Service Component Architecture Spring Component Implementation Specification Version 1.1

Committee Specification Draft 01 / Public Review Draft 01

23 May 2011

Specification URIs:

This version:

http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csprd01/sca-springci-spec-v1.1-csprd01.doc
http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csprd01/sca-springci-spec-v1.1-csprd01.pdf (Authoritative)
http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csprd01/sca-springci-spec-v1.1-csprd01.html

Previous version:

N/A

Latest version:

http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.doc
http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.pdf (Authoritative)
http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/sca-springci-spec-v1.1.html

Technical Committee:

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

Chairs:

David Booz, IBM
Anish Karmarkar, Oracle

Editors:

David Booz, IBM
Mike Edwards, IBM
Anish Karmarkar, Oracle

Related work:

This specification replaces or supersedes:

- Service Component Architecture Spring Component Implementation Specification Version 1.00, March 21 2007

This specification is related to:

- Service Component Architecture Assembly Model Specification Version 1.1
- SCA Policy Framework Version 1.1
- XML schemas: sca-springci-spec/v1.1/csprd01/xsd/

Declared XML namespaces:

http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810
http://docs.oasis-open.org/ns/opencsa/sca/200912

Abstract:

The SCA Spring component implementation specification specifies how the Spring Framework **[SPRING]** can be used with SCA. This specification extends the SCA Assembly Model by defining how a Spring Framework application context provides an implementation of an SCA component, including its various attributes such as services, references, and properties and how that application context is used in SCA as a component implementation type. The goals of this specification are:

Coarse-grained integration: The integration with Spring is at the SCA Component level, where a Spring application context provides a component implementation, exposing services and using references via SCA. This means that a Spring application context defines the internal structure of an implementation.

Start from SCA Component Type: Use of Spring Framework to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

Start from Spring context: Generation of an SCA Component from any Spring application context and use that component within an SCA assembly.

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" 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).

Citation format:

When referencing this specification the following citation format should be used:

[SCA-Spring-CI-v1.1]

Service Component Architecture Spring Component Implementation Specification Version 1.1. 23 May 2011. OASIS Committee Specification Draft 01 / Public Review Draft 01. http://docs.oasis-open.org/opencsa/sca-j/sca-springci-spec/v1.1/csprd01/sca-springci-spec-v1.1-csprd01.pdf.

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 name "OASIS" is a trademark 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.

Table of Contents

1 Introduction

The SCA Spring component implementation specification specifies how the Spring Framework **[SPRING]** can be used with SCA. This specification extends the SCA Assembly Model by defining how a Spring Framework application context provides an implementation of an SCA component, including its various attributes such as services, references, and properties and how that application context is used in SCA as a component implementation type. The goals of this specification are:

Coarse-grained integration: The integration with Spring is at the SCA Component level, where a Spring application context provides a component implementation, exposing services and using references via SCA. This means that a Spring application context defines the internal structure of a component implementation.

Start from SCA Component Type: Use of Spring Framework to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

Start from Spring context: Generation of an SCA Component from any Spring context and use that component within an SCA assembly.

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*, IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt

[SCA-ASSEMBLY] OASIS Committee Draft 06, "SCA Assembly Model Specification V1.1", August 2010 http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.pdf

[SCA-POLICY] OASIS Committee Draft 04, "SCA Policy Framework Specification Version 1.1", September 2010 http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd04.pdf

[JAVA-CAA] OASIS Committee Draft 05, "SCA Java Common Annotations and APIs Specification Version 1.1", November 2010 http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-csd05.pdf

[SPRING] Spring Framework Specification http://static.springsource.org/spring/docs/2.5.x/reference/index.html

1.3 Non-Normative References

[SCA-SPRING-TC] OASIS Working Draft 01, "TestCases for the Spring Component Implementation Specification V1.1", April 2011 http://www.oasis-open.org/committees/download.php/41746/sca-springci-1.1-testcases-wd01.odt

1.4 Testcases

The TestCases for the SCA Spring Component Implementation Specification Version 1.1 **[SCA-SPRING-TC]** defines the TestCases for the SCA-J Spring Component Implementation specification.The TestCases represent a series of tests that SCA runtimes are expected to pass in order to claim conformance to the requirements of the SCA-J Spring Component Implementation specification.

2 Spring application context as component implementation

A Spring Application Context can be used as an implementation of an SCA component.

Conceptually, this can be represented as follows:

Figure 1 below illustrates an SCA composite containing two components, both of which are implemented by Spring application contexts.

