



SCA Policy Framework Version 1.1

Committee Draft 02/Public Review 01 – rev6-issue93

January 7 2010~~17 November 2009~~

Specification URIs:

This 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)

Previous Version:

<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.html>
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.doc>
<http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf> (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 SCA Policy TC

Chair(s):

David Booz, IBM <booz@us.ibm.com>
Ashok Malhotra, Oracle <ashok.malhotra@oracle.com>

Editor(s):

David Booz, IBM <booz@us.ibm.com>
Michael J. Edwards, IBM <mike.edwards@uk.ibm.com>
Ashok Malhotra, Oracle <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 03, "SCA Assembly Model Specification Version 1.1", March 2009.
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.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/200903 . 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, 2009. 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	Attachment Rules - PolicySets	2224
4.3	Direct Attachment of PolicySets	2224
4.4	External Attachment of PolicySets Mechanism	2423
4.4.1	The Form of the @attachTo Attribute	2423
4.4.2	Cases Where Multiple PolicySets are attached to a Single Artifact	2625
4.4.3	XPath Functions for the @attachTo Attribute	2725
4.4.3.1	Interface Related Functions	2726
4.4.3.2	Intent Based Functions	2827
4.4.3.3	URI Based Function	2827
4.5	Usage of @requires attribute for specifying intents	2928
4.5.1	Implementation Hierarchy of an Element	2928
4.5.2	Structural Hierarchy of an Element	2928
4.5.3	Combining Implementation and Structural Policy Data	3029
4.5.4	Examples	3029
4.6	Usage of Intent and Policy Set Attachment together	3234
4.7	Intents and PolicySets on Implementations and Component Types	3234
4.8	Intents on Interfaces	3334
4.9	BindingTypes and Related Intents	3332
4.10	Treatment of Components with Internal Wiring	3433
4.10.1	Determining Wire Validity and Configuration	3534
4.11	Preparing Services and References for External Connection	3534

4.12	Guided Selection of PolicySets using Intents	3634
4.12.1	Matching Intents and PolicySets	3635
5	Implementation Policies	3837
5.1	Natively Supported Intents.....	3938
5.2	Writing PolicySets for Implementation Policies	3938
5.2.1	Non WS-Policy Examples	4038
6	Roles and Responsibilities	4140
6.1	Policy Administrator	4140
6.2	Developer.....	4140
6.3	Assembler.....	4140
6.4	Deployer.....	4241
7	Security Policy	4342
7.1	SCA Security Intents.....	4342
7.2	Interaction Security Policy	4342
7.2.1	Qualifiers	4443
7.3	Implementation Security Policy Intent	4443
8	Reliability Policy.....	4544
8.1	Policy Intents	4544
8.2	End-to-end Reliable Messaging	4746
9	Transactions	4847
9.1	Out of Scope	4847
9.2	Common Transaction Patterns.....	4847
9.3	Summary of SCA transaction policies	4948
9.4	Global and local transactions.....	4948
9.4.1	Global transactions.....	4948
9.4.2	Local transactions	4948
9.5	Transaction implementation policy	5049
9.5.1	Managed and non-managed transactions.....	5049
9.5.2	OneWay Invocations	5150
9.6	Transaction interaction policies	5251
9.6.1	Handling Inbound Transaction Context.....	5251
9.6.2	Handling Outbound Transaction Context.....	5453
9.6.3	Combining implementation and interaction intents	5554
9.6.4	Web services binding for propagatesTransaction policy.....	5554
10	Miscellaneous Intents	5655
11	Conformance	5756
A	Schemas.....	5857
A.1	sca-policy.xsd	6257
B	XML Files.....	6559
B.1	Intent Definitions	6559
C	Conformance	7064
C.1	Conformance Targets	7064
C.2	Conformance Items	7064
D	Acknowledgements	7670

1 | E Revision History..... 7774

2

3

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	<code>docs.oasis-open.org/ns/opencsa/sca/200903</code> 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	<code>http://www.w3.org/2006/07/ws-policy</code>	[WS-Policy]
xs	<code>http://www.w3.org/2001/XMLSchema</code>	[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 03, “Service Component Architecture Assembly Model Specification Version 1.1”, March 2009.
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf>
- [SCA-Java-Annotations]** OASIS Committee Draft 02, “SCA Java Common Annotations and APIs Specification Version 1.1”, February 2009.

31		http://www.oasis-open.org/committees/download.php/31427/sca-javacaa-1.1-spec-cd02.pdf
32		
33	[SCA-WebServicesBinding]	
34		OASIS Committee Draft 01, "SCA Web Services Binding Specification Version 1.1", August 2008.
35		
36		http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd01.pdf
37		
38	[WSDL]	Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language
39		– Appendix http://www.w3.org/TR/2006/CR-wsdl20-20060327/
40	[WS-AtomicTransaction]	
41		Web Services Atomic Transaction (WS-AtomicTransaction)
42		http://docs.oasis-open.org/ws-tx/ws-atomic-transaction/2006/06/
43		
44	[WSDL-Ids]	SCA WSDL 1.1 Element Identifiers – forthcoming W3C Note
45		http://dev.w3.org/cvsweb/~checkout~/2006/ws-policy/wsdl11elementidentifiers.html
46		
47	[WS-Policy]	Web Services Policy (WS-Policy)
48		http://www.w3.org/TR/ws-policy
49	[WS-PolicyAttach]	Web Services Policy Attachment (WS-PolicyAttachment)
50		http://www.w3.org/TR/ws-policy-attachment
51	[XPath]	XML Path Language (XPath) Version 1.0.
52		http://www.w3.org/TR/xpath
53	[XML-Schema2]	XML Schema Part 2: Datatypes Second Edition XML Schema Part 2: Datatypes
54		Second Edition, Oct. 28 2004.
55		http://www.w3.org/TR/xmlschema-2/

56 1.4 Naming Conventions

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

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

67 2 Overview

68 2.1 Policies and PolicySets

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

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

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

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

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

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

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

96 2.2 Intents describe the requirements of Components, Services and 97 References

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

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

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

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

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

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

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

126 **2.3 Determining which policies apply to a particular wire**

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

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

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

3 Framework Model

151

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

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

3.1 Intents

160

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

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

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

181 **If the configured instance of a binding is in conflict with the intents and policy sets selected for that**
182 **instance, the SCA runtime MUST raise an error.**~~If the configured instance of a binding is in conflict with~~
183 ~~the intents and policy sets selected for that instance, the SCA runtime MUST raise an error.~~ [POL30001].

184 For example, a web service binding which requires the SOAP intent but which points to a WSDL binding
185 that does not specify SOAP.

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

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

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

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

202 Intents, their qualifiers and their defaults are defined using the pseudo schema in Snippet 3-1:
203

```
204 <intent name="xs:NCName"  
205     constrains = "list of QNames"?  
206     attachTo = "xs:string"  
207     requires="list of QNames"?  
208     excludes="list of QNames"?  
209     mutuallyExclusive="boolean"?  
210     intentType="xs:string"? >  
211   <description> xs:string.</description?>  
212   <qualifier name = "xs:string" default = "xs:boolean" ?>*</qualifier?>  
213     <description> xs:string.</description?>  
214 </intent?>
```

216 *Snippet 3-1: intent Pseudo-Schema*
217

218 Where the intent element has the following attributes:

- 219 • @name (1..1) - an NCName that defines the name of the intent. **The QName for an intent MUST be**
220 **unique amongst the set of intents in the SCA Domain. The QName for an intent MUST be unique**
221 **amongst the set of intents in the SCA Domain.** [POL30002]
 - 222 • @constrains (0..1) - a list of QNames that specifies the SCA constructs that this intent is meant to
223 configure. If a value is not specified for this attribute then the intent can apply to any SCA element.
224 Note that the “constrains” attribute can name an abstract element type, such as sca:binding in our
225 running example. This means that it will match against any binding used within an SCA composite
226 file. An SCA element can match @constrains if its type is in a substitution group.
 - 227 — @attachTo (0..1) - a string which is an XPath 1.0 expression identifying one or more elements in the
228 Domain. It is used to declare which set of elements the policySet is actually attached to. **The**
229 **contents of @attachTo MUST match the XPath 1.0 production Expr.** [POL30019] **The @attachTo**
230 **attribute uses the “Deployed Composite Infoset” as described in Appendix A “The Deployed**
231 **Composites Infoset”. See the section on “Attaching Intents and PolicySets to SCA Constructs” for**
232 **more details on how this attribute is used.**
 - 233 • @requires (0..1) - contains a list of QNames of intents which defines the set of all intents that the
234 referring intent requires. In essence, the referring intent requires all the intents named to be satisfied.
235 This attribute is used to compose an intent from a set of other intents. **Each QName in the @requires**
236 **attribute MUST be the QName of an intent in the SCA Domain. Each QName in the @requires**
237 **attribute MUST be the QName of an intent in the SCA Domain.** [POL30015] This use is further
238 described in [Section 3.3](#).
 - 239 • @excludes (0..1) - a list of QNames of intents that cannot be used with this intent. Intents might
240 describe a policy that is incompatible or otherwise unrealizable when specified with other intents, and
241 therefore are considered to be mutually exclusive. **Each QName in the @excludes attribute MUST be**
242 **the QName of an intent in the SCA Domain.** [POL30016]
- 243 Two intents are mutually exclusive when any of the following are true:
- 244 – One of the two intents lists the other intent in its @excludes list.
 - 245 – Both intents list the other intent in their respective @excludes list.
- 246 Where one intent is attached to an element of an SCA composite and another intent is attached to
247 one of the element’s parents, the intent(s) that are effectively attached to the element differs

248 depending on whether the two intents are mutually exclusive (see @excludes above and section
249 [4.7.4.5 Attaching intentsUsage of @requires attribute for specifying intents](#)).

- 250 • @mutuallyExclusive (0..1) - a boolean with a default of "false". If this attribute is present and has a
251 value of "true" it indicates that the qualified intents defined for this intent are mutually exclusive.
- 252 • @intentType attribute (0..1) defines whether the intent is an interaction intent or an implementation
253 intent. A value of "interaction", which is the default value, indicates that the intent is an interaction
254 intent. A value of "implementation" indicates that the intent is an implementation intent.

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

- 257 • @name (1..1) - declares the name of the qualifier. **The name of each qualifier MUST be unique within**
258 **the intent definition. The name of each qualifier MUST be unique within the intent definition.**
259 **[POL30005].**
- 260 • @default (0..1) - a boolean value with a default value of "false". If @default="true" the particular
261 qualifier is the default qualifier for the intent. **If an intent has more than one qualifier, one and only**
262 **one MUST be declared as the default qualifier. If an intent has more than one qualifier, one and only**
263 **one MUST be declared as the default qualifier. [POL30004]. If only one qualifier for an intent is given**
264 **it MUST be used as the default qualifier for the intent. If only one qualifier for an intent is given it**
265 **MUST be used as the default qualifier for the intent. [POL30025]**
- 266 • qualifier/description (0..1) - an xs:string that holds a textual description of the qualifier.

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

269

```
270 <intent name="confidentiality" constrains="sca:binding">  
271   <description>  
272     Communication through this binding must prevent  
273     unauthorized users from reading the messages.  
274   </description>  
275   <qualifier name="transport">  
276     <description>Automatic encryption by transport  
277     </description>  
278   </qualifier>  
279   <qualifier name="message" default='true'>  
280     <description>Encryption applied to each message  
281     </description>  
282   </qualifier>  
283 </intent>
```

284 *Snippet 3-2: Example intent Definition*

285

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

288 SCA normatively defines a set of core intents that all SCA implementations are expected to support, to
289 ensure a minimum level of portability. Users of SCA can define new intents, or extend the qualifier set of
290 existing intents. **An SCA Runtime MUST include in the Domain the set of intent definitions contained in**
291 **the Policy_Intents_Definitions.xml described in the appendix "Intent Definitions" of the SCA Policy**
292 **specification. An SCA Runtime MUST include in the Domain the set of intent definitions contained in the**
293 **Policy_Intents_Definitions.xml described in the appendix "Intent Definitions" of the SCA Policy**
294 **specification. [POL30024]** It is also good practice for the Domain to include concrete policies which satisfy
295 these intents (this may be achieved through the provision of appropriate binding types and
296 implementation types, augmented by policy sets that apply to those binding types and implementation
297 types).

298 The normatively defined intents in the SCA specification might evolve in future versions of this
299 specification. New intents could be added, additional qualifiers could be added to existing intents and the

300 default qualifier for existing intents could change. Such changes would cause the namespace for the SCA
301 specification to change.

302 3.2 Interaction Intents and Implementation Intents

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

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

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

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

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

317 3.3 Profile Intents

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

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

325 Requiring a profile intent is semantically identical to requiring the list of intents that are listed in its
326 @requires attribute. **If a profile intent is attached to an artifact, all the intents listed in its @requires**
327 **attribute MUST be satisfied as described in section 4.14.** ~~If a profile intent is attached to an artifact, all the~~
328 ~~intents listed in its @requires attribute MUST be satisfied as described in section 4.12.~~ [POL30007]

329 An example of a profile intent is an intent called **messageProtection** which is a shortcut for specifying
330 both **confidentiality** and **integrity**, where **integrity** means to protect against modification, usually by
331 signing. The intent definition is shown in [Snippet 3-3](#) ~~Snippet 3-3~~:

```
332  
333 <intent name="messageProtection"  
334   constrains="sca:binding"  
335   requires="confidentiality integrity">  
336   <description>  
337     Protect messages from unauthorized reading or modification.  
338   </description>  
339 </intent>
```

340 *Snippet 3-3: Example Profile Intent*

341 3.4 PolicySets

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

344 The pseudo schema for policySet is shown in Snippet 3-4:

```
345  
346 <policySet name="NCName"
```

```

347     provides="listOfQNames"?
348     appliesTo="xs:string"?
349     attachTo="xs:string"?
350     xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200903
351     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
352     <policySetReference name="xs:QName"/>*
353     <intentMap/>*
354     <xs:any>*
355 </policySet>

```

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

357

358 PolicySet has the attributes:

- 359 • @name (1..1) - the name for the policySet. The value of the @name attribute is the local part of a
360 QName. **The QName for a policySet MUST be unique amongst the set of policySets in the SCA**
361 **Domain. The QName for a policySet MUST be unique amongst the set of policySets in the SCA-**
362 **Domain.** [POL30017]
- 363 • @appliesTo (0..1) - a string which is an XPath 1.0 expression identifying one or more SCA constructs
364 this policySet can configure. **The contents of @appliesTo MUST match the XPath 1.0 [XPATH]**
365 **production Expr. The contents of @appliesTo MUST match the XPath 1.0 [XPATH] production Expr.**
366 [POL30018] The @appliesTo attribute uses the "Deployed Composites Infoset" as described in
367 [Appendix A The Deployed Composites Infoset](#) Section 4.4.1 "The Form of the @attachTo Attribute".
- 368 • @attachTo (0..1) - a string which is an XPath 1.0 expression identifying one or more elements in the
369 Domain. It is used to declare which set of elements the policySet is actually attached to. **The**
370 **contents of @attachTo MUST match the XPath 1.0 production Expr.** [POL30019] The @attachTo
371 attribute uses the "Deployed Composite Infoset" as described in Section 4.4.1 "The Form of the
372 @attachTo Attribute". See the section on ["Attaching Intents and PolicySets to SCA Constructs"](#) for
373 more details on how this attribute is used.
- 374 • @provides (0..1) - a list of intent QNames (that can be qualified), which declares the intents the
375 PolicySet provides.

376 PolicySet contains one or more of the element children

- 377 • intentMap element
- 378 • policySetReference element
- 379 • xs:any extensibility element

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

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

390 For example, the policySet element below declares that it provides

391 **serverAuthentication.message** and **reliability** for the "binding.ws" SCA binding.

```

392 <policySet name="SecureReliablePolicy"
393     provides="serverAuthentication.message exactlyOne"
394     appliesTo="//sca:binding.ws"
395     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
396     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
397

```

```

398 <wsp:PolicyAttachment>
399   <!-- policy expression and policy subject for
400     "basic server authentication" -->
401   ...
402 </wsp:PolicyAttachment>
403 <wsp:PolicyAttachment>
404 <!-- policy expression and policy subject for
405   "reliability" -->
406   ...
407 </wsp:PolicyAttachment>
408 </policySet>

```

409 *Snippet 3-5: Example policySet Definition*

410

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

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

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

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

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

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

439 | Note that section [4.12.14.10.1](#) on Wire Validity specifies that the strict version of the WS-Policy
440 intersection algorithm is used to establish wire validity and determine the policies to be used. The strict
441 version of policy intersection algorithm ignores the ignorable attribute on assertions. This means that the
442 ignorable facility of WS-Policy cannot be used in policySets.

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

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

447 3.4.1 IntentMaps

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

450 The pseudo-schema for intentMaps is given in Snippet 3-6:

451

```
452 <intentMap provides="xs:QName">  
453   <qualifier name="xs:string"?>  
454     <xs:any>*</xs:any>  
455   </qualifier>  
456 </intentMap>
```

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

458

459 **When a policySet element contains a set of intentMap children, the value of the @provides attribute of**
460 **each intentMap MUST correspond to an unqualified intent that is listed within the @provides attribute**
461 **value of the parent policySet element. When a policySet element contains a set of intentMap children, the**
462 **value of the @provides attribute of each intentMap MUST correspond to an unqualified intent that is listed**
463 **within the @provides attribute value of the parent policySet element. [POL30008]**

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

466 **For each qualifiable intent listed as a member of the @provides attribute list of a policySet element, there**
467 **MUST be no more than one corresponding intentMap element that declares the unqualified form of that**
468 **intent in its @provides attribute. In other words, each intentMap within a given policySet uniquely provides**
469 **for a specific intent. For each qualifiable intent listed as a member of the @provides attribute list of a**
470 **policySet element, there MUST be no more than one corresponding intentMap element that declares the**
471 **unqualified form of that intent in its @provides attribute. In other words, each intentMap within a given**
472 **policySet uniquely provides for a specific intent. [POL30010]**

473 **The @provides attribute value of each intentMap that is an immediate child of a policySet MUST be**
474 **included in the @provides attribute of the parent policySet. The @provides attribute value of each**
475 **intentMap that is an immediate child of a policySet MUST be included in the @provides attribute of the**
476 **parent policySet. [POL30021]**

477 An intentMap element contains qualifier element children. Each qualifier element corresponds to a
478 qualified intent where the unqualified form of that intent is the value of the @provides attribute value of
479 the parent intentMap. The qualified intent is either included explicitly in the value of the enclosing
480 policySet's @provides attribute or implicitly by that @provides attribute including the unqualified form of
481 the intent.

