



---

# SCA Policy Framework Version 1.1

## Committee Draft 03/Public Review 02

5 May 2010

### Specification URIs:

#### This Version:

<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd03.html>  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd03.doc>  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd03.pdf> (Authoritative)

#### Previous Version:

<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.html>  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.doc>  
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd02.pdf> (Authoritative)

#### 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 Service Component Architecture / Policy (SCA-Policy) TC

### Chair(s):

David Booz, IBM <[booz@us.ibm.com](mailto:booz@us.ibm.com)>  
Ashok Malhotra, Oracle <[ashok.malhotra@oracle.com](mailto:ashok.malhotra@oracle.com)>

### Editor(s):

David Booz, IBM <[booz@us.ibm.com](mailto:booz@us.ibm.com)>  
Michael J. Edwards, IBM <[mike\\_edwards@uk.ibm.com](mailto:mike_edwards@uk.ibm.com)>  
Ashok Malhotra, Oracle <[ashok.malhotra@oracle.com](mailto:ashok.malhotra@oracle.com)>

### Related work:

This specification replaces or supercedes:

- [SCA Policy Framework Specification Version 1.00 March 07, 2007](#)

This specification is related to:

OASIS Committee Draft 05, "SCA Assembly Model Specification Version 1.1", January 2010.  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.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/200912](http://docs.oasis-open.org/ns/opencsa/sca/200912). 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/jpr.php>).

---

## Notices

Copyright © OASIS® 2005, 2010. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS" 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.....	7
1.1	Terminology.....	7
1.2	XML Namespaces.....	7
1.3	Normative References.....	7
1.4	Naming Conventions.....	8
2	Overview.....	9
2.1	Policies and PolicySets.....	9
2.2	Intents describe the requirements of Components, Services and References.....	9
2.3	Determining which policies apply to a particular wire.....	10
3	Framework Model.....	11
3.1	Intents.....	11
3.2	Interaction Intents and Implementation Intents.....	13
3.3	Profile Intents.....	14
3.4	PolicySets.....	14
3.4.1	IntentMaps.....	16
3.4.2	Direct Inclusion of Policies within PolicySets.....	18
3.4.3	Policy Set References.....	18
4	Attaching Intents and PolicySets to SCA Constructs.....	21
4.1	Attachment Rules – Intents.....	21
4.2	Direct Attachment of Intents.....	21
4.3	External Attachment of Intents and PolicySets.....	22
4.4	Attachment Rules - PolicySets.....	22
4.5	Direct Attachment of PolicySets.....	22
4.6	External Attachment of PolicySets.....	23
4.6.1	Cases Where Multiple PolicySets are attached to a Single Artifact.....	24
4.7	Attaching intents to SCA elements.....	24
4.7.1	Implementation Hierarchy of an Element.....	24
4.7.2	Structural Hierarchy of an Element.....	24
4.7.3	Combining Implementation and Structural Policy Data.....	25
4.7.4	Examples.....	26
4.8	Usage of Intent and Policy Set Attachment together.....	27
4.9	Intents and PolicySets on Implementations and Component Types.....	27
4.10	Intents on Interfaces.....	27
4.11	BindingTypes and Related Intents.....	28
4.12	Treatment of Components with Internal Wiring.....	29
4.12.1	Determining Wire Validity and Configuration.....	30
4.13	Preparing Services and References for External Connection.....	30
4.14	Deployment.....	30
4.14.1	Redeployment of Intents and PolicySets.....	31
4.15	Matching Intents and PolicySets.....	32
5	Implementation Policies.....	34

5.1	Natively Supported Intents.....	35
5.2	Writing PolicySets for Implementation Policies .....	35
5.2.1	Non WS-Policy Examples .....	36
6	Roles and Responsibilities .....	37
6.1	Policy Administrator .....	37
6.2	Developer.....	37
6.3	Assembler.....	37
6.4	Deployer.....	38
7	Security Policy .....	39
7.1	Security Policy Intents .....	39
7.2	Interaction Security Policy .....	39
7.2.1	Qualifiers .....	40
7.3	Implementation Security Policy Intent .....	40
8	Reliability Policy.....	41
8.1	Reliability Policy Intents.....	41
8.2	End-to-end Reliable Messaging .....	43
9	Transactions.....	44
9.1	Out of Scope.....	44
9.2	Common Transaction Patterns.....	44
9.3	Summary of SCA Transaction Policies.....	45
9.4	Global and local transactions.....	45
9.4.1	Global transactions.....	45
9.4.2	Local transactions .....	45
9.5	Transaction implementation policy .....	46
9.5.1	Managed and non-managed transactions.....	46
9.5.2	OneWay Invocations .....	47
9.5.3	Asynchronous Implementations .....	48
9.6	Transaction interaction policies .....	48
9.6.1	Handling Inbound Transaction Context.....	48
9.6.2	Handling Outbound Transaction Context.....	50
9.6.3	Combining implementation and interaction intents .....	52
9.6.4	Interaction intents with asynchronous implementations.....	52
9.6.5	Web Services Binding for propagatesTransaction policy.....	52
10	Miscellaneous Intents.....	53
11	Conformance .....	54
A	Defining the Deployed Composites Infoset .....	55
A.1	XPath Functions for the @attachTo Attribute .....	57
A.1.1	Interface Related Functions .....	57
A.1.2	Intent Based Functions.....	58
A.1.3	URI Based Function .....	59
B	Schemas.....	60
B.1	sca-policy.xsd .....	60
C	XML Files.....	63
C.1	Intent Definitions .....	63
D	Conformance .....	68

D.1	Conformance Targets .....	68
D.2	Conformance Items .....	68
E	Acknowledgements .....	75
F	Revision History.....	76

# 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	<a href="http://docs.oasis-open.org/ns/opencsa/sca/200912">docs.oasis-open.org/ns/opencsa/sca/200912</a> This is assumed to be the default namespace in this specification. xs:QNames that appear without a prefix are from the SCA namespace.	[SCA-Assembly]
acme	Some namespace; a generic prefix	
wsp	<a href="http://www.w3.org/2006/07/ws-policy">http://www.w3.org/2006/07/ws-policy</a>	[WS-Policy]
xs	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>	[XML Schema Datatypes]

Table 1-1: XML Namespaces and Prefixes

## 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.
- [SCA-Assembly] OASIS Committee Draft 05, “Service Component Architecture Assembly Model Specification Version 1.1”, January 2010.  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd05.pdf>
- [SCA-Java-Annotations] OASIS Committee Draft 04, “SCA Java Common Annotations and APIs Specification Version 1.1”, February 2010.

28		<a href="http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf">http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-cd04.pdf</a>
29	<b>[SCA-WebServicesBinding]</b>	
30		OASIS Committee Draft 03, "SCA Web Services Binding Specification Version
31		1.1", July 2009.
32		<a href="http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd03.pdf">http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-</a>
33		<a href="http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd03.pdf">cd03.pdf</a>
34	<b>[WSDL]</b>	Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
35		– Appendix <a href="http://www.w3.org/TR/2006/CR-wsdl20-20060327/">http://www.w3.org/TR/2006/CR-wsdl20-20060327/</a>
36	<b>[WS-AtomicTransaction]</b>	
37		OASIS Standard, "Web Services Atomic Transaction Version 1.2", February
38		2009.
39		<a href="http://docs.oasis-open.org/ws-tx/wsdl2006/06">http://docs.oasis-open.org/ws-tx/wsdl2006/06</a> .
40	<b>[WSDL-Ids]</b>	SCA WSDL 1.1 Element Identifiers – forthcoming W3C Note
41		<a href="http://dev.w3.org/cvsweb/~checkout~/2006/ws/policy/wsdl11elementidentifiers.html">http://dev.w3.org/cvsweb/~checkout~/2006/ws/policy/wsdl11elementidentifiers.ht</a>
42		<a href="http://dev.w3.org/cvsweb/~checkout~/2006/ws/policy/wsdl11elementidentifiers.html">ml</a>
43	<b>[WS-Policy]</b>	Web Services Policy (WS-Policy)
44		<a href="http://www.w3.org/TR/ws-policy">http://www.w3.org/TR/ws-policy</a>
45	<b>[WS-PolicyAttach]</b>	Web Services Policy Attachment (WS-PolicyAttachment)
46		<a href="http://www.w3.org/TR/ws-policy-attach">http://www.w3.org/TR/ws-policy-attach</a>
47	<b>[XPATH]</b>	XML Path Language (XPath) Version 1.0.
48		<a href="http://www.w3.org/TR/xpath">http://www.w3.org/TR/xpath</a>
49	<b>[XML-Schema2]</b>	XML Schema Part 2: Datatypes Second Edition XML Schema Part 2: Datatypes
50		Second Edition, Oct. 28 2004.
51		<a href="http://www.w3.org/TR/xmlschema-2/">http://www.w3.org/TR/xmlschema-2/</a>

## 52 1.4 Naming Conventions

53 This specification follows some naming conventions for artifacts defined by the specification, as follows:

- 54 • For the names of elements and the names of attributes within XSD files, the names follow the  
55 CamelCase convention, with all names starting with a lower case letter, e.g. <element  
56 name="policySet" type="..."/>.
- 57 • For the names of types within XSD files, the names follow the CamelCase convention with all names  
58 starting with an upper case letter, e.g. <complexType name="PolicySet">.
- 59 • For the names of intents, the names follow the CamelCase convention, with all names starting with a  
60 lower case letter, EXCEPT for cases where the intent represents an established acronym, in which  
61 case the entire name is in upper case. An example of an intent which is an acronym is the "SOAP"  
62 intent.



---

## 63 2 Overview

### 64 2.1 Policies and PolicySets

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

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

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

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

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

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

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

### 92 2.2 Intents describe the requirements of Components, Services and 93 References

94 SCA **intents** are used to describe the abstract policy requirements of a component or the requirements of  
95 interactions between components represented by services and references. Intents provide a means for  
96 the developer and the assembler to state these requirements in a high-level abstract form, independent of  
97 the detailed configuration of the runtime and bindings, which involve the role of application deployer.  
98 Intents support late binding of services and references to particular SCA bindings, since they assist the  
99 deployer in choosing appropriate bindings and concrete policies which satisfy the abstract requirements  
100 expressed by the intents.

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

106 SCA favors the late binding approach since it promotes re-use of components. It allows the use of  
107 components in new application contexts, which might require the use of different bindings and different

108 concrete policies. Forcing early decisions on which bindings and policies to use is likely to limit re-use and  
109 limit the ability to use a component in a new context.

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

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

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

## 122 **2.3 Determining which policies apply to a particular wire**

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

132 In order for a service to be accessed by a wide range of clients, it is good practice for the service to  
133 support multiple alternative policies within a particular domain. So, if a service requires message  
134 confidentiality, instead of insisting on one specific encryption technology, the service can have a policySet  
135 which has a number of alternative encryption technologies, any of which are acceptable to the service.  
136 Equally, a reference can have a policySet attached which defines the range of encryption technologies  
137 which it is capable of using. Typically, the set of policies used for a given domain will reflect the  
138 capabilities of the binding and of the runtime being used for the service and for the reference.

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

---

## 147 3 Framework Model

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

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

### 156 3.1 Intents

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

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

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

177 If the configured instance of a binding is in conflict with the intents and policy sets selected for that  
178 instance, the SCA runtime MUST raise an error. [POL30001]. For example, a web service binding which  
179 requires the SOAP intent but which points to a WSDL binding that does not specify SOAP.

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

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

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

193 In general, SCA allows the developer or assembler to attach multiple qualifiers for a single

194 qualifiable intent to the same SCA construct. However, domain-specific constraints can prevent the use of  
195 some combinations of qualifiers (from the same qualifiable intent).

196 Intents, their qualifiers and their defaults are defined using the pseudo schema in Snippet 33-1:

197

```
198 <intent name="xs:NCName"  
199     constrains="list of QNames"?  
200     requires="list of QNames"?  
201     excludes="list of QNames"?  
202     mutuallyExclusive="boolean"?  
203     intentType="xs:string"? >  
204 <description> xs:string.</description?>  
205 <qualifier name="xs:string" default="xs:boolean" ?>*</description?>  
206 <description> xs:string.</description?>  
207 </qualifier>  
208 </intent>
```

209 *Snippet 3-1: intent Pseudo-Schema*

210

211 Where the intent element has the following attributes:

212 • @name (1..1) - an NCName that defines the name of the intent. **The QName for an intent MUST be**  
213 **unique amongst the set of intents in the SCA Domain.** [POL30002]

214 • @constrains (0..1) - a list of QNames that specifies the SCA constructs that this intent is meant to  
215 configure. If a value is not specified for this attribute then the intent can apply to any SCA element.

216 Note that the “constrains” attribute can name an abstract element type, such as sca:binding in our  
217 running example. This means that it will match against any binding used within an SCA composite  
218 file. An SCA element can match @constrains if its type is in a substitution group.

219 • @requires (0..1) - contains a list of QNames of intents which defines the set of all intents that the  
220 referring intent requires. In essence, the referring intent requires all the intents named to be satisfied.  
221 This attribute is used to compose an intent from a set of other intents. **Each QName in the @requires**  
222 **attribute MUST be the QName of an intent in the SCA Domain.** [POL30015] This use is further  
223 described in [Profile Intents](#).

224 • @excludes (0..1) - a list of QNames of intents that cannot be used with this intent. Intents might  
225 describe a policy that is incompatible or otherwise unrealizable when specified with other intents, and  
226 therefore are considered to be mutually exclusive. **Each QName in the @excludes attribute MUST be**  
227 **the QName of an intent in the SCA Domain.** [POL30016]

228 Two intents are mutually exclusive when any of the following are true:

- 229 – One of the two intents lists the other intent in its @excludes list.
- 230 – Both intents list the other intent in their respective @excludes list.

231 Where one intent is attached to an element of an SCA composite and another intent is attached to  
232 one of the element’s parents, the intent(s) that are effectively attached to the element differs  
233 depending on whether the two intents are mutually exclusive (see @excludes above and “Attaching  
234 intents to SCA elements”).

235 • @mutuallyExclusive (0..1) - a boolean with a default of “false”. If this attribute is present and has a  
236 value of “true” it indicates that the qualified intents defined for this intent are mutually exclusive.

237 • @intentType attribute (0..1) defines whether the intent is an interaction intent or an implementation  
238 intent. A value of "interaction", which is the default value, indicates that the intent is an interaction  
239 intent. A value of "implementation" indicates that the intent is an implementation intent.

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

- 242 • @name (1..1) - declares the name of the qualifier. The name of each qualifier MUST be unique within  
243 the intent definition. [POL30005].
- 244 • @default (0..1) - a boolean value with a default value of "false". If @default="true" the particular  
245 qualifier is the default qualifier for the intent. If an intent has more than one qualifier, one and only  
246 one MUST be declared as the default qualifier. [POL30004]. If only one qualifier for an intent is given  
247 it MUST be used as the default qualifier for the intent. [POL30025]
- 248 • qualifier/description (0..1) - an xs:string that holds a textual description of the qualifier.

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

251

```
252 <intent name="confidentiality" constrains="sca:binding">
253   <description>
254     Communication through this binding must prevent
255     unauthorized users from reading the messages.
256   </description>
257   <qualifier name="transport">
258     <description>Automatic encryption by transport
259     </description>
260   </qualifier>
261   <qualifier name="message" default='true'>
262     <description>Encryption applied to each message
263     </description>
264   </qualifier>
265 </intent>
```

266 *Snippet 3-2: Example intent Definition*

267

268 An Intent can be contributed to the SCA Domain by including its definition in a definitions.xml file within a  
269 Contribution in the Domain. Details of the definitions.xml files are described in the [SCA Assembly Model](#)  
270 [SCA-Assembly].

271 SCA normatively defines a set of core intents that all SCA implementations are expected to support, to  
272 ensure a minimum level of portability. Users of SCA can define new intents, or extend the qualifier set of  
273 existing intents. An SCA Runtime MUST include in the Domain the set of intent definitions contained in  
274 the Policy\_Intents\_Definitions.xml described in the appendix "Intent Definitions" of the SCA Policy  
275 specification. [POL30024] It is also good practice for the Domain to include concrete policies which satisfy  
276 these intents (this may be achieved through the provision of appropriate binding types and  
277 implementation types, augmented by policy sets that apply to those binding types and implementation  
278 types).

279 The normatively defined intents in the SCA specification might evolve in future versions of this  
280 specification. New intents could be added, additional qualifiers could be added to existing intents and the  
281 default qualifier for existing intents could change. Such changes would cause the namespace for the SCA  
282 specification to change.

## 283 3.2 Interaction Intents and Implementation Intents

284 An interaction intent is an intent designed to influence policy which applies to a service, a reference and  
285 the wires that connect them. Interaction intents affect wire matching between the two ends of a wire  
286 and/or the set of bytes that flow between the reference and the service when a service invocation takes  
287 place.

288 Interaction intents typically apply to <binding/> elements.

289 An implementation intent is an intent designed to influence policy which applies to an implementation  
290 artifact or to the relationship of that artifact to the runtime code which is used to execute the artifact.  
291 Implementation intents do not affect wire matching between references and services, nor do they affect  
292 the bytes that flow between a reference and a service.

293 Implementation intents often apply to <implementation/> elements, but they can also apply to <binding/>  
294 elements, where the desire is to influence the activity of the binding implementation code and how it  
295 interacts with the remainder of the runtime code for the implementation.

296 Interaction intents and implementation intents are distinguished by the value of the @intentType attribute  
297 in the intent definition.

### 298 3.3 Profile Intents

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

301 The presence of @requires attribute in the intent definition signifies that this is a profile intent. The  
302 @requires attribute can include all kinds of intents, including qualified intents and other profile intents.  
303 However, while a profile intent can include qualified intents, it cannot be a qualified intent. Thus, **the**  
304 **name of a profile intent MUST NOT have a "." in it.** [POL30006]

305 Requiring a profile intent is semantically identical to requiring the list of intents that are listed in its  
306 @requires attribute. **If a profile intent is attached to an artifact, all the intents listed in its @requires**  
307 **attribute MUST be satisfied as described in section 4.154.15.** [POL30007]

308 An example of a profile intent is an intent called **messageProtection** which is a shortcut for specifying  
309 both **confidentiality** and **integrity**, where **integrity** means to protect against modification, usually by  
310 signing. The intent definition is shown in Snippet 33-3:

```
311 <intent name="messageProtection"  
312   constrains="sca:binding"  
313   requires="confidentiality integrity">  
314   <description>  
315     Protect messages from unauthorized reading or modification.  
316   </description>  
317 </intent>
```

319 *Snippet 3-3: Example Profile Intent*

### 320 3.4 PolicySets

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

323 The pseudo schema for policySet is shown in Snippet 33-4:

```
324 <policySet name="NCName"  
325   provides="listOfQNames"?  
326   appliesTo="xs:string"?  
327   attachTo="xs:string"?  
328   xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912  
329   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">  
330   <policySetReference name="xs:QName"/>*  
331   <intentMap/>*  
332   <xs:any>*  
333 </policySet>
```

335 *Snippet 3-4: policySet Pseudo-Schema*

336

337 PolicySet has the attributes:

- 338 • @name (1..1) - the name for the policySet. The value of the @name attribute is the local part of a  
339 QName. **The QName for a policySet MUST be unique amongst the set of policySets in the SCA**  
340 **Domain.** [POL30017]

- 341 • @appliesTo (0..1) - a string which is an XPath 1.0 expression identifying one or more SCA constructs  
342 this policySet can configure. **The contents of @appliesTo MUST match the XPath 1.0 [XPATH]  
343 production Expr.** [POL30018] The @appliesTo attribute uses the "Deployed Composites Infoset" as  
344 described in Appendix A: The Deployed Composites Infoset section.
- 345 • @attachTo (0..1) - a string which is an XPath 1.0 expression identifying one or more elements in the  
346 Domain. It is used to declare which set of elements the policySet is actually attached to. **The  
347 contents of @attachTo MUST match the XPath 1.0 production Expr.** [POL30019] The XPath value of  
348 the @attachTo attribute is evaluated against the "Deployed Composite Infoset" as described in  
349 Appendix A: Defining the Deployed Composites Infoset. See the section on "[Attaching Intents and  
350 PolicySets to SCA Constructs](#)" for more details on how this attribute is used.
- 351 • @provides (0..1) - a list of intent QNames (that can be qualified), which declares the intents the  
352 PolicySet provides.