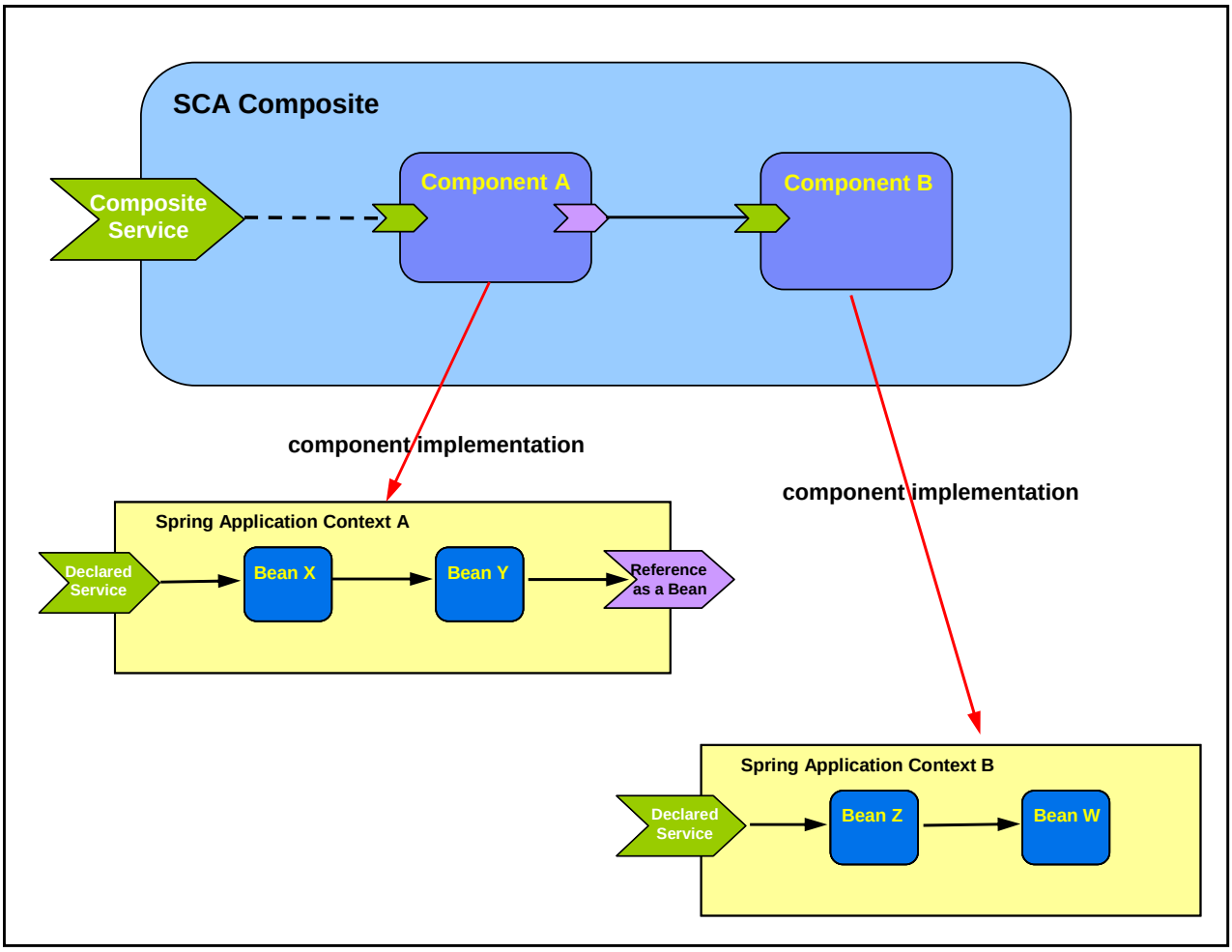

Figure 1 SCA Domain with two Spring application contexts as component implementations

Each component has one declared service. Component A is implemented by an application context, Context A, composed of two Spring beans. Here, bean X is exposed as an SCA service. Bean Y has a reference to an external SCA service. This service reference is wired to the service offered by the second component, Component B which is also implemented by another Spring context, Context B, which has a single declared service, which is provided by Bean Z.

A component that uses Spring for an implementation can wire SCA services and references without introducing SCA metadata into the Spring configuration. The Spring context does not need to know anything about the SCA environment. All binding and policy declarations occur in the SCA runtime implementation and does not enter into the Spring space.

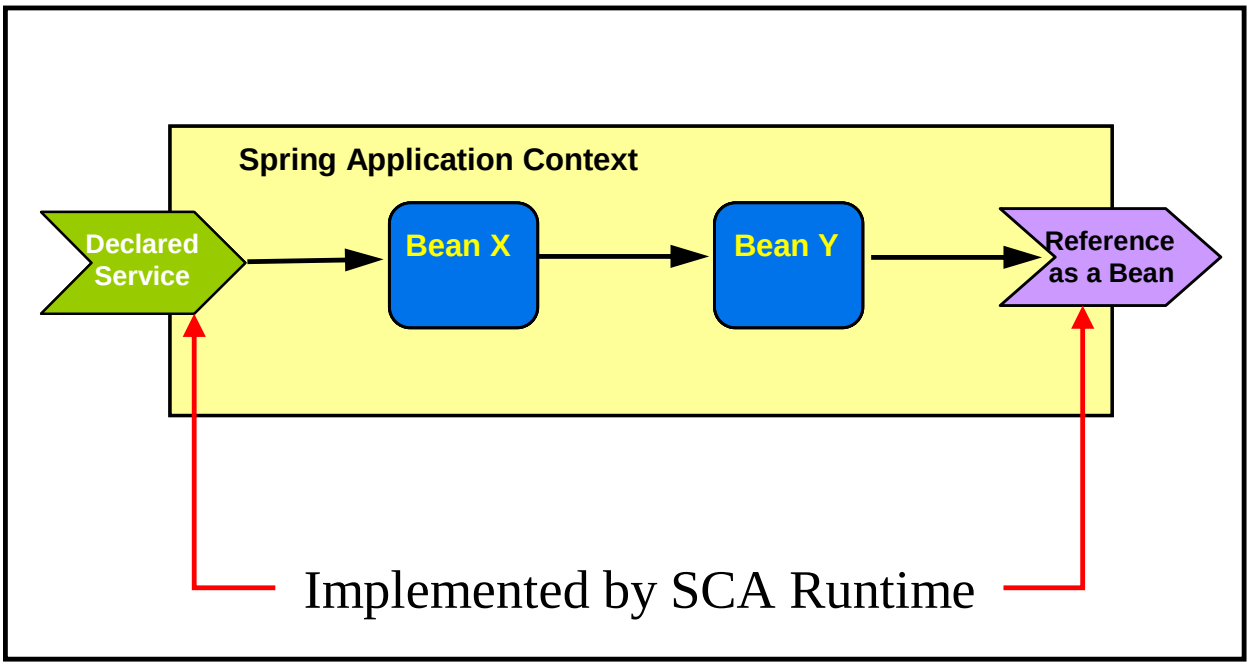

Figure 2 Relationship of Spring Application Context artifacts to SCA Services & References

Figure 2 shows two of the points where the SCA runtime interacts with the Spring context: services and references. Each service is offered by one of the Spring beans within the application context, while each reference is a property dependency of one or more of the Spring beans in the application context which is not satisfied by another bean. Services and references can be introspected from a Spring application context which has no SCA specific features. However, for greater control, it is possible to annotate the application context file with SCA-specific extensions, for example this can be done to limit the number of services offered by a particular application context.

Any policy specification is done in the SCA composite - and this extends to the running application where service calls into the Spring application context have policy related processing performed by the SCA runtime (e.g. decryption of encrypted messages) before the final message is delivered to the target Spring bean. In the same way, on outbound service invocations from the application context, references supplied by the SCA runtime can provide policy implementation.

