



# Service Component Architecture Spring Component Implementation Specification Version 1.1

**Working Draft 054**

**0814 August 201008**

**Specification URIs:**

**This Version:**

- <http://docs.oasis-open.org/sca-j/sca-springci-1.1-spec-WD054.html>
- <http://docs.oasis-open.org/sca-j/sca-springci-1.1-spec-WD054.doc>
- <http://docs.oasis-open.org/sca-j/sca-springci-1.1-spec-WD054.pdf>

**Previous Version:**

**Latest Version:**

- <http://docs.oasis-open.org/sca-j/sca-springci-draft-20070926.html>
- <http://docs.oasis-open.org/sca-j/sca-springci-draft-20070926.doc>
- <http://docs.oasis-open.org/sca-j/sca-springci-draft-20070926.pdf>

**Latest Approved Version:**

**Technical Committee:**

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

**Chair(s):**

Dave Booz, IBM  
Mark Combellack, Avaya

**Editor(s):**

David Booz, IBM  
Mike Edwards, IBM  
Anish Karmarkar, Oracle  
Ashok Malhotra, Oracle  
Peter Peshev, SAP

**Related work:**

This specification replaces or supercedes:

- 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
- Service Component Architecture Policy Framework Specification Version 1.1
- [Service Component Architecture Java Common Annotations and APIs Specification Version 1.1](#)

**Declared XML Namespace(s):**

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

sca-springci-1.1-spec-WD054

Copyright © OASIS® 2007, 201008. All Rights Reserved.

0814 August 201009

Page 1 of 29

Field Code Changed

Field Code Changed

Field Code Changed

**Abstract:**

The SCA Spring component implementation specification specifies the how the Spring Framework can be used with SCA. The goals of this effort 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:** It should be possible to use Spring to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

**Start from Spring context:** It should be possible to generate an SCA Composite from any Spring application context and use that composite 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" 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/>.

---

## Notices

Copyright © OASIS® 2007, 20~~1009~~. All Rights Reserved.

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", [insert specific trademarked names and abbreviations here] are 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.

# Table of Contents

1	Introduction	6
1.1	Terminology	6
1.2	Normative References	6
1.3	Non-Normative References	6
2	Spring application context as component implementation	7
2.1	Structure of a Spring Application Context	8
2.1.1	Spring Beans	9
2.1.2	Property and Constructor Argument References	9
2.2	Direct use of SCA references within a Spring configuration	10
2.3	Explicit declaration of SCA related beans inside a Spring Application Context	11
2.3.1	SCA Service element	11
2.3.2	SCA Reference element	12
2.3.3	SCA Property element	13
2.3.4	Example of a Spring Application Context with SCA Spring Extension Elements	14
2.4	Handling multiple application contexts in SCA runtime	14
3	Component Type of a Spring Application Context	16
3.1	Introspecting the Type Implied by a Spring Bean Reference	18
4	Specifying the Spring Implementation Type in an Assembly	20
5	Conformance	22
5.1	SCA Spring Component Implementation Composite Document	22
5.2	SCA Spring Application Context Document	22
5.3	SCA Runtime	22
A.	XML Schemas	23
A.1	sca-implementation-spring.xsd	23
A.2	SCA Spring Extension schema - sca-spring-extension.xsd	23
B.	Conformance Items	25
C.	Acknowledgements	27
D.	Non-Normative Text	28
E.	Revision History	29
1	Introduction	5
1.1	Terminology	5
1.2	Normative References	5
1.3	Non-Normative References	5
2	Spring application context as component implementation	6
2.1	Structure of a Spring Application Context	7
2.1.1	Spring Beans	8
2.1.2	Property and Constructor Argument References	8
2.2	Direct use of SCA references within a Spring configuration	9
2.3	Explicit declaration of SCA related beans inside a Spring Application Context	10
2.3.1	SCA Service element	10
2.3.2	SCA Reference element	11
2.3.3	SCA Property element	12
2.3.4	Example of a Spring Application Context with SCA Spring Extension Elements	12