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

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

488

```
489 <policySet name="SecureMessagingPolicies"  
490   provides="confidentiality"  
491   appliesTo="binding.ws"  
492   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
493   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">  
494   <intentMap provides="confidentiality" >  
495     <qualifier name="transport">  
496       <wsp:PolicyAttachment>  
497         <!-- policy expression and policy subject for  
498         "transport" alternative -->
```

```

499     ...
500     </wsp:PolicyAttachment>
501     <wsp:PolicyAttachment>
502     ...
503     </wsp:PolicyAttachment>
504 </qualifier>
505 <qualifier name="message">
506     <wsp:PolicyAttachment>
507         <!-- policy expression and policy subject for
508             "message" alternative -->
509     ...
510     </wsp:PolicyAttachment>
511 </qualifier>
512 </intentMap>
513 </policySet>

```

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

515

516 PolicySets can embed policies that are defined in any policy language. Although WS-Policy is the most
517 common language for expressing interaction policies, it is possible to use other policy languages Snippet
518 3-8 is an example of a policySet that embeds a policy defined in a proprietary language. This policy
519 provides "serverAuthentication" for binding.ws.

520

```

521 <policySet name="AuthenticationPolicy"
522     provides="serverAuthentication"
523     appliesTo="binding.ws"
524     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
525     <e:policyConfiguration xmlns:e="http://example.com">
526         <e:authentication type = "X509"/>
527             <e:trustedCAStore type="JKS"/>
528             <e:keyStoreFile>Foo.jks</e:keyStoreFile>
529             <e:keyStorePassword>123</e:keyStorePassword>
530         </e:authentication>
531     </e:policyConfiguration>
532 </policySet>

```

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

534 3.4.2 Direct Inclusion of Policies within PolicySets

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

540 **Following the inclusion of all policySet references, when a policySet element directly contains**
541 **wsp:policyAttachment children or policies using extension elements, the set of policies specified as**
542 **children MUST satisfy all the intents expressed using the @provides attribute value of the policySet**
543 **element.** ~~Following the inclusion of all policySet references, when a policySet element directly contains~~
544 ~~wsp:policyAttachment children or policies using extension elements, the set of policies specified as~~
545 ~~children MUST satisfy all the intents expressed using the @provides attribute value of the policySet~~
546 ~~element.~~ [POL30011] The intent names in the @provides attribute of the policySet can include names of
547 profile intents.

548 3.4.3 Policy Set References

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

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

555 **The set of intents in the @provides attribute of a referenced policySet MUST be a subset of the set of**
556 **intents in the @provides attribute of the referencing policySet. The set of intents in the @provides**
557 **attribute of a referenced policySet MUST be a subset of the set of intents in the @provides attribute of the**
558 **referencing policySet.** [POL30013] Qualified intents are a subset of their parent qualifiable intent.

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

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

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

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

573 Snippet 3-9 is an example to illustrate the inclusion of two other policySets in a policySet element:

574

```
575 <policySet name="BasicAuthMsgProtSecurity"  
576     provides="serverAuthentication confidentiality"  
577     appliesTo="binding.ws"  
578     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
579   <policySetReference name="acme:ServerAuthenticationPolicies"/>  
580   <policySetReference name="acme:ConfidentialityPolicies"/>  
581 </policySet>
```

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

583

584 The policySet in Snippet 3-9 refers to policySets for **serverAuthentication** and
585 **confidentiality** and, by reference, provides policies and policy subject alternatives in these
586 domains.

587 If the policySets referred to in Snippet 3-9 have the following content:

588

```
589 <policySet name="ServerAuthenticationPolicies"  
590     provides="serverAuthentication"  
591     appliesTo="binding.ws"  
592     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
593   <wsp:PolicyAttachment>  
594     <!-- policy expression and policy subject for  
595           "basic server authentication" -->  
596     ...  
597   </wsp:PolicyAttachment>  
598 </policySet>  
599  
600 <policySet name="acme:ConfidentialityPolicies"  
601     provides="confidentiality"  
602     bindings="binding.ws"  
603     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
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 3-10: Example Included policySets for Snippet 3-9*

625

626 The result of the inclusion of policySets via policySetReferences would be semantically
627 equivalent to Snippet 3-11.

628

```

629 <policySet name="BasicAuthMsgProtSecurity"
630     provides="serverAuthentication confidentiality" appliesTo="binding.ws"
631     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
632     <wsp:PolicyAttachment>
633         <!-- policy expression and policy subject for
634             "basic server authentication" -->
635         ...
636     </wsp:PolicyAttachment>
637     <intentMap provides="confidentiality" >
638         <qualifier name="transport">
639             <wsp:PolicyAttachment>
640                 <!-- policy expression and policy subject for
641                     "transport" alternative -->
642                 ...
643             </wsp:PolicyAttachment>
644             <wsp:PolicyAttachment>
645                 ...
646             </wsp:PolicyAttachment>
647         </qualifier>
648         <qualifier name="message">
649             <wsp:PolicyAttachment>
650                 <!-- policy expression and policy subject for
651                     "message" alternative -->
652                 ...
653             </wsp:PolicyAttachment>
654         </qualifier>
655     </intentMap>
656 </policySet>

```

657 *Snippet 3-11: Equivalent policySet*

658 4 Attaching Intents and PolicySets to SCA Constructs

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

662 4.1 Attachment Rules — Intents

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

- 665 • Direct Attachment mechanism which is described in Section 4.2.
- 666 • External Attachment mechanism which is described in Section 4.3.

667 SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms
668 for policySet attachment. [POL40010] SCA implementations supporting only the External Attachment
669 mechanism MUST ignore the policy sets that are applicable via the Direct Attachment mechanism.
670 [POL40011] SCA implementations supporting only the Direct Attachment mechanism MUST ignore the
671 policy sets that are applicable via the External Attachment mechanism. [POL40012] SCA
672 implementations supporting both Direct Attachment and External Attachment mechanisms MUST ignore
673 policy sets applicable to any given SCA element via the Direct Attachment mechanism when there exist
674 policy sets applicable to the same SCA element via the External Attachment mechanism [POL40001]

675 4.14.2 Direct Attachment of Intents

676 Intents can be attached to any SCA element used in the definition of components and composites.
677 Intents are attached by using the **@requires** attribute or the <requires> child element. The @requires
678 attribute takes as its value a list of intent names. Similarly, the <requires> element takes as its value a list
679 of intent names. Intents can also be attached to interface definitions. For WSDL portType elements
680 (WSDL 1.1) the @requires attribute can be used to attach the list of intents that are needed by the
681 interface. Other interface languages can define their own mechanism for attaching a list of intents. Any
682 intents attached to an interface definition artifact, such as a WSDL portType, MUST be added to the
683 intents attached to the service or reference to which the interface definition applies. If no intents are
684 attached to the service or reference then the intents attached to the interface definition artifact become
685 the only intents attached to the service or reference. Any intents attached to an interface definition artifact,
686 such as a WSDL portType, MUST be added to the intents attached to the service or reference to which
687 the interface definition applies. If no intents are attached to the service or reference then the intents
688 attached to the interface definition artifact become the only intents attached to the service or reference.
689 [POL40027]

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

693 For example:

```
694  
695 <service requires="acme:IntentName1 acme:IntentName2">  
696   <binding.xxx/>  
697   ...  
698 </service>  
699  
700 <reference requires="acme:IntentName1 acme:IntentName2">  
701   <binding.xxx/>  
702   ...  
703 </reference>
```

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

```

705 <service>
706   <requires intents="acme:IntentName1 acme:IntentName2"/>
707   <binding.xxx/>
708   ...
709 </service>
710
711 <reference>
712   <requires intents="acme:IntentName1 acme:IntentName2"/>
713   <binding.xxx/>
714   ...
715 </reference>

```

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

717 **4.3 External Attachment of Intents**

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

722 An intent is attached to one or more elements through the @attachTo attribute of the intent.

723 During the deployment of SCA composites, all policySets within the Domain with an attachTo attribute
724 MUST be evaluated to determine which policySets are attached to the newly deployed composite.
725 [POL40013]

726 During the deployment of an SCA policySet, the behavior of an SCA runtime MUST take ONE of the
727 following forms:

- 728 • The policySet is immediately attached to all deployed composites which satisfy the @attachTo
729 attribute of the policySet.
- 730 • The policySet is attached to a deployed composite which satisfies the @attachTo attribute of the
731 policySet when the composite is re-deployed.

732

733 When External Attachment is used for both intents and policySets, intents must be attached before
734 policySets [POL40xxx]

735

736 **4.24.4 Attachment Rules - PolicySets**

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

- 739 • **Direct Attachment** mechanism which is described in Section-4.3.4.5-
- 740 • **External Attachment** mechanism which is described in Section-4.4.4.6

741 SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms
742 for policySet attachment. SCA runtimes MUST support at least one of the Direct Attachment and External
743 Attachment mechanisms for policySet attachment. [POL40010] SCA implementations supporting only the
744 External Attachment mechanism MUST ignore the policy sets that are applicable via the Direct
745 Attachment mechanism. SCA implementations supporting only the External Attachment mechanism
746 MUST ignore the policy sets that are applicable via the Direct Attachment mechanism. [POL40011] SCA
747 implementations supporting only the Direct Attachment mechanism MUST ignore the policy sets that are
748 applicable via the External Attachment mechanism. SCA implementations supporting only the Direct
749 Attachment mechanism MUST ignore the policy sets that are applicable via the External Attachment
750 mechanism. [POL40012] SCA implementations supporting both Direct Attachment and External
751 Attachment mechanisms MUST ignore policy sets applicable to any given SCA element via the Direct
752 Attachment mechanism when there exist policy sets applicable to the same SCA element via the External
753 Attachment mechanism. SCA implementations supporting both Direct Attachment and External Attachment

754 mechanisms MUST ignore policy sets applicable to any given SCA element via the Direct Attachment
755 mechanism when there exist policy sets applicable to the same SCA element via the External Attachment
756 mechanism [POL40001]

757 4.34.5 Direct Attachment of PolicySets

758 Direct Attachment of PolicySets can be achieved by

- 759 • Using the optional **@policySets** attribute of the SCA element
- 760 • Adding an optional child **<policySetAttachment/>** element to the SCA element

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

762 For example:

763

```
764 <service> or <reference>...  
765   <binding.binding-type policySets="listOfQNames">  
766   </binding.binding-type>  
767   ...  
768 </service> or </reference>
```

769 *Snippet 4-3: Example of @policySets on a service*

770

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

772

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

774 *Snippet 4-4: policySetAttachment Pseudo-Schema*

775

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

777

778 For example:

779

```
780 <service> or <reference>...  
781   <binding.binding-type>  
782     <policySetAttachment name="sns:EnterprisePolicySet">  
783   </binding.binding-type>  
784   ...  
785 </service> or </reference>
```

786 *Snippet 4-5: Example of policySetAttachment in a service or reference*

787

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

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

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

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

803

804 | See also section [4.14.14.12.1](#) for a discussion of how intents are used to guide the selection and
805 application of specific policySets.

806 **4.44.6 External Attachment of PolicySets Mechanism**

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

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

813 a) through the @attachTo attribute of the policySet

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

815 **During the deployment of SCA composites, all policySets within the Domain with an attachTo attribute**
816 **MUST be evaluated to determine which policySets are attached to the newly deployed composite.**
817 [\[POL40013\]](#)

818 **During the deployment of an SCA policySet, the behavior of an SCA runtime MUST take ONE of the**
819 **following forms:**

- 820 • **The policySet is immediately attached to all deployed composites which satisfy the @attachTo**
821 **attribute of the policySet.**

822 ~~—The policySet is attached to a deployed composite which satisfies the @attachTo attribute of the~~
823 ~~policySet when the composite is re-deployed. During the deployment of an SCA policySet, the~~
824 ~~behavior of an SCA runtime MUST take ONE of the following forms:~~

- 825 • ~~The policySet is immediately attached to all deployed composites which satisfy the @attachTo~~
826 ~~attribute of the policySet.~~

- 827 • ~~The policySet is attached to a deployed composite which satisfies the @attachTo attribute of the~~
828 ~~policySet when the composite is re-deployed.~~

829 [\[POL40026\]](#)

830

831 When External Attachment is used for both intents and policySets, intents must be attached before
832 policySets [\[POL40xxx\]](#)

833

834 ~~4.4.1 The Form of the @attachTo Attribute~~

835 ~~The @attachTo attribute of a policySet is an XPath1.0 expression identifying a SCA element to which the~~
836 ~~policySet is attached.—~~

837 ~~The XPath applies to the **Deployed Composites Infoset**—i.e. to all deployed SCA composite files [SCA-~~
838 ~~Assembly] in the Domain, with the special characteristics:~~

839 ~~1.—The Domain is treated as a special composite, with a blank name—""~~

840 ~~2.—The @attachTo XPath expression is evaluated against the Deployed Composite Infoset following the~~
841 ~~deployment of a deployment composite. Where one composite includes one or more other~~
842 ~~composites, it is the including composite which is addressed by the XPath and its contents are the~~
843 ~~result of preprocessing all of the include elements~~

844 ~~Where the policySet is intended to be specific to a particular component, the structuralURI [SCA-~~
845 ~~Assembly] of the component is used along with the URIRef() XPath function to attach a policySet to~~
846 ~~a specific use of a nested component. The XPath expression can make use of the unique-~~
847 ~~structuralURI to indicate specific use instances, where different policySets need to be used for those~~
848 ~~different instances.~~

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

851 ~~1. For direct attachment (using a @policySet attribute on an element or a <policySetAttachment/>~~
852 ~~subelement)~~

853 ~~2. By reference from another policySet element~~

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

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

859 ~~• the identification of elements to which specific intents apply.~~

860 ~~This permits the attachment of a policySet to be linked to specific intents on the target element –for~~
861 ~~example, a policySet relating to encryption of messages can be targeted to services and references~~
862 ~~which have the **confidentiality** intent applied.~~

863 ~~• the targeting of subelements of an interface, including operations and messages.~~

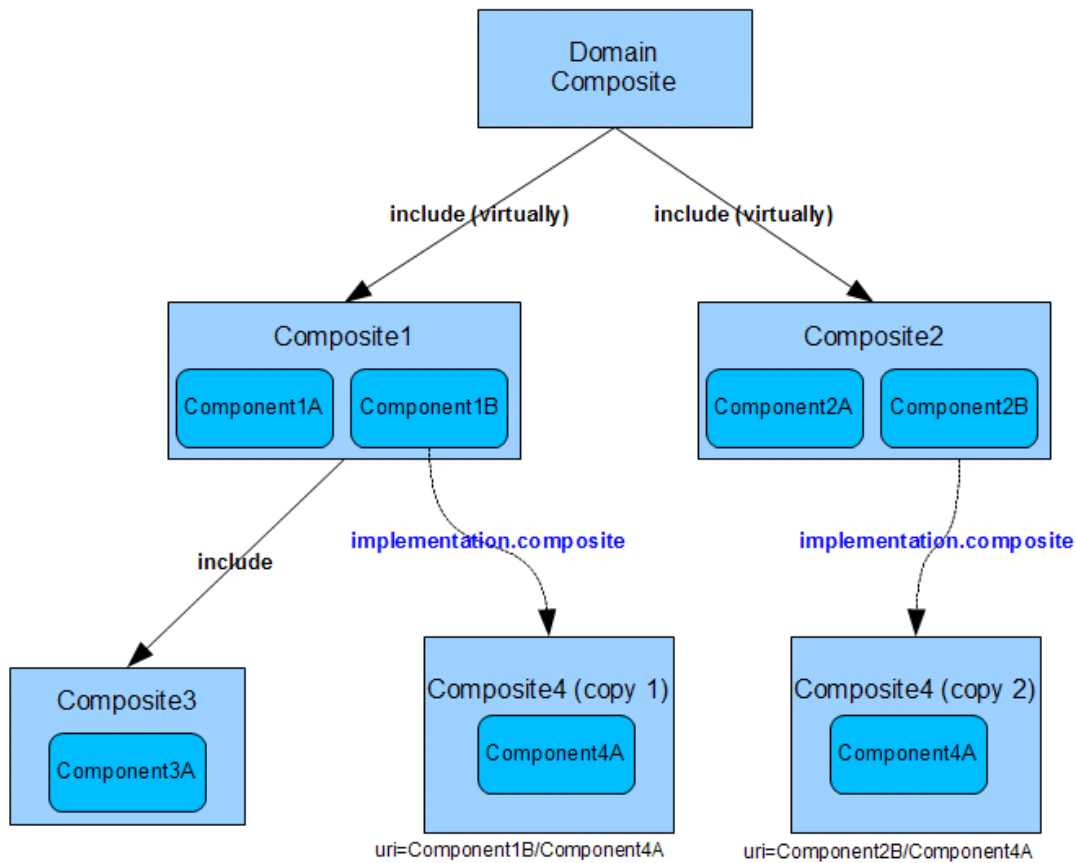
864 ~~This permits the attachment of a policySet to an individual operation or to an individual message~~
865 ~~within an interface, separately from the policies that apply to other operations or messages in the~~
866 ~~interface.~~

867 ~~• the targeting of a specific use of a component, through its unique structuralURI [SCA-Assembly].~~

868 ~~This permits the attachment of a policySet to a specific use of a component in one context, that can~~
869 ~~be different from the policySet(s) that are applied to other uses of the same component.~~

870 ~~Detail of the available XPath functions is given in the section "XPath Functions for the @attachTo~~
871 ~~Attribute".~~

872



873

874 *Figure 4-1 Example Domain Composite Infoset*

875

876 The SCA Domain in Figure 4-1 has been constructed from the composites and components shown in the
 877 figure. Composite1 and Composite2 were deployed into the Domain as described in [SCA-Assembly].
 878 Composite3 is included in Composite1 using the SCA include mechanism described in [SCA-Assembly].
 879 Composite4 is used as an implementation of Components 1B and 2B. Following the deployment of all the
 880 composites, the Domain contains:

- 881 • 3 Composites that can be addressed as part of the Deployed Composites InfoSet; Composite1,
 882 Composite2 and Composite4.
- 883 • all the components shown in the diagram. Components 1A, 2A, 3A, 4A (twice) are leaf
 884 components.

885

886 The following snippets show example usage of the @attachTo attribute and provide the outcome based
 887 on the Domain in Figure 4-1.

888

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

890 *Snippet 4-6: Example attachTo all Instances of a Name*

891

892 **attach to both instances of Component4A**

893

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

894

895 *Snippet 4-7: Example attachTo a Specific Instance via a Path*

896
897 *attach to the unique instance of Component4A when used by Component2B (Component2B is a*
898 *component at the Domain level)*

899
900

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

901 *Snippet 4-8: Example attachTo Instances with an intent*

902
903 *attach to the services of Component3A which have the intent "intent1" applied*

904
905