2.1 Structure of a Spring Application Context

Spring **[SPRING]** applications are described by a declarative XML file called a Spring Application Context. The structure of the parts of a Spring Application context relevant to SCA is outlined in the following pseudo-schema

```
<beans>
   <bean id="xs:string" name="xs:string" class="xs:string"
      scope="xs:string">*
      <property name="xs:string" value="xs:string"? ref="xs:string"?>*
         <value type="xs:string"?/>?
         <bean/>?
         <ref bean="xs:string"? local="xs:IDREF"? parent="xs:string"?/>?
         <idref bean="xs:string" local="xs:IDREF"?/>?
         <list/>?
         <map/>?
         <set/>?
         <lookup-method/>?
         <replaced-method/>?
      </property>
```

```
        <constructor-arg ref="xs:string"? index="xs:string"
           type="xs:string"? value="xs:string"?>*
           <value/>?
           <bean/>?
           <ref bean="xs:string"/>?
           <idref bean="xs:string"/>?
           <list/>?
           <map/>?
           <set/>?
           <props/>?
        </constructor-arg>
        <meta/>*
        <qualifier/>*
        <lookup-method/>*
        <replaced-method/>*
        <any/>*
     </bean>
</beans>
```

Example 1: Pseudo-schema for the Spring Application Context

2.1.1 Spring Beans

The application context consists of a set of <bean/> definitions, where each bean is a Java class that can offer service(s) which are available for use by other beans - and in the context of SCA, a bean can become an SCA service of the component that uses the Spring application context as its implementation.

The Java class of a <bean/> is defined by its @class attribute.

2.1.1.1 Bean ID & Name

A <bean/> can be given either zero or one ID, and can be given zero or more names, using its @id and @name attributes. These names have to be unique within the application context. The id and names can be used to refer to the bean, for example, when one bean has a dependency on another bean.

However, it is possible for a bean to have no ID and no names. From an SCA perspective, such anonymous beans are purely for use within the application context - anonymous beans cannot be used for an SCA service, for example.

2.1.1.2 Inner Beans

As can be seen from the pseudo-schema in Example 1, it is possible to nest a <bean/> within another <bean/> declaration. Nested beans of this kind are termed ***inner beans***. Inner beans are purely for use within the application context and have no direct relationship with SCA.

2.1.1.3 Bean Properties

A <bean/> can have zero or more <property/> subelements. Each <property/> represents a dependency of the bean class, which have to be injected into the class when it is instantiated. Injection is typically be means of a setter method on the bean class.

From a Spring perspective, the property value is simply a Java primitive or Java class that is needed by the bean class. From an SCA perspective, a property could be an SCA property or a property could be an SCA reference to a target service, depending on the type of the <property/>.

2.1.1.4 Bean Constructor Arguments

A <bean/> can have zero or more <constructor-arg/> subelements. These elements are very similar to <property/> elements in that they represent a dependency of the bean class, which have to be injected into the class when it is instantiated. The difference between <constructor-arg/> elements and

<property/> elements is that <constructor-arg/> values are injected into the class through parameters on the bean class constructor method, rather than through setter methods.

2.1.2 Property and Constructor Argument References

<property/> and <constructor-arg/> elements can supply their dependencies "by value", through data held directly within the element, by means of the @value attribute, the <value/> subelement or the <bean/> subelement.

Collections can be supplied to a bean class by means of the <list/>, <set/> and <map/> subelements.

Of relevance to SCA are <property/> and <constructor-arg/> elements that supply their dependencies "by reference", where they contain references to data supplied elsewhere. Typically, these references are to other <bean/> elements in the same application context. However, when using a Spring application context within an SCA environment, the references can be to SCA references and SCA properties, configured by the SCA component using the application context as its implementation.

References are made using the @ref attribute and the <ref/> and <idref/> subelements of <property/> and <constructor-arg/> elements. It is also possible to have references within collections, since <list/>, <set/> and <map/> subelements can contain <ref/> and <idref/> entries.

Each @ref attribute, <ref/> element or <idref/> identifies another bean within the application context, via its ID or its one of its names.

For SCA, it is possible to have references of this type mapped to SCA references or SCA properties, simply by means of having those references left "dangling" - ie not pointing to any bean within the application context. Alternatively, SCA references and SCA properties can be explicity modelled within the Spring application context using extension elements, as described in the section "Explicit declaration of SCA related beans inside a Spring Application Context".

2.2 Direct use of SCA references within a Spring configuration

The SCA runtime hosting the Spring application context implementing a component creates a parent application context in which all SCA references are defined as beans using the SCA reference name as the bean name. These beans are automatically visible in the child (user application) context.

The following Spring configuration provides a model for Spring application context A, expressed in figure 1 above. In this example, there are two Spring beans, X and Y. The Spring bean named “X” provides a service which is configured and invoked via SCA and Spring bean Y contains a reference to a service which is supplied by SCA.

```
<beans>
    <bean id="X" class="org.xyz.someapp.SomeClass">
       <property name="foo" ref="Y"/>
     </bean>
    <bean id="Y" class"org.xyz.someapp.SomeOtherClass">
      <property name="bar" ref="SCAReference"/>
    </bean>
</beans>
```

Two beans are defined. The bean named “X” contains one property (i.e. reference) named “foo” which refers to the second bean in the context, named “Y”. The bean “Y” also has a single property named “bar”, which refers to the SCA service reference, with the name “SCAReference”

The SCA composite contains service and reference definitions for a component that uses the Spring application context as its implementation, with appropriate binding information:

```
<composite name="BazComposite">
  <component name="SpringComponent">
     <implementation.spring location="…"/>
```

```
        <service name="X"/>
        <reference name="SCAReference" .../> <!-- binding info specified -->
      </component>
    </composite>
```

The only part of this that is specific to Spring is the **<implementation.spring>** element. The **location** attribute of that element specifies the Spring application context file(s) to use, either as a direct pointer to a single file, or via a reference to an archive file or a directory that contains one or more Spring application context files (see the section "Specifying the Spring Implementation Type in an Assembly" for more details).

Each **<service>** element used with **<implementation.spring>** by default includes the name of the Spring bean that is to be exposed as an SCA service in its name attribute. So, for Spring, the name attribute of a service plays two roles: it identifies a Spring bean, and it names the service for the component. The service element above has a name of "X", so there is a Spring bean with that name. The SCA component also contains a <reference> element named "SCAReference". The reference name becomes an addressable name within the Spring application context – so, in this case, "SCAReference" can be referred to by bean "Y" in the Spring configuration above.

