



SCA Policy Framework Version 1.1

**Committee Draft 02/Public Review 01 – rev6-
issue93Rev4**

February 249, 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.....	9
1.1	Terminology.....	9
1.2	XML Namespaces.....	9
1.3	Normative References.....	9
1.4	Naming Conventions.....	10
2	Overview.....	11
2.1	Policies and PolicySets.....	11
2.2	Intents describe the requirements of Components, Services and References.....	11
2.3	Determining which policies apply to a particular wire.....	12
3	Framework Model.....	13
3.1	Intents.....	13
3.2	Interaction Intents and Implementation Intents.....	15
3.3	Profile Intents.....	16
3.4	PolicySets.....	16
3.4.1	IntentMaps.....	18
3.4.2	Direct Inclusion of Policies within PolicySets.....	20
3.4.3	Policy Set References.....	20
4	Attaching Intents and PolicySets to SCA Constructs.....	23
4.1	Attachment Rules – Intents.....	23
4.2	Direct Attachment of Intents.....	23
4.3	External Attachment of Intents.....	24
4.4	Attachment Rules - PolicySets.....	24
4.5	Direct Attachment of PolicySets.....	25
4.6	External Attachment of PolicySets.....	26
4.6.1	Cases Where Multiple PolicySets are attached to a Single Artifact.....	28
4.7	Attaching Intents to SCA Elements.....	31
4.7.1	Implementation Hierarchy of an Element.....	31
4.7.2	Structural Hierarchy of an Element.....	31
4.7.3	Combining Implementation and Structural Policy Data.....	32
4.7.4	Examples.....	32
4.8	Usage of Intent and Policy Set Attachment together.....	34
4.9	Intents and PolicySets on Implementations and Component Types.....	34
4.10	Intents on Interfaces.....	35
4.11	BindingTypes and Related Intents.....	35
4.12	Treatment of Components with Internal Wiring.....	36
4.12.1	Determining Wire Validity and Configuration.....	37
4.13	Preparing Services and References for External Connection.....	37
4.14	Deployment.....	37
4.14.1	Matching Intents and PolicySets.....	39
5	Implementation Policies.....	41

5.1	Natively Supported Intents.....	42
5.2	Writing PolicySets for Implementation Policies	42
5.2.1	Non WS-Policy Examples	43
6	Roles and Responsibilities	44
6.1	Policy Administrator	44
6.2	Developer.....	44
6.3	Assembler.....	44
6.4	Deployer.....	45
7	Security Policy	46
7.1	Security Policy Intents	46
7.2	Interaction Security Policy	46
7.2.1	Qualifiers	47
7.3	Implementation Security Policy Intent	47
8	Reliability Policy.....	48
8.1	Reliability Policy Intents	48
8.2	End-to-end Reliable Messaging	50
9	Transactions	51
9.1	Out of Scope	51
9.2	Common Transaction Patterns	51
9.3	Summary of SCA Transaction Policies.....	52
9.4	Global and local transactions.....	52
9.4.1	Global transactions.....	52
9.4.2	Local transactions	52
9.5	Transaction implementation policy	53
9.5.1	Managed and non-managed transactions.....	53
9.5.2	OneWay Invocations	54
9.6	Transaction interaction policies	55
9.6.1	Handling Inbound Transaction Context.....	55
9.6.2	Handling Outbound Transaction Context.....	57
9.6.3	Combining implementation and interaction intents	58
9.6.4	Web services binding for propagatesTransaction policy.....	58
10	Miscellaneous Intents	59
11	Conformance	60
A	Defining the Deployed Composites Infoset	61
A.1	XPath Functions for the @attachTo Attribute	64
A.1.1	Interface Related Functions	64
A.1.2	Intent Based Functions.....	65
A.1.3	URI Based Function	66
B	Schemas.....	67
B.1	sca-policy.xsd	67
C	XML Files.....	70
C.1	Intent Definitions	70
D	Conformance	75
D.1	Conformance Targets	75

D.2	Conformance Items	75
E	Acknowledgements	81
F	Revision History.....	82
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	21
4.3	Direct Attachment of PolicySets	21
4.4	External Attachment of PolicySets Mechanism.....	23
4.4.1	The Form of the @attachTo Attribute.....	23
4.4.2	Cases Where Multiple PolicySets are attached to a Single Artifact.....	25
4.4.3	XPath Functions for the @attachTo Attribute.....	25
4.4.3.1	Interface Related Functions	26
4.4.3.2	Intent Based Functions	27
4.4.3.3	URI Based Function.....	27
4.5	Usage of @requires attribute for specifying intents.....	28
4.5.1	Implementation Hierarchy of an Element	28
4.5.2	Structural Hierarchy of an Element	28
4.5.3	Combining Implementation and Structural Policy Data	29
4.5.4	Examples.....	29
4.6	Usage of Intent and Policy Set Attachment together.....	31
4.7	Intents and PolicySets on Implementations and Component Types.....	31
4.8	Intents on Interfaces	31
4.9	BindingTypes and Related Intents.....	32
4.10	Treatment of Components with Internal Wiring	33
4.10.1	Determining Wire Validity and Configuration	34
4.11	Preparing Services and References for External Connection.....	34
4.12	Guided Selection of PolicySets using Intents.....	34

4.12.1	Matching Intents and PolicySets	35
5	Implementation Policies	37
5.1	Natively Supported Intents	38
5.2	Writing PolicySets for Implementation Policies	38
5.2.1	Non-WS Policy Examples	38
6	Roles and Responsibilities	40
6.1	Policy Administrator	40
6.2	Developer	40
6.3	Assembler	40
6.4	Deployer	41
7	Security Policy	42
7.1	SCA Security Intents	42
7.2	Interaction Security Policy	42
7.2.1	Qualifiers	43
7.3	Implementation Security Policy Intent	43
8	Reliability Policy	44
8.1	Policy Intents	44
8.2	End-to-end Reliable Messaging	46
9	Transactions	47
9.1	Out of Scope	47
9.2	Common Transaction Patterns	47
9.3	Summary of SCA transaction policies	48
9.4	Global and local transactions	48
9.4.1	Global transactions	48
9.4.2	Local transactions	48
9.5	Transaction implementation policy	49
9.5.1	Managed and non-managed transactions	49
9.5.2	OneWay Invocations	50
9.6	Transaction interaction policies	51
9.6.1	Handling Inbound Transaction Context	51
9.6.2	Handling Outbound Transaction Context	53
9.6.3	Combining implementation and interaction intents	54
9.6.4	Web services binding for propagates Transaction policy	54
10	Miscellaneous Intents	55
11	Conformance	56
A	Schemas	57
A.1	sca-policy.xsd	57
B	XML Files	59
B.1	Intent Definitions	59
C	Conformance	64
C.1	Conformance Targets	64
C.2	Conformance Items	64
D	Acknowledgements	70
E	Revision History	71

1
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.

151 3 Framework Model

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.

160 3.1 Intents

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. [POL30001]. For example, a web service binding which
183 requires the SOAP intent but which points to a WSDL binding that does not specify SOAP.

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

187

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

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

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

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

202

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

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

215

216 Where the intent element has the following attributes:

- 217 • @name (1..1) - an NCName that defines the name of the intent. **The QName for an intent MUST be**
218 **unique amongst the set of intents in the SCA Domain.** [POL30002]
- 219 • @constrains (0..1) - a list of QNames that specifies the SCA constructs that this intent is meant to
220 configure. If a value is not specified for this attribute then the intent can apply to any SCA element.
221 • —Note that the “constrains” attribute can name an abstract element type, such as sca:binding in our
222 running example. This means that it will match against any binding used within an SCA composite
223 file. An SCA element can match @constrains if its type is in a substitution group.
- 224
- 225 • @requires (0..1) - contains a list of QNames of intents which defines the set of all intents that the
226 referring intent requires. In essence, the referring intent requires all the intents named to be satisfied.
227 This attribute is used to compose an intent from a set of other intents. **Each QName in the @requires**
228 **attribute MUST be the QName of an intent in the SCA Domain.** [POL30015] This use is further
229 described in [Section 3.3](#).
- 230 • @excludes (0..1) - a list of QNames of intents that cannot be used with this intent. Intents might
231 describe a policy that is incompatible or otherwise unrealizable when specified with other intents, and
232 therefore are considered to be mutually exclusive. **Each QName in the @excludes attribute MUST be**
233 **the QName of an intent in the SCA Domain.** [POL30016]

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

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

237 Where one intent is attached to an element of an SCA composite and another intent is attached to
238 one of the element’s parents, the intent(s) that are effectively attached to the element differs
239 depending on whether the two intents are mutually exclusive (see @excludes above and section 4.5
240 Usage of @requires attribute for specifying intents).

- 241 • @mutuallyExclusive (0..1) - a boolean with a default of “false”. If this attribute is present and has a
242 value of “true” it indicates that the qualified intents defined for this intent are mutually exclusive.
- 243 • @intentType attribute (0..1) defines whether the intent is an interaction intent or an implementation
244 intent. A value of "interaction", which is the default value, indicates that the intent is an interaction
245 intent. A value of "implementation" indicates that the intent is an implementation intent.

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

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

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

257

```
258 <intent name="confidentiality" constrains="sca:binding">  
259   <description>  
260     Communication through this binding must prevent  
261     unauthorized users from reading the messages.  
262   </description>  
263   <qualifier name="transport">  
264     <description>Automatic encryption by transport  
265     </description>  
266   </qualifier>  
267   <qualifier name="message" default='true'>  
268     <description>Encryption applied to each message  
269     </description>  
270   </qualifier>  
271 </intent>
```

272 *Snippet 3-2: Example intent Definition*

273

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

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

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

288 3.2 Interaction Intents and Implementation Intents

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

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

294 An implementation intent is an intent designed to influence policy which applies to an implementation
295 artifact or to the relationship of that artifact to the runtime code which is used to execute the artifact.

296 Implementation intents do not affect wire matching between references and services, nor do they affect
297 the bytes that flow between a reference and a service.
298 Implementation intents often apply to <implementation/> elements, but they can also apply to <binding/>
299 elements, where the desire is to influence the activity of the binding implementation code and how it
300 interacts with the remainder of the runtime code for the implementation.
301 Interaction intents and implementation intents are distinguished by the value of the @intentType attribute
302 in the intent definition.

303 3.3 Profile Intents

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

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

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

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

```
316 <intent name="messageProtection"  
317     constrains="sca:binding"  
318     requires="confidentiality integrity">  
319   <description>  
320     Protect messages from unauthorized reading or modification.  
321   </description>  
322 </intent>
```

324 *Snippet 3-3: Example Profile Intent*

325 3.4 PolicySets

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

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

```
329  
330 <policySet name="NCName"  
331     provides="listOfQNames"?  
332     appliesTo="xs:string"?  
333     attachTo="xs:string"?  
334     xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200903  
335     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">  
336   <policySetReference name="xs:QName"/> *  
337   <intentMap/> *  
338   <xs:any> *  
339 </policySet>
```

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

341

342 PolicySet has the attributes:

- 343 • @name (1..1) - the name for the policySet. The value of the @name attribute is the local part of a
344 QName. **The QName for a policySet MUST be unique amongst the set of policySets in the SCA**
345 **Domain.** [POL30017]
- 346 | —@appliesTo (0..1) - a string which is an XPath 1.0 expression identifying one or more SCA constructs
347 this policySet can configure. **The contents of @appliesTo MUST match the XPath 1.0 [XPATH]**
348 **production Expr.** [POL30018] The @appliesTo attribute uses the "Deployed Composites Infoset" as
349 described in [Appendix A The Deployed Composites Infoset](#)
- 350 | • ~~Section 4.4.1 "The Form of the @attachTo Attribute".~~
- 351 • @attachTo (0..1) - a string which is an XPath 1.0 expression identifying one or more elements in the
352 Domain. It is used to declare which set of elements the policySet is actually attached to. **The**
353 **contents of @attachTo MUST match the XPath 1.0 production Expr.** [POL30019] ~~The XPath value of~~
354 ~~the @attachTo attribute is evaluated against the "Deployed Composite Infoset" as described in The-~~
355 ~~@attachTo attribute uses the "Deployed Composite Infoset" as described in as described in Appendix~~
356 ~~A "The Deployed Composites Infoset". Section 4.4.1 "The Form of the @attachTo Attribute".~~ See the
357 section on "Attaching Intents and PolicySets to SCA Constructs" for more details on how this attribute
358 is used.
- 359 • @provides (0..1) - a list of intent QNames (that can be qualified), which declares the intents the
360 PolicySet provides.

361 PolicySet contains one or more of the element children

- 362 • intentMap element
- 363 • policySetReference element
- 364 • xs:any extensibility element

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

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

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

```

377 <policySet name="SecureReliablePolicy"
378     provides="serverAuthentication.message exactlyOne"
379     appliesTo="//sca:binding.ws"
380     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
381     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
382   <wsp:PolicyAttachment>
383     <!-- policy expression and policy subject for
384         "basic server authentication" -->
385     ...
386   </wsp:PolicyAttachment>
387   <wsp:PolicyAttachment>
388     <!-- policy expression and policy subject for
389         "reliability" -->
390     ...
391   </wsp:PolicyAttachment>
392 </policySet>
393 
```

394 *Snippet 3-5: Example policySet Definition*

395

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

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

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

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

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

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

424 Note that section 4.10.1 on Wire Validity specifies that the strict version of the WS-Policy intersection
425 algorithm is used to establish wire validity and determine the policies to be used. The strict version of
426 policy intersection algorithm ignores the ignorable attribute on assertions. This means that the ignorable
427 facility of WS-Policy cannot be used in policySets.

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

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

432 **3.4.1 IntentMaps**

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

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

436

```
437 <intentMap provides="xs:QName">  
438   <qualifier name="xs:string"?  
439     <xs:any>*  
440   </qualifier>  
441 </intentMap>
```

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

443

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

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

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

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

455 An intentMap element contains qualifier element children. Each qualifier element corresponds to a
 456 qualified intent where the unqualified form of that intent is the value of the @provides attribute value of
 457 the parent intentMap. The qualified intent is either included explicitly in the value of the enclosing
 458 policySet's @provides attribute or implicitly by that @provides attribute including the unqualified form of
 459 the intent.

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

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

```

466
467 <policySet name="SecureMessagingPolicies"
468   provides="confidentiality"
469   appliesTo="binding.ws"
470   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
471   xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
472   <intentMap provides="confidentiality" >
473     <qualifier name="transport">
474       <wsp:PolicyAttachment>
475         <!-- policy expression and policy subject for
476           "transport" alternative -->
477         ...
478       </wsp:PolicyAttachment>
479       <wsp:PolicyAttachment>
480         ...
481       </wsp:PolicyAttachment>
482     </qualifier>
483     <qualifier name="message">
484       <wsp:PolicyAttachment>
485         <!-- policy expression and policy subject for
486           "message" alternative -->
487         ...
488       </wsp:PolicyAttachment>
489     </qualifier>
490   </intentMap>
491 </policySet>

```

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

493

494 PolicySets can embed policies that are defined in any policy language. Although WS-Policy is the most
 495 common language for expressing interaction policies, it is possible to use other policy languages Snippet
 496 3-8 is an example of a policySet that embeds a policy defined in a proprietary language. This policy
 497 provides "serverAuthentication" for binding.ws.

498

```

499 <policySet name="AuthenticationPolicy"
500     provides="serverAuthentication"
501     appliesTo="binding.ws"
502     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
503   <e:policyConfiguration xmlns:e="http://example.com">
504     <e:authentication type="X509"/>
505     <e:trustedCAStore type="JKS"/>
506     <e:keyStoreFile>Foo.jks</e:keyStoreFile>
507     <e:keyStorePassword>123</e:keyStorePassword>
508   </e:authentication>
509 </e:policyConfiguration>
510 </policySet>

```

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

512 3.4.2 Direct Inclusion of Policies within PolicySets

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

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

523 3.4.3 Policy Set References

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

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

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

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

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

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

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

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

548

```
549 <policySet name="BasicAuthMsgProtSecurity"  
550     provides="serverAuthentication confidentiality"  
551     appliesTo="binding.ws"  
552     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
553   <policySetReference name="acme:ServerAuthenticationPolicies"/>  
554   <policySetReference name="acme:ConfidentialityPolicies"/>  
555 </policySet>
```

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

557

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

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

562

```
563 <policySet name="ServerAuthenticationPolicies"  
564     provides="serverAuthentication"  
565     appliesTo="binding.ws"  
566     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
567   <wsp:PolicyAttachment>  
568     <!-- policy expression and policy subject for  
569       "basic server authentication" -->  
570     ...  
571   </wsp:PolicyAttachment>  
572 </policySet>  
573  
574 <policySet name="acme:ConfidentialityPolicies"  
575     provides="confidentiality"  
576     bindings="binding.ws"  
577     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">  
578   <intentMap provides="confidentiality" >  
579     <qualifier name="transport">  
580       <wsp:PolicyAttachment>  
581         <!-- policy expression and policy subject for  
582           "transport" alternative -->  
583         ...  
584       </wsp:PolicyAttachment>  
585       <wsp:PolicyAttachment>  
586         ...  
587       </wsp:PolicyAttachment>  
588     </qualifier>  
589     <qualifier name="message">  
590       <wsp:PolicyAttachment>  
591         <!-- policy expression and policy subject for  
592           "message" alternative -->  
593         ...  
594       </wsp:PolicyAttachment>  
595     </qualifier>  
596   </intentMap>  
597 </policySet>
```

598 *Snippet 3-10: Example Included policySets for Snippet 3-9*

599

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

602

```
603 <policySet name="BasicAuthMsgProtSecurity"
```

```

604     provides="serverAuthentication confidentiality" appliesTo="binding.ws"
605     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
606     <wsp:PolicyAttachment>
607         <!-- policy expression and policy subject for
608             "basic server authentication" -->
609         ...
610     </wsp:PolicyAttachment>
611     <intentMap provides="confidentiality" >
612         <qualifier name="transport">
613             <wsp:PolicyAttachment>
614                 <!-- policy expression and policy subject for
615                     "transport" alternative -->
616                 ...
617             </wsp:PolicyAttachment>
618             <wsp:PolicyAttachment>
619                 ...
620             </wsp:PolicyAttachment>
621         </qualifier>
622         <qualifier name="message">
623             <wsp:PolicyAttachment>
624                 <!-- policy expression and policy subject for
625                     "message" alternative -->
626                 ...
627             </wsp:PolicyAttachment>
628         </qualifier>
629     </intentMap>
630 </policySet>

```

631 *Snippet 3-11: Equivalent policySet*

632 4 Attaching Intents and PolicySets to SCA Constructs

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

636 4.1 Attachment Rules — Intents

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

- 639 • Direct Attachment mechanism which is described in Section 4.2.
- 640 • External Attachment mechanism which is described in Section 4.3.

