SCA Service Component Architecture

Spring Component Implementation Specification

SCA Version 1.0, March 21 2007

Technical Contacts:

Michael Beisiegel	IBM Corporation
Dave Booz	IBM Corporation
Adrian Colyer	Interface21
Hal Hildebrand	Oracle
Jim Marino	BEA Systems, Inc.
Ken Tam	BEA Systems, Inc

Copyright Notice

License

Table of Contents

1. Spring Client and Implementation Model

1.1. Goals

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 will be at the SCA Composite level, where a Spring application context provides a complete composite, exposing services and using references via SCA. This means that a Spring application context defines the internal structure of a composite implementation.

Start from SCA Component Type: It should be possible to use Spring to implement any SCA Composite 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 context and use that composite within an SCA assembly.

1.2. Spring Application Context as Composite Implementation

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

Figure 1 below illustrates a simple SCA domain composed of two composites, both of which are implemented by Spring application contexts.

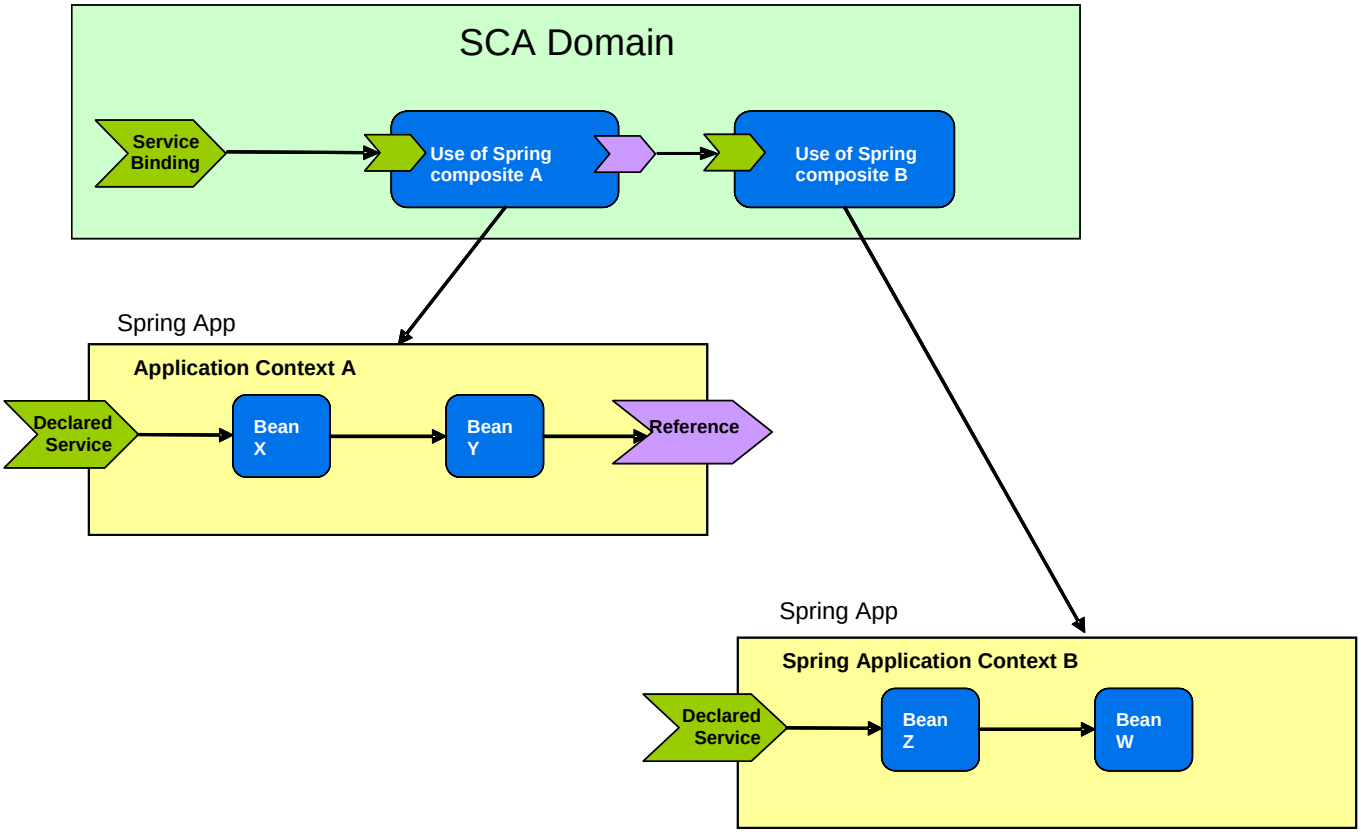

Figure 1 SCA Domain with two Spring application contexts as composite components

In this figure, there are two composites defined by separate Spring Application Contexts, each with one declared service. Composite A is composed of two Spring beans, and bean X is exposed to SCA through an

SCA service. Bean Y has a reference to an external SCA service. This service reference is wired to another Spring context, Composite B, which has a single declared service entry point, 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.