The SCA runtime is responsible for setting up the references and exposing them as beans with their indicated names in the spring context. This is usually accomplished by creating a parent context which has the appropriate beans defined and the context supplied by the implementation becomes the child of this context. Thus, the references – e.g. the "SCAReference" that bean "Y" uses for its "bar" property – are available to the context.

2.3 Explicit declaration of SCA related beans inside a Spring Application Context

It is possible to explicitly declare SCA-related beans inside a Spring application context. A bean within the application context can be declared to be an SCA service. References to beans made within the application context can be declared to be either SCA properties or SCA references.

These capabilities are provided by means of a set of SCA extension elements, which can be placed within a Spring application context. The SCA extension elements are declared in the SCA Spring Extension schema - sca-spring-extensions.xsd - which is shown in Appendix A. SCA extension elements within a Spring application context MUST conform to the SCA Spring Extension schema declared in sca-spring-extensions.xsd. [SPR20006]

For example, to declare a bean that represents the service referred to by an SCA reference named "SCAReference" the following is declared in the application context:

```
<sca:reference name="SCAReference" type="com.xyz.SomeType/>
```

The SCA Spring extension elements are:

- **<sca:reference>** This element defines a Spring bean representing an SCA service which is external to the Spring application context.
- **<sca:property>** This element defines a Spring bean which represents a property of the SCA component which configures the Spring composite.
- **<sca:service>** This element defines a bean that the Spring composite exposes as an SCA service.

2.3.1 SCA Service element

The SCA service element declares a service that is offered by the Spring application context as an SCA service. When an application context contains one or more SCA service elements, these elements declare all the services that are made available by the application context when it is used as a component implementation. In this way, the service elements provide the developer with a means to control which Spring beans are exposed as SCA services - if no SCA service elements are present in the application context, the default behaviour is to expose all the Spring beans as SCA services.

The SCA service element can also declare other attributes of the SCA service. In particular, policy can be associated with the service using the @requires and @policySets attributes.

The pseudo-schema for the service element is:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sca=
          "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"

...
    <sca:service name="xs:NCName"
                        type="xs:NCName"?
                        target="xs:NCName"
                        requires="list of xs:QName"?
                        policySets="list of xs:QName"?/>
...

</beans>
```

The ***service*** element has the following ***attributes***:

- ***name : NCName (1..1)*** - the name of the service. The value of the @name attribute of an <sca:service/> subelement of a <beans/> element MUST be unique amongst the <service/> subelements of the <beans/> element. [SPR20001]
- ***type : NCName (0..1)*** - the type of the service, declared as the fully qualified name of a Java class. If omitted, the type of the service is introspected from the Spring bean class identified by the @target attribute.
- ***target : NCName (1..1)*** - the name of a <bean/> element within the application context which provides the service declared by the sca:service element.The @target attribute of a <service/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element. [SPR20002]
- ***requires : QName (0..1)*** - a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- ***policySets : QName (0..1)*** - a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.

2.3.2 SCA Reference element

The SCA reference element declares an SCA reference that is made by the Spring application context. When an application context contains one or more SCA reference elements, each of these elements acts as if it were a Spring <bean/> element, offering a target which can satisfy a reference from a <bean/> element within the application context. Each SCA reference element appears as an reference element in the componentType of the Spring implementation and the reference can be configured by the SCA component using that implementation - in particular, the reference can be wired to an appropriate target service.

The SCA reference element can also declare other attributes of the SCA reference. In particular, policy can be associated with the reference using the @requires and @policySets attributes.

The pseudo-schema for the reference element is:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sca=
           "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"

...
```

```
        <sca:reference name="xs:NCName"
                       type="xs:NCName"
                       default="xs:NCName"?
                       requires="list of xs:QName"?
                       policySets="list of xs:QName"?/>
    ...
    </beans>
```

The ***reference*** element has the following ***attributes***:

- ***name : NCName (1..1)*** - the name of the reference. The value of the @name attribute of an <sca:reference/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <reference/> subelements, <property/> subelements and the <bean/> subelements of the <beans/> element. [SPR20003]
- ***type : NCName (1..1)*** - the type of the reference, declared as the fully qualified name of a Java class.
- ***default : NCName (0..1)*** - the name of a <bean/> element within the application context which provides the reference declared by the sca:reference element if the component using the application context as an implementation does not wire the reference to a target service.The @default attribute of a <reference/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element. [SPR20004]
- ***requires : QName (0..1)*** - a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- ***policySets : QName (0..1)*** - a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.

2.3.3 SCA Property element

The SCA property element declares an SCA property which can be used by the Spring application context. When an application context contains one or more SCA property elements, each of these elements acts as if it were a Spring <bean/> element, offering a target which can satisfy a referece from a <bean/> element within the application context. Each SCA property element appears as a property element in the componentType of the Spring implementation and the property can be configured by the SCA component using that implementation - the component can provide a value for the property.

The pseudo-schema for the property element is:

```
    <beans xmlns="http://www.springframework.org/schema/beans"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca-
    j/spring/200810"

    ...
        <sca:property name="xs:NCName"
                      type="xs:NCName"/>
    ...
    </beans>
```

The ***property*** element has the following ***attributes***:

- ***name : NCName (1..1)*** - the name of the property. The value of the @name attribute of an <sca:property/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <property/> subelements, <reference/> subelements and the <bean/> subelements of the <beans/> element. [SPR20005]
- ***type : NCName (1..1)*** - the type of the property, declared as the fully qualified name of a Java class.