```
4. //component/binding.we
```

906 *Snippet 4-9: Example attachTo Instances with a binding*

907
908 *attach to the web services binding of all components with a service or reference with a Web services*
909 *binding*

910
911

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

912 *Snippet 4-10: Example attachTo a Specific Instance via Path and Name*

913
914 *attach to Component1A at the Domain level*

915 **4.4.24.6.1 Cases Where Multiple PolicySets are attached to a Single** 916 **Artifact**

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

920 **4.4.34.6.2 XPath Functions for the @attachTo Attribute**

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

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

- 925
- 926 • Picking out a specific interface
 - 927 • Picking out a specific operation in an interface
 - 928 • Picking out a specific message in an operation in an interface
 - 929 • Picking out artifacts with specific intents

929 **4.4.3.14.6.2.1 Interface Related Functions**

930 **InterfaceRef(InterfaceName)**

931 picks out an interface identified by InterfaceName

932 **OperationRef(InterfaceName/OperationName)**

933 picks out the operation OperationName in the interface InterfaceName

934 **MessageRef(InterfaceName/OperationName/MessageName)**

935 picks out the message MessageName in the operation OperationName in the interface
936 InterfaceName.

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

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

940 Examples of the Interface functions:

941

```
942 InterfaceRef( "MyInterface" )
```

943 *Snippet 4-644: Example use of InterfaceRef*

944

945 picks out an interface with the name "MyInterface"

946

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

948 *Snippet 4-742: Example use of OperationRef with a Path*

949

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

951

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

953 *Snippet 4-843: Example use of OperationRef without a Path*

954

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

956

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

958 *Snippet 4-944: Example use of MessageRef with a Path*

959

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

962

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

964 *Snippet 4-1045: Example use of MessageRef with a Path with Wildcards*

965

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

967 **4.4.3.24.6.2.2 Intent Based Functions**

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

971 **IntentRefs(IntentList)**

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

973

```
974 IntentRefs( "intent1" )
```

975 *Snippet 4-1146: Example use of InterntRef*

976

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

978

```
979 IntentRefs( "intent1 intent2" )
```

980 *Snippet 4-1247: Example use of IntentRef with Multiple intents*

981

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

983

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

985 *Snippet 4-1348: Example use of IntentRef with Not Operator*

986

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

988 4.4.3-34.6.2.3 **URI Based Function**

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

992 **URIRef(URI)**

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

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

995 Example:

996

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

998 *Snippet 4-1419: Example use of URIRef*

999

1000

1001

1002 picks out the particular use of a component – where component lowest_comp_name is used within the
1003 implementation of middle_comp_name within the implementation of the top-level (Domain level)
1004 component top_comp_name.

1005 4.54.7 **Attaching intents to SCA elements**

1006 A list of intents can be attached to any SCA element ~~by using the @requires attribute or the <requires>~~
1007 ~~subelement either directly or by external attachment as described in sections 4.2 and 4.3 above.~~

1008 The intents which apply to a given element ~~depend on~~ include:

- 1009 • the intents ~~expressed in its @requires attribute and/or its <requires> subelement~~ attached to it either
1010 directly or externally.
- 1011 • intents derived from the structural hierarchy of the element
- 1012 • intents derived from the implementation hierarchy of the element

1013 When computing the intents that apply to a particular element, the @constrains attribute of each relevant
1014 intent is checked against the element. If the intent in question does not apply to that element it is simply
1015 discarded.

1016 **Any two intents applied to a given element MUST NOT be mutually exclusive** [[POL40009](#)]. Specific
1017 examples are discussed later in this document.

1018 4.5.14.7.1 Implementation Hierarchy of an Element

1019 The **implementation hierarchy** occurs where a component configures an implementation and also
1020 where a composite promotes a service or reference of one of its components. The implementation
1021 hierarchy involves:

- 1022 • a composite service or composite reference element is in the implementation hierarchy of the
1023 component service/component reference element which they promote
- 1024 • the component element and its descendent elements (for example, service, reference,
1025 implementation) configure aspects of the implementation. Each of these elements is in the
1026 implementation hierarchy of the **corresponding** element in the componentType of the
1027 implementation.

1028 Rule 1: **The intents declared on elements lower in the implementation hierarchy of a given element MUST**
1029 **be applied to the element. The intents declared on elements lower in the implementation hierarchy of a**
1030 **given element MUST be applied to the element. [POL40014] A qualifiable intent expressed lower in the**
1031 **hierarchy can be qualified further up the hierarchy, in which case the qualified version of the intent MUST**
1032 **apply to the higher level element. A qualifiable intent expressed lower in the hierarchy can be qualified**
1033 **further up the hierarchy, in which case the qualified version of the intent MUST apply to the higher level**
1034 **element. [POL40004]**

1035 4.5.24.7.2 Structural Hierarchy of an Element

1036 The structural hierarchy of an element consists of its parent element, grandparent element and so on up
1037 to the <composite/> element in the composite file containing the element.

1038 As an example, for the composite in [Snippet 4-15](#) [Snippet 4-16](#):

```
1039  
1040 <composite name="C1" requires="i1">  
1041   <service name="CS" promotes="X/S">  
1042     <binding.ws requires="i2">  
1043   </service>  
1044   <component name="X">  
1045     <implementation.java class="foo"/>  
1046     <service name="S" requires="i3">  
1047   </component>  
1048 </composite>
```

1049 *Snippet 4-1520: Example Composite to Illustrate Structural Hierarchy*

1050
1051 - the structural hierarchy of the component service element with the name "S" is the component element
1052 named "X" and the composite element named "C1". Service "S" has intent "i3" and also has the intent "i1"
1053 if i1 is not mutually exclusive with i3.

1054 **Rule2: The intents declared on elements higher in the structural hierarchy of a given element MUST be**
1055 **applied to the element EXCEPT**

- 1056 • **if any of the inherited intents is mutually exclusive with an intent applied on the element, then the**
1057 **inherited intent MUST be ignored**

1058 **—if the overall set of intents from the element itself and from its structural hierarchy contains both an**
1059 **unqualified version and a qualified version of the same intent, the qualified version of the intent MUST**
1060 **be used. Rule2: The intents declared on elements higher in the structural hierarchy of a given element**
1061 **MUST be applied to the element EXCEPT**

- 1062 • **if any of the inherited intents is mutually exclusive with an intent applied on the element, then the**
1063 **inherited intent MUST be ignored**

- 1064 • **if the overall set of intents from the element itself and from its structural hierarchy contains both an**
1065 **unqualified version and a qualified version of the same intent, the qualified version of the intent MUST**
1066 **be used.**

1067 [POL40005]

1068 4.5.34.7.3 Combining Implementation and Structural Policy Data

1069 When there are intents present in both hierarchies implementation intents are calculated before the
1070 structural intents. In other words, when combining implementation hierarchy and structural hierarchy
1071 policy data, Rule 1 MUST be applied BEFORE Rule 2. [POL40015]

1072 Note that each of the elements in the hierarchy below a <component> element, such as <service/>,
1073 <reference/> or <binding/>, inherits intents from the equivalent elements in the componentType of the
1074 implementation used by the component. So the <service/> element of the <component> inherits any
1075 intents on the <service/> element with the same name in the <componentType> - and a <binding/>
1076 element under the service in the component inherits any intents on the <binding/> element of the service
1077 (with the same name) in the componentType. Errors caused by mutually exclusive intents appearing on
1078 corresponding elements in the component and on the componentType only occur when those elements
1079 match one-to-one. Mutually exclusive intents can validly occur on elements that are at different levels in
1080 the structural hierarchy (as defined in Rule 2).

1081 Note that it might often be the case that <binding/> elements will be specified in the structure under the
1082 <component/> element in the composite file (especially at the Domain level, where final deployment
1083 configuration is applied) - these elements might have no corresponding elements defined in the
1084 componentType structure. In this situation, the <binding/> elements don't acquire any intents from the
1085 componentType directly (ie there are no elements in the implementation hierarchy of the <binding/>
1086 elements), but those <binding/> elements will acquire intents "flowing down" their structural hierarchy as
1087 defined in Rule 2 - so, for example if the <service/> element is marked with @requires="confidentiality",
1088 the bindings of that service will all inherit that intent, assuming that they don't have their own exclusive
1089 intents specified.

1090 Also, for example, where say a component <service.../> element has an intent that is mutually exclusive
1091 with an intent in the componentType<service.../> element with the same name, it is an error, but this
1092 differs when compared with the case of the <component.../> element having an intent that is mutually
1093 exclusive with an intent on the componentType <service/> element - because they are at different
1094 structural levels: the intent on the <component/> is ignored for that <service/> element and there is no
1095 error.

1096 4.5.44.7.4 Examples

1097 As an example, consider the composite in ~~Snippet 4-17~~:[the snippet below](#):

1098

```
1099 <composite name="C1" requires="i1">  
1100   <service name="CS" promotes="X/S">  
1101     <binding.ws requires="i2">  
1102   </service>  
1103   <component name="X">  
1104     <implementation.java class="foo"/>  
1105     <service name="S" requires="i3">  
1106   </component>  
1107 </composite>
```

1108 *Snippet 4-~~1624~~: Example composite with intents*

1109

1110 ...the component service with name "S" has the service named "S" in the componentType of
1111 the implementation in its implementation hierarchy, and the composite service named "CS"
1112 has the component service named "S" in its implementation hierarchy. Service "CS"
1113 acquires the intent "i3" from service "S" - and also gets the intent "i1" from its containing
1114 composite "C1" IF i1 is not mutually exclusive with i3.

1115 When intents apply to an element following the rules described and where no policySets are
1116 attached to the element, the intents for the element can be used to select appropriate
1117 policySets during deployment, using the external attachment mechanism.

1118 Consider the composite in [Snippet 4-17](#)~~Snippet 4-18~~:

1119

```
1120 <composite requires="confidentiality">  
1121   <service name="foo" .../>  
1122   <reference name="bar" requires="confidentiality.message"/>  
1123 </composite>
```

1124 *Snippet 4-~~1722~~*: Example reference with intents

1125

1126 ...in this case, the composite declares that all of its services and references guarantee confidentiality in
1127 their communication, but the “bar” reference further qualifies that requirement to specifically require
1128 message-level security. The “foo” service element has the default qualifier specified for the confidentiality
1129 intent (which might be transport level security) while the “bar” reference has the **confidentiality.message**
1130 intent.

1131 Consider the variation in [Snippet 4-18](#)~~Snippet 4-19~~ where a qualified intent is specified at the composite
1132 level:

1133

```
1134 <composite requires="confidentiality.transport">  
1135   <service name="foo" .../>  
1136   <reference name="bar" requires="confidentiality.message"/>  
1137 </composite>
```

1138 *Snippet 4-~~1823~~*: Example Qualified intents

1139

1140 In this case, both the **confidentiality.transport** and the **confidentiality.message** intent
1141 are applied for the reference ‘bar’. If there are no bindings that support this combination, an
1142 error will be generated. However, since in some cases multiple qualifiers for the same intent
1143 can be valid or there might be bindings that support such combinations, the SCA
1144 specification allows this.

1145 It is also possible for a qualified intent to be further qualified. In our example, the
1146 **confidentiality.message** intent could be further qualified to indicate whether just the body of a message
1147 is protected, or the whole message (including headers) is protected. So, the second-level qualifiers might
1148 be “body” and “whole”. The default qualifier might be “whole”. If the “bar” reference from [Snippet](#)
1149 [4-18](#)~~Snippet 4-19~~ wanted only body confidentiality, it would state:

1150

```
1151 <reference name="bar" requires="acme:confidentiality.message.body"/>
```

1152 *Snippet 4-~~1924~~*: Example Second Level Qualifier

1153

1154 The definition of the second level of qualification for an intent follows the same rules. As with other
1155 qualified intents, the name of the intent is constructed using the name of the qualifiable intent, the
1156 delimiter “.”, and the name of the qualifier.

1157 **4.64.8 Usage of Intent and Policy Set Attachment together**

1158 As indicated above, it is possible to attach both intents and policySets to an SCA element during
1159 development. The most common use cases for attaching both intents and concrete policySets to an
1160 element are with binding and reference elements.

1161 When the @requires attribute or the <requires> subelement and one or both of the direct policySet
1162 attachment mechanisms are used together during development, it indicates the intention of the developer
1163 to configure the element, such as a binding, by the application of specific policySet(s) to this element.

1164 [The same behavior can be enabled by external attachment of intents and policySets.](#)

1165
1166 Developers who attach intents and policySets in conjunction with each other need to be aware of the
1167 implications of how the policySets are selected and how the intents are utilized to select specific
1168 intentMaps, override defaults, etc. The details are provided in the Section [Guided Selection of](#)
1169 [PolicySets using Intents.](#)

1170 **4.74.9 Intents and PolicySets on Implementations and Component** 1171 **Types**

1172 It is possible to specify intents and policySets within a component's implementation, which get exposed to
1173 SCA through the corresponding *component type*. How the intents or policies are specified within an
1174 implementation depends on the implementation technology. For example, Java can use an @requires
1175 annotation to specify intents.

1176 The intents and policySets specified within an implementation can be found on the
1177 <sca:implementation.*> and the <sca:service> and <sca:reference> elements of the component type.
1178 [The example below shows direct attachment of intents and policySets using the @requires and](#)
1179 [@policySets attributes:](#)

1180

```
1181 <componentType>  
1182   <implementation.* requires="listOfQNames" policySets="listOfQNames">  
1183     ...  
1184   </implementation>  
1185   <service name="myService" requires="listOfQNames"  
1186     policySets="listOfQNames">  
1187     ...  
1188   </service>  
1189   <reference name="myReference" requires="listOfQNames"  
1190     policySets="listOfQNames">  
1191     ...  
1192   </reference>  
1193   ...  
1194 </componentType>
```

1195 *Snippet 4-2025: Example of intents on an implementation*

1196

1197 Intents expressed in the component type are handled according to the rule defined for the implementation
1198 hierarchy. See [Intent rule 2](#)

1199 For explicitly listed policySets, the list in the component using the implementation can override policySets
1200 from the component type. [If a component has any policySets attached to it \(by any means\), then any](#)
1201 [policySets attached to the componentType MUST be ignored. If a component has any policySets attached](#)
1202 [to it \(by any means\), then any policySets attached to the componentType MUST be ignored. \[POL40006\].](#)
1203 [Similarly, If a component has any policySets attached to it \(by any means\), then any policySets attached](#)
1204 [to the componentType MUST be ignored. \[POL4xxxx\].](#)

1205 **4.84.10 Intents on Interfaces**

1206 Interfaces are used in association with SCA services and references. These interfaces can be declared
1207 in SCA composite files and also in SCA componentType files. The interfaces can be defined using a
1208 number of different interface definition languages which include WSDL, Java interfaces and C++ header
1209 files.

1210 It is possible for some interfaces to be referenced from an implementation rather than directly from any
1211 SCA files. An example of this usage is a Java implementation class file that has a reference declared
1212 that in turn uses a Java interface defined separately. When this occurs, the interface definition is treated
1213 from an SCA perspective as part of the componentType of the implementation, logically being part of the
1214 declaration of the related service or reference element.

1215 Both the declaration of interfaces in SCA and also the definitions of interfaces can carry policy-related
1216 information. In particular, both the declarations and the definitions can have either intents attached to
1217 them, or policySets attached to them - or both. For SCA declarations, the intents and policySets always
1218 apply to the whole of the interface (ie all operations and all messages within each operation). For
1219 interface definitions, intents and policySets can apply to the whole interface or they can apply only to
1220 specific operations within the interface or they can even apply only to specific messages within particular
1221 operations. (To see how this is done, refer to the places in the SCA specifications that deal with the
1222 relevant interface definition language)

1223 This means, in effect, that there are 4 places which can hold policy related information for interfaces:

- 1224 1. The interface definition file that is referenced from the component type.
- 1225 2. The interface declaration for a service or reference in the component type
- 1226 3. The interface definition file that is referenced from the component declaration in a composite
- 1227 4. The interface declaration within a component

1228 **When calculating the set of intents and set of policySets which apply to either a service element or to a**
1229 **reference element of a component, intents and policySets from the interface definition and from the**
1230 **interface declaration(s) MUST be applied to the service or reference element and to the binding**
1231 **element(s) belonging to that element. When calculating the set of intents and set of policySets which apply**
1232 **to either a service element or to a reference element of a component, intents and policySets from the**
1233 **interface definition and from the interface declaration(s) MUST be applied to the service or reference**
1234 **element and to the binding element(s) belonging to that element.** [POL40016]

1235 **The locations where interfaces are defined and where interfaces are declared in the componentType and**
1236 **in a component MUST be treated as part of the implementation hierarchy as defined in Section 4.5**
1237 **Attaching intents to SCA elements. The locations where interfaces are defined and where interfaces are**
1238 **declared in the componentType and in a component MUST be treated as part of the implementation**
1239 **hierarchy as defined in Section 4.5 Attaching intents to SCA elements.** [POL40019]

1240 4.94.11 BindingTypes and Related Intents

1241 SCA Binding types implement particular communication mechanisms for connecting components
1242 together. See detailed discussion in the [SCA Assembly Specification](#) [SCA-Assembly]. Some binding
1243 types can realize intents inherently by virtue of the kind of protocol technology they implement (e.g. an
1244 SSL binding would natively support confidentiality). For these kinds of binding types, it might be the case
1245 that using that binding type, without any additional configuration, provides a concrete realization of an
1246 intent. In addition, binding instances which are created by configuring a binding type might be able to
1247 provide some intents by virtue of their configuration. It is important to know, when selecting a binding to
1248 satisfy a set of intents, just what the binding types themselves can provide and what they can be
1249 configured to provide.

1250 The bindingType element is used to declare a class of binding available in a SCA Domain. The pseudo-
1251 schema for the bindingType element is shown in [Snippet 4-21](#) [Snippet 4-22](#):

1252