3	Component Type of a Spring Application Context .....	14
4	Specifying the Spring Implementation Type in an Assembly .....	17
5	Conformance .....	18
5.1	SCA Spring Component Implementation Composite Document .....	18
5.2	SCA Spring Application Context Document .....	18
5.3	SCA Runtime .....	18
A.	XML Schemas .....	19
A.1	sca-implementation-spring.xsd .....	19
A.2	SCA Spring Extension schema — sca-spring-extension.xsd .....	19
B.	Conformance Items .....	21
C.	Acknowledgements .....	23
D.	Non Normative Text .....	24
E.	Revision History .....	25

---

# 1 Introduction

The SCA Java Client and Implementation model for Spring specifies the how the Spring Framework can be used with SCA. The goals of this effort 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:** It is possible to use Spring to implement any SCA Component that uses WSDL or Java interfaces to define services, possibly with some SCA specific extensions.

**Start from Spring context:** It is possible to generate 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*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [SCA-ASSEMBLY] SCA Assembly Model Specification V1.1  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec.pdf>
- [SCA-POLICY] SCA Policy Framework Specification V1.1  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf>
- [JAVA-CAA] [SCA SCA-J Common Annotations and APIs Specification Version 1.1](http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf)  
<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf>
- [SPRING] Spring Framework Specification  
<http://static.springsource.org/spring/docs/2.5.x/reference/index.html>

## 1.3 Non-Normative References

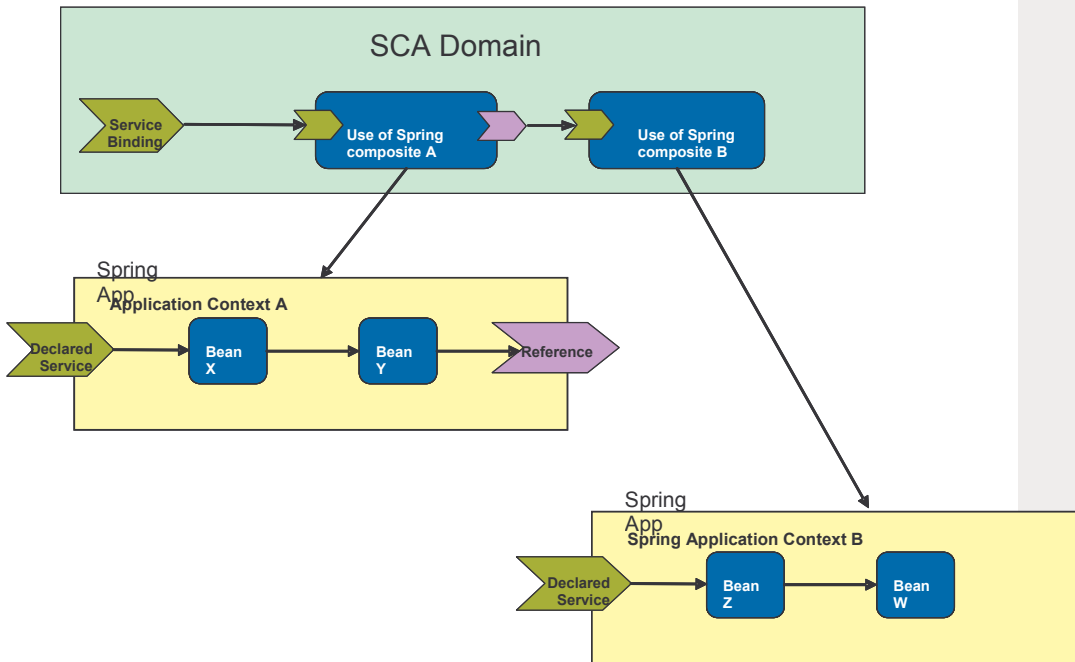
- TBD TBD

30  
31  
32  
33  
34  
35  
36

## 2 Spring application context as component implementation

A Spring Application Context can be used as an implementation within an SCA component. Conceptually, this can be represented as follows:

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

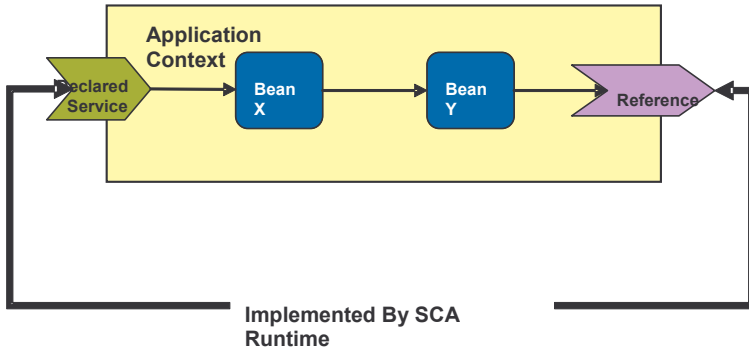


37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48

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 second component which is also implemented by another Spring context, Context B, which has a single declared service, which is wired to 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 knows very little about the SCA environment. All policy enforcement occurs in the SCA runtime implementation and does not enter into the Spring space.



49  
50 *Figure 2*

51 Figure 2 shows two of the points where the SCA runtime interacts with the Spring context:  
52 services and references. Any policy enforcement is done by the SCA runtime on calls into the  
53 Spring application context before the final message is delivered to the target Spring bean.  
54 On outbound calls from the application context, references supplied by the SCA can provide policy  
55 enforcement

## 56 2.1 Structure of a Spring Application Context

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

```
60 <beans>
61   <bean id="xs:string" name="xs:string" class="xs:string"
62     scope="xs:string">*
63     <property name="xs:string" value="xs:string"? ref="xs:string"?>*
64       <value type="xs:string"?/>?
65       <bean/>?
66       <ref bean="xs:string"? local="xs:IDREF"? parent="xs:string"?/>?
67       <idref bean="xs:string" local="xs:IDREF"?/>?
68     </property>
69     <list/>?
70     <map/>?
71     <set/>?
72     <lookup-method/>?
73     <replaced-method/>?
74   </bean>
75   <constructor-arg ref="xs:string"? index="xs:string"
76     type="xs:string"? value="xs:string"?>*
77     <value/>?
78     <bean/>?
79     <ref bean="xs:string"/>?
80     <idref bean="xs:string"/>?
81     </constructor-arg>
82     <list/>?
83     <map/>?
84     <set/>?
85     <props/>?
86   </constructor-arg>
87   <meta/>*
88   <qualifier/>*
89   <lookup-method/>*
90   <replaced-method/>*
91   </beans>
```



90 </bean>

91 </beans>

92 Example 1: Pseudo-schema for the Spring Application Context

### 93 2.1.1 Spring Beans

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

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

#### 98 2.1.1.1 Bean ID & Name

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

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

Formatted: Font: Not Bold, Not Italic

#### 106 2.1.1.2 Inner Beans

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

#### 110 2.1.1.3 Bean Properties

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

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

#### 117 2.1.1.4 Bean Constructor Arguments

118 A <bean/> can have zero or more <constructor-arg/> subelements. These elements are very similar to  
119 | <property/> elements in that they represent a dependency of the bean class, which ~~must have to~~ be  
120 injected into the class when it is instantiated. The difference between <constructor-arg/> elements and  
121 <property/> elements is that <constructor-arg/> values are injected into the class through parameters on  
122 the bean class constructor method, rather than through setter methods.