2.3.4 Example of a Spring Application Context with SCA Spring Extension Elements

The following example shows a Spring application context that exposes one service, SCAService, and explicitly defines an SCA reference, SCAReference. The "bar2" property of bean Y is configured with an SCA property with name "sca-property-name".

```
<beans>

   <!-- An explicit reference, which is used by bean "Y" -->
   <sca:reference name=”SCAReference” type="com.xyz.SomeType"/>

   <bean name="X">
      <property name="foo" ref="Y"/>
   </bean>

   <bean name="Y">
      <property name="bar" ref="SCAReference"/>
      <property name="bar2" ref="sca-property-name"/>
   </bean>

   <!-- expose an SCA property named “sca-property-name” -->
   <sca:property name="sca-property-name" type="java.lang.String"/>

   <!-- Expose the bean "X" as an SCA service named "SCAService" -->
   <sca:service name="SCAService" type="org.xyz.someapp.SomeInterface"
      target="X"/>

</beans>
```

The componentType of the application context is:

```
<componentType>

   <service name="SCAService>
      <interface.java interface="org.xyz.someapp.SomeInterface"/>
   </service>

   <reference name="SCAReference">
      <interface.java interface="com.xyz.SomeType"/>
   </reference>

   <property name="sca-property-name" type="xsd:string"/>

</componentType>
```

2.3.5 Example of a Spring Application Context without Extension Elements

The following example shows a Spring application context that has no SCA extension elements. The application context has one bean, named "X" that is exposed as an SCA service, while a second bean, "Y" has a property that is not satisfied within the application context and has an (introspected) type which is an interface - in SCA terms this unsatisfied property is a reference.

```
<beans>

   <bean id="X" class="org.xyz.someapp.SomeClass">
      <property name="foo" ref="Y"/>
   </bean>
```

```
        <bean id="Y" class"org.xyz.someapp.SomeOtherClass">
           <property name="bar" ref="SCAReference"/>
        </bean>

    </beans>
```

The componentType of the application context is:

```
    <componentType>

        <service name="X">
           <interface.java interface="org.xyz.someapp.SomeClass"/>
        </service>

        <reference name="SCAReference">
           <interface.java interface="com.xyz.SomeInterface"/>
        </reference>

    </componentType>
```

Here it is assumed that the class "org.xyz.someapp.SomeClass" defines it own interface and that the introspected type of the "bar" property of bean "Y" is "com.xyz.SomeInterface".

2.4 Handling multiple application contexts in SCA runtime

When the <implementation.spring> element's @location attribute specifies the Spring application context file(s) to use via a reference to an archive file or a directory (see the section "Specifying the Spring Implementation Type in an Assembly" for more details) and that location contains more than one Spring application context file, then the SCA runtime has to create a combined application context for the collection of paths identified by the "Spring-Context" header in the MANIFEST.MF file.

As an example, take the “Spring-Context” header in the MANIFEST.MF file defined as shown below:

```
Spring-Context: application-context1.xml; application-context2.xml;
                application-context3.xml
```

In this case, the SCA runtime has to construct an application context for the set of files identified from the “Spring-Context” header in the MANIFEST.MF file, by configuring the individual application contexts in a hierarchy such that a child application context can see beans defined in a parent, but not vice-versa.

In multiple application context scenario, each individual application context definition file identified from the “Spring-Context” header in the MANIFEST.MF file, can have its own SCA services, references and properties defined either implicitly or explicitly.

Spring supports the loading of multiple application contexts through other mechanisms. For example, application contexts can be loaded in a parent/child hierarchy using the Spring ClassPathXmlApplicationContext:

```
<beans>
  <bean name=”bean1” class=”…….”/>
  <bean name=”bean2” class=”…….”/>
  <bean
  class=”org.springframework.context.support.ClassPathXmlApplicationContext”>
    <constructor-arg>
      <list>
```

```
            <value>context1.xml</value>
            <value>context2.xml</value>
            <value>context3.xml</value>
        </list>
      </constructor-arg>
    </bean>
</beans>
```

In this case, the 3 contexts context1.xml, context2.xml, context3.xml are loaded by the ClassPathXmlApplicationContext bean as child application contexts. Such application contexts can be loaded and used when the parent context is used as an SCA component implementation, but these application contexts do not contribute to the componentType of the Spring implementation (and they are not introspected by the SCA Spring runtime).

In multiple application context scenario, the SCA runtime MUST raise an error when multiple <sca:service> elements are identified with the same name amongst the set of application context files identified from the “Spring-Context” header in the MANIFEST.MF file. [SPR20007]

Spring supports duplicate bean definitions for multiple application context scenarios. For example, a bean with the same id or name can be defined in multiple application contexts and in such cases Spring overrides the older bean definition with the later bean definition. When no <sca:service/> element is present in any of the application context file identified from the collection of application context paths identified by the "Spring-Context" header in the MANIFEST.MF file, then the SCA runtime MUST use implicit service determination only for the later bean definition. [SPR20008]

In multiple application context scenario, the SCA runtime MUST determine the componentType by applying the rules defined in the section “Component Type of a Spring Application Context” to the combined application context and not to the individual application context files. [SPR20009]

For example, when at least one <sca:service/> element is present in any one of the application context file identified from the collection of paths identified by the "Spring-Context" header in the MANIFEST.MF file, then no implicit service determination is used for any of the application contexts and only services explicitly declared with <sca:service/> elements appear in the componentType of the Spring implementation.

3 Component Type of a Spring Application Context

An SCA runtime MUST introspect the componentType of an implementation.spring application context following the rules defined in the section "Component Type of a Spring Application Context". [SPR30001]

The introspected component type MUST be a compliant SCA Java XML Document as defined in section 12.1 of [JAVA-CAA]. [SPR30003] The introspected component type uses <interface.java> as the interface type for the introspected services and references. Section 12.1 of [JAVA-CAA] requires compliance with the all requirements of <interface.java>.

The component type of a Spring Application Context is introspected from the application context as follows:

A <service/> element exists for each <sca:service/> element in the application context, where:

- @name attribute is the value of the @name attribute of the sca:service element
- @requires attribute is omitted unless the <sca:service/> element has a @requires attribute, in which case the @requires attribute is present with its value equal to the value of the @requires attribute of the <sca:service/> element
- @policySets attribute is omitted unless the <sca:service/> element has a @policySets attribute, in which case the @policySets attribute is present with its value equal to the value of the @policySets attribute of the <sca:service/> element
- interface.java child element is present with the @interface attribute set to the fully qualified name of the interface class identified by the @type attribute of the sca:service element. If the @type attribute is not present on the <sca:service/> element, then the interface.java element has its @interface attribute set to the fully qualified name of the Java class of the spring <bean/> element identified by the @target attribute of the <sca:service/> element.
- binding child element is omitted
- callback child element is omitted

