



Service Component Architecture Assembly Model Specification Version 1.2

Committee Specification Draft 01

19 July 2011

Specification URIs

This version:

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/csd01/sca-assembly-spec-v1.2-csd01.pdf> (Authoritative)
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/csd01/sca-assembly-spec-v1.2-csd01.html>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/csd01/sca-assembly-spec-v1.2-csd01.doc>

Previous version:

N/A

Latest version:

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/sca-assembly-spec-v1.2.pdf> (Authoritative)
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/sca-assembly-spec-v1.2.html>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/sca-assembly-spec-v1.2.doc>

Technical Committee:

[OASIS Service Component Architecture / Assembly \(SCA-Assembly\) TC](#)

Chairs:

Martin Chapman (martin.chapman@oracle.com), Oracle
Mike Edwards (mike_edwards@uk.ibm.com), IBM

Editors:

Michael Beisiegel (mbgl@us.ibm.com), IBM
Anish Karmarkar (Anish.Karmarkar@oracle.com), Oracle
Sanjay Patil (sanjay.patil@sap.com), SAP
Michael Rowley (michael.rowley@activevos.com), Active Endpoints

Additional artifacts:

This prose specification is one component of a Work Product which also includes:

- XML schemas: <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2/csd01/xsd/>

Related work:

This specification replaces or supersedes:

- *Service Component Architecture Assembly Model Specification Version 1.1*. Latest version. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec-v1.1.html>
- *Service Component Architecture Assembly Model Specification Version 1.00*. March 15, 2007. http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf

This specification is related to:

- *SCA Policy Framework Version 1.1*. Latest version. <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.html>

Declared XML namespaces:

- <http://docs.oasis-open.org/ns/opencsa/sca/200912>

Abstract:

Service Component Architecture (SCA) provides a programming model for building applications and solutions based on a Service Oriented Architecture. It is based on the idea that business function is provided as a series of services, which are assembled together to create solutions that serve a particular business need. These composite applications can contain both new services created specifically for the application and also business function from existing systems and applications, reused as part of the composition. SCA provides a model both for the composition of services and for the creation of service components, including the reuse of existing application function within SCA composites. In addition, SCA also provides a model for organizing components that produce and consume events and the processing of such events.

SCA is a model that aims to encompass a wide range of technologies for service components and for the access methods which are used to connect them. For components, this includes not only different programming languages, but also frameworks and environments commonly used with those languages. For access methods, SCA compositions allow for the use of various communication and service access technologies that are in common use, including, for example, Web services, Messaging systems, and Remote Procedure Call (RPC).

The SCA Assembly Model consists of a series of artifacts which define the configuration of an SCA Domain in terms of composites which contain assemblies of service components and the connections and related artifacts which describe how they are linked together.

This document describes the SCA Assembly Model, which covers

- A model for the assembly of services, both tightly coupled and loosely coupled
- A model for applying infrastructure capabilities to services and to service interactions, including Security and Transactions
- A model for event processing and Pub/Sub -- a particular style of organizing components that produce and consume events in which the producing components are decoupled from the consuming components

Status:

This document was last revised or approved by the OASIS Service Component Architecture / Assembly (SCA-Assembly) TC on the above date. The level of approval is also listed above. Check the "Latest 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-assembly/>.

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-assembly/ipr.php>).

Citation format:

When referencing this specification the following citation format should be used:

[SCA-Assembly-v1.2]

Service Component Architecture Assembly Model Specification Version 1.2. 19 July 2011. OASIS Committee Specification Draft 01. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-spec/v1.2-csd01/sca-assembly-spec-v1.2-csd01.html>.

Notices

Copyright © OASIS Open 2011. 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 name "OASIS" is a trademark 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.....	8
1.1	Terminology.....	8
1.2	Normative References.....	8
1.3	Non-Normative References.....	10
1.4	Naming Conventions.....	10
1.5	Testcases.....	11
1.6	Changes in Version 1.2.....	11
2	Overview.....	12
2.1	Event Processing and Pub/Sub Overview.....	13
2.1.1	Terminology.....	14
2.1.2	Connections from Producers to Consumers.....	14
2.1.3	Declaration of Event Types on Producers and Consumers.....	15
2.2	Diagram used to Represent SCA Artifacts.....	16
2.3	Pub/Sub Examples.....	18
2.3.1	Multiple Producers linked to multiple Consumers via a Channel - within a Composite.....	18
2.3.2	Producers linked to Consumers via Domain Channels.....	19
2.3.3	Composite with Promotion of Producers and Consumers.....	21
3	Implementation and ComponentType.....	24
3.1	Component Type.....	24
3.1.1	Service.....	25
3.1.2	Reference.....	26
3.1.3	Consumer.....	28
3.1.4	Producer.....	29
3.1.5	Property.....	30
3.1.6	Implementation.....	32
3.2	Example ComponentType.....	32
3.3	Example Implementation.....	33
4	Component.....	36
4.1	Implementation.....	37
4.2	Service.....	38
4.3	Reference.....	40
4.3.1	Specifying the Target Service(s) for a Reference.....	42
4.4	Consumer.....	43
4.5	Producer.....	44
4.6	Property.....	45
4.6.1	Property Type Compatibility.....	49
4.6.2	Property Value File Format.....	49
4.7	Example Component.....	49
5	Composite.....	53
5.1	Service.....	55
5.1.1	Service Examples.....	56
5.2	Reference.....	58
5.2.1	Example Reference.....	60

5.3	Property	62
5.3.1	Property Examples	63
5.4	Wire.....	65
5.4.1	Wire Examples	67
5.4.2	Autowire.....	68
5.4.3	Autowire Examples.....	69
5.5	Consumer	72
5.6	Producer	73
5.7	Channels.....	74
5.8	Scopes of Channels.....	75
5.9	The Default Domain Channel	75
5.10	The URI of a Channel	75
5.11	Using Composites as Component Implementations	75
5.11.1	Component Type of a Composite used as a Component Implementation	76
5.11.2	Example of Composite used as a Component Implementation	79
5.12	Using Composites through Inclusion	80
5.12.1	Included Composite Examples.....	81
5.13	Composites which Contain Component Implementations of Multiple Types	83
5.14	Structural URI of Components.....	83
6	Interface.....	85
6.1	Local and Remotable Interfaces	86
6.2	Interface Compatibility	86
6.2.1	Compatible Interfaces	87
6.2.2	Compatible Subset	87
6.2.3	Compatible Superset.....	88
6.3	Bidirectional Interfaces	89
6.4	Long-running Request-Response Operations	90
6.4.1	Background	90
6.4.2	Definition of "long-running"	90
6.4.3	The asyncInvocation Intent	90
6.4.4	Requirements on Bindings	90
6.4.5	Implementation Type Support	90
6.5	SCA-Specific Aspects for WSDL Interfaces	91
6.6	WSDL Interface Type	92
6.6.1	Example of interface.wsdl	92
7	Binding.....	94
7.1	Messages containing Data not defined in the Service Interface	97
7.2	WireFormat	97
7.3	OperationSelector	97
7.4	Form of the URI of a Deployed Binding.....	98
7.4.1	Non-hierarchical URIs	98
7.4.2	Determining the URI scheme of a deployed binding.....	98
7.5	SCA Binding	98
7.5.1	Example SCA Binding	100
7.6	Web Service Binding	100

7.7 JMS Binding.....	101
8 Representation of Events and Event Types	102
8.1 Event Type and Metadata.....	102
8.2 Event Type Definition Language Used by SCA.....	102
8.3 Events with No Event Type.....	102
9 Filters: Selecting Subsets of Events.....	104
9.1 Form of Explicit Filter Elements.....	104
9.2 Event Type Filters	105
9.2.1 Use of <eventType> on a Producer	105
9.2.2 Event Type Filter Examples	106
9.3 Business Data Filters.....	107
9.3.1 XPATH 1.0 Dialect	107
10 SCA Definitions	108
11 Extension Model.....	109
11.1 Defining an Interface Type.....	109
11.2 Defining an Implementation Type.....	110
11.3 Defining a Binding Type.....	112
11.4 Defining an Import Type	114
11.5 Defining an Export Type	116
12 Packaging and Deployment	118
12.1 Domains.....	118
12.2 Contributions.....	118
12.2.1 SCA Artifact Resolution.....	119
12.2.2 SCA Contribution Metadata Document	121
12.2.3 Contribution Packaging using ZIP	123
12.3 States of Artifacts in the Domain	123
12.4 Installed Contribution	124
12.4.1 Installed Artifact URIs.....	124
12.5 Operations for Contributions.....	124
12.5.1 install Contribution & update Contribution.....	124
12.5.2 add Deployment Composite & update Deployment Composite.....	125
12.5.3 remove Contribution	125
12.6 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts	125
12.7 Domain-Level Composite	126
12.7.1 add To Domain-Level Composite.....	126
12.7.2 remove From Domain-Level Composite	126
12.7.3 get Domain-Level Composite.....	126
12.7.4 get QName Definition	126
12.8 Dynamic Behaviour of Wires in the SCA Domain	127
12.9 Dynamic Behaviour of Component Property Values	127
13 SCA Runtime Considerations.....	128
13.1 Error Handling.....	128
13.1.1 Errors which can be Detected at Deployment Time.....	128
13.1.2 Errors which are Detected at Runtime	128
14 Conformance	129

14.1 SCA Documents	129
14.2 SCA Runtime	129
14.2.1 Optional Items	130
Appendix A. XML Schemas	131
A.1 sca.xsd	131
A.2 sca-core.xsd	131
A.3 sca-binding-sca.xsd.....	140
A.4 sca-interface-java.xsd	141
A.5 sca-interface-wsdl.xsd.....	141
A.6 sca-implementation-java.xsd.....	141
A.7 sca-implementation-composite.xsd.....	141
A.8 sca-binding-webservice.xsd	142
A.9 sca-binding-jms.xsd.....	142
A.10 sca-policy.xsd	142
A.11 sca-contribution.xsd	142
A.12 sca-definitions.xsd.....	143
Appendix B. SCA Concepts.....	145
B.1 Binding.....	145
B.2 Component.....	145
B.3 Service.....	145
B.3.1 Remotable Service.....	145
B.3.2 Local Service.....	146
B.4 Reference	146
B.5 Implementation	146
B.6 Interface.....	146
B.7 Composite	147
B.8 Composite inclusion	147
B.9 Property	147
B.10 Domain	147
B.11 Wire	147
B.12 SCA Runtime.....	147
B.13 Channel	148
B.14 Consumer.....	148
B.15 Producer	148
B.16 Filters.....	148
B.17 Event Types	148
Appendix C. Conformance Items.....	149
C.1 Non-mandatory Items.....	160
Appendix D. Acknowledgements	162
Appendix E. Non-Normative Text.....	164
Appendix F. Revision History	165

1 Introduction

This document describes the **SCA Assembly Model, which** covers

- A model for the assembly of services, both tightly coupled and loosely coupled
- A model for applying infrastructure capabilities to services and to service interactions, including Security and Transactions
- A model for event processing and pub/sub -- a particular style of organizing components that produce and consume events in which the producing components are decoupled from the consuming components

The document starts with a short overview of the SCA Assembly Model.

The next part of the document describes the core elements of SCA, SCA components and SCA composites.

The final part of the document defines how the SCA assembly model can be extended.

This specification is defined in terms of Infoset and not in terms of XML 1.0, even though the specification uses XML 1.0 terminology. A mapping from XML to infoset is trivial and it is suggested that this is used for any non-XML serializations.

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 Normative References

[RFC2119]

S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*,
IETF RFC 2119, March 1997.

<http://www.ietf.org/rfc/rfc2119.txt>

[SCA-Java]

OASIS Committee Draft 03, "SCA POJO Component Implementation Specification Version 1.1",
November 2010

<http://docs.oasis-open.org/opencsa/sca-j/sca-javaci-1.1-spec-csprd03.pdf>

[SCA-Common-Java]

OASIS Committee Draft 05, "SCA Java Common Annotations and APIs Specification Version 1.1",
November 2010

<http://docs.oasis-open.org/opencsa/sca-j/sca-javacaa-1.1-spec-csd05.pdf>

[SCA BPEL]

OASIS Committee Draft 02, "SCA WS-BPEL Client and Implementation Specification Version 1.1",
March 2009

<http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec-cd02.pdf2.pdf>

[WSDL-11]

42 WSDL Specification version 1.1

43 <http://www.w3.org/TR/wsd>

44

45 **[SCA-WSBINDING]**

46 OASIS Committee Draft 04, "*SCA Web Services Binding Specification Version 1.1*", May 2010

47 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd04.pdf>

48

49 **[SCA-POLICY]**

50 OASIS Committee Draft 04, "*SCA Policy Framework Specification Version 1.1*", September 2010

51 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.pdf>

52

53 **[SCA-JMSBINDING]**

54 OASIS Committee Draft 05, "*SCA JMS Binding Specification Version 1.1 Version 1.1*", November
55 2010

56 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-jmsbinding-1.1-spec-csprd03.pdf>

57

58 **[SCA-CPP-Client]**

59 OASIS Committee Draft 06, "*SCA Client and Implementation for C++ Specification Version 1.1*",
60 October 2010

61 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd06.pdf>

62

63 **[SCA-C-Client]**

64 OASIS Committee Draft 06, "*SCA Client and Implementation for C Specification Version 1.1*",
65 October 2010

66 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd06.pdf>

67

68 **[ZIP-FORMAT]**

69 ZIP Format Definition

70 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>

71

72 **[XML-INFOSET]**

73 Infoset Specification

74 <http://www.w3.org/TR/xml-infoset/>

75

76 **[WSDL11_Identifiers]**

77 WSDL 1.1 Element Identifiers

78 <http://www.w3.org/TR/wsd11elementidentifiers/>

79

80 **[SCA-TSA]**

81 OASIS Committee Draft 01, "*Test Suite Adaptation for SCA Assembly Model Version 1.1*
82 *Specification*", July 2010

83 <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-testsuite-adaptation-cd01.pdf>

84

85 **[SCA-IMPLTYPDOC]**

86 OASIS Committee Draft 01, "*Implementation Type Documentation Requirements for SCA Assembly*
87 *Model Version 1.1 Specification*", July 2010
88 [http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation-
90 cd01.pdf](http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-impl-type-documentation-
89 cd01.pdf)

91 **[SCA-ASSEMBLY-11]**

92 OASIS Committee Specification Draft 08, "*Service Component Architecture Assembly Model*
93 *Specification Version 1.1*", June 2011
94 <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-csd08.pdf>

95 **[WS-ED]**

96 W3C Candidate Recommendation, "Web Services Event Descriptions (WS-EventDescriptions)", April
97 2011
98 <http://www.w3.org/TR/2011/CR-ws-event-descriptions-20110428>

100 **1.3 Non-Normative References**

101 **[SDO]**

102 OASIS Committee Draft 02, "*Service Data Objects Specification Version 3.0*", November 2009
103 <http://www.oasis-open.org/committees/download.php/35313/sdo-3.0-cd02.zip>

104 **[JAX-WS]**

105 JAX-WS Specification
106 <http://jcp.org/en/jsr/detail?id=224>

107 **[WSI-BP]**

108 WS-I Basic Profile
109 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>

110 **[WSI-BSP]**

111 WS-I Basic Security Profile
112 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>

113 **[WS-BPEL]**

114 OASIS Standard, "*Web Services Business Process Execution Language Version 2.0*", April 2007
115 <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>

116 **[SCA-ASSEMBLY-TC]**

117 OASIS Committee Draft, "TestCases for the SCA Assembly Model Version 1.1 Specification", August
118 2010.
119 <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-testcases.pdf>

120 **1.4 Naming Conventions**

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

- 129 • For the names of elements and the names of attributes within XSD files, the names follow the
130 CamelCase convention, with all names starting with a lower case letter.
131 e.g. <element name="componentType" type="sca:ComponentType"/>
- 132 • For the names of types within XSD files, the names follow the CamelCase convention with all names
133 starting with an upper case letter.
134 eg. <complexType name="ComponentService">
- 135 For the names of intents, the names follow the CamelCase convention, with all names starting with a
136 lower case letter, EXCEPT for cases where the intent represents an established acronym, in which case
137 the entire name is in upper case.
138 An example of an intent which is an acronym is the "SOAP" intent.

139 1.5 Testcases

140 The TestCases for the SCA Assembly Model Version 1.1 Specification [**SCA-ASSEMBLY-TC**] defines
141 the TestCases for the SCA Assembly specification. The TestCases represent a series of tests that SCA
142 runtimes are expected to pass in order to claim conformance to the requirements of the SCA Assembly
143 specification.

144 1.6 Changes in Version 1.2

145 This version applies all the errata against version 1.1 [**SCA-ASSEMBLY-11**], and adds support for **Event**
146 **Processing** and **Pub/Sub** in the SCA Assembly Model, which deals with:

- 147 • **Event Processing**, which is computing that performs operations on events, including creating,
148 reading, transforming, and deleting events or event objects/representations. Event Processing
149 components interact by creating event messages, which are then distributed to other Event
150 Processing components. An Event Processing component can, in addition, interact with other
151 SCA components using SCA's regular service invocation mechanisms.
- 152 • **Publication** and **Subscription** (often shortened to **Pub/Sub**), which is a particular style of
153 organizing the components which produce and consume events in which the producing
154 components are decoupled from the consuming components. Components that are interested in
155 consuming events specify their interest through a subscription rather than an interface. The same
156 event may be received by multiple subscribers.

157 2 Overview

158 Service Component Architecture (SCA) provides a programming model for building applications and
159 solutions based on a Service Oriented Architecture. It is based on the idea that business function is
160 provided as a series of services, which are assembled together to create solutions that serve a particular
161 business need. These composite applications can contain both new services created specifically for the
162 application and also business function from existing systems and applications, reused as part of the
163 composition. SCA provides a model both for the composition of services and for the creation of service
164 components, including the reuse of existing application function within SCA composites. In addition, SCA
165 also provides a model for organizing components that produce and consume events and the processing
166 of such events.

167 SCA is a model that aims to encompass a wide range of technologies for service components and for the
168 access methods which are used to connect them. For components, this includes not only different
169 programming languages, but also frameworks and environments commonly used with those languages.
170 For access methods, SCA compositions allow for the use of various communication and service access
171 technologies that are in common use, including, for example, Web services, Messaging systems and
172 Remote Procedure Call (RPC).

173 The SCA **Assembly Model** consists of a series of artifacts which define the configuration of an SCA
174 Domain in terms of composites which contain assemblies of service components and the connections
175 and related artifacts which describe how they are linked together.

176 One basic artifact of SCA is the **component**, which is the unit of construction for SCA. A component
177 consists of a configured instance of an implementation, where an implementation is the piece of program
178 code providing business functions. The business function is offered for use by other components as
179 **services**. Implementations can depend on services provided by other components – these dependencies
180 are called **references**. Implementations can have settable **properties**, which are data values which
181 influence the operation of the business function. To support the Pub/Sub style of interaction, an
182 implementation can also have **consumers** and **producers**. Consumers and producers can specify the
183 kind of events that they are interested in consuming or producing, respectively. The component
184 **configures** the implementation by providing values for the properties, by wiring the references to services
185 provided by other components, and by connecting the producers/consumers to channels.

186 SCA allows for a wide variety of implementation technologies, including "traditional" programming
187 languages such as Java, C++, and BPEL, but also scripting languages such as PHP and JavaScript and
188 declarative languages such as XQuery and SQL.

189 SCA describes the content and linkage of an application in assemblies called **composites**. Composites
190 can contain components, services, references, consumers, producers, channels, and property
191 declarations, plus the wiring that describes the connections between these elements. Composites can
192 group and link components built from different implementation technologies, allowing appropriate
193 technologies to be used for each business task. In turn, composites can be used as complete component
194 implementations: providing services, producers and depending on references, consumers, and with
195 settable property values. Such composite implementations can be used in components within other
196 composites, allowing for a hierarchical construction of business solutions, where high-level services are
197 implemented internally by sets of lower-level services. The content of composites can also be used as
198 groupings of elements which are contributed by inclusion into higher-level compositions.

199 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services
200 providing an area of business functionality that is controlled by a single organization. As an example, for
201 the accounts department in a business, the SCA Domain might cover all financial related function, and it
202 might contain a series of composites dealing with specific areas of accounting, with one for customer
203 accounts, another dealing with accounts payable. To help build and configure the SCA Domain,
204 composites can be used to group and configure related artifacts.

