-SCA-enhanced Java EE Archive

Java EE modules, in particular EJB modules and Web Modules are frequently designed for re-use in more than one application. In particular EJB session beans provide a means to offer re-usable implementations of business interfaces. In addition Java EE modules can use EJB references as a point of variation to integrate with the assembly of a hosting application. The following sections describe the introspected component type for non-SCA-enhanced Java EE archives.

7.1.1 The Component Type of non-SCA-enhanced EJB Module

The introspected component type of an EJB module is defined by the following algorithm:

1. Each EJB 3 business interface with unqualified name ***intf*** of a session bean ***bean*** translates into a service by the name ***bean_intf***. The interface of the service and the requirement for EJB intent is derived as in sections 6.1.1 and 6.1.2.
2. In the absence of optional extensions, the component type of a non-SCA-enhanced EJB module does not contain SCA references. However, as an optional extension of the way in which SCA support is provided for EJB modules, an SCA runtime can choose to provide the capability of re-wiring EJB references using SCA. If an SCA runtime provides this optional extension, then the following rule is applied. Each EJB 3 reference with name ***ref*** of a session bean ***bean*** translates into an SCA reference of name ***bean_ref***. The interface of the reference is derived according to section 6.1.1. The reference's name may require escaping as defined in section 6.6.

For example, an EJB 3 module ***reusemodule.jar*** may contain a session bean definition ***UsesOthersBean***

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
package com.sample;

import javax.ejb.EJB;
import javax.ejb.Stateless;

@Stateless(name="UsesOthersBean")
public class UsesOthersBean implements UsesOthersLocal {

   @EJB
   private IUOBRefService ref;

   // ...

}
```

that, by use of annotations in this case, has an EJB reference by name ***com.sample.UsesOthersBean/ref*** and the business interface ***IUOBRefService*** (note that alternatively the EJB reference could have been declared in the module's deployment descriptor ***META-INF/ejb-jar.xml***).

When using this within implementation.jee this gives a component type of the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
    <service name="UsesOthersBean_UsesOthersLocal" requires="ejb">
        <interface.java interface="com.sample.UsesOthersLocal" />
    </service>

    <reference name="UsesOthersBean_com.sample.UsesOthersBean_ref"
               requires="ejb">
        <interface.java interface="com.sample.IUOBRefService" />
    </reference>
</componentType>
```

7.1.2 The Component Type of a non-SCA-enhanced Web Module

As for EJB modules, Web Modules may be re-usable. The introspected component type of a Web Module conforming to the Java Servlet Specification Version 2.5 ([6]) is defined as follows:

1. In the absence of optional extensions, the component type of a non-SCA-enhanced Web module does not contain SCA references. However, as an optional extension of the way in which SCA support is provided for Web modules, an SCA runtime can choose to provide the capability of re-wiring EJB references using SCA. If an SCA runtime provides this optional extension, then the following rule is applied. Each EJB 3 reference with name ***ref*** of translates into an SCA reference of name ***ref***. The interface of the reference is derived according to section 6.6. The reference's name may require escaping as defined in section 6.6.

For example, a Web application with the following Servlet

```
package com.sample;

import java.io.IOException;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class ReusableServlet extends javax.servlet.http.HttpServlet implements
javax.servlet.Servlet {

   @EJB
   private UsesOthersLocal uobean;
```

Formatted: English U.S.

Deleted: 712

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
   public void service(ServletRequest req, ServletResponse resp)
            throws ServletException, IOException {
         // ...
   }
}
```

implies the following component type

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
    <reference name="com.sample.ReusableServlet_uobean" requires="ejb">
        <interface.java interface="com.sample.UsesOthersLocal" />
    </reference>
</componentType>
```

7.1.3 The Component Type of a non-SCA-enhanced Java EE Application

The introspected component type of a non-SCA-enhanced Java EE application is defined as follows:

Each EJB 3 session bean business interface with unqualified name ***intf*** of a session bean with mapped name ***mname*** translates into a service by the name ***mname_intf***. The interface of the service is derived as in section 6.1.1. The service name is subject to escaping rules as described in section 6.6.