If there are no <sca:service/> elements in the application context, one <service/> element exists for each service implemented by each top-level <bean/> element in the application context ***except*** for bean elements where any of the following apply:

- <bean/> elements @class attribute is absent
- <bean/> elements @abstract attribute value is set to “true”
- <bean/> elements @factory-bean attribute value is set
- <bean/> elements @factory-method attribute value is set
- <bean/> elements @parent attribute value is set to reference another bean in the application context
- <bean/> elements @class attribute value is set to reference the native spring binary classes starting with "org.springframework"

where each <service/> element has the following characteristics:

- @name attribute value is the value of the @id attribute of the <bean/> element if present, otherwise it is the first name from the value of @name attribute of the <bean/> element
- @requires attribute is omitted
- @policySets attribute is omitted
- interface.java child element is present with the @interface attribute set to the fully qualified name of the interface class introspected from the bean class declared in the @class attribute of the

<bean/> element, as follows:
- if the bean class implements zero Java interfaces annotated with @Remotable, the interface class is the bean class itself
- if the bean class implements exactly one Java interface that is annotated with @Remotable, the interface class is the Java interface class which is annotated with @Remotable
- where there are no explicit <sca:service/> elements in the application context, if a bean class implements two or more Java interfaces which are annotated with @Remotable, this is not allowed and the SCA runtime MUST raise an error. [SPR30005]

- binding child element is omitted
- callback child element is omitted

Note that as described in the SCA Assembly Model specification

[SCA-ASSEMBLY] the @name attribute has to be unique amongst all <service/> elements in the componentType.

Where a Spring Bean implementation class implements more than one interface, the Bean can be exposed as either a single service or as multiple services, through the use of explicit <sca:service/> elements, where each <sca:service/> element references the same <bean/> element but where the @type attribute uses only one of the interfaces provided by the bean.

Where there are no <sca:service/> elements, the bean is exposed as a single service with an interface that is the defined either by the bean class itself, or is defined by a single @Remotable interface implemented by the bean class. It is not premitted for the bean class to implement two or more @Remotable interfaces in this case - this can only be done with the use of explicit <sca:service/> elements.

Note that a <bean/> element nested within another <bean/> element (an inner bean) is never exposed directly as an SCA service.

A <reference/> element exists for each <sca:reference/> element in the application context, where:

- @name attribute is the value of the @name attribute of the sca:reference element
- @autowire attribute is omitted
- @wiredByImpl attribute is omitted
- @target attribute is omitted
- @multiplicity attribute is set to (1..1) unless the <sca:reference/> element has the @default attribute present in which case it is set to (0..1)
- @requires attribute is omitted unless the <sca:reference/> element has a @requires attribute, in which case the @requires attribute is present with its value equal to the value of the @requires attribute of the <sca:reference/> element
- @policySets attribute is omitted unless the <sca:reference/> element has a @policySets attribute, in which case the @policySets attribute is present with its value equal to the value of the @policySets attribute of the <sca:reference/> element
- interface.java child element is present, with the interface attribute set to the fully qualified name of the interface class identified by the @type attribute of the <sca:reference/> element
- binding child element is omitted
- callback child element is omitted

A <property/> element exists for each <sca:property/> element in the application context, where:

- @name attribute is the value of the @name attribute of the <sca:property/> element
- @value attribute is omitted

- @type attribute is set to the XML type implied by the JAXB mapping of the Java class identified by the @type attribute of the <sca:property/> element
- @element attribute is omitted
- @many attribute is set to "false"
- @mustSupply attribute is set to "true"

IF there are no <sca:reference/> elements AND no <sca:property> elements in the application context, then references and properties are defined by the bean references in the application context which are not found in the application context as follows:

A <reference/> element exists for each unique bean reference in the application context to a bean which is not found in the application context and where the bean reference refers to a Java interface class:

- @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/> element that makes the reference, or the reference name derived from the subelements of the <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- @autowire attribute is omitted
- @wiredByImple attribute is omitted
- @target attribute is omitted
- @multiplicity attribute is set to (1..1)
- @requires attribute is omitted
- @policySets attribute is omitted
- interface.java child element is present, with the interface attribute set as follows:
    1. if only one bean refers to the bean reference, then the interface attribute is set to the fully qualified name of the interface class identified by the bean reference
    2. if two or more beans refer to the bean reference, each bean reference identifies an interface class. Each interface class in the collection of interface classes has to be either the same as, or an ancestor of, or a descendent of, every other interface class in that collection. If this condition does not hold true then the SCA runtime MUST raise an error. [SPR30002] . The interface attribute is set to the fully qualified name of the interface class which has the highest depth in the inheritance tree in the set of interface classes - i.e. it is set to the most specific subclass amongst all the interface classes identified by the bean references.

       For example, if two bean A and B refer to a bean reference C, and the interface class identified by bean A for reference C is I1, and that of bean B is I2, and if I2 is a subclass of I1, then the interface attribute value for the introspected implicit reference is set to I2.
- binding child element is omitted
- callback child element is omitted

A <property/> element exists for each unique bean reference in the application context to a bean which is not found in the application context and where the bean reference does not refer to a Java interface class:

- @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/> element that makes the reference, or the reference name derived from the subelements of the <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- @value attribute is omitted
- @type attribute is set to the XML type implied by the JAXB mapping of the Java class identified by the bean reference

- @element attribute is omitted
- @many attribute is set to "false"
- @mustSupply attribute is set to "true"

The Spring Component Implementation type does not support the use of Component Type side files, as defined in the SCA Assembly Model specification

[SCA-ASSEMBLY], so that the effective componentType of a Spring Application Context is determined completely by introspection of the Spring Application Context.

It is beyond the scope of this specification to define the interpretation of the annotations specified in the SCA Common Annotations and API Specification

[JAVA-CAA]. , except for those annotations that are required to be supported by the SCA Java interface.

The SCA runtime MUST support the SCA annotations which are applicable to an interface class which is referenced by an <interface.java/> element in the introspected component type of a Spring application context. [SPR30004]

Other than the annotations which apply to a Java interface referenced from an <interface.java/> element in the component type, an implementation can ignore SCA annotations that are present in classes used by the application context.

3.1 Introspecting the Type Implied by a Spring Bean Reference

In the case where a reference or a property in the component type is derived by introspection of bean references, the type of the reference or property is determined by introspection of the related property setter method or constructor method of the Bean which is the source of the reference.