### 123 2.1.2 Property and Constructor Argument References

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

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

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

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

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

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

## 143 2.2 Direct use of SCA references within a Spring configuration

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

148 The following Spring configuration provides a model for Spring application context A, expressed in  
149 figure 1 above. In this example, there are two Spring beans, X and Y. The bean named "X" is the  
150 entry point from SCA into the Spring context and Spring bean Y contains a reference to a service  
151 supplied by SCA.

```
152 <beans>
153     <bean id="X" class="org.xyz.someapp.SomeClass">
154         <property name="foo" ref="Y"/>
155     </bean>
156     <bean id="Y" class="org.xyz.someapp.SomeOtherClass">
157         <property name="bar" ref="SCAReference"/>
158     </bean>
159 </beans>
```

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

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

```
165 <composite name="BazComposite">
166     <component name="SpringComponent">
167         <implementation.spring location=".."/>
168         <service name="X"/>
169         <reference name="SCAReference" ../> <!-- binding info specified -->
170     </component>
171 </composite>
```

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

177 Each <service> element used with <implementation.spring> by default includes the name of  
178 the Spring bean that is to be exposed as an SCA service in its name attribute. So, for Spring, the

179 name attribute of a service plays two roles: it identifies a Spring bean, and it names the service for the  
180 component. The service element above has a name of "X", so there is a Spring bean with that name.  
181 The SCA component also contains a <reference> element named "SCAReference". The reference  
182 name becomes an addressable name within the Spring application context – so, in this case,  
183 "SCAReference" can be referred to by bean "Y" in the Spring configuration above.

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

## 189 2.3 Explicit declaration of SCA related beans inside a Spring 190 Application Context

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

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

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

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

202 The SCA Spring extension elements are:

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

### 209 2.3.1 SCA Service element

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

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

219 The pseudo-schema for the service element is:

```
220 <beans xmlns="http://www.springframework.org/schema/beans"  
221       xmlns:xs="http://www.w3.org/2001/XMLSchema"  
222       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
223       xmlns:sca=  
224           "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"  
225       ...  
226       <sca:service name="xs:NCName"
```

```

228 |         type="xs:NCName"?
229 |         target="xs:NCName"
230 |         requires="list of xs:QName"?
231 |         policySets="list of xs:QName"?/>
232 |     ...
233 |
234 | </beans>

```

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

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

## 252 2.3.2 SCA Reference element

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

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

262 The pseudo-schema for the reference element is:

```

263 | <beans xmlns="http://www.springframework.org/schema/beans"
264 |       xmlns:xs="http://www.w3.org/2001/XMLSchema"
265 |       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
266 |       xmlns:sca=
267 |         "http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
268 |
269 |     ...
270 |     <sca:reference name="xs:NCName"
271 |                 type="xs:NCName"
272 |                 default="xs:NCName"?
273 |                 requires="list of xs:QName"?
274 |                 policySets="list of xs:QName"?/>
275 |     ...
276 | </beans>
277 |

```

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

- 279 • **name : NCName (1..1)** - the name of the reference. The value of the @name attribute  
280 of an <sca:reference/> subelement of a <beans/> element MUST be unique amongst the  
281 @name attributes of the <reference/> subelements, <property/> subelements and the  
282 <bean/> subelements of the <beans/> element. The value of the @name attribute of an  
283 <sca:reference/> subelement of a <beans/> element MUST be unique amongst the  
284 @name attributes of the <reference/> subelements, <property/> subelements and the  
285 <bean/> subelements of the <beans/> element. [SPR20003]
- 286 • **type : NCName (1..1)** - the type of the reference, declared as the fully qualified name of  
287 a Java class.
- 288 • **default : NCName (0..1)** - the name of a <bean/> element within the application  
289 context which provides the reference declared by the sca:reference element if the  
290 component using the application context as an implementation does not wire the reference  
291 to a target service. The @default attribute of a <reference/> subelement of a <beans/>  
292 element MUST have the value of the @name attribute of one of the <bean/> subelements  
293 of the <beans/> element. [SPR20004]
- 294 • **requires : QName (0..1)** - a list of policy intents. See the [Policy Framework specification](#)  
295 [\[POLICY\]](#) for a description of this attribute.
- 296 • **policySets : QName (0..1)** - a list of policy sets. See the [Policy Framework specification](#)  
297 [\[POLICY\]](#) for a description of this attribute.