205 SCA defines an XML file format for its artifacts. These XML files define the portable representation of the
206 SCA artifacts. An SCA runtime might have other representations of the artifacts represented by these
207 XML files. In particular, component implementations in some programming languages might have

208 attributes or properties or annotations which can specify some of the elements of the SCA Assembly
209 model. The XML files define a static format for the configuration of an SCA Domain. An SCA runtime
210 might also allow for the configuration of the Domain to be modified dynamically.

211 **2.1 Event Processing and Pub/Sub Overview**

212 In addition to the **Service-Reference Model** described above, the SCA **Assembly Model** also supports
213 the **Pub-Sub Model** which can be used by components to communicate with each other. With **service**
214 **invocation**, one component, the client, invokes an operation on a service reference, which causes that
215 operation to be invoked on a second component, the service provider. The significant characteristics of
216 service invocation are that:

- 217 • Each invocation by the client on a reference operation causes one invocation of the operation on
218 one service provider
- 219 • The operation itself typically has some implied semantics – the client is expecting some specific
220 task to be performed by the service provider, possibly involving specific data being returned by
221 the provider
- 222 • A particular operation is typically grouped with a set of other related operations, as defined by an
223 interface, which as a whole make up the service offered by the provider. The need to implement
224 the interface as a whole is a requirement for the code implementing the components. There is
225 also a requirement that the complete set of operations declared on a reference is supplied by the
226 service provider.
- 227 • The provider may respond to the operation invocation with zero or more response messages.
228 These messages may be returned synchronously or asynchronously, but they are returned to the
229 client component that made the original invocation. That they are returned is part of the service
230 contract between the client and the provider

231 In contrast, in **event processing** applications one component, the producer, creates a message called an
232 event, which is sent out and can be received by any number of other components, called consumers.
233 The significant characteristics of this mechanism are that:

- 234 • Each event created by a producer may be received by zero, one or many consumer components.
235 The producer is unaware of the specific consumers or the number of consumers that receive any
236 event.
- 237 • The consumer cannot respond to an event received – there is in principle no knowledge of the
238 producer component and no route provided by which a response message could be sent to it.
239 The component receiving an event can in turn send out events (or invoke services), but there is
240 no implication that the original producer component will receive any of those events.
- 241 • What is done when a consumer receives an event has no implied semantics – the consumer can
242 do what it likes with the event and there are no semantics agreed with the producer
- 243 • There is no requirement that a consumer consumes all of the event types that can be produced
244 by a given producer. Neither is there a requirement that a producer produces all of the event
245 types that can be consumed by a consumer. Unlike services, there is no matching of an interface
246 on the producer to an interface on the consumer.
- 247 • There is also no direct relationship between event types and the implementation operations or
248 methods used to produce or consume them – e.g., a single operation can handle one event type
249 or many event types, as desired by the writer of the implementation code.
- 250 • A consumer can filter which events it is prepared to accept – there is no guarantee that it actually
251 does anything with a given event. The filtering may be on the event type or on the business data
252 within the event or on other metadata associated with the event.

253 Service operations which are **one-way** are close in nature to the sending and receiving of events, but it is
254 notable that for one-way service operations the client component must be aware of the number of target
255 services (multiplicity 0..n or 1..n specified) and the client has to call the operation once for each target.
256 For an event, the producer component simply sends a single event once through its producer – the event

257 is sent to all the consumer components that have expressed interest in that event and are connected
258 (including none), without the producer component being aware of the number or of the recipients.

259 Event processing involves more loosely-coupled method of combining components into an application
260 than using service interfaces. Events place fewer requirements on the components at each end of the
261 communication. Effectively, in event processing it is only the event types that are shared between the
262 producers and the consumers.

263 Loose coupling is further emphasized through the use of **Pub/Sub**. With Pub/Sub, producers are not
264 connected directly to any consumers – instead, a group of zero or more producers is connected with a
265 group of zero or more consumers through a logical intermediary, called a **Channel**. The producers
266 publish events to the channel and the consumers receive events from the channel. The actual origin of
267 an event received by a consumer can be any of the producers – without the consumer being directly
268 connected to any of the producers.

269 In SCA event processing, component implementations may have zero or more **producers** and zero or
270 more **consumers**. The producers and consumers can indicate which event type(s) they deal with. SCA
271 components configure implementations to express where producer events are published to and where
272 consumer events are subscribed from.

273 2.1.1 Terminology

- 274 • *event* – a message sent to zero or more parties that contains information about a situation that
275 has occurred
- 276 • *producer* - entity that creates events
- 277 • *consumer* - entity that receives events
- 278 • *subscription* - records a consumer's interest in receiving specific kinds of events from some
279 location
- 280 • *source* – the place from which a consumer receives events
- 281 • *target* – the place to which a producer sends events
- 282 • *publication* – the sending of an event from a producer to some targets
- 283 • *event type* – every event instance can have an associated event type. Each event type is
284 identified by a unique QName and has an associated shape that can be described using XML
285 Schema global element declaration, and optionally, constraints on the event instance
- 286 • *channel* – a mechanism to connect a set of producers with a set of consumers
- 287 • *filter* - a mechanism for refining the set of events received by a consumer. A filter may operate on
288 business data within the event itself, or on metadata about the event.

289 2.1.2 Connections from Producers to Consumers

290 In SCA, events flow from producers to consumers along logical routes that are defined by the
291 configuration of composites and the components and channels they contain. In particular, components
292 configure producers by declaring targets for the events created by the producer. Components configure
293 consumers by specifying sources for the events received by the consumer and specifying the kind of
294 events that are of interest.

295 2.1.2.1 Linking Producers to Consumers

296 Event producers can be linked to event consumers via a third party called a channel, where producers
297 are configured with the channel as a target and consumers are configured with the channel as a source.
298 Using this mechanism, producers and consumers are not directly connected. It is also possible for the
299 producer(s) to connect to a Domain channel (see the Section on Scopes of Channels) at a different time
300 than when the consumer(s) connect to the same channel.

301 A producer declares where the messages it produces are sent through a list of one or more target URIs in
302 its **@target** attribute. The form of the target URIs include:

- 303 • The URI of a channel in the same composite as the producer, in the form ***channelName***
- 304 • The URI of of a channel at the Domain level in the form ***/channelName***

305 A consumer declares the sources for the messages it receives through a list of one or more source URIs
306 in its **@source** attribute. The form of the source URIs include:

- 307 • The URI of a channel in the same composite in the form ***channelName***
- 308 • The URI of a channel at the Domain level in the form ***/channelName***

309 2.1.2.2 Producers, Consumers, and Composites

310 When an assembler creates a composite that is intended for use as an implementation, the assembler
311 can decide whether consumers and producers within the composite are visible outside the scope of the
312 composite or not.

313 The assembler can also decide on what level of control is given to the higher level component that is
314 using the composite as its implementation – i.e., the assembler can decide what appears in the
315 component type of the composite, which can then be configured by the higher level component.

316 One technique which enables component producers to send events outside the composite and for
317 component consumers to receive events from outside the composite is to configure producers and/or
318 consumers of components inside the composite to use domain channels – that is, channels at the
319 Domain level. See the Section Scopes of Channels for more details on domain channels. This approach
320 "hard wires" the producers and consumers within the composite - the higher level component cannot
321 reconfigure the sources and targets.

322 An alternative technique for configuring a component producer element is to declare a ***composite***
323 ***producer*** element which ***promotes*** the component producer. Similarly a component consumer can be
324 configured by declaring a ***composite consumer*** element which ***promotes*** the component consumer.
325 When producers and consumers are promoted in this way, and the composite is used as the
326 implementation of some higher level component, the assembler of the higher level composite can control
327 where the events flow to and from, through configuration of the higher level component. This technique
328 promotes reuse of the lower level composite in different contexts.

329 Each producer and consumer can be connected to zero or more channels. If a producer is not connected
330 then any events it produces are discarded and are not received by any consumer. If a consumer is not
331 connected, then it never receives any events.

332 A Composite can contain one or more Channels. Events can be sent to a channel by producers within
333 the composite, if connected, and events may be received from a channel by consumers within the
334 composite, if connected.

335 2.1.3 Declaration of Event Types on Producers and Consumers

336 Producers can declare the set of event types that they produce in the implementation, component
337 producer or composite producer. Consumers can declare the set of event types that they handle in the
338 implementation, component consumer or composite consumer. Similarly, channels can declare the set of
339 event types that they handle in the composite configuration. It is also possible to declare that a producer,
340 a consumer, or a channel handles any event type.

341 The value of declaring the events that are produced and consumed by components and channels is that:

- 342 • When the event types produced and consumed are explicitly declared, it may be possible to avoid the
343 need for runtime event filters on the consumers, providing an optimized path for the handling of the
344 events.
- 345 • Because the channel, producer and consumer declarations can include a list of event types, it is
346 possible to report an error or a warning when a producer or a consumer is connected to a channel,
347 where there is no chance that the produced events will be accepted by the channel or the consumer
348 will ever get any event.

349

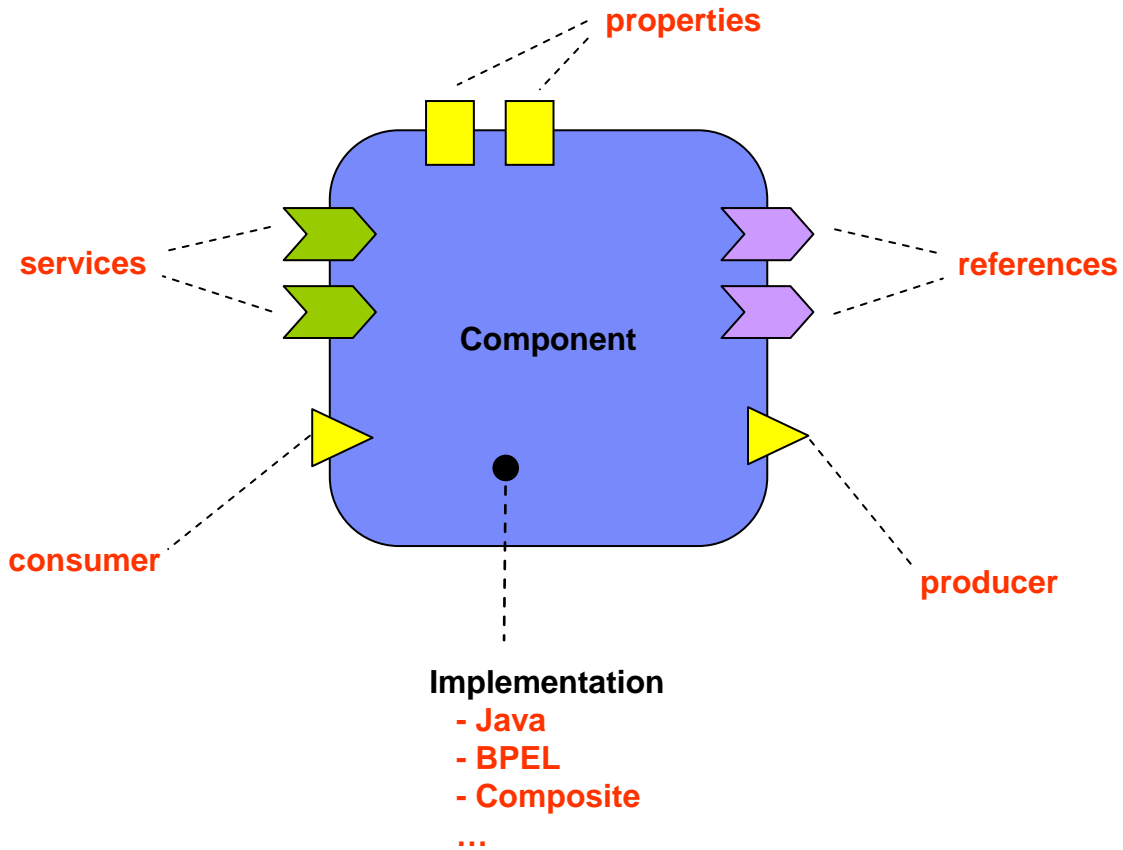
350 The following always apply:

- 351 • A producer SHOULD only produce event type it has declared. [ASM20101]
- 352 • An SCA Runtime MAY reject events of a type from a producer which does not declare that it
- 353 produces events of that type. [ASM20102]

354 2.2 Diagram used to Represent SCA Artifacts

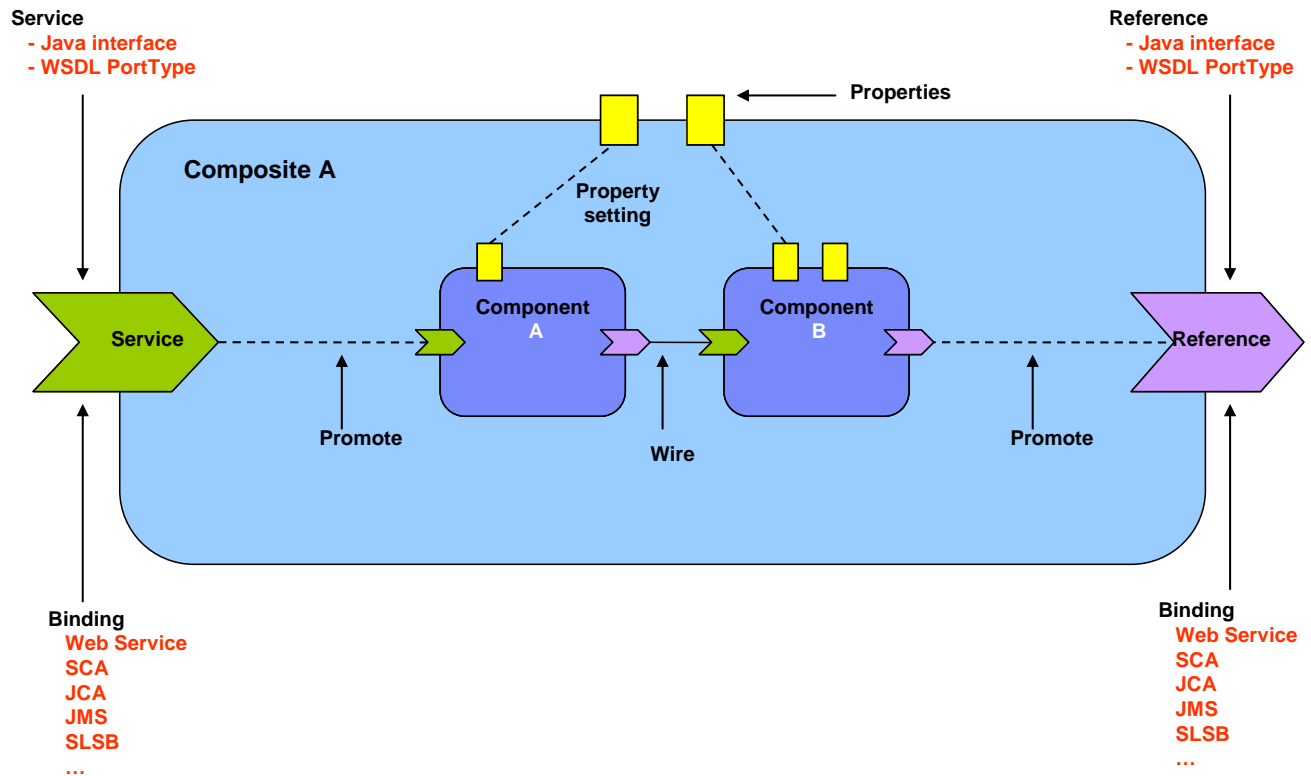
355 This document introduces diagrams to represent the various SCA artifacts, as a way of visualizing the
 356 relationships between the artifacts in a particular assembly. These diagrams are used in this document to
 357 accompany and illuminate the examples of SCA artifacts and do not represent any formal graphical
 358 notation for SCA.

359 Figure 2-1 illustrates some of the features of an SCA component:



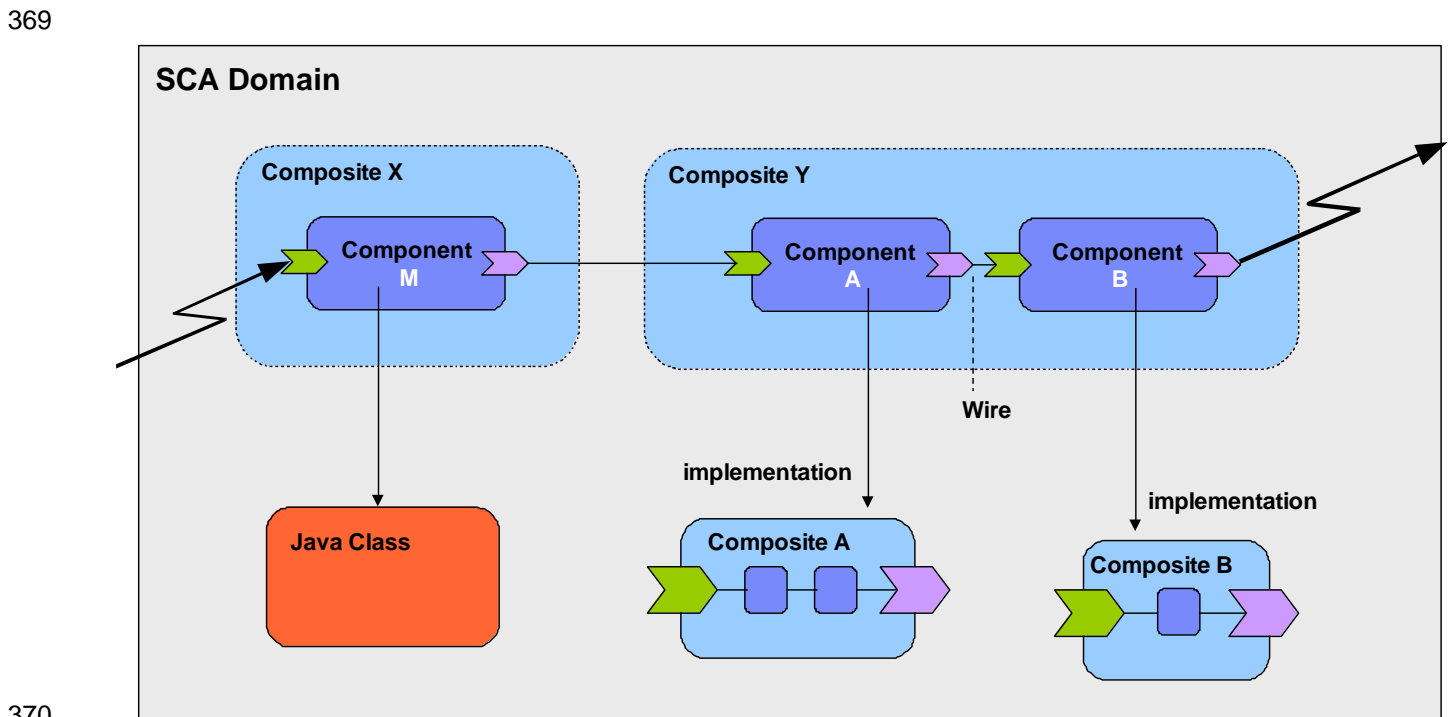
360
 361 *Figure 2-1: SCA Component Diagram*