In some cases, the type introspected by this process could be a generic type - for example a List<?>. In such cases, the formal type of the reference becomes Object. This will be interpreted as an SCA property with a Java type of Object, which maps to an XML type of <any/>.

4 Specifying the Spring Implementation Type in an Assembly

The following pseudo-schema defines the implementation element schema used for the Spring implementation type:.

```
<implementation.spring location="xs:anyURI"
                       requires="list of xs:QName"?
                       policySets="list of xs:QName"?/>
```

The implementation.spring element has the following attributes:

location : anyURI (1..1) – a URI pointing to the location of the Spring application context to use as the implementation.

The location URI can either be an absolute URI or it can be a relative URI. In the case where the location URI is a relative URI, the URI MUST be taken as being relative to the base of the contribution which contains the composite containing the <implementation.spring/> element. [SPR40009]

The implementation.spring @location attribute URI value MUST point to one of the following:
a) a Spring application context file
b) a Java archive file (JAR)
c) a directory
[SPR40001]

If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context. [SPR40002]
If the implementation.spring @location URI identifies a JAR archive file, then the file META-INF/MANIFEST.MF MUST be read from the archive. [SPR40003]
If the implementation.spring @location URI identifies a directory, then the file META-INF/MANIFEST.MF underneath that directory MUST be read from the directory. [SPR40004]

If the MANIFEST.MF file contains a header "Spring-Context" of the format:
  Spring-Context ::= path (';' path)*
where path is a relative path with respect to the @location URI, then each path specified in the header MUST identify a Spring application context configuration file. [SPR40008]

If present, all the Spring application context configuration files identified by the "Spring-Context" header in the MANIFEST.MF file MUST be collectively used to build the Spring application context for implementation.spring element. [SPR40005]

If there is no MANIFEST.MF file or if there is no Spring-Context header within the MANIFEST.MF file, the Spring application context MUST be built using all the *.xml files in the META-INF/spring subdirectory within the JAR identified by the @location URI or underneath the directory specified by the @location URI. [SPR40006]

- ***requires : QName (0..n)*** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- ***policySets : QName (0..n)*** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.

686 The <implementation.spring> element MUST conform to the schema defined in sca-implementation-
687 spring.xsd. [SPR40007]
688

5 Conformance

The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification, are considered to be authoritative and take precedence over the XML schema defined in the appendix of this document.

There are three categories of artifacts that this specification defines conformance for: SCA Spring Component Implementation Composite Document, SCA Spring Application Context Document and SCA Runtime.

5.1 SCA Spring Component Implementation Composite Document

An SCA Spring Component Implementation Composite Document is an SCA Composite Document, as defined by the SCA Assembly Model Specification Section 13.1 [ASSEMBLY], that uses the <implementation.spring> element. Such an SCA Spring Component Implementation Composite Document MUST be a conformant SCA Composite Document, as defined by [ASSEMBLY], and MUST comply with additional constraints on the document content as defined in Appendix B.

5.2 SCA Spring Application Context Document

An SCA Spring Application Context Document is a Spring Framework Application Context Document, as defined by the Spring Framwork Specification [SPRING], that uses the SCA Spring extensions defined in Section 2. Such an SCA Spring Application Context Document MUST be a conformant Spring Framework Application Context Document, as defined by [SPRING], and MUST comply with the requirements specified in Section 2 of this specification.

5.3 SCA Runtime

An implementation that claims to conform to this specification MUST meet the following conditions:

1. The implementation MUST meet all the conformance requirements defined by the SCA Assembly Model Specification [ASSEMBLY].
2. The implementation MUST reject an SCA Spring Component Implementation Composite Document that does not conform to the sca-implementation-spring.xsd schema.
3. The implementation MUST reject an SCA Spring Application Context Document that does not conform to the sca-spring-extensions.xsd schema.
4. The implementation MUST comply with all statements related to an SCA Runtime, specified in 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have to be implemented.

A. XML Schemas

A.1 sca-implementation-spring.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2011. All Rights Reserved.
     OASIS trademark, IPR and other policies apply.  -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">

  <include schemaLocation="sca-core-1.1-cd06.xsd"/>
  <element name="implementation.spring" type="sca:SpringImplementation"
         substitutionGroup="sca:implementation"/>
  <complexType name="SpringImplementation">
    <complexContent>
      <extension base="sca:Implementation">
        <sequence>
          <any namespace="##other" processContents="lax" minOccurs="0"
            maxOccurs="unbounded"/>
        </sequence>
        <attribute name="location" type="anyURI" use="required"/>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

A.2 SCA Spring Extension schema - sca-spring-extensions.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2011. All Rights Reserved.
     OASIS trademark, IPR and other policies apply.  -->
<xsd:schema
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  xsi:schemaLocation="
       http://docs.oasis-open.org/ns/opencsa/sca/200912
       http://docs.oasis-open.org/opencsa/sca-assembly/sca-core-1.1-cd06.xsd"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace=
    "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810">

      <xsd:element name="reference">
            <xsd:complexType>
                  <any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:attribute name="name" type="xsd:NCName"
                        use="required"/>
                  <xsd:attribute name="type" type="xsd:NCName"
```

```
                        use="required"/>
                  <xsd:attribute name="default" type="xsd:NCName"
                        use="optional"/>
                  <xsd:attribute name="requires" type="sca:listOfQNames"
                        use="optional"/>
                  <xsd:attribute name="policySets" type="sca:listOfQNames"
                        use="optional"/>
                  <xsd:anyAttribute namespace="##other" processContents="lax"
                        use="optional"/>
            </xsd:complexType>
      </xsd:element>

      <xsd:element name="property">
            <xsd:complexType>
                  <any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:attribute name="name" type="xsd:NCName"
                        use="required"/>
                  <xsd:attribute name="type" type="xsd:NCName"
                        use="required"/>
                  <xsd:anyAttribute namespace="##other" processContents="lax"
                        use="optional"/>
            </xsd:complexType>
      </xsd:element>