641 4.14.2 Direct Attachment of Intents

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

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

655 For example:

656

```
657 <service requires="acme:IntentName1 acme:IntentName2">  
658   <binding.xxx/>  
659   ...  
660 </service>  
661  
662 <reference requires="acme:IntentName1 acme:IntentName2">  
663   <binding.xxx/>  
664   ...  
665 </reference>
```

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

667

```
668 <service>  
669   <requires intents="acme:IntentName1 acme:IntentName2" />  
670   <binding.xxx/>  
671   ...  
672 </service>  
673  
674 <reference>  
675   <requires intents="acme:IntentName1 acme:IntentName2" />  
676   <binding.xxx/>  
677   ...  
678 </reference>
```

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

4.3 External Attachment of Intents

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

Intents can be attached to one or more SCA elements by using the intentAttachment element. This has the pseudo-schema:

```
<intentAttachment intents = "sca:listOfQNames" attachTo = "xs:string" ... />
```

@intents is a required attribute that takes as its value a list of QNames identifying intents.

@attachTo is a required attribute which takes as its value a string which is an XPath 1.0 expression identifying one or more elements in the Domain. It is used to declare which set of elements the intents are actually attached to. The contents of @attachTo MUST match the XPath 1.0 production Expr. [POL300xx] The XPath value of the @attachTo attribute is evaluated against the "Deployed Composite Infoset" as described in Appendix A "The Deployed Composites Infoset".

NOTE: We need to say somewhere that intentAttachment elements appear in the definitions.xml file.

During the deployment of SCA composites, all intentAttachment elements within the Domain MUST be evaluated to determine which intents are attached to the elements of the deployed composite. [POL400xx]

During the deployment of an SCA intent, the behavior of an SCA runtime MUST take ONE of the following forms:

- The intent is immediately attached to all deployed composites which satisfy the @attachTo attribute of the policySet.
- The intent is attached to a deployed composite which satisfies the @attachTo attribute of the intent when the composite is re-deployed. [POL400xx]

4.24.4 Attachment Rules - PolicySets

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

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

SCA runtimes MUST support at least one of the Direct Attachment and External Attachment mechanisms for policySet attachment. [POL40010] SCA implementations supporting only the External Attachment mechanism MUST ignore the policySetpolicy-sets that are applicable via the Direct Attachment mechanism. [POL40011] SCA implementations supporting only the Direct Attachment mechanism MUST ignore the policySetpolicy-sets that are applicable via the External Attachment mechanism. [POL40012] SCA implementations supporting both Direct Attachment and External Attachment mechanisms MUST ignore policySetpolicy-sets applicable to any given SCA element via the Direct Attachment mechanism when there exist policySetpolicy-sets applicable to the same SCA element via the External Attachment mechanism [POL40001]

4.34.5 Direct Attachment of PolicySets

Direct Attachment of PolicySets can be achieved by

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

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

For example:

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

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

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

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

Snippet 4-4: policySetAttachment Pseudo-Schema

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

For example:

```
<service> or <reference>...
  <binding.binding-type>
    <policySetAttachment name="sns:EnterprisePolicySet">
  </binding.binding-type>
  ...
</service> or </reference>
```

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

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

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

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

770

771 See also section 4.12.1 for a discussion of how intents are used to guide the selection and application of
772 specific policySets.

773 **4.44.6 External Attachment of PolicySets Mechanism**

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

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

780 a) through the @attachTo attribute of the policySet

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

782 During the deployment of SCA composites, all policySets within the Domain with an @attachTo attribute
783 MUST be evaluated to determine which policySets are attached to the ~~elements of the~~ newly deployed
784 composite. [POL40013]

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

- 787 • The policySet is immediately attached to all deployed composites which satisfy the @attachTo
788 attribute of the policySet.
- 789 • The policySet is attached to a deployed composite which satisfies the @attachTo attribute of the
790 policySet when the composite is re-deployed.

791 [POL40026]

792

793 ~~4.4.1 The Form of the @attachTo Attribute~~

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

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

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

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

803 ~~Where the policySet is intended to be specific to a particular component, the structuralURI [SCA-~~
804 ~~Asssembly] of the component is used along with the URIRef() XPath function to attach a policySet to~~
805 ~~a specific use of a nested component. The XPath expression can make use of the unique-~~
806 ~~structuralURI to indicate specific use instances, where different policySets need to be used for those~~
807 ~~different instances.~~

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

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

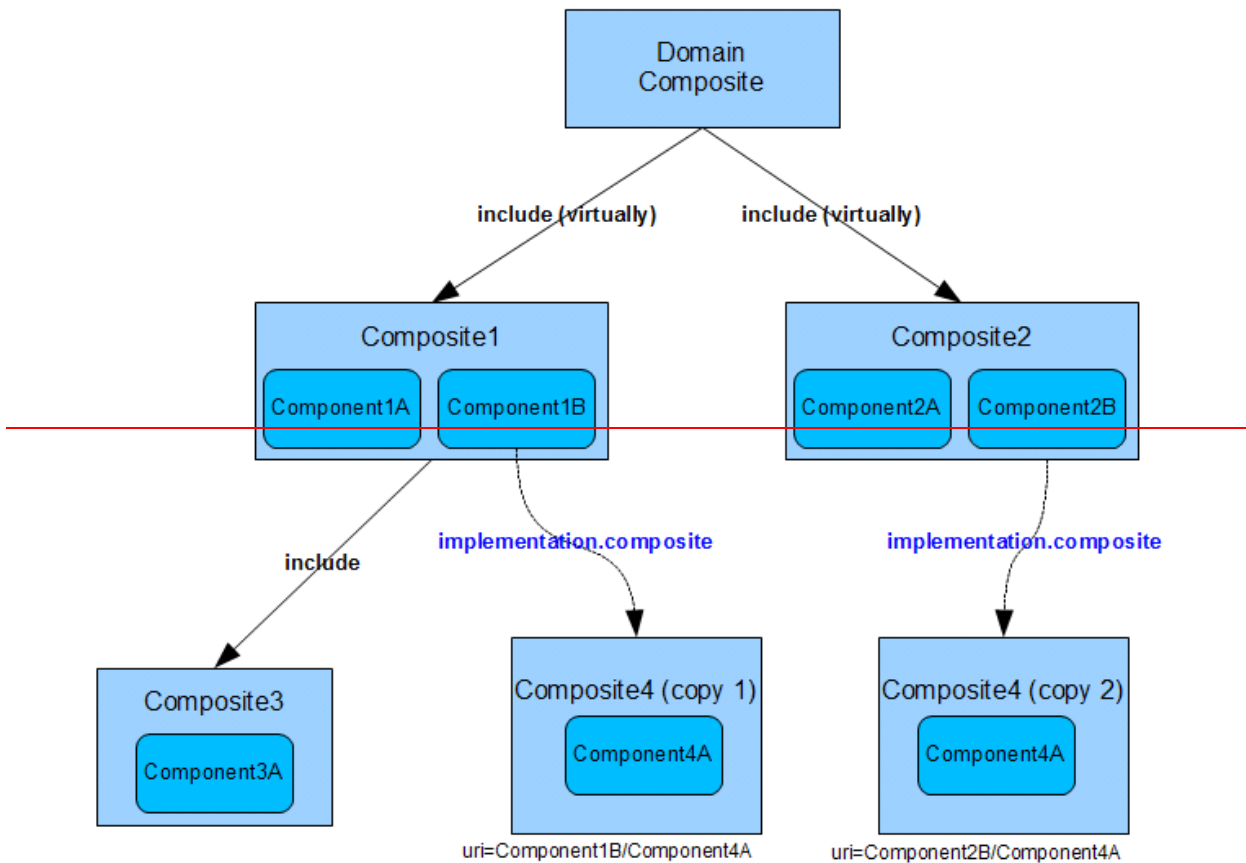
812 ~~2. By reference from another policySet element~~

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

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

- 818 • the identification of elements to which specific intents apply.
 819 This permits the attachment of a policySet to be linked to specific intents on the target element—for
 820 example, a policySet relating to encryption of messages can be targeted to services and references
 821 which have the **confidentiality** intent applied.
- 822 • the targeting of subelements of an interface, including operations and messages.
 823 This permits the attachment of a policySet to an individual operation or to an individual message
 824 within an interface, separately from the policies that apply to other operations or messages in the
 825 interface.
- 826 • the targeting of a specific use of a component, through its unique structuralURI [SCA-Assembly].
 827 This permits the attachment of a policySet to a specific use of a component in one context, that can
 828 be different from the policySet(s) that are applied to other uses of the same component.

829 Detail of the available XPath functions is given in the section "XPath Functions for the @attachTo-
 830 Attribute".
 831



832
 833 *Figure 4-1 Example Domain Composite Infoset*

834
 835 The SCA Domain in Figure 4-1 has been constructed from the composites and components shown in the
 836 figure. Composite1 and Composite2 were deployed into the Domain as described in [SCA-Assembly].

837 Composite3 is included in Composite1 using the SCA include mechanism described in [SCA-Assembly].
838 Composite4 is used as an implementation of Components 1B and 2B. Following the deployment of all the
839 composites, the Domain contains:

- 840 • 3 Composites that can be addressed as part of the Deployed Composites InfoSet; Composite1,
841 Composite2 and Composite4.
- 842 • all the components shown in the diagram. Components 1A, 2A, 3A, 4A (twice) are leaf
843 components.

844
845 The following snippets show example usage of the @attachTo attribute and provide the outcome based
846 on the Domain in Figure 4-1.

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

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

850
851 attach to both instances of Component4A

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

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

854
855 attach to the unique instance of Component4A when used by Component2B (Component2B is a
856 component at the Domain level)

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

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

859
860 attach to the services of Component3A which have the intent "intent1" applied

```
861 4. //component/binding.ws
```

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

863
864 attach to the web services binding of all components with a service or reference with a Web services
865 binding

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

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

868
869 attach to Component1A at the Domain level

870 4.4.24.6.1 Cases Where Multiple PolicySets are attached to a Single 871 Artifact

872
873 Multiple PolicySets can be attached to a single artifact. This can happen either as the result of one or
874 more direct attachments or as the result of one or more external attachments which target the particular
875 artifact.

879 **4.4.3 XPath Functions for the @attachTo Attribute**

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

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

- 884 • Picking out a specific interface
- 885 • Picking out a specific operation in an interface
- 886 • Picking out a specific message in an operation in an interface
- 887 • Picking out artifacts with specific intents

888 **4.4.3.1 Interface Related Functions**

889 **InterfaceRef(InterfaceName)**

890 picks out an interface identified by InterfaceName

891 **OperationRef(InterfaceName/OperationName)**

892 picks out the operation OperationName in the interface InterfaceName

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

894 picks out the message MessageName in the operation OperationName in the interface
895 InterfaceName.

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

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

899 Examples of the Interface functions:

900
901 `InterfaceRef("MyInterface")`

902 *Snippet 4-11: Example use of InterfaceRef*

903
904 picks out an interface with the name "MyInterface"

905
906 `OperationRef("MyInterface/MyOperation")`

907 *Snippet 4-12: Example use of OperationRef with a Path*

908
909 picks out the operation named "MyOperation" within the interface named "MyInterface"

910
911 `OperationRef("*/MyOperation")`

912 *Snippet 4-13: Example use of OperationRef without a Path*

913
914 picks out the operation named "MyOperation" from any interface

915
916 `MessageRef("MyInterface/MyOperation/MyMessage")`

917 *Snippet 4-14: Example use of MessageRef with a Path*

918

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

921
922 `MessageRef("*/*/MyMessage")`

923 *Snippet 4-15: Example use of MessageRef with a Path with Wildcards*

924
925 picks out the message named "MyMessage" from any operation in any interface

926 **4.4.3.2 Intent Based Functions**

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

930 **IntentRefs(IntentList)**

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

932
933 `IntentRefs("intent1")`

934 *Snippet 4-16: Example use of IntentRef*

935
936 picks out an artifact to which intent named "intent1" is attached

937
938 `IntentRefs("intent1 intent2")`

939 *Snippet 4-17: Example use of IntentRef with Multiple intents*

940
941 picks out an artifact to which intents named "intent1" AND "intent2" are attached

942
943 `IntentRefs("intent1 !intent2")`

944 *Snippet 4-18: Example use of IntentRef with Not Operator*

945
946 picks out an artifact to which intent named "intent1" is attached but NOT the intent named "intent2"

947 **4.4.3.3 URI Based Function**

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

951 **URIRef(URI)**

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

953 For a full description of structuralURIs, see the SCA Assembly specification [SCA-Assembly].

954 **Example:**

955
956 `URIRef("top_comp_name/middle_comp_name/lowest_comp_name")`

957 *Snippet 4-19: Example use of URIRef*

958

959 ~~picks out the particular use of a component—where component lowest_comp_name is used within the~~
960 ~~implementation of middle_comp_name within the implementation of the top-level (Domain-level)~~
961 ~~component top_comp_name.~~

962 4.54.7 Attaching Intents to SCA Elements

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

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

- 967 • the intents ~~expressed in its @requires attribute and/or its <requires> subelement attached to it either~~
968 ~~directly or externally.~~
- 969 • intents derived from the structural hierarchy of the element
- 970 • intents derived from the implementation hierarchy of the element

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

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

976 4.5.14.7.1 Implementation Hierarchy of an Element

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

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

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

990 4.5.24.7.2 Structural Hierarchy of an Element

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

993 As an example, for the composite in Snippet 4-16:

```
994  
995 <composite name="C1" requires="i1">  
996   <service name="CS" promotes="X/S">  
997     <binding.ws requires="i2">  
998   </service>  
999   <component name="X">  
1000     <implementation.java class="foo"/>  
1001     <service name="S" requires="i3">  
1002   </component>  
1003 </composite>
```

1004 Snippet 4-6: Example Composite to Illustrate Structural Hierarchy

1005
1006 - the structural hierarchy of the component service element with the name "S" is the component element
1007 named "X" and the composite element named "C1". Service "S" has intent "i3" and also has the intent "i1"
1008 if i1 is not mutually exclusive with i3.

1009 **Rule2: The intents declared on elements higher in the structural hierarchy of a given element MUST be**
1010 **applied to the element EXCEPT**

- 1011 • if any of the inherited intents is mutually exclusive with an intent applied on the element, then the
1012 inherited intent MUST be ignored
- 1013 • if the overall set of intents from the element itself and from its structural hierarchy contains both an
1014 unqualified version and a qualified version of the same intent, the qualified version of the intent MUST
1015 be used.

1016 [POL40005]

1017 **4.5.34.7.3 Combining Implementation and Structural Policy Data**

1018 When there are intents present in both hierarchies implementation intents are calculated before the
1019 structural intents. In other words, when combining implementation hierarchy and structural hierarchy
1020 policy data, Rule 1 MUST be applied BEFORE Rule 2. [POL40015]

1021 Note that each of the elements in the hierarchy below a <component> element, such as <service/>,
1022 <reference/> or <binding/>, inherits intents from the equivalent elements in the componentType of the
1023 implementation used by the component. So the <service/> element of the <component> inherits any
1024 intents on the <service/> element with the same name in the <componentType> - and a <binding/>
1025 element under the service in the component inherits any intents on the <binding/> element of the service
1026 (with the same name) in the componentType. Errors caused by mutually exclusive intents appearing on
1027 corresponding elements in the component and on the componentType only occur when those elements
1028 match one-to-one. Mutually exclusive intents can validly occur on elements that are at different levels in
1029 the structural hierarchy (as defined in Rule 2).