362 Figure 2-2 illustrates some of the features of a composite assembled using a set of components:
 363



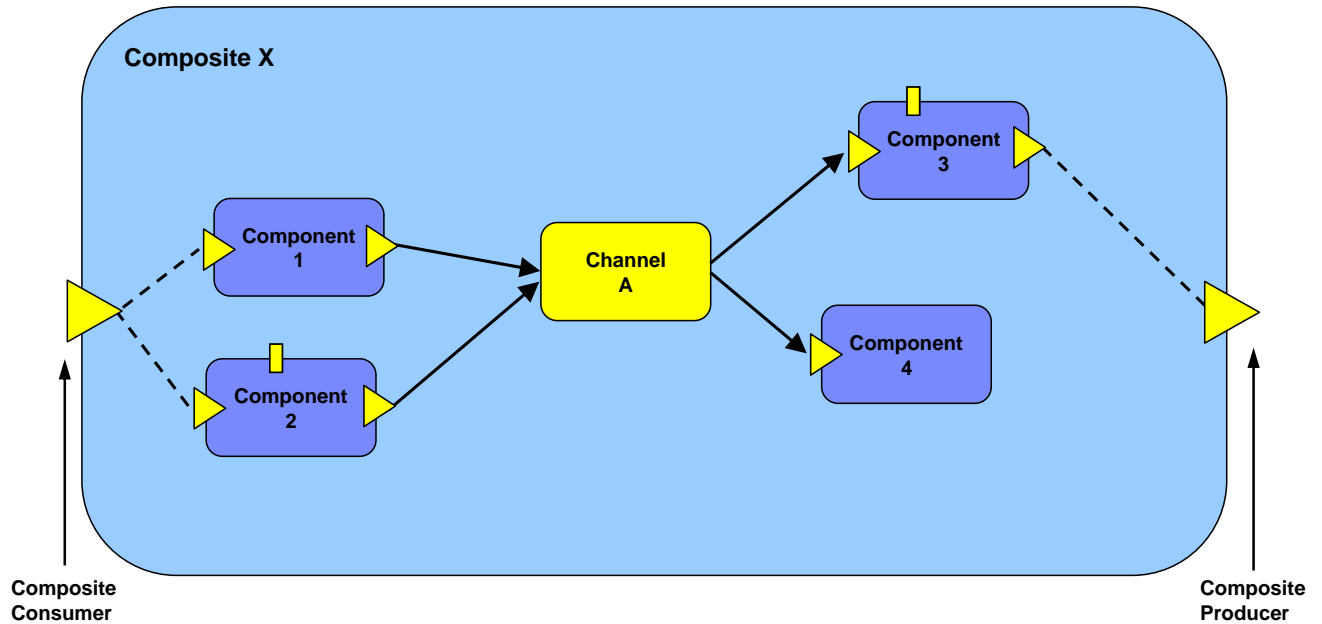
364
365 Figure 2-2: SCA Composite Diagram

366
367 Figure 2-3 illustrates an SCA Domain assembled from a series of high-level composites, some of which
368 are in turn implemented by lower-level composites:



370
371 Figure 2-3: SCA Domain Diagram

372 Figure 2-4 shows and SCA composite involving components that communicate using Pub/Sub:



374
375

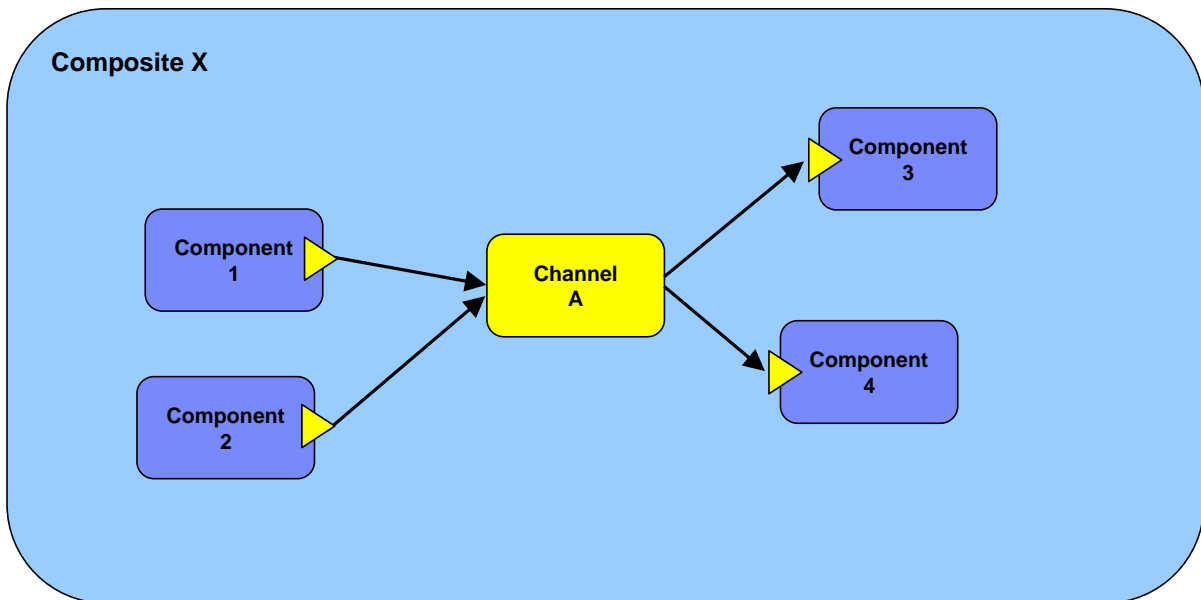
376 *Figure 2-4: SCA Composite Diagram with Pub/Sub*

377 2.3 Pub/Sub Examples

378 2.3.1 Multiple Producers linked to multiple Consumers via a Channel - 379 within a Composite

380 This example is of multiple component producers, which send events to multiple component consumers
381 via a Channel, which decouples the producers from the consumers. The assembly is represented by the
382 diagram in Figure 2-5:

383



384
385

386 *Figure 2-5: Producers Linked to Consumers Via a Local Channel*

387

388 The corresponding XML for this example follows:

389

```
390 <composite name="CompositeX"
391   xmlns="http://www.oesa.org/xmlns/sca/1.0"
392   targetNamespace="http://example.org/example1">
393
394   <component name="Component1">
395     <implementation.java class="org.example.Component1Impl"/>
396     <producer name="Foo_Events" target="ChannelA"/>
397   </component>
398
399   <component name="Component2">
400     <implementation.java class="org.example.Component2Impl"/>
401     <producer name="Foo_Events" target="ChannelA"/>
402   </component>
403
404   <component name="Component3">
405     <implementation.java class="org.example.Component3Impl"/>
406     <consumer name="Foo_Handling" source="ChannelA"/>
407   </component>
408
409   <component name="Component4">
410     <implementation.java class="org.example.Component4Impl"/>
411     <consumer name="Foo_Handling" source="ChannelA"/>
412   </component>
413
414   <channel name="ChannelA"/>
415
416 </composite>
```

417

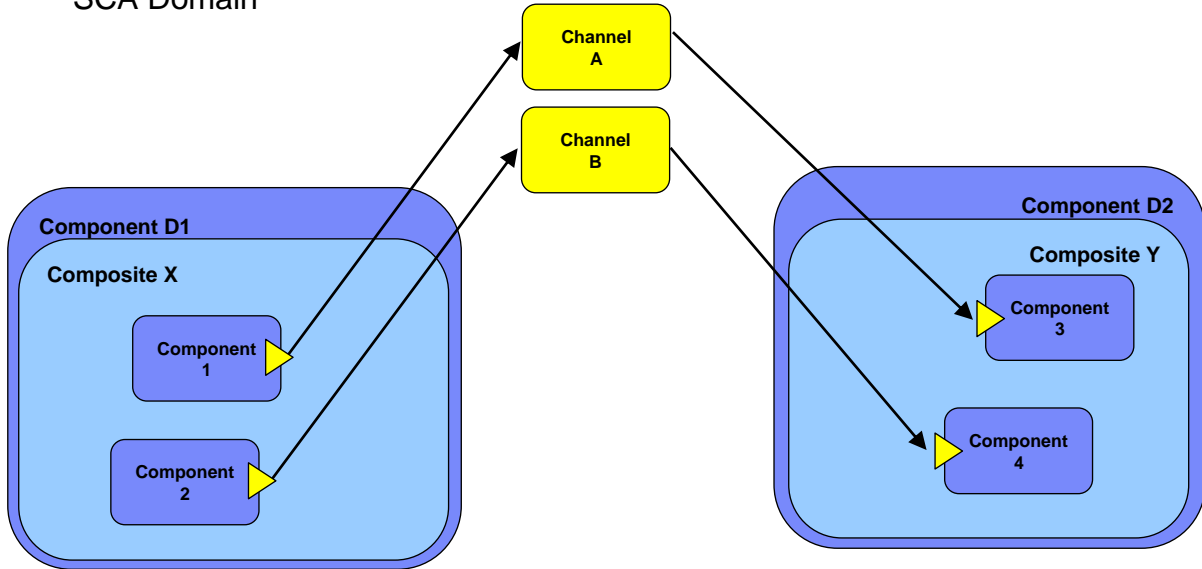
418 In this example, the @target attribute of the producers links them to ChannelA and the @source attribute
419 of the consumers links them to ChannelA. All events from Component1 and Component2 are routed
420 through the ChannelA and are sent to Component3 and Component4.

421 **2.3.2 Producers linked to Consumers via Domain Channels**

422 In this example, component producers of components nested within a domain component transmit events
423 via Domain Channels to component consumers which are also nested below the domain level within a
424 second domain component. This is represented in the Figure 2-6:

425

SCA Domain



426

427

428 *Figure: 2-6: Producers linked to Consumers via Domain Channels*

429

430 For CompositeX:

```
431 <composite name="CompositeX"
432   xmlns="http://www.osoa.org/xmlns/sca/1.0"
433   targetNamespace="http://example.org/example1">
434
435   <component name="Component1">
436     <implementation.java class="org.example.Component1Impl"/>
437     <producer name="Foo_Events" target="/ChannelA"/>
438   </component>
439
440   <component name="Component2">
441     <implementation.java class="org.example.Component2Impl"/>
442     <producer name="Foo_Events" target="/ChannelB"/>
443   </component>
444
445 </composite>
```

446 For CompositeY:

```
447 <composite name="CompositeY"
448   xmlns="http://www.osoa.org/xmlns/sca/1.0"
449   targetNamespace="http://example.org/example1">
450
451   <component name="Component3">
452     <implementation.java class="org.example.Component3Impl"/>
453     <consumer name="Foo_Handling" source="/ChannelA"/>
454   </component>
455
456   <component name="Component4">
457     <implementation.java class="org.example.Component4Impl"/>
458     <consumer name="Foo_Handling" source="/ChannelB"/>
459   </component>
460
461 </composite>
```

462

463 Note the @target and @source attributes of the producers and consumers use the "/<channel-name>"
464 notation to indicate the connection to a channel at the domain level.

465 The following is an example of one way in which the Channels could be deployed to the Domain:

```
466 <composite name="ChannelContribution"  
467   xmlns="http://www.osea.org/xmlns/sca/1.0"  
468   targetNamespace="http://example.org/example1">  
469  
470   <channel name="ChannelA"/>  
471  
472   <channel name="ChannelB"/>  
473  
474 </composite>
```

475
476 The following is an example of two deployment composites that could be used to deploy the two domain-
477 level components (ComponentD1 and ComponentD2):

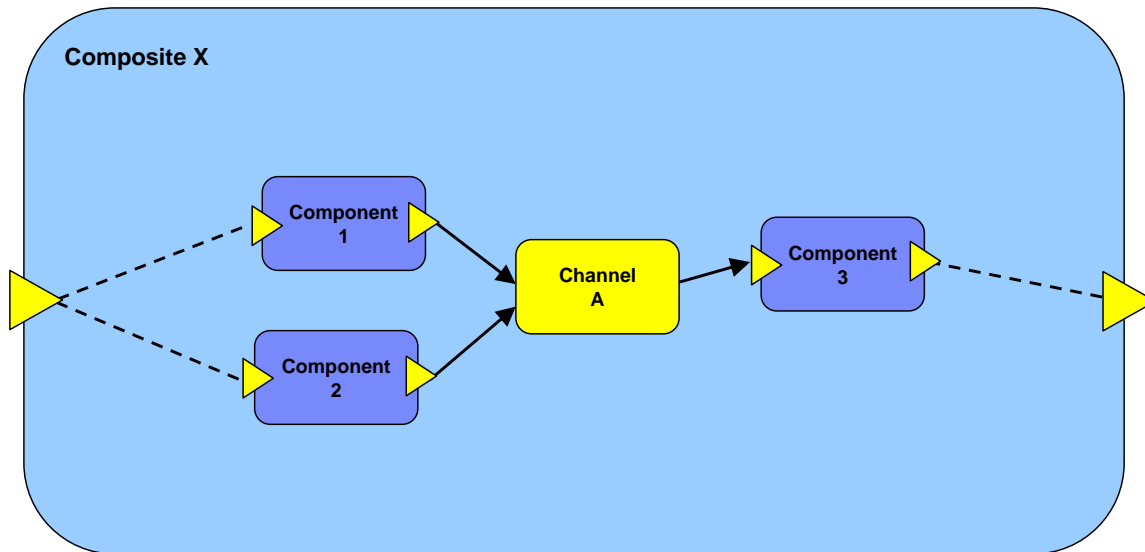
```
478 <composite name="ComponentD1Contribution"  
479   xmlns="http://www.osea.org/xmlns/sca/1.0"  
480   targetNamespace="http://example.org/example1"  
481   xmlns:xmp="http://example.org/example1">  
482  
483   <component name="ComponentD1">  
484     <implementation.composite name="xmp:CompositeX"/>  
485   </component>  
486 </composite>  
487  
488 <composite name="ComponentD2Contribution"  
489   xmlns="http://www.osea.org/xmlns/sca/1.0"  
490   targetNamespace="http://example.org/example1"  
491   xmlns:xmp="http://example.org/example1">  
492  
493   <component name="ComponentD2">  
494     <implementation.composite name="xmp:CompositeY"/>  
495   </component>  
496  
497 </composite>
```

498
499 Note that the domain level components ComponentD1 and ComponentD2 are unable to configure the
500 channels that are used as sources and targets by the components in the lower level composites.

501 **2.3.3 Composite with Promotion of Producers and Consumers**

502 This example shows how a composite can be constructed so that the composite promotes some
503 component consumers and promotes some component producers. This is represented in Figure 2-7.

504



505
506

507 *Figure: 2-7 Promotion of Consumers and Producers by a Composite*

508

509 The corresponding XML for this example follows:

510

```

511 <composite name="CompositeX"
512   xmlns="http://www.oxa.org/xmlns/sca/1.0"
513   targetNamespace="http://example.org/example1">
514
515   <consumer name="Bar_Handling"
516     promotes="Component1/BarHandling Component2/Bar_Handling"/>
517
518   <component name="Component1">
519     <implementation.java class="org.example.Component1Impl"/>
520     <consumer name="Bar_Handling"/>
521     <producer name="Foo_Events" target="ChannelA"/>
522   </component>
523
524   <component name="Component2">
525     <implementation.java class="org.example.Component2Impl"/>
526     <consumer name="Bar_Handling"/>
527     <producer name="Foo_Events" target="ChannelA"/>
528   </component>
529
530   <channel name="ChannelA"/>
531
532   <component name="Component3">
533     <implementation.java class="org.example.Component3Impl"/>
534     <consumer name="Foo_Handling" source="ChannelA"/>
535     <producer name="Special_Events"/>
536   </component>
537
538   <producer name="Special_Events" promotes="Component3/Special_Events"/>
539
540 </composite>

```

541

542 Here, CompositeX has a consumer element named Bar_Handling and producer element named
543 Special_Events. The Bar_Handling consumer promotes the consumers of Component1 and
544 Component2. The Special_Events producer promotes the producer of Component3.

545 When CompositeX is used as an implementation by a higher-level component, the consumer and
546 producer elements of the composite permit the assembler of the higher level component to control where
547 the events relating to this composite are sent to and received from, through configuration of the higher
548 level component. The Component Type of CompositeX above is as follows:

```
549 <componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"  
550     <consumer name="Bar_Handling" />  
551     <producer name="Special_Events" />  
552  
553 </componentType>
```

556

557 3 Implementation and ComponentType

558 Component **implementations** are concrete implementations of business function which provide services
559 and/or which make references to services provided elsewhere. An implementation can also have event
560 producers and consumers. Each producer sends events of one or more event types, while each
561 consumer receives events of one or more event types. Producers and consumers declare the set of
562 event types that they handle through a list of event types. It is also possible to declare that a producer or
563 a consumer handles any event type. In addition, an implementation can have some settable property
564 values.

565 SCA allows a choice of any one of a wide range of **implementation types**, such as Java, BPEL or C++,
566 where each type represents a specific implementation technology. The technology might not simply
567 define the implementation language, such as Java, but might also define the use of a specific framework
568 or runtime environment. Examples include SCA Composite, Java implementations done using the Spring
569 framework or the Java EE EJB technology.

570 **Services, references, consumers, producers, and properties** are the **configurable aspects of an**
571 **implementation**. SCA refers to them collectively as the **component type**.

572 Depending on the implementation type, the implementation can declare the services, references,
573 consumers, producers, and properties that it has and it also might be able to set values for all the
574 characteristics of those services, references, consumers, producers, and properties.

575 So, for example:

- 576 • for a service, the implementation might define the interface, binding(s), a URI, intents, and policy sets,
577 including details of the bindings
- 578 • for a reference, the implementation might define the interface, binding(s), target URI(s), intents, policy
579 sets, including details of the bindings
- 580 • for a consumer, the implementation can define event filters, intents, policy sets, bindings
- 581 • for a producer, the implementation can define event types, intents, policy sets, bindings
- 582 • for a property the implementation might define its type and a default value
- 583 • the implementation itself might define policy intents or concrete policy sets

584 The means by which an implementation declares its services, references, consumers, producers, and
585 properties depend on the type of the implementation. For example, some languages like Java, provide
586 annotations which can be used to declare this information inline in the code.

587 Most of the characteristics of the services, references, consumers, producers, and properties can be
588 overridden by a component that uses and configures the implementation, or the component can decide
589 not to override those characteristics. Some characteristics cannot be overridden, such as intents. Other
590 characteristics, such as interfaces, can only be overridden in particular controlled ways (see [the](#)
591 [Component section](#) for details).

592 3.1 Component Type

593 **Component type** represents the configurable aspects of an implementation. A component type consists
594 of services that are offered, references to other services that can be wired, consumers to which events
595 are delivered, producers that send out events, and properties that can be set. The settable properties and
596 the settable references to services and the settable consumers and producers are configured by a
597 component that uses the implementation.

598 An implementation type specification (for example, the [WS-BPEL Client and Implementation Specification](#)
599 [Version 1.1 \[SCA BPEL\]](#)) specifies the mechanism(s) by which the component type associated with an
600 implementation of that type is derived.

601 Since SCA allows a broad range of implementation technologies, it is expected that some implementation
602 technologies (for example, the [Java Component Implementation Specification Version 1.1 \[SCA-Java\]](#))

603 allow for introspecting the implementation artifact(s) (for example, a Java class) to derive the component
604 type information. Other implementation technologies might not allow for introspection of the
605 implementation artifact(s). In those cases where introspection is not allowed, SCA encourages the use of
606 a SCA component type side file. A **component type side file** is an XML file whose document root
607 element is `sca:componentType`.

608 The implementation type specification defines whether introspection is allowed, whether a side file is
609 allowed, both are allowed or some other mechanism specifies the component type. The component type
610 information derived through introspection is called the **introspected component type**. In any case, the
611 implementation type specification specifies how multiple sources of information are combined to produce
612 the **effective component type**. The effective component type is the component type metadata that is
613 presented to the using component for configuration.

614 **The extension of a componentType side file name MUST be .componentType. [ASM40001]** The name
615 and location of a componentType side file, if allowed, is defined by the implementation type specification.

616 If a component type side file is not allowed for a particular implementation type, the effective component
617 type and introspected component type are one and the same for that implementation type.

618 For the rest of this document, when the term 'component type' is used it refers to the 'effective component
619 type'.

620 Snippet 3-1 shows the componentType pseudo-schema:

621

```
622 <?xml version="1.0" encoding="ASCII"?>  
623 <!-- Component type schema snippet -->  
624 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
625  
626     <service ... />*  
627     <reference ... />*  
628     <consumer ... />*  
629     <producer ... />*  
630     <property ... />*  
631     <implementation ... />?  
632  
633 </componentType>
```

634 *Snippet 3-1: componentType Pseudo-Schema*

635

636 The **componentType** element has the **child elements**:

- 637 • **service : Service (0..n)** – see [component type service section](#).
- 638 • **reference : Reference (0..n)** – see [component type reference section](#).
- 639 • **consumer: Consumer (0..n)** – see [component type consumer section](#).
- 640 • **producer: Producer (0..n)** – see [component type producer section](#).
- 641 • **property : Property (0..n)** – see [component type property section](#).
- 642 • **implementation : Implementation (0..1)** – see [component type implementation](#)
643 [section](#).