### 299 2.3.3 SCA Property element

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

307 The pseudo-schema for the property element is:

```
308 <beans xmlns="http://www.springframework.org/schema/beans"
309       xmlns:xs="http://www.w3.org/2001/XMLSchema"
310       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
311       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca-
312 j/spring/200810"
313
314   ...
315   <sca:property name="xs:NCName"
316               type="xs:NCName"/>
317   ...
318 </beans>
```

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

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

330

### 331 2.3.4 Example of a Spring Application Context with SCA Spring Extension 332 Elements

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

```
336 <beans>
337
338     <!-- An explicit reference, which is used by bean "Y" -->
339     <sca:reference name="SCAResource" type="com.xyz.SomeType"/>
340
341     <bean name="X">
342         <property name="foo" ref="Y"/>
343     </bean>
344
345     <bean name="Y">
346         <property name="bar" ref="SCAResource"/>
347         <property name="goo" ref="sca-property-name"/>
348     </bean>
349
350     <!-- expose an SCA property named "sca-property-name" -->
351     <sca:property name="sca-property-name" type="java.lang.String"/>
352
353     <!-- Expose the bean "X" as an SCA service named "SCAService" -->
354     <sca:service name="SCAService" type="org.xyz.someapp.SomeInterface"
355         target="X"/>
356
357 </beans>
```

## 358 2.4 Handling multiple application contexts in SCA runtime

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

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

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

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

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

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

```
377
378 <beans>
379     <bean name="bean1" class="....." />
380     <bean name="bean2" class="....." />
381     <bean
382         class="org.springframework.context.support.ClassPathXmlApplicationContext">
```

Formatted: Heading 2,H2

Formatted: Font: (Default) Courier New

Formatted: Font: (Default) Courier New

Formatted: Font: (Default) Courier New

```
383 <constructor-arg>
384 <list>
385 <value>context1.xml</value>
386 <value>context2.xml</value>
387 <value>context3.xml</value>
388 </list>
389 </constructor-arg>
390 </bean>
391 </beans>
```

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

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

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

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

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

Formatted: Highlight

Formatted: Font color: Red

Formatted: Highlight

Formatted: Font color: Red

Formatted: Highlight

Formatted: Font color: Red

415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456

### 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 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 then services are defined by each of the top-level <bean/> elements in the application context:

- <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:

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, found by introspection of the bean class declared by the bean element, where:

- @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

Formatted: Font: Bold, Italic

Formatted: Bulleted + Level: 1 + Aligned at: 0.25" + Indent at: 0.5"



- 457
- 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
- 458
- binding child element is omitted
- 459
- callback child element is omitted
- 460
- 461

462

463 Note that as described in the SCA Assembly Model specification **[SCA-ASSEMBLY]** the @name attribute

464 has to be unique amongst all <service/> elements in the componentType.

465 Where a Spring Bean implementation class implements more than one interface, the Bean can be

466 exposed as either a single service or as multiple services, through the use of explicit <sca:service/>

467 elements, where each <sca:service/> element references the same <bean/> element but where the

468 @type attribute uses only one of the interfaces provided by the bean.

469 Where there are no <sca:service/> elements, the bean is exposed as a single service with an interface

470 that is the defined by the bean class itself.

471 Note that a <bean/> element nested within another <bean/> element (an inner bean) is never exposed

472 directly as an SCA service.