1030 Note that it might often be the case that <binding/> elements will be specified in the structure under the
1031 <component/> element in the composite file (especially at the Domain level, where final deployment
1032 configuration is applied) - these elements might have no corresponding elements defined in the
1033 componentType structure. In this situation, the <binding/> elements don't acquire any intents from the
1034 componentType directly (ie there are no elements in the implementation hierarchy of the <binding/>
1035 elements), but those <binding/> elements will acquire intents "flowing down" their structural hierarchy as
1036 defined in Rule 2 - so, for example if the <service/> element is marked with @requires="confidentiality",
1037 the bindings of that service will all inherit that intent, assuming that they don't have their own exclusive
1038 intents specified.

1039 Also, for example, where say a component <service.../> element has an intent that is mutually exclusive
1040 with an intent in the componentType<service.../> element with the same name, it is an error, but this
1041 differs when compared with the case of the <component.../> element having an intent that is mutually
1042 exclusive with an intent on the componentType <service/> element - because they are at different
1043 structural levels: the intent on the <component/> is ignored for that <service/> element and there is no
1044 error.

1045 **4.5.44.7.4 Examples**

1046 As an example, consider the composite in [Snippet 4-21](#) ~~Snippet 4-17~~: [the snippet below](#):

```
1047  
1048 <composite name="C1" requires="i1">  
1049   <service name="CS" promotes="X/S">  
1050     <binding.ws requires="i2">  
1051     </service>  
1052   <component name="X">  
1053     <implementation.java class="foo"/>  
1054     <service name="S" requires="i3">
```



```
1055     </component>
1056 </composite>
```

1057 *Snippet 4-7: Example composite with intents*

1058
1059 ...the component service with name "S" has the service named "S" in the componentType of
1060 the implementation in its implementation hierarchy, and the composite service named "CS"
1061 has the component service named "S" in its implementation hierarchy. Service "CS"
1062 acquires the intent "i3" from service "S" – and also gets the intent "i1" from its containing
1063 composite "C1" IF i1 is not mutually exclusive with i3.

1064 When intents apply to an element following the rules described and where no policySets are
1065 attached to the element, the intents for the element can be used to select appropriate
1066 policySets during deployment, using the external attachment mechanism.

1067 Consider the composite in Snippet 4-18:

```
1068  
1069 <composite requires="confidentiality">
1070   <service name="foo" .../>
1071   <reference name="bar" requires="confidentiality.message" />
1072 </composite>
```

1073 *Snippet 4-8: Example reference with intents*

1074
1075 ...in this case, the composite declares that all of its services and references guarantee confidentiality in
1076 their communication, but the "bar" reference further qualifies that requirement to specifically require
1077 message-level security. The "foo" service element has the default qualifier specified for the confidentiality
1078 intent (which might be transport level security) while the "bar" reference has the **confidentiality.message**
1079 intent.

1080 Consider the variation in Snippet 4-19 where a qualified intent is specified at the composite level:

```
1081  
1082 <composite requires="confidentiality.transport">
1083   <service name="foo" .../>
1084   <reference name="bar" requires="confidentiality.message" />
1085 </composite>
```

1086 *Snippet 4-9: Example Qualified intents*

1087
1088 In this case, both the **confidentiality.transport** and the **confidentiality.message** intent
1089 are applied for the reference 'bar'. If there are no bindings that support this combination, an
1090 error will be generated. However, since in some cases multiple qualifiers for the same intent
1091 can be valid or there might be bindings that support such combinations, the SCA
1092 specification allows this.

1093 It is also possible for a qualified intent to be further qualified. In our example, the
1094 **confidentiality.message** intent could be further qualified to indicate whether just the body of a message
1095 is protected, or the whole message (including headers) is protected. So, the second-level qualifiers might
1096 be "body" and "whole". The default qualifier might be "whole". If the "bar" reference from Snippet 4-19
1097 wanted only body confidentiality, it would state:

```
1098  
1099 <reference name="bar" requires="acme:confidentiality.message.body" />
```

1100 *Snippet 4-10: Example Second Level Qualifier*

1101

1102 The definition of the second level of qualification for an intent follows the same rules. As with other
1103 qualified intents, the name of the intent is constructed using the name of the qualifiable intent, the
1104 delimiter “.”, and the name of the qualifier.

1105 **4.64.8 Usage of Intent and Policy Set Attachment together**

1106 As indicated above, it is possible to attach both intents and policySets to an SCA element during
1107 development. The most common use cases for attaching both intents and concrete policySets to an
1108 element are with binding and reference elements.

1109 When the @requires attribute or the <requires> subelement and one or both of the direct policySet
1110 attachment mechanisms are used together during development, it indicates the intention of the developer
1111 to configure the element, such as a binding, by the application of specific policySet(s) to this element.

1112 [The same behavior can be enabled by external attachment of intents and policySets.](#)

1113
1114 Developers who attach intents and policySets in conjunction with each other need to be aware of the
1115 implications of how the policySets are selected and how the intents are utilized to select specific
1116 intentMaps, override defaults, etc. The details are provided in the Section [Guided Selection of](#)
1117 [PolicySets using Intents.](#)

1118 **4.74.9 Intents and PolicySets on Implementations and Component** 1119 **Types**

1120 It is possible to specify intents and policySets within a component’s implementation, which get exposed to
1121 SCA through the corresponding *component type*. How the intents or policies are specified within an
1122 implementation depends on the implementation technology. For example, Java can use an @requires
1123 annotation to specify intents.

1124 The intents and policySets specified within an implementation can be found on the

1125 <sca:implementation.*> and the <sca:service> and <sca:reference> elements of the component type.^{1,7}

1126 [Snippet 4-25 The for example below shows direct attachment of intents and policySets using the](#)
1127 [@requires and @policySets attributes:](#)

```
1128  
1129 <componentType>  
1130   <implementation.* requires="listOfQNames" policySets="="listOfQNames">  
1131     ...  
1132   </implementation>  
1133   <service name="myService" requires="listOfQNames"  
1134     policySets="listOfQNames">  
1135     ...  
1136   </service>  
1137   <reference name="myReference" requires="listOfQNames"  
1138     policySets="="listOfQNames">  
1139     ...  
1140   </reference>  
1141   ...  
1142 </componentType>
```

1143 *Snippet 4-11: Example of intents on an implementation*

1144
1145 Intents expressed in the component type are handled according to the rule defined for the implementation
1146 hierarchy. See [Intent rule 2](#)

1147 For explicitly listed policySets, the list in the component using the implementation can override policySets
1148 from the component type. **If a component has any policySets attached to it (by any means), then any**
1149 **policySets attached to the componentType MUST be ignored.** [POL40006]

1150 4.84.10 Intents on Interfaces

1151 Interfaces are used in association with SCA services and references. These interfaces can be declared
1152 in SCA composite files and also in SCA componentType files. The interfaces can be defined using a
1153 number of different interface definition languages which include WSDL, Java interfaces and C++ header
1154 files.

1155 It is possible for some interfaces to be referenced from an implementation rather than directly from any
1156 SCA files. An example of this usage is a Java implementation class file that has a reference declared
1157 that in turn uses a Java interface defined separately. When this occurs, the interface definition is treated
1158 from an SCA perspective as part of the componentType of the implementation, logically being part of the
1159 declaration of the related service or reference element.

1160 Both the declaration of interfaces in SCA and also the definitions of interfaces can carry policy-related
1161 information. In particular, both the declarations and the definitions can have either intents attached to
1162 them, or policySets attached to them - or both. For SCA declarations, the intents and policySets always
1163 apply to the whole of the interface (ie all operations and all messages within each operation). For
1164 interface definitions, intents and policySets can apply to the whole interface or they can apply only to
1165 specific operations within the interface or they can even apply only to specific messages within particular
1166 operations. (To see how this is done, refer to the places in the SCA specifications that deal with the
1167 relevant interface definition language)

1168 This means, in effect, that there are 4 places which can hold policy related information for interfaces:

- 1169 1. The interface definition file that is referenced from the component type.
- 1170 2. The interface declaration for a service or reference in the component type
- 1171 3. The interface definition file that is referenced from the component declaration in a composite
- 1172 4. The interface declaration within a component

1173 When calculating the set of intents and set of policySets which apply to either a service element or to a
1174 reference element of a component, intents and policySets from the interface definition and from the
1175 interface declaration(s) MUST be applied to the service or reference element and to the binding
1176 element(s) belonging to that element. [POL40016]

1177 The locations where interfaces are defined and where interfaces are declared in the componentType and
1178 in a component MUST be treated as part of the implementation hierarchy as defined in Section 4.5
1179 Attaching intents to SCA elements. [POL40019]

1180 4.94.11 BindingTypes and Related Intents

1181 SCA Binding types implement particular communication mechanisms for connecting components
1182 together. See detailed discussion in the [SCA Assembly Specification](#) [SCA-Assembly]. Some binding
1183 types can realize intents inherently by virtue of the kind of protocol technology they implement (e.g. an
1184 SSL binding would natively support confidentiality). For these kinds of binding types, it might be the case
1185 that using that binding type, without any additional configuration, provides a concrete realization of an
1186 intent. In addition, binding instances which are created by configuring a binding type might be able to
1187 provide some intents by virtue of their configuration. It is important to know, when selecting a binding to
1188 satisfy a set of intents, just what the binding types themselves can provide and what they can be
1189 configured to provide.

1190 The bindingType element is used to declare a class of binding available in a SCA Domain. The pseudo-
1191 schema for the bindingType element is shown in Snippet 4-22:

1192

```
1193 <bindingType type="NCName"  
1194     alwaysProvides="listOfQNames" ?  
1195     mayProvide="listOfQNames" ? />
```

1196 *Snippet 4-12: bindingTypePseudo-Schema*

1197

- 1198 • @type (1..1) – declares the NCName of the bindingType, which is used to form the QName of the
1199 bindingType. The QName of the bindingType MUST be unique amongst the set of bindingTypes in
1200 the SCA Domain. [POL40020]
- 1201 • @alwaysProvides (0..1) – a list of intent QNames that are natively provided. A natively provided intent
1202 is hard-coded into the binding implementation. The function represented by the intent cannot be
1203 turned off.
- 1204 • @mayProvides (0..1) – a list of intent QNames that are natively provided by the binding
1205 implementation, but which are activated only when present in the intent set that is applied to a binding
1206 instance.

1207 A binding implementation MUST implement all the intents listed in the @alwaysProvides and
1208 @mayProvides attributes. [POL40021]

1209 The kind of intents a given binding might be capable of providing, beyond these inherent intents, are
1210 implied by the presence of policySets that declare the given binding in their @appliesTo attribute.

1211 For example, if the policySet in Snippet 4-23 is available in a SCA Domain it says that the (example)
1212 foo:binding.ssl can provide “reliability” in addition to any other intents it might provide inherently.

1213

```
1214 <policySet name="ReliableSSL" provides="exactlyOnce"  
1215     appliesTo="foo:binding.ssl">  
1216     ...  
1217 </policySet>
```

1218 *Snippet 4-13: Example policySet Applied to a binding*

1219 **4.104.12 Treatment of Components with Internal Wiring**

1220 This section discusses the steps involved in the development and deployment of a component and its
1221 relationship to selection of bindings and policies for wiring services and references.

1222 The SCA developer starts by defining a component. Typically, this contains services and references. It
1223 can also have intents *attacheddefined* at various locations within composite and component types as well
1224 as policySets *attacheddefined* at various locations.

1225 Both for ease of development as well as for deployment, the wiring constraints to relate services and
1226 references need to be determined. This is accomplished by matching constraints of the services and
1227 references to those of corresponding references and services in other components.

1228 In this process, the intents, and the policySets that apply to both sides of a wire play an important role. In
1229 addition, concrete policies need to be selected that satisfy the intents for the service and the reference
1230 and are also compatible with each other. For services and references that make use of bidirectional
1231 interfaces, the same determination of matching policySets also has to take place for callbacks.

1232 Determining compatibility of wiring plays an important role prior to deployment as well as during the
1233 deployment phases of a component. For example, during development, it helps a developer to determine
1234 whether it is possible to wire services and references using the *-policySets* available in the development
1235 environment. During deployment, the wiring constraints determine whether wiring can be achievable. It
1236 also aids in adding additional concrete policies or making adjustments to concrete policies in order to
1237 deliver the constraints. Here are the concepts that are needed in making wiring decisions:

- 1238 • The set of intents that individually apply to *each* service or reference.
- 1239 • When possible the intents that are applied to the service, the reference and callback (if any) at the
1240 other end of the wire. This set is called the *required intent set* and only applies when dealing with a
1241 wire connecting two components within the same SCA Domain. When external connections are
1242 involved, from clients or to services that are outside the SCA domain, intents are only available for the
1243 end of the connection that is inside the domain. See Section "[Preparing Services and References
1244 for External Connection](#)" for more details.
- 1245 • The policySets that apply to each service or reference.

1246 The set of provided intents for a binding instance is the union of the set of intents listed in the
1247 "alwaysProvides" attribute and the set of intents listed in the "mayProvides" attribute of its binding type.
1248 The capabilities represented by the "alwaysProvides" intent set are always present, irrespective of the
1249 configuration of the binding instance. Each capability represented by the "mayProvides" intent set is only
1250 present when the list of intents applied to the binding instance (either applied directly, or inherited)
1251 contains the particular intent (or a qualified version of that intent, if the intent set contains an unqualified
1252 form of a qualifiable intent). When an
1253 intent is directly provided by the binding type, there is no need to apply a policy set that provides that
1254 intent.
1255 When bidirectional interfaces are in use, the same process of selecting policySets to provide the intents is
1256 also performed for the callback bindings.

1257 **4.10.14.12.1 Determining Wire Validity and Configuration**

1258 The above approach determines the policySets that are used in conjunction with the binding instances
1259 listed for services and references. For services and references that are resolved using SCA wires, the
1260 policySets chosen on each side of the wire might or might not be compatible. The following approach is
1261 used to determine whether they are compatible and whether the wire is valid. If the wire uses a
1262 bidirectional interface, then the following technique ensures that valid configured policySets can be found
1263 for both directions of the bidirectional interface.

1264 The SCA runtime MUST determine the compatibility of the policySets at each end of a wire using the
1265 compatibility rules of the policy language used for those policySets. [POL40022] The policySets at each
1266 end of a wire MUST be incompatible if they use different policy languages. [POL40023] However, there is
1267 a special case worth mentioning:

- 1268 • If both sides of the wire use identical policySets (by referring to the same policySet by its QName in
1269 both sides of the wire), then they are compatible.

1270 Where the policy language in use for a wire is WS-Policy, strict WS-Policy intersection MUST be used to
1271 determine policy compatibility. [POL40024]

1272 In order for a reference to connect to a particular service, the policies of the reference MUST intersect
1273 with the policies of the service. [POL40025]

1274 **4.11.4.13 Preparing Services and References for External 1275 Connection**

1276 Services and references are sometimes not intended for SCA wiring, but for communication with software
1277 that is outside of the SCA domain. References can contain bindings that specify the endpoint address of
1278 a service that exists outside of the current SCA domain. Services can specify bindings that can be
1279 exposed to clients that are outside of the SCA domain.

1280 Matching service/reference policies across the SCA Domain boundary MUST use WS-Policy compatibility
1281 (strict WS-Policy intersection) if the policies are expressed in WS-Policy syntax. [POL40007] For other
1282 policy languages, the policy language defines the comparison semantics.

1283 For external services and references that make use of bidirectional interfaces, the same determination of
1284 matching policies has to also take place for the callback.

1285 The policies that apply to the service/reference are computed as discussed in [Guided Selection of
1286 PolicySets using Intents](#).

1287 **4.14 Deployment Guided Selection of PolicySets using Intents**

1288 The SCA Assembly Specification [SCA-Assembly] describes how to gather together SCA
1289 artifacts and deploy them to create executable components. This section discusses the Policy aspects of
1290 deployment: how intents, intentAttachments and policySets are gathered together, how intents are

1291 satisfied by the policies in the policySets and the conditions under which redeployment becomes
1292 necessary as intents, intentAttachments and policySets change.

1293
1294 When a composite is deployed, the SCA runtime has to re-evaluate the ~~external attachment~~ XPath
1295 expression which is the value of the @attachTo attribute of every intentAttachment and policySet in the
1296 SCA Domain. For each intentAttachment To start the Policy aspect of the deployment process, the
1297 intents that are available in the SCA domain, are examined and the XPath expressions that are the
1298 values of their @attachTo attributes is are evaluated and the intent is s are attached to the SCA elements
1299 selected by the @attachTo XPath expressions. Note that the @attachTo attribute may be missing for a
1300 particular intentAttachment or its value may be empty, in which case no attachment is performed.

1301 llowing this, if external attachment of policySets is supported, then each policySet in the SCA domain is
1302 examined; the value of the @attachTo attributes are evaluated and the policySets are attached to the
1303 SCA elements selected by the XPath expression. If the @attachTo attribute is missing or its value is
1304 empty, no attachment is performed for that particular policySet.

1305 When an intent is deployed and the SCA runtime supports external policySet attachment, the SCA
1306 runtime has to re-evaluate the external attachment XPath expression of every policySet in the SCA
1307 Domain.