353 PolicySet contains one or more of the element children

- 354 • intentMap element
- 355 • policySetReference element
- 356 • xs:any extensibility element

357 Any mix of the above types of elements, in any number, can be included as children of the policySet  
358 element including extensibility elements. There are likely to be many different policy languages for  
359 specific binding technologies and domains. In order to allow the inclusion of any policy language within a  
360 policySet, the extensibility elements can be from any namespace and can be intermixed.

361 The SCA policy framework expects that [WS-Policy](#) will be a common policy language for expressing  
362 interaction policies, especially for Web Service bindings. Thus a common usecase is to attach WS-  
363 Policies directly as children of <policySet> elements; either directly as <wsp:Policy> elements, or as  
364 <wsp:PolicyReference> elements or using <wsp:PolicyAttachment>. These three elements, and others,  
365 can be attached using the extensibility point provided by the <xs:any> in the pseudo schema above. See  
366 example below.

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

```
369
370 <policySet name="SecureReliablePolicy"
371   provides="serverAuthentication.message exactlyOne"
372   appliesTo="//sca:binding.ws"
373   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
374   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
375   <wsp:PolicyAttachment>
376     <!-- policy expression and policy subject for
377       "basic server authentication" -->
378     ...
379   </wsp:PolicyAttachment>
380   <wsp:PolicyAttachment>
381     <!-- policy expression and policy subject for
382       "reliability" -->
383     ...
384   </wsp:PolicyAttachment>
385 </policySet>
```

386 *Snippet 3-5: Example policySet Definition*

387  
388 PolicySet authors need to be aware of the evaluation of the @appliesTo attribute in order to designate  
389 meaningful values for this attribute. Although policySets can be attached to any element in an SCA  
390 composite, the applicability of a policySet is not scoped by where it is attached in the SCA framework.  
391 Rather, policySets always apply to either binding instances or implementation elements regardless of

392 where they are attached. In this regard, the SCA policy framework does not scope the applicability of the  
393 policySet to a specific attachment point in contrast to other frameworks, such as WS-Policy.

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

397 With this design principle in mind, an XPath expression that is the value of an @appliesTo attribute  
398 designates what a policySet applies to. Note that the XPath expression will always be evaluated against  
399 the Domain Composite Infoset as described in Section 4.4.1 "The Form of the @attachTo Attribute". The  
400 policySet will apply to any child binding or implementation elements returned from the expression. So, for  
401 example, appliesTo="//binding.ws" will match any web service binding. If  
402 appliesTo="//binding.ws[@impl='axis']" then the policySet would apply only to web service bindings that  
403 have an @impl attribute with a value of 'axis'.

404 When writing policySets, the author needs to ensure that the policies contained in the policySet always  
405 satisfy the intents in the @provides attribute. Specifically, when using [WS-Policy](#) the optional attribute  
406 and the exactlyOne operator can result in alternative policies and uncertainty as to whether a particular  
407 alternative satisfies the advertised intents.

408 If the WS-Policy attribute optional = 'true' is attached to a policy assertion, it results in two policy  
409 alternatives, one that includes and one that does not include the assertion. During wire validation it is  
410 impossible to predict which of the two alternatives will be selected -if the absence of the policy assertion  
411 does not satisfy the intent, then it is possible that the intent is not actually satisfied when the policySet is  
412 used.

413 Similarly, if the WS-Policy operator exactlyOne is used, only one of the set of policy assertions within the  
414 operator is actually used at runtime. If the set of assertions is intended to satisfy one or more intents, it is  
415 vital to ensure that each policy assertion in the set actually satisfies the intent(s).

416 Note that section 4.12.1 on Wire Validity specifies that the strict version of the WS-Policy intersection  
417 algorithm is used to establish wire validity and determine the policies to be used. The strict version of  
418 policy intersection algorithm ignores the ignorable attribute on assertions. This means that the ignorable  
419 facility of WS-Policy cannot be used in policySets.

420 For further discussion on attachment of policySets and the computation of applicable policySets, please  
421 refer to [Section 4](#).

422 A policySet can be contributed to the SCA Domain by including its definition in a definitions.xml file within  
423 a Contribution in the Domain. Details of the definitions.xml files are described in the [SCA Assembly Model](#)  
424 [SCA-Assembly].

### 425 **3.4.1 IntentMaps**

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

428 The pseudo-schema for intentMaps is given in Snippet 33-6:

429

```
430 <intentMap provides="xs:QName">  
431   <qualifier name="xs:string"?>  
432     <xs:any>*</xs:any>  
433   </qualifier>  
434 </intentMap>
```

435 *Snippet 3-6: intentMap Pseudo-Schema*

436

437 When a policySet element contains a set of intentMap children, the value of the @provides attribute of  
438 each intentMap MUST correspond to an unqualified intent that is listed within the @provides attribute  
439 value of the parent policySet element. [POL30008]



440 If a policySet specifies a qualifiable intent in the @provides attribute, and it provides an intentMap for the  
441 qualifiable intent then that intentMap MUST specify all possible qualifiers for that intent. [POL30020]

442 For each qualifiable intent listed as a member of the @provides attribute list of a policySet element, there  
443 MUST be no more than one corresponding intentMap element that declares the unqualified form of that  
444 intent in its @provides attribute. In other words, each intentMap within a given policySet uniquely provides  
445 for a specific intent. [POL30010]

446 The @provides attribute value of each intentMap that is an immediate child of a policySet MUST be  
447 included in the @provides attribute of the parent policySet. [POL30021]

448 An intentMap element contains qualifier element children. Each qualifier element corresponds to a  
449 qualified intent where the unqualified form of that intent is the value of the @provides attribute value of  
450 the parent intentMap. The qualified intent is either included explicitly in the value of the enclosing  
451 policySet's @provides attribute or implicitly by that @provides attribute including the unqualified form of  
452 the intent.

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

456 As an example, the policySet element in Snippet 3-7 declares that it provides **confidentiality** using the  
457 @provides attribute. The alternatives (transport and message) it contains each specify the policy and  
458 policy subject they provide. The default is "transport".

```
459  
460 <policySet name="SecureMessagingPolicies"  
461   provides="confidentiality"  
462   appliesTo="//binding.ws"  
463   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
464   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">  
465   <intentMap provides="confidentiality" >  
466     <qualifier name="transport">  
467       <wsp:PolicyAttachment>  
468         <!-- policy expression and policy subject for  
469           "transport" alternative -->  
470         ...  
471       </wsp:PolicyAttachment>  
472     <wsp:PolicyAttachment>  
473     ...  
474   </wsp:PolicyAttachment>  
475   </qualifier>  
476   <qualifier name="message">  
477     <wsp:PolicyAttachment>  
478       <!-- policy expression and policy subject for  
479         "message" alternative -->  
480       ...  
481     </wsp:PolicyAttachment>  
482   </qualifier>  
483 </intentMap>  
484 </policySet>
```

485 *Snippet 3-7: Example policySet with an intentMap*

486  
487 PolicySets can embed policies that are defined in any policy language. Although WS-Policy is the most  
488 common language for expressing interaction policies, it is possible to use other policy languages Snippet  
489 33-8 is an example of a policySet that embeds a policy defined in a proprietary language. This policy  
490 provides "serverAuthentication" for binding.ws.

```
491  
492 <policySet name="AuthenticationPolicy"  
493   provides="serverAuthentication"  
494   appliesTo="//binding.ws"
```

```

495     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
496     <e:policyConfiguration xmlns:e="http://example.com">
497         <e:authentication type = "X509"/>
498         <e:trustedCAStore type="JKS"/>
499         <e:keyStoreFile>Foo.jks</e:keyStoreFile>
500         <e:keyStorePassword>123</e:keyStorePassword>
501     </e:authentication>
502     </e:policyConfiguration>
503 </policySet>

```

504 *Snippet 3-8: Example policySet Using a Proprietary Language*

## 505 3.4.2 Direct Inclusion of Policies within PolicySets

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

511 Following the inclusion of all policySet references, when a policySet element directly contains  
512 wsp:policyAttachment children or policies using extension elements, the set of policies specified as  
513 children **MUST** satisfy all the intents expressed using the @provides attribute value of the policySet  
514 element. [POL30011] The intent names in the @provides attribute of the policySet can include names of  
515 profile intents.

## 516 3.4.3 Policy Set References

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

520 When a policySet element contains policySetReference element children, the @name attribute of a  
521 policySetReference element designates a policySet defined with the same value for its @name attribute.  
522 Therefore, the @name attribute is a QName.

523 The set of intents in the @provides attribute of a referenced policySet **MUST** be a subset of the set of  
524 intents in the @provides attribute of the referencing policySet. [POL30013] Qualified intents are a subset  
525 of their parent qualifiable intent.

526 The usage of a policySetReference element indicates a copy of the element content children of the  
527 policySet that is being referred is included within the referring policySet. If the result of inclusion results in  
528 a reference to another policySet, the inclusion step is repeated until the contents of a policySet does not  
529 contain any references to other policySets.

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

533 Note that, since the attributes of a referenced policySet are effectively removed/ignored by this process, it  
534 is the responsibility of the author of the referring policySet to include any necessary intents in the  
535 @provides attribute of the policySet making the reference so that the policySet correctly advertises its  
536 aggregate policy.

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

540 Snippet 33-9 is an example to illustrate the inclusion of two other policySets in a policySet element:

```

541
542     <policySet name="BasicAuthMsgProtSecurity"
543         provides="serverAuthentication confidentiality"

```

```

544     appliesTo="//binding.ws"
545     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
546     <policySetReference name="acme:ServerAuthenticationPolicies"/>
547     <policySetReference name="acme:ConfidentialityPolicies"/>
548 </policySet>

```

549 *Snippet 3-9: Example policySet Including Other policySets*

550

551 The policySet in Snippet 33-9 refers to policySets for **serverAuthentication** and  
552 **confidentiality** and, by reference, provides policies and policy subject alternatives in these  
553 domains.

554 If the policySets referred to in Snippet 33-9 have the following content:

555

```

556 <policySet name="ServerAuthenticationPolicies"
557     provides="serverAuthentication"
558     appliesTo="//binding.ws"
559     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
560 <wsp:PolicyAttachment>
561     <!-- policy expression and policy subject for
562         "basic server authentication" -->
563     ...
564 </wsp:PolicyAttachment>
565 </policySet>

566 <policySet name="acme:ConfidentialityPolicies"
567     provides="confidentiality"
568     bindings="binding.ws"
569     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
570 <intentMap provides="confidentiality" >
571 <qualifier name="transport">
572 <wsp:PolicyAttachment>
573     <!-- policy expression and policy subject for
574         "transport" alternative -->
575     ...
576     </wsp:PolicyAttachment>
577 <wsp:PolicyAttachment>
578     ...
579 </wsp:PolicyAttachment>
580 </qualifier>
581 <qualifier name="message">
582 <wsp:PolicyAttachment>
583     <!-- policy expression and policy subject for
584         "message" alternative -->
585     ...
586     </wsp:PolicyAttachment>
587 </qualifier>
588 </intentMap>
589 </policySet>

```

591 *Snippet 3-10: Example Included policySets for Snippet 33-9*

592

593 The result of the inclusion of policySets via policySetReferences would be semantically  
594 equivalent to Snippet 33-11.

595

```

596 <policySet name="BasicAuthMsgProtSecurity"
597     provides="serverAuthentication confidentiality" appliesTo="//binding.ws"
598     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">
599 <wsp:PolicyAttachment>

```

```
600     <!-- policy expression and policy subject for
601         "basic server authentication" -->
602     ...
603 </wsp:PolicyAttachment>
604 <intentMap provides="confidentiality" >
605     <qualifier name="transport">
606         <wsp:PolicyAttachment>
607             <!-- policy expression and policy subject for
608                 "transport" alternative -->
609             ...
610         </wsp:PolicyAttachment>
611     <wsp:PolicyAttachment>
612         ...
613     </wsp:PolicyAttachment>
614 </qualifier>
615 <qualifier name="message">
616     <wsp:PolicyAttachment>
617         <!-- policy expression and policy subject for
618             "message" alternative -->
619         ...
620     </wsp:PolicyAttachment>
621 </qualifier>
622 </intentMap>
623 </policySet>
```

624 *Snippet 3-11: Equivalent policySet*

---

## 625 4 Attaching Intents and PolicySets to SCA Constructs

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

### 629 4.1 Attachment Rules – Intents

630 One or more intents can be attached to any SCA element used in the definition of components and  
631 composites. The attachment can be specified by using the following two mechanisms:

- 632 • **Direct Attachment** mechanism which is described in the section "Direct Attachment of Intents".
- 633 • **External Attachment** mechanism which is described in the section "External Attachment of  
634 Intents".

### 635 4.2 Direct Attachment of Intents

636 Intents can be attached to any SCA element used in the definition of components and composites.  
637 Intents are attached by using the **@requires** attribute or the <requires> child element. The @requires  
638 attribute takes as its value a list of intent names. Similarly, the <requires> element takes as its value a list  
639 of intent names. Intents can also be attached to interface definitions. For WSDL portType elements  
640 (WSDL 1.1) the @requires attribute can be used to attach the list of intents that are needed by the  
641 interface. Other interface languages can define their own mechanism for attaching a list of intents. Any  
642 intents attached to an interface definition artifact, such as a WSDL portType, MUST be added to the  
643 intents attached to the service or reference to which the interface definition applies. If no intents are  
644 attached to the service or reference then the intents attached to the interface definition artifact become  
645 the only intents attached to the service or reference. [POL40027]

646 Because intents specified on interfaces can be seen by both the provider and the client of a service, it is  
647 appropriate to use them to specify characteristics of the service that both the developers of provider and  
648 the client need to know.

649 For example:

650

```
651 <service requires="acme:IntentName1 acme:IntentName2">  
652   <binding.xxx/>  
653   ...  
654 </service>  
655  
656 <reference requires="acme:IntentName1 acme:IntentName2">  
657   <binding.xxx/>  
658   ...  
659 </reference>
```

660 *Snippet 4-1: Example of @requires on a service or a reference*

```
661 <service>  
662   <requires intents="acme:IntentName1 acme:IntentName2" />  
663   <binding.xxx/>  
664   ...  
665 </service>  
666  
667 <reference>  
668   <requires intents="acme:IntentName1 acme:IntentName2" />  
669   <binding.xxx/>  
670   ...  
671 </reference>
```

672 Snippet 4-2: Example of a <requires> subelement to attach intents to a service or a reference

### 673 4.3 External Attachment of Intents and PolicySets

674 External Attachment of intents and policySets is used for deployment-time application of intents and  
675 policySets to SCA elements. It is called "external attachment" because the principle of the mechanism is  
676 that the attachment is declared in a place that is separate from the composite files that contain the  
677 elements. This separation provides the deployer with a way to attach intents and/or policySets without  
678 having to modify the artifacts where the intents and policySets are attached.

679 Intents and policySets can be attached to one or more SCA elements by using the externalAttachment  
680 element, which is declared within a definitions file.

681 The pseudo-schema for the externalAttachment element is shown in Snippet 44-3.

682

```
683 <externalAttachment intents="sca:listOfQNames"  
684 policySets="sca:listofQNames"  
685 attachTo = "xs:string" />
```

686 Snippet 4-3: Pseudo-schema for externalAttachment element

687

688 The **externalAttachment** element has the **attributes**:

- 689 • **@intents : listOfQNames (0..1)** A list of QNames identifying intents which are attached to the  
690 elements declared in the @attachTo attribute.
- 691 • **@policySets : listOfQNames (0..1)**. A list of QNames identifying policySets which are attached to  
692 the elements declared in the @attachTo attribute
- 693 • **@attachTo : string (1..1)**. A string containing an XPath 1.0 expression identifying one or more  
694 elements in the Domain. It is used to declare which set of elements the intents are attached to.  
695 The contents of the @attachTo attribute of an externalAttachment element MUST match the XPath  
696 1.0 production Expr. [POL40035] The XPath value of the @attachTo attribute is evaluated against the  
697 "Deployed Composite Infoset" as described in the appendix section "The Deployed Composites  
698 Infoset".

699

### 700 4.4 Attachment Rules - PolicySets

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

- 703 • **Direct Attachment** mechanism which is described in [Direct Attachment of PolicySets](#).
- 704 • **External Attachment** mechanism which is described in [External Attachment of PolicySets](#).

705 SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms  
706 for policySet attachment. [POL40010] SCA implementations supporting only the External Attachment  
707 mechanism MUST ignore the policy sets that are applicable via the Direct Attachment mechanism.  
708 [POL40011] SCA implementations supporting only the Direct Attachment mechanism MUST ignore the  
709 policy sets that are applicable via the External Attachment mechanism. [POL40012] SCA  
710 implementations supporting both Direct Attachment and External Attachment mechanisms MUST ignore  
711 policy sets applicable to any given SCA element via the Direct Attachment mechanism when there exist  
712 policy sets applicable to the same SCA element via the External Attachment mechanism [POL40001]

### 713 4.5 Direct Attachment of PolicySets

714 Direct Attachment of PolicySets can be achieved by

- 715 • Using the optional **@policySets** attribute of the SCA element

- 716 • Adding an optional child **<policySetAttachment/>** element to the SCA element

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

718 For example:

719

```
720 <service> or <reference>...
721   <binding.binding-type policySets="listOfQNames">
722   </binding.binding-type>
723   ...
724 </service> or </reference>
```

725 *Snippet 4-4: Example of @policySets on a service*

726

727 The **<policySetAttachment/>** element is an alternative way to attach a policySet to an SCA composite.

728

```
729 <policySetAttachment name="xs:QName" />
```

730 *Snippet 4-5: policySetAttachment Pseudo-Schema*

731

- 732 • @name (1..1) – the QName of a policySet.

733

734 For example:

735

```
736 <service> or <reference>...
737   <binding.binding-type>
738     <policySetAttachment name="sns:EnterprisePolicySet">
739   </binding.binding-type>
740   ...
741 </service> or </reference>
```

742 *Snippet 4-6: Example of policySetAttachment in a service or reference*

743

744 Where an element has both a @policySets attribute and a **<policySetAttachment/>** child element, the  
745 policySets declared by both are attached to the element.

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

- 747 • It is possible to specify QoS requirements by attaching abstract intents to an element at the time of  
748 development. In this case, it is implied that the concrete bindings and policies that satisfy the abstract  
749 intents are not assigned at development time but the intents are used **to select the concrete**  
750 **Bindings and Policies** at deployment time. Concrete policies are encapsulated within policySets  
751 that are applied during deployment using the external attachment mechanism. The intents associated  
752 with a SCA element is the union of intents specified for it and its parent elements subject to the  
753 detailed rules below.
- 754 • It is also possible to specify QoS requirements for an element by using both intents and concrete  
755 policies contained in directly attached policySets at development time. In this case, it is possible **to**  
756 **configure the policySets, by overriding the default settings in the specified policySets using**  
757 **intents**. The policySets associated with a SCA element is the union of policySets specified for it and  
758 its parent elements subject to the detailed rules below.

759 See also "Matching Intents and PolicySets" for a discussion of how intents are used to guide the selection  
760 and application of specific policySets.