In the absence of optional extensions, the component type of a non-SCA-enhanced Java EE application does not contain SCA references. However, as an optional extension of the way in which SCA support is provided for Java EE applications, an SCA runtime can choose to provide the capability of re-wiring EJB references using SCA. If an SCA runtime provides this optional extension, then the following rule is applied:

Each EJB 3 remote reference of each session bean within the Java EE application is exposed as an SCA reference. If the remote reference has the name ***ref*** and the name of the session bean is ***beanname***, the SCA reference name is ***beanname_ref***. The reference has an interface derived according to section 6.6. The reference name is subject to the escaping rules as described in section 6.6. Each EJB reference has a target (within the Java EE application) that is the EJB identified by the configuration metadata within the Java EE application - it is this target which may be overridden by a new target identified in the SCA metadata of the component using the Java EE application. The multiplicity of the generated reference is 0..1. The generated reference has the “ejb” intent applied :

```
<intent name="ejb" constrains="sca:binding">
  <description> The EJB intent requires that all of the semantics required by
the Java EE specification for a communication to or from an EJB must be
honored </description>
</intent>
```

This optional extension is not required to be provided by any given SCA runtime and that, as a result, it is unadvisable to rely on the capability of rewiring EJB references when porting applications between different runtimes.

7.2 The Component Type of an SCA-enhanced Java EE Archive

A Java EE archive that contains an application composite (see the section 5.1.3) has the component type of the application composite as its introspected component type when used with the Java EE implementation type.

Example: Assume the right hand side application from the example in section Domain Level Assembly of SCA-enhanced Java EE Applications is packaged in an archive ***application.ear*** and is used as part of a

Formatted: French France

Deleted: 712

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

larger non-Java EE contribution that declares a service component in some other composite that uses the archive ***application.ear*** as implementation artifact.

In that case the component type of the EAR archive would expose one service, the ***AccountReporting*** service:

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
   <service name="AccountReporting">
         <binding.ws/>
         <interface.java interface="services.accounting.AccountReporting"/>
   </service>
</componentType>
```

Or, graphically:

Deleted: 712

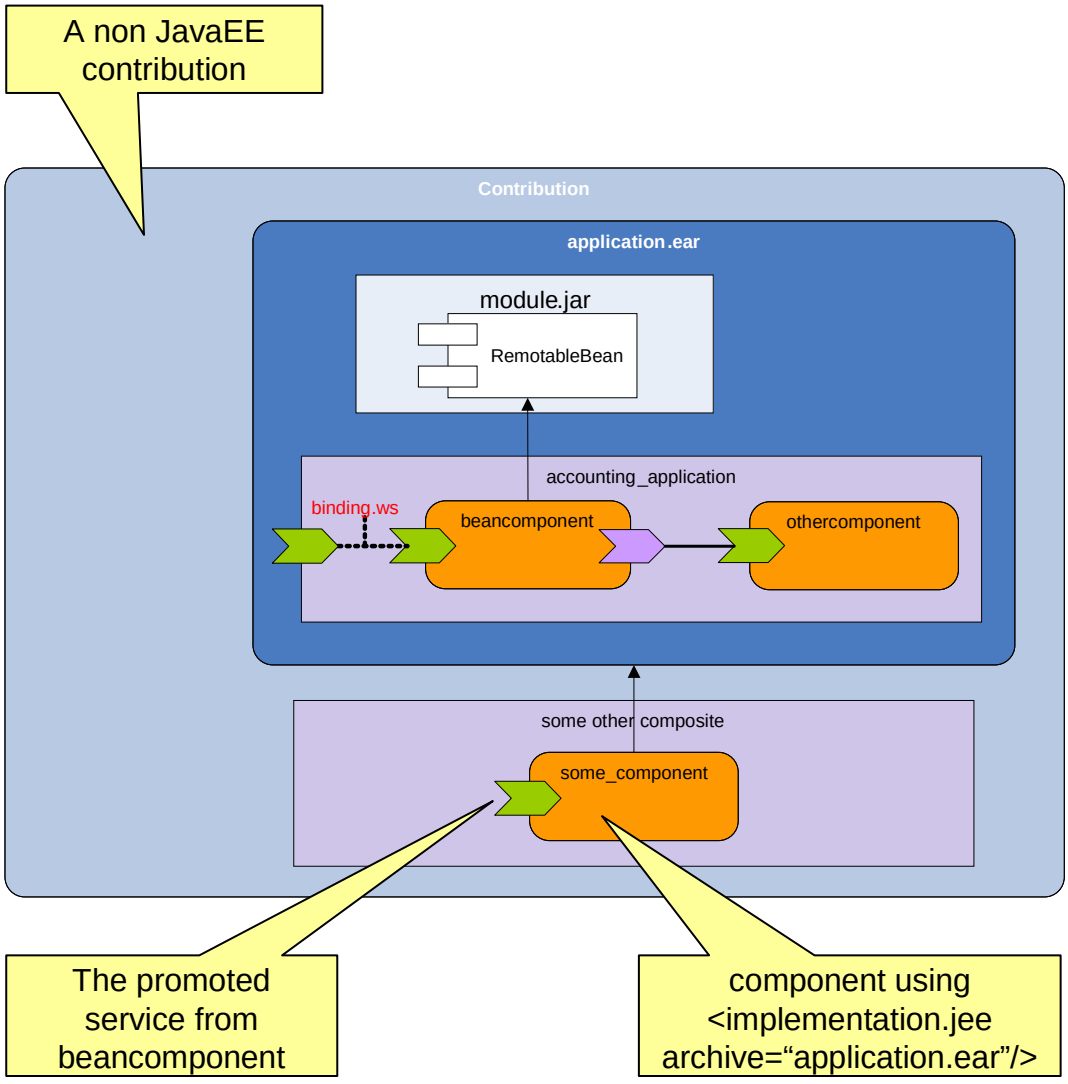
Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8



1044
1045 ***Figure 7: Java EE EAR used as a component implementation***

1046

1047 This way, the application composite provides fine-grained control over what services, references, and
1048 properties are exposed from a Java EE archive.

1049 In cases where a given non-enhanced Java EE archive is already in use as a service component
1050 implementation and the need arises to extend it by SCA assembly metadata, it is desirable to have a
1051 smooth and controlled transition from the exposure defined for non-enhanced archives.

1052 That can be achieved using the **includeDefaults** attribute that can be specified on composite and
1053 component elements. It has the default value "***false***" and is defined in the name space ***http://docs.oasis-***
1054 ***open.org/ns/opencsa/sca-j/200903***.

1055 Using this attribute on the application composite's composite declaration with a value "***true***" leads to a
1056 (logical) inclusion of SCDL definitions into the application composite that reproduce the component type

Deleted: ***808***
Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

of the Java EE archive as if it was not SCA-enhanced, alongside any elements in the component type that result from the SCA enhancements of the Java EE archive.

For a Java EE application archive, the included SCDL is constructed by the following algorithm:

1. For every EJB or Web Module that has services or references exposed according to section 6, a corresponding implementation.ejb or implementation.web component is included, if that EJB or Web Module is not used as a component implementation elsewhere already.
2. For every service or reference that is derived according to section 6, a composite level service or reference declaration is included, by the same name, promoting the corresponding EJB service or reference.

Corresponding algorithms apply for the case of a standalone Web Module (section 7.1.2) and a standalone EJB module (section 7.1.1).

Example (continued): Assume furthermore that the EJB module ***module.jar*** additionally contains the ***AccountServiceImpl*** session bean of section 6.1.2 and the application composite is modified as shown below (note the use of ***includeDefaults***).

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="accounting_application"
   targetNamespace="http://www.sample.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   xmlns:scajee="http://docs.oasis-open.org/ns/opencsa/sca-j/200903"
   scajee:includeDefaults=”true”
>

   <service name="AccountReporting"
          promote="beancomponent/AccountServiceRemote">
          <binding.ws/>
   </service>

   <component name="beancomponent">
          <implementation.ejb ejb-link="module.jar#RemotableBean"/>
          <property name="currency">EUR</property>
   </component>
</composite>
```

Deleted: 712

Deleted: 808

That alone would not change the component type of the archive. However, if we additionally assume the session bean ***AccountServiceImpl*** is given a mapped name ***services/accounting/AccountService***, the component type of the EAR archive would expose two services, ***AccountReporting***, ***services_accounting_AccountService_AccountServic***e.

