

# SCA Policy Association Framework

Michael Beisiegel, IBM; Nickolas Kavantzas, Oracle; Ashok Malhotra,  
Oracle; Greg Pavlik, Oracle; Chris Sharp, IBM

{[mbgl@us.ibm.com](mailto:mbgl@us.ibm.com), [nickolas.kavantzas@oracle.com](mailto:nickolas.kavantzas@oracle.com), [ashok.malhotra@oracle.com](mailto:ashok.malhotra@oracle.com),  
[greg.pavlik@oracle.com](mailto:greg.pavlik@oracle.com), [sharp@uk.ibm.com](mailto:sharp@uk.ibm.com)}

**Abstract:** SCA (Service Component Architecture) is a collaborative effort by the leading vendors in the enterprise software space to define an architecture for building applications and systems using a Service-Oriented Architecture. SCA allows developers to define components that implement business logic, which offer their capabilities through services and consume functions offered by other components through references in a relatively abstract manner. This paper discusses how infrastructure and Quality of Service constraints and capabilities can be associated with SCA artifacts either as abstract desires or as concrete policies.

**Keywords:** SCA, QoS, Policy

## 1 Introduction

SCA (Service Component Architecture) is a collaborative effort by the leading vendors in the enterprise such as Web Services [1]. The collaboration started in 2005 and is ongoing. Another paper in this conference describes the SCA framework in detail.

SCA provides a programming model for building applications and systems based on a Service Oriented Architecture. It is based on the idea that business function is provided as a collection of components which are assembled together to create solutions to serve a particular business need. These components offer their capabilities through service-oriented interfaces called services and which consume functions offered by other components through service-oriented interfaces called references. SCA extends and complements prior approaches to implementing services, and builds on open standards. The SCA architecture divides the steps in building a service-oriented application into two major parts:

The **implementation** of components which provide services and consume other services

The **assembly** of sets of components to build business applications, through the **wiring** of references to services.

SCA emphasizes the decoupling of service implementation and of service assembly from the details of Quality of Service (QoS) or infrastructure capabilities and from the details of the access methods used to invoke services. SCA components operate at a business level and use a minimum of middleware APIs.

SCA supports service *implementations* written using any one of several programming languages including conventional object-oriented and procedural languages such as Java™, PHP, C++, COBOL; XML-centric languages such as BPEL and XSLT; and declarative languages such as SQL and XQuery. SCA also supports a range of programming styles, including asynchronous and message-oriented, in addition to the synchronous call-and-return style.

SCA supports *bindings* to a wide range of access mechanisms used to invoke services. These include Web services, Messaging systems and CORBA IIOP. Bindings are handled declaratively and are independent of the implementation code. Infrastructure capabilities, such as Security, Transactions and the use of Reliable Messaging are also handled declaratively and are separated from the implementation code.

The capture and expression of non-functional requirements is an important aspect of service definition, and has impact on SCA throughout the lifecycle of components and compositions. SCA provides a framework to support specification of infrastructure capabilities constraints and Quality of Service (QoS) expectations, from component design through to concrete deployment. Specifically, this paper describes the SCA policy association framework that allows policies and policy subjects specified using [WS-Policy\[2\]](#) and [WS-PolicyAttachment\[3\]](#) and possibly other policy languages to be associated with interactions between SCA components as well as with component implementations.

DISCLAIMER: This paper describes the SCA Policy Attachment Framework in its current state of design. Work on the framework is continuing and so the details are likely to change.

## **2 Policy Framework**

### **2.1 Overview**

The SCA policy framework defines the following key concepts:

#### **Intents**

... allow the SCA developer to specify abstract Quality of Service capabilities or requirements independent of their concrete realization.

#### **Profiles**

... allow the SCA developer to express collections of abstract QoS intents.

#### **Policy Set**

... collects together concrete policy and policy subject pairings, potentially from different policy domains, and declares which intents that they realize collectively.