## 761 4.6 External Attachment of PolicySets

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

767 A PolicySet is attached to one or more elements in one of two ways:

768 a) through the @attachTo attribute of the policySet

769 b) through the @attachTo attribute of an <externalAttachment/> element which references the policySet  
770 in its @policySets attribute

771 c) through a reference (via policySetReference) from a policySet that uses the @attachTo attribute.

772

### 773 4.6.1 Cases Where Multiple PolicySets are attached to a Single Artifact

774 Multiple PolicySets can be attached to a single artifact. This can happen either as the result of one or  
775 more direct attachments or as the result of one or more external attachments which target the particular  
776 artifact.

## 777 4.7 Attaching intents to SCA elements

778 A list of intents can be attached to any SCA element by using the @requires attribute or the <requires>  
779 subelement.

780 The intents which apply to a given element depend on

- 781 • the intents expressed in its @requires attribute and/or its <requires> subelement
- 782 • intents derived from the structural hierarchy of the element
- 783 • intents derived from the implementation hierarchy of the element

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

787 Any two intents applied to a given element MUST NOT be mutually exclusive [POL40009]. Specific  
788 examples are discussed later in this document.

### 789 4.7.1 Implementation Hierarchy of an Element

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

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

799 Rule 1: The intents declared on elements lower in the implementation hierarchy of a given element MUST  
800 be applied to the element. [POL40014] A qualifiable intent expressed lower in the hierarchy can be  
801 qualified further up the hierarchy, in which case the qualified version of the intent MUST apply to the  
802 higher level element. [POL40004]



## 803 4.7.2 Structural Hierarchy of an Element

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

806 As an example, for the composite in Snippet 44-7::

807

```
808 <composite name="C1" requires="i1">  
809   <service name="CS" promotes="X/S">  
810     <binding.ws requires="i2">  
811   </service>  
812   <component name="X">  
813     <implementation.java class="foo"/>  
814     <service name="S" requires="i3">  
815   </component>  
816 </composite>
```

817 *Snippet 4-7: Example Composite to Illustrate Structural Hierarchy*

818

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

822 **Rule2: The intents declared on elements higher in the structural hierarchy of a given element MUST be  
823 applied to the element EXCEPT**

- 824 • **if any of the inherited intents is mutually exclusive with an intent applied on the element, then the  
825 inherited intent MUST be ignored**
- 826 • **if the overall set of intents from the element itself and from its structural hierarchy contains both an  
827 unqualified version and a qualified version of the same intent, the qualified version of the intent MUST  
828 be used.**

829 [POL40005]

## 830 4.7.3 Combining Implementation and Structural Policy Data

831 When there are intents present in both hierarchies implementation intents are calculated before the  
832 structural intents. In other words, **When combining implementation hierarchy and structural hierarchy  
833 policy data, Rule 1 MUST be applied BEFORE Rule 2.** [POL40015]

834 Note that each of the elements in the hierarchy below a <component> element, such as <service/>,  
835 <reference/> or <binding/>, inherits intents from the equivalent elements in the componentType of the  
836 implementation used by the component. So the <service/> element of the <component> inherits any  
837 intents on the <service/> element with the same name in the <componentType> - and a <binding/>  
838 element under the service in the component inherits any intents on the <binding/> element of the service  
839 (with the same name) in the componentType. Errors caused by mutually exclusive intents appearing on  
840 corresponding elements in the component and on the componentType only occur when those elements  
841 match one-to-one. Mutually exclusive intents can validly occur on elements that are at different levels in  
842 the structural hierarchy (as defined in Rule 2).

843 Note that it might often be the case that <binding/> elements will be specified in the structure under the  
844 <component/> element in the composite file (especially at the Domain level, where final deployment  
845 configuration is applied) - these elements might have no corresponding elements defined in the  
846 componentType structure. In this situation, the <binding/> elements don't acquire any intents from the  
847 componentType directly (ie there are no elements in the implementation hierarchy of the <binding/>  
848 elements), but those <binding/> elements will acquire intents "flowing down" their structural hierarchy as  
849 defined in Rule 2 - so, for example if the <service/> element is marked with @requires="confidentiality",  
850 the bindings of that service will all inherit that intent, assuming that they don't have their own exclusive  
851 intents specified.

852 Also, for example, where say a component <service.../> element has an intent that is mutually exclusive  
853 with an intent in the componentType<service.../> element with the same name, it is an error, but this  
854 differs when compared with the case of the <component.../> element having an intent that is mutually  
855 exclusive with an intent on the componentType <service/> element - because they are at different  
856 structural levels: the intent on the <component/> is ignored for that <service/> element and there is no  
857 error.

## 858 4.7.4 Examples

859 As an example, consider the composite in Snippet 44-8:

860

```
861 <composite name="C1" requires="i1">  
862   <service name="CS" promotes="X/S">  
863     <binding.ws requires="i2">  
864   </service>  
865   <component name="X">  
866     <implementation.java class="foo"/>  
867     <service name="S" requires="i3">  
868   </component>  
869 </composite>
```

870 *Snippet 4-8: Example composite with intents*

871

872 ...the component service with name "S" has the service named "S" in the componentType of the  
873 implementation in its implementation hierarchy, and the composite service named "CS" has the  
874 component service named "S" in its implementation hierarchy. Service "CS" acquires the intent "i3" from  
875 service "S" – and also gets the intent "i1" from its containing composite "C1" IF i1 is not mutually  
876 exclusive with i3.

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

880 Consider the composite in Snippet 44-9:

881

```
882 <composite requires="confidentiality">  
883   <service name="foo" .../>  
884   <reference name="bar" requires="confidentiality.message" />  
885 </composite>
```

886 *Snippet 4-9: Example reference with intents*

887

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

893 Consider the variation in Snippet 44-10 where a qualified intent is specified at the composite level:

894

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

899 *Snippet 4-10: Example Qualified intents*

900

901 In this case, both the **confidentiality.transport** and the **confidentiality.message** intent are applied for  
902 the reference 'bar'. If there are no bindings that support this combination, an error will be generated.  
903 However, since in some cases multiple qualifiers for the same intent can be valid or there might be  
904 bindings that support such combinations, the SCA specification allows this.  
905

## 906 4.8 Usage of Intent and Policy Set Attachment together

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

910 When the @requires attribute or the <requires> subelement and one or both of the direct policySet  
911 attachment mechanisms are used together during development, it indicates the intention of the developer  
912 to configure the element, such as a binding, by the application of specific policySet(s) to this element.

913 The same behavior can be enabled by external attachment of intents and policySets.

914 Developers who attach intents and policySets in conjunction with each other need to be aware of the  
915 implications of how the policySets are selected and how the intents are utilized to select specific  
916 intentMaps, override defaults, etc. The details are provided in the Section [Guided Selection of  
917 PolicySets using Intents](#).

## 918 4.9 Intents and PolicySets on Implementations and Component Types

919 It is possible to specify intents and policySets within a component's implementation, which get exposed to  
920 SCA through the corresponding *component type*. How the intents or policies are specified within an  
921 implementation depends on the implementation technology. For example, Java can use an @requires  
922 annotation to specify intents.

923 The intents and policySets specified within an implementation can be found on the  
924 <sca:implementation.\*> and the <sca:service> and <sca:reference> elements of the component type.  
925 Snippet 44-11 shows direct attachment of intents and policySets using the @requires and @policySets  
926 attributes:

```
927 <componentType>  
928   <implementation.* requires="listOfQNames" policySets="="listOfQNames">  
929     ...  
930   </implementation>  
931   <service name="myService" requires="listOfQNames"  
932     policySets="listOfQNames">  
933     ...  
934   </service>  
935   <reference name="myReference" requires="listOfQNames"  
936     policySets="="listOfQNames">  
937     ...  
938   </reference>  
939   ...  
940 </componentType>
```

941 *Snippet 4-11: Example of intents on an implementation*

942

943 Intents expressed in the component type are handled according to the rule defined for the implementation  
944 hierarchy. See [Intent rule 2](#)

945 For explicitly listed policySets, the list in the component using the implementation can override policySets  
946 from the component type. **If a component has any policySets attached to it (by any means), then any  
947 policySets attached to the componentType MUST be ignored. [POL40006]**

## 948 4.10 Intents on Interfaces

949 Interfaces are used in association with SCA services and references. These interfaces can be declared  
950 in SCA composite files and also in SCA componentType files. The interfaces can be defined using a  
951 number of different interface definition languages which include WSDL, Java interfaces and C++ header  
952 files.

953 It is possible for some interfaces to be referenced from an implementation rather than directly from any  
954 SCA files. An example of this usage is a Java implementation class file that has a reference declared  
955 that in turn uses a Java interface defined separately. When this occurs, the interface definition is treated  
956 from an SCA perspective as part of the componentType of the implementation, logically being part of the  
957 declaration of the related service or reference element.

958 Both the declaration of interfaces in SCA and also the definitions of interfaces can carry policy-related  
959 information. In particular, both the declarations and the definitions can have either intents attached to  
960 them, or policySets attached to them - or both. For SCA declarations, the intents and policySets always  
961 apply to the whole of the interface (ie all operations and all messages within each operation). For  
962 interface definitions, intents and policySets can apply to the whole interface or they can apply only to  
963 specific operations within the interface or they can even apply only to specific messages within particular  
964 operations. (To see how this is done, refer to the places in the SCA specifications that deal with the  
965 relevant interface definition language)

966 This means, in effect, that there are 4 places which can hold policy related information for interfaces:

- 967 1. The interface definition file that is referenced from the component type.
- 968 2. The interface declaration for a service or reference in the component type
- 969 3. The interface definition file that is referenced from the component declaration in a composite
- 970 4. The interface declaration within a component

971 When calculating the set of intents and set of policySets which apply to either a service element or to a  
972 reference element of a component, intents and policySets from the interface definition and from the  
973 interface declaration(s) MUST be applied to the service or reference element and to the binding  
974 element(s) belonging to that element. [POL40016]

975 The locations where interfaces are defined and where interfaces are declared in the componentType and  
976 in a component MUST be treated as part of the implementation hierarchy as defined in Section 4.5  
977 Attaching intents to SCA elements. [POL40019]

## 978 4.11 BindingTypes and Related Intents

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

988 The bindingType element is used to declare a class of binding available in a SCA Domain. The pseudo-  
989 schema for the bindingType element is shown in Snippet 44-12:

990

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

994 *Snippet 4-12: bindingTypePseudo-Schema*

995

996 • @type (1..1) – declares the NCName of the bindingType, which is used to form the QName of the  
997 bindingType. The QName of the bindingType MUST be unique amongst the set of bindingTypes in  
998 the SCA Domain. [POL40020]

999 • @alwaysProvides (0..1) – a list of intent QNames that are natively provided. A natively provided intent  
1000 is hard-coded into the binding implementation. The function represented by the intent cannot be  
1001 turned off.

1002 • @mayProvides (0..1) – a list of intent QNames that are natively provided by the binding  
1003 implementation, but which are activated only when present in the intent set that is applied to a binding  
1004 instance.

1005 A binding implementation MUST implement all the intents listed in the @alwaysProvides and  
1006 @mayProvides attributes. [POL40021]

1007 The kind of intents a given binding might be capable of providing, beyond these inherent intents, are  
1008 implied by the presence of policySets that declare the given binding in their @appliesTo attribute.

1009 For example, if the policySet in Snippet 44-13 is available in a SCA Domain it says that the (example)  
1010 foo:binding.ssl can provide “reliability” in addition to any other intents it might provide inherently.

1011

```
1012 <policySet name="ReliableSSL" provides="exactlyOnce"  
1013     appliesTo="//foo:binding.ssl">  
1014     ...  
1015 </policySet>
```

1016 *Snippet 4-13: Example policySet Applied to a binding*

## 1017 4.12 Treatment of Components with Internal Wiring

1018 This section discusses the steps involved in the development and deployment of a component and its  
1019 relationship to selection of bindings and policies for wiring services and references.

1020 The SCA developer starts by defining a component. Typically, this contains services and references. It  
1021 can also have intents attached at various locations within composite and component types as well as  
1022 policySets attached at various locations.

1023 Both for ease of development as well as for deployment, the wiring constraints to relate services and  
1024 references need to be determined. This is accomplished by matching constraints of the services and  
1025 references to those of corresponding references and services in other components.

1026 In this process, the intents, and the policySets that apply to both sides of a wire play an important role. In  
1027 addition, concrete policies need to be selected that satisfy the intents for the service and the reference  
1028 and are also compatible with each other. For services and references that make use of bidirectional  
1029 interfaces, the same determination of matching policySets also has to take place for callbacks.

1030 Determining wire compatibility plays an important role prior to deployment as well as during the  
1031 deployment phases of a component. For example, during development, it helps a developer to determine  
1032 whether it is possible to wire services and references using the policySets available in the development  
1033 environment. During deployment, the wiring constraints determine whether wiring can be achievable. It  
1034 also aids in adding additional concrete policies or making adjustments to concrete policies in order to  
1035 deliver the constraints. Here are the concepts that are needed in making wiring decisions:

- 1036 • The set of intents that individually apply to *each* service or reference.
- 1037 • When possible the intents that are applied to the service, the reference and callback (if any) at the  
1038 other end of the wire. This set is called the *required intent set* and only applies when dealing with a  
1039 wire connecting two components within the same SCA Domain. When external connections are  
1040 involved, from clients or to services that are outside the SCA domain, intents are only available for the  
1041 end of the connection that is inside the domain. See Section "[Preparing Services and References  
1042 for External Connection](#)" for more details.

- 1043 • The policySets that apply to each service or reference.
- 1044 The set of provided intents for a binding instance is the union of the set of intents listed in the  
1045 "alwaysProvides" attribute and the set of intents listed in the "mayProvides" attribute of its binding type.  
1046 The capabilities represented by the "alwaysProvides" intent set are always present, irrespective of the  
1047 configuration of the binding instance. Each capability represented by the "mayProvides" intent set is only  
1048 present when the list of intents applied to the binding instance (either applied directly, or inherited)  
1049 contains the particular intent (or a qualified version of that intent, if the intent set contains an unqualified  
1050 form of a qualifiable intent). When an intent is directly provided by the binding type, there is no need to  
1051 apply a policy set that provides that intent.
- 1052 When bidirectional interfaces are in use, the same process of selecting policySets to provide the intents is  
1053 also performed for the callback bindings.

#### 1054 **4.12.1 Determining Wire Validity and Configuration**

1055 The above approach determines the policySets that are used in conjunction with the binding instances  
1056 listed for services and references. For services and references that are resolved using SCA wires, the  
1057 policySets chosen on each side of the wire might or might not be compatible. The following approach is  
1058 used to determine whether they are compatible and whether the wire is valid. If the wire  
1059 uses a bidirectional interface, then the following technique ensures that valid configured  
1060 policySets can be found for both directions of the bidirectional interface.

1061 The SCA runtime MUST determine the compatibility of the policySets at each end of a wire using the  
1062 compatibility rules of the policy language used for those policySets. [POL40022] The policySets at each  
1063 end of a wire MUST be incompatible if they use different policy languages. [POL40023] However, there is  
1064 a special case worth mentioning:

- 1065 • If both sides of the wire use identical policySets (by referring to the same policySet by its QName in  
1066 both sides of the wire), then they are compatible.

1067 Where the policy language in use for a wire is WS-Policy, strict WS-Policy intersection MUST be used to  
1068 determine policy compatibility. [POL40024]

1069 In order for a reference to connect to a particular service, the policies of the reference MUST intersect  
1070 with the policies of the service. [POL40025]

#### 1071 **4.13 Preparing Services and References for External Connection**

1072 Services and references are sometimes not intended for SCA wiring, but for communication with software  
1073 that is outside of the SCA domain. References can contain bindings that specify the endpoint address of  
1074 a service that exists outside of the current SCA domain. Services can specify bindings that can be  
1075 exposed to clients that are outside of the SCA domain.

1076 Matching service/reference policies across the SCA Domain boundary MUST use WS-Policy compatibility  
1077 (strict WS-Policy intersection) if the policies are expressed in WS-Policy syntax. [POL40007] For other  
1078 policy languages, the policy language defines the comparison semantics.

1079 For external services and references that make use of bidirectional interfaces, the same determination  
1080 of matching policies has to also take place for the callback.

1081 The policies that apply to the service/reference are computed as discussed in [Guided Selection of  
1082 PolicySets using Intents](#).

#### 1083 **4.14 Deployment**

1084 The SCA Assembly Specification [SCA-Assembly] describes how to contribute SCA artifacts to the SCA  
1085 Domain, and how to deploy them to create running components. This section discusses the Policy  
1086 aspects of deployment: how intents, externalAttachments and policySets are contributed, how intents are  
1087 satisfied by concrete policies in policySets and the process of redeployment when intents,  
1088 externalAttachments or policySets change.

1089 Intents, externalAttachments and policySets can be contributed to the Domain contained within  
1090 contributions. These contributions might only contain policy artifacts or they might also contain  
1091 composites and related artifacts. Intents and policySets can be attached to elements within a composite  
1092 either by direct attachment (where an attribute or child element performs the attachment) or they can be  
1093 attached through the external attachment mechanism.

1094 When a composite is deployed, the intents which are attached to each element must be evaluated, both  
1095 the directly attached intents and intents attached through external attachment. For external attachment,  
1096 this means evaluating the @attachTo attribute of each externalAttachment element with a non-empty  
1097 @intents attribute in the SCA Domain - the intents are attached to those elements which are selected by  
1098 the XPath expression in the externalAttachment/@attachTo attribute.

1099 **During the deployment of SCA composites, first all <externalAttachment/> elements within the Domain  
1100 MUST be evaluated to determine which intents are attached to elements in the newly deployed composite  
1101 and then all policySets within the Domain with an @attachTo attribute or <externalAttachment> elements  
1102 that attach policySets MUST be evaluated to determine which policySets are attached to elements in the  
1103 newly deployed composite. [POL40034]**

1104 Once the intents attached to the elements of a composite are known, the policySets attached to each  
1105 element are evaluated. If external attachment of policySets is supported, then each policySet in the  
1106 Domain is examined and the XPath expression of the @attachTo attribute is evaluated and the policySet  
1107 is attached to SCA elements selected by the expression.

1108 **The SCA runtime MUST raise an error if the @attachTo XPath expression resolves to an SCA <property>  
1109 element, or any of its children.[POL40002]**

1110 The algorithm for matching intents with policySets is described in the section "[Matching Intents and  
1111 PolicySets](#)".

#### 1112 **4.14.1 Redeployment of Intents and PolicySets**

1113 Intents and policySets can be managed separately from other SCA artifacts. It is possible for an SCA  
1114 runtime to allow deployment of new intents, new externalAttachments and policySets, modification of  
1115 existing intents, externalAttachments and policySets or the undeployment of existing intents,  
1116 externalAttachments and policySets, while composites and components are deployed or are running in  
1117 the Domain. Collectively, this is referred to as **the redeployment of intents and policySets**.

1118 Redeployment can be caused by:

- 1119 • Adding an externalAttachment element to the Domain
- 1120 • Adding a policySet with a non-empty attachTo attribute to the Domain
- 1121 • Changing the structure of an intent or policySet in the Domain that is directly or externally  
1122 attached.
- 1123 • Changing the attachTo, policySets or intents attribute of a externalAttachment in the Domain.
- 1124 • Removing directly attached intents or policySets from the Domain.
- 1125 • Removing one or more externalAttachment elements from the Domain.

