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/irp.php).

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

Copyright © OASIS® 2007, 2010. 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  
    2.4 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 ................................................................................................. 29  
E  Revision History ...................................................................................................... 29  
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 .................................... 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  
    2.4 Example of a Spring Application Context with SCA Spring Extension Elements ... 12
<table>
<thead>
<tr>
<th>Component Type of a Spring Application Context</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specifying the Spring Implementation Type in an Assembly</td>
<td>17</td>
</tr>
<tr>
<td>Conformance</td>
<td>18</td>
</tr>
<tr>
<td>5.1 SCA Spring Component Implementation Composite Document</td>
<td>18</td>
</tr>
<tr>
<td>5.2 SCA Spring Application Context Document</td>
<td>18</td>
</tr>
<tr>
<td>5.3 SCA Runtime</td>
<td>18</td>
</tr>
<tr>
<td>A. XML Schemas</td>
<td>19</td>
</tr>
<tr>
<td>A.1 sca-implementation-spring.xsd</td>
<td>19</td>
</tr>
<tr>
<td>A.2 SCA Spring Extension schema – sca-spring-extension.xsd</td>
<td>19</td>
</tr>
<tr>
<td>B. Conformance Items</td>
<td>21</td>
</tr>
<tr>
<td>C. Acknowledgements</td>
<td>23</td>
</tr>
<tr>
<td>D. Non-Normative Text</td>
<td>24</td>
</tr>
<tr>
<td>E. Revision History</td>
<td>25</td>
</tr>
</tbody>
</table>
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


1.3 Non-Normative References

TBD TBD
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.

![SCA Domain Diagram]

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.
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.
2.1 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 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 **must** 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.

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.

2.1.2 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.

2.1.3 Bean Properties

A <bean/> can have zero or more <property/> subelements. Each <property/> represents a dependency of the bean class, which **must** 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 **required** 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/>.

2.1.4 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 **must** 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" - i.e. not pointing to any bean within the application context. Alternatively, SCA references and SCA properties can be explicitly 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 composite 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 bean named "X" is the entry point from SCA into the Spring context and Spring bean Y contains a reference to a service supplied by SCA.

```xml
<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="SCAReturnReference"/>
  </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, given 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:

```xml
<composite name="BazComposite">
  <component name="SpringComponent">
    <implementation.spring location=".."/>
    <service name="x"/>
    <reference name="SCAReference" ...
  </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
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 - ssa-spring-extension.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 ssa-spring-extension.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:

```xml
<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:

```xml
<sca:service xmlns="http://docs.oasis-open.org/ns/opensca/ssa-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. 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:

```xml
<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema"

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 `<bean/>` element MUST be unique amongst the @name attributes of the `<reference/>` subelements, `<property/>` subelements and the `<bean/>` subelements of the `<bean/>` element. The value of the @name attribute of an `< sca: reference/>` subelement of a `<bean/>` element MUST be unique amongst the @name attributes of the `<reference/>` subelements, `<property/>` subelements and the `<bean/>` subelements of the `<bean/>` element. [SPR20003]

• **type : NCName (0..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 reference 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:

```
<bean
  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"/>
  ...
</bean>
```

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 `<bean/>` element MUST be unique amongst the @name attributes of the `<property/>` subelements, `<reference/>` subelements and the `<bean/>` subelements of the `<bean/>` element. The value of the @name attribute of an `<sca:property/>` subelement of a `<bean/>` element MUST be unique amongst the @name attributes of the `<property/>` subelements, `<reference/>` subelements and the `<bean/>` subelements of the `<bean/>` 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 "goo" property of bean Y is configured with an SCA property with name "sca-property-name".

```xml
<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="goo" 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>
```

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.