1308 The SCA runtime MUST raise an error if the value of the @attachTo XPath expression resolves to an
1309 SCA <property> element, or any of its children. [POL40002]

1310
1311 If both intents as well as policySets need to be attached externally to SCA elements
1312 The intents MUST be attached before policySets [POL4xxxx]

1313
1314 The algorithm for matching intents with policySets is described in the following subsection.
1315 As discussed in SCA Assembly Specification [SCA-Assembly] artifacts in the SCA domain are in one
1316 of three states:

- 1317 1. Installed
- 1318 2. Deployed
- 1319 3. Running

1320 Intents and policySets may be managed separately from other SCA artifacts and may change while other
1321 artifacts are in one of the above states. This may lead to situations that cause the set of artifacts in the
1322 SCA domain to become inconsistent with deployed or running artifacts. This can happen if:

- 1323 • A new intentAttachment is introduced into the domain
- 1324 • An existing intentAttachment is removed from the domain.
- 1325 • The @attachTo attribute of an existing intentAttachment changes.
- 1326 • A new policySet with a non-empty @attachTo attribute is introduced into the domain and
1327 external attachment of policySets is supported
- 1328 • The @attachTo attribute of an existing policySet is changed and external attachment of
1329 policySets is supported

1330 When the set of artifacts in the domain becomes inconsistent with the depoyed/running artifacts, the
1331 deployer will want to redeploy at some point. Since redeployment may or may not succeed, The
1332 deployer must consider whether to stop running artifacts before redeploying. Also, in some cases, it
1333 may be possible to change the policies on running artifacts without stopping and restarting. This, too,
1334 would influence when to redeploy.

1335 Redeployment, when it occurs, would first perform external attachment of intents followed by
1336 external attachment of policySets (see [POL4xxxx] above). After this, the algorithm described below for
1337 matching intents with policySets would be run. This algorithm may succeed or fail, in that the set of
1338 intents in the domain may or may not be satisfied.

1339 If the algorithm fails, because one or more intents are left unsatisfied, an error will be raised and the
1340 deployer[DAB2] may wish to correct the error and attempt to redeploy[DAB3]. In this case the changed
1341 artifacts are not deployed and, no change SHOULD be made to deployed and running artifacts
1342 [POL4xxxx].

1343 p[DAB4]

1344 If the algorithm succeeds in that all intents are satisfied, then the policies attached to one or more
1345 deployed SCA elements may change. When policies are added, removed or replaced by deployment
1346 actions, the components whose policies are affected by these deployment actions MAY have their
1347 policies updated by the SCA runtime dynamically without the need to stop and restart those components.
1348 [POL4xxxx]. NOTE: Corresponds to [ASM12014]

1349 Where components are updated by deployment actions (their configuration is changed in some way,
1350 which includes changing the policies of component references), the new configuration MUST apply to all
1351 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY
1352 choose to maintain existing instances with the old configuration of components updated by deployment
1353 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.
1354 [ASM12016]

1355

1356

1357 ~~This section describes the selection of concrete policies that provide a set of intents~~
1358 ~~expressed for an element. The purpose is to construct the set of concrete policies that are attached to an~~
1359 ~~element taking into account the explicitly declared policySets that are attached to an element as well as~~
1360 ~~policySets that are externally attached. The aim is to satisfy all of the intents expressed for each element.~~

1361 ~~If the unqualified form of a qualifiable intent is attached to an element, it can be satisfied by a policySet~~
1362 ~~that specifies any one of qualified forms of the intent in the value of its @provides attribute, or it can be~~
1363 ~~satisfied by a policySet which @provides the unqualified form of the intent. If the qualified form of the~~
1364 ~~intent is attached to an element then it can be satisfied only by a policy that @provides that qualified form~~
1365 ~~of the intent.~~

1366 **4.14.1 -Matching Intents and PolicySets**

1367 ~~This section describes the selection of concrete policies that provide the~~
1368 ~~requirements expressed by the set of intents associated with an SCA element. The purpose is to~~
1369 ~~construct the set of concrete policies that are attached to an element taking into account the explicitly~~
1370 ~~declared policySets that are attached to an element as well as policySets that are externally attached.~~
1371 ~~The aim is to satisfy all of the intents applied to associated with each element.~~

1372 ~~If the unqualified form of a qualifiable intent is attached to an element, it can be satisfied by a~~
1373 ~~policySet that specifies any one of qualified forms of the intent in the value of its @provides attribute, or it~~
1374 ~~can be satisfied by a policySet which @provides the unqualified form of the intent. If the qualified form of~~
1375 ~~the intent is attached to an element then it can be satisfied only by a policy that @provides that qualified~~
1376 ~~form of the intent.~~

1377 **4.11.1**

1378 **Note: In the following, the following rule is observed when an intent set is computed.**

1379 When a profile intent is encountered in either a global @requires attribute, an intent/@requires attribute, a
1380 <requires> subelement or a policySet/@provides attribute, the profile intent is immediately replaced by
1381 the intents that it composes (i.e. all the intents that appear in the profile intent's @requires attribute). This
1382 rule is applied recursively until profile intents do not appear in an intent set. [This is stated generally here,
1383 in order to not have to restate this at multiple places].

1384 The **required intent set** that is attached to an element is:

- 1385 1. The set of intents ~~specified in the element's @requires attribute.~~ attached to the element either by
1386 direct attachment or external attachment via the mechanisms described in sections 4.2 and 4.3.

- 1387 2. add any intents found in any related interface definition or declaration, as described in the section
 1388 [4.10 Intents on Interfaces](#).
- 1389 3. add any intents found on elements below the target element in its implementation hierarchy as
 1390 defined in Rule 1 in Section 4.5
- 1391 4. add any intents ~~found in the @requires attributes and <requires> subelements of attached to~~ each
 1392 ancestor element in the element's structural hierarchy as defined in [Rule 2](#) in Section 4.5
- 1393 5. ~~removeless~~ any intents that do not include the target element's type in their @constrains attribute.
- 1394 6. remove the unqualified version of an intent if the set also contains a qualified version of that intent

1395 **If the required intent set contains a mutually exclusive pair of intents the SCA runtime MUST reject the**
 1396 **document containing the element and raise an error.** [POL40017]

1397 The **directly provided intent set** for an element is the set of intents listed in the @alwaysProvides
 1398 attribute combined with the set of intents listed in the @mayProvides attribute of the bindingType or
 1399 implementationType declaration for a binding or implementation element respectively.

1400 The **set of PolicySets attached to an element** include those **explicitly specified** using the @policySets
 1401 attribute or the <policySetAttachment/> element and those which are **externally attached**.

1402 A policySet **applies to** a target element if the result of the XPath expression contained in the policySet's
 1403 @appliesTo attribute, when evaluated against the document containing the target element, includes the
 1404 target element. For example, @appliesTo="binding.ws[@impl='axis']" matches any binding.ws element
 1405 that has an @impl attribute value of 'axis'.

1406 The set of **explicitly specified** policySets for an element is:

- 1407 1. The union of the policySets specified in the element's @policySets attribute and those specified in
 1408 any <policySetAttachment/> child element(s).
- 1409 2. add the policySets declared in the @policySets attributes and <policySetAttachment/> elements from
 1410 elements in the structural hierarchy of the element.
- 1411 3. remove any policySet where the policySet does not apply to the target element.
 1412 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*

1413 The set of **externally attached** policySets for an element is:

- 1414 1. Each <PolicySet/> in the Domain where the element is targeted by the @attachTo attribute of the
 1415 policySet
- 1416 2. remove any policySet where the policySet does not apply to the target element.
 1417 *It is not an error for a policySet to be attached to an element to which it doesn't apply.*

1418 A policySet **provides an intent** if any of the statements are true:

- 1419 1. The intent is contained in the ~~policySet~~ @provides list of the policySet.
- 1420 2. The intent is a qualified intent and the unqualified form of the intent is contained in the ~~policySet~~
 1421 @provides list of the policySet.
- 1422 3. The policySet @provides list contains a qualified form of the intent (where the intent is qualifiable).

1423 **All intents in the required intent set for an element SHOULD be provided by the directly provided intents**
 1424 **set and the set of policySets that apply to the element.** [POL40018]

1425 If the combination of implementationType / bindingType / collection of policySets does not satisfy all of
 1426 the intents which apply to the element, the configuration is not valid. However, an SCA Runtime can allow
 1427 a deployer to force deployment even in the presence of such errors as long as a warning is issued or
 1428 some other indication is provided that deployment has been forced. Details of the behavior of the
 1429 deployer in such situations are not specified in this specification.

5 Implementation Policies

1430

1431 The basic model for Implementation Policies is very similar to the model for interaction policies described
1432 above. Abstract QoS requirements, in the form of intents, can be associated with SCA component
1433 implementations to indicate implementation policy requirements. These abstract capabilities are mapped
1434 to concrete policies via policySets at deployment time. Alternatively, policies can be associated directly
1435 with component implementations using policySets. Intents and policySets can be attached to associated-
1436 with an implementation using any of the mechanisms described in section 4 above.

1437 Snippet 5-1 shows how one way of associating intents can be associated with an implementation:

1438

1439

```
1440 <component name="xs:NCName" ... >  
1441   <implementation.* ... requires="listOfQNames">  
1442     ...  
1443   </implementation>  
1444   ...  
1445 </component>
```

1446 *Snippet 5-1: Example of intents Associated with an implementation*

1447

1448 If, for example, one of the intent names in the value of the @requires attribute is 'logging', this indicates
1449 that all messages to and from the component have to be logged. The technology used to implement the
1450 logging is unspecified. Specific technology is selected when the intent is mapped to a policySet (unless
1451 the implementation type has native support for the intent, as described in the next section). A list of
1452 implementation intents can also be specified by any ancestor element of the <sca:implementation>
1453 element. The effective list of implementation intents is the union of intents specified on the
1454 implementation element and all its ancestors.

1455 In addition, one or more policySets can be specified directly by associating them with the implementation
1456 of a component.

1457

```
1458 <component name="xs:NCName" ... >  
1459 <implementation.* ... policySets="="listOfQNames">  
1460   ...  
1461 </implementation>  
1462   ...  
1463 </component>
```

1464 *Snippet 5-2: Example of policySets Associated with an implementation*

1465

1466 Snippet 5-2 shows how intents and policySets can be specified on a component. It is also possible to
1467 specify intents and policySets within the implementation. How this is done is defined by the
1468 implementation type.

1469 The intents and policy sets are specified on the <sca:implementation.*> element within the component
1470 type. This is important because intent and policy set definitions need to be able to specify that they
1471 constrain an appropriate implementation type.

1472

```
1473 <componentType>  
1474   <implementation.* requires="listOfQNames" policySets="listOfQNames">  
1475     ...  
1476   </implementation>  
1477   ...
```

1478 `</componentType>`

1479 *Snippet 5-3: intents and policySets Constraining an implementation*

1480
1481 When applying policies, the intents attached to the implementation are added to the intents attached to
1482 the using component. For the explicitly listed policySets, the list in the component can override policySets
1483 from the componentType.

1484 Some implementation intents are targeted at `<binding/>` elements rather than at `<implementation/>`
1485 elements. This occurs in cases where there is a need to influence the operation of the binding
1486 implementation code rather than the code directly related to the implementation itself. Implementation
1487 elements of this kind will have a `@constrains` attribute pointing to a binding element, with a `@intentType`
1488 of "implementation".

1489 5.1 Natively Supported Intents

1490 Each implementation type (e.g. `<sca:implementation.java>` or `<sca:implementation.bpel>`) has an
1491 **implementation type definition** within the SCA Domain. An implementation type definition is declared
1492 using an `implementationType` element within a `<definitions/>` declaration. The pseudo-schema for the
1493 `implementationType` element is shown in Snippet 5-4:

1494

```
1495 <implementationType type="QName"  
1496 alwaysProvides="listOfQNames"? mayProvide="listOfQNames"? />
```

1497 *Snippet 5-4: implementationType Pseudo-Schema*

1498

1499 The implementation Type element has the following attributes:

- 1500 • **name : QName (1..1)** - the name of the implementationType. The implementationType name attribute
1501 MUST be the QName of an XSD global element definition used for implementation elements of that
1502 type. [POL50001] For example: "sca:implementation.java".
- 1503 • **alwaysProvides : list of QNames (0..1)** - a set of intents. The intents in the alwaysProvides set are
1504 always provided by this implementation type, whether the intents are attached to the using
1505 component or not.
- 1506 • **mayProvide : list of QNames (0..1)** - a set of intents. The intents in the mayProvide set are provided
1507 by this implementation type if the intent in question is attached to the using component.

1508 5.2 Writing PolicySets for Implementation Policies

1509 | The `@appliesTo` and `@attachTo` attributes for a policySet takes an XPath expression that is applied to a
1510 service, reference, binding or an implementation element. For implementation policies, in most cases, all
1511 that is needed is the QName of the implementation type. Implementation policies can be expressed using
1512 any policy language (which is to say, any configuration language). For example, XACML or EJB-style
1513 annotations can be used to declare authorization policies. Other capabilities could be configured using
1514 completely proprietary configuration formats.

1515 | For example, a policySet declared to turn on trace-level logging for a BPEL component `could` be
1516 declared as is Snippet 5-5:

1517

```
1518 <policySet name="loggingPolicy" provides="acme:logging.trace"  
1519 appliesTo="sca:implementation.bpel" ...>  
1520 <acme:processLogging level="3"/>  
1521 </policySet>
```

1522 *Snippet 5-5: Example policySet Applied to implementation.bpel*

1523 **5.2.1 Non WS-Policy Examples**

1524 Authorization policies expressed in XACML [could](#) be used in the framework in two ways:

1525 1. Embed XACML expressions directly in the PolicyAttachment element using the extensibility elements
1526 discussed above, or

1527 2. Define WS-Policy assertions to wrap XACML expressions.

1528 For EJB-style authorization policy, [the same approach could be used](#):

1529 1. Embed EJB-annotations in the PolicyAttachment element using the extensibility elements discussed
1530 above, or

1531 2. Use the WS-Policy assertions defined as wrappers for EJB annotations.

1532 6 Roles and Responsibilities

1533 There are 4 roles that are significant for the SCA Policy Framework. The following is a list of the roles and
1534 the artifacts that the role creates:

- 1535 • Policy Administrator – policySet definitions and intent definitions
- 1536 • Developer – Implementations and component types
- 1537 • Assembler - Composites
- 1538 • Deployer – Composites and the SCA Domain (including the logical Domain-level composite)

1539 6.1 Policy Administrator

1540 An intent represents a requirement that a developer or assembler can make, which ultimately have to be
1541 satisfied at runtime. The full definition of the requirement is the informal text description in the intent
1542 definition.

1543 The **policy administrator**'s job is to both define the intents that are available and to define the policySets
1544 that represent the concrete realization of those informal descriptions for some set of binding type or
1545 implementation types. See the sections on intent and policySet definitions for the details of those
1546 definitions.

1547 6.2 Developer

1548 When it is possible for a component to be written without assuming a specific binding type for its services
1549 and references, then the **developer** uses intents to specify requirements in a binding neutral way.

1550 If the developer requires a specific binding type for a component, then the developer can specify bindings
1551 and policySets with the implementation of the component. Those bindings and policySets will be
1552 represented in the component type for the implementation (although that component type might be
1553 generated from the implementation).

1554 If any of the policySets used for the implementation include intentMaps, then the default choice for the
1555 intentMap can be overridden by an assembler or deployer by requiring a qualified intent that is present in
1556 the intentMap.

1557 6.3 Assembler

1558 An **assembler** creates composites. Because composites are implementations, an assembler is like a
1559 developer, except that the implementations created by an assembler are composites made up of other
1560 components wired together. So, like other developers, the assembler can specify intents or bindings or
1561 policySets on any service or reference of the composite.

1562 However, in addition the definition of composite-level services and references, it is also possible for the
1563 assembler to use the policy framework to further configure components within the composite. The
1564 assembler can add additional requirements to any component's services or references or to the
1565 component itself (for implementation policies). The assembler can also override the bindings or
1566 policySets used for the component. See the assembly specification's description of overriding rules for
1567 details on overriding.

1568 As a shortcut, an assembler can also specify intents and policySets on any element in the composite
1569 definition, which has the same effect as specifying those intents and policySets on every applicable
1570 binding or implementation below that element (where applicability is determined by the @appliesTo
1571 attribute of the policySet definition or the @constrains attribute of the intent definition).

1572 **6.4 Deployer**

1573 A **deployer** deploys implementations (typically composites) into the SCA Domain. It is the
1574 deployers job to make the final decisions about all configurable aspects of an implementation that is to be
1575 deployed and to make sure that all intents are satisfied.

1576 If the deployer determines that an implementation is correctly configured as it is, then the implementation
1577 can be deployed directly. However, more typically, the deployer will create a new composite, which
1578 contains a component for each implementation to be deployed along with any changes to the bindings or
1579 policySets that the deployer desires.

1580 When the deployer is determining whether the existing list of policySets is correct for a component, the
1581 deployer needs to consider both the explicitly listed policySets as well as the policySets that will be
1582 chosen according to the algorithm specified in [Guided Selection of PolicySets using Intents](#).

7 Security Policy

1583

1584 The SCA Security Model provides SCA developers the flexibility to specify the necessary level of security
1585 protection for their components to satisfy business requirements without the burden of understanding
1586 detailed security mechanisms.

1587 The SCA Policy framework distinguishes between two types of policies: **interaction policy** and
1588 **implementation policy**. Interaction policy governs the communications between clients and service
1589 providers and typically applies to Services and References. In the security space, interaction policy is
1590 concerned with client and service provider authentication and message protection requirements.
1591 Implementation policy governs security constraints on service implementations and typically applies to
1592 Components. In the security space, implementation policy concerns include access control, identity
1593 delegation, and other security quality of service characteristics that are pertinent to the service
1594 implementations.