The logical include to the application composite constructed following the algorithm above is this:

```
<service name="services_accounting_AccountService_AccountService"
   promotes="AccountServiceImpl/AccountService" />

<component name="AccountServiceImpl">
   <implementation.ejb ejb-link="module.jar#AccountServiceImpl" />
</component>
```

Comment: FIXME

As a result, we would get the following component type:

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
   <service name="AccountReporting">
          <binding.ws/>
   </service>

   <service name="services_accounting_AccountService_AccountService"/>
</componentType>
```

Deleted: 712

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

Or, graphically:

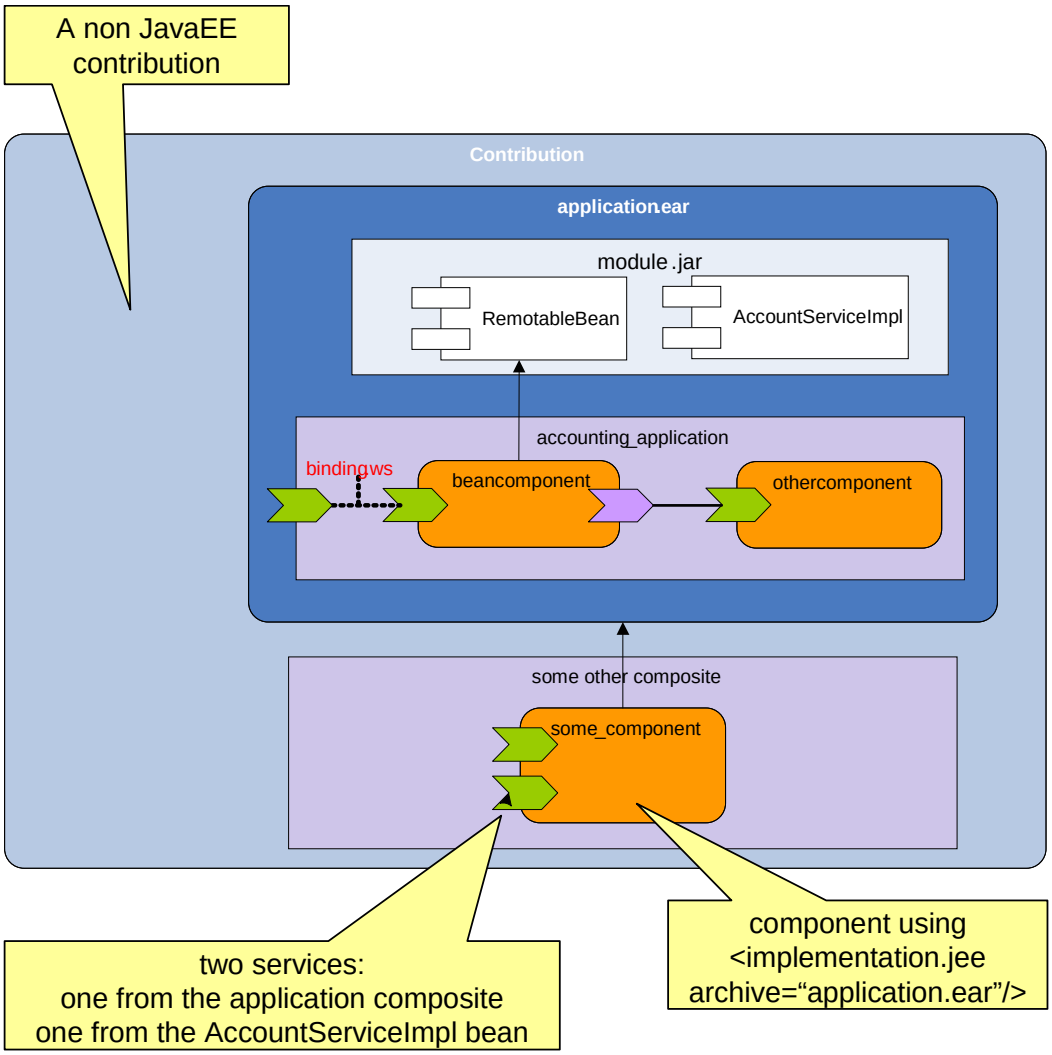


Figure 8: Java EE extended archive used as a Component

The same result can be achieved by declaring the ***includeDefaults*** attribute on a component declaration that uses the ***AccountServiceImpl*** session bean as implementation:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="accounting_application"
   targetNamespace="http://www.sample.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   xmlns:scajee="http://docs.oasis-open.org/ns/opencsa/sca-j/200903"
>

   <service name="AccountReporting"
         promote="beancomponent/AccountServiceRemote">
```

Deleted: 712

Deleted: 808

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

```
        <binding.ws/>
    </service>

    <component name="beancomponent">
        <implementation.ejb ejb-link="module.jar#RemotableBean" />
        <property name="currency">EUR</property>
    </component>

    <component name="accounting" scajee:includeDefaults=”true”>
        <implementation.ejb ejb-link="module.jar#AccountServiceImpl"/>
    </component>
</composite>
```

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

A. Use Cases

The following sections describe use cases which are intended to be illustrative of the concepts described in this specification and are therefore non-normative.