1126

1127 Note that an SCA runtime can choose to disallow redeployment of intents and policySets.

1128 If an SCA runtime supports the redeployment of intents and policySets, there is an implication that the  
1129 changed intent and policySet artifacts can change the configuration of composites and components in the  
1130 Domain. How the changes are implemented is determined by the design of the SCA runtime concerned,  
1131 but there are three general approaches, as outlined in the SCA Assembly specification [SCA-Assembly]:

- 1132 • the SCA runtime can require that all existing running component instances affected by the  
1133 configuration changes are stopped and then restarted using the new configuration
- 1134 • the SCA runtime can leave existing running component instances unchanged, but any new  
1135 component instances are created using the new configuration

- 1136
- 1137
- 1138
- 1139
- 1140
- 1141
- the SCA runtime can deploy the new or changed intents, externalAttachments and policySets to the SCA Domain but not activate the changes until some time in the future. Running component instances and new component instances are not affected (i.e., the component configuration is not changed) by the newly deployed intents, externalAttachments and policySets until the SCA runtime activates those changes. The means and mechanism for performing this activation is outside the scope of this specification.

1142 Redeployment of intents and policySets, when it occurs, first performs external attachment of intents followed by external attachment of policySets. After this, the algorithm for matching intents with policySets is executed. The redeployment process may succeed or it may fail, in that the set of intents attached to artifacts in the domain may or may not be satisfied. If the process of redeployment of intents, externalAttachments and/or policySets fails because one or more intents are left unsatisfied, an error MUST be raised. [POL40029] If the process of redeployment of intents, externalAttachments and/or policySets fails, the changed intents, externalAttachments and/or policySets MUST NOT be deployed and no change is made to deployed and running artifacts. [POL40030]

1150 If the redeployment of intents, externalAttachments and policySets succeeds in that all intents are satisfied, then the policies attached to one or more deployed SCA elements may change. When redeployment of intents, externalAttachments and policySets succeeds, the components whose policies are affected by the redeployment MAY have their policies updated by the SCA runtime dynamically without the need to stop and restart those components. [POL40031]

1155 Where components are updated by redeployment of intents, externalAttachments and policySets (their configuration is changed in some way, which includes changing the policies associated with a component), the new configuration MUST apply to all new instances of those components once the redeployment is complete. [POL40032] Where a component configuration is changed by the redeployment of intents, externalAttachments and policySets, the SCA runtime either MAY choose to maintain existing instances with the old configuration of the component, or the SCA runtime MAY choose to stop and discard existing instances of the component. [POL40033]

## 1162 4.15 Matching Intents and PolicySets

1163 This section describes the selection of concrete policies that provide the requirements expressed by the set of intents associated with an SCA element. The purpose is to construct the set of concrete policies that are attached to an element taking into account the explicitly declared policySets that are attached to an element as well as policySets that are externally attached. The aim is to satisfy all of the intents that apply to each element.

1168 If the unqualified form of a qualifiable intent is attached to an element, it can be satisfied by a policySet that specifies any one of qualified forms of the intent in the value of its @provides attribute, or it can be satisfied by a policySet which @provides the unqualified form of the intent. If the qualified form of the intent is attached to an element then it can be satisfied only by a policy that @provides that qualified form of the intent.

1173

1174 **Note: In the following, the following rule is observed when an intent set is computed.**

1175 When a profile intent is encountered in either a global @requires attribute, an intent/@requires attribute, a <requires> subelement or a policySet/@provides attribute, the profile intent is immediately replaced by the intents that it composes (i.e. 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 here, in order to not have to restate this at multiple places].

1180 The **required intent set** that is attached to an element is:

- 1181
- 1182
- 1183
- 1184
- 1185
1. The set of intents attached to the element either by direct attachment or external attachment via the mechanisms described in the sections "Direct Attachment of Intents" and "External Attachment of Intents".
  2. add any intents found in any related interface definition or declaration, as described in the section "Intents on Interfaces".



- 1186 3. add any intents found on elements below the target element in its implementation hierarchy as  
1187 defined in Rule 1 in the section "[Implementation Hierarchy of an Element](#)".
- 1188 4. add any intents attached to each ancestor element in the element's structural hierarchy as defined in  
1189 [Rule 2](#) in in the section "[Structural Hierarchy of an Element](#)"
- 1190 5. remove any intents that do not include the target element's type in their @constrains attribute.
- 1191 6. remove the unqualified version of an intent if the set also contains a qualified version of that intent
- 1192 **If the required intent set contains a mutually exclusive pair of intents the SCA runtime MUST reject the**  
1193 **document containing the element and raise an error. [POL40017]**
- 1194 The **directly provided intent set** for an element is the set of intents listed in the @alwaysProvides  
1195 attribute combined with the set of intents listed in the @mayProvides attribute of the bindingType or  
1196 implementationType declaration for a binding or implementation element respectively.
- 1197 The **set of PolicySets attached to an element** include those **explicitly specified** using the @policySets  
1198 attribute or the <policySetAttachment/> element and those which are **externally attached**.
- 1199 A policySet **applies to** a target element if the result of the XPath expression contained in the policySet's  
1200 @appliesTo attribute, when evaluated against the document containing the target element, includes the  
1201 target element. For example, @appliesTo="//binding.ws[@impl='axis']" matches any binding.ws element  
1202 that has an @impl attribute value of 'axis'.
- 1203 The set of **explicitly specified** policySets for an element is:
- 1204 1. The union of the policySets specified in the element's @policySets attribute and those specified in  
1205 any <policySetAttachment/> child element(s).
- 1206 2. add the policySets declared in the @policySets attributes and <policySetAttachment/> elements from  
1207 elements in the structural hierarchy of the element.
- 1208 3. remove any policySet where the policySet does not apply to the target element.  
1209 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*
- 1210 The set of **externally attached** policySets for an element is:
- 1211 1. Each <PolicySet/> in the Domain where the element is targeted by the @attachTo attribute of the  
1212 policySet
- 1213 2. Each PolicySet that is attached to the target element through use of the <externalAttachment/>  
1214 element
- 1215 3. remove any policySet where the policySet does not apply to the target element.  
1216 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*
- 1217 A policySet **provides an intent** if any of the statements are true:
- 1218 1. The intent is contained in the @provides list of the policySet.
- 1219 2. The intent is a qualified intent and the unqualified form of the intent is contained in the @provides list  
1220 of the policySet.
- 1221 3. The policySet @provides list contains a qualified form of the intent (where the intent is qualifiable).
- 1222 **All intents in the required intent set for an element MUST be provided by the directly provided intents set**  
1223 **and the set of policySets that apply to the element, or else an error is raised. [POL40018]**
- 1224

---

## 5 Implementation Policies

1225

1226 The basic model for Implementation Policies is very similar to the model for interaction policies described  
1227 above. Abstract QoS requirements, in the form of intents, can be associated with SCA component  
1228 implementations to indicate implementation policy requirements. These abstract capabilities are mapped  
1229 to concrete policies via policySets at deployment time. Alternatively, policies can be associated directly  
1230 with component implementations using policySets. Intents and policySets can be attached to an  
1231 implementation using any of the mechanisms described in “Attaching Intents and PolicySets to SCA  
1232 Constructs”.

1233 Snippet 55-1 shows one way of associating intents with an implementation:

1234

```
1235 <component name="xs:NCName" ... >  
1236   <implementation.* ... requires="listOfQNames">  
1237     ...  
1238   </implementation>  
1239   ...  
1240 </component>
```

1241 *Snippet 5-1: Example of intents Associated with an implementation*

1242

1243 If, for example, one of the intent names in the value of the @requires attribute is 'logging', this indicates  
1244 that all messages to and from the component have to be logged. The technology used to implement the  
1245 logging is unspecified. Specific technology is selected when the intent is mapped to a policySet (unless  
1246 the implementation type has native support for the intent, as described in the next section). A list of  
1247 implementation intents can also be specified by any ancestor element of the <sca:implementation>  
1248 element. The effective list of implementation intents is the union of intents specified on the  
1249 implementation element and all its ancestors.

1250 In addition, one or more policySets can be specified directly by associating them with the implementation  
1251 of a component.

1252

```
1253 <component name="xs:NCName" ... >  
1254   <implementation.* ... policySets="listOfQNames">  
1255     ...  
1256   </implementation>  
1257   ...  
1258 </component>
```

1259 *Snippet 5-2: Example of policySets Associated with an implementation*

1260

1261 Snippet 55-2 shows how intents and policySets can be specified on a component. It is also possible to  
1262 specify intents and policySets within the implementation. How this is done is defined by the  
1263 implementation type.

1264 The intents and policy sets are specified on the <sca:implementation.\*> element within the component  
1265 type. This is important because intent and policy set definitions need to be able to specify that they  
1266 constrain an appropriate implementation type.

1267

```
1268 <componentType>  
1269   <implementation.* requires="listOfQNames" policySets="listOfQNames">  
1270     ...  
1271   </implementation>  
1272   ...
```

1273 `</componentType>`

1274 *Snippet 5-3: intents and policySets Constraining an implementation*

1275

1276 When applying policies, the intents attached to the implementation are added to the intents attached to  
1277 the using component. For the explicitly listed policySets, the list in the component can override policySets  
1278 from the componentType.

1279 Some implementation intents are targeted at `<binding/>` elements rather than at `<implementation/>`  
1280 elements. This occurs in cases where there is a need to influence the operation of the binding  
1281 implementation code rather than the code directly related to the implementation itself. Implementation  
1282 elements of this kind will have a `@constrains` attribute pointing to a binding element, with a `@intentType`  
1283 of "implementation".

## 1284 5.1 Natively Supported Intents

1285 Each implementation type (e.g. `<sca:implementation.java>` or `<sca:implementation.bpel>`) has an  
1286 **implementation type definition** within the SCA Domain. An implementation type definition is declared  
1287 using an `implementationType` element within a `<definitions/>` declaration. The pseudo-schema for the  
1288 `implementationType` element is shown in Snippet 55-4:

1289

```
1290 <implementationType type="QName"  
1291 alwaysProvides="listOfQNames"? mayProvide="listOfQNames"? />
```

1292 *Snippet 5-4: implementationType Pseudo-Schema*

1293

1294 The implementation Type element has the following attributes:

- 1295 • **name : QName (1..1)** - the name of the implementationType. The implementationType name attribute  
1296 MUST be the QName of an XSD global element definition used for implementation elements of that  
1297 type. [POL50001] For example: "sca:implementation.java".
- 1298 • **alwaysProvides : list of QNames (0..1)** - a set of intents. The intents in the alwaysProvides set are  
1299 always provided by this implementation type, whether the intents are attached to the using  
1300 component or not.
- 1301 • **mayProvide : list of QNames (0..1)** - a set of intents. The intents in the mayProvide set are provided  
1302 by this implementation type if the intent in question is attached to the using component.

## 1303 5.2 Writing PolicySets for Implementation Policies

1304 The `@appliesTo` and `@attachTo` attributes for a policySet take an XPath expression that is applied to a  
1305 service, reference, binding or an implementation element. For implementation policies, in most cases, all  
1306 that is needed is the QName of the implementation type. Implementation policies can be expressed using  
1307 any policy language (which is to say, any configuration language). For example, XACML or EJB-style  
1308 annotations can be used to declare authorization policies. Other capabilities could be configured using  
1309 completely proprietary configuration formats.

1310 For example, a policySet declared to turn on trace-level logging for a BPEL component could be declared  
1311 as is Snippet 55-5:

1312

```
1313 <policySet name="loggingPolicy" provides="acme:logging.trace"  
1314 appliesTo="//sca:implementation.bpel" ...>  
1315 <acme:processLogging level="3"/>  
1316 </policySet>
```

1317 *Snippet 5-5: Example policySet Applied to implementation.bpel*

1318 **5.2.1 Non WS-Policy Examples**

1319 Authorization policies expressed in XACML could be used in the framework in two ways:

1320 1. Embed XACML expressions directly in the PolicyAttachment element using the extensibility elements  
1321 discussed above, or

1322 2. Define WS-Policy assertions to wrap XACML expressions.

1323 For EJB-style authorization policy, the same approach could be used:

1324 1. Embed EJB-annotations in the PolicyAttachment element using the extensibility elements discussed  
1325 above, or

1326 2. Use the WS-Policy assertions defined as wrappers for EJB annotations.

---

## 1327 6 Roles and Responsibilities

1328 There are 4 roles that are significant for the SCA Policy Framework. The following is a list of the roles and  
1329 the artifacts that the role creates:

- 1330 • Policy Administrator – policySet definitions and intent definitions
- 1331 • Developer – Implementations and component types
- 1332 • Assembler - Composites
- 1333 • Deployer – Composites and the SCA Domain (including the logical Domain-level composite)

### 1334 6.1 Policy Administrator

1335 An intent represents a requirement that a developer or assembler can make, which ultimately have to be  
1336 satisfied at runtime. The full definition of the requirement is the informal text description in the intent  
1337 definition.

1338 The **policy administrator**'s job is to both define the intents that are available and to define the policySets  
1339 that represent the concrete realization of those informal descriptions for some set of binding type or  
1340 implementation types. See the sections on intent and policySet definitions for the details of those  
1341 definitions.

### 1342 6.2 Developer

1343 When it is possible for a component to be written without assuming a specific binding type for its services  
1344 and references, then the **developer** uses intents to specify requirements in a binding neutral way.

1345 If the developer requires a specific binding type for a component, then the developer can specify bindings  
1346 and policySets with the implementation of the component. Those bindings and policySets will be  
1347 represented in the component type for the implementation (although that component type might be  
1348 generated from the implementation).

1349 If any of the policySets used for the implementation include intentMaps, then the default choice for the  
1350 intentMap can be overridden by an assembler or deployer by requiring a qualified intent that is present in  
1351 the intentMap.

### 1352 6.3 Assembler

1353 An **assembler** creates composites. Because composites are implementations, an assembler is like a  
1354 developer, except that the implementations created by an assembler are composites made up of other  
1355 components wired together. So, like other developers, the assembler can specify intents or bindings or  
1356 policySets on any service or reference of the composite.

1357 However, in addition the definition of composite-level services and references, it is also possible for the  
1358 assembler to use the policy framework to further configure components within the composite. The  
1359 assembler can add additional requirements to any component's services or references or to the  
1360 component itself (for implementation policies). The assembler can also override the bindings or  
1361 policySets used for the component. See the assembly specification's description of overriding rules for  
1362 details on overriding.

1363 As a shortcut, an assembler can also specify intents and policySets on any element in the composite  
1364 definition, which has the same effect as specifying those intents and policySets on every applicable  
1365 binding or implementation below that element (where applicability is determined by the @appliesTo  
1366 attribute of the policySet definition or the @constrains attribute of the intent definition).

## 1367 **6.4 Deployer**

1368 A **deployer** deploys implementations (typically composites) into the SCA Domain. It is the  
1369 deployers job to make the final decisions about all configurable aspects of an implementation that is to be  
1370 deployed and to make sure that all intents are satisfied.

1371 If the deployer determines that an implementation is correctly configured as it is, then the implementation  
1372 can be deployed directly. However, more typically, the deployer will create a new composite, which  
1373 contains a component for each implementation to be deployed along with any changes to the bindings or  
1374 policySets that the deployer desires.

1375 When the deployer is determining whether the existing list of policySets is correct for a component, the  
1376 deployer needs to consider both the explicitly listed policySets as well as the policySets that will be  
1377 chosen according to the algorithm specified in [Guided Selection of PolicySets using Intents](#).

---

## 1378 7 Security Policy

1379 The SCA Security Model provides SCA developers the flexibility to specify the necessary level of security  
1380 protection for their components to satisfy business requirements without the burden of understanding  
1381 detailed security mechanisms.

1382 The SCA Policy framework distinguishes between two types of policies: *interaction policy* and  
1383 *implementation policy*. Interaction policy governs the communications between clients and service  
1384 providers and typically applies to Services and References. In the security space, interaction policy is  
1385 concerned with client and service provider authentication and message protection requirements.  
1386 Implementation policy governs security constraints on service implementations and typically applies to  
1387 Components. In the security space, implementation policy concerns include access control, identity  
1388 delegation, and other security quality of service characteristics that are pertinent to the service  
1389 implementations.

1390 The SCA security interaction policy can be specified via intents or policySets. Intents represent security  
1391 quality of service requirements at a high abstraction level, independent from security protocols, while  
1392 policySets specify concrete policies at a detailed level, which are typically security protocol specific.

1393 The SCA security policy can be specified either in an SCA composite or by using the External Policy  
1394 Attachment Mechanism or by annotations in the implementation code. Language-specific annotations are  
1395 described in the respective language Client and Implementation specifications.

### 1396 7.1 Security Policy Intents

1397 The SCA security specification defines the following intents to specify interaction policy:

1398 serverAuthentication, clientAuthentication, confidentiality, and integrity.

- 1399 • **serverAuthentication** – When *serverAuthentication* is present, an SCA runtime MUST ensure that  
1400 the server is authenticated by the client. [POL70013]
- 1401 • **clientAuthentication** – When *clientAuthentication* is present, an SCA runtime MUST ensure that the  
1402 client is authenticated by the server. [POL70014]
- 1403 • **authentication** – this is a profile intent that requires only clientAuthentication. It is included for  
1404 backwards compatibility.
- 1405 • **mutualAuthentication** – this is a profile intent that includes the serverAuthentication and the  
1406 clientAuthentication intents just described.
- 1407 • **confidentiality** – the confidentiality intent is used to indicate that the contents of a message are  
1408 accessible only to those authorized to have access (typically the service client and the service  
1409 provider). A common approach is to encrypt the message, although other methods are possible.  
1410 When confidentiality is present, an SCA Runtime MUST ensure that only authorized entities can view  
1411 the contents of a message. [POL70009]
- 1412 • **integrity** – the integrity intent is used to indicate that assurance is that the contents of a message  
1413 have not been tampered with and altered between sender and receiver. A common approach is to  
1414 digitally sign the message, although other methods are possible. When *integrity* is present, an SCA  
1415 Runtime MUST ensure that the contents of a message are not altered. [POL70010]

1416 The formal definitions of these intents are in the [Intent Definitions appendix](#).

### 1417 7.2 Interaction Security Policy

1418 Any one of the three security intents can be further qualified to specify more specific business  
1419 requirements. Two qualifiers are defined by the SCA security specification: transport and message, which  
1420 can be applied to any of the above three intent's.

## 1421 7.2.1 Qualifiers

1422 **transport** – the transport qualifier specifies that the qualified intent is realized at the transport or transfer  
1423 layer of the communication protocol, such as HTTPS. When a serverAuthentication, clientAuthentication,  
1424 confidentiality or integrity intent is qualified by message, an SCA Runtime MUST delegate  
1425 serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the message layer  
1426 of the communication protocol. [POL70011]

1427 **message** – the message qualifier specifies that the qualified intent is realized at the message level of the  
1428 communication protocol. When a serverAuthentication, clientAuthentication, confidentiality or integrity  
1429 intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication,  
1430 clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication  
1431 protocol. [POL70012]

1432

1433 Snippet 77-1 shows the usage of intents and qualified intents.

1434

```
1435 <composite name="example" requires="confidentiality">  
1436   <service name="foo"/>  
1437   ...  
1438   <reference name="bar" requires="confidentiality.message"/>  
1439 </composite>
```

1440 *Snippet 7-1: Example using Qualified Intents*

1441

1442 In this case, the composite declares that all of its services and references have to guarantee  
1443 confidentiality in their communication by setting requires="confidentiality". This applies to the "foo"  
1444 service. However, the "bar" reference further qualifies that requirement to specifically require message-  
1445 level security by setting requires="confidentiality.message".

## 1446 7.3 Implementation Security Policy Intent

1447 The SCA Security specification defines the **authorization** intent to specify implementation policy.

1448 **authorization** – the authorization intent is used to indicate that a client needs to be authorized before  
1449 being allowed to use the service. Being authorized means that a check is made as to whether any  
1450 policies apply to the client attempting to use the service, and if so, those policies govern whether or not  
1451 the client is allowed access. When **authorization** is present, an SCA Runtime MUST ensure that the client  
1452 is authorized to use the service. [POL70001]

1453 This unqualified authorization intent implies that basic "Subject-Action-Resource" authorization support is  
1454 required, where Subject may be as simple as a single identifier representing the identity of the client,  
1455 Action may be a single identifier representing the operation the client intends to apply to the Resource,  
1456 and the Resource may be a single identifier representing the identity of the Resource to which the Action  
1457 is intended to be applied.