644 3.1.1 Service

645 **A Service** represents an addressable interface of the implementation. The service is represented
646 by a **service element** which is a child of the componentType element. There can be **zero or**
647 **more** service elements in a componentType. Snippet 3-2 shows the componentType pseudo-
648 schema with the pseudo-schema for a service child element:

649

```
650 <?xml version="1.0" encoding="ASCII"?>  
651 <!-- Component type service schema snippet -->
```

```

652 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
653
654   <service name="xs:NCName"
655     requires="list of xs:QName"? policySets="list of xs:QName"?>*
656     <interface ... />
657     <binding ... />*
658     <callback?
659       <binding ... />+
660     </callback>
661     <requires/>*
662     <policySetAttachment/>*
663   </service>
664
665   <reference ... />*
666   <consumer ... />*
667   <producer ... />*
668   <property ... />*
669   <implementation ... />?
670
671 </componentType>

```

672 Snippet 3-2: *componentType* Pseudo-Schema with service Child Element

673

674 The **service** element has the **attributes**:

- 675 • **name : NCName (1..1)** - the name of the service. The @name attribute of a <service/> child element
676 of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
677 [\[ASM40003\]](#)
- 678 • **requires : listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
679 [\[SCA-POLICY\]](#) for a description of this attribute.
- 680 • **policySets : listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
681 [\[SCA-POLICY\]](#) for a description of this attribute.

682 The **service** element has the **child elements**:

- 683 • **interface : Interface (1..1)** - A service has **one interface**, which describes the operations provided
684 by the service. For details on the interface element see [the Interface section](#).
- 685 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as children. If the
686 binding element is not present it defaults to <binding.sca>. Details of the binding element are
687 described in [the Bindings section](#).
- 688 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
689 defined, and the callback element has one or more **binding** elements as subelements. The **callback**
690 and its binding subelements are specified if there is a need to have binding details used to handle
691 callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
692 For details on callbacks, see [the Bidirectional Interfaces section](#).
- 693 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
694 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 695 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
696 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
697 description of this element.

698 3.1.2 Reference

699 A **Reference** represents a requirement that the implementation has on a service provided by another
700 component. The reference is represented by a **reference element** which is a child of the componentType
701 element. There can be **zero or more** reference elements in a component type definition. Snippet 3-3
702 shows the componentType pseudo-schema with the pseudo-schema for a reference child element:

703

```

704 <?xml version="1.0" encoding="ASCII"?>
705 <!-- Component type reference schema snippet -->
706 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
707
708   <service ... />*
709
710   <reference name="xs:NCName"
711     autowire="xs:boolean"?
712     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
713     wiredByImpl="xs:boolean"? requires="list of xs:QName"?
714     policySets="list of xs:QName"?>*
715     <interface ... />
716     <binding ... />*
717     <callback?>
718       <binding ... />+
719     </callback>
720     <requires/>*
721     <policySetAttachment/>*
722   </reference>
723
724   <consumer ... />*
725   <producer ... />*
726   <property ... />*
727   <implementation ... />?
728
729 </componentType>

```

730 *Snippet 3-3: componentType Pseudo-Schema with reference Child Element*

731

732 The **reference** element has the **attributes**:

- 733 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a <reference/> child
734 element of a <componentType/> MUST be unique amongst the reference elements of that
735 <componentType/>. [ASM40004]
- 736 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to
737 target services. The multiplicity can have the following values
738 – 0..1 – zero or one wire can have the reference as a source
739 – 1..1 – one wire can have the reference as a source
740 – 0..n - zero or more wires can have the reference as a source
741 – 1..n – one or more wires can have the reference as a source
742 If @multiplicity is not specified, the default value is "1..1".
- 743 • **autowire : boolean (0..1)** - whether the reference is autowired, as described in [the Autowire section](#).
744 Default is false.
- 745 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the reference is
746 wired to the target(s) configured on the reference. If set to "true" it indicates that the target of the
747 reference is set at runtime by the implementation code (e.g. by the code obtaining an endpoint
748 reference by some means and setting this as the target of the reference through the use of
749 programming interfaces defined by the relevant Client and Implementation specification). If
750 @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be
751 ignored by the runtime. [ASM40006]
- 752 • **requires : listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
753 [\[SCA-POLICY\]](#) for a description of this attribute.
- 754 • **policySets : listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
755 [\[SCA-POLICY\]](#) for a description of this attribute.

756 The **reference** element has the **child elements**:

- 757 • **interface : Interface (1..1)** - A reference has **one interface**, which describes the operations used by
758 the reference. The interface is described by an **interface element** which is a child element of the
759 reference element. For details on the interface element see [the Interface section](#).
 - 760 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as children.
761 Details of the binding element are described in the [Bindings section](#).
762 When used with a reference element, a binding element specifies an endpoint which is the target of
763 that binding. A reference cannot mix the use of endpoints specified via binding elements with target
764 endpoints specified via the @target attribute. If the @target attribute is set, the reference cannot also
765 have binding subelements. If binding elements with endpoints are specified, each endpoint uses the
766 binding type of the binding element in which it is defined.
 - 767 • **callback (0..1) / binding : Binding (1..n)** - a **callback** element is used if the interface has a callback
768 defined and the callback element has one or more **binding** elements as subelements. The **callback**
769 and its binding subelements are specified if there is a need to have binding details used to handle
770 callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
771 For details on callbacks, see [the Bidirectional Interfaces section](#).
 - 772 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
773 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
 - 774 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
775 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
776 description of this element.
- 777 For a full description of the setting of target service(s) for a reference, see the section "[Specifying the](#)
778 [Target Service\(s\) for a Reference](#)".

779 3.1.3 Consumer

780 **A Consumer** is represented by a **consumer element** which is a child of the componentType element.
781 There can be **zero or more** consumer elements in a componentType. Snippet 3-4 shows the
782 componentType pseudo-schema with the pseudo-schema for a consumer child element:

```
783 <!-- Component type consumer schema snippet -->
784 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
785
786   <service ... />*
787   <reference ... />*
788
789   <consumer name="xs:NCName"
790     requires="list of xs:QName"? policySets="list of xs:QName"?>*
791     <filters ... />?
792     <requires/>*
793     <policySetAttachment/>*
794   </consumer>
795
796   <producer ... />*
797   <property ... />*
798   <implementation ... />?
799
800 </componentType>
```

801 *Snippet 3-4: componentType Pseudo-Schema with consumer Child Element*

802 The **consumer** element has the **attributes**:

- 803 • **name : NCName (1..1)** - the name of the consumer. The @name attribute of a <consumer/> child
804 element of a <componentType/> MUST be unique amongst the consumer elements of that
805 <componentType/>. [\[ASM40101\]](#)
- 806 • **requires : listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
807 [\[SCA-POLICY\]](#) for a description of this attribute.

- 808 • **policySets : listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
809 [\[SCA-POLICY\]](#) for a description of this attribute.

810 The **consumer** element has the **child elements**:

- 811 • **filter : Filter (0..1)** - A consumer has **zero or one filter**, which specify the events of interest for the
812 consumer. For details on the filter element see [the Filters: Selecting Subsets of Events section](#).
- 813 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
814 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 815 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
816 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
817 description of this element.

818 In the SCA Assembly Model an implementation-specific artifact, for example a Java Class, can manifest
819 itself as both an SCA service and an SCA consumer in the component Type of that implementation. This
820 dual manifestation of an artifact is allowed to facilitate those usecases where the distinction between the
821 two is not relevant to the implementation code. In such a case, it is the assembler's responsibility to
822 decide whether the artifact would receive events from a channel or whether it would receive method
823 invocations from a reference. It is up to each implementation type to decide whether such a feature is
824 supported and how the distinction between dual, consumer-only, and service-only manifestation of an
825 artifact is made. When implementation has an artifact that has dual manifestation, the effective
826 component type of that implementation contains a component type service corresponding to the artifact
827 and a consumer corresponding to the same artifact. The name of the component type service and the
828 name of the component type consumer associated with the artifact are the same. In such a case, an
829 assembler cannot use both the service and the consumer.

830 For artifacts that have service/consumer dual manifestation, the service interface **MUST**

- 831 • be non-bidirectional
- 832 • have only one-way operations

833 [\[ASM40102\]](#)

834

835 **Note:** For this dual nature to work the TC has to define a WSDL-to-EDL mapping.
836 **Note:** The F2F notes do not contain a decision about dual nature
837 **service/consumer ... and recollections of various ppl who attended the f2f vary.**

838 3.1.4 Producer

839 **A Producer** is represented by a **producer element** which is a child of the componentType element.
840 There can be **zero or more** producer elements in a componentType. Snippet 3-5 shows the
841 componentType pseudo-schema with the pseudo-schema for a producer child element:

```
842 <!-- Component type consumer schema snippet -->
843 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
844
845     <service ... />*
846     <reference ... />*
847     <consumer ... />*
848
849     <producer name="xs:NCName"
850         requires="list of xs:QName"? policySets="list of xs:QName"?>*
851         <eventType ... />?
852         <requires/>*
853         <policySetAttachment/>*
854     </producer>
855
856     <property ... />*
857     <implementation ... />?
858
859 </componentType>
```

860 Snippet 3-5: *componentType* Pseudo-Schema with producer Child Element

861

862 The **producer** element has the **attributes**:

- 863 • **name** : **NCName (1..1)** - the name of the producer. The @name attribute of a <producer/> child
864 element of a <componentType/> MUST be unique amongst the producer elements of that
865 <componentType/>. [ASM40103]
- 866 • **requires** : **listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
867 [SCA-POLICY] for a description of this attribute.
- 868 • **policySets** : **listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
869 [SCA-POLICY] for a description of this attribute.

870 The **producer** element has the **child elements**:

- 871 • **eventType** : **EventType (0..1)** - A producer has **zero or one eventType** child subelement. See the
872 Section Use of <eventType> on a Producer.
- 873 • **requires** : **requires (0..n)** - A service element has **zero or more requires subelements**. See the
874 [Policy Framework specification](#) [SCA-POLICY] for a description of this element.
- 875 • **policySetAttachment** : **policySetAttachment (0..n)** - A service element has **zero or more**
876 **policySetAttachment subelements**. See the [Policy Framework specification](#) [SCA-POLICY] for a
877 description of this element.

878 In the SCA Assembly Model an implementation-specific artifact, for example a Java Class instance
879 variable, can manifest itself as both an SCA reference and an SCA producer in the component Type of
880 that implementation. This dual manifestation of an artifact is allowed to facilitate those usecases where
881 the distinction between the two is not relevant to the implementation code. In such a case, it is the
882 assembler's responsibility to decide whether the artifact would send events to a channel or whether the it
883 would invoke methods on a service. It is up to each implementation type to decide whether such a feature
884 is supported and how the distinction between dual, producer-only, and reference-only nature of an artifact
885 is made. When implementation has an artifact that has dual manifestation, the effective component type
886 of that implementation contains a component type reference corresponding to the artifact and a producer
887 corresponding to the same artifact. The name of the component type reference and the name of the
888 component type producer associated with the artifact are the same. In such a case, an assembler cannot
889 use both the reference and the producer.

890 For artifacts that have reference/producer dual manifestation, the reference interface MUST

- 891 • be non-bidirectional
- 892 • have only one-way operations

893 [ASM40104]

894 **Note:** For this dual nature to work the TC has to define a WSDL-to-EDL mapping.

895 3.1.5 Property

896 **Properties** allow for the configuration of an implementation with externally set values. Each Property is
897 defined as a property element. The componentType element can have **zero or more property elements**
898 as its children. Snippet 3-6 shows the componentType pseudo-schema with the pseudo-schema for a
899 reference child element:

900

```
901 <?xml version="1.0" encoding="ASCII"?>
902 <!-- Component type property schema snippet -->
903 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
904
905     <service ... /*
906     <reference ... /*
907     <consumer ... /*
908     <producer ... /*
```

909
910
911
912
913
914
915
916
917

```
<property name="xs:NCName" (type="xs:QName" | element="xs:QName")
  many="xs:boolean"? mustSupply="xs:boolean"?>*
  default-property-value?
</property>

<implementation ... />?

</componentType>
```

918 *Snippet 3-6: componentType Pseudo-Schema with property Child Element*

919

920 The **property** element has the **attributes**:

- 921 • **name : NCName (1..1)** - the name of the property. The @name attribute of a <property/> child
922 element of a <componentType/> MUST be unique amongst the property elements of that
923 <componentType/>. [ASM40005]
- 924 • one of (1..1):
 - 925 – **type : QName** - the type of the property defined as the qualified name of an XML schema type.
926 The value of the property @type attribute MUST be the QName of an XML schema type.
927 [ASM40007]
 - 928 – **element : QName** - the type of the property defined as the qualified name of an XML schema
929 global element – the type is the type of the global element. The value of the property @element
930 attribute MUST be the QName of an XSD global element. [ASM40008]

931 A single property element MUST NOT contain both a @type attribute and an @element attribute.
932 [ASM40010]

- 933 • **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). In the
934 case of a multi-valued property, it is presented to the implementation as a collection of property
935 values. If many is not specified, it takes a default value of false.
- 936 • **mustSupply : boolean (0..1)** - whether the property value needs to be supplied by the component
937 that uses the implementation. Default value is "false". When the componentType has
938 @mustSupply="true" for a property element, a component using the implementation MUST supply a
939 value for the property since the implementation has no default value for the property. [ASM40011] If
940 the implementation has a default-property-value then @mustSupply="false" is appropriate, since the
941 implication of a default value is that it is used when a value is not supplied by the using component.
- 942 • **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property. The value of
943 the property @file attribute MUST be a dereferencable URI to a file containing the value for the
944 property. [ASM40012] The URI can be an absolute URI or a relative URI. For a relative URI, it is
945 taken relative to the base of the contribution containing the implementation. For a description of the
946 format of the file, see the section on Property Value File Format.

947 The property element can contain a default property value as its content. The form of the default property
948 value is as described in the section on Component Property.

949 The value for a property is supplied to the implementation of a component at the time that the
950 implementation is started. The implementation can use the supplied value in any way that it chooses. In
951 particular, the implementation can alter the internal value of the property at any time. However, if the
952 implementation queries the SCA system for the value of the property, the value as defined in the SCA
953 composite is the value returned.

954 The componentType property element can contain an SCA default value for the property declared by the
955 implementation. However, the implementation can have a property which has an implementation defined
956 default value, where the default value is not represented in the componentType. An example of such a
957 default value is where the default value is computed at runtime by some code contained in the
958 implementation. If a using component needs to control the value of a property used by an implementation,
959 the component sets the value explicitly. The SCA runtime MUST ensure that any implementation default

960 property value is replaced by a value for that property explicitly set by a component using that
961 implementation. [ASM40009]

962 3.1.6 Implementation

963 **Implementation** represents characteristics inherent to the implementation itself, in particular intents and
964 policies. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of intents and policies.
965 Snippet 3-7 shows the componentType pseudo-schema with the pseudo-schema for a implementation
966 child element:

967

```
968 <?xml version="1.0" encoding="ASCII"?>  
969 <!-- Component type implementation schema snippet -->  
970 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
971  
972     <service ... /*>  
973     <reference ... /*>  
974     <property ... /*>  
975  
976     <implementation requires="list of xs:QName"?  
977         policySets="list of xs:QName"?>  
978         <requires/*>  
979         <policySetAttachment/*>  
980     </implementation?>  
981  
982 </componentType>
```

983 *Snippet 3-7: componentType Pseudo-Schema with implementation Child Element*

984

985 The **implementation** element has the **attributes**:

- 986 • **requires : listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this attribute.
- 987
- 988 • **policySets : listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this attribute.
- 989

990 The **implementation** element has the **subelements**:

- 991 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 992
- 993 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 994
- 995

996 3.2 Example ComponentType

997 Snippet 3-8 shows the contents of the componentType file for the MyValueServiceImpl implementation.
998 The componentType file shows the services, references, and properties of the MyValueServiceImpl
999 implementation. In this case, Java is used to define interfaces:

```
1000 <?xml version="1.0" encoding="ASCII"?>  
1001 <componentType xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912  
1002     xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
1003     xmlns:storm="http://example.org/storm">  
1004  
1005     <service name="MyValueService">  
1006         <interface.java interface="services.myvalue.MyValueService"/>  
1007     </service>  
1008  
1009     <reference name="customerService">  
1010         <interface.java interface="services.customer.CustomerService"/>
```



```

1011 </reference>
1012 <reference name="stockQuoteService">
1013     <interface.java
1014         interface="services.stockquote.StockQuoteService"/>
1015 </reference>
1016
1017 <consumer name="stormAlertConsumer">
1018     <filters>
1019         <eventType.sca qnames="storm:SouthAmerica" />
1020     </filters>
1021 </consumer>
1022
1023 <producer name="stormAlertProducer">
1024     <eventType.sca qnames="storm:NorthAmerica" />
1025 </producer>
1026
1027 <property name="currency" type="xsd:string">USD</property>
1028
1029 </componentType>

```

1030 *Snippet 3-8: Example componentType*

1031 3.3 Example Implementation

1032 Snippet 3-9 and Snippet 3-10 are an example implementation, written in Java.

1033 **AccountServiceImpl** implements the **AccountService** interface, which is defined via a Java interface
1034 (TODO: update when the Java CI adds support for events):

```

1035 package services.account;
1036
1037 @Remotable
1038 public interface AccountService {
1039
1040     AccountReport getAccountReport(String customerID);
1041 }

```

1042 *Snippet 3-9: Example Interface in Java*

1043
1044 Snippet 3-10 is a full listing of the AccountServiceImpl class, showing the Service it implements, plus the
1045 service references it makes and the settable properties that it has. Notice the use of Java annotations to
1046 mark SCA aspects of the code, including the @Property, @Reference and @Service annotations:

```

1047 package services.account;
1048
1049 import java.util.List;
1050
1051 import commonj.sdo.DataFactory;
1052
1053 import org.oasisopen.sca.annotation.Property;
1054 import org.oasisopen.sca.annotation.Reference;
1055 import org.oasisopen.sca.annotation.Service;
1056
1057 import services.accountdata.AccountDataService;
1058 import services.accountdata.CheckingAccount;
1059 import services.accountdata.SavingsAccount;
1060 import services.accountdata.StockAccount;
1061 import services.stockquote.StockQuoteService;
1062
1063 @Service(AccountService.class)
1064 public class AccountServiceImpl implements AccountService {
1065
1066     @Property
1067     private String currency = "USD";
1068
1069     @Reference
1070     private AccountDataService accountDataService;
1071     @Reference

```

```

1072 private StockQuoteService stockQuoteService;
1073
1074 public AccountReport getAccountReport(String customerID) {
1075
1076     DataFactory dataFactory = DataFactory.INSTANCE;
1077     AccountReport accountReport =
1078         (AccountReport) dataFactory.create (AccountReport.class);
1079     List accountSummaries = accountReport.getAccountSummaries();
1080
1081     CheckingAccount checkingAccount = accountDataService.getCheckingAccount (customerID);
1082     AccountSummary checkingAccountSummary =
1083         (AccountSummary) dataFactory.create (AccountSummary.class);
1084     checkingAccountSummary.setAccountNumber (checkingAccount.getAccountNumber ());
1085     checkingAccountSummary.setAccountType ("checking");
1086
1087     checkingAccountSummary.setBalance (fromUSDollarToCurrency (checkingAccount.getBalance ());
1088     accountSummaries.add (checkingAccountSummary);
1089
1090     SavingsAccount savingsAccount = accountDataService.getSavingsAccount (customerID);
1091     AccountSummary savingsAccountSummary =
1092         (AccountSummary) dataFactory.create (AccountSummary.class);
1093     savingsAccountSummary.setAccountNumber (savingsAccount.getAccountNumber ());
1094     savingsAccountSummary.setAccountType ("savings");
1095
1096     savingsAccountSummary.setBalance (fromUSDollarToCurrency (savingsAccount.getBalance ());
1097     accountSummaries.add (savingsAccountSummary);
1098
1099     StockAccount stockAccount = accountDataService.getStockAccount (customerID);
1100     AccountSummary stockAccountSummary =
1101         (AccountSummary) dataFactory.create (AccountSummary.class);
1102     stockAccountSummary.setAccountNumber (stockAccount.getAccountNumber ());
1103     stockAccountSummary.setAccountType ("stock");
1104     float balance =
1105
1106     (stockQuoteService.getQuote (stockAccount.getSymbol ()) * stockAccount.getQuantity ());
1107     stockAccountSummary.setBalance (fromUSDollarToCurrency (balance));
1108     accountSummaries.add (stockAccountSummary);
1109
1110     return accountReport;
1111 }
1112
1113 private float fromUSDollarToCurrency(float value) {
1114
1115     if (currency.equals ("USD")) return value; else
1116     if (currency.equals ("EURO")) return value * 0.8f; else
1117     return 0.0f;
1118 }
1119 }

```

1120 *Snippet 3-10: Example Component Implementation in Java*

1121

1122 Snippet 3-11 is the SCA componentType definition for the AccountServiceImpl, derived by introspection
1123 of the code above:

```

1124 <?xml version="1.0" encoding="ASCII"?>
1125 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1126     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
1127
1128     <service name="AccountService">
1129         <interface.java interface="services.account.AccountService"/>
1130     </service>
1131     <reference name="accountDataService">
1132         <interface.java
1133             interface="services.accountdata.AccountDataService"/>
1134     </reference>
1135     <reference name="stockQuoteService">
1136         <interface.java
1137             interface="services.stockquote.StockQuoteService"/>
1138     </reference>

```

```
1139
1140     <property name="currency" type="xsd:string"/>
1141
1142 </componentType>
```

1143 *Snippet 3-11: Example componentType for Implementation in Snippet 3-10*

1144

1145 Note that the componentType property element for "currency" has no default value declared, despite the
1146 code containing an initializer for the property field setting it to "USD". This is because the initializer cannot
1147 be introspected at runtime and the value cannot be extracted.