```
1253 <bindingType type="NCName"  
1254     alwaysProvides="listOfQNames"?  
1255     mayProvide="listOfQNames"?/>
```

1256 *Snippet 4-2126: bindingTypePseudo-Schema*

1257

- 1258 • @type (1..1) – declares the NCName of the bindingType, which is used to form the QName of the
1259 bindingType. **The QName of the bindingType MUST be unique amongst the set of bindingTypes in**
1260 **the SCA Domain. The QName of the bindingType MUST be unique amongst the set of bindingTypes**
1261 **in the SCA Domain.** [POL40020]
- 1262 • @alwaysProvides (0..1) – a list of intent QNames that are natively provided. A natively provided intent
1263 is hard-coded into the binding implementation. The function represented by the intent cannot be
1264 turned off.
- 1265 • @mayProvides (0..1) – a list of intent QNames that are natively provided by the binding
1266 implementation, but which are activated only when present in the intent set that is applied to a binding
1267 instance.

1268 **A binding implementation MUST implement all the intents listed in the @alwaysProvides and**
1269 **@mayProvides attributes. A binding implementation MUST implement all the intents listed in the**
1270 **@alwaysProvides and @mayProvides attributes.** [POL40021]

1271 The kind of intents a given binding might be capable of providing, beyond these inherent intents, are
1272 implied by the presence of policySets that declare the given binding in their @appliesTo attribute.

1273 For example, if the policySet in [Snippet 4-22](#) ~~Snippet 4-23~~ is available in a SCA Domain it says that the
1274 (example) foo:binding.ssl can provide “reliability” in addition to any other intents it might provide
1275 inherently.

1276

```
1277 <policySet name="ReliableSSL" provides="exactlyOnce"
1278     appliesTo="foo:binding.ssl">
1279     ...
1280 </policySet>
```

1281 *Snippet 4-~~22~~7: Example policySet Applied to a binding*

1282 **4.104.12 Treatment of Components with Internal Wiring**

1283 This section discusses the steps involved in the development and deployment of a component and its
1284 relationship to selection of bindings and policies for wiring services and references.

1285 The SCA developer starts by defining a component. Typically, this contains services and references. It
1286 can also have intents ~~attached~~defined at various locations within composite and component types as well
1287 as policySets ~~attached~~defined at various locations.

1288 Both for ease of development as well as for deployment, the wiring constraints to relate services and
1289 references need to be determined. This is accomplished by matching constraints of the services and
1290 references to those of corresponding references and services in other components.

1291 In this process, the intents, and the policySets that apply to both sides of a wire play an important role. In
1292 addition, concrete policies need to be selected that satisfy the intents for the service and the reference
1293 and are also compatible with each other. For services and references that make use of bidirectional
1294 interfaces, the same determination of matching policySets also has to take place for callbacks.

1295 Determining compatibility of wiring plays an important role prior to deployment as well as during the
1296 deployment phases of a component. For example, during development, it helps a developer to determine
1297 whether it is possible to wire services and references using the -policySets available in the development
1298 environment. During deployment, the wiring constraints determine whether wiring can be achievable. It
1299 also aids in adding additional concrete policies or making adjustments to concrete policies in order to
1300 deliver the constraints. Here are the concepts that are needed in making wiring decisions:

- 1301 • The set of intents that individually apply to *each* service or reference.
- 1302 • When possible the intents that are applied to the service, the reference and callback (if any) at the
1303 other end of the wire. This set is called the *required intent set* and only applies when dealing with a
1304 wire connecting two components within the same SCA Domain. When external connections are
1305 involved, from clients or to services that are outside the SCA domain, intents are only available for the

1306 end of the connection that is inside the domain. See Section "[Preparing Services and References](#)
1307 [for External Connection](#)" for more details.

1308 • The policySets that apply to each service or reference.

1309 The set of provided intents for a binding instance is the union of the set of intents listed in the
1310 "alwaysProvides" attribute and the set of intents listed in the "mayProvides" attribute of its binding type.
1311 The capabilities represented by the "alwaysProvides" intent set are always present, irrespective of the
1312 configuration of the binding instance. Each capability represented by the "mayProvides" intent set is only
1313 present when the list of intents applied to the binding instance (either applied directly, or inherited)
1314 contains the particular intent (or a qualified version of that intent, if the intent set contains an unqualified
1315 form of a qualifiable intent). When an

1316 -intent is directly provided by the binding type, there is no need to apply a policy set that provides that
1317 intent.

1318 When bidirectional interfaces are in use, the same process of selecting policySets to provide the intents is
1319 also performed for the callback bindings.

1320 **4.10.14.12.1 Determining Wire Validity and Configuration**

1321 The above approach determines the policySets that are used in conjunction with the binding instances
1322 listed for services and references. For services and references that are resolved using SCA wires, the
1323 policySets chosen on each side of the wire might or might not be compatible. The following approach is
1324 used to determine whether they are compatible and whether the wire is valid. If the wire
1325 uses a bidirectional interface, then the following technique ensures that valid configured
1326 policySets can be found for both directions of the bidirectional interface.

1327 ~~The SCA runtime MUST determine the compatibility of the policySets at each end of a wire using the~~
1328 ~~compatibility rules of the policy language used for those policySets. The SCA runtime MUST determine~~
1329 ~~the compatibility of the policySets at each end of a wire using the compatibility rules of the policy~~
1330 ~~language used for those policySets. [POL40022] The policySets at each end of a wire MUST be~~
1331 ~~incompatible if they use different policy languages. The policySets at each end of a wire MUST be~~
1332 ~~incompatible if they use different policy languages. [POL40023] However, there is a special case worth~~
1333 mentioning:

1334 • If both sides of the wire use identical policySets (by referring to the same policySet by its QName in
1335 both sides of the wire), then they are compatible.

1336 ~~Where the policy language in use for a wire is WS-Policy, strict WS-Policy intersection MUST be used to~~
1337 ~~determine policy compatibility. Where the policy language in use for a wire is WS-Policy, strict WS-Policy~~
1338 ~~intersection MUST be used to determine policy compatibility. [POL40024]~~

1339 ~~In order for a reference to connect to a particular service, the policies of the reference MUST intersect~~
1340 ~~with the policies of the service. In order for a reference to connect to a particular service, the policies of~~
1341 ~~the reference MUST intersect with the policies of the service. [POL40025]~~

1342 **4.114.13 Preparing Services and References for External** 1343 **Connection**

1344 Services and references are sometimes not intended for SCA wiring, but for communication with software
1345 that is outside of the SCA domain. References can contain bindings that specify the endpoint address of
1346 a service that exists outside of the current SCA domain. Services can specify bindings that can be
1347 exposed to clients that are outside of the SCA domain.

1348 ~~Matching service/reference policies across the SCA Domain boundary MUST use WS-Policy compatibility~~
1349 ~~(strict WS-Policy intersection) if the policies are expressed in WS-Policy syntax. Matching~~
1350 ~~service/reference policies across the SCA Domain boundary MUST use WS-Policy compatibility (strict~~
1351 ~~WS-Policy intersection) if the policies are expressed in WS-Policy syntax. [POL40007] For other policy~~
1352 languages, the policy language defines the comparison semantics.

1353 For external services and references that make use of bidirectional interfaces, the same determination
1354 of matching policies has to also take place for the callback.

1355 The policies that apply to the service/reference are computed as discussed in [Guided Selection of](#)
1356 [PolicySets using Intents](#).

1357 **4.12.14 Guided Selection of PolicySets using Intents**

1358 This section describes the selection of concrete policies that provide a set of intents
1359 expressed for an element. The purpose is to construct the set of concrete policies that are attached to an
1360 element taking into account the explicitly declared policySets that are attached to an element as well as
1361 policySets that are externally attached. The aim is to satisfy all of the intents expressed for each element.

1362 If the unqualified form of a qualifiable intent is attached to an element, it can be satisfied by a policySet
1363 that specifies any one of qualified forms of the intent in the value of its @provides attribute, or it can be
1364 satisfied by a policySet which @provides the unqualified form of the intent. If the qualified form of the
1365 intent is attached to an element then it can be satisfied only by a policy that @provides that qualified form
1366 of the intent.

1367 **4.12.14.14.1 Matching Intents and PolicySets**

1368 **Note: In the following, the following rule is observed when an intent set is computed.**

1369 When a profile intent is encountered in either a global @requires attribute, an intent/@requires attribute, a
1370 <requires> subelement or a policySet/@provides attribute, the profile intent is immediately replaced by
1371 the intents that it composes (i.e. all the intents that appear in the profile intent's @requires attribute). This
1372 rule is applied recursively until profile intents do not appear in an intent set. [This is stated generally here,
1373 in order to not have to restate this at multiple places].

1374 The **required intent set** that is attached to an element is:

- 1375 1. The set of intents ~~specified in the element's @requires attribute, attached to the element either by~~
1376 ~~direct attachment or using external attachment via the mechanisms described in sections 4.2 and 4.3.~~
- 1377 2. add any intents found in any related interface definition or declaration, as described in the section
1378 [4.10 Intents on Interfaces](#).
- 1379 3. add any intents found on elements below the target element in its implementation hierarchy as
1380 defined in Rule 1 in Section 4.5
- 1381 4. add any intents ~~found in the @requires attributes and <requires> subelements of attached to~~ each
1382 ancestor element in the element's structural hierarchy as defined in [Rule 2](#) in Section 4.5
- 1383 5. ~~removeless~~ any intents that do not include the target element's type in their @constrains attribute.
- 1384 6. remove the unqualified version of an intent if the set also contains a qualified version of that intent

1385 ~~If the required intent set contains a mutually exclusive pair of intents the SCA runtime MUST reject the~~
1386 ~~document containing the element and raise an error. If the required intent set contains a mutually~~
1387 ~~exclusive pair of intents the SCA runtime MUST reject the document containing the element and raise an~~
1388 ~~error.~~ [POL40017]

1389 The **directly provided intent set** for an element is the set of intents listed in the @alwaysProvides
1390 attribute combined with the set of intents listed in the @mayProvides attribute of the bindingType or
1391 implementationType declaration for a binding or implementation element respectively.

1392 The **set of PolicySets attached to an element** include those **explicitly specified** using the @policySets
1393 attribute or the <policySetAttachment/> element and those which are **externally attached**.

1394 A policySet **applies to** a target element if the result of the XPath expression contained in the policySet's
1395 @appliesTo attribute, when evaluated against the document containing the target element, includes the
1396 target element. For example, @appliesTo="binding.ws[@impl='axis']" matches any binding.ws element
1397 that has an @impl attribute value of 'axis'.

1398 The set of **explicitly specified** policySets for an element is:

- 1399 1. The union of the policySets specified in the element's @policySets attribute and those specified in
1400 any <policySetAttachment/> child element(s).
- 1401 2. add the policySets declared in the @policySets attributes and <policySetAttachment/> elements from
1402 elements in the structural hierarchy of the element.
- 1403 3. remove any policySet where the policySet does not apply to the target element.
1404 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*

1405 The set of **externally attached** policySets for an element is:

- 1406 1. Each <PolicySet/> in the Domain where the element is targeted by the @attachTo attribute of the
1407 policySet
- 1408 2. remove any policySet where the policySet does not apply to the target element.
1409 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*

1410 A policySet **provides an intent** if any of the statements are true:

- 1411 1. The intent is contained in the **policySet** @provides list of the policySet.
- 1412 2. The intent is a qualified intent and the unqualified form of the intent is contained in the **policySet**
1413 @provides list of the policySet.
- 1414 3. The policySet @provides list contains a qualified form of the intent (where the intent is qualifiable).

1415 **All intents in the required intent set for an element SHOULD be provided by the directly provided intents**
1416 **set and the set of policySets that apply to the element. All intents in the required intent set for an element**
1417 **SHOULD be provided by the directly provided intents set and the set of policySets that apply to the**
1418 **element.** [POL40018]

1419 If the combination of implementationType / bindingType / collection of policySets does not satisfy all of
1420 the intents which apply to the element, the configuration is not valid. However, an SCA Runtime can allow
1421 a deployer to force deployment even in the presence of such errors as long as a warning is issued or
1422 some other indication is provided that deployment has been forced. Details of the behavior of the
1423 deployer in such situations are not specified in this specification.

5 Implementation Policies

1424

1425 The basic model for Implementation Policies is very similar to the model for interaction policies described
1426 above. Abstract QoS requirements, in the form of intents, can be associated with SCA component
1427 implementations to indicate implementation policy requirements. These abstract capabilities are mapped
1428 to concrete policies via policySets at deployment time. Alternatively, policies can be associated directly
1429 with component implementations using policySets. [Intents and policySets can be associated with an
1430 implementation using any of the mechanisms described above.](#)

1431 Snippet 5-1 shows [how one way of associating intents can be associated](#) with an implementation:
1432

1433

1434

```
1434 <component name="xs:NCName" ... >  
1435   <implementation.* ... requires="listOfQNames">  
1436     ...  
1437   </implementation>  
1438   ...  
1439 </component>
```

1440 *Snippet 5-1: Example of intents Associated with an implementation*

1441

1442 If, for example, one of the intent names in the value of the @requires attribute is 'logging', this indicates
1443 that all messages to and from the component [have](#) to be logged. The technology used to implement the
1444 logging is unspecified. Specific technology is selected when the intent is mapped to a policySet (unless
1445 the implementation type has native support for the intent, as described in the next section). A list of
1446 implementation intents can [also](#) be specified by any ancestor element of the <sca:implementation>
1447 element. The effective list of implementation intents is the union of intents specified on the
1448 implementation element and all its ancestors.

1449 In addition, one or more policySets can be specified directly by associating them with the implementation
1450 of a component.

1451

```
1452 <component name="xs:NCName" ... >  
1453   <implementation.* ... policySets="listOfQNames">  
1454     ...  
1455   </implementation>  
1456   ...  
1457 </component>
```

1458 *Snippet 5-2: Example of policySets Associated with an implementation*

1459

1460 Snippet 5-2 shows how intents and policySets can be specified on a component. It is also possible to
1461 specify intents and policySets within the implementation. How this is done is defined by the
1462 implementation type.

1463 The intents and policy sets are specified on the <sca:implementation.*> element within the component
1464 type. This is important because intent and policy set definitions need to be able to specify that they
1465 constrain an appropriate implementation type.

1466

```
1467 <componentType>  
1468   <implementation.* requires="listOfQNames" policySets="listOfQNames">  
1469     ...  
1470   </implementation>  
1471   ...
```

1472 </componentType>

1473 Snippet 5-3: intents and policySets Constraining an implementation

1474

1475 When applying policies, the intents attached to the implementation are added to the intents attached to
1476 the using component. For ~~both intents and policySets the explicitly listed policySets~~, the list associated
1477 ~~with~~ the component can override intents and policySets from the componentType.

1478 Some implementation intents are targeted at <binding/> elements rather than at <implementation/>
1479 elements. This occurs in cases where there is a need to influence the operation of the binding
1480 implementation code rather than the code directly related to the implementation itself. Implementation
1481 elements of this kind will have a @constrains attribute pointing to a binding element, with a @intentType
1482 of "implementation".

1483 5.1 Natively Supported Intents

1484 Each implementation type (e.g. <sca:implementation.java> or <sca:implementation.bpel>) has an
1485 **implementation type definition** within the SCA Domain. An implementation type definition is declared
1486 using an implementationType element within a <definitions/> declaration. The pseudo-schema for the
1487 implementationType element is shown in Snippet 5-4:

1488

```
1489 <implementationType type="QName"  
1490 alwaysProvides="listOfQNames"? mayProvide="listOfQNames"? />
```

1491 Snippet 5-4: implementationType Pseudo-Schema

1492

1493 The implementation Type element has the following attributes:

- 1494 • **name : QName (1..1)** - the name of the implementationType. The implementationType name attribute
1495 MUST be the QName of an XSD global element definition used for implementation elements of that
1496 type. The implementationType name attribute MUST be the QName of an XSD global element
1497 definition used for implementation elements of that type. [POL50001] For example:
1498 "sca:implementation.java".
- 1499 • **alwaysProvides : list of QNames (0..1)** - a set of intents. The intents in the alwaysProvides set are
1500 always provided by this implementation type, whether the intents are attached to the using
1501 component or not.
- 1502 • **mayProvide : list of QNames (0..1)** - a set of intents. The intents in the mayProvide set are provided
1503 by this implementation type if the intent in question is attached to the using component.

1504 5.2 Writing PolicySets for Implementation Policies

1505 The @appliesTo and @attachTo attributes for a policySet takes an XPath expression that is applied to a
1506 service, reference, binding or an implementation element. For implementation policies, in most cases, all
1507 that is needed is the QName of the implementation type. Implementation policies can be expressed using
1508 any policy language (which is to say, any configuration language). For example, XACML or EJB-style
1509 annotations can be used to declare authorization policies. Other capabilities could be configured using
1510 completely proprietary configuration formats.

1511 For example, a policySet declared to turn on trace-level logging for a BPEL component could be
1512 declared as is Snippet 5-5:

1513