---

## 8 Reliability Policy

1458

1459 Failures can affect the communication between a service consumer and a service provider.

1460 Depending on the characteristics of the binding, these failures could cause messages to be redelivered,  
1461 delivered in a different order than they were originally sent out or even worse, could cause messages to  
1462 be lost. Some transports like JMS provide built-in reliability features such as “at least once” and “exactly  
1463 once” message delivery. Other transports like HTTP need to have additional layers built on top of them to  
1464 provide some of these features.

1465 The events that occur due to failures in communication can affect the outcome of the service invocation.  
1466 For an implementation of a stock trade service, a message redelivery could result in a new trade. A client  
1467 (i.e. consumer) of the same service could receive a fault message if trade orders are not delivered to the  
1468 service implementation in the order they were sent out. In some cases, these failures could have dramatic  
1469 consequences.

1470 An SCA developer can anticipate some types of failures and work around them in service  
1471 implementations. For example, the implementation of a stock trade service could be designed to support  
1472 duplicate message detection. An implementation of a purchase order service could have built in logic that  
1473 orders the incoming messages. In these cases, service implementations don't need the binding layers to  
1474 provide these reliability features (e.g. duplicate message detection, message ordering). However, this  
1475 comes at a cost: extra complexity is built in the service implementation. Along with business logic, the  
1476 service implementation has additional logic that handles these failures.

1477 Although service implementations can work around some of these types of failures, it is worth noting that  
1478 workarounds are not always possible. A message can be lost or expire even before it is delivered to the  
1479 service implementation.

1480 Instead of handling some of these issues in the service implementation, a better way is to use a binding  
1481 or a protocol that supports reliable messaging. This is better, not just because it simplifies application  
1482 development, it can also lead to better throughput. For example, there is less need for application-level  
1483 acknowledgement messages. A binding supports reliable messaging if it provides features such as  
1484 message delivery guarantees, duplicate message detection and message ordering.

1485 It is very important for the SCA developer to be able to require, at design-time, a binding or protocol that  
1486 supports reliable messaging. SCA defines a set of policy intents that can be used for specifying reliable  
1487 messaging Quality of Service requirements. These reliable messaging intents establish a contract  
1488 between the binding layer and the application layer (i.e. service implementation or the service consumer  
1489 implementation) (see below).

### 8.1 Reliability Policy Intents

1490 Based on the use-cases described above, the following policy intents are defined:

1491 1. **atLeastOnce** - The binding implementation guarantees that a message that is successfully sent by a  
1492 service consumer is delivered to the destination (i.e. service implementation). The message could be  
1493 delivered more than once to the service implementation. **When *atLeastOnce* is present, an SCA  
1494 Runtime MUST deliver a message to the destination service implementation, and MAY deliver  
1495 duplicates of a message to the service implementation. [POL80001]**

1496 The binding implementation guarantees that a message that is successfully sent by a service  
1497 implementation is delivered to the destination (i.e. service consumer). The message could be  
1498 delivered more than once to the service consumer.  
1499

1500 2. **atMostOnce** - The binding implementation guarantees that a message that is successfully sent by a  
1501 service consumer is not delivered more than once to the service implementation. The binding  
1502 implementation does not guarantee that the message is delivered to the service implementation.  
1503 **When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service**

1504 implementation, and MUST NOT deliver duplicates of a message to the service implementation.  
1505 [POL80002]

1506 The binding implementation guarantees that a message that is successfully sent by a service  
1507 implementation is not delivered more than once to the service consumer. The binding implementation  
1508 does not guarantee that the message is delivered to the service consumer.

1509 3. **ordered** – The binding implementation guarantees that the messages sent by a service client via a  
1510 single service reference are delivered to the target service implementation in the order in which they  
1511 were sent by the service client. This intent does not guarantee that messages that are sent by a  
1512 service client are delivered to the service implementation. Note that this intent has nothing to say  
1513 about the ordering of messages sent via different service references by a single service client, even if  
1514 the same service implementation is targeted by each of the service references. **When ordered is  
1515 present, an SCA Runtime MUST deliver messages sent by a single source to a single destination  
1516 service implementation in the order that the messages were sent by that source.** [POL80003]

1517 For service interfaces that involve messages being sent back from the service implementation to the  
1518 service client (eg. a service with a callback interface), for this intent, the binding implementation  
1519 guarantees that the messages sent by the service implementation over a given wire are delivered to  
1520 the service client in the order in which they were sent by the service implementation. This intent does  
1521 not guarantee that messages that are sent by the service implementation are delivered to the service  
1522 consumer.

1523 4. **exactlyOnce** - The binding implementation guarantees that a message sent by a service consumer is  
1524 delivered to the service implementation. Also, the binding implementation guarantees that the  
1525 message is not delivered more than once to the service implementation. **When exactlyOnce is  
1526 present, an SCA Runtime MUST deliver a message to the destination service implementation and  
1527 MUST NOT deliver duplicates of a message to the service implementation.** [POL80004]

1528 The binding implementation guarantees that a message sent by a service implementation is delivered  
1529 to the service consumer. Also, the binding implementation guarantees that the message is not  
1530 delivered more than once to the service consumer.

1531 NOTE: This is a profile intent, which is composed of *atLeastOnce* and *atMostOnce*.

1532 This is the most reliable intent since it guarantees the following:

- 1533 – message delivery – all the messages sent by a sender are delivered to the service  
1534 implementation (i.e. Java class, BPEL process, etc.).
- 1535 – duplicate message detection and elimination – a message sent by a sender is not processed  
1536 more than once by the service implementation.

1537 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1538 How can a binding implementation guarantee that a message that it receives is delivered to the service  
1539 implementation? One way to do it is by persisting the message and keeping redelivering it until it is  
1540 processed by the service implementation. That way, if the system crashes after delivery but while  
1541 processing it, the message will be redelivered on restart and processed again. Since a message could be  
1542 delivered multiple times to the service implementation, this technique usually requires the service  
1543 implementation to perform duplicate message detection. However, that is not always possible. Often  
1544 times service implementations that perform critical operations are designed without having support for  
1545 duplicate message detection. Therefore, they cannot *process* an incoming message more than once.

1546 Also, consider the scenario where a message is delivered to a service implementation that does not  
1547 handle duplicates - the system crashes after a message is delivered to the service implementation but  
1548 before it is completely processed. Does the underlying layer redeliver the message on restart? If it did  
1549 that, there is a risk that some critical operations (e.g. sending out a JMS message or updating a DB table)  
1550 will be executed again when the message is processed. On the other hand, if the underlying layer does  
1551 not redeliver the message, there is a risk that the message is never completely processed.

1552 This issue cannot be safely solved unless all the critical operations performed by the service

1553 implementation are running in a transaction. Therefore, *exactlyOnce* cannot be assured without involving  
1554 the service implementation. In other words, an *exactlyOnce* message delivery does not guarantee  
1555 *exactlyOnce* message processing unless the service implementation is transactional. It's worth noting that  
1556 this is a necessary condition but not sufficient. The underlying layer (e.g. binding implementation,  
1557 container) would have to ensure that a message is not redelivered to the service implementation after the  
1558 transaction is committed. As an example, a way to ensure it when the binding uses JMS is by making  
1559 sure the operation that acknowledges the message is executed in the same transaction the service  
1560 implementation is running in.

## 1561 **8.2 End-to-end Reliable Messaging**

1562 Failures can occur at different points in the message path: in the binding layer on the sender side, in the  
1563 transport layer or in the binding layer on the receiver side. The SCA service developer doesn't really care  
1564 where the failure occurs. Whether a message was lost due to a network failure or due to a crash of the  
1565 machine where the service is deployed, is not that important. What is important is that the contract  
1566 between the application layer (i.e. service implementation or service consumer) and the binding layer is  
1567 not violated (e.g. a message that was successfully transmitted by a sender is always delivered to the  
1568 destination; a message that was successfully transmitted by a sender is not delivered more than once to  
1569 the service implementation, etc). It is worth noting that the binding layer could throw an exception when a  
1570 sender (e.g. service consumer, service implementation) sends a message out. This is not considered a  
1571 successful message transmission.

1572 In order to ensure the semantics of the reliable messaging intents, the entire message path, which is  
1573 composed of the binding layer on the client side, the transport layer and the binding layer on the service  
1574 side, has to be reliable.

---

## 1575 9 Transactions

1576 SCA recognizes that the presence or absence of infrastructure for ACID transaction coordination has a  
1577 direct effect on how business logic is coded. In the absence of ACID transactions, developers have to  
1578 provide logic that coordinates the outcome, compensates for failures, etc. In the presence of ACID  
1579 transactions, the underlying infrastructure is responsible for ensuring the ACID nature of all interactions.  
1580 SCA provides declarative mechanisms for describing the transactional environment needed by the  
1581 business logic.

1582 Components that use a synchronous interaction style can be part of a single, distributed ACID transaction  
1583 within which all transaction resources are coordinated to either atomically commit or rollback. The  
1584 transmission or receipt of oneway messages can, depending on the transport binding, be coordinated as  
1585 part of an ACID transaction as illustrated in the “*OneWay Invocations*” section below. Well-known, higher-  
1586 level patterns such as store-and-forward queuing can be accomplished by composing transacted one-  
1587 way messages with reliable-messaging policies.

1588 This document describes the set of abstract policy intents – both implementation intents and interaction  
1589 intents – that can be used to describe the requirements on a concrete service component and binding  
1590 respectively.

### 1591 9.1 Out of Scope

1592 The following topics are outside the scope of this document:

- 1593 • The means by which transactions are created, propagated and established as part of an execution  
1594 context. These are details of the SCA runtime provider and binding provider.
- 1595 • The means by which a transactional resource manager (RM) is accessed. These include, but are not  
1596 restricted to:
  - 1597 – abstracting an RM as an `sca:component`
  - 1598 – accessing an RM directly in a language-specific and RM-specific fashion
  - 1599 – abstracting an RM as an `sca:binding`

### 1600 9.2 Common Transaction Patterns

1601 In the absence of any transaction policies there is no explicit transactional behavior defined for the SCA  
1602 service component or the interactions in which it is involved and the transactional behavior is  
1603 environment-specific. An SCA runtime provider can choose to define an out of band default transactional  
1604 behavior that applies in the absence of any transaction policies.

1605 Environment-specific default transactional behavior can be overridden by specifying transactional intents  
1606 described in this document. The most common transaction patterns can be summarized:

1607 **Managed, shared global transaction pattern** – the service always runs in a global transaction context  
1608 regardless of whether the requester runs under a global transaction. If the requester does run under a  
1609 transaction, the service runs under the same transaction. Any outbound, synchronous request-response  
1610 messages will – unless explicitly directed otherwise – propagate the service’s transaction context. This  
1611 pattern offers the highest degree of data integrity by ensuring that any transactional updates are  
1612 committed atomically

1613 **Managed, local transaction pattern** – the service always runs in a managed local transaction context  
1614 regardless of whether the requester runs under a transaction. Any outbound messages will not propagate  
1615 any transaction context. This pattern is advisable for services that wish the SCA runtime to demarcate  
1616 any resource manager local transactions and do not require the overhead of atomicity.

1617 The use of transaction policies to specify these patterns is illustrated later in Table 99-2.

## 1618 9.3 Summary of SCA Transaction Policies

1619 This specification defines implementation and interaction policies that relate to transactional QoS in  
1620 components and their interactions. The SCA transaction policies are specified as intents which represent  
1621 the transaction quality of service behavior offered by specific component implementations or bindings.

1622 SCA transaction policy can be specified either in an SCA composite or annotatively in the implementation  
1623 code. Language-specific annotations are described in the respective language binding specifications, for  
1624 example the [SCA Java Common Annotations and APIs specification](#) [SCA-Java-Annotations].

1625 This specification defines the following implementation transaction policies:

- 1626 • `managedTransaction` – Describes the service component’s transactional environment.
- 1627 • `transactedOneWay` and `immediateOneWay` – two mutually exclusive intents that describe whether  
1628 the SCA runtime will process `OneWay` messages immediately or will enqueue (from a client  
1629 perspective) and dequeue (from a service perspective) a `OneWay` message as part of a global  
1630 transaction.

1631 This specification also defines the following interaction transaction policies:

- 1632 • `propagatesTransaction` and `suspendsTransaction` – two mutually exclusive intents that describe  
1633 whether the SCA runtime propagates any transaction context to a service or reference on a  
1634 synchronous invocation.

1635 Finally, this specification defines a profile intent called `managedSharedTransaction` that combines the  
1636 `managedTransaction` intent and the `propagatesTransaction` intent so that the ***managed, shared global***  
1637 ***transaction pattern*** is easier to configure.

## 1638 9.4 Global and local transactions

1639 This specification describes “managed transactions” in terms of either “global” or “local” transactions. The  
1640 “managed” aspect of managed transactions refers to the transaction environment provided by the SCA  
1641 runtime for the business component. Business components can interact with other business components  
1642 and with resource managers. The managed transaction environment defines the transactional context  
1643 under which such interactions occur.

### 1644 9.4.1 Global transactions

1645 From an SCA perspective, a global transaction is a unit of work scope within which transactional work is  
1646 atomic. If multiple transactional resource managers are accessed under a global transaction then the  
1647 transactional work is coordinated to either atomically commit or rollback regardless using a 2PC protocol.  
1648 A global transaction can be propagated on synchronous invocations between components – depending  
1649 on the interaction intents described in this specification - such that multiple, remote service providers can  
1650 execute distributed requests under the same global transaction.

### 1651 9.4.2 Local transactions

1652 From a resource manager perspective a resource manager local transaction (RMLT) is simply the  
1653 absence of a global transaction. But from an SCA perspective it is not enough to simply declare that a  
1654 piece of business logic runs without a global transaction context. Business logic might need to access  
1655 transactional resource managers without the presence of a global transaction. The business logic  
1656 developer still needs to know the expected semantic of making one or more calls to one or more resource  
1657 managers, and needs to know when and/or how the resource managers local transactions will be  
1658 committed. The term *local transaction containment* (LTC) is used to describe the SCA environment where  
1659 there is no global transaction. The boundaries of an LTC are scoped to a remotable service provider  
1660 method and are not propagated on invocations between components. Unlike the resources in a global  
1661 transaction, RMLTs coordinated within a LTC can fail independently.

1662

- 1663 The two most common patterns for components using resource managers outside a global transaction  
 1664 are:
- 1665 • The application desires each interaction with a resource manager to commit after every interaction.  
 1666 This is the default behavior provided by the **noManagedTransaction** policy (defined below in  
 1667 “[Transaction implementation policy](#)”) in the absence of explicit use of RMLT verbs by the application.
  - 1668 • The application desires each interaction with a resource manager to be part of an extended local  
 1669 transaction that is committed at the end of the method. This behavior is specified by the  
 1670 **managedTransaction.local** policy (defined below in “[Transaction implementation policy](#)”).

1671 While an application can use interfaces provided by the resource adapter to explicitly demarcate resource  
 1672 manager local transactions (RMLT), this is a generally undesirable burden on applications, which typically  
 1673 prefer all transaction considerations to be managed by the SCA runtime. In addition, once an application  
 1674 codes to a resource manager local transaction interface, it might never be redeployed with a different  
 1675 transaction environment since local transaction interfaces might not be used in the presence of a global  
 1676 transaction. This specification defines intents to support both these common patterns in order to provide  
 1677 portability for applications regardless of whether they run under a global transaction or not.

## 1678 9.5 Transaction implementation policy

### 1679 9.5.1 Managed and non-managed transactions

1680 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents describe the  
 1681 transactional environment needed by a service component or composite. SCA provides transaction  
 1682 environments that are managed by the SCA runtime in order to remove the burden of coding transaction  
 1683 APIs directly into the business logic. The **managedTransaction** and **noManagedTransaction** intents  
 1684 can be attached to the `sca:composite` or `sca:componentType` elements.

1685 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents are defined as  
 1686 follows:

- 1687 • **managedTransaction** – a managed transaction environment is necessary in order to run this  
 1688 component. The specific type of managedTransaction needed is not constrained. The valid qualifiers  
 1689 for this intent are mutually exclusive.
  - 1690 – **managedTransaction.global** – There has to be an atomic transaction in order to run this  
 1691 component. For a component marked with **managedTransaction.global**, the SCA runtime  
 1692 MUST ensure that a global transaction is present before dispatching any method on the  
 1693 component. [POL90003] The SCA runtime uses any transaction propagated from the client  
 1694 or else begins and completes a new transaction. See the **propagatesTransaction** intent  
 1695 below for more details.
  - 1696 – **managedTransaction.local** – indicates that the component cannot tolerate running as part  
 1697 of a global transaction. A component marked with **managedTransaction.local** MUST run  
 1698 within a local transaction containment (LTC) that is started and ended by the SCA runtime.  
 1699 [POL90004] Any global transaction context that is propagated to the hosting SCA runtime is  
 1700 not visible to the target component. Any interaction under this policy with a resource manager  
 1701 is performed in an extended resource manager local transaction (RMLT). Upon successful  
 1702 completion of the invoked service method, any RMLTs are implicitly requested to commit by  
 1703 the SCA runtime. Note that, unlike the resources in a global transaction, RMLTs so  
 1704 coordinated in a LTC can fail independently. If the invoked service method completes with a  
 1705 non-business exception then any RMLTs are implicitly rolled back by the SCA runtime. In this  
 1706 context a business exception is any exception that is declared on the component interface  
 1707 and is therefore anticipated by the component implementation. The manner in which  
 1708 exceptions are declared on component interfaces is specific to the interface type – for  
 1709 example, Java interface types declare Java exceptions, WSDL interface types define  
 1710 `wsdl:faults`. Local transactions MUST NOT be propagated outbound across remotable  
 1711 interfaces. [POL90006]

1712 • **noManagedTransaction** – indicates that the component runs without a managed transaction, under  
1713 neither a global transaction nor an LTC. A transaction that is propagated to the hosting SCA runtime  
1714 MUST NOT be joined by the hosting runtime on behalf of a component marked with  
1715 noManagedTransaction. [POL90007] When interacting with a resource manager under this policy, the  
1716 application (and not the SCA runtime) is responsible for controlling any resource manager local  
1717 transaction boundaries, using resource-provider specific interfaces (for example a Java  
1718 implementation accessing a JDBC provider has to choose whether a Connection is set to  
1719 autoCommit(true) or else it has to call the Connection commit or rollback method). SCA defines no  
1720 APIs for interacting with resource managers.

1721 • **(absent)** – The absence of a transaction implementation intent leads to runtime-specific behavior. A  
1722 runtime that supports global transaction coordination can choose to provide a default behavior that is  
1723 the managed, shared global transaction pattern but it is not mandated to do so.

1724 The formal definitions of these intents are in the [Intent Definitions appendix](#).

## 1725 9.5.2 OneWay Invocations

1726 When a client uses a reference and sends a OneWay message then any client transaction context is not  
1727 propagated. However, the OneWay invocation on the reference can itself be **transacted**. Similarly, from a  
1728 service perspective, any received OneWay message cannot propagate a transaction context but the  
1729 delivery of the OneWay message can be **transacted**. A **transacted** OneWay message is a one-way  
1730 message that - because of the capability of the service or reference binding - can be enqueued (from a  
1731 client perspective) or dequeued (from a service perspective) as part of a global transaction.

1732 SCA defines two mutually exclusive implementation intents, **transactedOneWay** and  
1733 **immediateOneWay**, that determine whether OneWay messages are transacted or delivered immediately.