1148 For full details about Java implementations, see the [Java Component Implementation Specification \[SCA-](#)
1149 [Java\]](#). Other implementation types have their own specification documents.

4 Component

1150

1151 **Components** are the basic elements of business function in an SCA assembly, which are combined into
1152 complete business solutions by SCA composites.

1153 **Components** are configured *instances of implementations*. Components provide and consume
1154 services and/or events. More than one component can use and configure the same implementation,
1155 where each component configures the implementation differently.

1156 Components are declared as subelements of a composite in a file with a **.composite** extension. A
1157 component is represented by a **component element** which is a child of the composite element. There
1158 can be **zero or more** component elements within a composite. Snippet 4-1 shows the composite pseudo-
1159 schema with the pseudo-schema for the component child element:

1160

```
1161 <?xml version="1.0" encoding="UTF-8"?>  
1162 <!-- Component schema snippet -->  
1163 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
1164 ...  
1165   <component name="xs:NCName" autowire="xs:boolean"?  
1166     requires="list of xs:QName"? policySets="list of xs:QName"?>*  
1167     <implementation ... />?  
1168     <service ... />*  
1169     <reference ... />*  
1170     <consumer ... />*  
1171     <producer ... />*  
1172     <property ... />*  
1173     <requires/>*  
1174     <policySetAttachment/>*  
1175   </component>  
1176   ...  
1177 </composite>
```

1178 *Snippet 4-1: composite Pseudo-Schema with component Child Element*

1179

1180 The **component** element has the **attributes**:

- 1181 • **name : NCName (1..1)** – the name of the component. The @name attribute of a <component/> child
1182 element of a <composite/> MUST be unique amongst the component elements of that <composite/>
1183 [\[ASM50001\]](#)
- 1184 • **autowire : boolean (0..1)** – whether contained component references are autowired, as described in
1185 [the Autowire section](#). Default is false.
- 1186 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1187 [\[SCA-POLICY\]](#) for a description of this attribute.
- 1188 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1189 [\[SCA-POLICY\]](#) for a description of this attribute.

1190 The **component** element has the **child elements**:

- 1191 • **implementation : ComponentImplementation (0..1)** – [see component implementation section](#).
- 1192 • **service : ComponentService (0..n)** – [see component service section](#).
- 1193 • **reference : ComponentReference (0..n)** – [see component reference section](#).
- 1194 • **consumer: Consumer (0..n)** – [see component consumer section](#).
- 1195 • **producer: Producer (0..n)** – [see component producer section](#).
- 1196 • **property : ComponentProperty (0..n)** – [see component property section](#).

- 1197 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1198 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1199 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1200 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
1201 description of this element.

1202 4.1 Implementation

1203 A component element has **one implementation element** as its child, which points to the implementation
1204 used by the component.

```
1205 <?xml version="1.0" encoding="UTF-8"?>
1206 <!-- Component Implementation schema snippet -->
1207 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1208   ...
1209   <component ... >*
1210     <implementation requires="list of xs:QName"?
1211       policySets="list of xs:QName"?
1212       <requires/>*
1213       <policySetAttachment/>*
1214     </implementation>
1215     <service ... />*
1216     <reference ... />*
1217     <consumer ... />*
1218     <producer ... />*
1219     <property ... />*
1220   </component>
1221   ...
1222 </composite>
```

1223 *Snippet 4-2: component Pseudo-Schema with implementation Child Element*

1224
1225 The component provides the extensibility point in the assembly model for different implementation types.
1226 The references to implementations of different types are expressed by implementation type specific
1227 implementation elements.

1228 For example the elements **implementation.java**, **implementation.bpel**, **implementation.cpp**, and
1229 **implementation.c** point to Java, BPEL, C++, and C implementation types respectively.
1230 **implementation.composite** points to the use of an SCA composite as an implementation.
1231 **implementation.spring** and **implementation.ejb** are used for Java components written to the Spring
1232 framework and the Java EE EJB technology respectively.

1233 Snippet 4-3 – Snippet 4-5 show implementation elements for the Java and BPEL implementation types
1234 and for the use of a composite as an implementation:

```
1235  
1236 <implementation.java class="services.myvalue.MyValueServiceImpl"/>
```

1237 *Snippet 4-3: Example implementation.java Element*

```
1238  
1239 <implementation.bpel process="ans:MoneyTransferProcess"/>
```

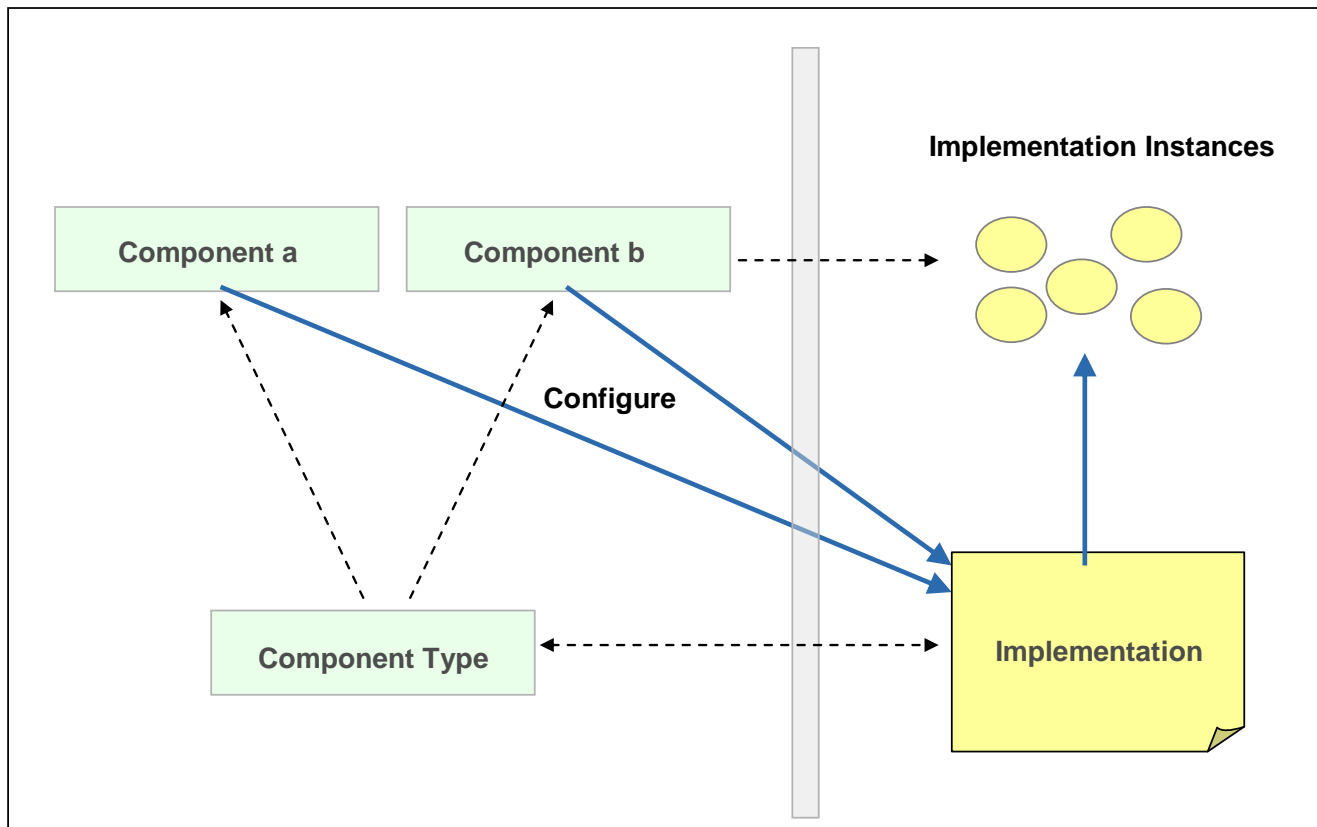
1240 *Snippet 4-4: Example implementation.bpel Element*

```
1241  
1242 <implementation.composite name="bns:MyValueComposite"/>
```

1243 *Snippet 4-5: Example implementation.composite Element*

1244
1245 New implementation types can be added to the model as described in the Extension Model section.

1246 At runtime, an **implementation instance** is a specific runtime instantiation of the implementation – its
 1247 runtime form depends on the implementation technology used. The implementation instance derives its
 1248 business logic from the implementation on which it is based, but the values for its properties and
 1249 references are derived from the component which configures the implementation.



1250
 1251 *Figure 4-1: Relationship of Component and Implementation*

1252 4.2 Service

1253 The component element can have **zero or more service elements** as children which are used to
 1254 configure the services of the component. The services that can be configured are defined by the
 1255 implementation. Snippet 4-6 shows the component pseudo-schema with the pseudo-schema for a service
 1256 child element:

1257

```

1258 <?xml version="1.0" encoding="UTF-8"?>
1259 <!-- Component Service schema snippet -->
1260 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1261   ...
1262   <component ... >*>
1263     <implementation ... />
1264     <service name="xs:NCName" requires="list of xs:QName"?
1265       policySets="list of xs:QName"?>*>
1266       <interface ... />?
1267       <binding ... />*>
1268       <callback?>
1269         <binding ... />+
1270       </callback>
1271       <requires/>*>
1272       <policySetAttachment/>*>
1273     </service>
1274   </reference ... />*>
  
```

```
1275     <consumer ... />*
1276     <producer ... />*
1277     <property ... />*
1278     </component>
1279     ...
1280 </composite>
```

1281 *Snippet 4-6: component Pseudo-Schema with service Child Element*

1282

1283 The **component service** element has the **attributes**:

1284 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service element of a
1285 <component/> MUST be unique amongst the service elements of that <component/> [ASM50002]
1286 The @name attribute of a service element of a <component/> MUST match the @name attribute of a
1287 service element of the componentType of the <implementation/> child element of the component.
1288 [ASM50003]

1289 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1290 [SCA-POLICY] for a description of this attribute.
1291 Note: The effective set of policy intents for the service consists of any intents explicitly stated in this
1292 @requires attribute, combined with any intents specified for the service by the implementation.

1293 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1294 [SCA-POLICY] for a description of this attribute.

1295 The **component service** element has the **child elements**:

1296 • **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the operations
1297 provided by the service. The interface is described by an **interface element** which is a child element
1298 of the service element. If no interface is specified, then the interface specified for the service in the
1299 componentType of the implementation is in effect. If an interface is declared for a component service,
1300 the interface MUST provide a compatible subset of the interface declared for the equivalent service in
1301 the componentType of the implementation [ASM50004] For details on the interface element see the
1302 [Interface section](#).

1303 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as children. If no
1304 binding elements are specified for the service, then the bindings specified for the equivalent service in
1305 the componentType of the implementation MUST be used, but if the componentType also has no
1306 bindings specified, then <binding.sca/> MUST be used as the binding. If binding elements are
1307 specified for the service, then those bindings MUST be used and they override any bindings specified
1308 for the equivalent service in the componentType of the implementation. [ASM50005] Details of the
1309 binding element are described in the [Bindings section](#). The binding, combined with any PolicySets in
1310 effect for the binding, needs to satisfy the set of policy intents for the service, as described in the
1311 [Policy Framework specification](#) [SCA-POLICY].

1312 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
1313 defined and the callback element has one or more **binding** elements as subelements. The **callback**
1314 and its binding subelements are specified if there is a need to have binding details used to handle
1315 callbacks. If the callback element is present and contains one or more binding child elements, then
1316 those bindings MUST be used for the callback. [ASM50006] If the callback element is not present, the
1317 behaviour is runtime implementation dependent.

1318 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1319 [Policy Framework specification](#) [SCA-POLICY] for a description of this element.

1320 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1321 **policySetAttachment subelements**. See the [Policy Framework specification](#) [SCA-POLICY] for a
1322 description of this element.

1323 4.3 Reference

1324 The component element can have **zero or more reference elements** as children which are used to
1325 configure the references of the component. The references that can be configured are defined by the
1326 implementation. Snippet 4-7 shows the component pseudo-schema with the pseudo-schema for a
1327 reference child element:

1328

```
1329 <?xml version="1.0" encoding="UTF-8"?>
1330 <!-- Component Reference schema snippet -->
1331 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1332 ...
1333 <component ... >*
1334 <implementation ... />
1335 <service ... />*
1336 <reference name="xs:NCName"
1337     target="list of xs:anyURI"? autowire="xs:boolean"?
1338     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
1339     nonOverridable="xs:boolean"
1340     wiredByImpl="xs:boolean"? requires="list of xs:QName"?
1341     policySets="list of xs:QName"?>*
1342 <interface ... />?
1343 <binding uri="xs:anyURI"? requires="list of xs:QName"?
1344     policySets="list of xs:QName"?/>*
1345 <callback?>
1346 <binding ... />+
1347 </callback>
1348 <requires/>*
1349 <policySetAttachment/>*
1350 </reference>
1351 <consumer ... />*
1352 <producer ... />*
1353 <property ... />*
1354 </component>
1355 ...
1356 </composite>
```

1357 *Snippet 4-7: component Pseudo-Schema with reference Child Element*

1358

1359 The **component reference** element has the **attributes**:

- 1360 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service element of a
1361 <component/> MUST be unique amongst the service elements of that <component/> [ASM50007]
1362 The @name attribute of a reference element of a <component/> MUST match the @name attribute of
1363 a reference element of the componentType of the <implementation/> child element of the component.
1364 [ASM50008]
- 1365 • **autowire : boolean (0..1)** – whether the reference is autowired, as described in the [Autowire section](#).
1366 The default value of the @autowire attribute MUST be the value of the @autowire attribute on the
1367 component containing the reference, if present, or else the value of the @autowire attribute of the
1368 composite containing the component, if present, and if neither is present, then it is "false".
1369 [ASM50043]
- 1370 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1371 [SCA-POLICY] for a description of this attribute.
1372 Note: The effective set of policy intents for the reference consists of any intents explicitly stated in this
1373 @requires attribute, combined with any intents specified for the reference by the implementation.
- 1374 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1375 [SCA-POLICY] for a description of this attribute.

1376 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to
1377 target services. Overrides the multiplicity specified for this reference in the componentType of the
1378 implementation. The multiplicity can have the following values

- 1379 – 0..1 – zero or one wire can have the reference as a source
- 1380 – 1..1 – one wire can have the reference as a source
- 1381 – 0..n - zero or more wires can have the reference as a source
- 1382 – 1..n – one or more wires can have the reference as a source

1383 The value of multiplicity for a component reference MUST only be equal or further restrict any value
1384 for the multiplicity of the reference with the same name in the componentType of the implementation,
1385 where further restriction means 0..n to 0..1 or 1..n to 1..1. [ASM50009]

1386 If not present, the value of multiplicity is equal to the multiplicity specified for this reference in the
1387 componentType of the implementation - if not present in the componentType, the value defaults to
1388 1..1.

1389 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on multiplicity setting.
1390 Each value wires the reference to a component service that resolves the reference. For more details
1391 on wiring see [the section on Wires](#). Overrides any target specified for this reference on the
1392 implementation.

1393 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that the
1394 implementation wires this reference dynamically. If set to "true" it indicates that the target of the
1395 reference is set at runtime by the implementation code (e.g. by the code obtaining an endpoint
1396 reference by some means and setting this as the target of the reference through the use of
1397 programming interfaces defined by the relevant Client and Implementation specification). If
1398 @wiredByImpl="true" is set for a reference, then the reference MUST NOT be wired statically within a
1399 composite, but left unwired. [ASM50010]

1400 • **nonOverridable : boolean (0..1)** - a boolean value, "false" by default, which indicates whether this
1401 component reference can have its targets overridden by a composite reference which promotes the
1402 component reference.
1403 If @nonOverridable==false, if any target(s) are configured onto the composite references which
1404 promote the component reference, then those targets **replace** all the targets explicitly declared on the
1405 component reference for any value of @multiplicity on the component reference. If no targets are
1406 defined on any of the composite references which promote the component reference, then any
1407 targets explicitly declared on the component reference are used. This means in effect that any targets
1408 declared on the component reference act as default targets for that reference.

1410 If a component reference has @multiplicity 0..1 or 1..1 and @nonOverridable==true, then the
1411 component reference MUST NOT be promoted by any composite reference. [ASM50042]

1413 If @nonOverridable==true, and the component reference @multiplicity is 0..n or 1..n, any targets
1414 configured onto the composite references which promote the component reference are added to any
1415 references declared on the component reference - that is, the targets are additive.

1416 The component reference element has the child elements:

1417 • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes the operations
1418 of the reference. The interface is described by an **interface element** which is a child element of the
1419 reference element. If no interface is specified, then the interface specified for the reference in the
1420 componentType of the implementation is in effect. If an interface is declared for a component
1421 reference, the interface MUST provide a compatible superset of the interface declared for the
1422 equivalent reference in the componentType of the implementation. [ASM50011] For details on the
1423 interface element see [the Interface section](#).

1424 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as children. If no
1425 binding elements are specified for the reference, then the bindings specified for the equivalent
1426 reference in the componentType of the implementation MUST be used. If binding elements are
1427 specified for the reference, then those bindings MUST be used and they override any bindings

1428 specified for the equivalent reference in the componentType of the implementation. [ASM50012] It is
1429 valid for there to be no binding elements on the component reference and none on the reference in
1430 the componentType - the binding used for such a reference is determined by the target service. See
1431 the section on the bindings of component services for a description of how the binding(s) applying to
1432 a service are determined.

1433 Details of the binding element are described in the Bindings section. The binding, combined with any
1434 PolicySets in effect for the binding, needs to satisfy the set of policy intents for the reference, as
1435 described in the Policy Framework specification [SCA-POLICY].

1436 A reference identifies zero or more target services that satisfy the reference. This can be done in a
1437 number of ways, which are fully described in section "Specifying the Target Service(s) for a
1438 Reference"

- 1439 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element used if the interface has a callback
1440 defined and the callback element has one or more **binding** elements as subelements. The **callback**
1441 and its binding subelements are specified if there is a need to have binding details used to handle
1442 callbacks. If the callback element is present and contains one or more binding child elements, then
1443 those bindings MUST be used for the callback. [ASM50006] If the callback element is not present,
1444 the behaviour is runtime implementation dependent.
- 1445 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1446 Policy Framework specification [SCA-POLICY] for a description of this element.
- 1447 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1448 **policySetAttachment subelements**. See the Policy Framework specification [SCA-POLICY] for a
1449 description of this element.

1450 4.3.1 Specifying the Target Service(s) for a Reference

1451 A reference defines zero or more target services that satisfy the reference. The target service(s) can be
1452 defined in the following ways:

- 1453 1. Through a value specified in the @target attribute of the reference element
- 1454 2. Through a target URI specified in the @uri attribute of a binding element which is
1455 a child of the reference element
- 1456 3. Through the setting of one or more values for binding-specific attributes and/or
1457 child elements of a binding element that is a child of the reference element
- 1458 4. Through the specification of @autowire="true" for the reference (or through
1459 inheritance of that value from the component or composite containing the
1460 reference)
- 1461 5. Through the specification of @wiredByImpl="true" for the reference
- 1462 6. Through the promotion of a component reference by a composite reference of the
1463 composite containing the component (the target service is then identified by the
1464 configuration of the composite reference)
- 1465 7. Through the presence of a <wire/> element which has the reference specified in
1466 its @source attribute.

