SCA Policy Framework Version 1.1

Working Draft 07 + Issue 15 Proposal

08 September 2008

Deleted: Committee

Deleted: 4

Deleted: 1

Deleted: 5

Deleted: 6

Deleted: 1528 Apri

Deleted: 7

Deleted: August

Deleted: July

Deleted: I0

Deleted: 2

Deleted:

Deleted: November

Deleted: 7

Specification URIs:
This Version:

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.html

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.doc

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf

Previous Version:

N/A

Latest Version:

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.doc

http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.pdf (Authoritative)

Technical Committee:

OASIS SCA Policy TC

Chair(s):

David Booz, IBM <booz@us.ibm.com>

Ashok Malhotra, Oracle <ashok.malhotra@oracle.com>

Editor(s):

Deleted: Jeff T. Anderson, Deloitte <jeffanderson@deloitte.ca>

David Booz, IBM <booz@us.ibm.com>

Michael J. Edwards, IBM <mike.edwards@uk.ibm.com>

Ashok Malhotra, Oracle <ashok.malhotra@oracle.com>

Michael Rowley, BEA <mrowley@bea.com>

Related work:

This specification replaces or supercedes:

- SCA Policy Framework

  SCA Policy Framework SCA Version 1.00 March 07, 2007

This specification is related to:

- SCA Assembly Specification

sca-assembly-1.1-spec-WD-02.doc

sca-assembly-1.1-spec-WD-02.pdf

Declared XML Namespace(s):

In this document, the namespace designated by the prefix “sca” is associated with the namespace URL docs.oasis-open.org/ns/opencsa/sca/200712 . This is also the default namespace for this document.

Abstract:

TBD

Status:

This document was last revised or approved by the SCA Policy 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-policy/.

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-policy/ipr.php.

.

Notices

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"and “SCA-Policy” 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

The capture and expression of non-functional requirements is an important aspect of service definition and has an impact on SCA throughout the lifecycle of components and compositions. SCA provides a framework to support specification of constraints, capabilities and QoS expectations from component design through to concrete deployment. This specification describes the framework and its usage.

Specifically, this section describes the SCA policy association framework that allows policies and policy subjects specified using WS-Policy [WS-Policy] and WS-PolicyAttachment [WS-PolicyAttach], as well as with other policy languages, to be associated with SCA components.

This document should be read in conjunction with the SCA Assembly Specification [SCA-Assembly]. Details of policies for specific policy domains can be found in sections 7, 8 and 9.

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 XML Namespaces

Prefixes and Namespaces used in this Specification

Prefix	XML Namespace	Specification
sca	docs.oasis-open.org/ns/opencsa/sca/200712 This is assumed to be the default namespace in this specification. xs:QNames that appear without a prefix are from the SCA namespace.	[SCA]
acme	Some namespace; a generic prefix	
wsp	http://www.w3.org/2006/07/ws-policy	[WS-Policy]
xs	http://www.w3.org/2001/XMLSchema	[XML Schema Datatypes]

1.3 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

[SCA] Service Component Architecture (SCA)
http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

[SCA-Assembly] Service Component Architecture Assembly Model Specification
http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

[SCA-Java-Annotations]
SCA Java Common Annotations and APIs
http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf

[WSDL] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language – Appendix http://www.w3.org/TR/2006/CR-wsdl20-20060327/

[WS-AtomicTransaction]
Web Services Atomic Transaction (WS-AtomicTransaction)
http://docs.oasis-open.org/ws-tx/wsat/2006/06.

[WSDL-Ids] SCA WSDL 1.1 Element Identifiers – forthcoming W3C Note
http://dev.w3.org/cvsweb/~checkout~/2006/ws/policy/wsdl11elementidentifiers.html

[WS-Policy] Web Services Policy (WS-Policy)
http://www.w3.org/TR/ws-policy

[WS-PolicyAttach] Web Services Policy Attachment (WS-PolicyAttachment)
http://www.w3.org/TR/ws-policy-attachment

[XML-Schema2] XML Schema Part 2: Datatypes Second Edition XML Schema Part 2: Datatypes Second Edition, Oct. 28 2004.
http://www.w3.org/TR/xmlschema-2/

Field Code Changed

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

2 Overview

2.1 Policies and PolicySets

The term ***Policy*** is used to describe some capability or constraint that can be applied to service components or to the interactions between service components represented by services and references. An example of a policy is that messages exchanged between a service client and a service provider must be encrypted, so that the exchange is confidential and cannot be read by someone who intercepts the messages.

Deleted: 21

Deleted: conversation

In SCA, services and references can have policies applied to them that affect the form of the interaction that takes place at runtime. These are called ***interaction policies***.

Service components can also have other policies applied to them which affect how the components themselves behave within their runtime container. These are called ***implementation policies***.

How particular policies are provided varies depending on the type of runtime container for implementation policies and on the binding type for interaction policies. Some policies may be provided as an inherent part of the container or of the binding – for example a binding using the https protocol will always provide encryption of the messages flowing between a reference and a service. Other policies can optionally be provided by a container or by a binding. It is also possible that some kinds of container or kinds of binding are incapable of providing a particular policy at all.

Deleted: may

Deleted: may

Deleted: be

In SCA, policies are held in ***policySets***, which may contain one or many policies, expressed in some concrete form, such as WS-Policy assertions. Each policySet targets a specific binding type or a specific implementation type. PolicySets are used to apply particular policies to a component or to the binding of a service or reference, through configuration information attached to a component or attached to a composite.

Deleted: ¶
¶

For example, a service can have a policy applied that requires all interactions (messages) with the service to be encrypted. A reference which is wired to that service needs to support sending and receiving messages using the specified encryption technology if it is going to use the service successfully.

Deleted: must

Deleted: be able

In summary, a service presents a set of interaction policies which it requires the references to use. In turn, each reference has a set of policies which define how it is capable of interacting with any service to which it is wired. An implementation or component can describe its requirements through a set of attached implementation policies.

2.2 Intents describe the requirements of Components, Services and References

SCA ***intents*** are used to describe the abstract policy requirements of a component or the requirements of interactions between components represented by services and references. Intents provide a means for the developer and the assembler to state these requirements in a high-level abstract form, independent of the detailed configuration of the runtime and bindings, which involve the role of application deployer. Intents support the late binding of

Deleted: is

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

services and references to particular SCA bindings, since they assist the deployer in choosing appropriate bindings and concrete policies which satisfy the abstract requirements expressed by the intents.

It is possible in SCA to attach policies to a service, to a reference or to a component at any time during the creation of an assembly, through the configuration of bindings and the attachment of policy sets. Attachment may be done by the developer of a component at the time when the component is written or it may be done later by the deployer at deployment time. SCA recommends a late binding model where the bindings and the concrete policies for a particular assembly are decided at deployment time.

Deleted: directly

SCA favors the late binding approach since it promotes re-use of components. It allows the use of components in new application contexts which may require the use of different bindings and different concrete policies. Forcing early decisions on which bindings and policies to use is likely to limit re-use and limit the ability to use a component in a new context.

For example, in the case of authentication, a service which requires its messages to be authenticated can be marked with an intent "**authentication**". This intent marks the service as requiring message authentication capability without being prescriptive about how it is achieved. At deployment time, when the binding is chosen for the service (say SOAP over HTTP), the deployer can apply suitable policies to the service which provide aspects of WS-Security and which supply a group of one or more authentication technologies.

In many ways, intents can be seen as restricting choices at deployment time. If a service is marked with the **confidentiality** intent, then the deployer must use a binding and a policySet that provides for the encryption of the messages.

The set of intents available to developers and assemblers can be extended by policy administrators. The SCA Policy Framework specification does define a set of intents which address the infrastructure capabilities relating to security, transactions and reliable messaging.

Deleted: arbitrarily

2.3 Determining which policies apply to a particular wire

In order for a reference to connect to a particular service, the policies of the reference must intersect with the policies of the service.

Multiple policies may be attached to both services and to references. Where there are multiple policies, they may be organized into policy domains, where each domain deals with some particular aspect of the interaction. An example of a policy domain is confidentiality, which covers the encryption of messages sent between a reference and a service. Each policy domain may have one or more policy. Where multiple policies are present for a particular domain, they represent alternative ways of meeting the requirements for that domain. For example, in the case of message integrity, there could be a set of policies, where each one deals with a particular security token to be used: e.g. X509, SAML, Kerberos. Any one of the tokens may be used - they will all ensure that the overall goal of message integrity is achieved.

Deleted: ¶

In order for a service to be accessed by a wide range of clients, it is good practice for the service to support multiple alternative policies within a particular domain. So, if a service requires message confidentiality, instead of insisting on one specific encryption technology,

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

the service can have a policySet which has a host of alternative encryption technologies, any of which are acceptable to the service. Equally, a reference can have a policySet attached which defines the range of encryption technologies which it is capable of using. Typically, the set of policies used for a given domain will reflect the capabilities of the binding and of the runtime being used for the service and for the reference.

When a service and a reference are wired together, the policies declared by the policySets at each end of the wire are matched to each other. SCA does not define how policy matching is done, but instead delegates this to the policy language (e.g. WS-Policy) used for the binding. For example, where WS-Policy is used as the policy language, the matching procedure looks at each domain in turn within the policy sets and looks for 1 or more policies which are in common between the service and the reference. When only one match is found, the matching policy is used. Where multiple matches are found, then the SCA runtime can choose to use any one of the matching policies. No match implies that the wire cannot be used - it is an error.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

3 Framework Model

The SCA Policy Framework model is comprised of ***intents*** and ***policySets***. Intents represent abstract assertions and Policy Sets contain concrete policies that may be applied to SCA bindings and implementations. The framework describes how intents are related to PolicySets. It also describes how intents and policySets are utilized to express the constraints that govern the behavior of SCA bindings and implementations. Both intents and policySets may be used to specify QoS requirements on services and references.

The following section describes the Framework Model and illustrates it using Interaction Policies. Implementation Policies follow the same basic model and are discussed later in section 1.5.

3.1 Intents

As discussed earlier, an ***intent*** is an abstract assertion about a specific Quality of Service (QoS) characteristic that is expressed independently of any particular implementation technology. An intent is thus used to describe the desired runtime characteristics of an SCA construct. Intents are typically defined by a policy administrator. See section [Policy Administrator] for a more detailed description of SCA roles with respect to Policy concepts, their definition and their use. The semantics of an intent may not always be available normatively, but could be expressed with documentation that is available and accessible.

For example, an intent named **integrity** may be specified to signify that communications should be protected from possible tampering. This specific intent may be declared as a requirement by some SCA artifacts, e.g. a reference. Note that this intent can be satisfied by a variety of bindings and with many different ways of configuring those bindings. Thus, the reference where the intent is expressed as a requirement could eventually be wired using either a web service binding (SOAP over HTTP) or with an EJB binding that communicates with an EJB via RMI/IIOP.

Intents can be used to express requirements for ***interaction policies*** or ***implementation policies***. The **integrity** intent in the above example is used to express a requirement for an interaction policy. Interaction policies are typically applied to a *service* or *reference*. They are meant to govern the communication between a client and a service provider. Intents may also be applied to SCA component implementations as requirements for ***implementation policies***. These intents specify the qualities of service that should be provided by a container as it runs the component. An example of such an intent could be a requirement that the component must run in a transaction.

For convenience and conciseness, it is often desirable to declare a single, higher-level intent to denote a requirement that could be satisfied by one of a number of lower-level intents. For example, the **confidentiality** intent requires either message-level encryption or transport-level encryption.

Both of these are abstract intents because the representation of the configuration necessary to realize these two kinds of encryption could vary from binding to binding, and each would also require additional parameters for configuration.

Deleted: the

Deleted: always

Deleted:

Deleted: i.e.

Deleted: intents that are

Deleted: An intent is defined using the following pseudo-schema:¶
¶

```
<intent name="NCName"¶
constrains="listOfQNames"
requires="listOfQNames"? >¶
<description>¶
<!-- description of the intent -->¶
</description>¶
</intent>
```

Deleted: ¶
¶

Deleted: Where:¶
¶
<#>@name attribute defines the name of the intent¶
¶
<#>@constrains attribute (optional) specifies the SCA constructs (SCA binding or¶ implementation) that this intent is meant to configure. If a value is not specified, it is¶ assumed that this intent is a qualified intent and inherits its constraint list from the qualifiable intent it is qualifying (see below). This attribute does not define the valid attach points of the intent. ¶
¶
Note that the “constrains” attribute may name a ... [1]

Formatted: English (U.S.)

Deleted: ¶
¶

Deleted: <#>@requires attribute (optional) defines the set of all intents t ... [2]

Deleted: ¶

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

An intent that can be completely satisfied by one of a choice of lower-level intents is referred to as a *qualifiable intent*. In order to express such intents, the intent name may contain a qualifier: a "." followed by a *xs:string* name. An intent name that includes a qualifier in its name is referred to as a *qualified intent*, because it is "qualifying" how the qualifiable intent is satisfied. A qualified intent can only qualify one qualifiable intent, so the name of the qualified intent includes the name of the qualifiable intent as a prefix, for example, **authentication.message**.

In general, SCA allows the developer or assembler to attach multiple qualifiers for a single qualifiable intent to the same SCA construct. However, domain-specific constraints may prevent the use of some combinations of qualifiers (from the same qualifiable intent).

Intents, their qualifiers and their defaults are defined using the following pseudo schema:

```
<intent name="xs:string" constrains ="list of QNames"
      requires="list of QNames" excludes="list of QNames"?
      mutuallyExclusive="boolean"? >
      <description> xs:string.</description>?
      <qualifier name = "xs:string"  default = "xs:boolean" ?>*
         <description> xs:string.</description>?
      </qualifier>
</intent>
```

Where:

- @name is a required attribute that defines the name of the intent

- @constrains attribute (optional) specifies the SCA constructs that this intent is meant to configure. If a value is not specified for this attribute then it can apply to any SCA element.

  Note that the "constrains" attribute may name an abstract element type, such as sca:binding in our running example. This means that it will match against any binding used within a SCDL file. A SCDL element may match @constrains if its type is in a substitution group.

- @requires attribute (optional) defines the set of all intents that the referring intent requires. In essence, the referring intent requires all the intents named to be satisfied. This attribute is used to compose an intent from a set of other intents. This use is further described in Section 3.2 below.

- @excludes attribute (optional) contains a list of the excluded intents as a set of QNames.

  Note that if one intent declares itself to be exclusive of some other intent, it is not required that the other intent also names the original intent in its exclude list, although it is good practice to do this.

  Where one intent is applied to a given artifact in a composition and another intent is applied to one of its parents, which intents apply to the artifact differs depending on whether the two intents are Additive or Mutually Exclusive.

  - Where the intents are Additive, both intents apply to the artifact and its child artifacts.

Deleted: an

Deleted: ,

Deleted: the name of a qualifiable intent

Deleted: ¶

Deleted: (separated by "."),

Deleted: See Usage of @requires attribute for specifying intents.

Deleted: Because qualified intents include the name of the qualifiable intent, the qualifiable intent definition does not need to list its valid qualifiers. The set of all qualified intents defined for that qualifiable intent determines the list of valid qualifiers. This is illust ... [3]

Deleted: Further, the ... [4]

Deleted: ¶

Deleted: <intent ... [5]

Deleted: ¶

Formatted ... [6]

Formatted: Body Text

Deleted:

Deleted:

Deleted:

Deleted:

Deleted:

Formatted ... [7]

Deleted:

Formatted ... [8]

Deleted: ¶

Formatted: Bullets and Numbering ... [9]

Formatted: Bullets and Numbering ... [10]

Deleted: m

Formatted ... [11]

Formatted: Bullets and Numbering ... [12]

Formatted ... [13]

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

- Where the intents are mutually exclusive, only the intent attached directly to the artifact applies to the artifact and to its child artifacts.