1595 The SCA security interaction policy can be specified via intents or policySets. Intents represent security
1596 quality of service requirements at a high abstraction level, independent from security protocols, while
1597 policySets specify concrete policies at a detailed level, which are typically security protocol specific.

1598 The SCA security policy can be specified either in an SCA composite or by using the External Policy
1599 Attachment Mechanism or by annotations in the implementation code. Language-specific annotations are
1600 described in the respective language Client and Implementation specifications.

7.1 SCA Security Policy Intents

1601

1602 The SCA security specification defines the following intents to specify interaction policy:

1603 serverAuthentication, clientAuthentication, confidentiality, and integrity.

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

7.2 Interaction Security Policy

1622

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. [POL70011]

1632 **message** – the message qualifier specifies that the qualified intent is realized at the message level of the
1633 communication protocol. When a serverAuthentication, clientAuthentication, confidentiality or integrity
1634 intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication,
1635 clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication
1636 protocol. [POL70012]

1637

1638 Snippet 7-1 shows the usage of intents and qualified intents.

1639

```
1640 <composite name="example" requires="confidentiality">  
1641   <service name="foo" />  
1642   ...  
1643   <reference name="bar" requires="confidentiality.message" />  
1644 </composite>
```

1645 *Snippet 7-1: Example using Qualified Intents*

1646

1647 In this case, the composite declares that all of its services and references have to guarantee
1648 confidentiality in their communication by setting requires="confidentiality". This applies to the "foo"
1649 service. However, the "bar" reference further qualifies that requirement to specifically require message-
1650 level security by setting requires="confidentiality.message".

1651 7.3 Implementation Security Policy Intent

1652 The SCA Security specification defines the **authorization** intent to specify implementation policy.

1653 **authorization** – the authorization intent is used to indicate that a client needs to be authorized before
1654 being allowed to use the service. Being authorized means that a check is made as to whether any
1655 policies apply to the client attempting to use the service, and if so, those policies govern whether or not
1656 the client is allowed access. When **authorization** is present, an SCA Runtime MUST ensure that the client
1657 is authorized to use the service. [POL70001]

1658 This unqualified authorization intent implies that basic "Subject-Action-Resource" authorization support is
1659 required, where Subject may be as simple as a single identifier representing the identity of the client,
1660 Action may be a single identifier representing the operation the client intends to apply to the Resource,
1661 and the Resource may be a single identifier representing the identity of the Resource to which the Action
1662 is intended to be applied.

8 Reliability Policy

1663

1664 Failures can affect the communication between a service consumer and a service provider.

1665 Depending on the characteristics of the binding, these failures could cause messages to be redelivered,
1666 delivered in a different order than they were originally sent out or even worse, could cause messages to
1667 be lost. Some transports like JMS provide built-in reliability features such as “at least once” and “exactly
1668 once” message delivery. Other transports like HTTP need to have additional layers built on top of them to
1669 provide some of these features.

1670 The events that occur due to failures in communication can affect the outcome of the service invocation.
1671 For an implementation of a stock trade service, a message redelivery could result in a new trade. A client
1672 (i.e. consumer) of the same service could receive a fault message if trade orders are not delivered to the
1673 service implementation in the order they were sent out. In some cases, these failures could have dramatic
1674 consequences.

1675 An SCA developer can anticipate some types of failures and work around them in service
1676 implementations. For example, the implementation of a stock trade service could be designed to support
1677 duplicate message detection. An implementation of a purchase order service could have built in logic that
1678 orders the incoming messages. In these cases, service implementations don't need the binding layers to
1679 provide these reliability features (e.g. duplicate message detection, message ordering). However, this
1680 comes at a cost: extra complexity is built in the service implementation. Along with business logic, the
1681 service implementation has additional logic that handles these failures.

1682 Although service implementations can work around some of these types of failures, it is worth noting that
1683 workarounds are not always possible. A message can be lost or expire even before it is delivered to the
1684 service implementation.

1685 Instead of handling some of these issues in the service implementation, a better way is to use a binding
1686 or a protocol that supports reliable messaging. This is better, not just because it simplifies application
1687 development, it can also lead to better throughput. For example, there is less need for application-level
1688 acknowledgement messages. A binding supports reliable messaging if it provides features such as
1689 message delivery guarantees, duplicate message detection and message ordering.

1690 It is very important for the SCA developer to be able to require, at design-time, a binding or protocol that
1691 supports reliable messaging. SCA defines a set of policy intents that can be used for specifying reliable
1692 messaging Quality of Service requirements. These reliable messaging intents establish a contract
1693 between the binding layer and the application layer (i.e. service implementation or the service consumer
1694 implementation) (see below).

1695 8.1 Reliability Policy Intents

1696 Based on the use-cases described above, the following policy intents are defined:

1697 1. **atLeastOnce** - The binding implementation guarantees that a message that is successfully sent by a
1698 service consumer is delivered to the destination (i.e. service implementation). The message could be
1699 delivered more than once to the service implementation. **When *atLeastOnce* is present, an SCA
1700 Runtime MUST deliver a message to the destination service implementation, and MAY deliver
1701 duplicates of a message to the service implementation. [POL80001]**

1702 The binding implementation guarantees that a message that is successfully sent by a service
1703 implementation is delivered to the destination (i.e. service consumer). The message could be
1704 delivered more than once to the service consumer.

1705 2. **atMostOnce** - The binding implementation guarantees that a message that is successfully sent by a
1706 service consumer is not delivered more than once to the service implementation. The binding
1707 implementation does not guarantee that the message is delivered to the service implementation.
1708 **When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service**

1709 implementation, and MUST NOT deliver duplicates of a message to the service implementation.
1710 [POL80002]

1711 The binding implementation guarantees that a message that is successfully sent by a service
1712 implementation is not delivered more than once to the service consumer. The binding implementation
1713 does not guarantee that the message is delivered to the service consumer.

1714 3. **ordered** – The binding implementation guarantees that the messages sent by a service client via a
1715 single service reference are delivered to the target service implementation in the order in which they
1716 were sent by the service client. This intent does not guarantee that messages that are sent by a
1717 service client are delivered to the service implementation. Note that this intent has nothing to say
1718 about the ordering of messages sent via different service references by a single service client, even if
1719 the same service implementation is targeted by each of the service references. **When ordered is
1720 present, an SCA Runtime MUST deliver messages sent by a single source to a single destination
1721 service implementation in the order that the messages were sent by that source.** [POL80003]

1722 For service interfaces that involve messages being sent back from the service implementation to the
1723 service client (eg. a service with a callback interface), for this intent, the binding implementation
1724 guarantees that the messages sent by the service implementation over a given wire are delivered to
1725 the service client in the order in which they were sent by the service implementation. This intent does
1726 not guarantee that messages that are sent by the service implementation are delivered to the service
1727 consumer.

1728 4. **exactlyOnce** - The binding implementation guarantees that a message sent by a service consumer is
1729 delivered to the service implementation. Also, the binding implementation guarantees that the
1730 message is not delivered more than once to the service implementation. **When exactlyOnce is
1731 present, an SCA Runtime MUST deliver a message to the destination service implementation and
1732 MUST NOT deliver duplicates of a message to the service implementation.** [POL80004]

1733 The binding implementation guarantees that a message sent by a service implementation is delivered
1734 to the service consumer. Also, the binding implementation guarantees that the message is not
1735 delivered more than once to the service consumer.

1736 NOTE: This is a profile intent, which is composed of *atLeastOnce* and *atMostOnce*.

1737 This is the most reliable intent since it guarantees the following:

- 1738 – message delivery – all the messages sent by a sender are delivered to the service
1739 implementation (i.e. Java class, BPEL process, etc.).
- 1740 – duplicate message detection and elimination – a message sent by a sender is not processed
1741 more than once by the service implementation.

1742 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1743 How can a binding implementation guarantee that a message that it receives is delivered to the service
1744 implementation? One way to do it is by persisting the message and keeping redelivering it until it is
1745 processed by the service implementation. That way, if the system crashes after delivery but while
1746 processing it, the message will be redelivered on restart and processed again. Since a message could be
1747 delivered multiple times to the service implementation, this technique usually requires the service
1748 implementation to perform duplicate message detection. However, that is not always possible. Often
1749 times service implementations that perform critical operations are designed without having support for
1750 duplicate message detection. Therefore, they cannot *process* an incoming
1751 message more than once.

1752 Also, consider the scenario where a message is delivered to a service implementation that does not
1753 handle duplicates - the system crashes after a message is delivered to the service implementation but
1754 before it is completely processed. Does the underlying layer redeliver the message on restart? If it did
1755 that, there is a risk that some critical operations (e.g. sending out a JMS message or updating a DB table)
1756 will be executed again when the message is processed. On the other hand, if the underlying layer does
1757 not redeliver the message, there is a risk that the message is never completely processed.

1758 This issue cannot be safely solved unless all the critical operations performed by the service

1759 implementation are running in a transaction. Therefore, *exactlyOnce* cannot be assured without involving
1760 the service implementation. In other words, an *exactlyOnce* message delivery does not guarantee
1761 *exactlyOnce* message processing unless the service implementation is transactional. It's worth noting that
1762 this is a necessary condition but not sufficient. The underlying layer (e.g. binding implementation,
1763 container) would have to ensure that a message is not redelivered to the service implementation after the
1764 transaction is committed. As an example, a way to ensure it when the binding uses JMS is by making
1765 sure the operation that acknowledges the message is executed in the same transaction the service
1766 implementation is running in.

1767 **8.2 End-to-end Reliable Messaging**

1768 Failures can occur at different points in the message path: in the binding layer on the sender side, in the
1769 transport layer or in the binding layer on the receiver side. The SCA service developer doesn't really care
1770 where the failure occurs. Whether a message was lost due to a network failure or due to a crash of the
1771 machine where the service is deployed, is not that important. What is important is that the contract
1772 between the application layer (i.e. service implementation or service consumer) and the binding layer is
1773 not violated (e.g. a message that was successfully transmitted by a sender is always delivered to the
1774 destination; a message that was successfully transmitted by a sender is not delivered more than once to
1775 the service implementation, etc). It is worth noting that the binding layer could throw an exception when a
1776 sender (e.g. service consumer, service implementation) sends a message out. This is not considered a
1777 successful message transmission.

1778 In order to ensure the semantics of the reliable messaging intents, the entire message path, which is
1779 composed of the binding layer on the client side, the transport layer and the binding layer on the service
1780 side, has to be reliable.

1781 9 Transactions

1782 SCA recognizes that the presence or absence of infrastructure for ACID transaction coordination has a
1783 direct effect on how business logic is coded. In the absence of ACID transactions, developers have to
1784 provide logic that coordinates the outcome, compensates for failures, etc. In the presence of ACID
1785 transactions, the underlying infrastructure is responsible for ensuring the ACID nature of all interactions.
1786 SCA provides declarative mechanisms for describing the transactional environment needed by the
1787 business logic.

1788 Components that use a synchronous interaction style can be part of a single, distributed ACID transaction
1789 within which all transaction resources are coordinated to either atomically commit or rollback. The
1790 transmission or receipt of oneway messages can, depending on the transport binding, be coordinated as
1791 part of an ACID transaction as illustrated in the *OneWay Invocations* section below. Well-known, higher-
1792 level patterns such as store-and-forward queuing can be accomplished by composing transacted one-
1793 way messages with reliable-messaging policies.

1794 This document describes the set of abstract policy intents – both implementation intents and interaction
1795 intents – that can be used to describe the requirements on a concrete service component and binding
1796 respectively.

1797 9.1 Out of Scope

1798 The following topics are outside the scope of this document:

- 1799 • The means by which transactions are created, propagated and established as part of an execution
1800 context. These are details of the SCA runtime provider and binding provider.
- 1801 • The means by which a transactional resource manager (RM) is accessed. These include, but are not
1802 restricted to:
 - 1803 – abstracting an RM as an `sca:component`
 - 1804 – accessing an RM directly in a language-specific and RM-specific fashion
 - 1805 – abstracting an RM as an `sca:binding`

1806 9.2 Common Transaction Patterns

1807 In the absence of any transaction policies there is no explicit transactional behavior defined for the SCA
1808 service component or the interactions in which it is involved and the transactional behavior is
1809 environment-specific. An SCA runtime provider can choose to define an out of band default transactional
1810 behavior that applies in the absence of any transaction policies.

1811 Environment-specific default transactional behavior can be overridden by specifying transactional intents
1812 described in this document. The most common transaction patterns can be summarized:

1813 **Managed, shared global transaction pattern** – the service always runs in a global transaction context
1814 regardless of whether the requester runs under a global transaction. If the requester does run under a
1815 transaction, the service runs under the same transaction. Any outbound, synchronous request-response
1816 messages will – unless explicitly directed otherwise – propagate the service’s transaction context. This
1817 pattern offers the highest degree of data integrity by ensuring that any transactional updates are
1818 committed atomically

1819 **Managed, local transaction pattern** – the service always runs in a managed local transaction context
1820 regardless of whether the requester runs under a transaction. Any outbound messages will not propagate
1821 any transaction context. This pattern is advisable for services that wish the SCA runtime to demarcate
1822 any resource manager local transactions and do not require the overhead of atomicity.

1823 The use of transaction policies to specify these patterns is illustrated later in Table 9-2.

1824 9.3 Summary of SCA Transaction Policies

1825 This specification defines implementation and interaction policies that relate to transactional QoS in
1826 components and their interactions. The SCA transaction policies are specified as intents which represent
1827 the transaction quality of service behavior offered by specific component implementations or bindings.

1828 SCA transaction policy can be specified either in an SCA composite or annotatively in the implementation
1829 code. Language-specific annotations are described in the respective language binding specifications, for
1830 example the [SCA Java Common Annotations and APIs specification](#) [SCA-Java-Annotations].

1831 This specification defines the following implementation transaction policies:

- 1832 • `managedTransaction` – Describes the service component’s transactional environment.
- 1833 • `transactedOneWay` and `immediateOneWay` – two mutually exclusive intents that describe whether
1834 the SCA runtime will process `OneWay` messages immediately or will enqueue (from a client
1835 perspective) and dequeue (from a service perspective) a `OneWay` message as part of a global
1836 transaction.

1837 This specification also defines the following interaction transaction policies:

- 1838 • `propagatesTransaction` and `suspendsTransaction` – two mutually exclusive intents that describe
1839 whether the SCA runtime propagates any transaction context to a service or reference on a
1840 synchronous invocation.

1841 Finally, this specification defines a profile intent called `managedSharedTransaction` that combines the
1842 `managedTransaction` intent and the `propagatesTransaction` intent so that the ***managed, shared global***
1843 ***transaction pattern*** is easier to configure.

1844 9.4 Global and local transactions

1845 This specification describes “managed transactions” in terms of either “global” or “local” transactions. The
1846 “managed” aspect of managed transactions refers to the transaction environment provided by the SCA
1847 runtime for the business component. Business components can interact with other business components
1848 and with resource managers. The managed transaction environment defines the transactional context
1849 under which such interactions occur.

1850 9.4.1 Global transactions

1851 From an SCA perspective, a global transaction is a unit of work scope within which transactional work is
1852 atomic. If multiple transactional resource managers are accessed under a global transaction then the
1853 transactional work is coordinated to either atomically commit or rollback regardless using a 2PC protocol.
1854 A global transaction can be propagated on synchronous invocations between components – depending
1855 on the interaction intents described in this specification - such that multiple, remote service providers can
1856 execute distributed requests under the same global transaction.

1857 9.4.2 Local transactions

1858 From a resource manager perspective a resource manager local transaction (RMLT) is simply the
1859 absence of a global transaction. But from an SCA perspective it is not enough to simply declare that a
1860 piece of business logic runs without a global transaction context. Business logic might need to access
1861 transactional resource managers without the presence of a global transaction. The business logic
1862 developer still needs to know the expected semantic of making one or more calls to one or more resource
1863 managers, and needs to know when and/or how the resource managers local transactions will be
1864 committed. The term *local transaction containment* (LTC) is used to describe the SCA environment where
1865 there is no global transaction. The boundaries of an LTC are scoped to a remotable service provider
1866 method and are not propagated on invocations between components. Unlike the resources in a global
1867 transaction, RMLTs coordinated within a LTC can fail independently.

1868

1869 The two most common patterns for components using resource managers outside a global transaction
1870 are:

- 1871 • The application desires each interaction with a resource manager to commit after every interaction.
1872 This is the default behavior provided by the **noManagedTransaction** policy (defined below in
1873 Transaction implementation policy) in the absence of explicit use of RMLT verbs by the application.
- 1874 • The application desires each interaction with a resource manager to be part of an extended local
1875 transaction that is committed at the end of the method. This behavior is specified by the
1876 **managedTransaction.local** policy (defined below in Transaction implementation policy).

1877 While an application can use interfaces provided by the resource adapter to explicitly demarcate resource
1878 manager local transactions (RMLT), this is a generally undesirable burden on applications, which typically
1879 prefer all transaction considerations to be managed by the SCA runtime. In addition, once an application
1880 codes to a resource manager local transaction interface, it might never be redeployed with a different
1881 transaction environment since local transaction interfaces might not be used in the presence of a global
1882 transaction. This specification defines intents to support both these common patterns in order to provide
1883 portability for applications regardless of whether they run under a global transaction or not.

