OASIS

Service Component Architecture (SCA) Tutorial : Part 2

Ashok Malhotra – Oracle
Anish Karmarkar – Oracle
David Booz - IBM

SCA

- A vendor-, technology-, language-neutral model for the creation of business systems using SOA by the composition and deployment of new and existing service components

What is SCA?

- Informal alliance of industry leaders
- to define a language-neutral programming model that meets the needs of enterprise developers who are developing software that exploits Service Oriented Architecture
- aims to provide a model for the creation of service components in a wide range of languages and a model for assembling service components into a business solution.
- components can interact with components outside the SCA system.

What is SCA?

- Informal alliance of industry leaders
- developing a set of specifications
- not a standards body but results will be moved to a standards body when the specifications reach maturity
- website: http://www.osoa.org/

OASIS

SCA Members

- BEA
- Cape Clear
- IBM
- IONA
- Oracle
- Progress Software
- Red Hat
- RogueWave
- SAP
- Siemens
- Software AG
- Sybase
- Sun
- Tibco

OASIS

SCA Policy Framework

SCA Version 1.00, February 15, 2007

Technical Contacts:
Michael Beisiegel, IBM (mbgl@us.ibm.com)
Dave Booz, IBM (booz@us.ibm.com)
Ching-Yun Chao, IBM (cyc@us.ibm.com)
Mike Edwards IBM (mike_edwards@uk.ibm.com)
Sabin Ielceanu, TIBCO (sabin@tibco.com)
Anish Karmarkar Oracle
(anish.karmarkar@oracle.com)
Ashok Malhotra, Oracle
(ashok.malhotra@oracle.com)
Eric Newcomer, IONA
(Eric.Newcomer@iona.com)
Sanjay Patil, SAP (sanjay.patil@sap.com)
Michael Rowley, BEA (mrowley@bea.com)
Chris Sharp, IBM (sharpc@uk.ibm.com)
Ümit Yalçinalp, SAP
(umit.yalcinalp@sap.com)

OASIS

Why Policy is Important for SCA

- Essentially, Policy provides flexibility – a component can be used in different configurations with different QoS requirements and with different bindings on its services and references.
- Policy complexity can be mitigated by starting with simple, relatively abstract requirements which are bound to several concrete realizations.

OASIS

Why Policy is Important for SCA – 2 Steps

1. Implementation of components which provide services and consume other services.
2. Assembly of components to build business applications through connecting services to references.

- In the first step, abstract QoS requirements can be associated with components.
- In the second step, these requirements are translated to policies associated with messages passed over service/reference pairs (interaction policies) or on components (implementation policies).

Lesson Plan

- Intents – abstract QoS requirements
- policySets – map intents to policies
- Use of intents and policySets for interaction policies
- Mapping intents to policies
- Intents for security and reliable messaging
- Use of intents and policySets for implementation policies

Intents

- Intents are abstract specifications of requirements independent of implementation technology or binding.
- The SCA developer uses intents to specify what he needs independent of deployment details which are added later in the process.
- For example, he may specify ‘confidentiality’ or ‘reliability’ without specifying:
  - **- How to achieve?**
  - **- Detailed characteristics?**

Intents (continued)

- In fact, he can say a little bit more ---
- Intents can be qualified – so he can say, for example, ‘confidentiality.transport’ or ‘confidentiality.message’
- These are called qualified intents.

```
<intent name="sca:confidentiality“
constrains="sca:binding"
        <description>
            Protect messages from unauthorized reading.
        </description>
</intent>


<intent name="sca:confidentiality.transport" />
```

Intents (continued) -- Profile Intents

- A profile intent is a macro – a single name for a collection of intents
- It is an intent name that can only be satisfied if all the underlying intents in its @requires list are satisfied.

```
<intent name="sca:messageProtection"
        constrains="sca:binding"
        requires="sca:confidentiality sca:integrity">
        <description>
              Protect messages from unauthorized
              reading or modification.
        </description>
</intent>
```

Intents (continued)

- The SCA Collaboration will define intents and qualified intents for security, reliability and transactionality as starting points.

- SCA Implementations can define other intents for functionality that is important for them.