      <xsd:element name="service">
            <xsd:complexType>
                  <any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:attribute name="name" type="xsd:NCName"
                        use="required"/>
                  <xsd:attribute name="type" type="xsd:NCName"
                        use="optional"/>
                  <xsd:attribute name="target" type="xsd:NCName"
                        use="required"/>
                  <xsd:attribute name="requires" type="sca:listOfQNames"
                        use="optional"/>
                  <xsd:attribute name="policySets" type="sca:listOfQNames"
                        use="optional"/>
                  <xsd:anyAttribute namespace="##other" processContents="lax"
                        use="optional"/>
            </xsd:complexType>
      </xsd:element>

</xsd:schema>
```

B. Conformance Items

Conformance ID	Description
[SPR20001]	The value of the @name attribute of an <sca:service/> subelement of a <beans/> element MUST be unique amongst the <service/> subelements of the <beans/> element.
[SPR20002]	The @target attribute of a <service/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element.
[SPR20003]	The value of the @name attribute of an <sca:reference/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <reference/> subelements, <property/> subelements and the <bean/> subelements of the <beans/> element.
[SPR20004]	The @default attribute of a <reference/> subelement of a <beans/> element MUST have the value of the @name attribute of one of the <bean/> subelements of the <beans/> element.
[SPR20005]	The value of the @name attribute of an <sca:property/> subelement of a <beans/> element MUST be unique amongst the @name attributes of the <property/> subelements, <reference/> subelements and the <bean/> subelements of the <beans/> element.
[SPR20006]	SCA extension elements within a Spring application context MUST conform to the SCA Spring Extension schema declared in sca-spring-extensions.xsd.
[SPR20007]	In multiple application context scenario, the SCA runtime MUST raise an error when multiple <sca:service> elements are identified with the same name amongst the set of application context files identified from the “Spring-Context” header in the MANIFEST.MF file.
[SPR20008]	When no <sca:service/> element is present in any of the application context file identified from the collection of application context paths identified by the "Spring-Context" header in the MANIFEST.MF file, then the SCA runtime MUST use implicit service determination only for the later bean definition.
[SPR20009]	In multiple application context scenario, the SCA runtime MUST determine the componentType by applying the rules defined in the section “Component Type of a Spring Application Context” to the combined application context and not to the individual application context files.
[SPR30001]	An SCA runtime MUST introspect the componentType of an implementation.spring application context following the rules defined in the section "Component Type of a Spring Application Context".
[SPR30002]	If this condition does not hold true then the SCA runtime MUST raise an error.
[SPR30003]	The introspected component type MUST be a compliant SCA Java XML Document as defined in section 12.1 of [JAVA-CAA].
[SPR30004]	The SCA runtime MUST support the SCA annotations which are applicable to an interface class which is referenced by an <interface.java/> element in the introspected component type of a Spring application context.

[SPR30005]	where there are no explicit <sca:service/> elements in the application context, if a bean class implements two or more Java interfaces which are annotated with @Remotable, this is not allowed and the SCA runtime MUST raise an error.
[SPR40001]	The implementation.spring @location attribute URI value MUST point to one of the following: a) a Spring application context file b) a Java archive file (JAR) c) a directory
[SPR40002]	If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context.
[SPR40003]	If the implementation.spring @location URI identifies a JAR archive file, then the file META-INF/MANIFEST.MF MUST be read from the archive.
[SPR40004]	If the implementation.spring @location URI identifies a directory, then the file META-INF/MANIFEST.MF underneath that directory MUST be read from the directory.
[SPR40005]	If present, all the Spring application context configuration files identified by the "Spring-Context" header in the MANIFEST.MF file MUST be collectively used to build the Spring application context for implementation.spring element.
[SPR40006]	If there is no MANIFEST.MF file or if there is no Spring-Context header within the MANIFEST.MF file, the Spring application context MUST be built using all the *.xml files in the META-INF/spring subdirectory within the JAR identified by the @location URI or underneath the directory specified by the @location URI.
[SPR40007]	The <implementation.spring> element MUST conform to the schema defined in sca-implementation-spring.xsd.
[SPR40008]	If the MANIFEST.MF file contains a header "Spring-Context" of the format: Spring-Context ::= path (';' path)* where path is a relative path with respect to the @location URI, then each path specified in the header MUST identify a Spring application context configuration file.
[SPR40009]	The location URI can either be an absolute URI or it can be a relative URI. In the case where the location URI is a relative URI, the URI MUST be taken as being relative to the base of the contribution which contains the composite containing the <implementation.spring/> element.

817

C. 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*
Mirza Begg	Individual
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
Derek Dougans	Individual
Mike Edwards	IBM
Ant Elder	IBM
Raymond Feng	IBM
James Hao	Primeton Technologies, Inc.
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Khanderao Kand	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
Scott Vorthmann	TIBCO Software Inc.
Feng Wang	Primeton Technologies, Inc. Changfeng Open Standards
Paul Yang	Platform Software
Yuan Yi	Primeton Technologies, Inc.

823

D. Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
1	2007-09-26	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
WD01	2008-11-24	Mike Edwards	Editorial cleanup Issue 64 resolution applied Issue 57 resolution applied
WD02	2009-07-20	Mike Edwards	Issue 164 resolution applied Added Appendix B - Conformance Items Issue 58 resolution applied (new Section 3) Issue 92 resolution applied - Section 3 Issue 59 resolution applied - Section 3
WD02 + Issue106	2009-08-06	Mike Edwards	Issue 106 (RFC2119) - added Section 4 - added Appendix A1 - added Appendix B
WD03	2009-08-07	Mike Edwards	All changes accepted.
WD04	2009-08-14	Mike Edwards	Issue 63 applied - Section 2 All changes accepted
WD05	2010-08-06	Anish Karmarkar	Issue 63 fully applied (few changes from the resolution were missing) Issue 149 resolution applied. Issue 166 resolution applied. Issue 167 resolution applied. Issue 173 & 175 resolution applied. Issue 150 resolution applied.
WD06	2011-02-23	Anish Karmarkar	Issue 182 resolution applied Issue 225 resolution applied Issue 229 resolution applied Ed fixes
WD07	2011-03-29	Mike Edwards	Issue 230 resolution applied Issue 237 resolution applied Formatting updates & editorial fixes. Added Acknowledgements entries

WD08	2011-05-02	Anish Karmarkar	Issue 228 resolution applied Added a ref to testcases doc and a new section that talks about it

827
828
829