1884 9.5 Transaction implementation policy

1885 9.5.1 Managed and non-managed transactions

1886 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents describe the
1887 transactional environment needed by a service component or composite. SCA provides transaction
1888 environments that are managed by the SCA runtime in order to remove the burden of coding transaction
1889 APIs directly into the business logic. The **managedTransaction** and **noManagedTransaction** intents
1890 can be attached to the `sca:composite` or `sca:componentType` elements.

1891 The mutually exclusive **managedTransaction** and **noManagedTransaction** intents are defined as
1892 follows:

- 1893 • **managedTransaction** – a managed transaction environment is necessary in order to run this
1894 component. The specific type of managedTransaction needed is not constrained. The valid qualifiers
1895 for this intent are mutually exclusive.
 - 1896 – **managedTransaction.global** – There has to be an atomic transaction in order to run this
1897 component. For a component marked with **managedTransaction.global**, the SCA runtime
1898 MUST ensure that a global transaction is present before dispatching any method on the
1899 component. [POL90003] The SCA runtime uses any transaction propagated from the client
1900 or else begins and completes a new transaction. See the **propagatesTransaction** intent
1901 below for more details.
 - 1902 – **managedTransaction.local** – indicates that the component cannot tolerate running as part
1903 of a global transaction. A component marked with **managedTransaction.local** MUST run
1904 within a local transaction containment (LTC) that is started and ended by the SCA runtime.
1905 [POL90004] Any global transaction context that is propagated to the hosting SCA runtime is
1906 not visible to the target component. Any interaction under this policy with a resource manager
1907 is performed in an extended resource manager local transaction (RMLT). Upon successful
1908 completion of the invoked service method, any RMLTs are implicitly requested to commit by
1909 the SCA runtime. Note that, unlike the resources in a global transaction, RMLTs so
1910 coordinated in a LTC can fail independently. If the invoked service method completes with a
1911 non-business exception then any RMLTs are implicitly rolled back by the SCA runtime. In this
1912 context a business exception is any exception that is declared on the component interface
1913 and is therefore anticipated by the component implementation. The manner in which
1914 exceptions are declared on component interfaces is specific to the interface type – for
1915 example, Java interface types declare Java exceptions, WSDL interface types define
1916 `wsdl:faults`. Local transactions MUST NOT be propagated outbound across remotable
1917 interfaces. [POL90006]

- 1918 • **noManagedTransaction** – indicates that the component runs without a managed transaction, under
1919 neither a global transaction nor an LTC. A transaction that is propagated to the hosting SCA runtime
1920 MUST NOT be joined by the hosting runtime on behalf of a component marked with
1921 noManagedTransaction. [POL90007] When interacting with a resource manager under this policy, the
1922 application (and not the SCA runtime) is responsible for controlling any resource manager local
1923 transaction boundaries, using resource-provider specific interfaces (for example a Java
1924 implementation accessing a JDBC provider has to choose whether a Connection is set to
1925 autoCommit(true) or else it has to call the Connection commit or rollback method). SCA defines no
1926 APIs for interacting with resource managers.
 - 1927 • **(absent)** – The absence of a transaction implementation intent leads to runtime-specific behavior. A
1928 runtime that supports global transaction coordination can choose to provide a default behavior that is
1929 the managed, shared global transaction pattern but it is not mandated to do so.
- 1930 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1931 9.5.2 OneWay Invocations

1932 When a client uses a reference and sends a OneWay message then any client transaction context is not
1933 propagated. However, the OneWay invocation on the reference can itself be **transacted**. Similarly, from a
1934 service perspective, any received OneWay message cannot propagate a transaction context but the
1935 delivery of the OneWay message can be **transacted**. A **transacted** OneWay message is a one-way
1936 message that - because of the capability of the service or reference binding - can be enqueued (from a
1937 client perspective) or dequeued (from a service perspective) as part of a global transaction.

1938 SCA defines two mutually exclusive implementation intents, **transactedOneWay** and
1939 **immediateOneWay**, that determine whether OneWay messages are transacted or delivered immediately.

1940 Either of these intents can be attached to the sca:service or sca:reference elements or they can be
1941 attached to the sca:component element, indicating that the intent applies to any service or reference
1942 element children.

1943 The intents are defined as follows:

- 1944 • **transactedOneWay** – When a reference is marked as transactedOneWay, any OneWay invocation
1945 messages MUST be transacted as part of a client global transaction. [POL90008]
1946 If the client component is not configured to run under a global transaction or if the binding does not
1947 support transactional message sending, then a reference MUST NOT be marked as
1948 transactedOneWay. [POL90009] If a service is marked as transactedOneWay, any OneWay
1949 invocation message MUST be received from the transport binding in a transacted fashion, under the
1950 target service's global transaction. [POL90010] The receipt of the message from the binding is not
1951 committed until the service transaction commits; if the service transaction is rolled back the the
1952 message remains available for receipt under a different service transaction. If the component is not
1953 configured to run under a global transaction or if the binding does not support transactional message
1954 receipt, then a service MUST NOT be marked as transactedOneWay. [POL90011]
- 1955 • **immediateOneWay** – When applied to a reference indicates that any OneWay invocation messages
1956 MUST be sent immediately regardless of any client transaction. [POL90012] When applied to a
1957 service indicates that any OneWay invocation MUST be received immediately regardless of any
1958 target service transaction. [POL90013] The outcome of any transaction under which an
1959 immediateOneWay message is processed has no effect on the processing (sending or receipt) of that
1960 message.

1961 The absence of either intent leads to runtime-specific behavior. The SCA runtime can send or receive a
1962 OneWay message immediately or as part of any sender/receiver transaction. The results of combining
1963 this intent and the **managedTransaction** implementation policy of the component sending or receiving
1964 the transacted OneWay invocation are summarized low.below in Table 9-1.

1965

transacted/immediate intent	managedTransaction (client or service implementation intent)	Results
transactedOneWay	managedTransaction.global	OneWay interaction (either client message enqueue or target service dequeue) is committed as part of the global transaction.
transactedOneWay	managedTransaction.local or noManagedTransaction	If a transactedOneWay intent is combined with the managedTransaction.local or noManagedTransaction implementation intents for either a reference or a service then an error MUST be raised during deployment. [POL90027]
immediateOneWay	Any value of managedTransaction	The OneWay interaction occurs immediately and is not transacted.
<absent>	Any value of managedTransaction	Runtime-specific behavior. The SCA runtime can send or receive a OneWay message immediately or as part of any sender/receiver transaction.

1966 Table 9-1 Transacted OneWay interaction intent

1967

1968 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1969 9.6 Transaction interaction policies

1970 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can be attached
 1971 either to an interface (e.g. Java annotation or WSDL attribute) or explicitly to an sca:service and
 1972 sca:reference XML element to describe how any client transaction context will be made available and
 1973 used by the target service component. Section 9.6.1 considers how these intents apply to service
 1974 elements and Section 9.6.2 considers how these intents apply to reference elements.

1975 The formal definitions of these intents are in the [Intent Definitions appendix](#).

1976 9.6.1 Handling Inbound Transaction Context

1977 The mutually exclusive **propagatesTransaction** and **suspendsTransaction** intents can be attached to
 1978 an sca:service XML element to describe how a propagated transaction context is handled by the SCA
 1979 runtime, prior to dispatching a service component. If the service requester is running within a transaction
 1980 and the service interaction policy is to propagate that transaction, then the primary business effects of the
 1981 provider's operation are coordinated as part of the client's transaction – if the client rolls back its
 1982 transaction, then work associated with the provider's operation will also be rolled back. This allows clients
 1983 to know that no compensation business logic is necessary since transaction rollback can be used.

1984 These intents specify a contract that has to be implemented by the SCA runtime. This aspect of a
 1985 service component is most likely captured during application design. The **propagatesTransaction** or
 1986 **suspendsTransaction** intent can be attached to sca:service elements and their children. The intents are
 1987 defined as follows:

- 1988 • **propagatesTransaction** – A service marked with propagatesTransaction MUST be dispatched under
 1989 any propagated (client) transaction. [POL90015] Use of the **propagatesTransaction** intent on a
 1990 service implies that the service binding MUST be capable of receiving a transaction context.
 1991 [POL90016] However, it is important to understand that some binding/policySet combinations that
 1992 provide this intent for a service will *need* the client to propagate a transaction context.

1993 In SCA terms, for a reference wired to such a service, this implies that the reference has to use either
 1994 the **propagatesTransaction** intent or a binding/policySet combination that does propagate a
 1995 transaction. If, on the other hand, the service does not *need* the client to provide a transaction (even
 1996 though it has the *capability* of joining the client's transaction), then some care is needed in the
 1997 configuration of the service. One approach to consider in this case is to use two distinct bindings on
 1998 the service, one that uses the **propagatesTransaction** intent and one that does not - clients that do
 1999 not propagate a transaction would then wire to the service using the binding without the
 2000 **propagatesTransaction** intent specified.

- 2001 • **suspendsTransaction** – A service marked with **suspendsTransaction** MUST NOT be dispatched
 2002 under any propagated (client) transaction. [POL90017]

2003 The absence of either interaction intent leads to runtime-specific behavior; the client is unable to
 2004 determine from transaction intents whether its transaction will be joined.

2005 The SCA runtime MUST ignore the **propagatesTransaction** intent for **OneWay** methods. [POL90025]

2006 These intents are independent from the implementation's **managedTransaction** intent and provides no
 2007 information about the implementation's transaction environment.

2008 The combination of these service interaction policies and the **managedTransaction** implementation
 2009 policy of the containing component completely describes the transactional behavior of an invoked service,
 2010 as summarized in Table 9-2:
 2011

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" [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.

2012 Table 9-2 Combining service transaction intents

2013

2014 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A
2015 runtime that supports global transaction coordination can choose to provide a default behavior that is the
2016 managed, shared global transaction pattern.

2017 9.6.2 Handling Outbound Transaction Context

2018 The mutually exclusive *propagatesTransaction* and *suspendsTransaction* intents can also be attached
2019 to an sca:reference XML element to describe whether any client transaction context is propagated to a
2020 target service when a synchronous interaction occurs through the reference. These intents specify a
2021 contract that has to be implemented by the SCA runtime. This aspect of a service component is most
2022 likely captured during application design.

2023 Either the *propagatesTransaction* or *suspendsTransaction* intent can be attached to sca:service
2024 elements and their children. The intents are defined as defined in Section 9.6.1.

2025 When used as a reference interaction intent, the meaning of the qualifiers is as follows:

- 2026 • **propagatesTransaction** – When a reference is marked with propagatesTransaction, any transaction
2027 context under which the client runs MUST be propagated when the reference is used for a request-
2028 response interaction [POL90020] The binding of a reference marked with propagatesTransaction has
2029 to be capable of propagating a transaction context. The reference needs to be wired to a service that
2030 can join the client's transaction. For example, any service with an intent that @requires
2031 *propagatesTransaction* can always join a client's transaction. The reference consumer can then be
2032 designed to rely on the work of the target service being included in the caller's transaction.
- 2033 • **suspendsTransaction** – When a reference is marked with suspendsTransaction, any transaction
2034 context under which the client runs MUST NOT be propagated when the reference is used.
2035 [POL90022] The reference consumer can use this intent to ensure that the work of the target service
2036 is not included in the caller's transaction. .
- 2037 • The absence of either interaction intent leads to runtime-specific behavior. The SCA runtime can
2038 choose whether or not to propagate any client transaction context to the referenced service,
2039 depending on the SCA runtime capability.

2040 These intents are independent from the client's *managedTransaction* implementation intent. The
2041 combination of the interaction intent of a reference and the *managedTransaction* implementation policy
2042 of the containing component completely describes the transactional behavior of a client's invocation of a
2043 service. Table 9-3 summarizes the results of the combination of either of these interaction intents with the
2044 *managedTransaction* implementation policy of the containing component.

2045

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 propagatesTransaction if component is marked with "ManagedTransaction.local" or with "noManagedTransaction" [POL90023]

suspendsTransaction	Any value of managedTransaction	The target service will not run under the same transaction as any client transaction. This combination is used for the managed, local transaction pattern described in Common Transaction Patterns.
---------------------	---------------------------------	--

2046 *Table 9-3 Transaction propagation reference intents*

2047
 2048 Note - the absence of either interaction or implementation intents leads to runtime-specific behavior. A
 2049 runtime that supports global transaction coordination can choose to provide a default behavior that is the
 2050 managed, shared global transaction pattern.

2051 Table 9-4 shows the valid combination of interaction and implementation intents on the client and service
 2052 that result in a single global transaction being used when a client invokes a service through a reference.

2053

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

2054 *Table 9-4 Intents for end-to-end transaction propagation*

2055
 2056 **Transaction context MUST NOT be propagated on OneWay messages.** [POL90024] The SCA runtime
 2057 ignores *propagatesTransaction* for OneWay operations.

2058 9.6.3 Combining implementation and interaction intents

2059 The **managed, local transaction pattern** can be configured quite easily by combining the
 2060 managedTransaction.global intent with the propagatesTransaction intent. This is illustrated in **Error!**
 2061 **Reference source not found..** In order to enable easier configuration of this pattern, a profile intent
 2062 called managedSharedTransaction is defined as in section **Error! Reference source not found..**

2063 9.6.4 Web services binding for propagatesTransaction policy

2064 Snippet 9-1 shows a policySet that provides the *propagatesTransaction* intent and applies to a Web
 2065 service binding (binding.ws). When used on a service, this policySet would require the client to send a
 2066 transaction context using the mechanisms described in the [Web Services Atomic Transaction](#) [WS-
 2067 AtomicTransaction] specification.

2068

```

2069 <policySet name="JoinsTransactionWS" provides="sca:propagatesTransaction"
2070           appliesTo="sca:binding.ws">
2071   <wsp:Policy>
2072     <wsat:ATAssertion
2073       xmlns:wsat="http://docs.oasis-open.org/ws-tx/wsat/2006/06"/>
2074   </wsp:Policy>
2075 </policySet>

```

2076 *Snippet 9-1: Example policySet Providing propagatesTransaction*

2077

10 Miscellaneous Intents

2078 The following are standard intents that apply to bindings and are not related to either security, reliable
2079 messaging or transactionality:

- 2080 • **SOAP** – The SOAP intent specifies that the SOAP messaging model is used for delivering messages.
2081 It does not require the use of any specific transport technology for delivering the messages, so for
2082 example, this intent can be supported by a binding that sends SOAP messages over HTTP, bare
2083 TCP or even JMS. If the intent is attached in an unqualified form then any version of SOAP is
2084 acceptable. Standard mutually exclusive qualified intents also exist for SOAP.1_1 and SOAP.1_2,
2085 which specify the use of versions 1.1 or 1.2 of SOAP respectively. When SOAP is present, an SCA
2086 Runtime MUST use the SOAP messaging model to deliver messages. [POL100001] When a SOAP
2087 intent is qualified with 1_1 or 1_2, then SOAP version 1.1 or SOAP version 1.2 respectively MUST be
2088 used to deliver messages. [POL100002]
- 2089 • **JMS** – The JMS intent does not specify a wire-level transport protocol, but instead requires that
2090 whatever binding technology is used, the messages are able to be delivered and received via the
2091 JMS API. When JMS is present, an SCA Runtime MUST ensure that the binding used to send and
2092 receive messages supports the JMS API. [POL100003]
- 2093 • **noListener** – This intent can only be used within the @requires attribute of a reference. The
2094 noListener intent MUST only be declared on a @requires attribute of a reference. [POL100004] It
2095 states that the client is not able to handle new inbound connections. It requires that the binding and
2096 callback binding be configured so that any response (or callback) comes either through a back
2097 channel of the connection from the client to the server or by having the client poll the server for
2098 messages. When noListener is present, an SCA Runtime MUST not establish any connection from a
2099 service to a client. [POL100005] An example policy assertion that would guarantee this is a WS-
2100 Policy assertion that applies to the <binding.ws> binding, which requires the use of WS-Addressing
2101 with anonymous responses (e.g. <wsaw:Anonymous>required</wsaw:Anonymous>” – see
2102 <http://www.w3.org/TR/ws-addr-wsdl/#anonelement>).
- 2103 • **asyncInvocation** – This intent can be attached to an operation or a complete interface, indicating
2104 that the operation(s) are long-running request-response operation(s) [SCA-Assembly]. It is also
2105 possible for a service to set the asyncInvocation intent when using an interface which is not marked
2106 with the asyncInvocation intent. This can be useful when reusing an existing interface definition that
2107 does not contain SCA information.
- 2108 • **EJB** - The EJB intent specifies that whatever wire-level transport technology is specified the
2109 messages are able to be delivered and received via the EJB API. When EJB is present, an SCA
2110 Runtime MUST ensure that the binding used to send and receive messages supports the EJB API.
2111 [POL100006]

2112 The formal definitions of these intents are in the [Intent Definitions appendix](#).

2113 11 Conformance

2114 The XML schema available at the namespace URI, defined by this specification, is considered to be
2115 authoritative and takes precedence over the XML Schema defined in the appendix of this document.