#### **Intent Maps**

... are a set of alternative concrete policy and policy subject pairs for a single policy domain that appear in a Policy Set

... declare defaults and fixed values for alternatives in a single domain

The SCA policy framework utilizes the following key concepts from the SCA assembly model:

## **Binding**

...some bindings may support intents implicitly through the nature of the protocols they implement

...other bindings may be configurable to provide support for intents through extension

(...SCA bindings may be configured using a combination of intents (via profiles) and concrete policies via policySets)

## **2.2 Framework Model**

SCA Framework model comprises of Intents, Profiles and Policy sets. These concepts are defined below.

### **2.2.1 Intents**

An intent is an abstract capability or requirement that is expressed independent of its realization.

Intent names are represented as case-sensitive strings in the SCA Policy Framework. The name typically designates the name of a policy domain. It may also contain an additional qualifier separated from the domain name by a “/”. It is possible to use a “.” in a domain name as a convention to indicate scoping. For example, “sec.confidentiality” names the confidentiality domain within the security area as an intent. Qualified intents designate an additional level of specificity. When the intent contains a qualifier (i.e. is a qualified intent), the value preceding the separator “/” designates the name of the intent and the value after the separator designates the specific qualifier.

A qualified intent is a member of a qualified intent set. A qualified intent set is the union of qualified intents formed from a single intent. As an example, the sec.confidentiality intent may be qualified with a qualifier from the following

qualifier set: {"message", "transport"}. The qualified intent set, therefore, is as follows: {"sec.confidentiality/message", "sec.confidentiality/transport"}.

To ensure a minimum level of portability, SCA will normatively define a set of core intents that all SCA implementations are expected to provide a concrete realization for. Users of SCA may define new intents, or extend the qualifier set of existing intents.

### 2.2.2 Profiles

Profile elements aggregate intent names. A set of intents (qualified or non-qualified) may be expressed by a profile element using the @intents attribute. This takes a space-separated list of intent names as its value.

For example:

```
<sca:profile intents="sec.authentication
                    rel.reliability
                    sec.confidentiality/message" />
```

The first two intents state that policies providing authentication and reliability are required. Although sec.authentication has a qualifier set it has been used in its unqualified form here. This means that some form of authentication is required, but the SCA developer has not specified the particular type of authentication mechanism used to be used. The third intent is used in its qualified form and states that a policy for a specific capability, "message", in the confidentiality domain is required.

The intents specified in the profile are mapped to concrete policies specified in policySets before deployment. An unqualified intent maps to a policySet that provides a policy or policies for the domain of the intent. If a given policySet provides multiple policies, corresponding to members of the qualified intent set for that domain, the default policy is selected. The specific algorithm used to map intents to PolicySets is outside the scope of this specification.

### 2.2.3 Policy Sets

A policySet element is used to define a set of concrete policies that correspond to a set of intents. The structure of the PolicySet element is as follows:

- It must contain a @name attribute that declares a name for the policy set. The value of the @name attribute is a QName.
- It may contain a @binding attribute. The value of the @binding attribute indicates the binding to which the policySet applies.
- It may contain a @provides element which is a space-separated list on intent names. The values in @provides indicate the intents (qualified or unqualified) that the policySet provides policies for.

A policySet can contain the following element children:

- intentMap element
- policySetReference element
- wsp:PolicyAttachment element
- xs:any extensibility element

Any mix of the above types of elements, in any number, can be included as children. The extensibility elements may be from any namespace and may be intermixed with the other types of child elements. The pseudo schema for policySets is shown below:

```

<sca:policySet name="xs:QName"
               provides="... list of intent
names... "?      bindings="binding
names"?
xmlns="http://www.oesa.org/xmlns/sca/1.0

xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/p
olicy">
  <sca:policySetReference name="xs:QName"/>*
  <sca:intentMap/>*
  <wsp:PolicyAttachment>*
  <xs:any>*
</sca:policySet>