1734 Either of these intents can be attached to the sca:service or sca:reference elements or they can be  
1735 attached to the sca:component element, indicating that the intent applies to any service or reference  
1736 element children.

1737 The intents are defined as follows:

1738 • **transactedOneWay** – When a reference is marked as transactedOneWay, any OneWay invocation  
1739 messages MUST be transacted as part of a client global transaction. [POL90008]  
1740 If the client component is not configured to run under a global transaction or if the binding does not  
1741 support transactional message sending, then a reference MUST NOT be marked as  
1742 transactedOneWay. [POL90009] If a service is marked as transactedOneWay, any OneWay  
1743 invocation message MUST be received from the transport binding in a transacted fashion, under the  
1744 target service's global transaction. [POL90010] The **transactedOneWay** intent MUST NOT be  
1745 attached to a request/response operation. [POL90028] The receipt of the message from the binding is  
1746 not committed until the service transaction commits; if the service transaction is rolled back the the  
1747 message remains available for receipt under a different service transaction. If the component is not  
1748 configured to run under a global transaction or if the binding does not support transactional message  
1749 receipt, then a service MUST NOT be marked as transactedOneWay. [POL90011]

1750 • **immediateOneWay** – When applied to a reference indicates that any OneWay invocation messages  
1751 MUST be sent immediately regardless of any client transaction. [POL90012] When applied to a  
1752 service indicates that any OneWay invocation MUST be received immediately regardless of any  
1753 target service transaction. [POL90013] The **immediateOneWay** intent MUST NOT be attached to a  
1754 request/response operation. [POL90029] The outcome of any transaction under which an  
1755 immediateOneWay message is processed has no effect on the processing (sending or receipt) of that  
1756 message.

1757 The absence of either intent leads to runtime-specific behavior. The SCA runtime can send or receive a  
1758 OneWay message immediately or as part of any sender/receiver transaction. The results of combining  
1759 this intent and the **managedTransaction** implementation policy of the component sending or receiving  
1760 the transacted OneWay invocation are summarized below in Table 99-1.

1761

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	If a transactedOneWay intent is combined with the managedTransaction.local or noManagedTransaction implementation intents for either a reference or a service then an error MUST be raised during deployment. [POL90027]
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 can send or receive a OneWay message immediately or as part of any sender/receiver transaction.

1762 Table 9-1 Transacted OneWay interaction intent

1763

1764 The formal definitions of these intents are in the [Intent Definitions appendix](#).

### 1765 9.5.3 Asynchronous Implementations

1766 SCA defines an intent called **asyncInvocation** that enables an SCA service to indicate that its  
1767 request/response operations are long running and therefore interactions with those operations really need  
1768 to be done asynchronously. The use of **asyncInvocation** with oneway operations is meaningless  
1769 because the one way operation is already asynchronous. Operations which implement this long running  
1770 behavior can make use of any transaction implementation intents on a component implementation or on  
1771 SCA references. However, implementations of operations which have long-running behaviour need to be  
1772 careful in how they use ACID transactions, which in general are not suited to operating over extended  
1773 time periods. Also see section 9.6.4 Interaction intents with asynchronous implementations for additional  
1774 considerations on the use of the **asyncInvocation** intent with transactions.

1775

## 1776 9.6 Transaction interaction policies

1777 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can be attached  
1778 either to an interface (e.g. Java annotation or WSDL attribute) or explicitly to an sca:service and  
1779 sca:reference XML element to describe how any client transaction context will be made available and  
1780 used by the target service component. Section 9.6.1 considers how these intents apply to service  
1781 elements and Section 9.6.2 considers how these intents apply to reference elements.

1782 The formal definitions of these intents are in the [Intent Definitions appendix](#).

### 1783 9.6.1 Handling Inbound Transaction Context

1784 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can be attached to  
1785 an sca:service XML element to describe how a propagated transaction context is handled by the SCA  
1786 runtime, prior to dispatching a service component. If the service requester is running within a transaction



1787 and the service interaction policy is to propagate that transaction, then the primary business effects of the  
1788 provider's operation are coordinated as part of the client's transaction – if the client rolls back its  
1789 transaction, then work associated with the provider's operation will also be rolled back. This allows clients  
1790 to know that no compensation business logic is necessary since transaction rollback can be used.

1791 These intents specify a contract that has to be implemented by the SCA runtime. This aspect of a  
1792 service component is most likely captured during application design. The **propagatesTransaction** or  
1793 **suspendsTransaction** intent can be attached to sca:service elements and their children. The intents are  
1794 defined as follows:

1795 • **propagatesTransaction** – A service marked with propagatesTransaction MUST be dispatched under  
1796 any propagated (client) transaction. [POL90015] Use of the **propagatesTransaction** intent on a  
1797 service implies that the service binding MUST be capable of receiving a transaction context.  
1798 [POL90016] However, it is important to understand that some binding/policySet combinations that  
1799 provide this intent for a service will *need* the client to propagate a transaction context.  
1800 In SCA terms, for a reference wired to such a service, this implies that the reference has to use either  
1801 the **propagatesTransaction** intent or a binding/policySet combination that does propagate a  
1802 transaction. If, on the other hand, the service does not *need* the client to provide a transaction (even  
1803 though it has the *capability* of joining the client's transaction), then some care is needed in the  
1804 configuration of the service. One approach to consider in this case is to use two distinct bindings on  
1805 the service, one that uses the **propagatesTransaction** intent and one that does not - clients that do  
1806 not propagate a transaction would then wire to the service using the binding without the  
1807 **propagatesTransaction** intent specified.

1808 • **suspendsTransaction** – A service marked with suspendsTransaction MUST NOT be dispatched  
1809 under any propagated (client) transaction. [POL90017]

1810 The absence of either interaction intent leads to runtime-specific behavior; the client is unable to  
1811 determine from transaction intents whether its transaction will be joined.

1812 The SCA runtime MUST ignore the propagatesTransaction intent for OneWay methods. [POL90025]

1813 These intents are independent from the implementation's **managedTransaction** intent and provides no  
1814 information about the implementation's transaction environment.

1815 The combination of these service interaction policies and the **managedTransaction** implementation  
1816 policy of the containing component completely describes the transactional behavior of an invoked service,  
1817 as summarized in Table 99-2:

1818

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 <b>managed, shared global transaction</b> pattern described in Common Transaction Patterns. This is equivalent to the managedSharedTransaction intent defined in section 9.6.3.
propagatesTransaction	managedTransaction.local or noManagedTransaction	A service <b>MUST NOT</b> be marked with "propagatesTransaction" if the component is marked with "managedTransaction.local" or with "noManagedTransaction" [POL90019]
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 <b>managed, local transaction</b> 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.

1819 *Table 9-2 Combining service transaction intents*

1820

1821 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A  
 1822 runtime that supports global transaction coordination can choose to provide a default behavior that is the  
 1823 managed, shared global transaction pattern.

1824 **9.6.2 Handling Outbound Transaction Context**

1825 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can also be attached  
 1826 to an sca:reference XML element to describe whether any client transaction context is propagated to a  
 1827 target service when a synchronous interaction occurs through the reference. These intents specify a  
 1828 contract that has to be implemented by the SCA runtime. This aspect of a service component is most  
 1829 likely captured during application design.

1830 Either the **propagatesTransaction** or **suspendsTransaction** intent can be attached to sca:service  
 1831 elements and their children. The intents are defined as defined in Section 9.6.1.

1832 When used as a reference interaction intent, the meaning of the qualifiers is as follows:

- 1833 • **propagatesTransaction** – When a reference is marked with propagatesTransaction, any transaction  
 1834 context under which the client runs **MUST** be propagated when the reference is used for a request-  
 1835 response interaction [POL90020] The binding of a reference marked with propagatesTransaction has

1836 to be capable of propagating a transaction context. The reference needs to be wired to a service that  
 1837 can join the client's transaction. For example, any service with an intent that @requires  
 1838 **propagatesTransaction** can always join a client's transaction. The reference consumer can then be  
 1839 designed to rely on the work of the target service being included in the caller's transaction.

1840 • **suspendsTransaction** – When a reference is marked with **suspendsTransaction**, any transaction  
 1841 context under which the client runs MUST NOT be propagated when the reference is used.  
 1842 [POL90022] The reference consumer can use this intent to ensure that the work of the target service  
 1843 is not included in the caller's transaction. .

1844 • The absence of either interaction intent leads to runtime-specific behavior. The SCA runtime can  
 1845 choose whether or not to propagate any client transaction context to the referenced service,  
 1846 depending on the SCA runtime capability.

1847 These intents are independent from the client's **managedTransaction** implementation intent. The  
 1848 combination of the interaction intent of a reference and the **managedTransaction** implementation policy  
 1849 of the containing component completely describes the transactional behavior of a client's invocation of a  
 1850 service. Table 99-3 summarizes the results of the combination of either of these interaction intents with  
 1851 the **managedTransaction** implementation policy of the containing component.

1852

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 <b>managed, shared global transaction</b> pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	A reference MUST NOT be marked with <b>propagatesTransaction</b> if component is marked with "ManagedTransaction.local" or with "noManagedTransaction" [POL90023]
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 <b>managed, local transaction</b> pattern described in Common Transaction Patterns.

1853 *Table 9-3 Transaction propagation reference intents*

1854

1855 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A  
 1856 runtime that supports global transaction coordination can choose to provide a default behavior that is the  
 1857 managed, shared global transaction pattern.

1858 Table 99-4 shows the valid combination of interaction and implementation intents on the client and  
 1859 service that result in a single global transaction being used when a client invokes a service through a  
 1860 reference.

1861

managedTransaction (client implementation intent)	reference interaction intent	service interaction intent	managedTransaction (service implementation intent)

managedTransaction.global	propagatesTransaction	propagatesTransaction	managedTransaction.global
---------------------------	-----------------------	-----------------------	---------------------------

1862 *Table 9-4 Intents for end-to-end transaction propagation*

1863

1864 Transaction context **MUST NOT** be propagated on OneWay messages. [POL90024] The SCA runtime  
 1865 ignores **propagatesTransaction** for OneWay operations.

### 1866 9.6.3 Combining implementation and interaction intents

1867 The **managed, local transaction pattern** can be configured quite easily by combining the  
 1868 managedTransaction.global intent with the propagatesTransaction intent. This is illustrated in Section 9.2  
 1869 Common Transaction Patterns. In order to enable easier configuration of this pattern, a profile intent  
 1870 called managedSharedTransaction is defined as in section C.1 Intent Definitions.

### 1871 9.6.4 Interaction intents with asynchronous implementations

1872 SCA defines an intent called **asyncInvocation** that enables an SCA service to indicate that its  
 1873 request/response operations are long running and therefore interactions with the service really need to be  
 1874 done asynchronously. Any of the transaction interaction intents can be used with an asynchronous  
 1875 implementation except for the **propagatesTransaction** intent. Due to the long running nature of this kind  
 1876 of implementation, inbound global transaction context cannot be propagated to the component  
 1877 implementation. As a result, the **propagatesTransaction** intent is mutually exclusive with the  
 1878 **asyncInvocation** intent. The **asyncInvocation** intent and the **propagatesTransaction** intent **MUST**  
 1879 **NOT** be applied to the same service or reference operation. [POL90030] When the **asyncInvocation**  
 1880 intent is applied to an SCA service, the SCA runtime **MUST** behave as if the **suspendsTransaction**  
 1881 intent is also applied to the service. [POL90031]

1882

### 1883 9.6.5 Web Services Binding for propagatesTransaction policy

1884 Snippet 99-1 shows a policySet that provides the **propagatesTransaction** intent and applies to a Web  
 1885 service binding (binding.ws). When used on a service, this policySet would require the client to send a  
 1886 transaction context using the mechanisms described in the [Web Services Atomic Transaction](#) [WS-  
 1887 AtomicTransaction] specification.

1888

```

1889 <policySet name="JoinsTransactionWS" provides="sca:propagatesTransaction"
1890           appliesTo="//sca:binding.ws">
1891   <wsp:Policy>
1892     <wsat:ATAssertion
1893       xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"/>
1894   </wsp:Policy>
1895 </policySet>

```

1896 *Snippet 9-1: Example policySet Providing propagatesTransaction*

1897

## 10 Miscellaneous Intents

1898 The following are standard intents that apply to bindings and are not related to either security, reliable  
1899 messaging or transactionality:

- 1900
- 1901 • **SOAP** – The SOAP intent specifies that the SOAP messaging model is used for delivering messages.  
1902 It does not require the use of any specific transport technology for delivering the messages, so for  
1903 example, this intent can be supported by a binding that sends SOAP messages over HTTP, bare  
1904 TCP or even JMS. If the intent is attached in an unqualified form then any version of SOAP is  
1905 acceptable. Standard mutually exclusive qualified intents also exist for SOAP.1\_1 and SOAP.1\_2,  
1906 which specify the use of versions 1.1 or 1.2 of SOAP respectively. When SOAP is present, an SCA  
1907 Runtime MUST use the SOAP messaging model to deliver messages. [POL100001] When a SOAP  
1908 intent is qualified with 1\_1 or 1\_2, then SOAP version 1.1 or SOAP version 1.2 respectively MUST be  
used to deliver messages. [POL100002]
  - 1909 • **JMS** – The JMS intent does not specify a wire-level transport protocol, but instead requires that  
1910 whatever binding technology is used, the messages are able to be delivered and received via the  
1911 JMS API. When JMS is present, an SCA Runtime MUST ensure that the binding used to send and  
1912 receive messages supports the JMS API. [POL100003]
  - 1913 • **noListener** – This intent can only be used within the @requires attribute of a reference. The  
1914 noListener intent MUST only be declared on a @requires attribute of a reference. [POL100004] It  
1915 states that the client is not able to handle new inbound connections. It requires that the binding and  
1916 callback binding be configured so that any response (or callback) comes either through a back  
1917 channel of the connection from the client to the server or by having the client poll the server for  
1918 messages. When noListener is present, an SCA Runtime MUST not establish any connection from a  
1919 service to a client. [POL100005] An example policy assertion that would guarantee this is a WS-  
1920 Policy assertion that applies to the <binding.ws> binding, which requires the use of WS-Addressing  
1921 with anonymous responses (e.g. <wsaw:Anonymous>required</wsaw:Anonymous>” – see  
1922 <http://www.w3.org/TR/ws-addr-wsdl/#anonelement>).
  - 1923 • **asyncInvocation** – This intent can be attached to a request/response operation or a complete  
1924 interface, indicating that the request/response operation(s) are long-running [SCA-Assembly]. The  
1925 SCA Runtime MUST ignore the asyncInvocation intent for one way operations. [POL100007] It is also  
1926 possible for a service to set the asyncInvocation intent when using an interface which is not marked  
1927 with the asyncInvocation intent. This can be useful when reusing an existing interface definition that  
1928 does not contain SCA information.
  - 1929 • **EJB** - The EJB intent specifies that whatever wire-level transport technology is specified the  
1930 messages are able to be delivered and received via the EJB API. When EJB is present, an SCA  
1931 Runtime MUST ensure that the binding used to send and receive messages supports the EJB API.  
1932 [POL100006]

1933 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948

---

## 11 Conformance

The XML schema available at the namespace URI, defined by this specification, is considered to be authoritative and takes precedence over the XML Schema defined in the appendix of this document.

An SCA runtime MUST reject a composite file that does not conform to the sca-policy-1.1.xsd schema. [POL110001]

An implementation that claims to conform to this specification MUST meet the following conditions:

1. The implementation MUST conform to the SCA Assembly Model Specification [Assembly].
2. SCA implementations MUST recognize the intents listed in Appendix B.1 of this specification. An implementationType / bindingType / collection of policySets that claims to implement a specific intent MUST process that intent in accord with any relevant Conformance Items in Appendix C related to the intent and the SCA Runtime options selected.
3. With the exception of 2, the implementation MUST comply with all statements in Appendix C: Conformance Items related to an SCA Runtime, notably all MUST statements have to be implemented.

1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983

---

## A Defining the Deployed Composites Infoset

The @attachTo attribute of a policySet or the @attachTo attribute of a <externalAttachment/> element is an XPath1.0 expression identifying SCA elements to which intents and/or policySets are attached. The XPath applies to the **Deployed Composites Infoset** for the SCA domain.

The Deployed Composites Infoset is constructed from all the deployed SCA composite files [SCA-Assembly] in the Domain, with the special characteristics:

1. The Domain is treated as a special composite, with a blank name - ""
2. The @attachTo/@ppliesTo XPath expression is evaluated against the Deployed Composite Infoset following the deployment of a deployment composite. 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 intent or policySet is intended to be specific to a particular component, the structuralURI [SCA-Assembly] of the component is used along with the URIRef() XPath function to attach a intent/policySet to a specific use of a nested component. The XPath expression can make use of the unique structuralURI to indicate specific use instances, where different intents/policySets need to be used for those different instances.

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 intent/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 structuralURI [SCA-Assembly].  
This permits the attachment of a intent/policySet to a specific use of a component in one context, that can be different from the policySet(s) that are applied to other uses of the same component.

Details of the available XPath functions is given in the section "[XPath Functions for the @attachTo Attribute](#)".

EXAMPLE:

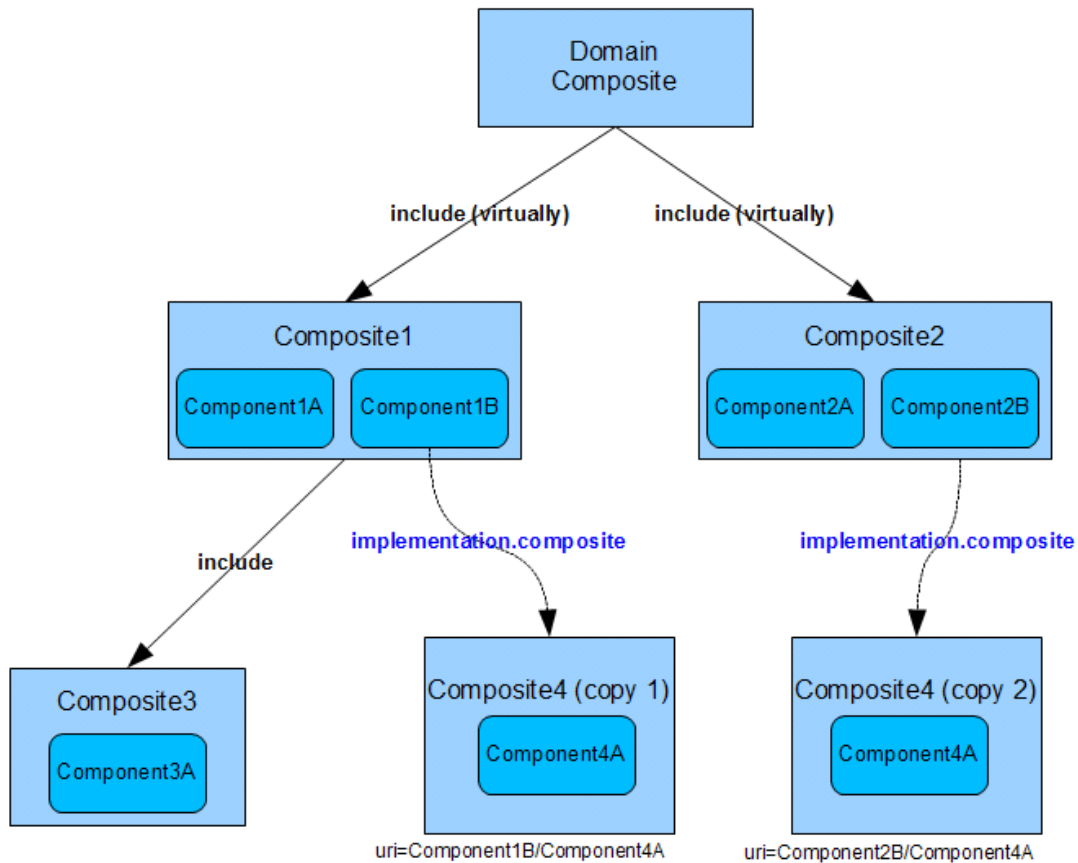


Figure A-1 Example Domain Composite Infoset

The SCA Domain in Figure A-1 has been constructed from the composites and components shown in the figure. Composite1 and Composite2 were deployed into the Domain as described in [SCA-Assembly]. Composite3 is included in Composite1 using the SCA include mechanism described in [SCA-Assembly]. Composite4 is used as an implementation of Components 1B and 2B. Following the deployment of all the composites, the Domain contains:

- 3 Composites that can be addressed as part of the Deployed Composites InfoSet; Composite1, Composite2 and Composite4.
- all the components shown in the diagram. Components 1A, 2A, 3A, 4A (twice) are leaf components.