2116 An SCA runtime MUST reject a composite file that does not conform to the sca-policy-1.1.xsd schema.
2117 [POL110001]

2118 An implementation that claims to conform to this specification MUST meet the following conditions:

- 2119 1. The implementation MUST conform to the SCA Assembly Model Specification [Assembly].
- 2120 2. SCA implementations MUST recognize the intents listed in Appendix B.1 of this specification. An
2121 implementationType / bindingType / collection of policySets that claims to implement a specific intent
2122 MUST process that intent in accord with any relevant Conformance Items in Appendix C related to
2123 the intent and the SCA Runtime options selected.
- 2124 3. With the exception of 2, the implementation MUST comply with all statements in [Appendix C](#):
2125 Conformance Items related to an SCA Runtime, notably all MUST statements have to be
2126 implemented.

A Defining the Deployed Composites Infoset

2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167

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

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

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:

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

2. For policySets by reference from another policySet element

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.

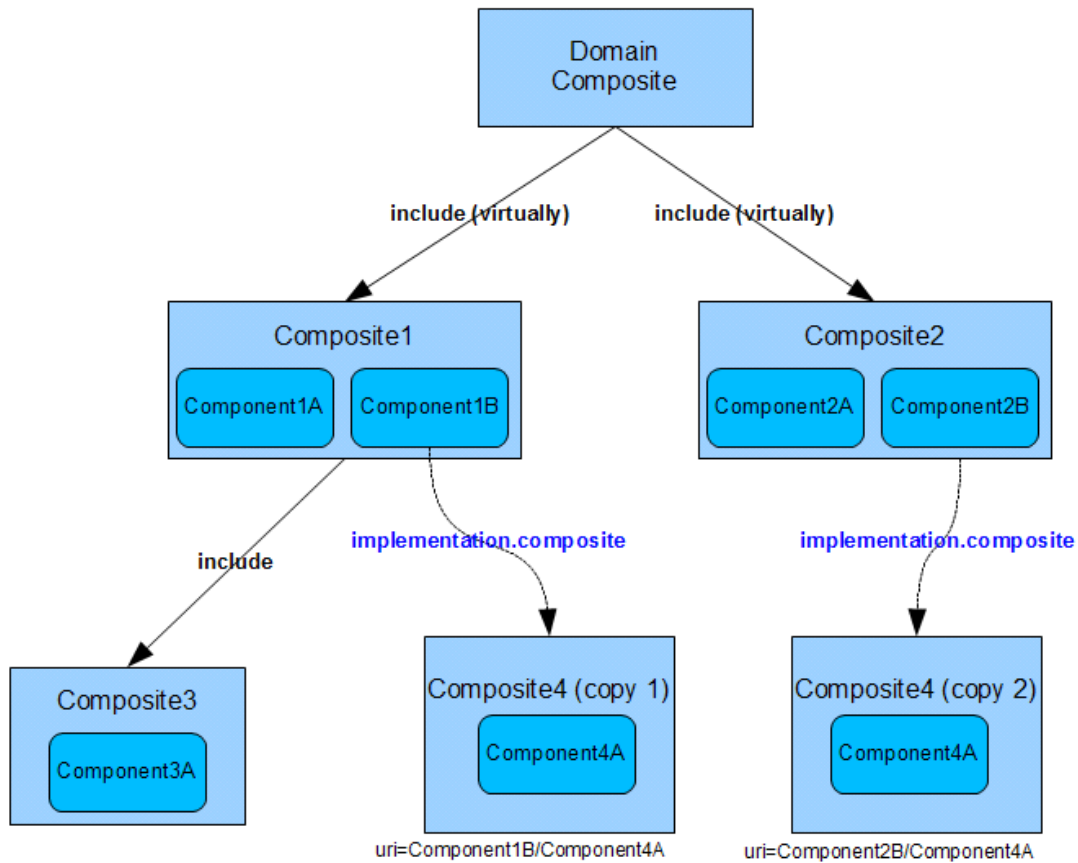
This permits the attachment of a intent/policySet to an individual operation or to an individual message within an interface, separately from the policies that apply to other operations or messages in the interface.

- 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:



2168

2169 *Figure A-1 Example Domain Composite Infoset*

2170

2171 The SCA Domain in Figure A-1 has been constructed from the composites and components shown in the
 2172 figure. Composite1 and Composite2 were deployed into the Domain as described in [SCA-Assembly].
 2173 Composite3 is included in Composite1 using the SCA include mechanism described in [SCA-Assembly].
 2174 Composite4 is used as an implementation of Components 1B and 2B. Following the deployment of all the
 2175 composites, the Domain contains:

- 2176 • 3 Composites that can be addressed as part of the Deployed Composites InfoSet; Composite1,
 2177 Composite2 and Composite4.
- 2178 • all the components shown in the diagram. Components 1A, 2A, 3A, 4A (twice) are leaf
 2179 components.

2180

2181 The following snippets show example usage of the @attachTo attribute and provide the outcome based
 2182 on the Domain in Figure A-1.

2183

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

2185 *Snippet A-1:Example attachTo all Instances of a Name*

2186

2187 attach to both instances of Component4A

2188

2189 `2. //component[URIRef("Component2B/Component4A")]`

2190 *Snippet A-2: Example attachTo a Specific Instance via a Path*

2191
2192 attach to the unique instance of Component4A when used by Component2B (Component2B is a
2193 component at the Domain level)

2194
2195 `3. //component[@name="Component3A"]/service[IntentRefs("intent1")]`

2196 *Snippet A-3:Example attachTo Instances with an intent*

2197
2198 attach to the services of Component3A which have the intent "intent1" applied

2199
2200 `4. //component/binding.ws`

2201 *Snippet A-4: Example attachTo Instances with a binding*

2202
2203 attach to the web services binding of all components with a service or reference with a Web services
2204 binding

2205
2206 `5. /composite[@name=" "]/component[@name="Component1A"]`

2207 *Snippet A-5:Example attachTo a Specific Instance via Path and Name*

2208
2209 attach to Component1A at the Domain level

2210
2211

2212 **A.1 XPath Functions for the @attachTo Attribute**

2213 This section defines utility functions that can be used in XPath expressions where otherwise it would be
2214 difficult to write the XPath expression to identify the elements concerned.

2215 This particularly applies in SCA to Interfaces and the child parts of interfaces (operations and messages).
2216 XPath Functions are defined below for the following:

- 2217 • Picking out a specific interface
- 2218 • Picking out a specific operation in an interface
- 2219 • Picking out a specific message in an operation in an interface
- 2220 • Picking out artifacts with specific intents

2221 **A.1.1 Interface Related Functions**

2222 **InterfaceRef(InterfaceName)**

2223 picks out an interface identified by InterfaceName

2224 **OperationRef(InterfaceName/OperationName)**

2225 picks out the operation OperationName in the interface InterfaceName

2226 **MessageRef(InterfaceName/OperationName/MessageName)**

2227 picks out the message MessageName in the operation OperationName in the interface
2228 InterfaceName.

2229 • "**" can be used for wildcarding of any of the names.

2230 The interface is treated as if it is a WSDL interface (for other interface types, they are treated as if
2231 mapped to WSDL using their regular mapping rules).

2232 Examples of the Interface functions:

2233

```
2234 InterfaceRef( "MyInterface" )
```

2235 Snippet A-6: Example use of InterfaceRef

2236

2237 picks out an interface with the name "MyInterface"

2238

```
2239 OperationRef( "MyInterface/MyOperation" )
```

2240 Snippet A-7: Example use of OperationRef with a Path

2241

2242 picks out the operation named "MyOperation" within the interface named "MyInterface"

2243

```
2244 OperationRef( "*/MyOperation" )
```

2245 Snippet A-8: Example use of OperationRef without a Path

2246

2247 picks out the operation named "MyOperation" from any interface

2248

```
2249 MessageRef( "MyInterface/MyOperation/MyMessage" )
```

2250 Snippet A-9: Example use of MessageRef with a Path

2251

2252 picks out the message named "MyMessage" from the operation named "MyOperation" within the interface
2253 named "MyInterface"

2254

```
2255 MessageRef( "*/*/MyMessage" )
```

2256 Snippet A-10: Example use of MessageRef with a Path with Wildcards

2257

2258 picks out the message named "MyMessage" from any operation in any interface

2259 **A.1.2 Intent Based Functions**

2260 For the following intent-based functions, it is the total set of intents which apply to the artifact which are
2261 examined by the function, including directly or externally attached intents plus intents acquired from the
2262 structural hierarchy and from the implementation hierarchy.

2263

2264 These functions cannot be used in the XPath value of the @attachTo attribute for intents

2265

2266 **IntentRefs(IntentList)**

2267 picks out an element where the intents applied match the intents specified in the IntentList:

2268

```
2269 IntentRefs( "intent1" )
```

2270 [Snippet A-11: Example use of IntentRef](#)

2271

2272 [picks out an artifact to which intent named "intent1" is attached](#)

2273

2274 `IntentRefs("intent1 intent2")`

2275 [Snippet A-12: Example use of IntentRef with Multiple intents](#)

2276

2277 [picks out an artifact to which intents named "intent1" AND "intent2" are attached](#)

2278

2279 `IntentRefs("intent1 !intent2")`

2280 [Snippet A-13: Example use of IntentRef with Not Operator](#)

2281

2282 [picks out an artifact to which intent named "intent1" is attached but NOT the intent named "intent2"](#)

2283 **A.1.3 URI Based Function**

2284 [The URIRef function is used to pick out a particular use of a nested component – ie where some Domain](#)
2285 [level component is implemented using a composite implementation, which in turn has one or more](#)
2286 [components implemented with the composite \(and so on to an arbitrary level of nesting\):](#)

2287 **URIRef(URI)**

2288 [picks out the particular use of a component identified by the structuralURI string URI.](#)

2289 [For a full description of structuralURIs, see the SCA Assembly specification \[SCA-Assembly\].](#)

2290 [Example:](#)

2291

2292 `URIRef("top_comp_name/middle_comp_name/lowest_comp_name")`

2293 [Snippet A-15: Example use of URIRef](#)

2294

2295 [picks out the particular use of a component – where component lowest_comp_name is used within the](#)
2296 [implementation of middle_comp_name within the implementation of the top-level \(Domain level\)](#)
2297 [component top_comp_name.](#)

2298

2299

AB Schemas

2300

A.1B.1 sca-policy.xsd

```
2301 <?xml version="1.0" encoding="UTF-8"?>
2302 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2303 OASIS trademark, IPR and other policies apply. -->
2304 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2305 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2306 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2307 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
2308 elementFormDefault="qualified">
2309
2310 <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
2311 <import namespace="http://www.w3.org/ns/ws-policy"
2312 schemaLocation="http://www.w3.org/2007/02/ws-policy.xsd"/>
2313
2314 <element name="intent" type="sca:Intent"/>
2315 <complexType name="Intent">
2316 <sequence>
2317 <element name="description" type="string" minOccurs="0"
2318 maxOccurs="1" />
2319 <element name="qualifier" type="sca:IntentQualifier"
2320 minOccurs="0" maxOccurs="unbounded" />
2321 <any namespace="##other" processContents="lax"
2322 minOccurs="0" maxOccurs="unbounded"/>
2323 </sequence>
2324 <attribute name="name" type="NCName" use="required"/>
2325 <attribute name="constrains" type="sca:listOfQNames"
2326 use="optional" />
2327 <attribute name="requires" type="sca:listOfQNames"
2328 use="optional" />
2329 <attribute name="excludes" type="sca:listOfQNames"
2330 use="optional" />
2331 <attribute name="mutuallyExclusive" type="boolean"
2332 use="optional" default="false"/>
2333 <attribute name="intentType"
2334 type="sca:InteractionOrImplementation"
2335 use="optional" default="interaction"/>
2336 <anyAttribute namespace="##other" processContents="lax"/>
2337 </complexType>
2338
2339 <complexType name="IntentQualifier">
2340 <sequence>
2341 <element name="description" type="string" minOccurs="0"
2342 maxOccurs="1" />
2343 </sequence>
2344 <attribute name="name" type="NCName" use="required"/>
2345 <attribute name="default" type="boolean" use="optional"
2346 default="false"/>
2347 </complexType>
2348
2349 <element name="requires">
2350 <complexType>
2351 <sequence minOccurs="0" maxOccurs="unbounded">
2352 <any namespace="##other" processContents="lax"/>
2353 </sequence>
2354 <attribute name="intents" type="sca:listOfQNames"
2355 use="required"/>
```

```

2356         <anyAttribute namespace="##other" processContents="lax" />
2357     </complexType>
2358 </element>
2359
2360     <element name="intentAttachment">
2361         <complexType>
2362             <sequence minOccurs="0" maxOccurs="unbounded">
2363                 <any namespace="##other" processContents="lax" />
2364             </sequence>
2365             <attribute name="intents" type="sca:listOfQNames"
2366                 use="required" />
2367             <attribute name="name" type="xs:string" use="required" />
2368         </complexType>
2369     </element>
2370
2371 </element>
2372
2373
2374 <element name="policySet" type="sca:PolicySet" />
2375 <complexType name="PolicySet">
2376     <choice minOccurs="0" maxOccurs="unbounded">
2377         <element name="policySetReference"
2378             type="sca:PolicySetReference" />
2379         <element name="intentMap" type="sca:IntentMap" />
2380         <any namespace="##other" processContents="lax" />
2381     </choice>
2382     <attribute name="name" type="NCName" use="required" />
2383     <attribute name="provides" type="sca:listOfQNames" />
2384     <attribute name="appliesTo" type="string" use="optional" />
2385     <attribute name="attachTo" type="string" use="optional" />
2386     <anyAttribute namespace="##other" processContents="lax" />
2387 </complexType>
2388
2389 <element name="policySetAttachment">
2390     <complexType>
2391         <sequence minOccurs="0" maxOccurs="unbounded">
2392             <any namespace="##other" processContents="lax" />
2393         </sequence>
2394         <attribute name="name" type="QName" use="required" />
2395         <anyAttribute namespace="##other" processContents="lax" />
2396     </complexType>
2397 </element>
2398
2399 <complexType name="PolicySetReference">
2400     <attribute name="name" type="QName" use="required" />
2401     <anyAttribute namespace="##other" processContents="lax" />
2402 </complexType>
2403
2404 <complexType name="IntentMap">
2405     <choice minOccurs="1" maxOccurs="unbounded">
2406         <element name="qualifier" type="sca:Qualifier" />
2407         <any namespace="##other" processContents="lax" />
2408     </choice>
2409     <attribute name="provides" type="QName" use="required" />
2410     <anyAttribute namespace="##other" processContents="lax" />
2411 </complexType>
2412
2413 <complexType name="Qualifier">
2414     <sequence minOccurs="0" maxOccurs="unbounded">

```



```
2415         <any namespace="##other" processContents="lax" />
2416     <sequence/>
2417     <attribute name="name" type="string" use="required" />
2418     <anyAttribute namespace="##other" processContents="lax" />
2419 </complexType>
2420
2421 <simpleType name="listOfNCNames">
2422     <list itemType="NCName" />
2423 </simpleType>
2424
2425 <simpleType name="InteractionOrImplementation">
2426     <restriction base="string">
2427         <enumeration value="interaction" />
2428         <enumeration value="implementation" />
2429     </restriction>
2430 </simpleType>
2431
2432 </schema>
```

2433 *Snippet A-1SCA Policy Schema*

2434 **BC XML Files**

2435 This appendix contains normative XML files that are defined by this specification.

2436 **B.1C.1 Intent Definitions**

2437 Intent definitions are contained within a Definitions file called Policy_Intents_Definitions.xml, which
2438 contain a <definitions/> element as follows:

```
2439 <?xml version="1.0" encoding="UTF-8"?>
2440 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
2441 OASIS trademark, IPR and other policies apply. -->
2442 <sca:definitions xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2443 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2444 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903">
2445
2446 <!-- Security related intents -->
2447 <sca:intent name="serverAuthentication" constrains="sca:binding"
2448 intentType="interaction">
2449 <sca:description>
2450 Communication through the binding requires that the
2451 server is authenticated by the client
2452 </sca:description>
2453 <sca:qualifier name="transport" default="true"/>
2454 <sca:qualifier name="message"/>
2455 </sca:intent>
2456
2457 <sca:intent name="clientAuthentication" constrains="sca:binding"
2458 intentType="interaction">
2459 <sca:description>
2460 Communication through the binding requires that the
2461 client is authenticated by the server
2462 </sca:description>
2463 <sca:qualifier name="transport" default="true"/>
2464 <sca:qualifier name="message"/>
2465 </sca:intent>
2466
2467 <sca:intent name="authentication"
2468 requires="sca:clientAuthentication">
2469 <sca:description>
2470 A convenience intent to help migration
2471 </sca:description>
2472 </sca:intent>
2473
2474 <sca:intent name="mutualAuthentication"
2475 requires="sca:clientAuthentication sca:serverAuthentication">
2476 <sca:description>
2477 Communication through the binding requires that the
2478 client and server to authenticate each other
2479 </sca:description>
2480 </sca:intent>
2481
2482 <sca:intent name="confidentiality" constrains="sca:binding"
2483 intentType="interaction">
2484 <sca:description>
2485 Communication through the binding prevents unauthorized
2486 users from reading the messages
2487 </sca:description>
2488 <sca:qualifier name="transport" default="true"/>
2489 <sca:qualifier name="message"/>
```

```

2490     </sca:intent>
2491
2492     <sca:intent name="integrity" constrains="sca:binding"
2493     intentType="interaction">
2494         <sca:description>
2495             Communication through the binding prevents tampering
2496             with the messages sent between the client and the service.
2497         </sca:description>
2498         <sca:qualifier name="transport" default="true" />
2499         <sca:qualifier name="message" />
2500     </sca:intent>
2501
2502     <sca:intent name="authorization" constrains="sca:implementation"
2503     intentType="implementation">
2504         <sca:description>
2505             Ensures clients are authorized to use services.
2506         </sca:description>
2507     </sca:intent>
2508
2509
2510 <!-- Reliable messaging related intents -->
2511     <sca:intent name="atLeastOnce" constrains="sca:binding"
2512     intentType="interaction">
2513         <sca:description>
2514             This intent is used to indicate that a message sent
2515             by a client is always delivered to the component.
2516         </sca:description>
2517     </sca:intent>
2518
2519     <sca:intent name="atMostOnce" constrains="sca:binding"
2520     intentType="interaction">
2521         <sca:description>
2522             This intent is used to indicate that a message that was
2523             successfully sent by a client is not delivered more than
2524             once to the component.
2525         </sca:description>
2526     </sca:intent>
2527
2528     <sca:intent name="exactlyOnce" requires="sca:atLeastOnce
2529     sca:atMostOnce"
2530     constrains="sca:binding" intentType="interaction">
2531         <sca:description>
2532             This profile intent is used to indicate that a message sent
2533             by a client is always delivered to the component. It also
2534             indicates that duplicate messages are not delivered to the
2535             component.
2536         </sca:description>
2537     </sca:intent>
2538
2539     <sca:intent name="ordered" constrains="sca:binding"
2540     intentType="interaction">
2541         <sca:description>
2542             This intent is used to indicate that all the messages are
2543             delivered to the component in the order they were sent by
2544             the client.
2545         </sca:description>
2546     </sca:intent>
2547
2548 <!-- Transaction related intents -->
2549     <sca:intent name="managedTransaction"
2550     excludes="sca:noManagedTransaction"
2551     mutuallyExclusive="true" constrains="sca:implementation"
2552     intentType="implementation">