```

As an example, the `policySet` element below declares that it provides “sec.authentication/basic” and “rel.reliability” using the `@provides` attribute, for the “binding.ws” SCA binding.

```
<sca:policySet name="SecureReliablePolicy"
               provides="sec.authentication
                       rel.reliability"
               bindings="sca:binding.ws"

  xmlns="http://www.oesa.org/xmlns/sca/1.0"
  xmlns:wsp=
    "http://schemas.xmlsoap.org/ws/2004/09/polic
y">
  <wsp:PolicyAttachment>

    <!-- policy expression and policy subject
         for "basic authentication" -->

  </wsp:PolicyAttachment>
  <wsp:PolicyAttachment>
    <!-- policy expression and policy subject
for      "reliability" -->
  </wsp:PolicyAttachment>
</sca:policySet>
```

### 2.2.3.1 *IntentMaps*

Intent maps given below contain concrete policies and policy subjects for a named policy domain in the `policySet`. The pseudo-schema for `intentMaps` is shown below:

```
<sca:intentMap provides="xs:QName"
               default="xs:string">
  <sca:qualifier name="xs:string">*

  <wsp:PolicyAttachment>*
    ...
  </wsp:PolicyAttachment>
</sca:qualifier>
<xs:any>*
</sca:intentMap>
```

If a `policySet` contains `intentMaps`, each `intentMap` provides alternative policies for a single policy domain. The intent contained in the `@provides` of the intent

map must be included in the @provides of the policySet. All intents in the @provides of the policySet must appear in the @provides of an intentMap. An intentMap element must contain qualifier element children. Each qualifier element corresponds to a qualified intent where the unqualified form of that intent is included in the @provides attribute value of the parent intentMap.

A qualifier element in an intentMap designates a set of concrete policy attachments that correspond to a qualified intent. Concrete policy attachments may be specified using [wsp:PolicyAttachment\[3\]](#) element children or by using extensibility elements specific to a policy domain.

The default attribute of an intentMap must correspond to one of the qualifier elements of its child qualifier elements. It represents the default choice of qualifier, when only the unqualified form of the intent has been specified as a requirement in a profile element.

As an example, the policySet element below declares that it provides “sca:confidentiality” using the @provides attribute. It contains a single intentMap. The qualifiers (transport and message) each specify the policy and policy subject they provide. The default is “transport”.

```
<sca:policySet name="SecureMessagingPolicies"
  provides="sca:confidentiality"
  bindings="sca:binding.ws"

xmlns="http://www.oesa.org/xmlns/sca/1.0"
  xmlns:wsp=
    "http://schemas.xmlsoap.org/ws/2004/09
/policy">
  <sca:intentMap
provides="confidentiality"
default="transport">
<sca:qualifier name="transport">
  <wsp:PolicyAttachment>

      <!-- policy expression and policy
subject          for          "transport"
alternative -->

</wsp:PolicyAttachment>
  <wsp:PolicyAttachment>
    ...
  </wsp:PolicyAttachment>
</sca:qualifier>
</sca:qualifier>
```

```

name="message">
<wsp:PolicyAttachment>

        <!-- policy expression and policy
subject          for "message" alternative"
-->
                                </wsp:PolicyAttachment>
                                </sca:qualifier>
        </sca:intentMap>
</sca:policySet>

```

### 2.2.3.2 *Direct Use of Attachments within Policy Sets:*

If a policySet contains a complete policy without the need for defaults or overriding, it can contain policy attachment elements directly without the use of intentMaps. In this case, there are two ways of including attachments within a policySet element. Either wsp:policyAttachment elements may be included directly as children or deployment specific extension elements (using xs:any) that designate concrete policy attachments may be included as children.

When a policySet element contains wsp:policyAttachment children, it is assumed that the set of ALL policy attachments specified as children satisfy the intents expressed using the @provides attribute value of the policySet element. This assumption also applies to deployment specific representation of concrete policies.

### 2.2.3.3 *Policy References*

A policySet may refer to other policySets by using a PolicySetReference element. This provides a recursive inclusion capability for intentMaps, policy attachments or other specific policies from different domains.