1467 Combinations of these different methods are allowed, and the following rules MUST be observed:

- 1468 • If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.
1469 [ASM50013]
- 1470 • If @autowire="true", the autowire procedure MUST only be used if no target is identified by any of the
1471 other ways listed above. It is not an error if @autowire="true" and a target is also defined through
1472 some other means, however in this case the autowire procedure MUST NOT be used. [ASM50014]
- 1473 • If a reference has a value specified for one or more target services in its @target attribute, there
1474 MUST NOT be any child <binding/> elements declared for that reference. [ASM50026]

- 1475 • If a binding element has a value specified for a target service using its @uri attribute, the binding
1476 element MUST NOT identify target services using binding specific attributes or elements.
1477 [ASM50015]
- 1478 • It is possible that a particular binding type uses more than a simple URI for the address of a target
1479 service. In cases where a reference element has a binding subelement that uses more than simple
1480 URI, the @uri attribute of the binding element MUST NOT be used to identify the target service - in
1481 this case binding specific attributes and/or child elements MUST be used. [ASM50016]
- 1482 • If any <wire/> element with its @replace attribute set to "true" has a particular reference specified in
1483 its @source attribute, the value of the @target attribute for that reference MUST be ignored and
1484 MUST NOT be used to define target services for that reference. [ASM50034]

1485 4.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

1486 The number of target services configured for a reference are constrained by the following rules.

- 1487 • A reference with multiplicity 0..1 MUST have no more than one target service defined. [ASM50039]
- 1488 • A reference with multiplicity 1..1 MUST have exactly one target service defined. [ASM50040]
- 1489 • A reference with multiplicity 1..n MUST have at least one target service defined. [ASM50041]
- 1490 • A reference with multiplicity 0..n can have any number of target services defined.

1491 Where it is detected that the rules for the number of target services for a reference have been violated,
1492 either at deployment or at execution time, an SCA Runtime MUST raise an error no later than when the
1493 reference is invoked by the component implementation. [ASM50022]

1494 For example, where a composite is used as a component implementation, wires and target services
1495 cannot be added to the composite after deployment. As a result, for components which are part of the
1496 composite, both missing wires and wires with a non-existent target can be detected at deployment time
1497 through a scan of the contents of the composite.

1498 A contrasting example is a component deployed to the SCA Domain. At the Domain level, the target of a
1499 wire, or even the wire itself, can form part of a separate deployed contribution and as a result these can
1500 be deployed after the original component is deployed. For the cases where it is valid for the reference to
1501 have no target service specified, the component implementation language specification needs to define
1502 the programming model for interacting with an untargetted reference.

1503 Where a component reference is promoted by a composite reference, the promotion MUST be treated
1504 from a multiplicity perspective as providing 0 or more target services for the component reference,
1505 depending upon the further configuration of the composite reference. These target services are in
1506 addition to any target services identified on the component reference itself, subject to the rules relating to
1507 multiplicity. [ASM50025]

1508 4.4 Consumer

1509 The component element can have **zero or more consumer elements** as children, which are used to
1510 configure the consumers of the component. The consumers that can be configured are defined by the
1511 implementation. Snippet 4-8 shows the component pseudo-schema with the pseudo-schema for a
1512 consumer child element.

1513

```
1514 <!-- Component Consumer schema snippet -->
1515 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1516 ...
1517   <component ... > *
1518     <implementation ... />
1519     <service ... /> *
1520     <reference ... /> *
1521     <consumer name="xs:NCName"
1522       requires="list of xs:QName"?
1523       policySets="list of xs:QName"?
1524       source="list of xs:anyURI"?
1525     />
1526   />
1527 />
```

```

1525     <filters/>?
1526     <requires/>*
1527     <policySetAttachment/>*
1528   </consumer>*
1529   <producer ... />*
1530   <property ... />*
1531 </component>
1532 ...
1533 </composite>

```

1534 *Snippet 4-8: Component Pseudo-Schema with consumer Child Element*

1535

1536 The consumer element has the following attributes:

- 1537 • **name: NCName (1..1)** – the name of the consumer. The @name attribute of a consumer element
1538 of a <component/> MUST be unique amongst the consumer elements of that <component/>.
1539 [ASM50101] The @name attribute of a consumer element of a <component/> MUST match the
1540 @name attribute of a consumer element of the componentType of the <implementation/> child
1541 element of the component. [ASM50102]
- 1542 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1543 [\[SCA-POLICY\]](#) for a description of this attribute.
1544 Note: The effective set of policy intents for the consumer consists of any intents explicitly stated in
1545 this @requires attribute, combined with any intents specified for the service by the
1546 implementation.
- 1547 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1548 [\[SCA-POLICY\]](#) for a description of this attribute.
- 1549 • **source: listOfAnyURIs (0..1)** – a list of one or more of event sources such as the URI of a
1550 channel. The form of the URI for a channel is described in section The URI of a Channel.

1551 The consumer element has the following child elements:

- 1552 • **filters: Filters (0..1)** – filter elements. See the section Filters: Selecting Subsets of Events for a
1553 detailed description of filters.
- 1554 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1555 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1556 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1557 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for
1558 a description of this element.

1559 The consumer can receive events from all of the event sources identified in the @source attribute. It is
1560 valid to specify no sources (ie the consumer is "unconnected"). If the consumer is unconnected, no
1561 events are received. If the name of the consumer is the same as a service within the same component,
1562 then both the consumer and the service MUST NOT be connected.

1563 4.5 Producer

1564 The component element can have **zero or more producer elements** as children, which are used to
1565 configure the producers of the component. The producers that can be configured are defined by the
1566 implementation. Snippet 4-9 shows the component pseudo-schema with the pseudo-schema for a
1567 producer child element.

1568

```

1569 <!-- Component Consumer schema snippet -->
1570 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1571   ...
1572   <component ... >*
1573     <implementation ... />
1574     <service ... />*

```

```

1575     <reference ... />*
1576     <consumer ... />*
1577     <producer name="xs:NCName"
1578         requires="list of xs:QName"?
1579         policySets="list of xs:QName"?
1580         target="list of xs:anyURI"?>
1581         <eventType ... />?
1582         <requires/>*
1583         <policySetAttachment/>*
1584     </producer>*
1585     <property ... />*
1586 </component>
1587 ...
1588 </composite>

```

1589 *Snippet 4-9: Component Pseudo-Schema with producer Child Element*

1590

1591 The producer element has the following attributes:

1592

- 1593 • **name: NCName (1..1)** – the name of the producer. The @name attribute of a producer element
1594 of a <component/> MUST be unique amongst the producer elements of that <component/>.
1595 [ASM50103] The @name attribute of a producer element of a <component/> MUST match the
1596 @name attribute of a producer element of the componentType of the <implementation/> child
1597 element of the component. [ASM50104]
- 1598 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1599 [\[SCA-POLICY\]](#) for a description of this attribute.
1600 Note: The effective set of policy intents for the producer consists of any intents explicitly stated in
1601 this @requires attribute, combined with any intents specified for the service by the
1602 implementation.
- 1603 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1604 [\[SCA-POLICY\]](#) for a description of this attribute.
- 1605 • **target: listOfURIs (0..1)** – a list of one or more of targets to which events are sent, such as the
1606 URI of a channel. Where multiple targets are identified, all the messages emitted by the producer
1607 are sent to each target. The form of the URI for a channel is described in section The URI of a
1608 Channel.

1609 The producer element has the following child elements:

- 1610 • **eventType : EventType (0..1)** - A producer has **zero or one eventType** child subelement. See
1611 Section Use of <eventType> on a Producer.
- 1612 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1613 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1614 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1615 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for
1616 a description of this element.

1617 Events produced by the producer are sent to all the targets identified in the @target attribute. It is valid to
1618 specify no targets (ie the producer is "unconnected") - in this case events produced are discarded.

1619 If the name of the producer is the same as a reference within the same component, then both the
1620 producer and the reference MUST NOT be connected.

1621 4.6 Property

1622 The component element has **zero or more property elements** as its children, which are used to
1623 configure data values of properties of the implementation. Each property element provides a value for the
1624 named property, which is passed to the implementation. The properties that can be configured and their

1625 types are defined by the component type of the implementation. An implementation can declare a
1626 property as multi-valued, in which case, multiple property values can be present for a given property.
1627 The property value can be specified in **one** of five ways:

- 1628 • As a value, supplied in the **@value** attribute of the property element.

1629 If the **@value** attribute of a component property element is declared, the type of the property **MUST**
1630 be an XML Schema simple type and the **@value** attribute **MUST** contain a single value of that type.
1631 [\[ASM50027\]](#)

1632 For example,

```
1633 <property name="pi" value="3.14159265" />
```

1634 *Snippet 4-10: Example property using @value attribute*

1635

- 1636 • As a value, supplied as the content of the **value** subelement(s) of the property element.

1637 If the **value** subelement of a component property is specified, the type of the property **MUST** be an
1638 XML Schema simple type or an XML schema complex type. [\[ASM50028\]](#)

1639 For example,

- 1640 – property defined using a XML Schema simple type and which contains a single value

```
1641 <property name="pi">  
1642   <value>3.14159265</value>  
1643 </property>
```

1644 *Snippet 4-11: Example property with a Simple Type Containing a Single Value*

1645

- 1646 – property defined using a XML Schema simple type and which contains multiple values

```
1647 <property name="currency">  
1648   <value>EURO</value>  
1649   <value>USDollar</value>  
1650 </property>
```

1651 *Snippet 4-12: Example property with a Simple Type Containing Multiple Values*

1652

- 1653 – property defined using a XML Schema complex type and which contains a single value

```
1654 <property name="complexFoo">  
1655   <value attr="bar">  
1656     <foo:a>TheValue</foo:a>  
1657     <foo:b>InterestingURI</foo:b>  
1658   </value>  
1659 </property>
```

1660 *Snippet 4-13: Example property with a Complex Type Containing a Single Value*

1661

- 1662 – property defined using a XML Schema complex type and which contains multiple values

```
1663 <property name="complexBar">  
1664   <value anotherAttr="foo">  
1665     <bar:a>AValue</bar:a>  
1666     <bar:b>InterestingURI</bar:b>  
1667   </value>  
1668   <value attr="zing">  
1669     <bar:a>BValue</bar:a>  
1670     <bar:b>BoringURI</bar:b>  
1671   </value>  
1672 </property>
```

1673 *Snippet 4-14: Example property with a Complex Type Containing Multiple Values*

1674

- 1675 • As a value, supplied as the content of the property element.

1676 If a component property value is declared using a child element of the <property/> element, the type
1677 of the property MUST be an XML Schema global element and the declared child element MUST be
1678 an instance of that global element. [ASM50029]

1679 For example,

- 1680 – property defined using a XML Schema global element declaration and which contains a single
1681 value

```
1682 <property name="foo">  
1683   <foo:SomeGED ...>...</foo:SomeGED>  
1684 </property>
```

1685 *Snippet 4-15: Example property with a Global Element Declaration Containing a Single Value*

1686

- 1687 – property defined using a XML Schema global element declaration and which contains multiple
1688 values

```
1689 <property name="bar">  
1690   <bar:SomeOtherGED ...>...</bar:SomeOtherGED>  
1691   <bar:SomeOtherGED ...>...</bar:SomeOtherGED>  
1692 </property>
```

1693 *Snippet 4-16: Example property with a Global Element Declaration Containing Multiple Values*

1694

- 1695 • By referencing a Property value of the composite which contains the component. The reference is
1696 made using the **@source** attribute of the property element.

1697 The form of the value of the @source attribute follows the form of an XPath expression. This form
1698 allows a specific property of the composite to be addressed by name. Where the composite property
1699 is of a complex type, the XPath expression can be extended to refer to a sub-part of the complex
1700 property value.

1701 So, for example, `source="$currency"` is used to reference a property of the composite called
1702 "currency", while `source="$currency/a"` references the sub-part "a" of the complex composite
1703 property with the name "currency".

- 1704 • By specifying a dereferencable URI to a file containing the property value through the **@file** attribute.
1705 The contents of the referenced file are used as the value of the property.

1706

1707 If more than one property value specification is present, the @source attribute takes precedence, then the
1708 @file attribute.

1709 For a property defined using a XML Schema simple type and for which a single value is desired, can be
1710 set either using the @value attribute or the <value> child element. The two forms in such a case are
1711 equivalent.

1712 When a property has multiple values set, all the values MUST be contained within a single property
1713 element. [ASM50044]

1714 The type of the property can be specified in **one** of two ways:

- 1715 • by the qualified name of a type defined in an XML schema, using the **@type** attribute
- 1716 • by the qualified name of a global element in an XML schema, using the **@element** attribute

1717 The property type specified for the property element of a component MUST be compatible with the type of
1718 the property with the same @name declared in the component type of the implementation used by the
1719 component. If no type is declared in the component property element, the type of the property declared in
1720 the componentType of the implementation MUST be used. [ASM50036]

1721 The meaning of "compatible" for property types is defined in [the section Property Type Compatibility](#).

1722 Snippet 4-17 shows the component pseudo-schema with the pseudo-schema for a property child
1723 element:

1724

1725

1726

1727

1728

1729

1730

1731

1732

1733

1734

1735

1736

1737

1738

1739

1740

1741

1742

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component Property schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
...
  <component ... >*
    <implementation ... />?
    <service ... />*
    <reference ... />*
    <property name="xs:NCName"
      (type="xs:QName" | element="xs:QName")?
      many="xs:boolean"?
      source="xs:string"? file="xs:anyURI"?
      value="xs:string"?>*
      [<value>+ | xs:any+ ]?
    </property>
  </component>
...
</composite>
```

1743 Snippet 4.17: component Pseudo-Schema with property Child Element

1744

1745 The **component property** element has the **attributes**:

1746

1747

1748

1749

1750

1751

1752

1753

1754

1755

1756

1757

1758

1759

1760

1761

1762

1763

1764

1765

1766

1767

1768

1769

1770

1771

1772

1773

- **name : NCName (1..1)** – the name of the property. The @name attribute of a property element of a <component/> MUST be unique amongst the property elements of that <component/>. [ASM50031] The @name attribute of a property element of a <component/> MUST match the @name attribute of a property element of the componentType of the <implementation/> child element of the component. [ASM50037]
- zero or one of (0..1):
 - **type : QName** – the type of the property defined as the qualified name of an XML schema type
 - **element : QName** – the type of the property defined as the qualified name of an XML schema global element – the type is the type of the global elementA single property element MUST NOT contain both a @type attribute and an @element attribute. [ASM50035]
- **source : string (0..1)** – an XPath expression pointing to a property of the containing composite from which the value of this component property is obtained.
- **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property. The value of the component property @file attribute MUST be a dereferencable URI to a file containing the value for the property. [ASM50045] The URI can be an absolute URI or a relative URI. For a relative URI, it is taken relative to the base of the contribution containing the composite in which the component is declared. For a description of the format of the file, see the section on Property Value File Format.
- **many : boolean (0..1)** – whether the property is single-valued (false) or multi-valued (true). Overrides the many specified for this property in the componentType of the implementation. The value can only be equal or further restrict, i.e. if the implementation specifies many true, then the component can say false. In the case of a multi-valued property, it is presented to the implementation as a Collection of property values. If many is not specified, it takes the value defined by the component type of the implementation used by the component.
- **value : string (0..1)** - the value of the property if the property is defined using a simple type.

The **component property** element has the **child element**:

- **value : any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a property that is defined using a XML Schema type. If a property is single-valued, the <value/>

1774 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST
1775 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

1776 4.6.1 Property Type Compatibility

1777 There are a number of situations where the declared type of a property element is matched with the
1778 declared type of another property element. These situations include:

- 1779 • Where a component <property/> sets a value for a property of an implementation, as declared in the
1780 componentType of the implementation
- 1781 • Where a component <property/> gets its value from the value of a composite <property/> by means
1782 of its @source attribute. This situation can also involve the @source attribute referencing a
1783 subelement of the composite <property/> value, in which case it is the type of the subelement which
1784 must be matched with the type of the component <property/>
- 1785 • Where the componentType of a composite used as an implementation is calculated and
1786 componentType <property/> elements are created for each composite <property/>

1787 In these cases where the types of two property elements are matched, the types declared for the two
1788 <property/> elements MUST be compatible [ASM50038]

1789 Two property types are compatible if they have the same XSD type (where declared as XSD types) or the
1790 same XSD global element (where declared as XSD global elements). For cases where the type of a
1791 property is declared using a different type system (eg Java), then the type of the property is mapped to
1792 XSD using the mapping rules defined by the appropriate implementation type specification

1793 4.6.2 Property Value File Format

1794 The format of the file which is referenced by the @file attribute of a component property or a
1795 componentType property is that it is an XML document which MUST contain an sca:values element which
1796 in turn contains one of:

- 1797 • a set of one or more <sca:value/> elements each containing a simple string - where the property
1798 type is a simple XML type
- 1799 • a set of one or more <sca:value/> elements or a set of one or more global elements - where the
1800 property type is a complex XML type

1801 [ASM50046]

1802

```
1803 <?xml version="1.0" encoding="UTF-8"?>  
1804 <values>  
1805   <value>MyValue</value>  
1806 </values>
```

1807 *Snippet 4-17: Property Value File Content for simple property type*

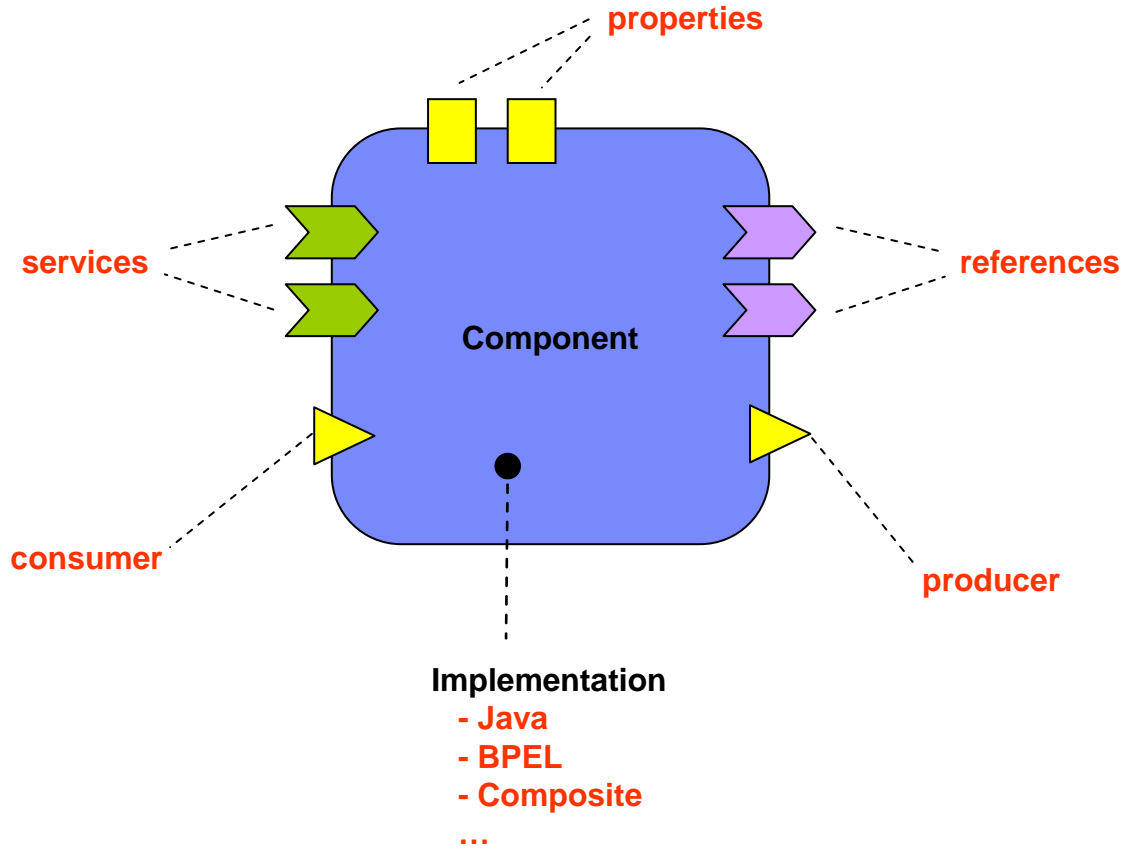
1808