473

474 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
- 475
- 476
- 477
- 478
- 479
- 480
- 481
- 482
- 483
- 484
- 485
- 486
- 487
- 488
- 489
- 490

491

492 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"
- 493
- 494
- 495
- 496
- 497
- 498
- 499

500

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

504

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

- 507 • @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/>  
508 element that makes the reference, or the reference name derived from the subelements of the  
509 <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- 510 • @autowire attribute is omitted
- 511 • @wiredByImple attribute is omitted
- 512 • @target attribute is omitted
- 513 • @multiplicity attribute is set to (1..1)
- 514 • @requires attribute is omitted
- 515 • @policySets attribute is omitted
- 516 • interface.java child element is present, with the interface attribute set to the fully qualified name of  
517 the interface class identified by the bean reference
- 518 • binding child element is omitted
- 519 • callback child element is omitted

520

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

- 524 • @name attribute is the value of the @ref attribute of the <property/> or <constructor-arg/>  
525 element that makes the reference, or the reference name derived from the subelements of the  
526 <property/> or <constructor-arg/> element (eg. @bean attribute of a <ref/> subelement)
- 527 • @value attribute is omitted
- 528 • @type attribute is set to the XML type implied by the JAXB mapping of the Java class identified  
529 by the bean reference
- 530 • @element attribute is omitted
- 531 • @many attribute is set to "false"
- 532 • @mustSupply attribute is set to "true"

533

534 The Spring Component Implementation type does not support the use of Component Type side files, as  
535 defined in the SCA Assembly Model specification [**SCA-ASSEMBLY**], so that the effective  
536 componentType of a Spring Application Context is determined completely by introspection of the Spring  
537 Application Context.

538 [It is beyond the scope of this specification to define the interpretation of the annotations specified in the](#)  
539 [SCA Common Annotations and API Specification \[JAVA-CAA\]. An implementation can ignore SCA](#)  
540 [annotations that are present in classes used by the application context.](#)

### 541 **3.1 Introspecting the Type Implied by a Spring Bean Reference**

542 [In the case where a reference or a property in the component type is derived by introspection of bean](#)  
543 [references, the type of the reference or property is determined by introspection of the related property](#)  
544 [setter method or constructor method of the Bean which is the source of the reference.](#)

Formatted: Heading 2,H2

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

548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598

## 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 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

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. 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.

599 The <implementation.spring> element MUST conform to the schema defined in sca-implementation-  
600 spring.xsd. [SPR40007]  
601

---

## 602 5 Conformance

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

606  
607 There are three categories of artifacts that this specification defines conformance for: SCA Spring  
608 Component Implementation Composite Document, SCA Spring Application Context Document and SCA  
609 Runtime.

### 610 5.1 SCA Spring Component Implementation Composite Document

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

### 616 5.2 SCA Spring Application Context Document

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

### 622 5.3 SCA Runtime

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

- 624 1. The implementation MUST meet all the conformance requirements defined by the SCA  
625 Assembly Model Specification [ASSEMBLY].
- 626 2. The implementation MUST reject an SCA Spring Component Implementation Composite  
627 Document that does not conform to the sca-implementation-spring.xsd schema.
- 628 3. The implementation MUST reject an SCA Spring Application Context Document that does not  
629 conform to the sca-spring-extension.xsd schema.
- 630 4. The implementation MUST comply with all statements related to an SCA Runtime, specified in  
631 'Appendix B: Conformance Items' of this specification, notably all mandatory statements have  
632 to be implemented.
- 633
- 634

635

## A. XML Schemas

636

### A.1 sca-implementation-spring.xsd

```

637 <?xml version="1.0" encoding="UTF-8"?>
638 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
639 OASIS trademark, IPR and other policies apply. -->
640 <schema xmlns="http://www.w3.org/2001/XMLSchema"
641 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
642 elementFormDefault="qualified"
643 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903">
644
645 <include schemaLocation="sca-core-1.1-cd03.xsd"/>
646 <element name="implementation.spring" type="sca:SpringImplementation"
647 substitutionGroup="sca:implementation"/>
648 <complexType name="SpringImplementation">
649 <complexContent>
650 <extension base="sca:Implementation">
651 <sequence>
652 <any namespace="##other" processContents="lax" minOccurs="0"
653 maxOccurs="unbounded"/>
654 </sequence>
655 <attribute name="location" type="anyURI" use="required"/>
656 </extension>
657 </complexContent>
658 </complexType>
659 </schema>
660