```
1514 <policySet name="loggingPolicy" provides="acme:logging.trace"  
1515 appliesTo="sca:implementation.bpel" ...>  
1516 <acme:processLogging level="3"/>  
1517 </policySet>
```


1518 *Snippet 5-5: Example policySet Applied to implemenation.bpel*

1519 **5.2.1 Non WS-Policy Examples**

1520 Authorization policies expressed in XACML [could](#) be used in the framework in two ways:

- 1521 1. Embed XACML expressions directly in the PolicyAttachment element using the extensibility elements
1522 discussed above, or
- 1523 2. Define WS-Policy assertions to wrap XACML expressions.

1524 For EJB-style authorization policy, [the same approach could be used](#):

- 1525 1. Embed EJB-annotations in the PolicyAttachment element using the extensibility elements discussed
1526 above, or
- 1527 2. Use the WS-Policy assertions defined as wrappers for EJB annotations.

1528 6 Roles and Responsibilities

1529 There are 4 roles that are significant for the SCA Policy Framework. The following is a list of the roles and
1530 the artifacts that the role creates:

- 1531 • Policy Administrator – policySet definitions and intent definitions
- 1532 • Developer – Implementations and component types
- 1533 • Assembler - Composites
- 1534 • Deployer – Composites and the SCA Domain (including the logical Domain-level composite)

1535 6.1 Policy Administrator

1536 An intent represents a requirement that a developer or assembler can make, which ultimately have to be
1537 satisfied at runtime. The full definition of the requirement is the informal text description in the intent
1538 definition.

1539 The **policy administrator**'s job is to both define the intents that are available and to define the policySets
1540 that represent the concrete realization of those informal descriptions for some set of binding type or
1541 implementation types. See the sections on intent and policySet definitions for the details of those
1542 definitions.

1543 6.2 Developer

1544 When it is possible for a component to be written without assuming a specific binding type for its services
1545 and references, then the **developer** uses intents to specify requirements in a binding neutral way.

1546 If the developer requires a specific binding type for a component, then the developer can specify bindings
1547 and policySets with the implementation of the component. Those bindings and policySets will be
1548 represented in the component type for the implementation (although that component type might be
1549 generated from the implementation).

1550 If any of the policySets used for the implementation include intentMaps, then the default choice for the
1551 intentMap can be overridden by an assembler or deployer by requiring a qualified intent that is present in
1552 the intentMap.

1553 6.3 Assembler

1554 An **assembler** creates composites. Because composites are implementations, an assembler is like a
1555 developer, except that the implementations created by an assembler are composites made up of other
1556 components wired together. So, like other developers, the assembler can specify intents or bindings or
1557 policySets on any service or reference of the composite.

1558 However, in addition the definition of composite-level services and references, it is also possible for the
1559 assembler to use the policy framework to further configure components within the composite. The
1560 assembler can add additional requirements to any component's services or references or to the
1561 component itself (for implementation policies). The assembler can also override the bindings or
1562 policySets used for the component. See the assembly specification's description of overriding rules for
1563 details on overriding.

1564 As a shortcut, an assembler can also specify intents and policySets on any element in the composite
1565 definition, which has the same effect as specifying those intents and policySets on every applicable
1566 binding or implementation below that element (where applicability is determined by the @appliesTo
1567 attribute of the policySet definition or the @constrains attribute of the intent definition).

1568 **6.4 Deployer**

1569 A **deployer** deploys implementations (typically composites) into the SCA Domain. It is the
1570 deployers job to make the final decisions about all configurable aspects of an implementation that is to be
1571 deployed and to make sure that all intents are satisfied.

1572 If the deployer determines that an implementation is correctly configured as it is, then the implementation
1573 can be deployed directly. However, more typically, the deployer will create a new composite, which
1574 contains a component for each implementation to be deployed along with any changes to the bindings or
1575 policySets that the deployer desires.

1576 When the deployer is determining whether the existing list of policySets is correct for a component, the
1577 deployer needs to consider both the explicitly listed policySets as well as the policySets that will be
1578 chosen according to the algorithm specified in [Guided Selection of PolicySets using Intents](#).

7 Security Policy

1579

1580 The SCA Security Model provides SCA developers the flexibility to specify the necessary level of security
1581 protection for their components to satisfy business requirements without the burden of understanding
1582 detailed security mechanisms.

1583 The SCA Policy framework distinguishes between two types of policies: *interaction policy* and
1584 *implementation policy*. Interaction policy governs the communications between clients and service
1585 providers and typically applies to Services and References. In the security space, interaction policy is
1586 concerned with client and service provider authentication and message protection requirements.
1587 Implementation policy governs security constraints on service implementations and typically applies to
1588 Components. In the security space, implementation policy concerns include access control, identity
1589 delegation, and other security quality of service characteristics that are pertinent to the service
1590 implementations.

1591 The SCA security interaction policy can be specified via intents or policySets. Intents represent security
1592 quality of service requirements at a high abstraction level, independent from security protocols, while
1593 policySets specify concrete policies at a detailed level, which are typically security protocol specific.

1594 The SCA security policy can be specified either in an SCA composite or by using the External Policy
1595 Attachment Mechanism or by annotations in the implementation code. Language-specific annotations are
1596 described in the respective language Client and Implementation specifications.

7.1 SCA Security Policy Intents

1597

1598 The SCA security specification defines the following intents to specify interaction policy:

1599 serverAuthentication, clientAuthentication, confidentiality, and integrity.

- 1600 • **serverAuthentication** – ~~When serverAuthentication is present, an SCA runtime MUST ensure that~~
1601 ~~the server is authenticated by the client.~~~~When serverAuthentication is present, an SCA runtime MUST~~
1602 ~~ensure that the server is authenticated by the client.~~ [POL70013]
- 1603 • **clientAuthentication** – ~~When clientAuthentication is present, an SCA runtime MUST ensure that the~~
1604 ~~client is authenticated by the server.~~~~When clientAuthentication is present, an SCA runtime MUST~~
1605 ~~ensure that the client is authenticated by the server.~~ [POL70014]
- 1606 • **authentication** – this is a profile intent that requires only clientAuthentication. It is included for
1607 backwards compatibility.
- 1608 • **mutualAuthentication** – this is a profile intent that includes the serverAuthentication and the
1609 clientAuthentication intents just described.
- 1610 • **confidentiality** – the confidentiality intent is used to indicate that the contents of a message are
1611 accessible only to those authorized to have access (typically the service client and the service
1612 provider). A common approach is to encrypt the message, although other methods are possible.
1613 ~~When confidentiality is present, an SCA Runtime MUST ensure that only authorized entities can view~~
1614 ~~the contents of a message.~~~~When confidentiality is present, an SCA Runtime MUST ensure that only~~
1615 ~~authorized entities can view the contents of a message.~~ [POL70009]
- 1616 • **integrity** – the integrity intent is used to indicate that assurance is that the contents of a message
1617 have not been tampered with and altered between sender and receiver. A common approach is to
1618 digitally sign the message, although other methods are possible.~~When integrity is present, an SCA~~
1619 ~~Runtime MUST ensure that the contents of a message are not altered.~~~~When integrity is present, an~~
1620 ~~SCA Runtime MUST ensure that the contents of a message are not altered.~~ [POL70010]

1621 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1622 7.2 Interaction Security Policy

1623 Any one of the three security intents can be further qualified to specify more specific business
1624 requirements. Two qualifiers are defined by the SCA security specification: transport and message, which
1625 can be applied to any of the above three intent's.

1626 7.2.1 Qualifiers

1627 **transport** – the transport qualifier specifies that the qualified intent is realized at the transport or transfer
1628 layer of the communication protocol, such as HTTPS. **When a serverAuthentication, clientAuthentication,
1629 confidentiality or integrity intent is qualified by message, an SCA Runtime MUST delegate
1630 serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the message layer
1631 of the communication protocol. When a serverAuthentication, clientAuthentication, confidentiality or
1632 integrity intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication,
1633 clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication
1634 protocol.** [POL70011]

1635 **message** – the message qualifier specifies that the qualified intent is realized at the message level of the
1636 communication protocol. **When a serverAuthentication, clientAuthentication, confidentiality or integrity
1637 intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication,
1638 clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication
1639 protocol. When a serverAuthentication, clientAuthentication, confidentiality or integrity intent is qualified by
1640 message, an SCA Runtime MUST delegate serverAuthentication, clientAuthentication, confidentiality and
1641 integrity, respectively, to the message layer of the communication protocol.** [POL70012]

1642
1643 Snippet 7-1 shows the usage of intents and qualified intents.