```
1809 <?xml version="1.0" encoding="UTF-8"?>  
1810 <values>  
1811   <foo:fooElement>  
1812     <foo:a>AValue</foo:a>  
1813     <foo:b>InterestingURI</foo:b>  
1814   </foo:fooElement>  
1815 </values/>
```

1816 *Snippet 4-18: Property Value File Content for a complex property type*

1817 4.7 Example Component

1818 Figure 4-2 shows the **component symbol** that is used to represent a component in an assembly
1819 diagram.



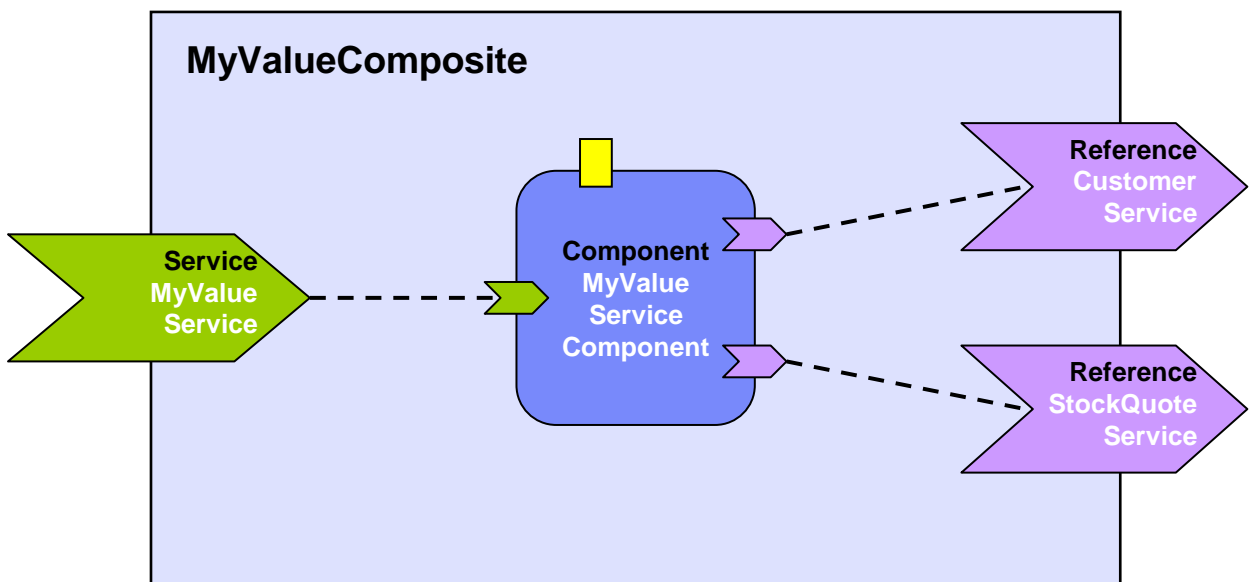
1820

1821 *Figure 4-2: Component symbol*

1822 Figure 4-3 shows the assembly diagram for the MyValueComposite containing the
 1823 MyValueServiceComponent.

1824 (TODO: modify the figure/example to include pub/sub/channels)

1825



1826

1827

1828 *Figure 4-3: Assembly diagram for MyValueComposite*

1829 Snippet 4-19 shows the MyValueComposite.composite file for the MyValueComposite containing
1830 the component element for the MyValueServiceComponent. A value is set for the property named
1831 currency, and the customerService and stockQuoteService references are promoted:

```
1832 <?xml version="1.0" encoding="ASCII"?>
1833 <!-- MyValueComposite_1 example -->
1834 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1835           targetNamespace="http://foo.com"
1836           name="MyValueComposite" >
1837
1838   <service name="MyValueService" promote="MyValueServiceComponent"/>
1839
1840   <component name="MyValueServiceComponent">
1841     <implementation.java
1842       class="services.myvalue.MyValueServiceImpl"/>
1843     <property name="currency">EURO</property>
1844     <reference name="customerService"/>
1845     <reference name="stockQuoteService"/>
1846   </component>
1847
1848   <reference name="CustomerService"
1849     promote="MyValueServiceComponent/customerService"/>
1850
1851   <reference name="StockQuoteService"
1852     promote="MyValueServiceComponent/stockQuoteService"/>
1853
1854 </composite>
```

1855 *Snippet 4-19: Example composite*

1856

1857 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of clarity
1858 – the references are defined by the MyValueServiceImpl implementation and there is no need to
1859 redeclare them on the component unless the intention is to wire them or to override some aspect of them.

1860 Snippet 4-20 gives an example of the layout of a composite file if both the currency property and the
1861 customerService reference of the MyValueServiceComponent are declared to be multi-valued (many=true
1862 for the property and multiplicity=0..n or 1..n for the reference):

```
1863 <?xml version="1.0" encoding="ASCII"?>
1864 <!-- MyValueComposite_2 example -->
1865 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1866           targetNamespace="http://foo.com"
1867           name="MyValueComposite" >
1868
1869   <service name="MyValueService" promote="MyValueServiceComponent"/>
1870
1871   <component name="MyValueServiceComponent">
1872     <implementation.java
1873       class="services.myvalue.MyValueServiceImpl"/>
1874     <property name="currency">
1875       <value>EURO</value>
1876       <value>Yen</value>
1877       <value>USDollar</value>
1878     </property>
1879     <reference name="customerService"
1880       target="InternalCustomer/customerService"/>
1881     <reference name="stockQuoteService"/>
1882   </component>
1883
1884   ...
1885
```

```
1886 <reference name="CustomerService"
1887     promote="MyValueServiceComponent/customerService"/>
1888
1889 <reference name="StockQuoteService"
1890     promote="MyValueServiceComponent/stockQuoteService"/>
1891
1892 </composite>
```

1893 *Snippet 4-20: Example composite with Multi-Valued property and reference*

1894

1895this assumes that the composite has another component called InternalCustomer (not shown) which
1896 has a service to which the customerService reference of the MyValueServiceComponent is wired as well
1897 as being promoted externally through the composite reference CustomerService.

5 Composite

1898

1899 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of
1900 composition within an SCA Domain. An **SCA composite** contains a set of components, channels,
1901 consumers, producers, services, references and the wires that interconnect them, plus a set of properties
1902 which can be used to configure components.

1903 Composites can be used as **component implementations** in higher-level composites – in other words
1904 the higher-level composites can have components that are implemented by composites. For more detail
1905 on the use of composites as component implementations see the section [Using Composites as](#)
1906 [Component Implementations](#).

1907 The content of a composite can be used within another composite through **inclusion**. When a composite
1908 is included by another composite, all of its contents are made available for use within the including
1909 composite – the contents are fully visible and can be referenced by other elements within the including
1910 composite. For more detail on the inclusion of one composite into another see the section [Using](#)
1911 [Composites through Inclusion](#).

1912 A composite can be used as a unit of deployment. When used in this way, composites contribute
1913 components and wires to an SCA Domain. A composite can be deployed to the SCA Domain either by
1914 inclusion or a composite can be deployed to the Domain as an implementation. For more detail on the
1915 deployment of composites, see the section dealing with the [SCA Domain](#).

1916 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite** element.
1917 Snippet 5-1 shows the pseudo-schema for the composite element:

1918

```
1919 <?xml version="1.0" encoding="ASCII"?>  
1920 <!-- Composite schema snippet -->  
1921 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
1922     targetNamespace="xs:anyURI"  
1923     name="xs:NCName" local="xs:boolean"?  
1924     autowire="xs:boolean"?  
1925     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1926  
1927     <include ... />*  
1928  
1929     <requires/>*  
1930     <policySetAttachment/>*  
1931  
1932     <service ... />*  
1933     <reference ... />*  
1934  
1935     <channel ... />*  
1936     <consumer ... />*  
1937     <producer ... />*  
1938  
1939     <property ... />*  
1940  
1941     <component ... />*  
1942  
1943     <wire ... />*  
1944  
1945 </composite>
```

1946 *Snippet 5-1: composite Pseudo-Schema*

1947

1948 The **composite** element has the **attributes**:

- 1949 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is an XML
1950 QName, in the namespace identified by the @targetNamespace attribute. A composite @name
1951 attribute value MUST be unique within the namespace of the composite. [ASM60001]
- 1952 • **targetNamespace : anyURI (1..1)** – an identifier for a target namespace into which the composite is
1953 declared
- 1954 • **local : boolean (0..1)** – whether all the components within the composite all run in the same
1955 operating system process. @local="true" for a composite means that all the components within the
1956 composite MUST run in the same operating system process. [ASM60002] local="false", which is the
1957 default, means that different components within the composite can run in different operating system
1958 processes and they can even run on different nodes on a network.
- 1959 • **autowire : boolean (0..1)** – whether contained component references are autowired, as described in
1960 the [Autowire section](#). Default is false.
- 1961 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1962 [\[SCA-POLICY\]](#) for a description of this attribute.
- 1963 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1964 [\[SCA-POLICY\]](#) for a description of this attribute.

1965 The **composite** element has the **child elements**:

- 1966 • **service : CompositeService (0..n)** – see composite service section.
- 1967 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1968 • **channel: Channel (0..n)** – see channel section.
- 1969 • **consumer : CompositeConsumer (0..n)** – see composite consumer section.
- 1970 • **producer: CompositeProducer (0..n)** – see composite producer section
- 1971 • **property : CompositeProperty (0..n)** – see composite property section.
- 1972 • **component : Component (0..n)** – see component section.
- 1973 • **wire : Wire (0..n)** – see composite wire section.
- 1974 • **include : Include (0..n)** – see composite include section
- 1975 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1976 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1977 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1978 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
1979 description of this element.

1980 Components contain configured implementations which hold the business logic of the composite. The
1981 components offer services and use references to other services and they send out events via producers
1982 and receive events through consumers.

1983 **Composite services** define the public services provided by the composite, which can be accessed from
1984 outside the composite. **Composite references** represent dependencies which the composite has on
1985 services provided elsewhere, outside the composite. Wires describe the connections between component
1986 services and component references within the composite. Included composites contribute the elements
1987 they contain to the using composite.

1988 Composite services involve the **promotion** of one service of one of the components within the composite,
1989 which means that the composite service is actually provided by one of the components within the
1990 composite. Composite references involve the **promotion** of one or more references of one or more
1991 components. Multiple component references can be promoted to the same composite reference, as long
1992 as each of the component references has an interface that is a compatible subset of the interface on the
1993 composite reference. Where multiple component references are promoted to the same composite
1994 reference, then they all share the same configuration, including the same target service(s).

1995 Composite services and composite references can use the configuration of their promoted services and
1996 references respectively (such as Bindings and Policy Sets). Alternatively composite services and

1997 composite references can override some or all of the configuration of the promoted services and
 1998 references, through the configuration of bindings and other aspects of the composite service or reference.
 1999 Component services and component references can be promoted to composite services and references
 2000 and also be wired internally within the composite at the same time. For a reference, this only makes
 2001 sense if the reference supports a multiplicity greater than 1.
 2002 Channels within the composite represent intermediaries transmitting events from producers to consumers
 2003 entirely within the composite. Composite consumers define public locations where events are received
 2004 from outside the composite. Composite producers represent places where the composite as a whole
 2005 sends out events. Composite consumers involve the **promotion** of one or more contained component
 2006 consumers. Composite producers involve the **promotion** of one or more contained component
 2007 producers.
 2008 Component producers can be promoted to composite producers and can be configured to send events to
 2009 other targets at the same time. Similarly, component consumers can be promoted to composite
 2010 consumers and can be configured to receive events from other sources at the same time.

2011 5.1 Service

2012 The **services of a composite** are defined by promoting services defined by components contained in the
 2013 composite. A component service is promoted by means of a composite **service element**.
 2014 A composite service is represented by a **service element** which is a child of the composite element.
 2015 There can be **zero or more** service elements in a composite. Snippet 5-2 shows the composite pseudo-
 2016 schema with the pseudo-schema for a service child element:
 2017

```

2018 <?xml version="1.0" encoding="ASCII"?>
2019 <!-- Composite Service schema snippet -->
2020 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
2021 ...
2022   <service name="xs:NCName" promote="xs:anyURI"
2023     requires="list of xs:QName"? policySets="list of xs:QName"?>*
2024     <interface ... />?
2025     <binding ... />*
2026     <callback?
2027       <binding ... />+
2028     </callback>
2029     <requires/>*
2030     <policySetAttachment/>*
2031   </service>
2032   ...
2033 </composite>
  
```

2034 *Snippet 5-2: composite Pseudo-Schema with service Child Element*

2035

2036 The **composite service** element has the **attributes**:

- 2037 • **name : NCName (1..1)** – the name of the service. The name of a composite `<service/>` element
 2038 **MUST be unique across all the composite services in the composite.** [ASM60003] The name of the
 2039 composite service can be different from the name of the promoted component service.
- 2040 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form `<component-`
 2041 `name/><service-name>`. The service name can be omitted if the target component only has one
 2042 service. The same component service can be promoted by more than one composite service. A
 2043 composite `<service/>` element's `@promote` attribute **MUST identify one of the component services**
 2044 **within that composite.** [ASM60004] `<include/>` processing **MUST take place before the processing of**
 2045 **the `@promote` attribute of a composite service is performed.** [ASM60038]
- 2046 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
 2047 [\[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or further qualify the required
 2048 intents defined by the promoted component service.

- 2049 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
2050 [\[SCA-POLICY\]](#) for a description of this attribute.

2051 The **composite service** element has the **child elements**, whatever is not specified is defaulted from the
2052 promoted component service.

- 2053 • **interface : Interface (0..1)** - an interface which describes the operations provided by the composite
2054 service. If a composite service interface is specified it **MUST** be the same or a compatible subset of
2055 the interface provided by the promoted component service. [\[ASM60005\]](#) The interface is described by
2056 **zero or one interface element** which is a child element of the service element. For details on the
2057 interface element see [the Interface section](#).

- 2058 • **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the
2059 promoted component service from the composite service perspective. The bindings defined on the
2060 component service are still in effect for local wires within the composite that target the component
2061 service. A service element has zero or more **binding elements** as children. Details of the binding
2062 element are described in the [Bindings section](#). For more details on wiring see [the Wiring section](#).

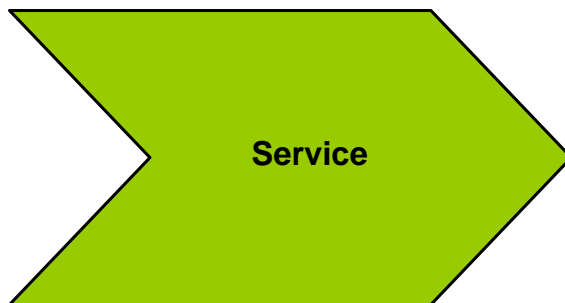
- 2063 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
2064 defined and the callback has one or more **binding** elements as subelements. The **callback** and its
2065 binding subelements are specified if there is a need to have binding details used to handle callbacks.
2066 Callback binding elements attached to the composite service override any callback binding elements
2067 defined on the promoted component service. If the callback element is not present on the composite
2068 service, any callback binding elements on the promoted service are used. If the callback element is
2069 not present at all, the behaviour is runtime implementation dependent.

- 2070 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2071 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

- 2072 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
2073 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2074 description of this element.

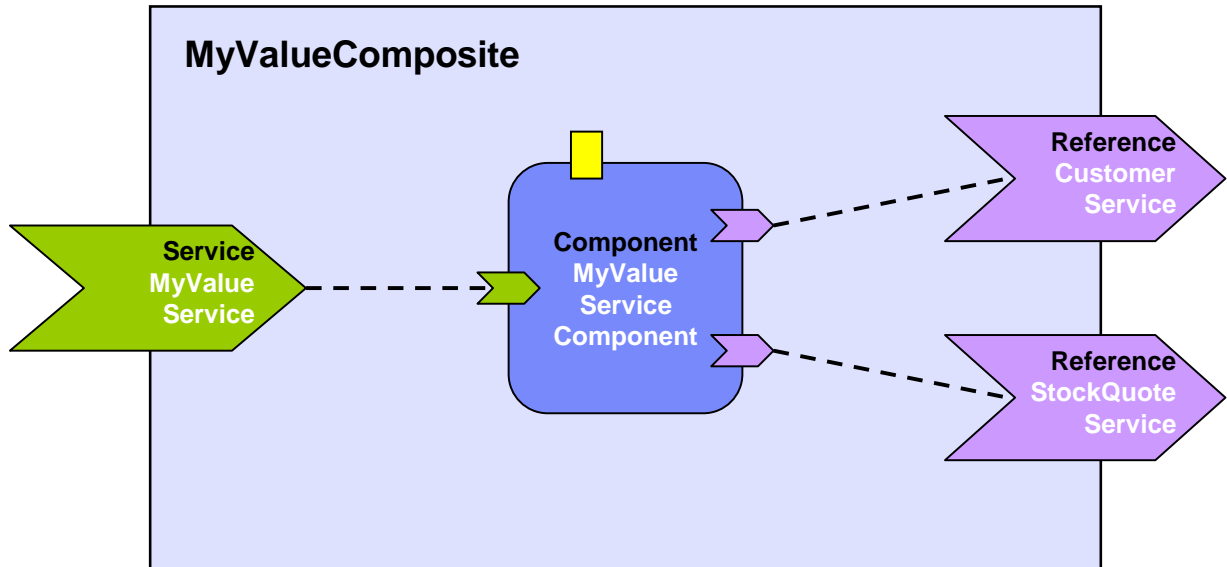
2075 5.1.1 Service Examples

2076 Figure 5-1 shows the service symbol that used to represent a service in an assembly diagram:



2077
2078 *Figure 5-1: Service symbol*

2079
2080 Figure 5-2 shows the assembly diagram for the MyValueComposite containing the service
2081 MyValueService.



2082
2083 *Figure 5-2: MyValueComposite showing Service*

2084
2085 Snippet 5-3 shows the MyValueComposite.composite file for the MyValueComposite containing the
2086 service element for the MyValueService, which is a promote of the service offered by the
2087 MyValueServiceComponent. The name of the promoted service is omitted since
2088 MyValueServiceComponent offers only one service. The composite service MyValueService is bound
2089 using a Web service binding.