- @mutuallyExclusive attribute (optional) with a default of "false". If this attribute is present and has a value of "true" is indicates that the qualified intents defined for this intent are mutually exclusive.

Formatted: Bulleted + Level: 1 + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm

One or more <qualifier> child elements MAY be used to define qualifiers for the intent. The attributes of <qualifier> are:

- @name is a required attribute that defines the name of the intent

Deleted: ¶

Formatted: Bullets and Numbering

- @default is an optional attribute that declares the particular qualifier to be the default qualifier for the intent. If an intent has more than one qualifier, one and only one of them MUST be declared as the default. Further, the names of the qualifiers must be unique within the intent definition.

Formatted: Bullets and Numbering

- The <qualifier> element may have an optional child element called "description" whose value is a xs:string.

Formatted: Bullets and Numbering

For example, the **confidentiality** intent which has qualified intents called **confidentiality.transport** and **confidentiality.message** may be defined as:

```
 <intent name="confidentiality" constrains="sca:binding">
      <description>
            Communication through this binding must prevent
            unauthorized users from reading the messages.
      </description>
      <qualifier name="transport">
         <description>Automatic encryption by transport
         </description>
      </qualifier>
      <qualifier name="message" default='true'>
        <description>Encryption applied to each message
        </description>
      </qualifier>
 </intent>
```

All the intents in a SCA Domain are defined in a global, domain-wide file named definitions.xml. Details of this file are described in the SCA Assembly Model [SCA-Assembly].

SCA normatively defines a set of core intents that all SCA implementations are expected to support, to ensure a minimum level of portability. Users of SCA may define new intents, or extend the qualifier set of existing intents.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

3.2 Profile Intents

An intent that is satisfied only by satisfying *all* of a set of other intents is called a **profile intent**. It can be used in the same way as any other intent.

The presence of @requires attribute in the intent definition signifies that this is a profile intent. The @requires attribute may include all kinds of intents, including qualified intents and other profile intents. However, while a profile intent can include qualified intents, it cannot BE a qualified intent (so its name must not have "." in it).

Deleted: I

Requiring a profile intent is always semantically identical to requiring the list of intents that are listed in its @requires attribute.

An example of a profile intent could be an intent called **messageProtection** which is a shortcut for specifying both **confidentiality** and **integrity**, where **integrity** means to protect against modification, usually by signing. The intent definition may look like the following:

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

3.3 PolicySets

A ***policySet*** element is used to define a set of concrete policies that apply to some binding type or implementation type, and which correspond to a set of intents provided by the policySet.

Deleted: The structure of the PolicySet element is as follows:

The pseudo schema for policySet is shown below:

```
<policySet name="NCName"
            provides="listOfQNames"
            appliesTo="xs:string"
            attachTo="xs:string"
            xmlns=http://www.osoa.org/xmlns/sca/1.0
            xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <policySetReference name="xs:QName"/>*
      <intentMap/>*
      <xs:any>*
</policySet>
```

PolicySet has the following attributes:

- The @name attribute declares a name for the policySet. The value of the @name attribute is a xs:QName.
- The @appliesTo attribute is used to determine which SCA constructs this policySet can configure. The contents of the attribute must match the XPath 1.0 production *Expr*.
- The @attachTo attribute is a string which is an XPath 1.0 expression identifying one or more elements in the SCDL within the Domain. It is used to declare which set of

Formatted: Bullets and Numbering

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

elements the policySet is actually attached to. See the section on "Attaching Intents and PolicySets to SCA Constructs" for more details on how this attribute is used.

- The @provides attribute, whose value is a list of intent names (that may or may not be qualified), designates the intents the PolicySet provides. Members of the list are xs:string values separated by a space character " ".

PolicySet contains one or more of the following element children

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

Any mix of the above types of elements, in any number, can be included as children of the policySet element including extensibility elements. There are likely to be many different policy languages for specific binding technologies and domains. In order to allow the inclusion of any policy language within a policySet, the extensibility elements may be from any namespace and may be intermixed. However, the SCA policy framework expects that WS-Policy will be a common policy language for expressing interaction policies, especially for Web Service bindings.

It is often desirable to attach WS-Policies directly as children of <policySet> elements; either directly as <wsp:Policy> elements, or as <wsp:PolicyReference> elements or using <wsp:PolicyAttachment>. These three elements, and others, can be attached using the extensibility point provided by the <xs:any> in the pseudo schema above. See example below.

For example, the policySet element below declares that it provides **authentication.message** and **reliability** for the "binding.ws" SCA binding.

```
<policySet name="SecureReliablePolicy"
           provides="authentication.message exactlyOne"
           appliesTo="sca:binding.ws"
           xmlns="http://www.osoa.org/xmlns/sca/1.0"
           xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <wsp:PolicyAttachment>
           <!-- policy expression and policy subject for
                "basic authentication" -->
            …
      </wsp:PolicyAttachment>
      <wsp:PolicyAttachment>
           <!-- policy expression and policy subject for
                "reliability" -->
            …
      </wsp:PolicyAttachment>
</policySet>
```

PolicySet authors should be aware of the evaluation of the @appliesTo attribute in order to designate meaningful values for this attribute. Although policySets may be attached to any element in the SCA design, the applicability of a policySet is not scoped by where it is attached in the SCA framework. Rather, policySets always apply to either binding instances or implementation elements regardless of where they are attached to. In this regard, the SCA policy framework does not scope the applicability of the policySet to a specific attachment point in contrast to other frameworks, such as WS-Policy. Attachment is a shorthand.

Deleted: It

Deleted: <#>wsp:PolicyAttachment element¶
<#>wsp:Policy element¶
<#>wsp:PolicyReference element¶

Formatted: Body Text, Adjust space between Latin and Asian text

Deleted: For this reason, wsp:PolicyAttachment is explicitly included in the schema for clarity.

Deleted: The pseudo schema for policySet is shown below:¶
¶
<policySet name="NCName"¶
provides="listOfQNames"¶
appliesTo="xs:string"¶
xmlns=http://www.osoa.org/xmlns/sca/1.0¶
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">¶
<policySetReference name="xs:QName"/>*¶
<intentMap/>*¶

Formatted: Body Text

Deleted: <wsp:PolicyAttachment>*¶
<wsp:Policy>*¶
<wsp:PolicyReference>*¶

Deleted: <xs:any>*¶
</policySet>¶
¶

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

With this design principle in mind, an XPath expression that is the value of an @appliesTo attribute designates what a policySet applies to. Note that the XPath expression will always be evaluated within the context of an attachment considering elements where binding instances or implementations are allowed to be present. The expression is evaluated against *the parent element of any binding or implementation element*. The policySet will apply to any child binding or implementation elements returned from the expression. So, for example, appliesTo=”binding.ws” will match any web service binding. If appliesTo=”binding.ws[@impl=’axis’]” then the policySet would apply only to web service bindings that have an @impl attribute with a value of ‘axis’.

For further discussion on attachment of policySets and the computation of applicable policySets, please refer to Section 4.

All the policySets in a SCA Domain are defined in a global, domain-wide file named definitions.xml. Details of this file are described in the SCA Assembly Model [SCA-Assembly].

SCA may normatively define a set of core policySets that all SCA implementations are expected to support, to ensure a minimum level of portability. Users of SCA may define new policySets as needed.

3.3.1 IntentMaps

Intent maps contain the concrete policies and policy subjects that are used to realize a specific intent that is provided by the policySet.

The pseudo-schema for intentMaps is given below:

```
<intentMap provides="xs:QName"
            >
      <qualifier name="xs:string">?
            <xs:any>*
            <intentMap/> ?
      </qualifier>
</intentMap>
```

Deleted: default="xs:string"

Deleted: <wsp:PolicyAttachment>*¶
…¶
</wsp:PolicyAttachment>¶

It is often desirable to attach WS-Policies directly as children of <qualifier> elements; either directly as <wsp:Policy> elements, or as <wsp:PolicyReference> elements or using <wsp:PolicyAttachment>. These three elements, and others, can be attached using the extensibility point provided by the <xs:any> in the pseudo schema above.

When a policySet element contains a set of intentMap elements, the value of the @provides attribute of each intentMap corresponds to an unqualified intent that is listed within the @provides attribute value of the parent policySet element.

If a policySet specifies a qualifiable intent in the @provides attribute, then it MUST include an intentMap element that specifies all possible qualifiers for that intent. If a qualified intent can be further qualified, then the qualifier element must also contain an intentMap.

For each intent (qualified or unqualified) listed as a member of the @provides attribute list of a policySet element, there may be at most one corresponding intentMap element that

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

declares the unqualified form of that intent in its @provides attribute. In other words, each intentMap within a given policySet must uniquely provide for a specific intent.

The @provides attribute value of each intentMap that is an immediate child of a policySet must be included in the @provides attribute of the parent policySet.

> An intentMap element must contain qualifier element children. Each qualifier element corresponds to a qualified intent where the unqualified form of that intent is the value of the @provides attribute value of the parent intentMap. The qualified intent is either included explicitly in the value of the enclosing policySet's @provides attribute or implicitly by that @provides attribute including the unqualified form of the intent. One of the qualifiers referenced in the intentMap MUST be the default qualifier defined for the qualifiable intent.

A qualifier element designates a set of concrete policy attachments that correspond to a qualified intent. The concrete policy attachments may be specified using wsp:PolicyAttachment element children or using extensibility elements specific to an environment.

As an example, the policySet element below declares that it provides **confidentiality** using the @provides attribute. The alternatives (transport and message) it contains each specify the policy and policy subject they provide. The default is "transport".

```
<policySet name="SecureMessagingPolicies"
            provides="confidentiality"
            appliesTo="binding.ws"
            xmlns="http://www.osoa.org/xmlns/sca/1.0"
            xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <intentMap provides="confidentiality" >
            <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>
</policySet>
```

PolicySets can embed policies that are defined in any policy language. Although WS-Policy is the most common language for expressing interaction policies, it is possible to use other policy languages. The following is an example of a policySet that embeds a policy defined in a proprietary language. This policy provides "authentication" for binding.ws.

Deleted: The default attribute of an intentMap must correspond to a qualified intent that is named on one of the child qualifier elements. This is used when the unqualified form of the intent has been specified as a requirement. The relationship between intents and policySets, and their use within SCDL is explained in more detail in section 1.5.

Deleted: ¶
¶

Deleted: default="transport"

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
<policySet name="AuthenticationPolicy"
            provides="authentication"
            appliesTo="binding.ws"
            xmlns="http://www.osoa.org/xmlns/sca/1.0">
      <e:policyConfiguration xmlns:e=”http://example.com”>
            <e:authentication type = “X509”/>
                  <e:trustedCAStore type=”JKS”/>
                  <e:keyStoreFile>Foo.jks</e:keyStoreFile>
                  <e:keyStorePassword>123</e:keyStorePassword>
            </e:authentication>
      </e:policyConfiguration>
</policySet>
```

The following example illustrates an intent map that defines policies for an intent with more than one level of qualification.

```
<policySet name=”SecurityPolicy” provides=”confidentiality”>
      <intentMap provides=”confidentiality” >
            <qualifier name=”message”>
                  <intentMap provides=”message” >
                        <qualifier name=”body”>
                              <! --- policy attachment for body encryption →
                        </qualifier>
                        <qualifier name=”whole”>
                              <! --- policy attachment for whole message
                              →encryption
                        </qualifier>
                  </intentMap>
            </qualifier>
            <qualifier name=”transport”>
                              <! --- policy attachment for transport
                              encryption →
            </qualifier>
      </intentMap>
</policySet>
```

Deleted: default=”message”

Deleted: default=”whole”

3.3.2 Direct Inclusion of Policies within PolicySets

In cases where there is no need for defaults or overriding for an intent included in the @provides of a policySet, the policySet element may contain policies or policy attachment elements directly without the use of intentMaps or policy set references. There are two ways of including policies directly within a policySet. Either the policySet contains one or more wsp:policyAttachment elements directly as children or it contains extension elements (using xs:any) that contain concrete policies.

When a policySet element directly contains wsp:policyAttachment children or policies using extension elements, it is assumed that the set of policies specified as children satisfy the intents expressed using the @provides attribute value of the policySet element. The intent names in the @provides attribute of the policySet may include names of profile intents.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

3.3.3 Policy Set References

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

Deleted: ¶

Deleted: ¶

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

The @appliesTo 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 the 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 usage of a policySetReference element indicates 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, the inclusion step is repeated until the contents of a policySet does not contain any references to other policySets.

When a policySet is applied to a particular element, the policies in the policy set include any standalone polices plus the policies from each intent map contained in the PolicySet as described below.

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 the @provides attribute if the policySet is 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.

Here is an example to illustrate the inclusion of two other policySets in a policySet element:

```
<policySet name="BasicAuthMsgProtSecurity"
            provides="authentication confidentiality"
            appliesTo="binding.ws"
            xmlns="http://www.osoa.org/xmlns/sca/1.0">
      <policySetReference name="acme:AuthenticationPolicies"/>
      <policySetReference name="acme:ConfidentialityPolicies"/>
</policySet>
```

The above policySet refers to policySets for **authentication** and **confidentiality** and, by reference, provides policies and policy subject alternatives in these domains.

If the policySets referred to have the following content:

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
<policySet name="AuthenticationPolicies"
            provides="authentication"
            appliesTo="binding.ws"
            xmlns="http://www.osoa.org/xmlns/sca/1.0">
      <wsp:PolicyAttachment>
            <!-- policy expression and policy subject for "basic
            authentication" -->
                  …
      </wsp:PolicyAttachment>
</policySet>

<policySet name="acme:ConfidentialityPolicies"
            provides="confidentiality"
            bindings="binding.ws"
            xmlns="http://www.osoa.org/xmlns/sca/1.0">
      <intentMap provides="confidentiality" >
            <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>
</policySet>
```

Deleted: default="transport"

The result of the inclusion of policySets via policySetReferences would be semantically equivalent to the following:

```
<policySet name="BasicAuthMsgProtSecurity"
                  provides="authentication confidentiality"
                  appliesTo="binding.ws"
                   xmlns="http://www.osoa.org/xmlns/sca/1.0">
      <wsp:PolicyAttachment>
            <!-- policy expression and policy subject for "basic
            authentication" -->
                  ...
      </wsp:PolicyAttachment>
      <intentMap provides="confidentiality" >
             <qualifier name="transport">
                  <wsp:PolicyAttachment>
                  <!-- policy expression and policy subject for "transport"
                  alternative -->
                  ...
                  </wsp:PolicyAttachment>
                  <wsp:PolicyAttachment>
                  ...
```

Deleted: default="transport"

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
                  </wsp:PolicyAttachment>
            </qualifier>
            <qualifier name="message">
                  <wsp:PolicyAttachment>
                  <!-- policy expression and policy subject for "message"
                  alternative -->
                  ...
                  </wsp:PolicyAttachment>
            </qualifier>
      </intentMap>
</policySet>
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

4 Attaching Intents and PolicySets to SCA Constructs

This section describes how intents and policySets are associated with SCA constructs. It describes the various attachment points and semantics for intents and policySets and their relationship to other SCA elements and how intents relate to policySets in these contexts.

4.1 Attachment Rules - Intents

Intents can be attached to any SCA element used in the definition of components and composites since an intent specifies an abstract requirement. The attachment is specified by using the optional ***@requires*** attribute. This attribute takes as its value a list of intent names. Intents can optionally be applied to interface definitions. For WSDL Port Type elements (WSDL 1.1) and for WSDL Interface elements (WSDL 2.0), the @requires attribute can be applied that holds a list of intent names that are required for the interface. Other interface languages may define their own mechanism for specifying a list of required intents. Any service or reference that uses an interface with required intents implicitly adds those intents to its own @requires list.