The following snippets show example usage of the @attachTo attribute and provide the outcome based on the Domain in Figure A-1.

```
1. //component[@name="Component4A"]
```

Snippet A-1: Example attachTo all Instances of a Name

attach to both instances of Component4A

```
2. //component[URIRef("Component2B/Component4A")]
```



2006 *Snippet A-2: Example attachTo a Specific Instance via a Path*

2007

2008 attach to the unique instance of Component4A when used by Component2B (Component2B is a  
2009 component at the Domain level)

2010

```
2011 3. //component[@name="Component3A"]/service[IntentRefs( "intent1" ) ]
```

2012 *Snippet A-3: Example attachTo Instances with an intent*

2013

2014 attach to the services of Component3A which have the intent "intent1" applied

2015

```
2016 4. //component/binding.ws
```

2017 *Snippet A-4: Example attachTo Instances with a binding*

2018

2019 attach to the web services binding of all components with a service or reference with a Web services  
2020 binding

2021

```
2022 5. /composite[@name=" "]/component[@name="Component1A"]
```

2023 *Snippet A-5: Example attachTo a Specific Instance via Path and Name*

2024

2025 attach to Component1A at the Domain level

2026

2027

## 2028 **A.1 XPath Functions for the @attachTo Attribute**

2029 This section defines utility functions that can be used in XPath expressions where otherwise it would be  
2030 difficult to write the XPath expression to identify the elements concerned.

2031 This particularly applies in SCA to Interfaces and the child parts of interfaces (operations and messages).  
2032 XPath Functions are defined below for the following:

- 2033 • Picking out a specific interface
- 2034 • Picking out a specific operation in an interface
- 2035 • Picking out a specific message in an operation in an interface
- 2036 • Picking out artifacts with specific intents

### 2037 **A.1.1 Interface Related Functions**

#### 2038 **InterfaceRef( InterfaceName )**

2039 picks out an interface identified by InterfaceName

#### 2040 **OperationRef( InterfaceName/OperationName )**

2041 picks out the operation OperationName in the interface InterfaceName

#### 2042 **MessageRef( InterfaceName/OperationName/MessageName )**

2043 picks out the message MessageName in the operation OperationName in the interface  
2044 InterfaceName.

- 2045 • "\*" can be used for wildcarding of any of the names.

2046 The interface is treated as if it is a WSDL interface (for other interface types, they are treated as if  
2047 mapped to WSDL using their regular mapping rules).

2048 Examples of the Interface functions:

2049

```
2050 InterfaceRef( "MyInterface" )
```

2051 *Snippet A-6: Example use of InterfaceRef*

2052

2053 picks out an interface with the name "MyInterface"

2054

```
2055 OperationRef( "MyInterface/MyOperation" )
```

2056 *Snippet A-7: Example use of OperationRef with a Path*

2057

2058 picks out the operation named "MyOperation" within the interface named "MyInterface"

2059

```
2060 OperationRef( "*/MyOperation" )
```

2061 *Snippet A-8: Example use of OperationRef without a Path*

2062

2063 picks out the operation named "MyOperation" from any interface

2064

```
2065 MessageRef( "MyInterface/MyOperation/MyMessage" )
```

2066 *Snippet A-9: Example use of MessageRef with a Path*

2067

2068 picks out the message named "MyMessage" from the operation named "MyOperation" within the interface  
2069 named "MyInterface"

2070

```
2071 MessageRef( "*/*/MyMessage" )
```

2072 *Snippet A-10: Example use of MessageRef with a Path with Wildcards*

2073

2074 picks out the message named "MyMessage" from any operation in any interface

## 2075 **A.1.2 Intent Based Functions**

2076 For the following intent-based functions, it is the total set of intents which apply to the artifact which are  
2077 examined by the function, including directly or externally attached intents plus intents acquired from the  
2078 structural hierarchy and from the implementation hierarchy.

2079

### 2080 **IntentRefs( IntentList )**

2081 picks out an element where the intents applied match the intents specified in the IntentList:

2082

```
2083 IntentRefs( "intent1" )
```

2084 *Snippet A-11: Example use of IntentRef*

2085

2086 picks out an artifact to which intent named "intent1" is attached

2087

```
2088 IntentRefs( "intent1 intent2" )
```

2089 *Snippet A-12: Example use of IntentRef with Multiple intents*

2090

2091 picks out an artifact to which intents named "intent1" AND "intent2" are attached

2092

```
2093 IntentRefs( "intent1 !intent2" )
```

2094 *Snippet A-13: Example use of IntentRef with Not Operator*

2095

2096 picks out an artifact to which intent named "intent1" is attached but NOT the intent named "intent2"

### 2097 **A.1.3 URI Based Function**

2098 The URIRef function is used to pick out a particular use of a nested component – i.e. where some  
2099 Domain level component is implemented using a composite implementation, which in turn has one or  
2100 more components implemented with the composite (and so on to an arbitrary level of nesting):

#### 2101 **URIRef( URI )**

2102 picks out the particular use of a component identified by the structuralURI string URI.

2103 For a full description of structuralURIs, see the SCA Assembly specification [\[SCA-Assembly\]](#).

2104 Example:

2105

```
2106 URIRef( "top_comp_name/middle_comp_name/lowest_comp_name" )
```

2107 *Snippet A-15: Example use of URIRef*

2108

2109 picks out the particular use of a component – where component lowest\_comp\_name is used within the  
2110 implementation of middle\_comp\_name within the implementation of the top-level (Domain level)  
2111 component top\_comp\_name.

2112

2113

## B Schemas

2114

### B.1 sca-policy.xsd

```
2115 <?xml version="1.0" encoding="UTF-8"?>
2116 <!-- Copyright(C) OASIS(R) 2005,2010. All Rights Reserved.
2117 OASIS trademark, IPR and other policies apply. -->
2118 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2119 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2120 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2121 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
2122 elementFormDefault="qualified">
2123
2124 <include schemaLocation="sca-core-1.1-cd05.xsd"/>
2125 <import namespace="http://www.w3.org/ns/ws-policy"
2126 schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd"/>
2127
2128 <element name="intent" type="sca:Intent"/>
2129 <complexType name="Intent">
2130 <sequence>
2131 <element name="description" type="string" minOccurs="0"
2132 maxOccurs="1" />
2133 <element name="qualifier" type="sca:IntentQualifier"
2134 minOccurs="0" maxOccurs="unbounded" />
2135 <any namespace="##other" processContents="lax"
2136 minOccurs="0" maxOccurs="unbounded" />
2137 </sequence>
2138 <attribute name="name" type="NCName" use="required"/>
2139 <attribute name="constrains" type="sca:listOfQNames"
2140 use="optional" />
2141 <attribute name="requires" type="sca:listOfQNames"
2142 use="optional" />
2143 <attribute name="excludes" type="sca:listOfQNames"
2144 use="optional" />
2145 <attribute name="mutuallyExclusive" type="boolean"
2146 use="optional" default="false"/>
2147 <attribute name="intentType"
2148 type="sca:InteractionOrImplementation"
2149 use="optional" default="interaction"/>
2150 <anyAttribute namespace="##other" processContents="lax" />
2151 </complexType>
2152
2153 <complexType name="IntentQualifier">
2154 <sequence>
2155 <element name="description" type="string" minOccurs="0"
2156 maxOccurs="1" />
2157 <any namespace="##other" processContents="lax" minOccurs="0"
2158 maxOccurs="unbounded" />
2159 </sequence>
2160 <attribute name="name" type="NCName" use="required"/>
2161 <attribute name="default" type="boolean" use="optional"
2162 default="false" />
2163 </complexType>
2164
2165 <element name="requires">
2166 <complexType>
2167 <sequence minOccurs="0" maxOccurs="unbounded">
2168 <any namespace="##other" processContents="lax" />
2169 </sequence>
```

```

2170         <attribute name="intents" type="sca:listOfQNames"
2171             use="required"/>
2172         <anyAttribute namespace="##other" processContents="lax"/>
2173     </complexType>
2174 </element>
2175
2176 <element name="externalAttachment">
2177     <complexType>
2178         <sequence minOccurs="0" maxOccurs="unbounded">
2179             <any namespace="##other"
2180                 processContents="lax"/>
2181         </sequence>
2182         <attribute name="intents" type="sca:listOfQNames"
2183             use="optional"/>
2184         <attribute name="policySets" type="sca:listOfQNames"
2185             use="optional"/>
2186         <attribute name="name" type="string"
2187             use="required"/>
2188         <anyAttribute namespace="##other"
2189             processContents="lax"/>
2190     </complexType>
2191 </element>
2192
2193 <element name="policySet" type="sca:PolicySet"/>
2194 <complexType name="PolicySet">
2195     <choice minOccurs="0" maxOccurs="unbounded">
2196         <element name="policySetReference"
2197             type="sca:PolicySetReference"/>
2198         <element name="intentMap" type="sca:IntentMap"/>
2199         <any namespace="##other" processContents="lax"/>
2200     </choice>
2201     <attribute name="name" type="NCName" use="required"/>
2202     <attribute name="provides" type="sca:listOfQNames"/>
2203     <attribute name="appliesTo" type="string" use="optional"/>
2204     <attribute name="attachTo" type="string" use="optional"/>
2205     <anyAttribute namespace="##other" processContents="lax"/>
2206 </complexType>
2207
2208 <element name="policySetAttachment">
2209     <complexType>
2210         <sequence minOccurs="0" maxOccurs="unbounded">
2211             <any namespace="##other" processContents="lax"/>
2212         </sequence>
2213         <attribute name="name" type="QName" use="required"/>
2214         <anyAttribute namespace="##other" processContents="lax"/>
2215     </complexType>
2216 </element>
2217
2218 <complexType name="PolicySetReference">
2219     <attribute name="name" type="QName" use="required"/>
2220     <anyAttribute namespace="##other" processContents="lax"/>
2221 </complexType>
2222
2223 <complexType name="IntentMap">
2224     <choice minOccurs="1" maxOccurs="unbounded">
2225         <element name="qualifier" type="sca:Qualifier"/>
2226         <any namespace="##other" processContents="lax"/>
2227     </choice>
2228     <attribute name="provides" type="QName" use="required"/>
2229     <anyAttribute namespace="##other" processContents="lax"/>
2230 </complexType>
2231
2232 <complexType name="Qualifier">

```

```
2233     <sequence minOccurs="0" maxOccurs="unbounded">
2234         <any namespace="##other" processContents="lax"/>
2235     </sequence>
2236     <attribute name="name" type="string" use="required"/>
2237     <anyAttribute namespace="##other" processContents="lax"/>
2238 </complexType>
2239
2240 <simpleType name="listOfNCNames">
2241     <list itemType="NCName"/>
2242 </simpleType>
2243
2244 <simpleType name="InteractionOrImplementation">
2245     <restriction base="string">
2246         <enumeration value="interaction"/>
2247         <enumeration value="implementation"/>
2248     </restriction>
2249 </simpleType>
2250
2251 </schema>
```

2252 *Snippet B-1SCA Policy Schema*

2253

## C XML Files

2254 This appendix contains normative XML files that are defined by this specification.

### 2255 C.1 Intent Definitions

2256 Intent definitions are contained within a Definitions file called sca-policy-1.1-intents-definitions.xml, which  
2257 contains a <definitions/> element as follows:

```
2258 <?xml version="1.0" encoding="UTF-8"?>
2259 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2260 OASIS trademark, IPR and other policies apply. -->
2261 <sca:definitions xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2262 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2263 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912">
2264
2265 <!-- Security related intents -->
2266 <sca:intent name="serverAuthentication" constrains="sca:binding"
2267 intentType="interaction">
2268 <sca:description>
2269 Communication through the binding requires that the
2270 server is authenticated by the client
2271 </sca:description>
2272 <sca:qualifier name="transport" default="true"/>
2273 <sca:qualifier name="message"/>
2274 </sca:intent>
2275
2276 <sca:intent name="clientAuthentication" constrains="sca:binding"
2277 intentType="interaction">
2278 <sca:description>
2279 Communication through the binding requires that the
2280 client is authenticated by the server
2281 </sca:description>
2282 <sca:qualifier name="transport" default="true"/>
2283 <sca:qualifier name="message"/>
2284 </sca:intent>
2285
2286 <sca:intent name="authentication"
2287 requires="sca:clientAuthentication">
2288 <sca:description>
2289 A convenience intent to help migration
2290 </sca:description>
2291 </sca:intent>
2292
2293 <sca:intent name="mutualAuthentication"
2294 requires="sca:clientAuthentication sca:serverAuthentication">
2295 <sca:description>
2296 Communication through the binding requires that the
2297 client and server to authenticate each other
2298 </sca:description>
2299 </sca:intent>
2300
2301 <sca:intent name="confidentiality" constrains="sca:binding"
2302 intentType="interaction">
2303 <sca:description>
2304 Communication through the binding prevents unauthorized
2305 users from reading the messages
2306 </sca:description>
2307 <sca:qualifier name="transport" default="true"/>
```

```

2308         <sca:qualifier name="message" />
2309     </sca:intent>
2310
2311     <sca:intent name="integrity" constrains="sca:binding"
2312 intentType="interaction">
2313         <sca:description>
2314             Communication through the binding prevents tampering
2315             with the messages sent between the client and the service.
2316         </sca:description>
2317         <sca:qualifier name="transport" default="true" />
2318         <sca:qualifier name="message" />
2319     </sca:intent>
2320
2321     <sca:intent name="authorization" constrains="sca:implementation"
2322 intentType="implementation">
2323         <sca:description>
2324             Ensures clients are authorized to use services.
2325         </sca:description>
2326     </sca:intent>
2327
2328
2329 <!-- Reliable messaging related intents -->
2330     <sca:intent name="atLeastOnce" constrains="sca:binding"
2331 intentType="interaction">
2332         <sca:description>
2333             This intent is used to indicate that a message sent
2334             by a client is always delivered to the component.
2335         </sca:description>
2336     </sca:intent>
2337
2338     <sca:intent name="atMostOnce" constrains="sca:binding"
2339 intentType="interaction">
2340         <sca:description>
2341             This intent is used to indicate that a message that was
2342             successfully sent by a client is not delivered more than
2343             once to the component.
2344         </sca:description>
2345     </sca:intent>
2346
2347     <sca:intent name="exactlyOnce" requires="sca:atLeastOnce
2348 sca:atMostOnce"
2349 constrains="sca:binding" intentType="interaction">
2350         <sca:description>
2351             This profile intent is used to indicate that a message sent
2352             by a client is always delivered to the component. It also
2353             indicates that duplicate messages are not delivered to the
2354             component.
2355         </sca:description>
2356     </sca:intent>
2357
2358     <sca:intent name="ordered" constrains="sca:binding"
2359 intentType="interaction">
2360         <sca:description>
2361             This intent is used to indicate that all the messages are
2362             delivered to the component in the order they were sent by
2363             the client.
2364         </sca:description>
2365     </sca:intent>
2366
2367 <!-- Transaction related intents -->
2368     <sca:intent name="managedTransaction"
2369 excludes="sca:noManagedTransaction"
2370 mutuallyExclusive="true" constrains="sca:implementation"

```



```

2371 intentType="implementation">
2372   <sca:description>
2373     A managed transaction environment is necessary in order to
2374     run the component. The specific type of managed transaction
2375     needed is not constrained.
2376   </sca:description>
2377   <sca:qualifier name="global" default="true">
2378     <sca:description>
2379       For a component marked with managedTransaction.global
2380       a global transaction needs to be present before dispatching
2381       any method on the component - using any transaction
2382       propagated from the client or else beginning and completing
2383       a new transaction.
2384     </sca:description>
2385   </sca:qualifier>
2386   <sca:qualifier name="local">
2387     <sca:description>
2388       A component marked with managedTransaction.local needs to
2389       run within a local transaction containment (LTC) that
2390       is started and ended by the SCA runtime.
2391     </sca:description>
2392   </sca:qualifier>
2393 </sca:intent>
2394
2395   <sca:intent name="noManagedTransaction"
2396   excludes="sca:managedTransaction"
2397   constrains="sca:implementation" intentType="implementation">
2398     <sca:description>
2399       A component marked with noManagedTransaction needs to run without
2400       a managed transaction, under neither a global transaction nor
2401       an LTC. A transaction propagated to the hosting SCA runtime
2402       is not joined by the hosting runtime on behalf of a
2403       component marked with noManagedtransaction.
2404     </sca:description>
2405   </sca:intent>
2406
2407   <sca:intent name="transactedOneWay" excludes="sca:immediateOneWay"
2408   constrains="sca:binding" intentType="implementation">
2409     <sca:description>
2410       For a reference marked as transactedOneWay any OneWay invocation
2411       messages are transacted as part of a client global
2412       transaction.
2413       For a service marked as transactedOneWay any OneWay invocation
2414       message are received from the transport binding in a
2415       transacted fashion, under the service's global transaction.
2416     </sca:description>
2417   </sca:intent>
2418
2419   <sca:intent name="immediateOneWay" excludes="sca:transactedOneWay"
2420   constrains="sca:binding" intentType="implementation">
2421     <sca:description>
2422       For a reference indicates that any OneWay invocation messages
2423       are sent immediately regardless of any client transaction.
2424       For a service indicates that any OneWay invocation is
2425       received immediately regardless of any target service
2426       transaction.
2427     </sca:description>
2428   </sca:intent>
2429
2430   <sca:intent name="propagatesTransaction"
2431   excludes="sca:suspendsTransaction"
2432   constrains="sca:binding" intentType="interaction">
2433     <sca:description>

```

```

2434     A service marked with propagatesTransaction is dispatched
2435     under any propagated (client) transaction and the service binding
2436     needs to be capable of receiving a transaction context.
2437     A reference marked with propagatesTransaction propagates any
2438     transaction context under which the client runs when the
2439     reference is used for a request-response interaction and the
2440     binding of a reference marked with propagatesTransaction needs to
2441     be capable of propagating a transaction context.
2442     </sca:description>
2443 </sca:intent>
2444
2445     <sca:intent name="suspendsTransaction"
2446         excludes="sca:propagatesTransaction"
2447     constrains="sca:binding" intentType="interaction">
2448         <sca:description>
2449             A service marked with suspendsTransaction is not dispatched
2450             under any propagated (client) transaction.
2451             A reference marked with suspendsTransaction does not propagate
2452             any transaction context under which the client runs when the
2453             reference is used.
2454         </sca:description>
2455     </sca:intent>
2456
2457     <sca:intent name="managedSharedTransaction"
2458         requires="sca:managedTransaction.global
2459     sca:propagatesTransaction">
2460         <sca:description>
2461             Used to indicate that the component requires both the
2462             managedTransaction.global and the propagatesTransactions
2463             intents
2464         </sca:description>
2465     </sca:intent>
2466
2467 <!-- Miscellaneous intents -->
2468 <sca:intent name="asyncInvocation" excludes="sca:propagatesTransaction"
2469     constrains="sca:binding" intentType="interaction">
2470     <sca:description>
2471         Indicates that request/response operations for the
2472         interface of this wire are "long running" and must be
2473         treated as two separate message transmissions
2474     </sca:description>
2475 </sca:intent>
2476
2477 <sca:intent name="EJB" constrains="sca:binding"
2478     intentType="interaction">
2479     <sca:description>
2480         Specifies that the EJB API is needed to communicate with
2481         the service or reference.
2482     </sca:description>
2483 </sca:intent>
2484
2485 <sca:intent name="SOAP" constrains="sca:binding"
2486     intentType="interaction" mutuallyExclusive="true">
2487     <sca:description>
2488         Specifies that the SOAP messaging model is used for delivering
2489         messages.
2490     </sca:description>
2491     <sca:qualifier name="v1_1" default="true"/>
2492     <sca:qualifier name="v1_2"/>
2493 </sca:intent>
2494
2495 <sca:intent name="JMS" constrains="sca:binding"
2496     intentType="interaction">

```

```
2497         <sca:description>
2498         Requires that the messages are delivered and received via the
2499         JMS API.
2500         </sca:description>
2501     </sca:intent>
2502
2503     <sca:intent name="noListener" constrains="sca:binding"
2504     intentType="interaction">
2505         <sca:description>
2506         This intent can only be used on a reference. Indicates that the
2507         client is not able to handle new inbound connections. The binding
2508         and callback binding are configured so that any
2509         response or callback comes either through a back channel of the
2510         connection from the client to the server or by having the client
2511         poll the server for messages.
2512         </sca:description>
2513     </sca:intent>
2514
2515 </sca:definitions>
```

2516 *Snippet C-1: SCA intent Definitions*

---

## 2517 D Conformance

### 2518 D.1 Conformance Targets

2519 The conformance items listed in the section below apply to the following conformance targets:

- 2520 • Document artifacts (or constructs within them) that can be checked statically.
- 2521 • SCA runtimes, which we may require to exhibit certain behaviors.