```
1644  
1645 <composite name="example" requires="confidentiality">  
1646   <service name="foo"/>  
1647   ...  
1648   <reference name="bar" requires="confidentiality.message"/>  
1649 </composite>
```

1650 *Snippet 7-1: Example using Qualified Intents*

1651
1652 In this case, the composite declares that all of its services and references have to guarantee
1653 confidentiality in their communication by setting requires="confidentiality". This applies to the "foo"
1654 service. However, the "bar" reference further qualifies that requirement to specifically require message-
1655 level security by setting requires="confidentiality.message".

1656 7.3 Implementation Security Policy Intent

1657 The SCA Security specification defines the **authorization** intent to specify implementation policy.

1658 **authorization** – the authorization intent is used to indicate that a client needs to be authorized before
1659 being allowed to use the service. Being authorized means that a check is made as to whether any
1660 policies apply to the client attempting to use the service, and if so, those policies govern whether or not
1661 the client is allowed access. **When authorization is present, an SCA Runtime MUST ensure that the client
1662 is authorized to use the service. When authorization is present, an SCA Runtime MUST ensure that the
1663 client is authorized to use the service.** [POL70001]

1664 This unqualified authorization intent implies that basic "Subject-Action-Resource" authorization support is
1665 required, where Subject may be as simple as a single identifier representing the identity of the client,
1666 Action may be a single identifier representing the operation the client intends to apply to the Resource,
1667 and the Resource may be a single identifier representing the identity of the Resource to which the Action
1668 is intended to be applied.

8 Reliability Policy

1669

1670 Failures can affect the communication between a service consumer and a service provider.

1671 Depending on the characteristics of the binding, these failures could cause messages to be redelivered,
1672 delivered in a different order than they were originally sent out or even worse, could cause messages to
1673 be lost. Some transports like JMS provide built-in reliability features such as “at least once” and “exactly
1674 once” message delivery. Other transports like HTTP need to have additional layers built on top of them to
1675 provide some of these features.

1676 The events that occur due to failures in communication can affect the outcome of the service invocation.
1677 For an implementation of a stock trade service, a message redelivery could result in a new trade. A client
1678 (i.e. consumer) of the same service could receive a fault message if trade orders are not delivered to the
1679 service implementation in the order they were sent out. In some cases, these failures could have dramatic
1680 consequences.

1681 An SCA developer can anticipate some types of failures and work around them in service
1682 implementations. For example, the implementation of a stock trade service could be designed to support
1683 duplicate message detection. An implementation of a purchase order service could have built in logic that
1684 orders the incoming messages. In these cases, service implementations don't need the binding layers to
1685 provide these reliability features (e.g. duplicate message detection, message ordering). However, this
1686 comes at a cost: extra complexity is built in the service implementation. Along with business logic, the
1687 service implementation has additional logic that handles these failures.

1688 Although service implementations can work around some of these types of failures, it is worth noting that
1689 workarounds are not always possible. A message can be lost or expire even before it is delivered to the
1690 service implementation.

1691 Instead of handling some of these issues in the service implementation, a better way is to use a binding
1692 or a protocol that supports reliable messaging. This is better, not just because it simplifies application
1693 development, it can also lead to better throughput. For example, there is less need for application-level
1694 acknowledgement messages. A binding supports reliable messaging if it provides features such as
1695 message delivery guarantees, duplicate message detection and message ordering.

1696 It is very important for the SCA developer to be able to require, at design-time, a binding or protocol that
1697 supports reliable messaging. SCA defines a set of policy intents that can be used for specifying reliable
1698 messaging Quality of Service requirements. These reliable messaging intents establish a contract
1699 between the binding layer and the application layer (i.e. service implementation or the service consumer
1700 implementation) (see below).

1701 **8.1 Reliability Policy Intents**

1702 Based on the use-cases described above, the following policy intents are defined:

1703 1. **atLeastOnce** - The binding implementation guarantees that a message that is successfully sent by a
1704 service consumer is delivered to the destination (i.e. service implementation). The message could be
1705 delivered more than once to the service implementation. **When atLeastOnce is present, an SCA
1706 Runtime MUST deliver a message to the destination service implementation, and MAY deliver
1707 duplicates of a message to the service implementation. When atLeastOnce is present, an SCA
1708 Runtime MUST deliver a message to the destination service implementation, and MAY deliver
1709 duplicates of a message to the service implementation. [POL80001]**

1710 The binding implementation guarantees that a message that is successfully sent by a service
1711 implementation is delivered to the destination (i.e. service consumer). The message could be
1712 delivered more than once to the service consumer.

1713 2. **atMostOnce** - The binding implementation guarantees that a message that is successfully sent by a
1714 service consumer is not delivered more than once to the service implementation. The binding
1715 implementation does not guarantee that the message is delivered to the service implementation.

1716 | ~~When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service~~
1717 | ~~implementation, and MUST NOT deliver duplicates of a message to the service implementation.~~
1718 | ~~When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service~~
1719 | ~~implementation, and MUST NOT deliver duplicates of a message to the service implementation.~~
1720 | [POL80002]

1721 | The binding implementation guarantees that a message that is successfully sent by a service
1722 | implementation is not delivered more than once to the service consumer. The binding implementation
1723 | does not guarantee that the message is delivered to the service consumer.

1724 | 3. **ordered** – The binding implementation guarantees that the messages sent by a service client via a
1725 | single service reference are delivered to the target service implementation in the order in which they
1726 | were sent by the service client. This intent does not guarantee that messages that are sent by a
1727 | service client are delivered to the service implementation. Note that this intent has nothing to say
1728 | about the ordering of messages sent via different service references by a single service client, even if
1729 | the same service implementation is targeted by each of the service references. ~~When *ordered* is~~
1730 | ~~present, an SCA Runtime MUST deliver messages sent by a single source to a single destination~~
1731 | ~~service implementation in the order that the messages were sent by that source.~~
1732 | ~~When *ordered* is~~
1733 | ~~present, an SCA Runtime MUST deliver messages sent by a single source to a single destination~~
1734 | ~~service implementation in the order that the messages were sent by that source.~~ [POL80003]

1734 | For service interfaces that involve messages being sent back from the service implementation to the
1735 | service client (eg. a service with a callback interface), for this intent, the binding implementation
1736 | guarantees that the messages sent by the service implementation over a given wire are delivered to
1737 | the service client in the order in which they were sent by the service implementation. This intent does
1738 | not guarantee that messages that are sent by the service implementation are delivered to the service
1739 | consumer.

1740 | 4. **exactlyOnce** - The binding implementation guarantees that a message sent by a service consumer is
1741 | delivered to the service implementation. Also, the binding implementation guarantees that the
1742 | message is not delivered more than once to the service implementation. ~~When *exactlyOnce* is~~
1743 | ~~present, an SCA Runtime MUST deliver a message to the destination service implementation and~~
1744 | ~~MUST NOT deliver duplicates of a message to the service implementation.~~
1745 | ~~When *exactlyOnce* is~~
1746 | ~~present, an SCA Runtime MUST deliver a message to the destination service implementation and~~
1747 | ~~MUST NOT deliver duplicates of a message to the service implementation.~~ [POL80004]

1747 | The binding implementation guarantees that a message sent by a service implementation is delivered
1748 | to the service consumer. Also, the binding implementation guarantees that the message is not
1749 | delivered more than once to the service consumer.

1750 | NOTE: This is a profile intent, which is composed of *atLeastOnce* and *atMostOnce*.

1751 | This is the most reliable intent since it guarantees the following:

- 1752 | – message delivery – all the messages sent by a sender are delivered to the service
1753 | implementation (i.e. Java class, BPEL process, etc.).
- 1754 | – duplicate message detection and elimination – a message sent by a sender is not processed
1755 | more than once by the service implementation.

1756 | The formal definitions of these intents are in the [Intent Definitions appendix](#).

1757 | How can a binding implementation guarantee that a message that it receives is delivered to the service
1758 | implementation? One way to do it is by persisting the message and keeping redelivering it until it is
1759 | processed by the service implementation. That way, if the system crashes after delivery but while
1760 | processing it, the message will be redelivered on restart and processed again. Since a message could be
1761 | delivered multiple times to the service implementation, this technique usually requires the service
1762 | implementation to perform duplicate message detection. However, that is not always possible. Often
1763 | times service implementations that perform critical operations are designed without having support for
1764 | duplicate message detection. Therefore, they cannot *process* an incoming
1765 | message more than once.

1766 Also, consider the scenario where a message is delivered to a service implementation that does not
1767 handle duplicates - the system crashes after a message is delivered to the service implementation but
1768 before it is completely processed. Does the underlying layer redeliver the message on restart? If it did
1769 that, there is a risk that some critical operations (e.g. sending out a JMS message or updating a DB table)
1770 will be executed again when the message is processed. On the other hand, if the underlying layer does
1771 not redeliver the message, there is a risk that the message is never completely processed.

1772 This issue cannot be safely solved unless all the critical operations performed by the service
1773 implementation are running in a transaction. Therefore, *exactlyOnce* cannot be assured without involving
1774 the service implementation. In other words, an *exactlyOnce* message delivery does not guarantee
1775 *exactlyOnce* message processing unless the service implementation is transactional. It's worth noting that
1776 this is a necessary condition but not sufficient. The underlying layer (e.g. binding implementation,
1777 container) would have to ensure that a message is not redelivered to the service implementation after the
1778 transaction is committed. As an example, a way to ensure it when the binding uses JMS is by making
1779 sure the operation that acknowledges the message is executed in the same transaction the service
1780 implementation is running in.

1781 **8.2 End-to-end Reliable Messaging**

1782 Failures can occur at different points in the message path: in the binding layer on the sender side, in the
1783 transport layer or in the binding layer on the receiver side. The SCA service developer doesn't really care
1784 where the failure occurs. Whether a message was lost due to a network failure or due to a crash of the
1785 machine where the service is deployed, is not that important. What is important is that the contract
1786 between the application layer (i.e. service implementation or service consumer) and the binding layer is
1787 not violated (e.g. a message that was successfully transmitted by a sender is always delivered to the
1788 destination; a message that was successfully transmitted by a sender is not delivered more than once to
1789 the service implementation, etc). It is worth noting that the binding layer could throw an exception when a
1790 sender (e.g. service consumer, service implementation) sends a message out. This is not considered a
1791 successful message transmission.

1792 In order to ensure the semantics of the reliable messaging intents, the entire message path, which is
1793 composed of the binding layer on the client side, the transport layer and the binding layer on the service
1794 side, has to be reliable.

1795 9 Transactions

1796 SCA recognizes that the presence or absence of infrastructure for ACID transaction coordination has a
1797 direct effect on how business logic is coded. In the absence of ACID transactions, developers have to
1798 provide logic that coordinates the outcome, compensates for failures, etc. In the presence of ACID
1799 transactions, the underlying infrastructure is responsible for ensuring the ACID nature of all interactions.
1800 SCA provides declarative mechanisms for describing the transactional environment needed by the
1801 business logic.

1802 Components that use a synchronous interaction style can be part of a single, distributed ACID transaction
1803 within which all transaction resources are coordinated to either atomically commit or rollback. The
1804 transmission or receipt of oneway messages can, depending on the transport binding, be coordinated as
1805 part of an ACID transaction as illustrated in the [OneWay Invocations](#) section below.
1806 Well-known, higher-level patterns such as store-and-forward queuing can be accomplished by composing
1807 transacted one-way messages with reliable-messaging policies.

1808 This document describes the set of abstract policy intents – both implementation intents and interaction
1809 intents – that can be used to describe the requirements on a concrete service component and binding
1810 respectively.

1811 9.1 Out of Scope

1812 The following topics are outside the scope of this document:

- 1813 • The means by which transactions are created, propagated and established as part of an execution
1814 context. These are details of the SCA runtime provider and binding provider.
- 1815 • The means by which a transactional resource manager (RM) is accessed. These include, but are not
1816 restricted to:
 - 1817 – abstracting an RM as an `sca:component`
 - 1818 – accessing an RM directly in a language-specific and RM-specific fashion
 - 1819 – abstracting an RM as an `sca:binding`

1820 9.2 Common Transaction Patterns

1821 In the absence of any transaction policies there is no explicit transactional behavior defined for the SCA
1822 service component or the interactions in which it is involved and the transactional behavior is
1823 environment-specific. An SCA runtime provider can choose to define an out of band default transactional
1824 behavior that applies in the absence of any transaction policies.

1825 Environment-specific default transactional behavior can be overridden by specifying transactional intents
1826 described in this document. The most common transaction patterns can be summarized:

1827 **Managed, shared global transaction pattern** – the service always runs in a global transaction context
1828 regardless of whether the requester runs under a global transaction. If the requester does run under a
1829 transaction, the service runs under the same transaction. Any outbound, synchronous request-response
1830 messages will – unless explicitly directed otherwise – propagate the service’s transaction context. This
1831 pattern offers the highest degree of data integrity by ensuring that any transactional updates are
1832 committed atomically

1833 **Managed, local transaction pattern** – the service always runs in a managed local transaction context
1834 regardless of whether the requester runs under a transaction. Any outbound messages will not propagate
1835 any transaction context. This pattern is advisable for services that wish the SCA runtime to demarcate
1836 any resource manager local transactions and do not require the overhead of atomicity.

1837 The use of transaction policies to specify these patterns is illustrated later in Table 9-2.

1838 9.3 Summary of SCA Transaction Policies

1839 This specification defines implementation and interaction policies that relate to transactional QoS in
1840 components and their interactions. The SCA transaction policies are specified as intents which represent
1841 the transaction quality of service behavior offered by specific component implementations or bindings.

1842 SCA transaction policy can be specified either in an SCA composite or annotatively in the implementation
1843 code. Language-specific annotations are described in the respective language binding specifications, for
1844 example the [SCA Java Common Annotations and APIs specification](#) [SCA-Java-Annotations].

1845 This specification defines the following implementation transaction policies:

- 1846 • `managedTransaction` – Describes the service component’s transactional environment.
- 1847 • `transactedOneWay` and `immediateOneWay` – two mutually exclusive intents that describe whether
1848 the SCA runtime will process `OneWay` messages immediately or will enqueue (from a client
1849 perspective) and dequeue (from a service perspective) a `OneWay` message as part of a global
1850 transaction.

1851 This specification also defines the following interaction transaction policies:

- 1852 • `propagatesTransaction` and `suspendsTransaction` – two mutually exclusive intents that describe
1853 whether the SCA runtime propagates any transaction context to a service or reference on a
1854 synchronous invocation.

1855 Finally, this specification defines a profile intent called `managedSharedTransaction` that combines the
1856 `managedTransaction` intent and the `propagatesTransaction` intent so that the ***managed, shared global***
1857 ***transaction pattern*** is easier to configure.

1858 9.4 Global and local transactions

1859 This specification describes “managed transactions” in terms of either “global” or “local” transactions. The
1860 “managed” aspect of managed transactions refers to the transaction environment provided by the SCA
1861 runtime for the business component. Business components can interact with other business components
1862 and with resource managers. The managed transaction environment defines the transactional context
1863 under which such interactions occur.

1864 9.4.1 Global transactions

1865 From an SCA perspective, a global transaction is a unit of work scope within which transactional work is
1866 atomic. If multiple transactional resource managers are accessed under a global transaction then the
1867 transactional work is coordinated to either atomically commit or rollback regardless using a 2PC protocol.
1868 A global transaction can be propagated on synchronous invocations between components – depending
1869 on the interaction intents described in this specification - such that multiple, remote service providers can
1870 execute distributed requests under the same global transaction.

1871 9.4.2 Local transactions

1872 From a resource manager perspective a resource manager local transaction (RMLT) is simply the
1873 absence of a global transaction. But from an SCA perspective it is not enough to simply declare that a
1874 piece of business logic runs without a global transaction context. Business logic might need to access
1875 transactional resource managers without the presence of a global transaction. The business logic
1876 developer still needs to know the expected semantic of making one or more calls to one or more resource
1877 managers, and needs to know when and/or how the resource managers local transactions will be
1878 committed. The term *local transaction containment* (LTC) is used to describe the SCA environment where
1879 there is no global transaction. The boundaries of an LTC are scoped to a remotable service provider
1880 method and are not propagated on invocations between components. Unlike the resources in a global
1881 transaction, RMLTs coordinated within a LTC can fail independently.

1882

1883 The two most common patterns for components using resource managers outside a global transaction
1884 are:

- 1885 • The application desires each interaction with a resource manager to commit after every interaction.
1886 This is the default behavior provided by the **noManagedTransaction** policy (defined below in
1887 Transaction implementation policy) in the absence of explicit use of RMLT verbs by the application.
- 1888 • The application desires each interaction with a resource manager to be part of an extended local
1889 transaction that is committed at the end of the method. This behavior is specified by the
1890 **managedTransaction.local** policy (defined below in Transaction implementation policy).

1891 While an application can use interfaces provided by the resource adapter to explicitly demarcate resource
1892 manager local transactions (RMLT), this is a generally undesirable burden on applications, which typically
1893 prefer all transaction considerations to be managed by the SCA runtime. In addition, once an application
1894 codes to a resource manager local transaction interface, it might never be redeployed with a different
1895 transaction environment since local transaction interfaces might not be used in the presence of a global
1896 transaction. This specification defines intents to support both these common patterns in order to provide
1897 portability for applications regardless of whether they run under a global transaction or not.

1898 9.5 Transaction implementation policy

1899 9.5.1 Managed and non-managed transactions

1900 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents describe the
1901 transactional environment needed by a service component or composite. SCA provides transaction
1902 environments that are managed by the SCA runtime in order to remove the burden of coding transaction
1903 APIs directly into the business logic. The **managedTransaction** and **noManagedTransaction** intents
1904 can be attached to the `sca:composite` or `sca:componentType` elements.

1905 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents are defined as
1906 follows:

- 1907 • **managedTransaction** – a managed transaction environment is necessary in order to run this
1908 component. The specific type of managedTransaction needed is not constrained. The valid qualifiers
1909 for this intent are mutually exclusive.
 - 1910 – **managedTransaction.global** – There has to be an atomic transaction in order to run this
1911 component. **For a component marked with managedTransaction.global, the SCA runtime**
1912 **MUST ensure that a global transaction is present before dispatching any method on the**
1913 **component.~~For a component marked with managedTransaction.global, the SCA runtime~~
1914 ~~MUST ensure that a global transaction is present before dispatching any method on the~~
1915 ~~component.~~ [POL90003] The SCA runtime uses any transaction propagated from the client
1916 or else begins and completes a new transaction. See the **propagatesTransaction** intent
1917 below for more details.**
 - 1918 – **managedTransaction.local** – indicates that the component cannot tolerate running as part
1919 of a global transaction. **A component marked with managedTransaction.local MUST run**
1920 **within a local transaction containment (LTC) that is started and ended by the SCA runtime.**~~A~~
1921 ~~component marked with managedTransaction.local MUST run within a local transaction~~
1922 ~~containment (LTC) that is started and ended by the SCA runtime.~~ [POL90004] Any global
1923 transaction context that is propagated to the hosting SCA runtime is not visible to the target
1924 component. Any interaction under this policy with a resource manager is performed in an
1925 extended resource manager local transaction (RMLT). Upon successful completion of the
1926 invoked service method, any RMLTs are implicitly requested to commit by the SCA runtime.
1927 Note that, unlike the resources in a global transaction, RMLTs so coordinated in a LTC can
1928 fail independently. If the invoked service method completes with a non-business exception
1929 then any RMLTs are implicitly rolled back by the SCA runtime. In this context a business
1930 exception is any exception that is declared on the component interface and is therefore
1931 anticipated by the component implementation. The manner in which exceptions are declared
1932 on component interfaces is specific to the interface type – for example, Java interface types

1933 declare Java exceptions, WSDL interface types define wsdl:faults. **Local transactions MUST**
1934 **NOT be propagated outbound across remotable interfaces.** ~~Local transactions MUST NOT be~~
1935 ~~propagated outbound across remotable interfaces.~~ [POL90006]

1936 • **noManagedTransaction** – indicates that the component runs without a managed transaction, under
1937 neither a global transaction nor an LTC. **A transaction that is propagated to the hosting SCA runtime**
1938 **MUST NOT be joined by the hosting runtime on behalf of a component marked with**
1939 **noManagedtransaction.** ~~A transaction that is propagated to the hosting SCA runtime MUST NOT be~~
1940 ~~joined by the hosting runtime on behalf of a component marked with noManagedtransaction.~~

1941 [POL90007] When interacting with a resource manager under this policy, the application (and not the
1942 SCA runtime) is responsible for controlling any resource manager local transaction boundaries, using
1943 resource-provider specific interfaces (for example a Java implementation accessing a JDBC provider
1944 has to choose whether a Connection is set to autoCommit(true) or else it has to call the Connection
1945 commit or rollback method). SCA defines no APIs for interacting with resource managers.

1946 • **(absent)** – The absence of a transaction implementation intent leads to runtime-specific behavior. A
1947 runtime that supports global transaction coordination can choose to provide a default behavior that is
1948 the managed, shared global transaction pattern but it is not mandated to do so.

1949 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1950 9.5.2 OneWay Invocations

1951 When a client uses a reference and sends a OneWay message then any client transaction context is not
1952 propagated. However, the OneWay invocation on the reference can itself be **transacted**. Similarly, from a
1953 service perspective, any received OneWay message cannot propagate a transaction context but the
1954 delivery of the OneWay message can be **transacted**. A **transacted** OneWay message is a one-way
1955 message that - because of the capability of the service or reference binding - can be enqueued (from a
1956 client perspective) or dequeued (from a service perspective) as part of a global transaction.

1957 SCA defines two mutually exclusive implementation intents, **transactedOneWay** and
1958 **immediateOneWay**, that determine whether OneWay messages are transacted or delivered immediately.

1959 Either of these intents can be attached to the sca:service or sca:reference elements or they can be
1960 attached to the sca:component element, indicating that the intent applies to any service or reference
1961 element children.

1962 The intents are defined as follows:

1963 • **transactedOneWay** – **When a reference is marked as transactedOneWay, any OneWay invocation**
1964 **messages MUST be transacted as part of a client global transaction.** ~~When a reference is marked as~~
1965 ~~transactedOneWay, any OneWay invocation messages MUST be transacted as part of a client global~~
1966 ~~transaction.~~ [POL90008]

1967 **If the client component is not configured to run under a global transaction or if the binding does not**
1968 **support transactional message sending, then a reference MUST NOT be marked as**
1969 **transactedOneWay.** ~~If the client component is not configured to run under a global transaction or if the~~
1970 ~~binding does not support transactional message sending, then a reference MUST NOT be marked as~~
1971 ~~transactedOneWay.~~ [POL90009] **If a service is marked as transactedOneWay, any OneWay**

1972 **invocation message MUST be received from the transport binding in a transacted fashion, under the**
1973 **target service's global transaction.** ~~If a service is marked as transactedOneWay, any OneWay~~
1974 ~~invocation message MUST be received from the transport binding in a transacted fashion, under the~~
1975 ~~target service's global transaction.~~ [POL90010] The receipt of the message from the binding is not

1976 committed until the service transaction commits; if the service transaction is rolled back the the
1977 message remains available for receipt under a different service transaction. **If the component is not**
1978 **configured to run under a global transaction or if the binding does not support transactional message**
1979 **receipt, then a service MUST NOT be marked as transactedOneWay.** ~~If the component is not~~
1980 ~~configured to run under a global transaction or if the binding does not support transactional message~~
1981 ~~receipt, then a service MUST NOT be marked as transactedOneWay.~~ [POL90011]

1982 • **immediateOneWay** – **When applied to a reference indicates that any OneWay invocation messages**
1983 **MUST be sent immediately regardless of any client transaction.** ~~When applied to a reference indicates~~

1984 that any OneWay invocation messages MUST be sent immediately regardless of any client
 1985 transaction. [POL90012] When applied to a service indicates that any OneWay invocation MUST be
 1986 received immediately regardless of any target service transaction. [POL90013] The outcome of any
 1987 transaction under which an immediateOneWay message is processed has no effect on the
 1988 processing (sending or receipt) of that message.

1989 The absence of either intent leads to runtime-specific behavior. The SCA runtime can send or receive a
 1990 OneWay message immediately or as part of any sender/receiver transaction. The results of combining
 1991 this intent and the **managedTransaction** implementation policy of the component sending or receiving
 1992 the transacted OneWay invocation are summarized low.below in Table 9-1.

transacted/immediate intent	managedTransaction (client or service implementation intent)	Results
transactedOneWay	managedTransaction.global	OneWay interaction (either client message enqueue or target service dequeue) is committed as part of the global transaction.
transactedOneWay	managedTransaction.local or noManagedTransaction	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.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.

1993
 1994 Table 9-1 Transacted OneWay interaction intent

1995
 1996 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1997 9.6 Transaction interaction policies

1998 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can be attached
 1999 either to an interface (e.g. Java annotation or WSDL attribute) or explicitly to an sca:service and
 2000 sca:reference XML element to describe how any client transaction context will be made available and
 2001 used by the target service component. Section [9.6.19.6.4](#) considers how these intents apply to service
 2002 elements and Section [9.6.29.6.2](#) considers how these intents apply to reference elements.

2003 The formal definitions of these intents are in the [Intent Definitions appendix](#).

2004 **9.6.1 Handling Inbound Transaction Context**

2005 The mutually exclusive *propagatesTransaction* and *suspendsTransaction* intents can be attached to
 2006 an sca:service XML element to describe how a propagated transaction context is handled by the SCA
 2007 runtime, prior to dispatching a service component. If the service requester is running within a transaction
 2008 and the service interaction policy is to propagate that transaction, then the primary business effects of the
 2009 provider’s operation are coordinated as part of the client's transaction – if the client rolls back its
 2010 transaction, then work associated with the provider's operation will also be rolled back. This allows clients
 2011 to know that no compensation business logic is necessary since transaction rollback can be used.

2012 These intents specify a contract that has to be implemented by the SCA runtime. This aspect of a
 2013 service component is most likely captured during application design. The *propagatesTransaction* or
 2014 *suspendsTransaction* intent can be attached to sca:service elements and their children. The intents are
 2015 defined as follows:

- 2016 • **propagatesTransaction** – A service marked with propagatesTransaction MUST be dispatched under
 2017 any propagated (client) transaction. [POL90015] Use of the *propagatesTransaction* intent on a
 2018 service implies that the service binding MUST be capable of receiving a transaction context. Use of
 2019 the *propagatesTransaction* intent on a service implies that the service binding MUST be capable of
 2020 receiving a transaction context. [POL90016] However, it is important to understand that some
 2021 binding/policySet combinations that provide this intent for a service will need the client to propagate a
 2022 transaction context.

2023 In SCA terms, for a reference wired to such a service, this implies that the reference has to use either
 2024 the *propagatesTransaction* intent or a binding/policySet combination that does propagate a
 2025 transaction. If, on the other hand, the service does not need the client to provide a transaction (even
 2026 though it has the *capability* of joining the client's transaction), then some care is needed in the
 2027 configuration of the service. One approach to consider in this case is to use two distinct bindings on
 2028 the service, one that uses the *propagatesTransaction* intent and one that does not - clients that do
 2029 not propagate a transaction would then wire to the service using the binding without the
 2030 *propagatesTransaction* intent specified.

- 2031 • **suspendsTransaction** – A service marked with suspendsTransaction MUST NOT be dispatched
 2032 under any propagated (client) transaction. [POL90017]

2033 The absence of either interaction intent leads to runtime-specific behavior; the client is unable to
 2034 determine from transaction intents whether its transaction will be joined.

2035 ~~The SCA runtime MUST ignore the propagatesTransaction intent for OneWay methods. The SCA runtime~~
 2036 ~~MUST ignore the propagatesTransaction intent for OneWay methods.~~ [POL90025]

2037 These intents are independent from the implementation’s *managedTransaction* intent and provides no
 2038 information about the implementation’s transaction environment.

2039 The combination of these service interaction policies and the *managedTransaction* implementation
 2040 policy of the containing component completely describes the transactional behavior of an invoked service,
 2041 as summarized in Table 9-2:

2042

service interaction intent	managedTransaction (component implementation intent)	Results
propagatesTransaction	managedTransaction.global	Component runs in propagated transaction if present, otherwise a new global transaction. This combination is used for the managed, shared global transaction pattern described in Common Transaction Patterns. This is equivalent to the managedSharedTransaction intent

		defined in section 9.6.3.
propagatesTransaction	managedTransaction.local or noManagedTransaction	A service MUST NOT be marked with "propagatesTransaction" if the component is marked with "managedTransaction.local" or with "noManagedTransaction" A service MUST NOT 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 managed, local transaction pattern described in Common Transaction Patterns. This is the default behavior for a runtime that does not support global transactions.
suspendsTransaction	noManagedTransaction	Component is responsible for managing its own local transactional resources.

2043 Table 9-2 Combining service transaction intents

2044

2045 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A
2046 runtime that supports global transaction coordination can choose to provide a default behavior that is the
2047 managed, shared global transaction pattern.

2048 9.6.2 Handling Outbound Transaction Context

2049 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can also be attached
2050 to an sca:reference XML element to describe whether any client transaction context is propagated to a
2051 target service when a synchronous interaction occurs through the reference. These intents specify a
2052 contract that has to be implemented by the SCA runtime. This aspect of a service component is most
2053 likely captured during application design.

2054 Either the **propagatesTransaction** or **suspendsTransaction** intent can be attached to sca:service
2055 elements and their children. The intents are defined as defined in Section 9.6.19-6.4.

2056 When used as a reference interaction intent, the meaning of the qualifiers is as follows:

- 2057 • **propagatesTransaction** – **When a reference is marked with propagatesTransaction, any transaction**
2058 **context under which the client runs MUST be propagated when the reference is used for a request-**
2059 **response interaction** ~~When a reference is marked with propagatesTransaction, any transaction context~~
2060 ~~under which the client runs MUST be propagated when the reference is used for a request-response~~
2061 ~~interaction~~ [POL90020] The binding of a reference marked with propagatesTransaction has to be
2062 capable of propagating a transaction context. The reference needs to be wired to a service that can
2063 join the client's transaction. For example, any service with an intent that @requires
2064 **propagatesTransaction** can always join a client's transaction. The reference consumer can then be
2065 designed to rely on the work of the target service being included in the caller's transaction.

- 2066 • **suspendsTransaction** – **When a reference is marked with `suspendsTransaction`, any transaction**
 2067 **context under which the client runs MUST NOT be propagated when the reference is used.**
 2068 **When a reference is marked with `suspendsTransaction`, any transaction context under which the client runs**
 2069 **MUST NOT be propagated when the reference is used.** [POL90022] The reference consumer can
 2070 use this intent to ensure that the work of the target service is not included in the caller's transaction. .

2071 • The absence of either interaction intent leads to runtime-specific behavior. The SCA runtime can
 2072 choose whether or not to propagate any client transaction context to the referenced service,
 2073 depending on the SCA runtime capability.

2074 These intents are independent from the client's ***managedTransaction*** implementation intent. The
 2075 combination of the interaction intent of a reference and the ***managedTransaction*** implementation policy
 2076 of the containing component completely describes the transactional behavior of a client's invocation of a
 2077 service. **Table 9-3** summarizes the results of the combination of either of these interaction
 2078 intents with the ***managedTransaction*** implementation policy of the containing component.

reference interaction intent	managedTransaction (client implementation intent)	Results
propagatesTransaction	managedTransaction.global	Target service runs in the client's transaction. This combination is used for the managed, shared global transaction pattern described in Common Transaction Patterns.
propagatesTransaction	managedTransaction.local or noManagedTransaction	A reference MUST NOT be marked with <code>propagatesTransaction</code> if component is marked with <code>"ManagedTransaction.local"</code> or with <code>"noManagedTransaction"</code> [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 managed, local transaction pattern described in Common Transaction Patterns.

2080 *Table 9-3 Transaction propagation reference intents*

2081
 2082 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A
 2083 runtime that supports global transaction coordination can choose to provide a default behavior that is the
 2084 managed, shared global transaction pattern.

2085 Table 9-4 shows the valid combination of interaction and implementation intents on the client and service
 2086 that result in a single global transaction being used when a client invokes a service through a reference.

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

2088 *Table 9-4 Intents for end-to-end transaction propagation*

2090 | ~~Transaction context MUST NOT be propagated on OneWay messages. Transaction context MUST NOT~~
2091 | ~~be propagated on OneWay messages.~~ [POL90024] The SCA runtime ignores **propagatesTransaction**
2092 | for OneWay operations.

2093 | 9.6.3 Combining implementation and interaction intents

2094 | The **managed, local transaction pattern** can be configured quite easily by combining the
2095 | managedTransaction.global intent with the propagatesTransaction intent. This is illustrated in Error!
2096 | Reference source not found.~~Error! Reference source not found.~~. In order to enable easier
2097 | configuration of this pattern, a profile intent called managedSharedTransaction is defined as in section
2098 | Error! Reference source not found.~~Error! Reference source not found.~~.

2099 | 9.6.4 Web services binding for propagatesTransaction policy

2100 | Snippet 9-1 shows a policySet that provides the **propagatesTransaction** intent and applies to a Web
2101 | service binding (binding.ws). When used on a service, this policySet would require the client to send a
2102 | transaction context using the mechanisms described in the [Web Services Atomic Transaction](#) [WS-
2103 | AtomicTransaction] specification.

2104 |