```xml
<beans>
  <bean name="bean1" class="...">
  <bean name="bean2" class="...">
  <bean
    class="org.springframework.context.support.ClassPathXmlApplicationContext">
```
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]

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 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:

- @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 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
• 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

• 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 by the bean class itself.

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 (e.g. `@bean` attribute of a `<ref/>` subelement)
- `@autowire` attribute is omitted
- `@wiredBy` 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 to the fully qualified name of the interface class identified by the bean reference
- 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 (e.g. `@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]. 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.
- **requires** – a list of policy intents. See the Policy Framework specification [POLICY] for a description of this attribute.
- **policySets** – a list of policy sets. See the Policy Framework specification [POLICY] for a description of this attribute.
The `<implementation.spring>` element MUST conform to the schema defined in `sca-implementation-spring.xsd`. [SPR40007]
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 Framework 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-extension.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
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:sca="http://docs.oasis-open.org/ns/opensca/sca-200903"
 xmlns:xs="http://www.w3.org/2000/XMLSchema"
 targetNamespace="http://docs.oasis-open.org/ns/opensca/sca-200903">
<element name="SpringImplementation">
<complexType>
<extension base="sca:Implementation">
<sequence>
<attribute name="location" type="anyURI" use="required"/>
</sequence>
</complexType>
</sca:SpringImplementation>
</xs:schema>
```