Policy Sets

- At deployment time, intents are mapped to concrete WS-Policies and

  WS-Policy Attachments via Policy Sets

- A Policy Set has the following structure:

```
<policySet name="xs:QName"
          provides="list of xs:QNames"
          appliesTo="XPath expression">

        <policySetReference name="xs:QName"/>*
        <intentMap/>*
        <wsp:PolicyAttachment>*
        <wsp:Policy>
        <wsp:PolicyReference>
        <xs:any>*
</policySet>
```

Intent Maps

- intentMaps map qualified intents to concrete policies
- Each <qualifier/> element associates a qualified intent name with one or more policy assertions (could be wsp:PolicyAttachment)
  - Each PolicyAttachment element contains a policy expression and a policy subject (what the policy applies to)
- All policies in an intentMap are from a single policy domain
- policySets aggregate intentMaps to create intent-to-policy mappings for multiple domains

Intent Maps

- intentMaps associate intent names with PolicyAttachments: WS-Policy expression plus policy subject

```
<intentMap provides="sca:confidentiality" default="transport">
        <qualifier name="transport">
                <wsp:PolicyAttachment>
                <!-- policy expression and policy subject for
                        "transport" alternative -->
                        …
                </wsp:PolicyAttachment>
                <wsp:PolicyAttachment>
                        ...
                </wsp:PolicyAttachment>
        </qualifier>
        <qualifier name="message">
                <wsp:PolicyAttachment>
                <!--  policy expression and policy subject for
                        "message" alternative” -->
                        ...
                </wsp:PolicyAttachment>
        </qualifier>
</intentMap>
```

Policy Sets

- Policy Sets can also contain Policies or References to Policies directly, without intent maps.

```
<policySet name="sca:userNameTokenHashPassword"
           provides="sca:authentication"
           appliesTo="sca:binding.ws">
  <wsp:Policy>
    <sp:SupportingToken>
      <wsp:Policy>
        <sp:UserNameToken>
          <wsp:Policy>
            <sp:HashPassword>
          </wsp:Policy>
        </sp:UserNameToken>
      </wsp:Policy>
    </sp:SupportingToken>
  </wsp:Policy>
</policySet>
```

Associating Policies with SCA Components

- Intents and/or policySets can be associated with any SCA component.
- At deployment time intents are mapped into Policies contained in policySets
- For example, attaching intents to a service definition

```
<service> or <reference>…
    <binding.binding-type requires="sca:confidentiality"
    </binding.binding-type>…
</service> or </reference>
```

Operation level Attachment

- <operation/> element used as policy attach point
- Service/reference/binding intents are still in effect

```
<service> or <reference>
  <operation name = "xs:string"
             policySet="xs:QName" ?
             requires="list of intent QNames" ? />
</service> or </reference>
```

```
<service> or <reference>
<binding.binding-type requires="list of intent QNames“
                      policySets="list of xs:QNames">
  <operation name = "xs:string"
             policySets="xs:QName" ?
             requires = "list of intent QNames" ? />*
      </binding.binding-type>
</service> or </reference>
```

OASIS

WSDL – direct attachment by extension

Intents can also be associated with interface elements using extensibility points.

```
<portType name="LoanService" sca:requires="sca:confidentiality">
        <operation name="apply">
           <input message="tns:ApplicationInput"/>
           <output message="tns:ApplicationOutput"/>
        </operation>
        <operation name="cancel">
        </operation>
        ...
</portType>
```

Java annotations

- Intents can also be associated directly with Java code
- Generic annotation – for intents without a specific annotation
- Specific annotations for commonly used intents

```
package org.osoa.sca.annotation;
import static org.osoa.sca.annotation.Confidentiality.*;
public class Foo {
   @Requires(CONFIDENTIALITY)
   @Reference
   public void setBar(Bar bar)
   …
}
```

```
package services.hello;
@Remotable
@Confidentiality(“message”)
public class HelloChildService extends HelloService {
        @Confidentiality(“transport”)
        public String hello(String message) {...}
        @Authentication
        String helloWorld(){...}
}
```

Associating Policy Sets with Bindings

- Use binding with an explicitly specified PolicySet
- Default alternatives can be overridden

```
<sca:service> or <sca:reference>…
    <sca:binding.binding-type policySets="sns:enterprisePolicy"
    </sca:binding.binding-type>…
</sca:service> or </sca:reference>
```

  - Overriding default intent alternatives in the PolicySet

```
<sca:reference name="RentalCarService">
  <sca:interface/>
  <sca:binding.WS policySet="sns:BasicSecurity"

requires="sca:authentication.certificateAuthentication
                sns:messageProtection.protectBodyAndHeader"/>
  </sca:binding.WS>
</sca:reference>
```