Because intents specified on interfaces can be seen by both the provider and the client of a service, it is appropriate to use them to specify characteristics of the service that both the developers of provider and the client need to know. For example, the fact that an interface is *conversational* is such a characteristic, since both the client and the service provider need to know about the conversational semantics.

For example:

```
<service> or <reference>…
      <binding.binding-type requires="listOfQNames"
      </binding.binding-type>…
</service> or </reference>
```

4.2 Attachment Rules - PolicySets

One or more policySets can be attached to any SCA element used in the definition of components and composites. The attachment is specified by using one of two mechanisms:

- ***Direct Attachment*** using the optional ***@policySets*** attribute of the SCA element
- the ***External Attachment*** mechanism

The policySets attribute takes as its value a list of policySet names.

For example:

```
<service> or <reference>…
      <binding.binding-type policySets="listOfQNames"
      </binding.binding-type>…
</service> or </reference>
```

The SCA Policy framework enables two distinct cases for utilizing intents and PolicySets:

Formatted: Body Text, Adjust space between Latin and Asian text

Formatted: Body Text, Adjust space between Latin and Asian text

Formatted: Font: Bold, Italic

Formatted: Font color: Black

Formatted: Heading 2, Adjust space between Latin and Asian text

Deleted: Similarly, one

Formatted: Font: Bold, Italic

Formatted: Font: Bold, Italic

Formatted: Bullets and Numbering

Deleted: .

Formatted: Font: Bold, Italic

Deleted: is

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

- It is possible to specify QoS requirements by specifying abstract intents utilizing the @requires element on an element at the time of development. In this case, it is implied that the concrete bindings and policies that satisfy the abstract intents are not assigned at development time but the intents are used ***to select the concrete Bindings and Policies*** at deployment time. Concrete policies are encapsulated within policySets that are applied during deployment using the external attachment mechanism. The intents associated with a SCA element is the union of intents specified for it and its parent elements subject to the detailed rules below.

- It is also possible to specify QoS requirements for an element by using both intents and concrete policies contained in directly attached policySets at development time. In this case, it is possible ***to configure the policySets, by overriding the default settings in the specified policySets using intents***. The policySets associated with a SCA element is the union of policySets specified for it and its parent elements subject to the detailed rules below.

  When computing the policySets that apply to a particular element, the @appliesTo attribute of each relevant policySet is checked against the element. If the policySet is attached directly to the element and does not apply to that element an error is raised. If a policySet that is attached to an ancestor element does not apply to the element in question, it is simply discarded.

These two different approaches of specifying policies are illustrated in detail below. Also discuss is how intents are used to guide the selection and application of specific policySets.

4.3 External Attachment of PolicySets Mechanism

The External Attachment mechanism for policySets is used for deployment-time application of policySets to SCA elements. It is called "external attachment" because the principle of the mechanism is that the place that declares the attachment is separate from the composite files which hold the elements. This separation provides the deployer with a way to attach policySets without having to modify the artifacts where they apply.

A PolicySet is attached to one or more elements in one of two ways:
a) through the use of a <PolicyAttachment/> element which is a child of a <definitions/> element in a definitions file
b) through the @attachTo attribute of the PolicySet

The pseudo-schema for the Policy Attachment element is:

```
<sca:definitions>
   ...
   <sca:PolicyAttachment policySet="QName" attachTo="xs:string"/> +
   ...
</sca:definitions>
```

The PolicyAttachment element attaches a single PolicySet to a set of locations in the SCDL. It has 2 attributes:

- policySet (required) – QName of the PolicySet to attach
- attachTo (required) – string which is an XPath 1.0 expression identifying one or more elements in the SCDL to which the policySet is attached (See below for details)

Formatted: Body Text, Bulleted + Level: 1 + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm, Adjust space between Latin and Asian text, Tabs: Not at 0.63 cm

Deleted: will not be

Deleted: will be

Deleted: will be available in a deployment environment

Formatted: Body Text, Bulleted + Level: 1 + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm, Adjust space between Latin and Asian text

Deleted: ¶
<#>¶

Deleted: will be

Deleted: We a

Formatted: Heading 2, Adjust space between Latin and Asian text

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Font: Courier New

Formatted: Bullets and Numbering

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

The meaning of the @attachTo attribute of the PolicyAttachment element is identical to the meaning of the @attachTo attribute of the PolicySet element. This is described in the next subsection.

4.3.1 The Form of the @attachTo Attribute

The @attachTo attribute of a PolicySet or of a PolicyAttachment is an XPath1.0 expression identifying a SCA element to which the PolicySet is attached.

The XPath applies to the ***Infoset for External Attachment*** – ie to SCA composite files, with the following special characteristics:

1. The Domain is treated as a special composite, with a blank name - ""

2. Where one composite includes one or more other composites, it is the including composite which is addressed by the XPath and its contents are the result of preprocessing all of the include elements

3. Where the PolicySet is intended to be specific to a particular use of a composite file (rather than to all uses), each (nested) component is given a unique URI for each use of the component, based on a concatenation of all the names of the components involved, starting with the name of the component at the Domain level.

   The XPath expression can make use of the unique URI to indicate specific use instances, where different policySets need to be used for those different instances.

Special case. Where the @attachTo attribute of a PolicySet is absent or is blank, the PolicySet cannot be used on its own for external attachment. It can be used:

1. For direct attachment

2. By reference from another PolicySet or from a <PolicyAttachment/> element

Such a policySet can in principle be applied to any element through these means.

The XPath expression for the @attachTo attribute can make use of a series of XPath functions which enable the expression to easily identify elements with specific characteristics that are not easily expressed with pure XPath. These functions enable:

- the identification of elements to which specific intents apply.
  This permits the attachment of a PolicySet to be linked to specific intents on the target element - for example, a PolicySet relating to encryption of messages can be targeted to services and references which have the ***confidentiality*** intent applied.

- the targeting of subelements of an interface, including operations and messages.
  This permits the attachment of a PolicySet to an individual operation or to an individual message within an interface, separately from the Policies that apply to other operations or messages in the interface.

- the targeting of a specific use of a component, through its unique URI.
  This permits the attachment of a PolicySet to a specific use of a component in one

Formatted: Heading 3, Adjust space between Latin and Asian text

Formatted: Font: Bold, Italic

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Font: Bold, Italic

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

context, that can be different from the PolicySet(s) that are applied to other uses of the same component.

Detail of the available XPath functions is given in a following section.

Examples of @attachTo attribute:

1. //component(@name="test3")

attach to all instances of a component named "test3"

2. //component/URIRef("top_level/test1/test3")

attach to the unique instance of component "test3" when used by component "test1" when used by component "top_level" (top_level is a component at the Domain level)

3. //component(@name="test3")/service(IntentRefs("intent1"))

Formatted: French (France)

selects the services of component "test3" which have the intent "intent1" applied

4. //component/binding.ws

selects the web services binding of all components with a service or reference with a Web services binding

5. /composite(@name="")/component(@name="fred")

selects a component with the name "fred" at the Domain level

Formatted: Heading 3, Adjust space between Latin and Asian text

4.3.2 Cases Where Multiple PolicySets are attached to a Single Artifact

Multiple PolicySets can be attached to a single artifact. This can happen either as the result of one or more direct attachments using the @policySets attribute plus one or more external attachments which target the particular artifact.

Where multiple PolicySets are attached to a single artifact, all of the PolicySets attached apply to the artifact.

Formatted: Heading 3, Adjust space between Latin and Asian text

4.3.3 XPath Functions for the @attachTo Attribute

Utility functions are useful in XPath expressions where otherwise it would be complex to write the XPath expression to identify the required elements.

This particularly applies in SCA to Interfaces and the child parts of interfaces (operations and messages). XPath Functions are proposed for the following:

- Picking out a specific interface
- Picking out a specific operation in an interface
- Picking out a specific message in an operation in an interface
- Picking out artifacts with specific intents

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

4.3.3.1 Interface Related Functions

InterfaceRef(InterfaceName)
picks out an interface identified by InterfaceName

OperationRef(InterfaceName/OperationName)
picks out the operation OperationName in the interface InterfaceName

MessageRef(InterfaceName/OperationName/MessageName)
picks out the message MessageName in the operation OperationName in the interface InterfaceName.

"*" can be used for wildcarding of any of the names.

The interface is treated as if it is a WSDL interface (for other interface types, they are treated as if mapped to WSDL using their regular mapping rules).

Examples of the Interface functions:

InterfaceRef("MyInterface")

picks out an interface with the name "MyInterface"

OperationRef("MyInterface/MyOperation")

picks out the operation named "MyOperation" within the interface named "MyInterface"

OperationRef("*/MyOperation")

picks out the operation named "MyOperation" from any interface

MessageRef("MyInterface/MyOperation/MyMessage")

picks out the message named "MyMessage" from the operation named "MyOperation" within the interface named "MyInterface"

MessageRef("*/*/MyMessage")

picks out the message named "MyMessage" from any operation in any interface

4.3.3.2 Intent Based Functions

For the following intent-based functions, it is the total set of intents which apply to the artifact which are examined by the function, including directly attached intents plus intents acquired from the structural hierarchy and from the implementation hierarchy.

IntentRefs(IntentList)
picks out an element where the intents applied match the intents specified in the IntentList:

IntentRefs("intent1")

picks out an artifact to which intent named "intent1" is attached

Formatted: Heading 4, Adjust space between Latin and Asian text

Formatted: Font: Bold

Formatted: Font: Not Bold

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Heading 4, Adjust space between Latin and Asian text

Formatted: Font: Bold

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

IntentRefs("intent1 intent2")
picks out an artifact to which intents named "intent1" AND "intent2" are attached

IntentRefs("intent1 !intent2")

picks out an artifact to which intent named "intent1" is attached but NOT the intent named "intent2"

4.3.3.3 URI Based Function

The following function is used to pick out a particular use of a nested components – ie where some Domain level component is implemented using a composite implementation which in turn may have one or more components implemented with a composite (and so on to an arbitrary level of nesting):

URIRef(URI)

picks out the particular use of a component identified by the URI string URI.

Example:

URIRef("top_comp_name/middle_comp_name/lowest_comp_name")

picks out the particular use of a component – where component lowest_comp_name is used within the implementation of middle_comp_name within the implementation of the top-level (Domain level) component top_comp_name.

4.4 Usage of @requires attribute for specifying intents

A list of intents can be specified for any SCA element by using the @requires attribute.

The intents which apply to a given element depend on

- the intents expressed in its @requires attribute
- intents derived from the structural hierarchy of the element
- intents derived from the implementation hierarchy of the element

When computing the intents that apply to a particular element, the @constrains attribute of each relevant intent is checked against the element. If the intent in question does not apply to that element it is simply discarded.

The structural hierarchy of an element consists of its parent element, grandparent element and so on up to the <composite/> element in the composite file containing the element.

As an example, for the following composite:

```
<composite name="C1" requires="i1">
   <service name="CS" promotes="X/S">
      <binding.ws requires="i2">
   </service>
   <component name="X">
       <implementation.java class="foo"/>
```

Formatted: Heading 4, Adjust space between Latin and Asian text

Formatted: Body Text, Adjust space between Latin and Asian text

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Bullets and Numbering

Deleted: ¶
As indicated

Deleted: , a

Deleted: optional

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
       <service name="S" requires="i3">
   </component>
</composite>
```

- the structural hierarchy of the component service element with the name "S" is the component element named "X" and the composite element named "C1". Service "S" has intent "i3" and also has the intent "i1" if i1 is not mutually exclusive with i3.

Rule 1: An element inherits any intents specified on the elements above it in its structural hierarchy EXCEPT

- if any of the inherited intents is mutually exclusive with an intent expressed on the element, then the inherited intent is ignored
- if the overall set of intents from the element itself and from its structural hierarchy contains both an unqualified version and a qualified version of the same intent, only the qualified version of the intent is used (whichever element was the source of the qualified intent)

The ***implementation hierarchy*** occurs where a component configures an implementation and also where a composite promotes a service or reference of one of its components. The implementation hierarchy involves:

- a composite service or composite reference element is in the implementation hierarchy of the component service/component reference element which they promote
- the component element and its descendent elements (for example, service, reference, implementation) configure aspects of the implementation. Each of these elements is in the implementation hierarchy of the ***corresponding*** element in the componentType of the implementation.

Rule 2: An element acquires the intents defined by the elements lower in its implementation hierarchy and it can only add intents or further qualify intents. Added intents MUST NOT be mutually exclusive with any of the intents attached lower in the hierarchy. A qualifiable intent expressed lower in the hierarchy can be qualified further up the hierarchy, in which case the qualified version of the intent applies to the higher level element. Intents from the implementation hierarchy take precedence over those from the structural hierarchy.

As an example, consider the following composite:

```
<composite name="C1" requires="i1">
   <service name="CS" promotes="X/S">
      <binding.ws requires="i2">
   </service>
   <component name="X">
       <implementation.java class="foo"/>
       <service name="S" requires="i3">
   </component>
</composite>
```

...the component service with name "S" has the service named "S" in the componentType of the implementation in its implementation hierarchy, and the composite service named "CS"

Formatted: Space Before: 4 pt, After: 4 pt, Adjust space between Latin and Asian text

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

has the component service named "S" in its implementation hierarchy. Service "CS" acquires the intent "i3" from service "S" – and also gets the intent "i1" from its containing composite "C1" IF i1 is not mutually exclusive with i3.

When intents apply to an element following the rules described and where no policySets are attached to the element, the intents for the element can be used to select appropriate policySets during deployment, using the external attachment mechanism.

Consider the following composite:

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

…in this case, the composite declares that all of its services and references must guarantee confidentiality in their communication, but the "bar" reference further qualifies that requirement to specifically require message-level security. The "foo" service element has the default qualifier specified for the confidentiality intent (which might be transport level security) while the "bar" reference has the **confidentiality.message** intent.

Consider this variation where a qualified intent is specified at the composite level:

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

In this case, both the **confidentiality.transport** *and* the **confidentiality.message** intent are required for the reference 'bar'. If there are no bindings that support this combination, an error will be generated. However, since in some cases multiple qualifiers for the same intent may be valid or there may be bindings that support such combinations, the SCA specification allows this.

It is also possible for a qualified intent to be further qualified. In our example, the **confidentiality.message** intent may be further qualified to indicate whether just the body of a message is protected, or the whole message (including headers) is protected. So, the second-level qualifiers might be "body" and "whole". The default qualifier might be "whole". If the "bar" reference from the example above wanted only body confidentiality, it would state:

```
<reference name="bar" requires="acme:confidentiality.message.body"/>
```

The definition of the second level of qualification for an intent follows the same rules. As with other qualified intents, the name of the intent is constructed using the name of the qualifiable intent, the delimiter ".", and the name of the qualifier.

4.5 Usage of @requires and @policySet attributes together

As indicated above, it is possible to attach both intents and policySets to an SCA element during development. The most common use cases for attaching both intents and concrete policySets to an element are with binding and reference elements.

Deleted: Stating intents with the @requires attribute of an element means that those intents are additionally required by every relevant element descendent. For example, specifying¶ requires="confidentiality" on a <composite> element is the equivalent to adding the same intent to the @requires list of every service and reference that is contained within that composite, including the services and references inside components. ... [14]

Deleted: are specifie ... [15]

Deleted: during ... [16]

Deleted: is

Deleted: computed

Deleted: are

Deleted: .

Deleted: The intents ... [17]

Deleted: Both qualifie ... [18]

Deleted: ¶

Deleted: I

Deleted: has

Deleted: d

Deleted: would

Deleted: y

Deleted: . When the ... [19]

Deleted: will

Deleted: use

Deleted: by the Polic ... [20]

Deleted: will use

Deleted: During polic ... [21]

Deleted: ¶

Deleted: <#>If a ... [22]

Deleted: 615

Deleted: ¶

Formatted: Bullets and Numbering ... [23]

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

When the @requires attribute and the @policySets attributes are used together during development, it indicates the intention of the developer to configure the element, such as a binding, by the application of specific policySet(s) to this element.

Developers using @requires and @policySet attributes in conjunction with each other must be aware of the implications of how the policySets are selected and how the intents are utilized to select specific intentMaps, override defaults, etc. The details are provided in the Section Guided Selection of PolicySets using Intents.

4.6 Operation-Level Intents and PolicySets on Services & References

It is possible to specify intents and policySets for a single service or reference operation in a way that applies to all the bindings of a service or reference. In this case, the syntax is to specify the operation directly under the <sca:service> or <sca:reference> element. The following example illustrates the placement of the <sca:operation> element:

```
<service> or <reference>
      <operation name = "xs:string"
            policySet="xs:QName"? requires="="listOfQNames"? />
</service> or </reference>
```

The SCA Runtime MUST execute the algorithm in section **Error! Reference source not found.** **Error! Reference source not found.** one time for each operation in a service or reference interface when operation level policy attachment (intents or policySets) is used.

4.7 Operation-Level Intents and PolicySets on Bindings

The above mechanism for specifying operation-specific required intents and policySets may also be applied to bindings. In this case, the syntax would be:

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

This makes it possible to specify required intents that are specific to one operation for a single binding. The SCA Runtime MUST execute the algorithm in **Error! Reference source not found.** **Error! Reference source not found.** one time for each operation in a service or reference interface when operation level policy attachment (intents or policySets) is used.

4.8 Intents and PolicySets on Implementations and Component Types

It is possible to specify required intents and policySets within a component's implementation, which get exposed to SCA through the corresponding *component type*. How the intents or policies are specified within an implementation depends on the

Deleted: that are in scope for

Deleted: *The same algorithm applies whether the intents guide the selection of policySets during deployment or whether a developer uses intents to choose the best alternative in a set of policySets that may apply by configuring policySets.*

Formatted: Bullets and Numbering

Deleted: 4.10

Deleted: Guided Selection of PolicySets using Intents

Formatted: Bullets and Numbering

Deleted: 4.10

Deleted: Guided Selection of PolicySets using Intents

Deleted: Similar to operations on implementations, the intents required for the operation are added to the effective list of required intents on the binding, and operation-level policySets override corresponding policySets specified for the binding (where a "corresponding" policySet @provides at least one common intent).¶

Formatted: Bullets and Numbering

Deleted: for

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

implementation technology. For example, Java can use an @requires annotation to specify intents.

Deleted: the

The required intents and policySets specified within an implementation can be found on the <sca:implementation.*> and the <sca:service> and <sca:reference> elements of the component type, for example:

Deleted: various

```
<omponentType>
      <implementation.* requires="listOfQNames"
            policySets="="listOfQNames">
             ...
      </implementation>
      <service name="myService" requires="listOfQNames"
            policySets="listOfQNames">
             ...
      </service>
      <reference name="myReference" requires="listOfQNames"
            policySets="="listOfQNames">
             ...
      </reference>
            …
</componentType>
```

Intents expressed in the component type are handled according to the rule defined for the implementation hierarchy.

Deleted: When applying policies, the i

Deleted: required by

For explicitly listed policySets, the list in the component using the implementation may override policySets from the component type. More precisely, a policySet on the componentType is considered to be overridden, and is not used, if it has a @provides list that includes an intent that is also listed in any component policySet @provides list.

Deleted: added to the intents¶
required by the using component

Deleted: the

Formatted: Bullets and Numbering

4.9 BindingTypes and Related Intents

SCA Binding types implement particular communication mechanisms for connecting components together. See detailed discussion in the SCA Assembly specification [SCA-Assembly]. Some binding types may realize intents inherently by virtue of the kind of protocol technology they implement (e.g. an SSL binding would natively support confidentiality). For these kinds of binding types, it may be the case that using that binding type, without any additional configuration, will provide a concrete realization of a required intent. In addition, binding instances which are created by configuring a bindingType may be able to provide some intents by virtue of its configuration. It is important to know, when selecting a binding to satisfy a set of intents, just what the binding types themselves can provide and what they can be configured to provide.

The bindingType element is used to declare a class of binding available in a SCA Domain. It declares the QName of the binding type, and the set of intents that are natively provided using the optional @alwaysProvides attribute. The intents listed by this attribute are always concretely realized by use of the given binding type. The binding type also declares the intents that it may provide by using the optional @mayProvide attribute. Intents listed as the value of this attribute can be provided by a binding instance configured from this binding type.

The pseudo-schema for the bindingType element is as follows:

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
<bindingType type="NCName"
            alwaysProvides="listOfQNames"? mayProvide="listOfQNames"?/>
```

The kind of intents a given binding might be capable of providing, beyond these inherent intents, are implied by the presence of policySets that declare the given binding in their @appliesTo attribute. An exception is binding.sca which is configured entirely by the intents listed in its @mayProvide and @alwaysProvides lists. There are no policySets with appliesTo="binding.sca".

For example, if the following policySet is available in a SCA Domain it says that the sca:binding.ssl can provide "reliability" in addition to any other intents it may provide inherently.

```
<policySet name="ReliableSSL" provides="exactlyOnce"
            appliesTo="binding.ssl">
            ...
</policySet>
```

4.10 Treatment of Components with Internal Wiring

Formatted: Bullets and Numbering

This section discusses the steps involved in the development and deployment of a component and its relationship to selection of bindings and policies for wiring services and references.

The SCA developer starts by defining a component. Typically, this will contain services and references. It may also have required intents defined at various locations within composite and component types as well as policySets defined at various locations.

Both for ease of development as well as for deployment, the wiring constraints to relate services and references need to be determined. This is accomplished by matching constraints of the services and references to those of corresponding references and services in other components.

In this process, the required intents, the binding instances, and the policySets that may apply to both sides of a wire play an important role. It must be possible to find binding instances on each side of a wire that are compatible with one another. In addition, concrete policies must be determined that satisfy the required intents for the service and the reference and are also compatible with each other. For services and references that make use of bidirectional interfaces, the same determination of matching bindings and policySets must also take place for the callbackReference and callbackService.

Determining compatibility of wiring plays an important role prior to deployment as well as during the deployment phases of a component. For example, during development, it helps a developer to determine whether it is possible to wire services and references when the bindings and policySets are available in the development environment. During deployment, the wiring constraints determine whether wiring can be achievable. It does also aid in adding additional concrete policies or making adjustments to concrete policies in order to deliver the constraints. Here are the concepts that are needed in making wiring decisions:

- The set of required wiring intents that individually apply to *each* service or reference.

- When possible the intents that are required by the service, the reference and callback (if any) at the other end of the wire. This set is called the *required intent set*

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

and is computed and MAY be used only when dealing with a wire connecting two components within the SCA Domain. When external connections are involved, from clients or to services that are outside the SCA domain, intents are only available for the end of the connection that is inside the domain. See Section "Preparing Services and References for External Connection" for more details.

- The binding instances that apply to each side of the wire.

- The policySets that apply to each service or reference.

There may be many binding instances specified for a reference/service. If there are no binding instances specified on a service or a reference, then <sca:binding.sca> is assumed.

The set of *provided intents* for a binding instance is the union of the intents listed in the "alwaysProvides" attribute and the "mayProvides" list of of its binding type (although the capabilities represented by the "mayProvides" intents will only be present if the intent is in the list of required intents for the binding instance). When an intent is directly provided by the binding type, there is no need to use policy set that provides that intent.

When bidirectional interfaces are in use, the same selection of binding instances and policySets that provide the required intent are also performed for the callback bindings.

4.10.1 Determining Wire Validity and Configuration

The above approach determines the policySets that should be used in conjunction with the binding instances listed for services and references. For services and references that are resolved using SCA wires, the bindings and policySets chosen on each side of the wire may or may not be compatible. The following approach is used to determine whether they are compatible and the wire is valid. If the wire uses a bidirectional interface, then the following technique must find that valid configured bindings can be found for both directions of the bidirectional interface.

Note that there may be many binding instances present at each side of the wire. The wiring compatibility algorithm below determines the compatibility of a wire by a pairwise choice of a binding instance and the corresponding policySets on each side of the wire.

A *potential binding pair* is a pair of binding instances, one on each end of the wire, that have the same binding type. Each binding instance in the pair has a set of policy sets that were determined by the algorithm of the last section. If any potential binding pair has policySets on each end that are *incompatible*, then that pair of binding instances is removed as an option. The compatibility of policySets is determined by the policy language contained in the policySets. However, there are some special cases worth mentioning:\

- If both sides of the wire use the identical policySet (by referring to the same policySet by its QName in both sides of the wire), then they are compatible.

- If the policySets contain WS-Policy attachments, then the following steps are used to determine their compatibility:

1) The sca:policySet

Deleted: The policySets that apply to a service or reference are determined by starting with the policySets that are explicitly specified on that service or reference, adding in the policy sets for any ancestor element, and then finding the smallest set of additional policySets that provide the required wiring intents that have not already been satisfied inherently by the binding instances. **(**Please refer to the Guided Selection of PolicySets using Intents for specifics of how the final set of policySets are determined. Selection of the policySets utilize the required wiring intents that are computed above.)¶
¶

Formatted: Bullets and Numbering

Formatted: Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

Formatted: Outline numbered + Level: 1 + Numbering Style: Bullet + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

Formatted: Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

2) Reference elements within the policySet elements are removed recursively by replacing each reference with an equivalent policy expression encapsulated with sca:policySet element.

3) The policy expressions within each policy set are normalized using WS-Policy normalization rules to obtain a set of alternatives on each side of the wire.

4) The resulting policy alternatives from each side of the wire are pairwise tested for compatibility using the WS-Policy intersection algorithm. WS-Policy's *strict* compatibility should be used by default.

5) If the result of the WS-Policy intersection algorithm is non-empty, then the policy sets are considered compatible.

Formatted: Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

Formatted: Outline numbered + Level: 3 + Numbering Style: 1, 2, 3, … + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

For other policy languages, the policy language defines the comparison semantics. Where such policy languages are standardized by the SCA specifications, the SCA specifications will reference the definition of the comparison semantics or, if no such definition exists, the SCA specifications will provide a definition.

4.11 Preparing Services and References for External Connection

Formatted: Bullets and Numbering

Services and references are sometimes not intended for SCA wiring, but for communication with software that is outside of the SCA domain. References may contain bindings that specify the endpoint address of a service that exists outside of the current SCA domain. Composite services that are deployed to the virtual domain composite specify bindings that can be exposed to clients that are outside of the SCA domain. When web service bindings are used, these services also may generate WSDL with attached policies that can be accessed by external clients (as described in the SCA Web Service Binding specification).

Component services and references that have been promoted to composite services and references may connect to references and services in another SCA Domain or a non-SCA Domain. This section discusses the steps involved in the preparing such a service or reference for external connection.

Essentially, this involves generating a WSDL interface for the service/reference and attaching to it policies that reflect abstract QoS requirements specified using intents and specific requirements using attached policySets. This section will discuss only the generation of policies. Generation of the WSDL interface is discussed in specifications for the various bindings, for example, binding.ws.

Matching service/reference policies across the SCA Domain boundary will use WS-Policy compatibility (strict WS-Policy intersection) if the policies are expressed in WS-Policy syntax. For other policy languages, the policy language defines the comparison semantics. Where such policy languages are standardized by the SCA specifications, the SCA specifications will reference the definition of the comparison semantics or, if no such definition exists, the SCA specifications will provide a definition.

For external services and references that make use of bidirectional interfaces, the same determination of matching policies must also take place for the callback.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

The policies that apply to the service/reference are now computed as discussed in Guided Selection of PolicySets using Intents.

4.12 Guided Selection of PolicySets using Intents

Formatted: Bullets and Numbering

This section describes the selection of concrete policies that satisfy a set of required intents expressed for an element. The purpose of the algorithm is to construct the set of concrete policies that apply to an element taking into account the explicitly declared policySets that may be attached to an element as well as the externally attached. The aim is to satisfy all of the intents expressed for each element.

Deleted: policySets available in the SCA Domain that are selected to match a required intent

Note: In the following algorithm, the following rule is observed whenever an intent set is computed.

When a profile intent is encountered in either a @requires or @provides attribute, it is assumed that the profile intent is immediately replaced by the intents that it is composed by, namely by all the intents that appear in the profile intent's @requires attribute. This rule is applied recursively until profile intents do not appear in an intent set. [This is stated generally, in order to not have to restate this processing step at multiple places in the algorithm].

Deleted: ¶

Formatted: Body Text, Adjust space between Latin and Asian text

Algorithm for Matching Intents and PolicySets:

A. Calculate the ***required intent set*** that applies to the target element as follows:

1. Start with the list of intents specified in the element's @requires attribute.
2. Add intents found in any related interface definition.
3. Add intents found in the inherited @requires attributes of each ancestor element in the element's structural hierarchy as defined in Rule 1 in Section 4.2.
4. Add intents found on elements below the target element in its implementation hierarchy as defined in Rule 2 in Section 4.2.
5. 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 policy matching step later described in step C.
6. Remove any intents that do not include the target element's type in their @constrains attribute.
7. 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.
8. Replace any remaining qualifiable intents with the default qualified form of that intent, according to the default qualifier in the definition of the intent.
9. If the list of intents contains a mutually exclusive pair of intents, raise an error.

Deleted: <#>Add intents found in the @requires attribute of each ancestor element.¶

Formatted: Bullets and Numbering

** The required intent set now contains all intents that must be provided for the target element.*

B. Remove all directly supported intents from the required intent set. Directly supported intents are:

- For a binding instance, the intents listed in the @alwaysProvides attribute of the binding type definition as well as the intents listed in the binding type's @mayProvides attribute that are selected when the binding instance is configured.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

- For a implementation instance, the intents listed in the @alwaysProvides attribute of the implementation type definition as well as the intents listed in the implementation type's @mayProvides attribute that are selected when the implementation instance is configured.

** The remaining required intents must be provided by policySets.*

C. Calculate the list of policySets which are attached to the target element.

The list of PolicySets which attached include those explicitly specified using the @policySets attribute and those which are externally attached.

In this calculation, a policySet *applies to* a target element if the XPath expression contained in the policySet's @appliesTo attribute is evaluated against the parent of the target element and the result of the XPath expression includes the target element. For example, @appliesTo="binding.ws[@impl='axis']" will match any binding.ws element that has an @impl attribute value of 'axis'.

The list of ***explicitly specified*** policySets is calculated as follows:

1. Start with the list of policySets specified in the element's @policySets attribute.
2. If any of these explicitly listed policySets does *not* apply to the target element (binding or implementation) then the composite is invalid. *The point of this rule is that it must have been a mistake to have explicitly listed a policySet on a binding or implementation element that cannot apply to that element.*
3. Include the values of @policySets attributes from ancestor elements.
4. 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.*

The list of ***externally attached*** policySets is calculated as follows:

1. For each <PolicyAttachment/> and <PolicySet/> element in the Domain, if the element is targeted by their @attachTo attribute, then the identified PolicySet applies to the element.
2. 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 be the target of a policySet which doesn't apply.*

A policySet matches a required intent if any of the following are true:

1. The required intent matches a provides intent in a policySet exactly.
2. The provides intent is a parent (e.g. prefix) of the required intent (in this case the policySet must have an intentMap entry for the requested qualifier)
3. The provides intent is more qualified than the required intent.