When a policySet element contains policySetReference element children, the @name attribute of the policySetReference element designates a policySet with the same value for its @name attribute.

The binding attribute of a referenced policySet must be compatible with that of the policySet referring to it. Compatibility, in the simplest case, is string equivalence of binding names.

The @provides attribute of a referenced policySet must include intent values that are compatible with one of the values of the @provides attribute of the referencing policySet. A compatible intent either is a value in the referencing

policySet's @provides attribute values or is a qualified value of one of the intents of the referencing policySet's @provides attribute value.

The use of a policySetReference element indicates that a copy of the element content children of the policySet that is being referred is included within the referring policySet. If the result of inclusion results in a reference to another policySet, inclusion is repeated until the contents of a policySet do not contain any references to other policy sets.

Note that, since the attributes of a referenced policySet are effectively removed/ignored by this process, it is the responsibility of the author of the referring policySet to include any necessary intents in its @provides attribute if they wish that policySet to correctly advertise its aggregate capabilities.

The default values when using this aggregate policySet come from the defaults in the included policySets. A single intent (or all qualified intents that comprise an intent) in a referencing policySet must only be included once by using references to other policySets.

The following example illustrates the inclusion of two other policy Sets in a policySet element:

```
<sca:policySet
  name="BasicAuthMsgProtSecurity"
  provides="authentication confidentiality"
  bindings="binding.ws"

  xmlns="http://www.osoa.org/xmlns/sca/1.0">
  <sca:policySetReference

  name="AuthenticationPolicies"/>
  <sca:policySetReference

  name="ConfidentialityPolicies"/>
</sca:policySet>
```

The above policySet refers to two other policySets for authentication and message protection and, by reference, provides policies and policy subject alternatives in these domains.

## 2.3 Attachment of SCA Policy Artifacts

This section describes the mechanisms used to associate profiles or policySets with SCA artifacts. It describes the various attachment points and semantics for profile elements and their relationship to other SCA elements within their scope of influence, and how profiles relate to policySets in these contexts.

### 2.3.1 Services and References

The SCA developer may declare any requirements that should be satisfied for interactions with their components by attaching a profile element as a child of a service or reference. The meaning of this attachment is that, for all interactions with the service (or via the reference), the intents listed by the profile element should be concretely realized, irrespective of the specific binding that may be chosen.

The following is a pseudo schema for attachment to services and references:

```
<sca:service> or <sca:reference>
    ...
    <sca:interface.interface-type/>
    <sca:profile intents="xs:string"/>?
    ...
</sca:service> or </sca:reference>
```

An example of this is:

```
<sca:service name="mySpecialService">
    <sca:interface.wsdl
portType="..." />
    <sca:profile
        intents="sec.authentication
rel.reliability"/>
    </sca:service>
```

Here, the developer is indicating that it is essential to the operation of mySpecialService that all interactions with it are authenticated and reliable.

Thus, the presence of a profile element, guides the choice of binding that may be used for the service/reference concerned. The selected binding must satisfy the requirements indicated by the profile. In the above example, therefore, any binding element used with the mySpecialService should be capable of realizing authentication and reliability.

Bindings may realize intents either natively by virtue of the kind of transport technology they implement (e.g. an SSL binding would natively support confidentiality automatically) or through configuration using concrete policy artifacts, contained in policySets. The following section looks at the relationship between these three elements (profiles, bindings and policySets) in more detail.

### 2.3.2 Bindings

Profiles may also be associated with bindings to indicate that the intents specified in the profile are mapped to concrete policies contained in policySet elements that are compatible with those bindings. Bindings may also be directly associated with a policySet and the policies specified by it.