A.1 Technology Integration

SCA can be used as the scale-out model for Java EE applications, allowing Java EE components to use, be used by, and share a common deployment lifecycle with components implemented in other technologies, for instance, BPEL.

As an example, imagine a sample shop in which the graphic user interface is implemented as a browser based application using a servlet or a JSF, the persistence logic is implemented in JPA and exposed using session beans, but the order process is implemented in BPEL. Using standard technologies, the Java EE components would have to access the BPEL process over its exposed web services. Conversely, in order for the implemented persistence logic to be used from the BPEL process, the session beans would have to be exposed as web services, typically using JAX-WS.

There are several drawbacks to this approach. Conceptually, the BPEL process is part of the application, however, in the standard deployment described above, the BPEL process is deployed separately from the Java EE application; they do not share life cycle or infrastructure. The use of WebServices as wire protocol implies some drawbacks. Transaction management and the enforcement of security policies become much more difficult, and the overhead associated with service invocations increases.

To make the example a bit more concrete, let us envisage that the application's web front-end, implemented as a servlet, invokes the BPEL process. The BPEL process in turn invokes a session bean called "OrderService", which uses JPA technology to persist the order information.

The first step is to prepare the servlet to make the cross technology call. This is done by adding a field with the appropriate business interface, and annotating it with an @Reference tag.

```
public class ControllerServlet extends HttpServlet implements Servlet {
   @Reference protected IOrderProcess orderProcess;
            …
   protected void service(HttpServletRequest request,
                  HttpServletResponse response) throws Exception {
…
   orderProcess.placeOrder(orderData);
…
}
```

Such a snippet should be familiar to anyone who has used the EJB client model. The main difference between the @EJB and the @Reference annotation is that @EJB tells the user which technology is being used to implement the service, whereas @Reference leaves this open. It is fixed at a later stage by the metadata in the SCA assembly.

The next step in creating a cross technology application in SCA is to create the assembly file that hooks together the components of the application, and links each to an implementation. In this case, there are three SCA components: the web front-end, the BPEL component, and the EJB that offers the persistence service. Note that there may be many more EJBs and web components in our Java EE application, we do not need to represent them all as SCA components. Only those Java EE components that will be wired to or from, or otherwise configured from SCA, need to be represented in the SCA assembly.

The following figure shows how the components are hooked together.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

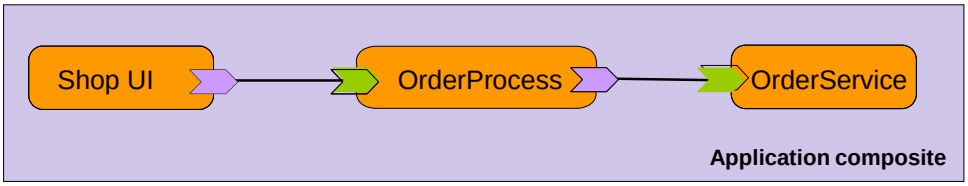


Figure 9: Example Shop application - SCA composite diagram

The significant part of the composite file looks like this:

```
<sca:component name="OrderService">
    <sca:implementation.ejb ejb-link="shop.ejb.jar#OrderService"/>
    <sca:service name="IOrderService">
      <sca:interface.java
           interface="sample.shop.services.IOrderService"/>
    </sca:service>
</sca:component>
<sca:component name="shop.ui>
    <sca:implementation.web web-uri="shop.web.war"/>
    <sca:reference name="orderProcess" target="OrderProcess"/>
</sca:component>
<sca:component name="OrderProcess">
    <sca:implementation.bpel process="shop.bpel"/>
    <sca:reference name="orderServicePL" target="OrderService">
    <sca:service name="OrderProcessRole"/>
</sca:component>
```

Formatted: English U.S.