A.2 SCA Spring Extension schema

- sca-spring-extension.xsd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
 xmlns="http://docs.oasis-open.org/ns/opensca/sca-j/spring/200810"
 xmlns:sca="http://docs.oasis-open.org/opensca/sca-assembly/sca-core-1.1-cd03.xsd"
 targetNamespace="http://docs.oasis-open.org/ns/opensca/sca-j/spring/200810">
<xs:element name="reference">
<complexType>
<sequence>
<attribute name="name" type="xs:NCName" use="required"/>
</sequence>
</complexType>
</xs:schema>
```
<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:attribute name="policySets" type="sca:listOfQNames"
      use="optional"/>
    <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="required"/>
    <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:complexType>
</xsd:element>
</xsd:element>
## B. Conformance Items

<table>
<thead>
<tr>
<th>Conformance ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SPR20001]</td>
<td>The value of the @name attribute of an <code>&lt;sca:service&gt;</code> subelement of a <code>&lt;beans&gt;</code> element MUST be unique amongst the <code>&lt;service&gt;</code> subelements of the <code>&lt;beans&gt;</code> element.</td>
</tr>
<tr>
<td>[SPR20002]</td>
<td>The @target attribute of a <code>&lt;service&gt;</code> subelement of a <code>&lt;beans&gt;</code> element MUST have the value of the @name attribute of one of the <code>&lt;bean&gt;</code> subelements of the <code>&lt;beans&gt;</code> element.</td>
</tr>
<tr>
<td>[SPR20003]</td>
<td>The value of the @name attribute of an <code>&lt;sca:reference&gt;</code> subelement of a <code>&lt;beans&gt;</code> element MUST be unique amongst the @name attributes of the <code>&lt;reference&gt;</code> subelements, <code>&lt;property&gt;</code> subelements and the <code>&lt;bean&gt;</code> subelements of the <code>&lt;beans&gt;</code> element.</td>
</tr>
<tr>
<td>[SPR20004]</td>
<td>The @default attribute of a <code>&lt;reference&gt;</code> subelement of a <code>&lt;beans&gt;</code> element MUST have the value of the @name attribute of one of the <code>&lt;bean&gt;</code> subelements of the <code>&lt;beans&gt;</code> element.</td>
</tr>
<tr>
<td>[SPR20005]</td>
<td>The value of the @name attribute of an <code>&lt;sca:property&gt;</code> subelement of a <code>&lt;beans&gt;</code> element MUST be unique amongst the @name attributes of the <code>&lt;property&gt;</code> subelements, <code>&lt;reference&gt;</code> subelements and the <code>&lt;bean&gt;</code> subelements of the <code>&lt;beans&gt;</code> element.</td>
</tr>
<tr>
<td>[SPR20006]</td>
<td>SCA extension elements within a Spring application context MUST conform to the SCA Spring Extension schema declared in <code>sca-spring-extension.xsd</code>.</td>
</tr>
<tr>
<td>[SPR20007]</td>
<td>In multiple application context scenario, the SCA runtime MUST raise an error when multiple <code>&lt;sca:service&gt;</code> elements are identified with the same name amongst the set of application context files identified from the &quot;Spring-Context&quot; header in the MANIFEST.MF file.</td>
</tr>
<tr>
<td>[SPR20008]</td>
<td>When no <code>&lt;sca:service&gt;</code> element is present in any of the application context file identified from the collection of application context paths identified by the &quot;Spring-Context&quot; header in the MANIFEST.MF file, then the SCA runtime MUST use implicit service determination only for the later bean definition.</td>
</tr>
<tr>
<td>[SPR20009]</td>
<td>In multiple application context scenario, the SCA runtime MUST determine the component type by applying the rules defined in the section &quot;Component Type of a Spring Application Context&quot; to the combined application context and not to the individual application context files.</td>
</tr>
<tr>
<td>[SPR30001]</td>
<td>An SCA runtime MUST introspect the componentType of an implementation.spring application context following the rules defined in the section &quot;Component Type of a Spring Application Context&quot;.</td>
</tr>
<tr>
<td>[SPR40001]</td>
<td>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</td>
</tr>
<tr>
<td>[SPR40002]</td>
<td>If the implementation.spring @location URI identifies a Spring application context file, it MUST be used as the Spring application context.</td>
</tr>
<tr>
<td>SPR40003</td>
<td>SPR40003</td>
</tr>
<tr>
<td>SPR40004</td>
<td>SPR40004</td>
</tr>
<tr>
<td>SPR40005</td>
<td>SPR40005</td>
</tr>
<tr>
<td>SPR40006</td>
<td>SPR40006</td>
</tr>
<tr>
<td>SPR40007</td>
<td>SPR40007</td>
</tr>
<tr>
<td>SPR40008</td>
<td>SPR40008</td>
</tr>
</tbody>
</table>
C. Acknowledgements

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

Participants:
[Participant Name, Affiliation | Individual Member]
[Participant Name, Affiliation | Individual Member]
D. Non-Normative Text
### E. Revision History

[optional; should not be included in OASIS Standards]

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Editor</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2007-09-26</td>
<td>Anish Karmarkar</td>
<td>Applied the OASIS template + related changes to the Submission</td>
</tr>
<tr>
<td>WD01</td>
<td>2008-11-24</td>
<td>Mike Edwards</td>
<td>Editorial cleanup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 64 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 57 resolution applied</td>
</tr>
<tr>
<td>WD02</td>
<td>2009-07-20</td>
<td>Mike Edwards</td>
<td>Issue 164 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added Appendix B - Conformance Items</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 58 resolution applied (new Section 3)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 92 resolution applied - Section 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 59 resolution applied - Section 3</td>
</tr>
<tr>
<td>WD02 + Issue106</td>
<td>2009-08-06</td>
<td>Mike Edwards</td>
<td>Issue 106 (RFC2119)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- added Section 4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- added Appendix A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- added Appendix B</td>
</tr>
<tr>
<td>WD03</td>
<td>2009-08-07</td>
<td>Mike Edwards</td>
<td>All changes accepted</td>
</tr>
<tr>
<td>WD04</td>
<td>2009-08-14</td>
<td>Mike Edwards</td>
<td>Issue 63 applied - Section 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>All changes accepted</td>
</tr>
<tr>
<td>WD05</td>
<td>2010-08-06</td>
<td>Anish Karmarkar</td>
<td>Issue 63 fully applied (few changes from the resolution were missing)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 149 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 166 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 167 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 173 &amp; 175 resolution applied</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Issue 150 resolution applied</td>
</tr>
</tbody>
</table>