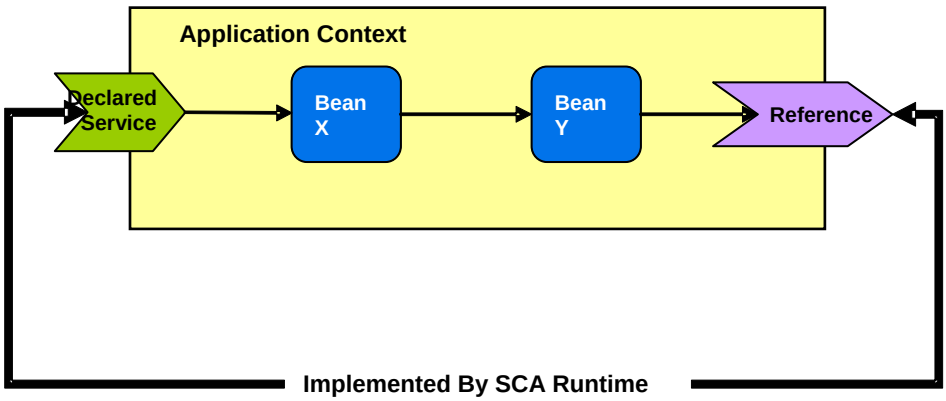

Figure 2

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

1.2.1. 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.

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

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

The SCA SCDL contains service and reference definitions for the Spring composite with appropriate binding information:

```
<composite name="BazComposite">
  <component name="SpringComponent">
```

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

The only part of this that is specific to Spring is the **<implementation.spring>** element. The **location** attribute of that element specifies the target uri of an archive file or directory that contains the Spring application context files. The resource paths to the Spring application context configuration files that are used to create the application context are then identified as follows:

If the resource identified by the location attribute is an archive file, then the file META-INF/MANIFEST.MF is read from the archive. If the location URI identifies a directory, then META-INF/MANIFEST.MF must exist underneath that directory. If the manifest 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 the set of paths specified in the header identify the context configuration files. If there is no MANIFEST.MF file or no Spring-Context header within that file, then the default behaviour is to build an application context using all the *.xml files in the META-INF/spring directory.

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

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

1.2.2. Explicit declaration of SCA related beans inside a Spring configuration

It is also possible to explicitly declare SCA-related beans inside a Spring configuration to proxy SCA references. When inheriting bean definitions created by an SCA runtime in a parent context, a bean defined in the child context with the same name as one in the parent context overrides it. The primary reason you may do this is to enable the Spring container to decorate the bean (using Spring AOP for example).

A reference to an SCA service (known as an SCA reference) is declared using the Spring SCA namespace support.

For example, to declare a bean that represents the service referred to by an SCA reference named "SCAReference" (as discussed in section 2.2.1) you would declare the following:

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

The Spring SCA namespace support provides three elements in total. These 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 the beans that the Spring composite exposes as services. It functions to provide component type information for the Spring composite. Specifically, the SCA runtime is responsible for creating the proper service bindings and applying required policies to those services based on SCDL configuration. If a <sca:service> entry is not configured in the parent SCDL, the SCA runtime

must throw a configuration error. If no <sca:service> elements are specified in the Spring application context, any bean may be exposed as a service. .

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

```
<beans>

   <!-- this definition is not required, and the bean SCAReference could also
         have been inherited from the parent context -->
   <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. Appendix

2.1. Spring SCA Namespace schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://www.springframework.org/schema/sca"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="http://www.springframework.org/schema/sca">

   <xsd:element name="composite">
         <xsd:complexType>
               <xsd:attribute name="component" use="required">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
               <xsd:attribute name="sca-adapter-class" use="optional">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
         </xsd:complexType>
   </xsd:element>

   <xsd:element name="reference">
         <xsd:complexType>
               <xsd:attribute name="name" use="required">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
               <xsd:attribute name="type" use="required">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
               <xsd:attribute name="default" use="optional">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
         </xsd:complexType>
   </xsd:element>

   <xsd:element name="property">
         <xsd:complexType>
               <xsd:attribute name="id" use="optional">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
               <xsd:attribute name="name" use="required">
                     <xsd:simpleType>
                           <xsd:restriction base="xsd:string"/>
                     </xsd:simpleType>
               </xsd:attribute>
               <xsd:attribute name="type" use="required">
```

```
                        <xsd:simpleType>
                              <xsd:restriction base="xsd:string"/>
                        </xsd:simpleType>
                  </xsd:attribute>
            </xsd:complexType>
      </xsd:element>

      <xsd:element name="service">
            <xsd:complexType>
                  <xsd:attribute name="name" use="required">
                        <xsd:simpleType>
                              <xsd:restriction base="xsd:string"/>
                        </xsd:simpleType>
                  </xsd:attribute>
                  <xsd:attribute name="type" use="required">
                        <xsd:simpleType>
                              <xsd:restriction base="xsd:string"/>
                        </xsd:simpleType>
                  </xsd:attribute>
                  <xsd:attribute name="target" use="required">
                        <xsd:simpleType>
                              <xsd:restriction base="xsd:string"/>
                        </xsd:simpleType>
                  </xsd:attribute>
            </xsd:complexType>
      </xsd:element>

</xsd:schema>
```

2.2. References

[1] Spring Framework

http://static.springframework.org/spring/docs/2.0.x/reference/index.html