```
2105 | <policySet name="JoinsTransactionWS" provides="sca:propagatesTransaction"  
2106 |           appliesTo="sca:binding.ws">  
2107 |   <wsp:Policy>  
2108 |     <wsat:ATAssertion  
2109 |       xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"/>  
2110 |   </wsp:Policy>  
2111 | </policySet>
```

2112 | *Snippet 9-1: Example policySet Providing propagatesTransaction*

2113

10 Miscellaneous Intents

2114 The following are standard intents that apply to bindings and are not related to either security, reliable
2115 messaging or transactionality:

- 2116 • **SOAP** – The SOAP intent specifies that the SOAP messaging model is used for delivering messages.
2117 It does not require the use of any specific transport technology for delivering the messages, so for
2118 example, this intent can be supported by a binding that sends SOAP messages over HTTP, bare
2119 TCP or even JMS. If the intent is attached in an unqualified form then any version of SOAP is
2120 acceptable. Standard mutually exclusive qualified intents also exist for SOAP.1_1 and SOAP.1_2,
2121 which specify the use of versions 1.1 or 1.2 of SOAP respectively. **When SOAP is present, an SCA
2122 Runtime MUST use the SOAP messaging model to deliver messages. When SOAP is present, an
2123 SCA Runtime MUST use the SOAP messaging model to deliver messages. [POL100001] When a
2124 SOAP intent is qualified with 1_1 or 1_2, then SOAP version 1.1 or SOAP version 1.2 respectively
2125 MUST be used to deliver messages. When a SOAP intent is qualified with 1_1 or 1_2, then SOAP
2126 version 1.1 or SOAP version 1.2 respectively MUST be used to deliver messages. [POL100002]**
- 2127 • **JMS** – The JMS intent does not specify a wire-level transport protocol, but instead requires that
2128 whatever binding technology is used, the messages are able to be delivered and received via the
2129 JMS API. **When JMS is present, an SCA Runtime MUST ensure that the binding used to send and
2130 receive messages supports the JMS API. When JMS is present, an SCA Runtime MUST ensure that
2131 the binding used to send and receive messages supports the JMS API. [POL100003]**
- 2132 • **noListener** – This intent can only be used within the @requires attribute of a reference. **The
2133 noListener intent MUST only be declared on a @requires attribute of a reference. The noListener
2134 intent MUST only be declared on a @requires attribute of a reference. [POL100004]** It states that the
2135 client is not able to handle new inbound connections. It requires that the binding and callback binding
2136 be configured so that any response (or callback) comes either through a back channel of the
2137 connection from the client to the server or by having the client poll the server for messages. **When
2138 noListener is present, an SCA Runtime MUST not establish any connection from a service to a
2139 client. When noListener is present, an SCA Runtime MUST not establish any connection from a
2140 service to a client. [POL100005]** An example policy assertion that would guarantee this is a WS-
2141 Policy assertion that applies to the <binding.ws> binding, which requires the use of WS-Addressing
2142 with anonymous responses (e.g. <wsaw:Anonymous>required</wsaw:Anonymous>” – see
2143 <http://www.w3.org/TR/ws-addr-wsdl/#anonelement>).
- 2144 • **asyncInvocation** – This intent can be attached to an operation or a complete interface, indicating
2145 that the operation(s) are long-running request-response operation(s) [SCA-Assembly]. It is also
2146 possible for a service to set the asyncInvocation intent when using an interface which is not marked
2147 with the asyncInvocation intent. This can be useful when reusing an existing interface definition that
2148 does not contain SCA information.
- 2149 • **EJB** - The EJB intent specifies that whatever wire-level transport technology is specified the
2150 messages are able to be delivered and received via the EJB API. **When EJB is present, an SCA
2151 Runtime MUST ensure that the binding used to send and receive messages supports the EJB
2152 API. When EJB is present, an SCA Runtime MUST ensure that the binding used to send and receive
2153 messages supports the EJB API. [POL100006]**

2154 The formal definitions of these intents are in the [Intent Definitions appendix](#).

2155 11 Conformance

2156 The XML schema available at the namespace URI, defined by this specification, is considered to be
2157 authoritative and takes precedence over the XML Schema defined in the appendix of this document.

2158 ~~An SCA runtime MUST reject a composite file that does not conform to the sca-policy-1.1.xsd schema.~~
2159 ~~An SCA runtime MUST reject a composite file that does not conform to the sca-policy-1.1.xsd schema.~~
2160 [POL110001]

2161 An implementation that claims to conform to this specification MUST meet the following conditions:

- 2162 1. The implementation MUST conform to the SCA Assembly Model Specification [Assembly].
- 2163 2. SCA implementations MUST recognize the intents listed in Appendix B.1 of this specification. An
2164 implementationType / bindingType / collection of policySets that claims to implement a specific intent
2165 MUST process that intent in accord with any relevant Conformance Items in Appendix C related to
2166 the intent and the SCA Runtime options selected.
- 2167 3. With the exception of 2, the implementation MUST comply with all statements in [Appendix C](#):
2168 Conformance Items related to an SCA Runtime, notably all MUST statements have to be
2169 implemented.

A Defining the Deployed Composites Infoset

2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211

The @attachTo attribute of an intent or a policySet is an XPath1.0 expression identifying a SCA element to which the intent or the policySet is attached. The XPath applies to the **Deployed Composites Infoset**

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

4. The Domain is treated as a special composite, with a blank name - ""
5. 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

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.

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

6. For direct attachment (using a @requires or @policySet attribute on an element or a <requires> or <policySetAttachment/> subelement)
7. For policySets by reference from another policySet element

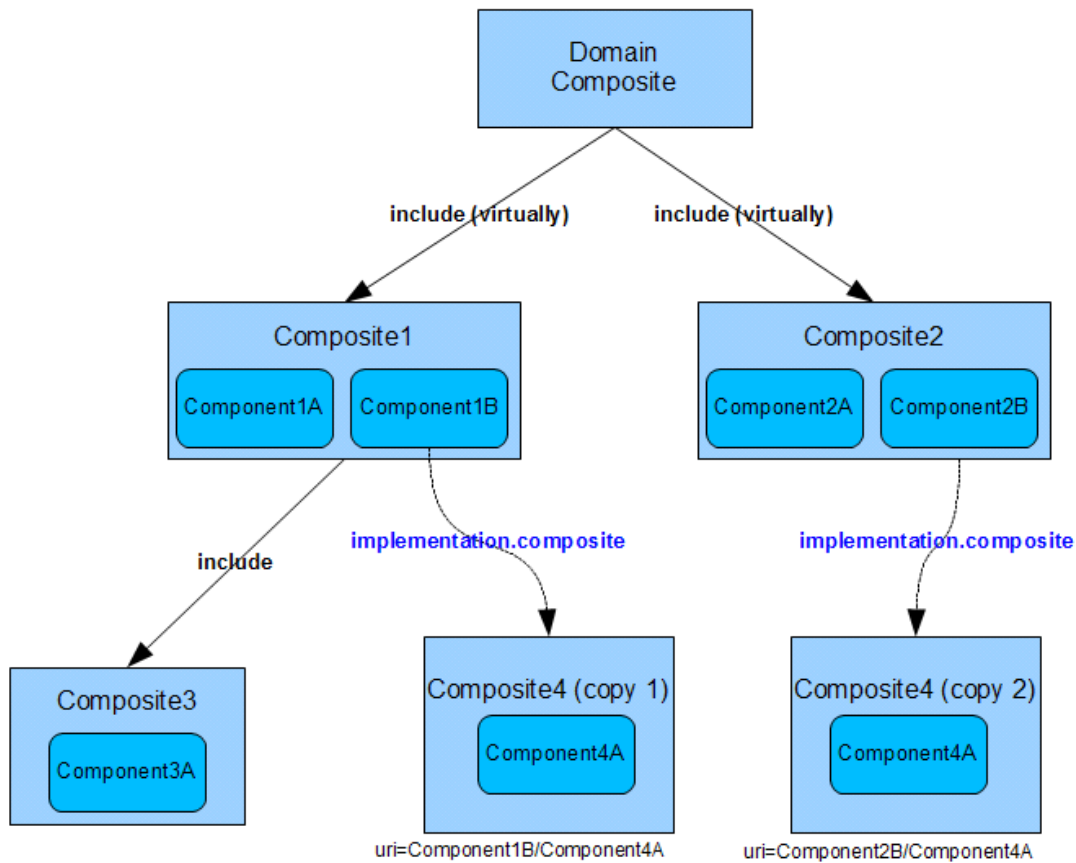
The SCA runtime MUST raise an error if the @attachTo XPath expression resolves to an SCA <property> element, or any of its children. [POL40002]

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:



2212

2213 *Figure A-1 Example Domain Composite Infoset*

2214

2215 The SCA Domain in re A-1 has been constructed from the composites and components shown in the
 2216 figure. Composite1 and Composite2 were deployed into the Domain as described in [SCA-Assembly].
 2217 Composite3 is included in Composite1 using the SCA include mechanism described in [SCA-Assembly].
 2218 Composite4 is used as an implementation of Components 1B and 2B. Following the deployment of all the
 2219 composites, the Domain contains:

- 2220 • 3 Composites that can be addressed as part of the Deployed Composites InfoSet; Composite1,
 2221 Composite2 and Composite4.
- 2222 • all the components shown in the diagram. Components 1A, 2A, 3A, 4A (twice) are leaf
 2223 components.

2224

2225 The following snippets show example usage of the @attachTo attribute and provide the outcome based
 2226 on the Domain in A-1.

2227

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

2229 *Snippet A-1:Example attachTo all Instances of a Name*

2230

2231 attach to both instances of Component4A

2232

2233 `2. //component[URIRef("Component2B/Component4A")]`

2234 *Snippet A-2: Example attachTo a Specific Instance via a Path*

2235
2236 attach to the unique instance of Component4A when used by Component2B (Component2B is a
2237 component at the Domain level)

2238
2239 `3. //component[@name="Component3A"]/service[IntentRefs("intent1")]`

2240 *Snippet A-3: Example attachTo Instances with an intent*

2241
2242 attach to the services of Component3A which have the intent "intent1" applied

2243
2244 `4. //component/binding.ws`

2245 *Snippet A-4: Example attachTo Instances with a binding*

2246
2247 attach to the web services binding of all components with a service or reference with a Web services
2248 binding

2249
2250 `5. /composite[@name=""]/component[@name="Component1A"]`

2251 *Snippet A-5: Example attachTo a Specific Instance via Path and Name*

2252
2253 attach to Component1A at the Domain level

2254

2255

AB Schemas

2256

A.1B.1 sca-policy.xsd

```
2257 <?xml version="1.0" encoding="UTF-8"?>
2258 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2259 OASIS trademark, IPR and other policies apply. -->
2260 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2261 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2262 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2263 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
2264 elementFormDefault="qualified">
2265
2266 <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
2267 <import namespace="http://www.w3.org/ns/ws-policy"
2268 schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd"/>
2269
2270 <element name="intent" type="sca:Intent"/>
2271 <complexType name="Intent">
2272 <sequence>
2273 <element name="description" type="string" minOccurs="0"
2274 maxOccurs="1" />
2275 <element name="qualifier" type="sca:IntentQualifier"
2276 minOccurs="0" maxOccurs="unbounded" />
2277 <any namespace="##other" processContents="lax"
2278 minOccurs="0" maxOccurs="unbounded"/>
2279 </sequence>
2280 <attribute name="name" type="NCName" use="required"/>
2281 <attribute name="constrains" type="sca:listOfQNames"
2282 use="optional"/>
2283 <attribute name="requires" type="sca:listOfQNames"
2284 use="optional"/>
2285 <attribute name="excludes" type="sca:listOfQNames"
2286 use="optional"/>
2287 <attribute name="mutuallyExclusive" type="boolean"
2288 use="optional" default="false"/>
2289 <attribute name="intentType"
2290 type="sca:InteractionOrImplementation"
2291 use="optional" default="interaction"/>
2292 <anyAttribute namespace="##other" processContents="lax"/>
2293 </complexType>
2294
2295 <complexType name="IntentQualifier">
2296 <sequence>
2297 <element name="description" type="string" minOccurs="0"
2298 maxOccurs="1" />
2299 </sequence>
2300 <attribute name="name" type="NCName" use="required"/>
2301 <attribute name="default" type="boolean" use="optional"
2302 default="false"/>
2303 </complexType>
2304
2305 <element name="requires">
2306 <complexType>
2307 <sequence minOccurs="0" maxOccurs="unbounded">
2308 <any namespace="##other" processContents="lax"/>
2309 </sequence>
2310 <attribute name="intents" type="sca:listOfQNames"
2311 use="required"/>
```



```

2312         <anyAttribute namespace="##other" processContents="lax"/>
2313     </complexType>
2314 </element>
2315
2316 <element name="policySet" type="sca:PolicySet"/>
2317 <complexType name="PolicySet">
2318     <choice minOccurs="0" maxOccurs="unbounded">
2319         <element name="policySetReference"
2320             type="sca:PolicySetReference"/>
2321         <element name="intentMap" type="sca:IntentMap"/>
2322         <any namespace="##other" processContents="lax"/>
2323     </choice>
2324     <attribute name="name" type="NCName" use="required"/>
2325     <attribute name="provides" type="sca:listOfQNames"/>
2326     <attribute name="appliesTo" type="string" use="optional"/>
2327     <attribute name="attachTo" type="string" use="optional"/>
2328     <anyAttribute namespace="##other" processContents="lax"/>
2329 </complexType>
2330
2331 <element name="policySetAttachment">
2332     <complexType>
2333         <sequence minOccurs="0" maxOccurs="unbounded">
2334             <any namespace="##other" processContents="lax"/>
2335         </sequence>
2336         <attribute name="name" type="QName" use="required"/>
2337         <anyAttribute namespace="##other" processContents="lax"/>
2338     </complexType>
2339 </element>
2340
2341 <complexType name="PolicySetReference">
2342     <attribute name="name" type="QName" use="required"/>
2343     <anyAttribute namespace="##other" processContents="lax"/>
2344 </complexType>
2345
2346 <complexType name="IntentMap">
2347     <choice minOccurs="1" maxOccurs="unbounded">
2348         <element name="qualifier" type="sca:Qualifier"/>
2349         <any namespace="##other" processContents="lax"/>
2350     </choice>
2351     <attribute name="provides" type="QName" use="required"/>
2352     <anyAttribute namespace="##other" processContents="lax"/>
2353 </complexType>
2354
2355 <complexType name="Qualifier">
2356     <sequence minOccurs="0" maxOccurs="unbounded">
2357         <any namespace="##other" processContents="lax"/>
2358     </sequence>
2359     <attribute name="name" type="string" use="required"/>
2360     <anyAttribute namespace="##other" processContents="lax"/>
2361 </complexType>
2362
2363 <simpleType name="listOfNCNames">
2364     <list itemType="NCName"/>
2365 </simpleType>
2366
2367 <simpleType name="InteractionOrImplementation">
2368     <restriction base="string">
2369         <enumeration value="interaction"/>
2370         <enumeration value="implementation"/>
2371     </restriction>
2372 </simpleType>
2373
2374 </schema>

```


2376

BC XML Files

2377

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

2378

B.1C.1 Intent Definitions

2379

Intent definitions are contained within a Definitions file called Policy_Intents_Definitions.xml, which contain a <definitions/> element as follows:

2380

2381