```

661

### A.2 SCA Spring Extension schema - sca-spring-extension.xsd

662

```

663 <?xml version="1.0" encoding="UTF-8"?>
664 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
665 OASIS trademark, IPR and other policies apply. -->
666 <xsd:schema
667 xmlns="http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810"
668 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
669 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
670 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
671 xsi:schemaLocation="
672 http://docs.oasis-open.org/ns/opencsa/sca/200903
673 http://docs.oasis-open.org/opencsa/sca-assembly/sca-core-1.1-cd03.xsd"
674 attributeFormDefault="unqualified"
675 elementFormDefault="qualified"
676 targetNamespace="
677 http://docs.oasis-open.org/ns/opencsa/sca-j/spring/200810">
678
679 <xsd:element name="reference">
680 <xsd:complexType>
681 <any namespace="##other" processContents="lax"
682 minOccurs="0" maxOccurs="unbounded"/>
683 <xsd:attribute name="name" type="xsd:NCName"
684 use="required"/>
685 <xsd:attribute name="type" type="xsd:NCName"

```

```

686         use="required"/>
687     <xsd:attribute name="default" type="xsd:NCName"
688         use="optional"/>
689     <xsd:attribute name="requires" type="sca:listOfQNames"
690         use="optional"/>
691     <xsd:attribute name="policySets" type="sca:listOfQNames"
692         use="optional"/>
693     <xsd:anyAttribute namespace="##other" processContents="lax"
694         use="optional"/>
695 </xsd:complexType>
696 </xsd:element>
697
698 <xsd:element name="property">
699     <xsd:complexType>
700         <any namespace="##other" processContents="lax"
701             minOccurs="0" maxOccurs="unbounded"/>
702         <xsd:attribute name="name" type="xsd:NCName"
703             use="required"/>
704         <xsd:attribute name="type" type="xsd:NCName"
705             use="required"/>
706         <xsd:anyAttribute namespace="##other" processContents="lax"
707             use="optional"/>
708     </xsd:complexType>
709 </xsd:element>
710
711 <xsd:element name="service">
712     <xsd:complexType>
713         <any namespace="##other" processContents="lax"
714             minOccurs="0" maxOccurs="unbounded"/>
715         <xsd:attribute name="name" type="xsd:NCName"
716             use="required"/>
717         <xsd:attribute name="type" type="xsd:NCName"
718             use="required|optional"/>
719         <xsd:attribute name="target" type="xsd:NCName"
720             use="required"/>
721         <xsd:attribute name="requires" type="sca:listOfQNames"
722             use="optional"/>
723         <xsd:attribute name="policySets" type="sca:listOfQNames"
724             use="optional"/>
725         <xsd:anyAttribute namespace="##other" processContents="lax"
726             use="optional"/>
727     </xsd:complexType>
728 </xsd:element>
729
730 </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][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][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-extension.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".
[SPR40001][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][SPR40002]	If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context.

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Highlight

Formatted: Font color: Red

Formatted: Highlight

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Highlight

[SPR40003][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][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][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][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][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 ) <sup>*</sup> 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.

733

---

## C. Acknowledgements

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

736 **Participants:**

737 [Participant Name, Affiliation | Individual Member]

738 [Participant Name, Affiliation | Individual Member]

739

---

## D. Non-Normative Text

741

## E. Revision History

742 [optional; should not be included in OASIS Standards]

743

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
<a href="#">WD05</a>	<a href="#">2010-08-06</a>	<a href="#">Anish Karmarkar</a>	<a href="#">Issue 63 fully applied (few changes from the resolution were missing)</a> <a href="#">Issue 149 resolution applied.</a> <a href="#">Issue 166 resolution applied.</a> <a href="#">Issue 167 resolution applied.</a> <a href="#">Issue 173 &amp; 175 resolution applied.</a> <a href="#">Issue 150 resolution applied.</a>

744

745