D. Remove all required intents that are provided by the specified policySets.

** All intents should now be satisfied.*

F. If the collection of policySets does not cover all the required intents, the configuration is not valid.

Deleted: explicitly specified

Deleted: that apply

Formatted: Font: Bold, Italic

Formatted: Body Text, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.63 cm + Tab after: 1.9 cm + Indent at: 1.9 cm, Adjust space between Latin and Asian text, Tabs: Not at 0.63 cm + 1.27 cm

Formatted: Font: Bold, Italic

Formatted: Body Text, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.63 cm + Tab after: 1.9 cm + Indent at: 1.9 cm, Adjust space between Latin and Asian text

Formatted: Font: Italic

Formatted: Body Text, Numbered + Level: 1 + Numbering Style: 1, 2, 3, … + Start at: 1 + Alignment: Left + Aligned at: 0.63 cm + Tab after: 1.9 cm + Indent at: 1.9 cm, Adjust space between Latin and Asian text, Tabs: Not at 0.63 cm + 1.27 cm

Deleted: ** The remaining required intents, if any, are provided by finding additional matching policySets within the SCA Domain.*¶
¶
E. Choose the smallest collection of additional policySets that match all remaining required intents.¶
¶
¶

Deleted: no

Deleted: s

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

When the configuration is not valid, it means that the required intents are not being correctly satisfied. However, an SCA Domain may allow a deployer to force deployment even in the presence of such errors. The behaviors and options enforced by a deployer is not specified.

Deleted: G. If there is not one unique smallest collection of policySets that satisfy all required intents, then the composite definition document is not valid. The composite definition must be changed so that either it has enough explicit policySets declared that the ambiguity is removed or additional intents are added to remove the ambiguity.¶
¶

Deleted: H. If a required intent is unqualified and matches a policySet that is also unqualified, then the intentMap entry for the qualifier that is marked with default=”true” should be used.¶

Deleted: ¶

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

5 Implementation Policies

The basic model for Implementation Policies is very similar to the model for interaction policies described above. Abstract QoS requirements, in the form of intents, may be associated with SCA component implementations to indicate implementation policy requirements. These abstract capabilities are mapped to concrete policies via policySets at deployment time. Alternatively, policies can be associated directly with component implementations.

The following example shows how intents can be associated with an implementation:

```
<component name="xs:NCName" … >
      <implementation.* …
            requires="listOfQNames">
            …
      </implementation>
            …
</component>
```

Formatted: English (U.S.)

If, for example, one of the intent names in the value of the @requires attribute is 'logging', 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 (unless the implementation type has native support for the intent, as described in the next section). A list of required implementation intents may also be specified by any ancestor element of the <sca:implementation> element. The effective list of required implementation intents is the union of intents specified on the implementation element and all its ancestors.

In addition, one or more policySets may be specified directly by associating them with the implementation of a component.

```
<component name="xs:NCName" … >
      <implementation.*
            policySets="="listOfQNames">
                  …
      </implementation>
                  …
</component>
```

If any of the explicitly listed policy sets includes an intent map, then the intent map entry used will be the one for the appropriate intent qualifier(s) listed in the effective list of required intents. If no qualifier is specified for an intent map's qualifiable intent, then the default qualifier is used.

The above example shows how intents and policySets may be specified on a component. It is also possible to specify required intents and policySets within the implementation. How this is done is defined by the implementation type.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

The required intents and policy sets are specified on the <sca:implementation.*> element within the component type. This is important because intent and policy set definitions need to be able to specify that they constrain an appropriate implementation type.

```
<componentType>
      <implementation.* requires="listOfQNames" policySets="listOfQNames">
            …
</implementation>
            …
</componentType>
```

Formatted: English (U.S.)

When applying policies, the intents required by the implementation are added to the intents required by the using component. For the explicitly listed policySets, the list in the component may override policySets from the component type. More precisely, a policySet on the componentType is considered to be overridden, and is not used, if it has a @provides list that includes an intent that is also listed in any component policySet @provides list.

5.1 Natively Supported Intents

Each implementation type (e.g. <sca.implementation.java> or <sca.implementation.bpel>) has an implementation type definition within the SCA Domain. The form of the implementation type definition is as follows:

```
<implementationType type="NCName"
            alwaysProvides="listOfQNames"? mayProvide="listOfQNames"?/>
```

The @type attribute should specify the QName of an XSD global element definition that will be used for implementation elements with of that type (e.g. sca:implementation.java). There are two lists of intents. The intents in the @mayProvide list are provided only for components that require them (they are present in the effective list of required intents). The intents in the @alwaysProvides list are provided irrespective of the list of required intents.

5.2 Operation-Level Intents and PolicySets on Implementations

It is also possible to declare implementation policies that apply only to specific operations of a service, rather than all of them, by associating intents and policySets with individual operations contained within implementations. The syntax is analogous to that proposed above. See the pseudo-schema below:

```
<component name="xs:NCName">
      <implementation.* policySets="listOfQNames"
                  requires="list of intent xs:QNames">
                   …
                  <operation name="xs:string" service="xs:string"?
                        policySets="listOfQNames"?
                  requires="listOfQNames"?/>*
                  …
      </implementation>
                   …
</component>
```

Formatted: English (U.S.)

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

As in the pseudo-schema displayed earlier, the intents associated with the operation appear as the value of the optional @requires attribute. PolicySets may also be explicitly associated with the operation by using the optional @policySets attribute. If a policySet that is listed in @policySets provides a qualifiable intent that also is listed in the effective required intent list, then the qualifier is used to override the default qualifier in the policySet.

Operations are identified by names which are xs:string values. The operation names will be names defined by the interface definition language. For example, for Java interfaces they will be Java names. For WSDL, they will be WSDL1.1 identifiers. See[WSDL -IDs] or WSDL 2.0 Component Identifier names See [WSDL]. If more than one service implemented by this implementation has an operation with the same name, then the @service attribute is required in order to disambiguate them. However, if more than one operation within a single service has the same name (i.e. it is overloaded) then the values of the attributes @requires and @policySet are associated with *all* operations with that name. SCA does not currently provide a means for disambiguating overloaded operations.

The algorithm for mapping of intents to policySets is described in Section Guided Selection of PolicySets using Intents.

5.3 Writing PolicySets for Implementation Policies

The @appliesTo attribute for a policySet takes an XPath expression that is applied to a binding or an implementation element. For implementation policies, in most cases, all that is needed is the QName of the implementation type. Implementation policies may be expressed using any policy language (which is to say, any configuration language). For example, XACML or EJB-style annotations may be used to declare authorization policies. Other capabilities could be configured using completely proprietary configuration formats. For example, a policySet declared to turn on trace-level logging for some fictional BPEL executions engine would be declared as follows:

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

PolicySets or intent map entries may include PolicyAttachment elements. A PolicyAttachment element has a child-element called AppliesTo followed by a policy expression. The AppliesTo indicates the subject that the policy applies to. In the SCA case, the policy subject is indicated by where the policySet is attached and so, this will generally be omitted. (This AppliesTo element should not be confused with the @appliesTo attribute for a policySet. They have quite different meanings.)

Following the AppliesTo is a policy expression. In WS-Policy [WS-Policy] this can be a WS-Policy expression or a WS-PolicyReference, For SCA, we need to generalize this to contain policy expressions in other policy languages.

5.3.1 Non WS-Policy Examples

Authorization policies expressed in XACML could be used in the framework in two ways:

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

1. Embed XACML expressions directly in the PolicyAttachment element using the extensibility elements discussed above, or
2. Define WS-Policy assertions to wrap XACML expressions.

For EJB-style authorization policy, the same approach could be used:

1. Embed EJB-annotations in the PolicyAttachment element using the extensibility elements discussed above, or
2. Use the WS-Policy assertions defined as wrappers for EJB annotations.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

6 Roles and Responsibilities

There are 4 roles that are significant for the SCA Policy Framework. The following is a list of the roles and the artifacts that the role creates:

- Policy Administrator – policySet definitions and intent definitions
- Developer – Implementations and component types
- Assembler - Composites
- Deployer – Composites and the SCA Domain (including the logical Domain-level composite)

Formatted: Indent: First line: 0 cm, Bulleted + Level: 1 + Aligned at: 0 cm + Tab after: 0 cm + Indent at: 0 cm

6.1 Policy Administrator

An intent represents a requirement that a developer or assembler can make, which ultimately must be satisfied at runtime. The full definition of the requirement is the informal text description in the intent definition.

The **policy administrator**'s job is to both define the intents that are available and to define the policySets that represent the concrete realization of those informal descriptions for some set of binding type or implementation types. See the sections on intent and policySet definitions for the details of those definitions.

6.2 Developer

When it is possible for a component to be written without assuming a specific binding type for its services and references, then the **developer** uses intents to specify requirements in a binding neutral way.

If the developer requires a specific binding type for a component, then the developer can specify bindings and policySets with the implementation of the component. Those bindings and policySets will be represented in the component type for the implementation (although that component type might be generated from the implementation).

If any of the policySets used for the implementation include intentMaps, then the default choice for the intentMap can be overridden by an assembler or deployer by requiring a qualified intent that is present in the intentMap.

6.3 Assembler

An **assembler** creates composites. Because composites are implementations, an assembler is like a developer, except that the implementations created by an assembler are composites made up of other components wired together. So, like other developers, the assembler can specify required intents or bindings or policySets on any service or reference of the composite.

However, in addition the definition of composite-level services and references, it is also possible for the assembler to use the policy framework to further configure components

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

within the composite. The assembler may add additional requirements to any component's services or references or to the component itself (for implementation policies). The assembler may also override the bindings or policySets used for the component. See the assembly specification's description of overriding rules for details on overriding.

As a shortcut, an assembler can also specify intents and policySets on any element in the composite definition, which has the same effect as specifying those intents and policySets on every applicable binding or implementation below that element (where applicability is determined by the @appliesTo attribute of the policySet definition or the @constrains attribute of the intent definition).

6.4 Deployer

A **deployer** deploys implementations (typically composites) into the SCA Domain. It is the deployers job to make the final decisions about all configurable aspects of an implementation that is to be deployed and to make sure that all required intents are satisfied.

If the deployer determines that an implementation is correctly configured as it is, then the implementation may be deployed directly. However, more typically, the deployer will create a new composite, which contains a component for each implementation to be deployed along with any changes to the bindings or policySets that the deployer desires.

1093 When the deployer is determining whether the existing list of policySets is correct for a component, the deployer needs to consider both the explicitly listed policySets as well as the policySets that will be chosen according to the algorithm specified in Guided Selection of PolicySets using Intents.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

7 Security Policy

The SCA Security Model provides SCA developers the flexibility to specify the required level of security protection for their components to satisfy business requirements without the burden of understanding detailed security mechanisms.

The SCA Policy framework distinguishes between two types of policies: ***interaction policy*** and ***implementation policy***. Interaction policy governs the communications between clients and service providers and typically applies to Services and References. In the security space, interaction policy is concerned with client and service provider authentication and message protection requirements. Implementation policy governs security constraints on service implementations and typically applies to Components. In the security space, implementation policy concerns include access control, identity delegation, and other security quality of service characteristics that are pertinent to the service implementations.

The SCA security interaction policy can be specified via intents or policySets. Intents represent security quality of service requirements at a high abstraction level, independent from security protocols, while policySets specify concrete policies at a detailed level which are typically security protocol specific.

The SCA security policy can be specified either in the SCDL or annotatively in the implementation code. Language-specific annotations are described in the respective language Client and Implementation specifications.

7.1 SCA Security Intents

The SCA security specification defines the following intents to specify interaction policy: authentication, confidentiality, and integrity.

authentication – the authentication intent is used to indicate that a 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 – the confidentiality intent is used to indicate 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, although other methods are possible.

integrity – the integrity intent is used to indicate that assurance is required 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, although other methods are possible.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

7.2 Interaction Security Policy

Any one of the three security intents may be further qualified to specify more specific business requirements. Two qualifiers are defined by the SCA security specification: transport and message, which can be applied to any of the above three intent's.

7.2.1 Qualifiers

transport – the transport qualifier specifies the qualified intent should be realized at the transport layer of the communication protocol.

message – the message qualifier specifies that the qualified intent should be realized at the message level of the communication protocol.

The following example snippet shows the usage of intents and qualified intents.

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

In this case, the composite declares that all of its services and references must guarantee confidentiality in their communication by setting requires="confidentiality". This applies to the "foo" service. However, the "bar" reference further qualifies that requirement to specifically require message-level security by setting requires="confidentiality.message".

7.2.2 Operation Level Intents

Intents may be specified at the operation level. The operation element does not distinguish operations with different arguments. Operation level intents override the service level intents of the same type. For example an operation level "confidentiality.message" intent would override service level "confidentiality" intent, but would not override other types of intents at service level such as "integrity" and "authentication" intents.

Use the following implementation as an example.

```
public interface HelloService {
String hello(String message);
}

import org.osoa.sca.annotations.*;

@Service(HelloServiceImpl.class)
public class HelloServiceImpl implements HelloService {
  public String hello(String message) {
...
}
```

Formatted: French (France)

Consider the following composite document:

```
<service name="HelloServiceImpl"
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
            requires="authentication integrity.transport
            confidentiality.transport">
      <interface.wsdl interface="…#wsdl.interface(HelloService)"/>
      <operation name="hello"
            requires="authentication.message integrity.message"/>
      <binding.ws/>
</service>
```

The effective QoS intent's on the "hello" operation of the HelloService are "authentication.message", "integrity.message", and "confidentiality.transport".

7.2.3 References to Concrete Policies

In addition to the SCA intent model's late binding approach, developers can reference concrete policies explicitly by attaching policySets directly, as shown below:

```
<service name="foo">
      <interface.wsdl interface="..." />
      <binding.ws policySets="acme:CorporatePolicySet3"/>
</service>
```

It is possible to use the @requires attribute and the @policySets attributes together during 1184 development, it indicates the intention of the developer to configure the element, such as a binding, by the application of specific @policySets that are in scope for this element using the computed intents that apply to this element. The @requires attribute designates a configuration of concrete policies specified by the policySets overiding the defaults specified in the policySets.

7.3 Implementation Security Policy

SCA security model provides a policy reference mechanism which can specify security implementation policy files external to the SCA composite document. Security implementation policy of component implementation such as EJB can be defined in J2EE deployment descriptor ejb-jar.xml which can be referred to by the policy reference document. Additionally SCA security model defines a security implementation policy that may be used by POJO component implementation as well as other type of component implementations.

7.3.1 Authorization and Security Identity Policy

Two policy assertions are defined which apply to implementations – ***Authorization*** and ***SecurityIdentity***. 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 such as composites, components, and operations. The approach and scope of the mapping of role names to security principals is SCA runtime implementation dependent. Scope implies the set of artifacts contained by some higher-level artifact, so that a composite contains components, a component contains services and references, services and reference contain an interface, an interface contains operations.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

Security Identity declares the security identity under which an operation will be executed. There are two mutually exclusive choices to configure the identity, <useCallerIdentity/> and <runAs/>. Both are represented as policy assertions that would be used within policySets created for implementations (i.e. implementation policies). The following policy assertions are defined:

```
<securityIdentity>
  <useCallerIdentity/>
  … or …
  <runAs role="xs:NCName"/>
</securityIdentity>
```

The <useCallerIdentity> policy assertion specifies that an operation will be executed under the invoker's principal. This is the default policy in the absence of a <securityIdentity> element. If the <securityIdentity> policy is <useCallerIdentity> (either explicitly or by default) and the caller did not authenticate, then the principal used is SCA runtime implementation dependent.

The <runAs> policy assertion specifies the name of a security role. Any code so annotated will run with the permissions of that role. How runAs role names are mapped to security principals is implementation dependent.

Authorization declarations describe the role constraints on a composite, component, service or reference. This declaration allows one of three mutually exclusive choices to configure authorization policy, <allow/>, <permitAll/> and <denyAll/>.

```
<authorization>
  <allow roles="listOfNCNames"/>
  … or …
  <permitAll/>
  … or …
  <denyAll/>
</authorization>
```

The <allow> element indicates that access is granted only to principals whose role corresponds to one of the role names listed in the @roles attribute. How role names are mapped to security principals is SCA Runtime implementation dependent (SCA does not define this).