There are several ways in which such a cross-technology application could be deployed. If we consider the BPEL process to be part of the application, conceptually on the same level as the application web or EJB components, then it makes sense to deploy the cross technology application as an ***SCA-enhanced Java EE archive***, that is, the SCA and BPEL artifacts are packed into a single EAR file. The following figure depicts the contents of this enhanced archive:

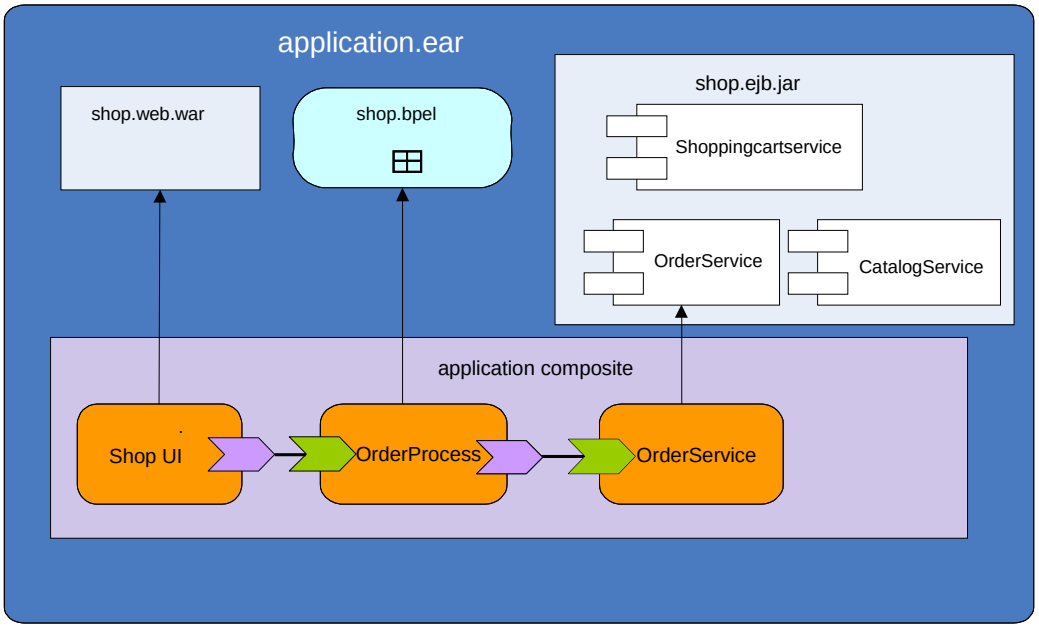
Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8



1210

1211 ***Figure 10: Example Order Application packaged as a single EAR***

1212

1213 An advantage of deploying an SCA-enhanced Java EE archive is that we can leverage the tooling,
1214 monitoring and application lifecycle management capability already present on the Java EE server.

1215 A.2 Extensibility for Java EE Applications

1216 An example of combining SCA and Java EE is the solution for the following problem -- a company (let's
1217 call it Acme) wishes to provide a Java EE application to its customer so that the customer can integrate
1218 this application into its own environment. Ideally the application should have some predefined "extension
1219 points" which allow the customer to hook in their own implementations replacing the default one. For
1220 example, the customer may wish to override some specific logic provided by the company Acme in an
1221 EJB and instead introduce its own existing functionality written in some non-Java programming model or
1222 via some of the predefined SCA possibilities (another EJB, JMS, WS call, etc.)

1223 Here it is assumed, that Acme predefine explicitly some extension points. Another possible use case that
1224 optionally some SCA runtimes may support is to allow each remote EJB reference in the Java EE
1225 application to be reconfigured (see section - 7.1.3 - The Component Type of a non-SCA-enhanced Java
1226 EE Application for more information).

1227 The provision of the extension point by Acme can be done in several ways

1228 - a fine grained approach using implementation.ejb as in section 5.1 or using implementation.jee as
1229 in section 7
1230 - by explicit usage of componentType side files
1231 - by exposing extension points via the @Reference annotation
1232 - via usage of application.composite with includeDefaults
1233 - via usage of other composite definitions.