```
<?xml version="1.0" encoding="UTF-8"?>
2382 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2383 OASIS trademark, IPR and other policies apply. -->
2384 <sca:definitions xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2385 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2386 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903">
2387
2388 <!-- Security related intents -->
2389 <sca:intent name="serverAuthentication" constrains="sca:binding"
2390 intentType="interaction">
2391 <sca:description>
2392 Communication through the binding requires that the
2393 server is authenticated by the client
2394 </sca:description>
2395 <sca:qualifier name="transport" default="true"/>
2396 <sca:qualifier name="message"/>
2397 </sca:intent>
2398
2399 <sca:intent name="clientAuthentication" constrains="sca:binding"
2400 intentType="interaction">
2401 <sca:description>
2402 Communication through the binding requires that the
2403 client is authenticated by the server
2404 </sca:description>
2405 <sca:qualifier name="transport" default="true"/>
2406 <sca:qualifier name="message"/>
2407 </sca:intent>
2408
2409 <sca:intent name="authentication"
2410 requires="sca:clientAuthentication">
2411 <sca:description>
2412 A convenience intent to help migration
2413 </sca:description>
2414 </sca:intent>
2415
2416 <sca:intent name="mutualAuthentication"
2417 requires="sca:clientAuthentication sca:serverAuthentication">
2418 <sca:description>
2419 Communication through the binding requires that the
2420 client and server to authenticate each other
2421 </sca:description>
2422 </sca:intent>
2423
2424 <sca:intent name="confidentiality" constrains="sca:binding"
2425 intentType="interaction">
2426 <sca:description>
2427 Communication through the binding prevents unauthorized
2428 users from reading the messages
2429 </sca:description>
2430 <sca:qualifier name="transport" default="true"/>
2431 <sca:qualifier name="message"/>
```

2431

```

2432     </sca:intent>
2433
2434     <sca:intent name="integrity" constrains="sca:binding"
2435 intentType="interaction">
2436         <sca:description>
2437             Communication through the binding prevents tampering
2438             with the messages sent between the client and the service.
2439         </sca:description>
2440         <sca:qualifier name="transport" default="true"/>
2441         <sca:qualifier name="message"/>
2442     </sca:intent>
2443
2444     <sca:intent name="authorization" constrains="sca:implementation"
2445 intentType="implementation">
2446         <sca:description>
2447             Ensures clients are authorized to use services.
2448         </sca:description>
2449     </sca:intent>
2450
2451 <!-- Reliable messaging related intents -->
2452
2453     <sca:intent name="atLeastOnce" constrains="sca:binding"
2454 intentType="interaction">
2455         <sca:description>
2456             This intent is used to indicate that a message sent
2457             by a client is always delivered to the component.
2458         </sca:description>
2459     </sca:intent>
2460
2461     <sca:intent name="atMostOnce" constrains="sca:binding"
2462 intentType="interaction">
2463         <sca:description>
2464             This intent is used to indicate that a message that was
2465             successfully sent by a client is not delivered more than
2466             once to the component.
2467         </sca:description>
2468     </sca:intent>
2469
2470     <sca:intent name="exactlyOnce" requires="sca:atLeastOnce
2471 sca:atMostOnce"
2472 constrains="sca:binding" intentType="interaction">
2473         <sca:description>
2474             This profile intent is used to indicate that a message sent
2475             by a client is always delivered to the component. It also
2476             indicates that duplicate messages are not delivered to the
2477             component.
2478         </sca:description>
2479     </sca:intent>
2480
2481     <sca:intent name="ordered" constrains="sca:binding"
2482 intentType="interaction">
2483         <sca:description>
2484             This intent is used to indicate that all the messages are
2485             delivered to the component in the order they were sent by
2486             the client.
2487         </sca:description>
2488     </sca:intent>
2489
2490 <!-- Transaction related intents -->
2491     <sca:intent name="managedTransaction"
2492         excludes="sca:noManagedTransaction"
2493         mutuallyExclusive="true" constrains="sca:implementation"
2494     intentType="implementation">

```

```

2495         <sca:description>
2496         A managed transaction environment is necessary in order to
2497         run the component. The specific type of managed transaction
2498         needed is not constrained.
2499         </sca:description>
2500         <sca:qualifier name="global" default="true">
2501             <sca:description>
2502             For a component marked with managedTransaction.global
2503             a global transaction needs to be present before dispatching
2504             any method on the component - using any transaction
2505             propagated from the client or else beginning and completing
2506             a new transaction.
2507             </sca:description>
2508         </sca:qualifier>
2509         <sca:qualifier name="local">
2510             <sca:description>
2511             A component marked with managedTransaction.local needs to
2512             run within a local transaction containment (LTC) that
2513             is started and ended by the SCA runtime.
2514             </sca:description>
2515         </sca:qualifier>
2516     </sca:intent>
2517
2518     <sca:intent name="noManagedTransaction"
2519     excludes="sca:managedTransaction"
2520     constrains="sca:implementation" intentType="implementation">
2521         <sca:description>
2522         A component marked with noManagedTransaction needs to run without
2523         a managed transaction, under neither a global transaction nor
2524         an LTC. A transaction propagated to the hosting SCA runtime
2525         is not joined by the hosting runtime on behalf of a
2526         component marked with noManagedtransaction.
2527         </sca:description>
2528     </sca:intent>
2529
2530     <sca:intent name="transactedOneWay" excludes="sca:immediateOneWay"
2531     constrains="sca:binding" intentType="implementation">
2532         <sca:description>
2533         For a reference marked as transactedOneWay any OneWay invocation
2534         messages are transacted as part of a client global
2535         transaction.
2536         For a service marked as transactedOneWay any OneWay invocation
2537         message are received from the transport binding in a
2538         transacted fashion, under the service's global transaction.
2539         </sca:description>
2540     </sca:intent>
2541
2542     <sca:intent name="immediateOneWay" excludes="sca:transactedOneWay"
2543     constrains="sca:binding" intentType="implementation">
2544         <sca:description>
2545         For a reference indicates that any OneWay invocation messages
2546         are sent immediately regardless of any client transaction.
2547         For a service indicates that any OneWay invocation is
2548         received immediately regardless of any target service
2549         transaction.
2550         </sca:description>
2551     </sca:intent>
2552
2553     <sca:intent name="propagatesTransaction"
2554     excludes="sca:suspendsTransaction"
2555     constrains="sca:binding" intentType="interaction">
2556         <sca:description>
2557         A service marked with propagatesTransaction is dispatched

```

```

2558         under any propagated (client) transaction and the service binding
2559         needs to be capable of receiving a transaction context.
2560         A reference marked with propagatesTransaction propagates any
2561         transaction context under which the client runs when the
2562         reference is used for a request-response interaction and the
2563         binding of a reference marked with propagatesTransaction needs to
2564         be capable of propagating a transaction context.
2565         </sca:description>
2566     </sca:intent>
2567
2568     <sca:intent name="suspendsTransaction"
2569         excludes="sca:propagatesTransaction"
2570     constrains="sca:binding" intentType="interaction">
2571         <sca:description>
2572             A service marked with suspendsTransaction is not dispatched
2573             under any propagated (client) transaction.
2574             A reference marked with suspendsTransaction does not propagate
2575             any transaction context under which the client runs when the
2576             reference is used.
2577         </sca:description>
2578     </sca:intent>
2579
2580     <sca:intent name="managedSharedTransaction"
2581         requires="sca:managedTransaction.global
2582     sca:propagatesTransaction">
2583         <sca:description>
2584             Used to indicate that the component requires both the
2585             managedTransaction.global and the propagatesTransactions
2586             intents
2587         </sca:description>
2588     </sca:intent>
2589
2590     <!-- Miscellaneous intents -->
2591     <sca:intent name="asyncInvocation" constrains="sca:binding"
2592         intentType="interaction">
2593         <sca:description>
2594             Indicates that request/response operations for the
2595             interface of this wire are "long running" and must be
2596             treated as two separate message transmissions
2597         </sca:description>
2598     </sca:intent>
2599
2600     <sca:intent name="EJB" constrains="sca:binding"
2601         intentType="interaction">
2602         <sca:description>
2603             Specifies that the EJB API is needed to communicate with
2604             the service or reference.
2605         </sca:description>
2606     </sca:intent>
2607
2608     <sca:intent name="SOAP" constrains="sca:binding"
2609         intentType="interaction" mutuallyExclusive="true">
2610         <sca:description>
2611             Specifies that the SOAP messaging model is used for delivering
2612             messages.
2613         </sca:description>
2614         <sca:qualifier name="v1_1" default="true"/>
2615         <sca:qualifier name="v1_2"/>
2616     </sca:intent>
2617
2618     <sca:intent name="JMS" constrains="sca:binding"
2619         intentType="interaction">
2620         <sca:description>

```

```
2621     Requires that the messages are delivered and received via the
2622     JMS API.
2623     </sca:description>
2624 </sca:intent>
2625
2626     <sca:intent name="noListener" constrains="sca:binding"
2627 intentType="interaction">
2628     <sca:description>
2629     This intent can only be used on a reference. Indicates that the
2630     client is not able to handle new inbound connections. The binding
2631     and callback binding are configured so that any
2632     response or callback comes either through a back channel of the
2633     connection from the client to the server or by having the client
2634     poll the server for messages.
2635     </sca:description>
2636 </sca:intent>
2637
2638 </sca:definitions>
```

2639 *Snippet C-1: SCA intent Definitions*

2640

GD Conformance

2641

C.1D.1 Conformance Targets

2642

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

2643

- Document artifacts (or constructs within them) that can be checked statically.

2644

- SCA runtimes, which we may require to exhibit certain behaviors.

2645

C.2D.2 Conformance Items

2646

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

2647

Conformance ID

Description

~~[POL30001]~~~~[POL30004]~~

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]~~~~[POL30002]~~

The QName for an intent MUST be unique amongst the set of intents in the SCA Domain.

~~[POL30004]~~~~[POL30004]~~

If an intent has more than one qualifier, one and only one MUST be declared as the default qualifier.

~~[POL30005]~~~~[POL30005]~~

The name of each qualifier MUST be unique within the intent definition.

~~[POL30006]~~~~[POL30006]~~

the name of a profile intent MUST NOT have a "." in it.

~~[POL30007]~~~~[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 4.144.12.

~~[POL30008]~~~~[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]~~~~[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]~~~~[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]~~~~[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] [POL30015]	Each QName in the @requires attribute MUST be the QName of an intent in the SCA Domain.
[POL30016] [POL30016]	Each QName in the @excludes attribute MUST be the QName of an intent in the SCA Domain.
[POL30017] [POL30017]	The QName for a policySet MUST be unique amongst the set of policySets in the SCA Domain.
[POL30018] [POL30018]	The contents of @appliesTo MUST match the XPath 1.0 [XPATH] production <i>Expr</i> .
[POL30019] [POL30019]	The contents of @attachTo MUST match the XPath 1.0 production <i>Expr</i> .
[POL30020] [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] [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] [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] [POL30025]	If only one qualifier for an intent is given it MUST be used as the default qualifier for the intent.
[POL40001] [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
[POL40002]	The SCA runtime MUST raise an error if the @attachTo XPath expression resolves to an SCA <property> element, or any of its children.
[POL40004] [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] [POL40005]	Rule2: The intents declared on elements higher in the structural hierarchy of a given element MUST be applied to the element EXCEPT <ul style="list-style-type: none"> 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] [POL40006]	If a component has any policySets attached to it (by any means), then any policySets attached to the componentType MUST be

ignored.

[POL40007][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][POL40009]

Any two intents applied to a given element MUST NOT be mutually exclusive

[POL40010][POL40010]

SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms for policySet attachment.

[POL40011][POL40011]

SCA implementations supporting only the External Attachment mechanism MUST ignore the policy sets that are applicable via the Direct Attachment mechanism.

[POL40012][POL40012]

SCA implementations supporting only the Direct Attachment mechanism MUST ignore the policy sets that are applicable via the External Attachment mechanism.

[POL40013][POL40013]

During the deployment of SCA composites, all policySets within the Domain with an attachTo attribute MUST be evaluated to determine which policySets are attached to the newly deployed composite.

[POL40014][POL40014]

The intents declared on elements lower in the implementation hierarchy of a given element MUST be applied to the element.

[POL40015][POL40015]

when combining implementation hierarchy and structural hierarchy policy data, Rule 1 MUST be applied BEFORE Rule 2.

[POL40016][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][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]

[POL40018]

All intents in the required intent set for an element SHOULD be provided by the directly provided intents set and the set of policySets that apply to the element.

[POL40019][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][POL40020]

The QName of the bindingType MUST be unique amongst the set of bindingTypes in the SCA Domain.

[POL40021][POL40021]

A binding implementation MUST implement all the intents listed in the @alwaysProvides and @mayProvides attributes.

[POL40022][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] [POL40023]	The policySets at each end of a wire MUST be incompatible if they use different policy languages.
[POL40024] [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] [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.
[POL40026]	During the deployment of an SCA policySet, the behavior of an SCA runtime MUST take ONE of the following forms:
<u>When External Attachment is used for both intents and policySets, intents must be attached before policySets</u> [POL40xxx]	<ul style="list-style-type: none"> • The policySet is immediately attached to all deployed composites which satisfy the @attachTo attribute of the policySet. • The policySet is attached to a deployed composite which satisfies the @attachTo attribute of the policySet when the composite is re-deployed.
[POL40026]	
[POL40027] [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.
[POL50001] [POL50001]	The implementationType name attribute MUST be the QName of an XSD global element definition used for implementation elements of that type.
[POL70001] [POL70001]	When <i>authorization</i> is present, an SCA Runtime MUST ensure that the client is authorized to use the service.
[POL70009] [POL70009]	When <i>confidentiality</i> is present, an SCA Runtime MUST ensure that only authorized entities can view the contents of a message.
[POL70010] [POL70010]	When <i>integrity</i> is present, an SCA Runtime MUST ensure that the contents of a message are not altered.
[POL70011] [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] [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] [POL70013]	When <i>serverAuthentication</i> is present, an SCA runtime MUST ensure that the server is authenticated by the client.

[POL70014] [POL70014]	When <i>clientAuthentication</i> is present, an SCA runtime MUST ensure that the client is authenticated by the server.
[POL80001] [POL80001]	When <i>atLeastOnce</i> 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] [POL80002]	When <i>atMostOnce</i> 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] [POL80003]	When <i>ordered</i> 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] [POL80004]	When <i>exactlyOnce</i> 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] [POL90003]	For a component marked with <i>managedTransaction.global</i> , the SCA runtime MUST ensure that a global transaction is present before dispatching any method on the component.
[POL90004] [POL90004]	A component marked with <i>managedTransaction.local</i> MUST run within a local transaction containment (LTC) that is started and ended by the SCA runtime.
[POL90006] [POL90006]	Local transactions MUST NOT be propagated outbound across remotable interfaces.
[POL90007] [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 <i>noManagedtransaction</i> .
[POL90008] [POL90008]	When a reference is marked as <i>transactedOneWay</i> , any <i>OneWay</i> invocation messages MUST be transacted as part of a client global transaction.
[POL90009] [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 <i>transactedOneWay</i> .
[POL90010] [POL90010]	If a service is marked as <i>transactedOneWay</i> , any <i>OneWay</i> invocation message MUST be received from the transport binding in a transacted fashion, under the target service's global transaction.
[POL90011] [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 <i>transactedOneWay</i> .
[POL90012] [POL90012]	When applied to a reference indicates that any <i>OneWay</i> invocation messages MUST be sent immediately regardless of any client transaction.

[POL90013][POL90013]	When applied to a service indicates that any OneWay invocation MUST be received immediately regardless of any target service transaction.
[POL90015][POL90015]	A service marked with propagatesTransaction MUST be dispatched under any propagated (client) transaction.
[POL90016][POL90016]	Use of the propagatesTransaction intent on a service implies that the service binding MUST be capable of receiving a transaction context.
[POL90017][POL90017]	A service marked with suspendsTransaction MUST NOT be dispatched under any propagated (client) transaction.
[POL90019] [POL90019]	A service MUST NOT be marked with "propagatesTransaction" if the component is marked with "managedTransaction.local" or with "noManagedTransaction"
[POL90020][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][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][POL90023]	A reference MUST NOT be marked with propagatesTransaction if component is marked with "ManagedTransaction.local" or with "noManagedTransaction"
[POL90024][POL90024]	Transaction context MUST NOT be propagated on OneWay messages.
[POL90025][POL90025]	The SCA runtime MUST ignore the propagatesTransaction intent for OneWay methods.
[POL90027][POL90027]	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.
[POL100001][POL100001]	When SOAP is present, an SCA Runtime MUST use the SOAP messaging model to deliver messages.
[POL100002][POL100002]	When a SOAP 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.
[POL100003][POL100003]	When JMS is present, an SCA Runtime MUST ensure that the binding used to send and receive messages supports the JMS API.
[POL100004][POL100004]	The noListener intent MUST only be declared on a @requires attribute of a reference.
[POL100005][POL100005]	When noListener is present, an SCA Runtime MUST not establish any connection from a service to a client.
[POL100006][POL100006]	When EJB is present, an SCA Runtime MUST ensure that the binding used to send and receive messages supports the EJB API.
[POL110001][POL110001]	An SCA runtime MUST reject a composite file that does not conform to the sca-policy-1.1.xsd schema.

2648 Table D-1: SCA Policy Normative Statements

2649

DE Acknowledgements

2650

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

2651

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
Nickolaos 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 Raeppe	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*

2652

2653
2654
2655

EF 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

2656

2657