The <permitAll/> and <denyAll/> policy assertions grant or deny access to all principals, respectively.

A policySet MAY contain more than one <authorization> or <securityIdentity> element, but the SCA Runtime MUST raise an error if more than one of either element is in effect at the same time. For example, multiple <authorization> elements can appear on different branches of an intent Map as long as only one of the branches will be in effect at runtime.

7.3.2 Implementation Policy Example

The following is an example implementation, written in Java. The AccountServiceImpl implements the ***AccountService*** interface, which is defined via a Java interface:

Deleted: <allow roles="listOfNCNames">¶
¶
When t

Deleted: is included in a policySet used on a component, then

Deleted: component can only be accessed by

Deleted: <permitAll/>¶
<denyAll/>¶
¶

Deleted: /

Deleted: <runAs role="xs:NCName">¶
¶
The <runAs> policy assertion specifies the name of a security role. Any code so annotated will run with the permissions of that role. How runAs role names are mapped to security principals is implementation dependent.¶

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
package services.account;

@Remotable

public interface AccountService{

        public AccountReport getAccountReport(String customerID);
}
```

The following is a composite that contains an AccountServiceComponent, which should be accessible by anyone with the “customer” role.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
            name="AccountService">
      <component name="AccountServiceComponent">*
            <implementation.java class="services.account.AccountServiceImpl"
                  policySets="acme:allow_customers"/>
      </component>
</composite>
```

Formatted: French (France)

The following is what the policySet definition looks like for this case.

```
<policySet name="allow_customers">
   <authorization>
      <allow roles="customers"/>
   </authorization>
</policySet>
```

Deleted: ¶

Formatted: Indent: First line: 0 cm

7.3.3 SCA Component Container Requirements

SCA component containers MUST support the SCA policy intent model including annotated intent and policySets reference. Additionally SCA component containers MUST satisfy the following security management requirements.

7.3.4 Security Identity Propagation

SCA container MUST establish security identity when authentication is required based on the security intents before executing the SCA component implementation. The security identity under which the operation is executed is determined by the run-as security policy. It is either the user identity who invokes the SCA operation or the identity that represents the run-as security role. When an SCA operation invokes other SCA services, SCA component container must propagate the security identity along with the SCA request.

7.3.5 Security Identity Of Async Callback

In SCA async programming model, the security identity that executes the callback operation by default should be the same as security identity under which the original operation was executed.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

7.3.6 Default Authorization Policy

It may happen that some operations are not assigned any security roles and are not marked as DenyAll or PermitAll. In the SCA deployment process, those operations must be assigned security roles or marked as DenyAll or PermitAll. At runtime time if any operations are not associated with any explicit authorization policy, no access control will be enforced on those operations, i.e., PermitAll.

7.3.7 Default RunAs Policy

Operations will be executed as if <useCallerIdentity/> were specified if no RunAs role policy is explicitly specified.

Deleted: under authentication user identity

8 Reliability Policy

Failures can affect the communication between a service consumer and a service provider. Depending on the characteristics of the binding, these failures could cause messages to be redelivered, delivered in a different order than they were originally sent out or even worse, could cause messages to be lost. Some transports like JMS provide built-in reliability features such as at least once and exactly once message delivery. Other transports like HTTP need to have additional layers built on top of them to provide some of these features.

The events that occur due to failures in communication may affect the outcome of the service invocation. For an implementation of a stock trade service, a message redelivery could result in a new trade. A client (i.e. consumer) of the same service could receive a fault message if trade orders are not delivered to the service implementation in the order they were sent out. In some cases, these failures could have dramatic consequences.

An SCA developer can anticipate some types of failures and work around them in service implementations. For example, the implementation of a stock trade service could be designed to support duplicate message detection. An implementation of a purchase order service could have built in logic that orders the incoming messages. In these cases, service implementations don't need the binding layers to provide these reliability features (e.g. duplicate message detection, message ordering). However, this comes at a cost: extra complexity is built in the service implementation. Along with business logic, the service implementation has additional logic that handles these failures.

Although service implementations can work around some of these types of failures, it is worth noting that is not always possible. A message may be lost or expire even before it is delivered to the service implementation.

Instead of handling some of these issues in the service implementation, a better way of doing it is to use a binding or a protocol that supports reliable messaging. This is better, not just because it simplifies application development, it may also lead to better throughput. For example, there is less need for application-level acknowledgement messages. A binding supports reliable messaging if it provides features such as message delivery guarantees, duplicate message detection and message ordering.

It is very important for the SCA developer to be able to require, at design-time, a binding or protocol that supports reliable messaging. SCA defines a set of policy intents that can be used for specifying reliable messaging Quality of Service requirements. These reliable messaging intents establish a contract between the binding layer and the application layer (i.e. service implementation or the service consumer implementation) (see below).

8.1 Policy Intents

Based on the use-cases described above, we define the following policy intents. It's worth noting that SCA does not provide support for attaching an intent at a message level. Therefore, an intent attached at an operation level applies to all the messages in the operation (e.g. both request and response messages for a request/response message exchange pattern).

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

1) **atLeastOnce** - The binding implementation guarantees that a message that is successfully sent by a service consumer is delivered to the destination (i.e. service implementation). The message could be delivered more than once to the service implementation.

The binding implementation guarantees that a message that is successfully sent by a service implementation is delivered to the destination (i.e. service consumer). The message could be delivered more than once to the service consumer.

2) **atMostOnce** - The binding implementation guarantees that a message that is successfully sent by a service consumer is not delivered more than once to the service implementation. The binding implementation does not guarantee that the message is delivered to the service implementation.

The binding implementation guarantees that a message that is successfully sent by a service implementation is not delivered more than once to the service consumer. The binding implementation does not guarantee that the message is delivered to the service consumer.

3) **ordered** – The binding implementation guarantees that the messages are delivered to the service implementation in the order in which they were sent by the service consumer. This intent does not guarantee that messages that are sent by a service consumer are delivered to the service implementation.

The binding implementation guarantees that the messages are delivered to the service consumer in the order in which they were sent by the service implementation. This intent does not guarantee that messages that are sent by the service implementation are delivered to the service consumer.

4) **exactlyOnce** - The binding implementation guarantees that a message sent by a service consumer is delivered to the service implementation. Also, the binding implementation guarantees that the message is not delivered more than once to the service implementation.

The binding implementation guarantees that a message sent by a service implementation is delivered to the service consumer. Also, the binding implementation guarantees that the message is not delivered more than once to the service consumer.

NOTE: This is a profile intent, which is composed of *atLeastOnce* and *atMostOnce*.

This is the most reliable intent since it guarantees the following:

- message delivery – all the messages sent by a sender are delivered to the service implementation (i.e. Java class, BPEL process, etc.).

- duplicate message detection and elimination – a message sent by a sender is not processed more than once by the service implementation.

How can a binding implementation guarantee that a message that it receives is delivered to the service implementation? One way to do it is by persisting the message and keeping redelivering it until it is processed by the service implementation. That way, if the system crashes after delivery but while processing it, the message will be redelivered on restart and processed again. Since a message could be delivered multiple times to the service implementation, this technique usually requires the service implementation to perform duplicate message detection. However, that is not always possible. Often times service implementations that perform critical operations are designed without having support for duplicate message detection. Therefore, they cannot *process* an incoming message more than once.

Also, consider the scenario where a message is delivered to a service implementation that does not handle duplicates - the system crashes after a message is delivered to the service implementation but before it is completely processed. Should the underlying layer redeliver the message on restart? If it did that, there is a risk that some critical operations (e.g. sending out a JMS message or updating a DB table) will be executed again when the message is processed. On the other hand, if the underlying layer does not redeliver the message, there is a risk that the message is never completely processed.

This issue cannot be safely solved unless all the critical operations performed by the service implementation are running in a transaction. Therefore, *exactlyOnce* cannot be assured without involving the service implementation. In other words, an *exactlyOnce* message delivery does not guarantee *exactlyOnce* message processing unless the service implementation is transactional. It's worth noting that this is a necessary condition but not sufficient. The underlying layer (e.g. binding implementation, container) would have to ensure that a message is not redelivered to the service implementation after the transaction is committed. As an example, a way to ensure it when the binding uses JMS is by making sure the operation that acknowledges the message is executed in the same transaction the service implementation is running in.

8.2 End to end Reliable Messaging

Failures can occur at different points in the message path: in the binding layer on the sender side, in the transport layer or in the binding layer on the receiver side. The SCA service developer doesn't really care where the failure occurs. Whether a message was lost due to a network failure or due to a crash of the machine where the service is deployed, is not that much important. What is important though, is that the contract between the application layer (i.e. service implementation or service consumer) and the binding layer is not violated (e.g. a message that was successfully transmitted by a sender is always delivered to the destination; a message that was successfully transmitted by a sender is not delivered more than once to the service implementation, etc). It is worth noting that the binding layer could throw an exception when a sender (e.g. service consumer, service implementation) sends a message out. This is not considered a successful message transmission.

In order to ensure the semantics of the reliable messaging intents, the entire message path, which is composed of the binding layer on the client side, the transport layer and the binding layer on the service side, must be reliable.