1234 Here is an example of just one such approach:

1235 The EJB from Acme would look like

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

```
package com.acme.extensibility.sample;
import javax.ejb.Stateless;
import org.osoa.sca.annotations.Reference;


@Stateless(name=" ACMEBean ")
public class BaseBean implements BaseLocal {
...
```

The default value for the field ***extensionPoint*** is the EJB as defined by the Java EE specifications, however the usage of the SCA @Reference annotation declares that it is possible for SCA to override that and inject a reference proxy capable of providing the service according to SCA rules.

```
private @Reference @EJB com.acme.extensibility.ExtensionInterface
extensionPoint;

public void  businessLogic() {
            extensionPoint.doSomething();
}
```

In order to contribute to the SCA domain and expose the reference, the Acme company puts the following two artifacts in the META-INF directory of the EAR :

```
<?xml version="1.0" encoding="UTF-8"?>
<contribution xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
            xmlns:acme="http://www.acme.com.org">
            <deployable composite="acme:AcmeCompositeName"/>
</contribution>
```

Formatted: French France

Deleted: 712

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="AcmeCompositeName"
   targetNamespace="http://www.acme.com"
   xmlns:acme="http://www.acme.com.org"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">

   <component name="ACME_component ">
      <implementation.ejb ejb-link="ACMEJAR.jar#ACMEBean "/>
      <reference name="extensionPoint" requires="ejb">
         <interface.java
            interface="com.acme.extensibility.ExtensionInterface"/>
      </reference>
   </component>
</composite>
```

Deleted: 712

Formatted: French France

Note: The “extensionPoint” reference is an EJB reference inside ACMEBean, so the service wired to this reference must be able to provide the “ejb” intent as described in section 6.6.

After exposing the extension point in such a way and delivering the EAR to the customer, the customer can wire the EJB to its own non-Java technology xyz using SCA. The following contribution to the domain demonstrates how this can be done, where the customer's implementation is used for the component ***CustomerCode*** and a separate ***<wire/>*** element is used to link the reference of the ACME_Component to the service provided CustomerCode:

```
<?xml version="1.0" encoding="UTF-8"?>
<composite name="CompositeName"
   targetNamespace="http://www.org.customer.foo"
   xmlns:customer="http://www.org.customer.foo"
   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
```

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 712

Deleted: 8

```
   <component name="CustomerCode">
      <implementation.java class="org.customer.foo.ExtensionImpl"/>
      <service name="ExtensionTarget">
         <interface.java
            interface="com.acme.extensibility.ExtensionInterface"/>
      </service>
   </component>
   <wire source="ACME_component/extensionPoint"
         target="CustomerCode/ExtensionTarget"/>
</composite>
```

Formatted: English U.S.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

B. Support for SCA Annotations

The following table provides information whether SCA annotations are supported in EJB classes or session bean interfaces. Some of the annotations defined in the Java Common Annotations and APIs specification [4] are redundant to Java EE annotations and concepts. These are labeled as "May be supported", it is expected for SCA runtimes supporting these annotations to detect impossible combinations that violate the Java EE specifications and reject such deployments. Other annotations are labeled as “may be supported” because they represent optional features.

Annotation	Support Level	Comment
AllowsPassByReference	MAY be supported	This is a hint to the runtime, which can be disregarded
Callback	MUST be supported	
ComponentName	MUST be supported	
Constructor	MUST NOT be supported	There are no constructors in EJBs
Context	MUST be supported	
Conversational	MUST be supported	Each interface of a stateful EJB is treated as it has @Conversational, so the annotation is redundant. In case of stateless EJB-s the stateless semantics still remains, please see the comment for @ConversationID
ConversationAttributes	MAY be supported	Providing ways to control the expiration of statefull EJBs by maxAge, maxIdleTime
ConversationID	MUST be supported for stateful MAY be supported for stateless	If there is @Conversational on the interface of stateless bean, the conversationID will be generated by the runtime and may be inserted, the stateless semantic will still be in effect
Destroy	MAY be supported	Equivalent to @PreDestroy in EJB
EagerInit	MUST NOT be supported	There is no composite scope, it has no meaning
EndsConversation	MAY be supported	Methods that are marked @Remove should be treated as if the corresponding interface method is marked @EndsConversation.Interface methods marked @EndsConversation MUST have corresponding implementation methods marked @Remove.
Init	MAY be supported	Equivalent to @postConstruct in EJB
Authentication , Confidentiality, Integrity , Intent, PolicySets, Requires	MUST be supported on fields already annotated with	

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

	@Reference MAY be supported on class, session bean interface or on field annotated with @EJB	
OneWay	MUST be supported on fields already annotated with @Reference MUST be supported as an annotation on interface methods. MUST NOT be supported on class, session bean interface or on field annotated with @EJB	There are async calls in EJB 3.1
Property	MUST be supported	
Reference	MUST be supported	
Remotable	MAY be supported	Redundant to @Remote.
Scope	MAY be supported	@Stateless and @Stateful are mappings of SCA stateless, and conversational scopes. Note that Composite scope is not permitted.
Service	MAY be supported	

Note that if an SCA runtime does not support one of the optionally supported annotations (ie those marked "MAY" above) then the runtime SHOULD report an error if it encounters one of these annotations in an EJB.

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8

C. XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
   elementFormDefault="qualified">

   <xs:include schemaLocation="sca-core.xsd"/>

   <xs:element name="implementation.ejb" type="EJBImplementation"
      substitutionGroup="implementation"/>
   <xs:complexType name="EJBImplementation">
      <xs:complexContent>
         <xs:extension base="Implementation">
            <xs:sequence>
               <xs:any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="ejb-link" type="xs:string" use="required"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="implementation.web" type="WebImplementation"
      substitutionGroup="implementation"/>
   <xs:complexType name="WebImplementation">
      <xs:complexContent>
         <xs:extension base="Implementation">
            <xs:sequence>
               <xs:any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="web-uri" type="xs:string" use="required"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
   <xs:element name="implementation.jee" type="JEEImplementation"
      substitutionGroup="implementation"/>
   <xs:complexType name="JEEImplementation">
      <xs:complexContent>
         <xs:extension base="Implementation">
            <xs:sequence>
               <xs:any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="archive" type="xs:string" use="required"/>
            <xs:anyAttribute namespace="##other" processContents="lax"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:schema>
```