### 2522 D.2 Conformance Items

2523 This section contains a list of conformance items for the SCA Policy Framework specification.

2524

Conformance ID	Description
[POL30001]	If the configured instance of a binding is in conflict with the intents and policy sets selected for that instance, the SCA runtime MUST raise an error.
[POL30002]	The QName for an intent MUST be unique amongst the set of intents in the SCA Domain.
[POL30004]	If an intent has more than one qualifier, one and only one MUST be declared as the default qualifier.
[POL30005]	The name of each qualifier MUST be unique within the intent definition.
[POL30006]	the name of a profile intent MUST NOT have a "." in it.
[POL30007]	If a profile intent is attached to an artifact, all the intents listed in its @requires attribute MUST be satisfied as described in section 4.15.
[POL30008]	When a policySet element contains a set of intentMap children, the value of the @provides attribute of each intentMap MUST correspond to an unqualified intent that is listed within the @provides attribute value of the parent policySet element.
[POL30010]	For each qualifiable intent listed as a member of the @provides attribute list of a policySet element, there MUST be no more than one corresponding intentMap element that declares the unqualified form of that intent in its @provides attribute. In other words, each intentMap within a given policySet uniquely provides for a specific intent.
[POL30011]	Following the inclusion of all policySet references, when a policySet element directly contains wsp:policyAttachment children or policies using extension elements, the set of policies specified as children MUST satisfy all the intents expressed using the @provides attribute value of the policySet element.
[POL30013]	The set of intents in the @provides attribute of a referenced policySet MUST be a subset of the set of intents in the @provides attribute of the referencing policySet.

- [POL30015] Each QName in the @requires attribute MUST be the QName of an intent in the SCA Domain.
- [POL30016] Each QName in the @excludes attribute MUST be the QName of an intent in the SCA Domain.
- [POL30017] The QName for a policySet MUST be unique amongst the set of policySets in the SCA Domain.
- [POL30018] The contents of @appliesTo MUST match the XPath 1.0 [XPATH] production *Expr*.
- [POL30019] The contents of @attachTo MUST match the XPath 1.0 production *Expr*.
- [POL30020] If a policySet specifies a qualifiable intent in the @provides attribute, and it provides an intentMap for the qualifiable intent then that intentMap MUST specify all possible qualifiers for that intent.
- [POL30021] 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.
- [POL30024] An SCA Runtime MUST include in the Domain the set of intent definitions contained in the Policy\_Intent\_Definitions.xml described in the appendix "Intent Definitions" of the SCA Policy specification.
- [POL30025] If only one qualifier for an intent is given it MUST be used as the default qualifier for the intent.
- [POL40001] SCA implementations supporting both Direct Attachment and External Attachment mechanisms MUST ignore policy sets applicable to any given SCA element via the Direct Attachment mechanism when there exist policy sets applicable to the same SCA element via the External Attachment mechanism
- The SCA runtime MUST raise an error if the @attachTo XPath expression resolves to an SCA <property> element, or any of its children. [POL40002]
- [POL40004] A qualifiable intent expressed lower in the hierarchy can be qualified further up the hierarchy, in which case the qualified version of the intent MUST apply to the higher level element.
- [POL40005] The intents declared on elements higher in the structural hierarchy of a given element MUST be applied to the element EXCEPT
- if any of the inherited intents is mutually exclusive with an intent applied on the element, then the inherited intent MUST be 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, the qualified version of the intent MUST be used.

- [POL40006] If a component has any policySets attached to it (by any means), then any policySets attached to the componentType MUST be ignored.
- [POL40007] Matching service/reference policies across the SCA Domain boundary MUST use WS-Policy compatibility (strict WS-Policy intersection) if the policies are expressed in WS-Policy syntax.
- [POL40009] Any two intents applied to a given element MUST NOT be mutually exclusive
- [POL40010] SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms for policySet attachment.
- [POL40011] SCA implementations supporting only the External Attachment mechanism MUST ignore the policy sets that are applicable via the Direct Attachment mechanism.
- [POL40012] SCA implementations supporting only the Direct Attachment mechanism MUST ignore the policy sets that are applicable via the External Attachment mechanism.
- [POL40014] The intents declared on elements lower in the implementation hierarchy of a given element MUST be applied to the element.
- [POL40015] When combining implementation hierarchy and structural hierarchy policy data, Rule 1 MUST be applied BEFORE Rule 2.
- [POL40016] When calculating the set of intents and set of policySets which apply to either a service element or to a reference element of a component, intents and policySets from the interface definition and from the interface declaration(s) MUST be applied to the service or reference element and to the binding element(s) belonging to that element.
- [POL40017] If the required intent set contains a mutually exclusive pair of intents the SCA runtime MUST reject the document containing the element and raise an error.
- [POL40018] All intents in the required intent set for an element MUST be provided by the directly provided intents set and the set of policySets that apply to the element, or else an error is raised.
- [POL40019] The locations where interfaces are defined and where interfaces are declared in the componentType and in a component MUST be treated as part of the implementation hierarchy as defined in Section 4.5 Attaching intents to SCA elements.
- [POL40020] The QName of the bindingType MUST be unique amongst the set of bindingTypes in the SCA Domain.
- [POL40021] A binding implementation MUST implement all the intents listed in the @alwaysProvides and @mayProvides attributes.
- [POL40022] The SCA runtime MUST determine the compatibility of the policySets at each end of a wire using the compatibility rules of the policy language used for those policySets.
- [POL40023] The policySets at each end of a wire MUST be incompatible if

- they use different policy languages.
- [POL40024] Where the policy language in use for a wire is WS-Policy, strict WS-Policy intersection MUST be used to determine policy compatibility.
- [POL40025] In order for a reference to connect to a particular service, the policies of the reference MUST intersect with the policies of the service.
- [POL40027] Any intents attached to an interface definition artifact, such as a WSDL portType, MUST be added to the intents attached to the service or reference to which the interface definition applies. If no intents are attached to the service or reference then the intents attached to the interface definition artifact become the only intents attached to the service or reference.
- [POL40029] If the process of redeployment of intents, externalAttachments and/or policySets fails because one or more intents are left unsatisfied, an error MUST be raised.
- [POL40030] If the process of redeployment of intents, externalAttachments and/or policySets fails, the changed intents, externalAttachments and/or policySets MUST NOT be deployed and no change is made to deployed and running artifacts.
- [POL40031] When redeployment of intents, externalAttachments and policySets succeeds, the components whose policies are affected by the redeployment MAY have their policies updated by the SCA runtime dynamically without the need to stop and restart those components.
- [POL40032] Where components are updated by redeployment of intents, externalAttachments and policySets (their configuration is changed in some way, which includes changing the policies associated with a component), the new configuration MUST apply to all new instances of those components once the redeployment is complete.
- [POL40033] Where a component configuration is changed by the redeployment of intents, externalAttachments and policySets, the SCA runtime either MAY choose to maintain existing instances with the old configuration of the component, or the SCA runtime MAY choose to stop and discard existing instances of the component.
- [POL40034] During the deployment of SCA composites, first all <externalAttachment/> elements within the Domain MUST be evaluated to determine which intents are attached to elements in the newly deployed composite and then all policySets within the Domain with an @attachTo attribute or <externalAttachment> elements that attach policySets MUST be evaluated to determine which policySets are attached to elements in the newly deployed composite.
- [POL40035] The contents of the @attachTo attribute of an externalAttachment element MUST match the XPath 1.0 production Expr.
- [POL50001] The implementationType name attribute MUST be the QName of

- an XSD global element definition used for implementation elements of that type.
- [POL70001] When *authorization* is present, an SCA Runtime MUST ensure that the client is authorized to use the service.
- [POL70009] When confidentiality is present, an SCA Runtime MUST ensure that only authorized entities can view the contents of a message.
- [POL70010] When *integrity* is present, an SCA Runtime MUST ensure that the contents of a message are not altered.
- [POL70011] When a serverAuthentication, clientAuthentication, confidentiality or integrity intent is qualified by transport, an SCA Runtime MUST delegate serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the transport layer of the communication protocol.
- [POL70012] When a serverAuthentication, clientAuthentication, confidentiality or integrity intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication protocol.
- [POL70013] When *serverAuthentication* is present, an SCA runtime MUST ensure that the server is authenticated by the client.
- [POL70014] When *clientAuthentication* is present, an SCA runtime MUST ensure that the client is authenticated by the server.
- [POL80001] When *atLeastOnce* is present, an SCA Runtime MUST deliver a message to the destination service implementation, and MAY deliver duplicates of a message to the service implementation.
- [POL80002] When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service implementation, and MUST NOT deliver duplicates of a message to the service implementation.
- [POL80003] When *ordered* is present, an SCA Runtime MUST deliver messages sent by a single source to a single destination service implementation in the order that the messages were sent by that source.
- [POL80004] When *exactlyOnce* is present, an SCA Runtime MUST deliver a message to the destination service implementation and MUST NOT deliver duplicates of a message to the service implementation.
- [POL90003] For a component marked with managedTransaction.global, the SCA runtime MUST ensure that a global transaction is present before dispatching any method on the component.
- [POL90004] A component marked with managedTransaction.local MUST run within a local transaction containment (LTC) that is started and ended by the SCA runtime.
- [POL90006] Local transactions MUST NOT be propagated outbound across remotable interfaces.



- [POL90007] A transaction that is propagated to the hosting SCA runtime MUST NOT be joined by the hosting runtime on behalf of a component marked with noManagedtransaction.
- [POL90008] When a reference is marked as transactedOneWay, any OneWay invocation messages MUST be transacted as part of a client global transaction.
- [POL90009] If the client component is not configured to run under a global transaction or if the binding does not support transactional message sending, then a reference MUST NOT be marked as transactedOneWay.
- [POL90010] If a service is marked as transactedOneWay, any OneWay invocation message MUST be received from the transport binding in a transacted fashion, under the target service's global transaction.
- [POL90011] If the component is not configured to run under a global transaction or if the binding does not support transactional message receipt, then a service MUST NOT be marked as transactedOneWay.
- [POL90012] When applied to a reference indicates that any OneWay invocation messages MUST be sent immediately regardless of any client transaction.
- [POL90013] When applied to a service indicates that any OneWay invocation MUST be received immediately regardless of any target service transaction.
- [POL90015] A service marked with propagatesTransaction MUST be dispatched under any propagated (client) transaction.
- [POL90016] Use of the **propagatesTransaction** intent on a service implies that the service binding MUST be capable of receiving a transaction context.
- [POL90017] A service marked with suspendsTransaction MUST NOT be dispatched under any propagated (client) transaction.
- [POL90019] A service MUST NOT be marked with "propagatesTransaction" if the component is marked with "managedTransaction.local" or with "noManagedTransaction"
- [POL90020] When a reference is marked with propagatesTransaction, any transaction context under which the client runs MUST be propagated when the reference is used for a request-response interaction
- [POL90022] When a reference is marked with suspendsTransaction, any transaction context under which the client runs MUST NOT be propagated when the reference is used.
- [POL90023] A reference MUST NOT be marked with propagatesTransaction if component is marked with "ManagedTransaction.local" or with "noManagedTransaction"
- [POL90024] Transaction context MUST NOT be propagated on OneWay messages.
- [POL90025] The SCA runtime MUST ignore the propagatesTransaction intent for OneWay methods.

[POL90027]	If a <b>transactedOneWay</b> intent is combined with the <b>managedTransaction.local</b> or <b>noManagedTransaction</b> implementation intents for either a reference or a service then an error <b>MUST</b> be raised during deployment.
[POL90028]	The <b>transactedOneWay</b> intent <b>MUST NOT</b> be attached to a request/response operation.
[POL90029]	The <b>immediateOneWay</b> intent <b>MUST NOT</b> be attached to a request/response operation.
[POL90030]	The <b>asyncInvocation</b> intent and the <b>propagatesTransaction</b> intent <b>MUST NOT</b> be applied to the same service or reference operation.
[POL90031]	When the <b>asyncInvocation</b> intent is applied to an SCA service, the SCA runtime <b>MUST</b> behave as if the <b>suspendsTransaction</b> intent is also applied to the service.
[POL100001]	When <b>SOAP</b> is present, an SCA Runtime <b>MUST</b> use the SOAP messaging model to deliver messages.
[POL100002]	When a <b>SOAP</b> intent is qualified with <b>1_1</b> or <b>1_2</b> , then SOAP version 1.1 or SOAP version 1.2 respectively <b>MUST</b> be used to deliver messages.
[POL100003]	When <b>JMS</b> is present, an SCA Runtime <b>MUST</b> ensure that the binding used to send and receive messages supports the JMS API.
[POL100004]	The <b>noListener</b> intent <b>MUST</b> only be declared on a <b>@requires</b> attribute of a reference.
[POL100005]	When <b>noListener</b> is present, an SCA Runtime <b>MUST</b> not establish any connection from a service to a client.
[POL100006]	When <b>EJB</b> is present, an SCA Runtime <b>MUST</b> ensure that the binding used to send and receive messages supports the EJB API.
[POL100007]	The SCA Runtime <b>MUST</b> ignore the <b>asyncInvocation</b> intent for one way operations.
[POL110001]	An SCA runtime <b>MUST</b> reject a composite file that does not conform to the <b>sca-policy-1.1.xsd</b> schema.

2525 Table D-1: SCA Policy Normative Statements

2526

## E Acknowledgements

2527 The following individuals have participated in the creation of this specification and are  
2528 gratefully acknowledged:

Participant Name	Affiliation
Jeff Anderson	Deloitte Consulting LLP
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Vladislav Bezrukov	SAP AG*
Henning Blohm	SAP AG*
David Booz	IBM
Fred Carter	AmberPoint
Tai-Hsing Cha	TIBCO Software Inc.
Martin Chapman	Oracle Corporation
Mike Edwards	IBM
Raymond Feng	IBM
Billy Feng	Primeton Technologies, Inc.
Robert Freund	Hitachi, Ltd.
Murty Gurajada	TIBCO Software Inc.
Simon Holdsworth	IBM
Michael Kanaley	TIBCO Software Inc.
Anish Karmarkar	Oracle Corporation
Nickolas Kavantzias	Oracle Corporation
Rainer Kerth	SAP AG*
Pundalik Kudapkar	TIBCO Software Inc.
Meeraj Kunnumpurath	Individual
Rich Levinson	Oracle Corporation
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Jeff Mischkinsky	Oracle Corporation
Dale Moberg	Axway Software*
Simon Nash	Individual
Bob Natale	Mitre Corporation*
Eisaku Nishiyama	Hitachi, Ltd.
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Martin Raeppele	SAP AG*
Fabian Ritzmann	Sun Microsystems
Ian Robinson	IBM
Scott Vorthmann	TIBCO Software Inc.
Eric Wells	Hitachi, Ltd.
Prasad Yendluri	Software AG, Inc.*
Alexander Zubev	SAP AG*

2529

2530  
2531  
2532

## F 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
8	Sept 8 2008	Mike Edwards	Applied resolution for Issue 15
9	Oct 17 2008	David Booz	Various formatting changes Applied 22 – Deleted text in Ch 9 Applied 42 – In section 3.3 Applied 46 – Many sections Applied 52,55 – Many sections Applied 53 – In section 3.3 Applied 56 – In section 3.1 Applied 58 – Many sections
10	Nov 26	David Booz	Applied camelCase words from Liason Applied 54 – many sections Applied 59 – section 4.2, 4.4.2 Applied 60 – section 8.1 Applied 61 – section 4.10, 4.12 Applied 63 – section 9
11	Dec 10	Mike Edwards	Applied 44 - section 3.1, 3.2 (new), 5.0, A.1 Renamed file to sca-policy-1.1-spec-CD01-Rev11
12	Dec 25	Ashok Malhotra	Added RFC 2119 keywords Renamed file to sca-policy-1.1-spec-CD01-Rev12
13	Feb 06 2009	Mike Edwards, Eric	All changes accepted

		Wells, Dave Booz	Revision of the RFC 2119 keywords and the set of normative statements - done in drafts a through g
14	Feb 10 2009	Mike Edwards	All changes accepted, comments removed.
15	Feb 10 2009	Mike Edwards	Issue 64 - Sections A1, B, 10, 9, 8
16	Feb 12, 2009	Ashok Malhotra	Issue 5 The single sca namespace is listed on the title page. Issue 32 clientAuthentication and serverAuthentication Issue 35 Conformance targets added to Appendix C Issue 48 Transaction defaults are not optional Issue 66 Tighten schema for intent Issue 67 Remove 'conversational'
17	Feb 16, 2009	Dave Booz	Issues 57, 69, 70, 71
CD02	Feb 21, 2009	Dave Booz	Editorial changes to make a CD
CD02-rev1	April 7, 2009	Dave Booz	Applied 72, 74,75,77
CD02-rev2	July 21, 2009	Dave Booz	Applied 81,84,85,86,95,96,98,99
CD02-rev3	Aug 12, 2009	Dave Booz	Applied 73,76,78,80,82,83,88,102
CD03-rev4	Sept 3, 2009	Dave Booz	Editorial cleanup to match OASIS templates
CD02-rev5	Nov 9, 2009	Dave Booz	Fixed latest URLs Applied: 79, 87, 90, 97, 100, 101, 103, 106, 107, 108
CD02-rev6	Nov 17, 2009	Dave Booz	Applied 94, 109
CD02-rev7	Jan 1, 2010	Dave Booz	Updated namespace to latest assembly Applied issues: 79,110,111,112,113,114,115
CD02-rev8	Mar 17, 2010	Dave Booz	Applied issue 93 Editorial updates to prepare for next CD
CD02-rev9	April 8, 2010	Ashok Malhotra, Dave Booz	More Editorial cleanup
CD03	May 5, 2010	Dave Booz	Applied 117, Front Matter and TOC updates

2533

2534