The following is a pseudo schema for attachment to a binding:

```
<sca:service> or <sca:reference>
    ...
    <sca:binding.binding-type                >*
      <sca:profile
        intents="xs:string"/>?
      <sca:/binding.binding-type>
    ...
</sca:service> or </sca:reference>
```

#### 2.3.2.1 Associating Profiles with Bindings

When a profile element is in scope of a binding element (i.e. it is either a sibling or child of that binding) then it should be used to ensure that the binding will satisfy the intents specified by the profile. In other words, it is “profiling” the usage of the binding in this instance.

To ensure this, any policySet that may be used to configure that binding (i.e. that declares the binding in question as a value of its binding attribute) should include all the intents expressed by the profile element in its provides list:

An intent that is unqualified in the profile element may be qualified in the policySet @provides attribute.

When an intent is both unqualified in the profile element and the policySet, it may be subsequently qualified by an intentMap/qualifier element contained within the policySet.

An intent that is qualified in the profile element may be unqualified in the @provides attribute of the policySet if and only if that policySet contains an intentMap/qualifier element for that qualified form of the intent.

In this manner, the profile element designates a selection of concrete policies specified by a policySet element (thus overriding the defaults specified in the policySet, should the policySet contain intentMaps).

Although an intentMap typically contains a number of concrete policies to represent different qualified intents, only one of the qualified intents relating to a given unqualified intent may be selected and enforced in a given usage of the policySet. The choice of which qualified intent to use is influenced by the use of profile elements, but sometimes a profile element may not be specific enough to list the particular qualified form of the intent. To ensure one and only one qualified form is chosen a default must be specified.

#### 2.3.2.2 *Associating Policy Sets with Bindings*

A binding element may specify one or more policySets to indicate the specific policies that will be used to configure the binding. The binding element indicates the name of the policy sets by using the @policySet attribute. This attribute can contain a space separated list of policySet names.

As is the case where a profile element is in scope of a binding element, a profile element that is a child of a binding element with a @policySet attribute may specify an attribute named @intents that contain a list of qualified intent values. When used, the value of @intents attribute specifies a set of intents that override the default intents provided by the policySets. This overriding is only possible when the policySet contains intentMaps. For example:

```
<sca:binding.ws policySet="MyEnterprisePolicy">
  <sca:profile
    intents="sec.authentication/cert
             sec.confidentiality/message"/>
</sca:binding.ws>
```

A child profile element adds to or further qualifies the intents specified on profile elements specified for service of reference elements that are a peer to binding elements. The following example shows child profile elements that further qualifies the sca.authentication intent that was specified as a peer to the binding element.

```
<sca:service>
  <sca:profile intents="sec.authentication
```

```

rel.reliability"/>
<sca:binding.ws>
    <sca:profile

intents="sec.authentication/cert"/>
</sca:binding.ws>
    <sca:binding.ejb>
        <sca:profile

intents="sec.authentication/kerberos"/>
</sca:binding.ejb>
</sca:service>

```

### 2.3.3 Implementation Policies

Abstract QoS requirements may be associated with SCA components to indicate implementation policies as shown in the following example.

```

<sca:component name='myComponent' >
    <sca:profile intents="logging"/>
</sca:component>

```

This indicates that all messages to and from the component must be logged. The technology used to implement the logging is unspecified. Specific technology is selected when the intent is mapped to a policySet.

Alternatively, one or more policySets may be specified directly by associating them with the component.

```

<sca:component name="myComponent">
    policySet="StandardImplementationPolicy"/>

```

In this usage, the default intents for each intentMap in the policySet are used.

If the default intents in the policySet(s) need to be overridden, this can be accomplished by specifying the overriding intents by name.

```

<sca:component name="myComponent">
    policySet="StandardImplementationPolicy"/>
    <sca:profile

intents="Access/restricted"/>
</sca:component>

```

## References

1. Service Component Architecture (SCA) <http://www.osoa.org>
2. Web Services Policy (WS-Policy) <http://www.w3.org/TR/2006/WD-ws-policy-20060731>
3. Web Services Policy Attachment (WS-PolicyAttachment) <http://www.w3.org/TR/2006/WD-ws-policy-attachment-20060731>