Deleted: 712

Deleted: 712

Formatted: German Germany

Formatted: French France

Formatted: English U.S.

Formatted: French France

Formatted: English U.S.

Formatted: French France

Formatted: English U.S.

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

D. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Henning Blohm	SAP AG*
David Booz	IBM
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Roberto Chinnici	Sun Microsystems
Pyounguk Cho	Oracle Corporation
Eric Clairambault	IBM
Mark Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
Mike Edwards	IBM
Raymond Feng	IBM
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Michael Keith	Oracle Corporation
Rainer Kerth	SAP AG*
Meeraj Kunnumpurath	Individual
Simon Laws	IBM
Yang Lei	IBM
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Jeff Mischkinsky	Oracle Corporation
Sriram Narasimhan	TIBCO Software Inc.
Simon Nash	Individual
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*
Ramkumar Ramalingam	IBM
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vladimir Savchenko	SAP AG*
Pradeep Simha	TIBCO Software Inc.
Raghav Srinivasan	Oracle Corporation

Deleted: 5

Deleted: 2

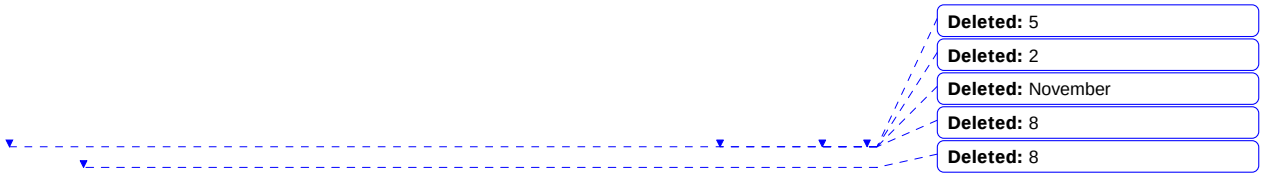
Deleted: November

Deleted: 8

Deleted: 8

Scott Vorthmann TIBCO Software Inc.
Feng Wang Primeton Technologies, Inc.
Robin Yang Primeton Technologies, Inc.

1368
1369



E. Non-Normative Text

Deleted: 5

Deleted: 2

Deleted: November

Deleted: 8

Deleted: 8

F. Revision History

Revision	Date	Editor	Changes Made
WD01	2008-08-15	Anish Karmarkar	Applied the OASIS template to the submission document.
WD02	2008-08-20	Dave Booz	Restructured the document to provide better flow for readers.
WD03	2008-08-22	Mike Edwards	Editorial changes from top to bottom. Recast all diagrams as PowerPoint format and adjusted text appearance and size.
WD04	2008-10-31	Dave Booz	Various editorials from committee review
WD04b	2008-11-07	Mike Edwards	Issue 83 resolution applied
WD05	2008-11-24	Mike Edwards	Agreed editorial changes from committee review
WD06	2009-09-14	Dave Booz	Application of issues: 89, 90, 116, 178

Deleted: 5
Deleted: 2
Deleted: November
Deleted: 8
Deleted: 8