```

```

2553         <sca:description>
2554         A managed transaction environment is necessary in order to
2555         run the component. The specific type of managed transaction
2556         needed is not constrained.
2557         </sca:description>
2558         <sca:qualifier name="global" default="true">
2559             <sca:description>
2560             For a component marked with managedTransaction.global
2561             a global transaction needs to be present before dispatching
2562             any method on the component - using any transaction
2563             propagated from the client or else beginning and completing
2564             a new transaction.
2565             </sca:description>
2566         </sca:qualifier>
2567         <sca:qualifier name="local">
2568             <sca:description>
2569             A component marked with managedTransaction.local needs to
2570             run within a local transaction containment (LTC) that
2571             is started and ended by the SCA runtime.
2572             </sca:description>
2573         </sca:qualifier>
2574     </sca:intent>

2575
2576     <sca:intent name="noManagedTransaction"
2577     excludes="sca:managedTransaction"
2578     constrains="sca:implementation" intentType="implementation">
2579         <sca:description>
2580         A component marked with noManagedTransaction needs to run without
2581         a managed transaction, under neither a global transaction nor
2582         an LTC. A transaction propagated to the hosting SCA runtime
2583         is not joined by the hosting runtime on behalf of a
2584         component marked with noManagedtransaction.
2585         </sca:description>
2586     </sca:intent>

2587
2588     <sca:intent name="transactedOneWay" excludes="sca:immediateOneWay"
2589     constrains="sca:binding" intentType="implementation">
2590         <sca:description>
2591         For a reference marked as transactedOneWay any OneWay invocation
2592         messages are transacted as part of a client global
2593         transaction.
2594         For a service marked as transactedOneWay any OneWay invocation
2595         message are received from the transport binding in a
2596         transacted fashion, under the service's global transaction.
2597         </sca:description>
2598     </sca:intent>

2599
2600     <sca:intent name="immediateOneWay" excludes="sca:transactedOneWay"
2601     constrains="sca:binding" intentType="implementation">
2602         <sca:description>
2603         For a reference indicates that any OneWay invocation messages
2604         are sent immediately regardless of any client transaction.
2605         For a service indicates that any OneWay invocation is
2606         received immediately regardless of any target service
2607         transaction.
2608         </sca:description>
2609     </sca:intent>

2610
2611     <sca:intent name="propagatesTransaction"
2612     excludes="sca:suspendsTransaction"
2613     constrains="sca:binding" intentType="interaction">
2614         <sca:description>
2615         A service marked with propagatesTransaction is dispatched

```

```

2616         under any propagated (client) transaction and the service binding
2617         needs to be capable of receiving a transaction context.
2618         A reference marked with propagatesTransaction propagates any
2619         transaction context under which the client runs when the
2620         reference is used for a request-response interaction and the
2621         binding of a reference marked with propagatesTransaction needs to
2622         be capable of propagating a transaction context.
2623         </sca:description>
2624     </sca:intent>
2625
2626     <sca:intent name="suspendsTransaction"
2627         excludes="sca:propagatesTransaction"
2628     constrains="sca:binding" intentType="interaction">
2629         <sca:description>
2630             A service marked with suspendsTransaction is not dispatched
2631             under any propagated (client) transaction.
2632             A reference marked with suspendsTransaction does not propagate
2633             any transaction context under which the client runs when the
2634             reference is used.
2635         </sca:description>
2636     </sca:intent>
2637
2638     <sca:intent name="managedSharedTransaction"
2639         requires="sca:managedTransaction.global
2640     sca:propagatesTransaction">
2641         <sca:description>
2642             Used to indicate that the component requires both the
2643             managedTransaction.global and the propagatesTransactions
2644             intents
2645         </sca:description>
2646     </sca:intent>
2647
2648     <!-- Miscellaneous intents -->
2649     <sca:intent name="asyncInvocation" constrains="sca:binding"
2650         intentType="interaction">
2651         <sca:description>
2652             Indicates that request/response operations for the
2653             interface of this wire are "long running" and must be
2654             treated as two separate message transmissions
2655         </sca:description>
2656     </sca:intent>
2657
2658     <sca:intent name="EJB" constrains="sca:binding"
2659         intentType="interaction">
2660         <sca:description>
2661             Specifies that the EJB API is needed to communicate with
2662             the service or reference.
2663         </sca:description>
2664     </sca:intent>
2665
2666     <sca:intent name="SOAP" constrains="sca:binding"
2667         intentType="interaction" mutuallyExclusive="true">
2668         <sca:description>
2669             Specifies that the SOAP messaging model is used for delivering
2670             messages.
2671         </sca:description>
2672         <sca:qualifier name="v1_1" default="true"/>
2673         <sca:qualifier name="v1_2"/>
2674     </sca:intent>
2675
2676     <sca:intent name="JMS" constrains="sca:binding"
2677         intentType="interaction">
2678         <sca:description>

```

```
2679         Requires that the messages are delivered and received via the
2680         JMS API.
2681         </sca:description>
2682     </sca:intent>
2683
2684     <sca:intent name="noListener" constrains="sca:binding"
2685     intentType="interaction">
2686         <sca:description>
2687             This intent can only be used on a reference. Indicates that the
2688             client is not able to handle new inbound connections. The binding
2689             and callback binding are configured so that any
2690             response or callback comes either through a back channel of the
2691             connection from the client to the server or by having the client
2692             poll the server for messages.
2693         </sca:description>
2694     </sca:intent>
2695
2696 </sca:definitions>
```

2697 *Snippet B-1: SCA intent Definitions*

2698 **GD Conformance**

2699 **C.1D.1 Conformance Targets**

2700 The conformance items listed in the section below apply to the following conformance targets:

- 2701
- Document artifacts (or constructs within them) that can be checked statically.
 - SCA runtimes, which we may require to exhibit certain behaviors.
- 2702

2703 **C.2D.2 Conformance Items**

2704 This section contains a list of conformance items for the SCA Policy Framework specification.

2705

Conformance ID	Description
[POL30001]	If the configured instance of a binding is in conflict with the intents and policy sets selected for that instance, the SCA runtime MUST raise an error.
[POL30002]	The QName for an intent MUST be unique amongst the set of intents in the SCA Domain.
[POL30004]	If an intent has more than one qualifier, one and only one MUST be declared as the default qualifier.
[POL30005]	The name of each qualifier MUST be unique within the intent definition.
[POL30006]	the name of a profile intent MUST NOT have a "." in it.
[POL30007]	If a profile intent is attached to an artifact, all the intents listed in its @requires attribute MUST be satisfied as described in section 4.12.
[POL30008]	When a policySet element contains a set of intentMap children, the value of the @provides attribute of each intentMap MUST correspond to an unqualified intent that is listed within the @provides attribute value of the parent policySet element.
[POL30010]	For each qualifiable intent listed as a member of the @provides attribute list of a policySet element, there MUST be no more than one corresponding intentMap element that declares the unqualified form of that intent in its @provides attribute. In other words, each intentMap within a given policySet uniquely provides for a specific intent.
[POL30011]	Following the inclusion of all policySet references, when a policySet element directly contains wsp:policyAttachment children or policies using extension elements, the set of policies specified as children MUST satisfy all the intents expressed using the @provides attribute value of the policySet element.
[POL30013]	The set of intents in the @provides attribute of a referenced policySet MUST be a subset of the set of intents in the @provides attribute of the referencing policySet.

- [POL30015] Each QName in the @requires attribute MUST be the QName of an intent in the SCA Domain.
- [POL30016] Each QName in the @excludes attribute MUST be the QName of an intent in the SCA Domain.
- [POL30017] The QName for a policySet MUST be unique amongst the set of policySets in the SCA Domain.
- [POL30018] The contents of @appliesTo MUST match the XPath 1.0 [XPATH] production *Expr*.
- [POL30019] The contents of @attachTo MUST match the XPath 1.0 production *Expr*.
- [POL30020] If a policySet specifies a qualifiable intent in the @provides attribute, and it provides an intentMap for the qualifiable intent then that intentMap MUST specify all possible qualifiers for that intent.
- [POL30021] The @provides attribute value of each intentMap that is an immediate child of a policySet MUST be included in the @provides attribute of the parent policySet.
- [POL30024] An SCA Runtime MUST include in the Domain the set of intent definitions contained in the Policy_Intent_Definitions.xml described in the appendix "Intent Definitions" of the SCA Policy specification.
- [POL30025] If only one qualifier for an intent is given it MUST be used as the default qualifier for the intent.
- [POL40001] SCA implementations supporting both Direct Attachment and External Attachment mechanisms MUST ignore policy sets applicable to any given SCA element via the Direct Attachment mechanism when there exist policy sets applicable to the same SCA element via the External Attachment mechanism
- [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] 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] Rule2: The intents declared on elements higher in the structural hierarchy of a given element MUST be applied to the element EXCEPT
- if any of the inherited intents is mutually exclusive with an intent applied on the element, then the inherited intent MUST be ignored
 - if the overall set of intents from the element itself and from its structural hierarchy contains both an unqualified version and a qualified version of the same intent, the qualified version of the intent MUST be used.
- [POL40006] If a component has any policySets attached to it (by any means), then any policySets attached to the componentType MUST be

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

	the policy language used for those policySets.
[POL40023]	The policySets at each end of a wire MUST be incompatible if they use different policy languages.
[POL40024]	Where the policy language in use for a wire is WS-Policy, strict WS-Policy intersection MUST be used to determine policy compatibility.
[POL40025]	In order for a reference to connect to a particular service, the policies of the reference MUST intersect with the policies of the service.
[POL40026]	During the deployment of an SCA policySet, the behavior of an SCA runtime MUST take ONE of the following forms: <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.
[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]	The implementationType name attribute MUST be the QName of an XSD global element definition used for implementation elements of that type.
[POL70001]	When <i>authorization</i> is present, an SCA Runtime MUST ensure that the client is authorized to use the service.
[POL70009]	When <i>confidentiality</i> is present, an SCA Runtime MUST ensure that only authorized entities can view the contents of a message.
[POL70010]	When <i>integrity</i> is present, an SCA Runtime MUST ensure that the contents of a message are not altered.
[POL70011]	When a serverAuthentication, clientAuthentication, confidentiality or integrity intent is qualified by transport, an SCA Runtime MUST delegate serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the transport layer of the communication protocol.
[POL70012]	When a serverAuthentication, clientAuthentication, confidentiality or integrity intent is qualified by message, an SCA Runtime MUST delegate serverAuthentication, clientAuthentication, confidentiality and integrity, respectively, to the message layer of the communication protocol.
[POL70013]	When <i>serverAuthentication</i> is present, an SCA runtime MUST ensure that the server is authenticated by the client.
[POL70014]	When <i>clientAuthentication</i> is present, an SCA runtime MUST ensure that the client is authenticated by the server.

- [POL80001] When *atLeastOnce* is present, an SCA Runtime MUST deliver a message to the destination service implementation, and MAY deliver duplicates of a message to the service implementation.
- [POL80002] When *atMostOnce* is present, an SCA Runtime MAY deliver a message to the destination service implementation, and MUST NOT deliver duplicates of a message to the service implementation.
- [POL80003] When *ordered* is present, an SCA Runtime MUST deliver messages sent by a single source to a single destination service implementation in the order that the messages were sent by that source.
- [POL80004] When *exactlyOnce* is present, an SCA Runtime MUST deliver a message to the destination service implementation and MUST NOT deliver duplicates of a message to the service implementation.
- [POL90003] For a component marked with `managedTransaction.global`, the SCA runtime MUST ensure that a global transaction is present before dispatching any method on the component.
- [POL90004] A component marked with `managedTransaction.local` MUST run within a local transaction containment (LTC) that is started and ended by the SCA runtime.
- [POL90006] Local transactions MUST NOT be propagated outbound across remutable interfaces.
- [POL90007] A transaction that is propagated to the hosting SCA runtime MUST NOT be joined by the hosting runtime on behalf of a component marked with `noManagedtransaction`.
- [POL90008] When a reference is marked as `transactedOneWay`, any `OneWay` invocation messages MUST be transacted as part of a client global transaction.
- [POL90009] If the client component is not configured to run under a global transaction or if the binding does not support transactional message sending, then a reference MUST NOT be marked as `transactedOneWay`.
- [POL90010] If a service is marked as `transactedOneWay`, any `OneWay` invocation message MUST be received from the transport binding in a transacted fashion, under the target service's global transaction.
- [POL90011] If the component is not configured to run under a global transaction or if the binding does not support transactional message receipt, then a service MUST NOT be marked as `transactedOneWay`.
- [POL90012] When applied to a reference indicates that any `OneWay` invocation messages MUST be sent immediately regardless of any client transaction.
- [POL90013] When applied to a service indicates that any `OneWay` invocation MUST be received immediately regardless of any target service

	transaction.
[POL90015]	A service marked with <code>propagatesTransaction</code> MUST be dispatched under any propagated (client) transaction.
[POL90016]	Use of the <i>propagatesTransaction</i> intent on a service implies that the service binding MUST be capable of receiving a transaction context.
[POL90017]	A service marked with <code>suspendsTransaction</code> MUST NOT be dispatched under any propagated (client) transaction.
[POL90019]	A service MUST NOT be marked with "propagatesTransaction" if the component is marked with "managedTransaction.local" or with "noManagedTransaction"
[POL90020]	When a reference is marked with <code>propagatesTransaction</code> , any transaction context under which the client runs MUST be propagated when the reference is used for a request-response interaction
[POL90022]	When a reference is marked with <code>suspendsTransaction</code> , any transaction context under which the client runs MUST NOT be propagated when the reference is used.
[POL90023]	A reference MUST NOT be marked with <code>propagatesTransaction</code> if component is marked with "ManagedTransaction.local" or with "noManagedTransaction"
[POL90024]	Transaction context MUST NOT be propagated on <code>OneWay</code> messages.
[POL90025]	The SCA runtime MUST ignore the <code>propagatesTransaction</code> intent for <code>OneWay</code> methods.
[POL90027]	If a <code>transactedOneWay</code> intent is combined with the <code>managedTransaction.local</code> or <code>noManagedTransaction</code> implementation intents for either a reference or a service then an error MUST be raised during deployment.
[POL100001]	When <i>SOAP</i> is present, an SCA Runtime MUST use the SOAP messaging model to deliver messages.
[POL100002]	When a <i>SOAP</i> intent is qualified with <code>1_1</code> or <code>1_2</code> , then SOAP version 1.1 or SOAP version 1.2 respectively MUST be used to deliver messages.
[POL100003]	When <i>JMS</i> is present, an SCA Runtime MUST ensure that the binding used to send and receive messages supports the JMS API.
[POL100004]	The <code>noListener</code> intent MUST only be declared on a <code>@requires</code> attribute of a reference.
[POL100005]	When <code>noListener</code> is present, an SCA Runtime MUST not establish any connection from a service to a client.
[POL100006]	When <i>EJB</i> is present, an SCA Runtime MUST ensure that the binding used to send and receive messages supports the EJB API.
[POL110001]	An SCA runtime MUST reject a composite file that does not conform to the <code>sca-policy-1.1.xsd</code> schema.

2706 Table C-1: SCA Policy Normative Statements

2707

DE Acknowledgements

2708

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

2709

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

2710

2711
2712
2713

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

2714

2715