8.3 Intent definitions

```
<?xml version="1.0" encoding="ASCII"?>
<definitions xmlns="http://www.osoa.org/xmlns/sca/1.0" >
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
      <intent name="atLeastOnce"
            appliesTo="sca:binding">
            <description>
                  This intent is used to indicate that a message sent
                  by a client is always delivered to the component.
            </description>
      </intent>

      <intent name="atMostOnce"
            appliesTo="sca:binding">
            <description>
                  This intent is used to indicate that a message that was
                  successfully sent by a client is not delivered more than
                  once to the component.

            </description>
      </intent>

      <intent name="ordered"
            appliesTo="sca:binding">
            <description>
                  This intent is used to indicate that all the messages
                  are delivered to the component in the order they were
                  sent by the client.
            </description>
      </intent>

      <intent name="exactlyOnce"
            appliesTo="sca:binding" requires="atLeastOnce atMostOnce">
            <description>
                  This profile intent is used to indicate that a message
                  sent by a client is always delivered to the component.
                  It also indicates that duplicate messages are not
                  delivered to the component.
            </description>
            </intent>
</definitions>
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

9 Miscellaneous Intents

The following are standard intents that apply to bindings and are not related to either security or reliable messaging:

SOAP – The SOAP intent specifies that the SOAP messaging model should be used for delivering messages. It does not require the use of any specific transport technology for delivering the messages, so for example, this intent can be supported by a binding that sends SOAP messages over HTTP, bare TCP or even JMS. If the intent is required in an unqualified form then any version of SOAP is acceptable. Standard qualified intents also exist for SOAP.1_1 and SOAP.1_2, which specify the use of versions 1.1 or 1.2 of SOAP respectively.

JMS – The JMS intent does not specify a wire-level transport protocol, but instead requires that whatever binding technology is used, the messages should be able to be delivered and received via the JMS API.

NoListener – This intent may only be used within the @requires attribute of a reference. It states that the client is not able to handle new inbound connections. It requires that the binding and callback binding be configured so that any response (or callback) comes either through a back channel of the connection from the client to the server or by having the client poll the server for messages. An example policy assertion that would guarantee this is a WS-Policy assertion that applies to the <binding.ws> binding, which requires the use of WS-Addressing with anonymous responses (e.g. <wsaw:Anonymous>required</wsaw:Anonymous>" – see http://www.w3.org/TR/ws-addr-wsdl/#anonelement).

BP.1_1 – This intent specifies the use of a binding that conforms to the WS-I Basic Profile version 1.1. Any binding or policySet that provides this intent should also provide the SOAP intent. However, the BP intent is not a *profile intent*, since it is not completely satisfied by the lower-level SOAP– there are additional semantic requirements.

Conversational - This intent is meant to be used on an interface, and indicates that the interface is "conversational" as defined in the SCA Assembly Specification [SCA-Assembly].

Deleted: <#>Conformance¶

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

Formatted: Bullets and Numbering

10 Transactions

SCA recognizes that the presence or absence of infrastructure for ACID transaction coordination has a direct effect on how business logic is coded. In the absence of ACID transactions, developers must provide logic that coordinates the outcome, compensates for failures, etc. In the presence of ACID transactions, the underlying infrastructure is responsible for ensuring the ACID nature of all interactions. SCA provides declarative mechanisms for describing the transactional environment required by the business logic. Components that use a synchronous interaction style can be part of a single, distributed ACID transaction within which all transaction resources are coordinated to either atomically commit or rollback. The transmission or receipt of oneway messages can, depending on the transport binding, be coordinated as part of an ACID transaction as illustrated in the *OneWay Invocations* section below. Well-known, higher-level patterns such as store-and-forward queuing can be accomplished by composing transacted one-way messages with reliable-messaging qualities of service.

Deleted: *OneWay Invocations*

Formatted: Font: Italic

This document describes the set of abstract policy intents – both implementation intents and interaction intents – that can be used to describe the requirements on a concrete service component and binding respectively.

Formatted: Bullets and Numbering

10.1 Out of Scope

The following topics are outside the scope of this document:

- The means by which transactions are created, propagated and established as part of an execution context. These are details of the SCA runtime provider and binding provider.
- The means by which a transactional resource manager (RM) is accessed. These include, but are not restricted to:
    - o abstracting an RM as an sca:component
    - o accessing an RM directly in a language-specific and RM-specific fashion
    - o abstracting an RM as an sca:binding

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

10.2 Common Transaction Patterns

In the absence of any transaction policies there is no explicit transactional behavior defined for the SCA service component or the interactions in which it is involved and the transactional behavior is environment-specific. An SCA runtime provider may choose to define an out of band default transactional behavior that applies in the absence of any transaction policies.

Environment-specific default transactional behavior may be overridden by specifying transactional intents described in the document. The most common transaction patterns can be summarized as follows:

Managed, shared global transaction **pattern** – the service always runs in a global transaction context regardless of whether the requester runs under a global transaction. If the requester does run under a transaction, the service runs under the same transaction. Any outbound, synchronous request-response messages will – unless explicitly directed otherwise – propagate the service's transaction context. This pattern offers the highest

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

degree of data integrity by ensuring that any transactional updates are committed atomically

***Managed, local transaction* pattern** – the service always runs in a managed local transaction context regardless of whether the requester runs under a transaction. Any outbound messages will not propagate any transaction context. This pattern is recommended for services that wish the SCA runtime to demarcate any resource manager local transactions and do not require the overhead of atomicity.

The use of transaction policies to specify these patterns is illustrated later in Table 2.

Deleted: Table 2

Deleted: Table 1

Formatted: Bullets and Numbering

10.3 Summary of SCA transaction policies

This specification defines implementation and interaction policies that relate to transactional QoS in components and their interactions. The SCA transaction policies are specified as intents which represent the transaction quality of service behavior offered by specific component implementations or bindings.

SCA transaction policy can be specified either in the SCDL or annotatively in the implementation code. Language-specific annotations are described in the respective language binding specifications, for example the SCA Java Common Annotations and APIs specification [SCA-Java-Annotations].

Deleted: [

Deleted: 3]

Deleted: .

Formatted: Bullets and Numbering

This specification defines the following implementation transaction policies:

- managedTransaction – Describes the service component’s transactional environment.
- transactedOneWay and immediateOneWay – two mutually exclusive intents that describe whether the SCA runtime will process OneWay messages immediately or will enqueue (from a client perspective) and dequeue (from a service perspective) a OneWay message as part of a global transaction.

This specification also defines the following interaction transaction policies:

- propagatesTransaction and suspendsTransaction – two mutually exclusive intents that describe whether the SCA runtime propagates any transaction context to a service or reference on a synchronous invocation. Note that transaction context MUST NOT be propagated on OneWay messages.

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

10.4 Global and local transactions

This specification describes “managed transactions” in terms of either “global” or “local” transactions. The “managed” aspect of managed transactions refers to the transaction environment provided by the SCA runtime for the business component. Business components may interact with other business components and with resource managers. The managed transaction environment defines the transactional context under which such interactions occur.

Formatted: Bullets and Numbering

10.4.1 Global transactions

From an SCA perspective, a global transaction is a unit of work scope within which transactional work is atomic. If multiple transactional resource managers are accessed under a global transaction then the transactional work is coordinated to either atomically commit or rollback regardless using a 2PC protocol. A global transaction can be propagated on synchronous invocations between components – depending on the interaction intents

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

described in this specification - such that multiple, remote service providers can execute distributed requests under the same global transaction.

Formatted: Bullets and Numbering

10.4.2 Local transactions

From a resource manager perspective a resource manager local transaction (RMLT) is simply the absence of a global transaction. But from an SCA persective iti is not enough to simply declare that a piece of business logic runs without a global transaction context. Business logic may need to access transactional resource managers without the presence of a global transaction. The business logic developer still needs to know the expected semantic of making one or more calls to one or more resource managers, and needs to know when and/or how the resource managers local transactions will be committed. The term *local transaction containment* (LTC) is used to describe the SCA environment where there is no global transaction. The boundaries of an LTC are scoped to a remotable service provider method and are not propagated on invocations between components. Unlike the resources in a global transaction, RMLTs coordinated within a LTC may fail independently.

The two most common patterns for components using resource managers outside a global transaction are:

Formatted: Bullets and Numbering

- The application desires each interaction with a resource manager to commit after every interaction. This is the default behavior provided by the **noManagedTransaction** policy (defined below in Transaction implementation policy) in the absence of explicit use of RMLT verbs by the application.
- The application desires each interaction with a resource manager to be part of an extended local transaction that is committed at the end of the method. This behavior is specified by the **managedTransaction.local** policy (defined below in Transaction implementation policy).

While an application may use interfaces provided by the resource adapter to explicitly demarcate resource manager local transactions (RMLT), this is a generally undesirable burden on applications which typically prefer all transaction considerations to be managed by the SCA runtime. In addition, once an application codes to a resource manager local transaction interface, it may never be redeployed with a different transaction environment since local transaction interfaces may not be used in the presence of a global transaction. This specification defines intents to support both these common patterns in order to provide portability for applications regardless of whether they run under a global transaction or not.

Formatted: Bullets and Numbering

10.5 Transaction implementation policy

10.5.1 Managed and non-managed transactions

The mutually exclusive ***managedTransaction*** and ***noManagedTransaction*** intents describe the transactional environment required by a service component or composite.. SCA provides transaction environments that are managed by the SCA runtime in order to remove the burden of coding transaction APIs directly into the business logic. The ***managedTransaction*** and ***noManagedTransaction*** intents can be attached to the sca:composite or sca:componentType XML elements.

The mutually exclusive ***managedTransaction*** and ***noManagedTransaction*** intents are defined as follows:

Formatted: Bullets and Numbering

- **managedTransaction** – There must be a managed transaction environment in order to run this component. The specific type of managedTransaction required is not constrained. The valid qualifiers for this intent are mutually exclusive and are defined as:

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

- **managedTransaction.global** – There must be an atomic transaction in order to run this component. The SCA runtime must ensure that a global transaction is present before dispatching any method on the component. The SCA runtime uses any transaction propagated from the client or else begins and completes a new transaction. See the ***propagatesTransaction*** intent below for more details.
- **managedTransaction.local** – The component cannot tolerate running as part of a global transaction, and will therefore run within a local transaction containment (LTC) that is started and ended by the SCA runtime. Any global transaction context that is propagated to the hosting SCA runtime must not be visible to the target component. Any interaction under this policy with a resource manager is performed in an extended resource manager local transaction (RMLT). Upon successful completion of the invoked service method, any RMLTs are implicitly requested to commit by the SCA runtime. Note that, unlike the resources in a global transaction, RMLTs so coordinated in a LTC may fail independently. If the invoked service method completes with a non-business exception then any RMLTs are implicitly rolled back by the SCA runtime. In this context a business exception is any exception that is declared on the component interface and is therefore anticipated by the component implementation. The manner in which exceptions are declared on component interfaces is specific to the interface type– for example Java interface types declare Java exceptions, WSDL interface types define wsdl:faults. Local transactions cannot be propagated outbound across remotable interfaces.
- **noManagedTransaction** – The component runs without a managed transaction, under neither a global transaction nor an LTC. A transaction that is propagated to the hosting SCA runtime MUST NOT be joined by the hosting runtime on behalf of this component. When interacting with a resource manager under this policy, the application (and not the SCA runtime) is responsible for controlling any resource manager local transaction boundaries, using resource-provider specific interfaces (for example a Java implementation accessing a JDBC provider must choose whether a Connection should be set to autoCommit(true) or else must call the Connection commit or rollback method). SCA defines no APIs for interacting with resource managers.
- **(absent)** – The absence of an implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern but is not required to do so.

10.5.2 OneWay Invocations

When a client uses a reference and sends a OneWay message then any client transaction context is not propagated. However, the OneWay invocation on the reference may, itself, be *transacted*. Similarly, from a service perspective, any received OneWay message cannot propagate a transaction context but the delivery of the OneWay message may be *transacted*. A *transacted* OneWay message is a one-way message that - because of the capability of the service or reference binding - can be enqueued (from a client perspective) or dequeued (from a service perspective) as part of a global transaction. SCA defines two mutually exclusive implementation intents, **transactedOneWay** and **immediateOneWay**, that determine whether OneWay messages must be transacted or delivered immediately. Either of these intents may be attached to the sca:service or sca:reference elements but a deployment error will occur if both intents are attached to the same element. Either of these

Formatted: Bullets and Numbering

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

intents may be attached to the sca:component element, indicating that the intent applies to any service or reference element children. The intents are defined as follows:

- **transactedOneWay** – When applied to a reference indicates that any OneWay invocation messages MUST be transacted as part of a client global transaction. If the client is not configured to run under a global transaction or if the binding does not support transactional message sending, then a deployment error occurs. When applied to a service indicates that any OneWay invocation message MUST be received from the transport binding in a transacted fashion, under the target service's global transaction. The receipt of the message from the binding is not committed until the service transaction commits; if the service transaction is rolled back the the message remains available for receipt under a different service transaction. If the service is not configured to run under a global transaction or if the binding does not support transactional message receipt, then a deployment error occurs.

- **immediateOneWay** – When applied to a reference indicates that any OneWay invocation messages is sent immediately regardless of any client transaction. When applied to a service indicates that any OneWay invocation is received immediately regardless of any target service transaction. The outcome of any transaction under which an immediateOneWay message is processed has no effect on the processing (sending or receipt) of that message.

Formatted: Bullets and Numbering

The absence of either intent leads to runtime-specific behavior. The SCA runtime may send or receive a OneWay message immediately or as part of any sender/receiver transaction. The results of combining this intent and the ***managedTransaction*** implementation policy of the component sending or receiving the transacted OneWay invocation are summarized below in Table 1.

Deleted: Table 1

Deleted: Table 1

transacted/immediate intent	**managedTransaction (client or service implementation intent)**	**Results**
transactedOneWay	managedTransaction.global	OneWay interaction (either client message enqueue or target service dequeue) is committed as part of the global transaction.
transactedOneWay	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
immediateOneWay	Any value of managedTransaction	The OneWay interaction occurs immediately and is not transacted.
<absent>	Any value of managedTransaction	Runtime-specific behavior. The SCA runtime may send or receive a OneWay message immediately or as part of any sender/receiver transaction.

Table 1 Transacted OneWay interaction intent

[**Note:** The SCA Assembly specification [SCA-Assembly] will need to specify the semantics of oneway sends. For example, can a oneway send result in a synchronous Runtime exception related to protocol error that occurs during the send?]

Deleted: 1]

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

10.6 Transaction interaction policies

Formatted: Bullets and Numbering

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may be attached either to an interface (e.g. Java annotation or WSDL attribute) or explicitly to an sca:service and sca:reference XML element to describe how any client transaction context will be made available and used by the target service component. Section 10.6.1 considers how these intents apply to service elements and Section 10.6.2 considers how these intents apply to reference elements.

Deleted: 10.6.1

Deleted: 1.5.1

Deleted: 10.6.2

Deleted: 1.5.2

10.6.1 Handling Inbound Transaction Context

Formatted: Bullets and Numbering

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may be attached to an sca:service XML element to describe how a propagated transaction context should be handled by the SCA runtime, prior to dispatching a service component. If the service requester is running within a transaction and the service interaction policy is to propagate that transaction, then the primary business effects of the provider's operation are coordinated as part of the client's transaction – if the client rolls back its transaction, then work associated with the provider's operation will also be rolled back. This allows clients to know that no compensation business logic is necessary since transaction rollback can be used.

These intents specify a contract that MUST be implemented by the SCA runtime. This aspect of a service component is most likely captured during application design. Either the ***propagatesTransaction*** or ***suspendsTransaction*** intent can be attached to sca:service elements and their children but a deployment error will occur if both intents are specified.

The intents are defined as follows:

- **propagatesTransaction** – The SCA runtime MUST ensure that the service is dispatched under any propagated (client) transaction. Use of the ***propagatesTransaction*** intent implies that the service binding MUST be capable of receiving a transaction context and that a service with this intent specified will always join a propagated transaction, if present. However, it is important to understand that some binding/policySet combinations that provide this intent for a service will *require* the client to propagate a transaction context. In SCA terms, for a reference wired to such a service, this implies that the reference must use either the ***propagatesTransaction*** intent or a binding/policySet combination that does propagate a transaction. If, on the other hand, the service does not *require* the client to provide a transaction (even though it has the *capability* of joining the client's transaction), then some care is needed in the configuration of the service. One approach to consider in this case is to use two distinct bindings on the service, one that uses the ***propagatesTransaction*** intent and one that does not - clients that do not propagate a transaction would then wire to the service using the binding without the ***propagatesTransaction*** intent specified.

Formatted: Bullets and Numbering

- **suspendsTransaction** – The SCA runtime MUST ensure that the service is NOT dispatched under any propagated (client) transaction.

The absence of either interaction intent leads to runtime-specific behavior; the client is unable to determine from transaction intents whether its transaction will be joined.

Transaction context is never propagated on OneWay messages. The SCA runtime ignores ***propagatesTransaction*** for OneWay methods.

These intents are independent from the implementation's ***managedTransaction*** intent and provides no information about the implementation's transaction environment.

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

The combination of these service interaction policies and the ***managedTransaction*** implementation policy of the containing component completely describes the transactional behavior of an invoked service, as summarized in Table 2.

Deleted: Table 2

Deleted: Table 2

service interaction intent	**managedTransaction (component implementation intent)**	**Results**
propagatesTransaction	managedTransaction.global	Component runs in propagated transaction if present, otherwise a new global transaction. This combination is used for the **managed, shared global transaction** pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
suspendsTransaction	managedTransaction.global	Component runs in a new global transaction
suspendsTransaction	managedTransaction.local	Component runs in a managed local transaction containment. This combination is used for the **managed, local transaction** pattern described in Common Transaction Patterns. This is the default behavior for a runtime that does not support global transactions.
suspendsTransaction	noManagedTransaction	Component is responsible for managing its own local transactional resources.

Table 2 Combining service transaction intents

Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern.

In the case where the **propagatesTransaction** intent conflicts with the component's ***managedTransaction.local*** intent, an appropriate error message must be issued at deployment. SCA tooling may also detect the error earlier in the development process.

Formatted: Bullets and Numbering

10.6.2 Handling Outbound Transaction Context

The mutually exclusive ***propagatesTransaction*** and ***suspendsTransaction*** intents may also be attached to an sca:reference XML element to describe whether any client transaction context should be propagated to a target service when a synchronous interaction occurs through the reference. These intents specify a contract that MUST be implemented by the SCA runtime. This aspect of a service component is most likely captured during application design. Either the ***propagatesTransaction*** or ***suspendsTransaction*** intent can be attached to sca:service elements and their children but a deployment error will occur if both

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

intents are specified. The intents are defined as defined in Section 10.6.1. When used as a reference interaction intent, the meaning of the qualifiers is as follows:

Deleted: 10.6.1

Deleted: 1.5.1

Formatted: Bullets and Numbering

- **propagatesTransaction** – any transaction context under which the client runs will be propagated when the reference is used for a request-response interaction. To satisfy policy framework rules, the reference binding MUST be capable of propagating a transaction context. The reference should be wired to a service that can join the client's transaction. For example, any service with an intent that @requires ***propagatesTransaction*** can always join a client's transaction. The reference consumer can then be designed to rely on the work of the target service being included in the caller's transaction.
- **suspendsTransaction** – any transaction context under which the client runs will not be propagated when the reference is used. The reference consumer can use this intent to ensure that the work of the target service is not included in the caller's transaction. .

The absence of either interaction intent leads to runtime-specific behavior. The SCA runtime may or may not propagate any client transaction context to the referenced service, depending on the SCA runtime capability.

These intents are independent from the client's ***managedTransaction*** implementation intent. The combination of the interaction intent of a reference and the ***managedTransaction*** implementation policy of the containing component completely describes the transactional behavior of a client's invocation of a service. Table 3 summarizes the results of the combination of either of these interaction intents with the ***managedTransaction*** implementation policy of the containing component.

Formatted: English (U.S.)

Deleted: Table 3

Deleted: Table 3

reference interaction intent	**managedTransaction (client implementation intent)**	**Results**
propagatesTransaction	managedTransaction.global	Target service runs in the client's transaction. This combination is used for the **managed, shared global transaction** pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	This is an "incompatible deployment" Error
suspendsTransaction	Any value of managedTransaction	The target service will not run under the same transaction as any client transaction. This combination is used for the **managed, local transaction** pattern described in Common Transaction Patterns.

Table 3 Transaction propagation reference intents

Formatted: English (U.S.)

Field Code Changed

Formatted: English (U.S.)

Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A runtime that supports global transaction coordination may choose to provide a default behavior that is the managed, shared global transaction pattern.
In the case where the **propagatesTransaction** reference intent conflicts with the using component's ***managedTransaction.local*** intent, an appropriate error message must be

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

issued at deployment. SCA tooling may also detect the error earlier in the development process.

Table 4 shows the valid combination of interaction and implementation intents on the client and service that result in a single global transaction being used when a client invokes a service through a reference.

Deleted: Table 4

Deleted: Table 4

managedTransaction (client implementation intent)	**reference interaction intent**	**service interaction intent**	**managedTransaction (service implementation intent)**
managedTransaction.global	propagatesTransaction	propagatesTransaction	managedTransaction.global

Table 4 Intents for end-to-end transaction propagation

Transaction context is never propagated on OneWay messages. The SCA runtime ignores ***propagatesTransaction*** for OneWay methods.

Formatted: Bullets and Numbering

10.6.3 Web services binding for propagatesTransaction policy

This specification defines the XML syntax for a policySet that provides the ***propagatesTransaction*** intent and applies to a Web service binding (binding.ws). When used on a service, this policySet requires the client to send a transaction context. This intent is provided on Web service interactions using the mechanisms described in the Web Services Atomic Transaction [WS-AtomicTransaction] specification. As such the policy is described using the wsat:ATAssertion defined by the WS-AtomicTransaction specification as follows:

```
<policySet name="JoinsTransactionWS" provides="sca:propagatesTransaction"
                                     appliesTo="sca:binding.ws">
   <wsp:Policy>
     <wsat:ATAssertion
          xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"/>
   </wsp:Policy>
</policySet>
```

Formatted: Bullets and Numbering

10.7 Example

The following example shows some of the transaction polices in use for an implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<componentType xmlns:sca=" http://www.osoa.org/xmlns/sca/1.0"
      requires="managedTransaction.global">

      <implementation.java class="com.acme.TransactionalComponent1"
            requires="managedTransaction.global">

      <service name="Service1" requires="propagatesTransaction">
            <interface />
      </service>
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
      <reference name="Reference1" requires="transactedOneWay">
            <interface />
      <reference>

      <implementation/>

</componentType>
```

10.8 Intent Definitions

The SCA Policy Framework specification defines an XML schema for defining abstract intents. The following XML snippet shows the intent definitions for the transaction policy domain.