Intents Provided by Bindings

- Some binding types may satisfy intents by virtue of their implementation technology. For example, an SSL binding would natively support confidentiality.
- Binding instances which are created by configuring a bindingType may be able to provide some intents by virtue of its configuration.
- When a binding type is defined in SCA, these properties are declared as values of the @alwaysProvides and @mayProvide attributes.
- Proprietary implementations on binding types may support different intents.

```
<bindingType type="xs:QName"
             alwaysProvides="list of intent QNames"?
             mayProvide = "list of intent QNames"?/>
```

Recursive Composition

- Intents CANNOT be overriden higher up in recursive composition
  - Intents can be further qualified (i.e. constrained)
- Intent set for SCDL element derived from the element and its ancestor elements
  - See example, both qualified intents MUST be satisfied by the binding/policySets attached to reference “bar”
- PolicySets can be overridden

```
<composite requires="confidentiality.transport">
  <service name="foo" />
  <reference name="bar"
      requires="confidentiality.message"/>
</composite>
```

Mapping Intents to Policy Sets

- We start with a component with abstract QoS requirements
- We want to deploy this with other components in a composite
- So, we must find bindings and/or policySets that satisfy the required intents. This is as follows:
  - Expand out all profile intents
  - Calculate the required intents set (discussed later)
  - Remove intents directly satisfied by the binding or implementation
  - Calculate the explicitly specified policySets (discussed later)
  - Remove intents satisfied by these policySets
  - Find the smallest collection of available policySets that satisfy remaining intents

Mapping Intents to Policy Sets

- Calculating the required intent set

  - Start with the list of intents specified in the element's @requires attribute.
  - Add intents found in the @requires attribute of each ancestor element.
  - If the element is a binding instance and its parent element (service, reference or callback) is wired, the required intents of the other side of the wire may be added to the intent set when they are available. This may simplify, or eliminate, the matching or policies between service and reference.
  - Remove any intents that do not include the target element's type in their @constrains attribute.
  - If the set of intents includes both a qualified version of an intent and an unqualified version of the same intent, remove the unqualified version from the set.

OASIS

Mapping Intents to Policy Sets

- Calculating the explicitly specified policySets
  - Start with the list of policySets specified in the element's @policySets attribute.
  - If any of these explicitly listed policySets does *not* apply to the target element (binding or implementation) then the composite is invalid. Include the values of @policySets attributes from ancestor elements.
  - Remove any policySet where the XPath expression in that policySet's @appliesTo attribute does not match the target element. *It is not an error for an element to inherit a policySet from an ancestor element which doesn't apply*

SCA Intents for Reliable Messaging

- **atLeastOnce:**

message sent by a client is always delivered.

- **atMostOnce:**

message sent by a client is delivered at most once.

- **exactlyOnce:**

message sent by client is delivered exactly once. Combination of atLeastOnce and atMostOnce

- **ordered:**

messages are delivered in the order they were sent by the client.

SCA Intents for Security

- **authentication:** requirement that the client must authenticate itself in order to use an SCA service. Typically, the client security infrastructure is responsible for the server authentication in order to guard against a "man in the middle" attack.

- **confidentiality:** requirement that the contents of a message are accessible only to those authorized to have access (typically the service client and the service provider). A common approach is to encrypt the message; other methods are possible.

- **integrity:** requirement that the contents of a message have not been tampered with and altered between sender and receiver. A common approach is to digitally sign the message; other methods are possible.

SCA Intents for Security - qualifiers

Each of the three basic security intents can be qualified by either “message”, or transport.

message: indicates that the facility is provided at the message level

transport: indicates that the facility provided by the transport, say, SSL

For example: confidentiality.message conveys a requirement that confidentiality be provided at the message level.

Implementation Policies

- Intents and PolicySets can be associated with implementations

```
<sca:component name="myComponent">
    <sca:implementation.* policySets="list of policySet xs:QNames"
                          requires="list of intent xs:QNames">
                ...
          <sca:operation name="xs:string" service="xs:string"?
                         policySets="list of policySet xs:QNames"?
                         requires = "list of intent xs:QNames"?/>*
                    ...
    </sca:implementation>
                    ...
</sca:component>
```

Example of non WS-Policy policySet

```
<policySet provides="sns:logging.trace"
           appliesTo="sca:implementation.bpel">
   <acme:processLogging level="3"/>
</policySet>
```

Security Implementation Policies: Policy Assertions

- Authorization controls who can access the protected SCA resources. A security role is an abstract concept that represents a set of access control constraints on SCA resources.. This is defined as:

```
<allow roles="list of role NCNames">
<permitAll/>
<denyAll/>
```

Security Identity declares the security identity under which an operation will be executed. This is defined as:

```
<runAs role="NCName">
```

Demo

Thank you!

Questions?