10.8.1 Intent.xml snippet

```
  <intent name="managedTransaction" constrains="sca:implementation">
      <description>
            Used to indicate the transaction environment desired by a 
      component 
            implementation.
      </description>
  </intent>

  <intent name="managedTransaction.global" constrains="sca:implementation">
      <description>
            Used to indicate that a component implementation requires a 
      managed 
            global transaction.
      </description>
  </intent>

  <intent name="managedTransaction.local" constrains="sca:implementation">
      <description>
            Used to indicate that a component implementation requires a 
      managed local 
            transaction.
      </description>
```

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Deleted: <?xml version="1.0" encoding="ASCII"?>

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

Formatted: French (France)

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
  </intent>

  <intent name="noManagedTransaction" constrains="sca:implementation">
        <description>
              Used to indicate that a component implementation will manage its 
        own 
              transaction resources.
        </description>
  </intent>


  <intent name="propagatesTransaction" constrains="sca:binding">
        <description>
              Used to indicate that a reference will propagate any client 
        transaction 
              or that a service will be dispatched under any received 
        transaction.
        </description>
  </intent>

  <intent name="suspendsTransaction" constrains="sca:binding">
        <description>
              Used to indicate that a reference will not propagate any client 
              transaction or that a service will not be dispatched under any 
        received 
              transaction.
        </description>
  </intent>


  <intent name="transactedOneWay" constrains="sca:binding">
        <description>
              Used to indicate that the component requires the SCA runtime to 
              transact OneWay send of messages as part of any client global 
              transaction or 
              to transact oneWay message receipt as part of any service global 
              transaction.
        </description>
  </intent>

  <intent name="immediateOneWay" constrains="sca:binding">
        <description>
              Used to indicate that the component requires the SCA runtime to 
              process the sending or receiving of OneWay messages immediately,
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
                    regardless of any transaction under which the sending/receiving
                    component runs.
        </description>
   </intent>
```

Deleted: </intents>

Deleted: <#>References¶
¶
[1] SCA Assembly Model Specification v1.0¶
http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf¶
¶
[2] SCA Policy Framework v1.0¶
http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf¶
¶
[3] SCA Java Common Annotations and APIs ¶
http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf¶
¶
[4] Web Services Atomic Transaction (WS-AtomicTransaction) .
http://docs.oasis-open.org/ws-tx/wsat/2006/06.¶

Formatted: Bullets and Numbering

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

11 Conformance

A. Schemas

A.1 XML Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright SCA Collaboration 2006, 2007 -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.osoa.org/xmlns/sca/1.0"
        xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
        xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
        elementFormDefault="qualified">

    <include schemaLocation="sca-core.xsd"/>
<import namespace="http://www.w3.org/ns/ws-policy
    "
schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd
        "/>


    <element name="intent" type="sca:Intent"/>
    <complexType name="Intent">
        <sequence>
        <element name="description" type="string" minOccurs="0"
        maxOccurs="1" />
        <element name="qualifier" type="sca:IntentQualifier"
        minOccurs="0" maxOccurs="unbounded" />
        </sequence>
        <any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded"/>
        <attribute name="name" type="NCName" use="required"/>
        <attribute name="constrains" type="sca:listOfQNames"
        use="optional"/>
        <attribute name="requires" type="sca:listOfQNames"
        use="optional"/>
        <attribute name="excludes" type="sca:listOfQNames"
        use="optional"/>
        <attribute name="mutuallyExclusive" type="boolean" use="optional"
        default=”false”/>

        <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <complexType name=”IntentQualifier”>
        <element name="description" type="string"   minOccurs="0"
        maxOccurs="1" />
          <attribute name="name" type="NCName"    use="required"/>
```

Formatted: Bullets and Numbering

Formatted: English (U.S.)

Formatted: French (France)

Deleted: http://schemas.xmlsoap.org/ws/2004/09/policy

Deleted: http://schemas.xmlsoap.org/ws/2004/09/ws-policy.xsd

Deleted: <element name="intent" type="sca:Intent"/>¶ <complexType name="Intent">¶ <sequence>¶ <element name="description" type="string" minOccurs="0"¶ maxOccurs="1" />¶ <any namespace="##other" processContents="lax"¶ minOccurs="0" maxOccurs="unbounded"/>¶ </sequence>¶ <attribute name="name" type="NCName" use="required"/>¶ <attribute name="constrains" type="sca:listOfQNames" use="required"/>¶ <attribute name="requires" type="sca:listOfQNames" use="optional"/>¶ <anyAttribute namespace="##any" processContents="lax"/>¶ </complexType>¶

Formatted: English (U.S.)

Formatted: English (U.S.)

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
          <attribute name="default" type="boolean" use="optional" default
        ="false"
    </complexType>
```

Constraint: If the intent definition contains one or more <qualifier> children, one and only one of the qualifier children MUST have the value of the default attribute set to 'true'. The values of the name attributes of the qualifiers within a single intent definition MUST be unique.

```
    <element name="policySet" type="sca:PolicySet"/>
    <complexType name="PolicySet">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element name="policySetReference"
                type="sca:PolicySetReference"/>
            <element name="intentMap" type="sca:IntentMap"/>

            <any namespace="##other" processContents="lax"/>
          </choice>
          <attribute name="name" type="NCName" use="required"/>
          <attribute name="provides" type="sca:listOfQNames"/>
          <attribute name="appliesTo" type="string" use="required"/>
          <attribute name="attachTo" type="string" use="optional"/>
          <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <element name="policyAttachment" type="sca:PolicyAttachment"/>
    <complexType name="PolicySet">
          <any namespace="##other" processContents="lax" minOccurs="0"
             maxOccurs="unbounded"/>
          <attribute name="policySet" type="QName"/>
          <attribute name="attachTo" type="string" use="required"/>
          <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <complexType name="PolicySetReference">
          <attribute name="name" type="QName" use="required"/>
          <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <complexType name="IntentMap">
          <choice minOccurs="1" maxOccurs="unbounded">
          <element name="qualifier" type="sca:Qualifier"/>
          <any namespace="##other" processContents="lax"/>
          </choice>
          <attribute name="provides" type="QName" use="required"/>

          <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <complexType name="Qualifier">
          <choice minOccurs="1" maxOccurs="unbounded">
          <element name="intentMap" type="sca:IntentMap"/>

          <any namespace="##other" processContents="lax"/>
```

Deleted: <element ref="wsp:PolicyAttachment"/>¶ <element ref="wsp:Policy"/>¶ <element ref="wsp:PolicyReference"/>

Formatted: Indent: Left: 0 cm

Formatted: Indent: First line: 1.27 cm

Formatted: Font: Courier New, Font color: Custom Color(RGB(0,129,129))

Formatted: English (U.S.)

Deleted: <attribute name="default" type="string" use="optional"/>

Deleted: <element ref="wsp:PolicyAttachment"/>

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

```
            </choice>
            <attribute name="name" type="string" use="required"/>
            <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <element name="securityIdentity" type="sca:SecurityIdentity"/>
    <complexType name="SecurityIdentity">
        <choice>
            <element name="useCallerIdentity" 
type="sca:UseCallerIdentity"/>
            <element name="runAs" type="sca:RunAs"/>
        </choice>
    </complexType>

    <complexType name="UseCallerIdentity"/>
    <complexType name="RunAs">
        <attribute name="role" type="string" use="required"/>
    </complexType>


    <element name="authorization" type="sca:Authorization"/>
    <complexType name="Authorization">
        <choice>
            <element name="allow" type="sca:Allow"/>
            <element name="permitAll" type="sca:PermitAll"/>
            <element name="denyAll" type="sca:DenyAll"/>
        </choice>
    </complexType>

    <complexType name="Allow">
        <attribute name="roles" type="string" use="required"/>
    </complexType>

    <complexType name="PermitAll"/>

    <complexType name="DenyAll"/>

    <simpleType name="listOfNCNames">
    <list itemType="NCName"/>
    </simpleType>

</schema>
```

Deleted: `<element name="allow" type="sca:Allow"/>¶`

Deleted: `<element name="permitAll" type="sca:PermitAll"/>¶`

Deleted: `<element name="denyAll" type="sca:DenyAll"/>¶`

Deleted:
```
<element name="runAs" type="sca:RunAs"/>¶
<complexType name="RunAs">¶
<attribute name="role" type="string" use="required"/>¶
</complexType>¶
¶
```

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

B. Acknowledgements

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

2750 # C. Non-Normative Text

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

D. Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
2	Nov 2, 2007	David Booz	Inclusion of OSOA errata and Issue 8
3	Nov 5, 2007	David Booz	Applied resolution of Issue 7, to Section 4.1 and 4.10. Fixed misc. typos/grammatical items.
4	Mar 10, 2008	David Booz	Inclusion of OSOA Transaction specification as Chapter 11. There are no textual changes other than formatting.
5	Apr 28 2008	Ashok Malhotra	Added resolutions to issues 17, 18, 24, 29, 37, 39 and 40,
6	July 7 2008	Mike Edwards	Added resolution for Issue 38
7	Aug 15 2008	David Booz	Applied Issue 26, 27
7 + Issue 15	Sept 8 2008	Mike Edwards	Proposal for Issue 15

Deleted: il

Deleted: ,

Deleted: ust

Deleted: 0

Deleted: 7

Deleted: 69

Deleted: 71

Deleted: 71

Page 12: [1] Deleted **asmalhot** **3/14/2008 2:10:00 PM**

Where:

@name attribute defines the name of the intent

@constrains attribute (optional) specifies the SCA constructs (SCA binding or implementation) that this intent is meant to configure. If a value is not specified, it is
assumed that this intent is a qualified intent and inherits its constraint list from the qualifiable intent it is qualifying (see below). This attribute does not define the valid attach points of the intent.

Note that the "constrains" attribute may name an abstract element type, such as sca:binding in our running example. This means that it will match against any binding used within a SCDL file. A SCDL element may match @constrains if its type is in a substitution group.

Page 12: [2] Deleted **asmalhot** **3/14/2008 2:10:00 PM**

@requires attribute (optional) defines the set of all intents that the referring intent requires. In essence, the referring intent requires all the intents named to be satisfied. This attribute is used to compose an intent from a set of other intents. This use is further described in Section 3.2 below.

The **confidentiality** intent may be defined as:

```
<intent name="confidentiality" constrains="sca:binding">
     <description>
           Communication through this binding must prevent
           unauthorized users from reading the messages.
     </description>
</intent>
```

Page 13: [3] Deleted **asmalhot** **3/14/2008 2:15:00 PM**

Because qualified intents include the name of the qualifiable intent, the qualifiable intent definition does not need to list its valid qualifiers. The set of all qualified intents defined for that qualifiable intent determines the list of valid qualifiers. This is illustrated by adding two additional intents to our example called **confidentiality.transport** and **confidentiality.message**. Note that the original intent definition or **confidentiality** does not change.

Page 13: [4] Deleted **asmalhot** **3/14/2008 2:15:00 PM**

Further, the @constrains attribute of a qualified intent is unnecessary because qualified intents inherit the @constrains attribute from the qualifiable intent. It is an error to specify @constrains in the definition of a qualified intent. The following are definitions of the transport and message qualifiers of the **confidentiality** intent.

Page 13: [5] Deleted **asmalhot** **3/14/2008 2:16:00 PM**

```
<intent name="confidentiality.transport" />
<intent name="confidentiality.message" />
```

Page 13: [6] Formatted **Mike Edwards** **9/8/2008 12:16:00 PM**

Font: Courier New, Font color: Blue

Page 13: [7] Formatted **Mike Edwards** **9/8/2008 12:17:00 PM**

Font: Courier New, Font color: Blue, French (France)

Page 13: [8] Formatted **Mike Edwards** **9/8/2008 12:16:00 PM**

Font: Courier New, Font color: Blue

Page 13: [9] Change **asmalhot** **3/14/2008 2:16:00 PM**

Formatted Bullets and Numbering

Page 13: [10] Change **asmalhot** **3/14/2008 2:16:00 PM**

Formatted Bullets and Numbering

Page 13: [11] Formatted **Mike Edwards** **9/8/2008 12:14:00 PM**

Indent: Left: 0.63 cm

Page 13: [12] Change **asmalhot** **3/14/2008 2:16:00 PM**

Formatted Bullets and Numbering

Page 13: [13] Formatted **Mike Edwards** **9/8/2008 3:40:00 PM**

Bulleted + Level: 1 + Aligned at: 0.63 cm + Tab after: 1.27 cm + Indent at: 1.27 cm

Page 30: [14] Deleted **Mike Edwards** **7/7/2008 1:14:00 PM**

Stating intents with the @requires attribute of an element means that those intents are additionally required by every relevant element descendent. For example, specifying
`requires=”confidentiality”` on a `<composite>` element is the equivalent to adding the same intent to the @requires list of every service and reference that is contained within that composite, including the services and references inside components. *Therefore, the computed intents that apply to a specific element is the union of all intents that are present in the @requires attribute values of its ancestors that apply to the specific type of element.* This is equivalent to listing an intent in the @requires list of all of descendent elements that match one of the xs:QName values of the @constrains attribute of an intent, taking into account the presence of substitution groups.

When computing the intents that apply to a particular element, the @constrains attribute of each relevant intent is checked against the element. If the intent in question does not apply to that element it is simply discarded.

Page 30: [15] Deleted **Mike Edwards** **7/7/2008 1:16:00 PM**

are specified with @requires attribute values of

Page 30: [16] Deleted **Mike Edwards** **7/7/2008 1:16:00 PM**

during development

Page 30: [17] Deleted **Mike Edwards** **7/7/2008 1:18:00 PM**

The intents specified for an element are also used to determine a specific mapping/choice other than the default, should the selected policySet contain intentMaps. The developer in this case is not choosing policySets that apply as they will be determined, if possible, during a later deployment step.

Page 30: [18] Deleted **Mike Edwards** **7/7/2008 1:19:00 PM**

Both qualified intents and their respective qualifiable intents, and profile intents, can be specified as values of a @requires attribute. In considering the set of intents that are computed for a specific element, however, the following rules must be observed.

> When the computed values of a @requires attribute includes both the qualified and unqualified form of a qualifiable intent, the unqualified form is ignored. For example, assume that the **confidentiality** intent uses **confidentiality.transport** as its default when specified as part of a PolicySet.

Page 30: [19] Deleted	**Mike Edwards**	**9/8/2008 1:52:00 PM**

. When the intent is matched with the appropriate policySet (by the assembler or deployer) to generate concrete policies that satisfies the intents, t

Page 30: [20] Deleted	**Mike Edwards**	**7/7/2008 1:22:00 PM**

by the PolicySet that is used at deployment time

Page 30: [21] Deleted	**Mike Edwards**	**7/7/2008 1:23:00 PM**

During policySet selection, it is only possible to override a qualifiable intent that doesn’t specify a qualifier. Thus, multiple qualifiers MUST NOT be specified for the same qualifiable intent as part of a computed intent set.

Page 30: [22] Deleted	**Mike Edwards**	**7/7/2008 1:24:00 PM**

> If a component type includes a list of required intents on a service or reference, it is *not* possible for a component that uses that component type to remove any of those required intents. However, if any of the intents are qualifiable intents, the component MAY specify a qualifier for that intent.

Page 30: [23] Change	**Dave Booz**	**3/5/2008 7:18:00 AM**

Formatted Bullets and Numbering