```

2090
2091 <?xml version="1.0" encoding="ASCII"?>
2092 <!-- MyValueComposite_4 example -->
2093 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2094           targetNamespace="http://foo.com"
2095           name="MyValueComposite" >
2096
2097   ...
2098
2099   <service name="MyValueService" promote="MyValueServiceComponent">
2100     <interface.java interface="services.myvalue.MyValueService"/>
2101     <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
2102               wsdl.port(MyValueService/MyValueServiceSOAP)"/>
2103   </service>
2104
2105   <component name="MyValueServiceComponent">
2106     <implementation.java
2107       class="services.myvalue.MyValueServiceImpl"/>
2108     <property name="currency">EURO</property>
2109     <service name="MyValueService"/>
2110     <reference name="customerService"/>
2111     <reference name="stockQuoteService"/>
2112   </component>
2113
2114   ...
2115
2116 </composite>

```

2117 *Snippet 5-3: Example composite with a service*

2118 5.2 Reference

2119 The **references of a composite** are defined by **promoting** references defined by components contained
2120 in the composite. Each promoted reference indicates that the component reference needs to be resolved
2121 by services outside the composite. A component reference is promoted using a composite **reference**
2122 **element**.

2123 A composite reference is represented by a **reference element** which is a child of a composite element.
2124 There can be **zero or more reference** elements in a composite. Snippet 5-4 shows the composite
2125 pseudo-schema with the pseudo-schema for a **reference** element:

2126

```
2127 <?xml version="1.0" encoding="ASCII"?>
2128 <!-- Composite Reference schema snippet -->
2129 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
2130 ...
2131   <reference name="xs:NCName" target="list of xs:anyURI"?
2132     promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
2133     multiplicity="0..1 or 1..1 or 0..n or 1..n"
2134     requires="list of xs:QName"? policySets="list of xs:QName"?>*
2135     <interface ... />?
2136     <binding ... />*
2137     <callback>?
2138       <binding ... />+
2139     </callback>
2140     <requires/>*
2141     <policySetAttachment/>*
2142   </reference>
2143   ...
2144 </composite>
```

2145 *Snippet 5-4: composite Pseudo-Schema with reference Child Element*

2146

2147 The **composite reference** element has the **attributes**:

- 2148 • **name : NCName (1..1)** – the name of the reference. The name of a composite <reference/> element
2149 **MUST be unique across all the composite references in the composite.** [ASM60006] The name of the
2150 composite reference can be different than the name of the promoted component reference.
- 2151 • **promote : anyURI (1..n)** – identifies one or more promoted component references. The value is a list
2152 of values of the form <component-name>/<reference-name> separated by spaces. The reference
2153 name can be omitted if the component has only one reference. Each of the URIs declared by a
2154 composite reference's @promote attribute **MUST identify a component reference within the**
2155 **composite.** [ASM60007] <include/> processing **MUST take place before the processing of the**
2156 **@promote attribute of a composite reference is performed.** [ASM60037]

2157 The same component reference can be promoted more than once, using different composite
2158 references, but only if the multiplicity defined on the component reference is 0..n or 1..n. The
2159 multiplicity on the composite reference can restrict accordingly.

2160 Where a composite reference promotes two or more component references:

- 2161 – **the interfaces of the component references promoted by a composite reference MUST be the**
2162 **same, or if the composite reference itself declares an interface then each of the component**
2163 **reference interfaces MUST be a compatible subset of the composite reference interface..**
2164 [ASM60008]
- 2165 – **the intents declared on a composite reference and on the component references which it**
2166 **promotes MUST NOT be mutually exclusive.** [ASM60009] The intents which apply to the
2167 composite reference in this case are the union of the intents specified for each of the promoted
2168 component references plus any intents declared on the composite reference itself. **If any intents**
2169 **in the set which apply to a composite reference are mutually exclusive then the SCA runtime**
2170 **MUST raise an error.** [ASM60010]

- 2171 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or further qualify the intents defined for the promoted component reference.
- 2172
- 2173
- 2174 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this attribute.
- 2175
- 2176 • **multiplicity : (1..1)** - Defines the number of wires that can connect the reference to target services.
- 2177 The multiplicity of a composite reference is always specified explicitly and can have one of the
- 2178 following values
 - 2179 – 0..1 – zero or one wire can have the reference as a source
 - 2180 – 1..1 – one wire can have the reference as a source
 - 2181 – 0..n - zero or more wires can have the reference as a source
 - 2182 – 1..n – one or more wires can have the reference as a source

2183 The multiplicity of a composite reference MUST be equal to or further restrict the multiplicity of each
 2184 of the component references that it promotes, with the exception that the multiplicity of the composite
 2185 reference does not have to require a target if there is already a target on the component reference.
 2186 This means that a component reference with multiplicity 1..1 and a target can be promoted by a
 2187 composite reference with multiplicity 0..1, and a component reference with multiplicity 1..n and one or
 2188 more targets can be promoted by a composite reference with multiplicity 0..n or 0..1. [ASM60011]

2189 The valid values for composite reference multiplicity are shown in the following tables:
 2190

Composite Reference multiplicity	Component Reference multiplicity (where there are no targets declared)			
	0..1	1..1	0..n	1..n
0..1	YES	NO	YES	NO
1..1	YES	YES	YES	YES
0..n	NO	NO	YES	NO
1..n	NO	NO	YES	YES

2191

Composite Reference multiplicity	Component Reference multiplicity (where there are targets declared)			
	0..1	1..1	0..n	1..n
0..1	YES	YES	YES	YES
1..1	YES	YES	YES	YES
0..n	NO	NO	YES	YES
1..n	NO	NO	YES	YES

2192

- 2193 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on multiplicity setting.
- 2194 Each value wires the reference to a service in a composite that uses the composite containing the

2195 reference as an implementation for one of its components. For more details on wiring see [the section](#)
2196 [on Wires](#).

2197 • **wiredByImpl : boolean (0..1)** – a boolean value. If set to "true" it indicates that the target of the
2198 reference is set at runtime by the implementation code (for example by the code obtaining an
2199 endpoint reference by some means and setting this as the target of the reference through the use of
2200 programming interfaces defined by the relevant Client and Implementation specification). If "true" is
2201 set, then the reference is not intended to be wired statically within a using composite, but left unwired.
2202 All the component references promoted by a single composite reference MUST have the same value
2203 for @wiredByImpl. [ASM60035] If the @wiredByImpl attribute is not specified on the composite
2204 reference, the default value is "true" if all of the promoted component references have a wiredByImpl
2205 value of "true", and the default value is "false" if all the promoted component references have a
2206 wiredByImpl value of "false". If the @wiredByImpl attribute is specified, its value MUST be "true" if all
2207 of the promoted component references have a wiredByImpl value of "true", and its value MUST be
2208 "false" if all the promoted component references have a wiredByImpl value of "false". [ASM60036]

2209 The **composite reference** element has the **child elements**, whatever is not specified is
2210 defaulted from the promoted component reference(s).

2211 • **interface : Interface (0..1) - zero or one interface element** which declares an interface for the
2212 composite reference. If a composite reference has an interface specified, it MUST provide an
2213 interface which is the same or which is a compatible superset of the interface(s) declared by the
2214 promoted component reference(s). [ASM60012] If no interface is declared on a composite reference,
2215 the interface from one of its promoted component references MUST be used for the component type
2216 associated with the composite. [ASM60013] For details on the interface element see [the Interface](#)
2217 [section](#).

2218 • **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as children. If
2219 one or more **bindings** are specified they **override** any and all of the bindings defined for the
2220 promoted component reference from the composite reference perspective. The bindings defined on
2221 the component reference are still in effect for local wires within the composite that have the
2222 component reference as their source. Details of the binding element are described in the [Bindings](#)
2223 [section](#). For more details on wiring see [the section on Wires](#).

2224 A reference identifies zero or more target services which satisfy the reference. This can be done in a
2225 number of ways, which are fully described in section "[Specifying the Target Service\(s\) for a](#)
2226 [Reference](#)".

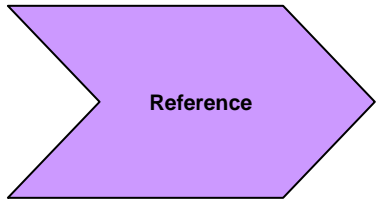
2227 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
2228 defined and the callback element has one or more **binding** elements as subelements. The **callback**
2229 and its binding subelements are specified if there is a need to have binding details used to handle
2230 callbacks. Callback binding elements attached to the composite reference override any callback
2231 binding elements defined on any of the promoted component references. If the callback element is
2232 not present on the composite service, any callback binding elements that are declared on all the
2233 promoted references are used. If the callback element is not present at all, the behaviour is runtime
2234 implementation dependent.

2235 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2236 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

2237 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
2238 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2239 description of this element.

2240 5.2.1 Example Reference

2241 Figure 5-3 shows the reference symbol that is used to represent a reference in an assembly diagram.



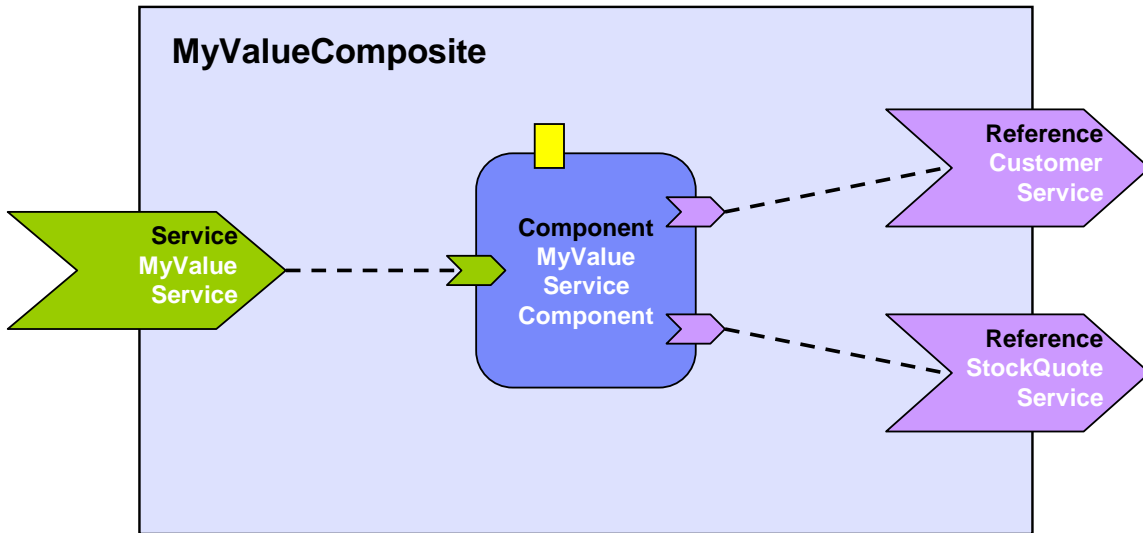
2242

2243 *Figure 5-3: Reference symbol*

2244

2245 Figure 5-4 shows the assembly diagram for the MyValueComposite containing the reference
 2246 CustomerService and the reference StockQuoteService.

2247



2248

2249 *Figure 5-4: MyValueComposite showing References*

2250

2251 Snippet 5-5 shows the MyValueComposite.composite file for the MyValueComposite containing the
 2252 reference elements for the CustomerService and the StockQuoteService. The reference CustomerService
 2253 is bound using the SCA binding. The reference StockQuoteService is bound using the Web service
 2254 binding. The endpoint addresses of the bindings can be specified, for example using the binding *@uri*
 2255 attribute (for details see the [Bindings](#) section), or overridden in an enclosing composite. Although in this
 2256 case the reference StockQuoteService is bound to a Web service, its interface is defined by a Java
 2257 interface, which was created from the WSDL portType of the target web service.

2258

```

2259 <?xml version="1.0" encoding="ASCII"?>
2260 <!-- MyValueComposite_3 example -->
2261 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2262           targetNamespace="http://foo.com"
2263           name="MyValueComposite" >
2264
2265   ...
2266
2267   <component name="MyValueServiceComponent">
2268     <implementation.java
2269       class="services.myvalue.MyValueServiceImpl"/>
2270     <property name="currency">EURO</property>
2271     <reference name="customerService"/>
2272     <reference name="stockQuoteService"/>
2273   </component>
2274
  
```

```

2275 <reference name="CustomerService"
2276     promote="MyValueServiceComponent/customerService">
2277     <interface.java interface="services.customer.CustomerService"/>
2278     <!-- The following forces the binding to be binding.sca -->
2279     <!-- whatever is specified by the component reference or -->
2280     <!-- by the underlying implementation -->
2281     <binding.sca/>
2282 </reference>
2283
2284 <reference name="StockQuoteService"
2285     promote="MyValueServiceComponent/stockQuoteService">
2286     <interface.java
2287         interface="services.stockquote.StockQuoteService"/>
2288     <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
2289         wsdl.port(StockQuoteService/StockQuoteServiceSOAP)"/>
2290 </reference>
2291
2292     ...
2293
2294 </composite>

```

2295 *Snippet 5-5: Example composite with a reference*

2296 5.3 Property

2297 **Properties** allow for the configuration of an implementation with externally set data values. A composite
2298 can declare zero or more properties. Each property has a type, which is either simple or complex. An
2299 implementation can also define a default value for a property. Properties can be configured with values in
2300 the components that use the implementation.

2301 Snippet 5-6 shows the composite pseudo-schema with the pseudo-schema for a **reference** element:

2302

```

2303 <?xml version="1.0" encoding="ASCII"?>
2304 <!-- Composite Property schema snippet -->
2305 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
2306     ...
2307     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
2308         many="xs:boolean"? mustSupply="xs:boolean"?*
2309         default-property-value?
2310     </property>
2311     ...
2312 </composite>

```

2313 *Snippet 5-6: composite Pseudo-Schema with property Child Element*

2314

2315 The **composite property** element has the **attributes**:

- 2316 • **name : NCName (1..1)** - the name of the property. The @name attribute of a composite property
2317 MUST be unique amongst the properties of the same composite. [ASM60014]
- 2318 • one of (1..1):
 - 2319 – **type : QName** – the type of the property - the qualified name of an XML schema type
 - 2320 – **element : QName** – the type of the property defined as the qualified name of an XML schema
2321 global element – the type is the type of the global element

2322 A single property element MUST NOT contain both a @type attribute and an @element
2323 attribute. [ASM60040]

- 2324 • **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). The
2325 default is **false**. In the case of a multi-valued property, it is presented to the implementation as a
2326 collection of property values.

- 2327 • **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the component that
2328 uses the composite – when mustSupply="true" the component has to supply a value since the
2329 composite has no default value for the property. A default-property-value is only worth declaring
2330 when mustSupply="false" (the default setting for the @mustSupply attribute), since the implication of
2331 a default value is that it is used only when a value is not supplied by the using component.

2332 The property element can contain a **default-property-value**, which provides default value for the
2333 property. The form of the default property value is as described in the section on [Component Property](#).

2334 Implementation types other than **composite** can declare properties in an implementation-dependent form
2335 (e.g. annotations within a Java class), or through a property declaration of exactly the form described
2336 above in a componentType file.

2337 Property values can be configured when an implementation is used by a component. The form of the
2338 property configuration is shown in the section on [Components](#).

2339 5.3.1 Property Examples

2340 For the example Property declaration and value setting in Snippet 5-8, the complex type in Snippet 5-7 is
2341 used as an example:

2342

```
2343 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"  
2344             targetNamespace="http://foo.com/"  
2345             xmlns:tns="http://foo.com/">  
2346   <!-- ComplexProperty schema -->  
2347   <xsd:element name="fooElement" type="tns:MyComplexType"/>  
2348   <xsd:complexType name="MyComplexType">  
2349     <xsd:sequence>  
2350       <xsd:element name="a" type="xsd:string"/>  
2351       <xsd:element name="b" type="xsd:anyURI"/>  
2352     </xsd:sequence>  
2353     <attribute name="attr" type="xsd:string" use="optional"/>  
2354   </xsd:complexType>  
2355 </xsd:schema>
```

2356 *Snippet 5-7: Complex Type for Snippet 5-8*

2357

2358 The composite in Snippet 5-8 demonstrates the declaration of a property of a complex type, with a default
2359 value, plus it demonstrates the setting of a property value of a complex type within a component:

2360

```
2361 <?xml version="1.0" encoding="ASCII"?>  
2362 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
2363           xmlns:foo="http://foo.com"  
2364           targetNamespace="http://foo.com"  
2365           name="AccountServices">  
2366   <!-- AccountServices Example1 -->  
2367   ...  
2368   <property name="complexFoo" type="foo:MyComplexType">  
2369     <value>  
2370       <foo:a>AValue</foo:a>  
2371       <foo:b>InterestingURI</foo:b>  
2372     </value>  
2373   </property>  
2374   <component name="AccountServiceComponent">  
2375     <implementation.java class="foo.AccountServiceImpl"/>  
2376     <property name="complexBar" source="$complexFoo"/>  
2377     <reference name="accountDataService"  
2378               target="AccountDataServiceComponent"/>  
2379   </component>  
2380 </composite>  
2381
```

```

2382         <reference name="stockQuoteService" target="StockQuoteService"/>
2383     </component>
2384     ...
2385
2386 </composite>
2387

```

2388 *Snippet 5-8: Example property with a Complex Type*

2389

2390 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the property is
 2391 defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the composite and it
 2392 references the example XSD, where MyComplexType is defined. The declaration of complexFoo
 2393 contains a default value. This is declared as the content of the property element. In this example, the
 2394 default value consists of the element **value** which is of type foo:MyComplexType and it has two child
 2395 elements <foo:a> and <foo:b>, following the definition of MyComplexType.

2396 In the component **AccountServiceComponent**, the component sets the value of the property
 2397 **complexBar**, declared by the implementation configured by the component. In this case, the type of
 2398 complexBar is foo:MyComplexType. The example shows that the value of the complexBar property is set
 2399 from the value of the complexFoo property – the **@source** attribute of the property element for
 2400 complexBar declares that the value of the property is set from the value of a property of the containing
 2401 composite. The value of the @source attribute is **\$complexFoo**, where complexFoo is the name of a
 2402 property of the composite. This value implies that the whole of the value of the source property is used to
 2403 set the value of the component property.

2404 Snippet 5-9 illustrates the setting of the value of a property of a simple type (a string) from **part** of the
 2405 value of a property of the containing composite which has a complex type:

2406

```

2407 <?xml version="1.0" encoding="ASCII"?>
2408 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2409           xmlns:foo="http://foo.com"
2410           targetNamespace="http://foo.com"
2411           name="AccountServices">
2412   <!-- AccountServices Example2 -->
2413   ...
2414   ...
2415   <property name="complexFoo" type="foo:MyComplexType">
2416     <value>
2417       <foo:a>AValue</foo:a>
2418       <foo:b>InterestingURI</foo:b>
2419     </value>
2420   </property>
2421   <component name="AccountServiceComponent">
2422     <implementation.java class="foo.AccountServiceImpl"/>
2423     <property name="currency" source="$complexFoo/a"/>
2424     <reference name="accountDataService"
2425               target="AccountDataServiceComponent"/>
2426     <reference name="stockQuoteService" target="StockQuoteService"/>
2427   </component>
2428   ...
2429 </composite>
2430
2431
2432
2433

```

2434 *Snippet 5-9: Example property with a Simple Type*

2435

2436 In the example in Snippet 5-9, the component **AccountServiceComponent** sets the value of a property
 2437 called **currency**, which is of type string. The value is set from a property of the composite
 2438 **AccountServices** using the @source attribute set to **\$complexFoo/a**. This is an XPath expression that

2439 selects the property name **complexFoo** and then selects the value of the **a** subelement of the value of
2440 complexFoo. The "a" subelement is a string, matching the type of the currency property.

2441 Further examples of declaring properties and setting property values in a component:

2442 – Declaration of a property with a simple type and a default value:

```
2443 <property name="SimpleTypeProperty" type="xsd:string">  
2444 <value>MyValue</value>  
2445 </property>
```

2446 *Snippet 5-10: Example property with a Simple Type and Default Value*

2447

2448 – Declaration of a property with a complex type and a default value:

```
2449 <property name="complexFoo" type="foo:MyComplexType">  
2450 <value>  
2451 <foo:a>AValue</foo:a>  
2452 <foo:b>InterestingURI</foo:b>  
2453 </value>  
2454 </property>
```

2455 *Snippet 5-11: Example property with a Complex Type and Default Value*

2456

2457 – Declaration of a property with a global element type:

```
2458 <property name="elementFoo" element="foo:fooElement">  
2459 <foo:fooElement>  
2460 <foo:a>AValue</foo:a>  
2461 <foo:b>InterestingURI</foo:b>  
2462 </foo:fooElement>  
2463 </property>
```

2464 *Snippet 5-12: Example property with a Global Element Type*

2465 5.4 Wire

2466 SCA wires within a composite connect source component references to target component services.

2467 One way of defining a wire is by **configuring a reference of a component using its @target attribute**.

2468 The reference element is configured with the wire-target-URI of the service(s) that resolve the reference.

2469 Multiple target services are valid when the reference has a multiplicity of 0..n or 1..n.

2470 An alternative way of defining a Wire is by means of a **wire element** which is a child of the composite
2471 element. There can be **zero or more** wire elements in a composite. This alternative method for defining
2472 wires is useful in circumstances where separation of the wiring from the elements the wires connect helps
2473 simplify development or operational activities. An example is where the components used to build a
2474 Domain are relatively static but where new or changed applications are created regularly from those
2475 components, through the creation of new assemblies with different wiring. Deploying the wiring
2476 separately from the components allows the wiring to be created or modified with minimum effort.

2477 Note that a Wire specified via a wire element is equivalent to a wire specified via the @target attribute of
2478 a reference. The rule which forbids mixing of wires specified with the @target attribute with the
2479 specification of endpoints in binding subelements of the reference also applies to wires specified via
2480 separate wire elements.

2481 Snippet 5-13 shows the composite pseudo-schema with the pseudo-schema for the wire child element:

2482

```
2483 <!-- Wires schema snippet -->  
2484 <composite ...>  
2485 ...  
2486 <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/*>  
2487 ...  
2488 </composite>
```

2489 Snippet 5-13: composite Pseudo-Schema with wire Child Element

2490

2491 The **reference element of a component** has a list of one or more of the following **wire-target-URI**
2492 values for the target, with multiple values separated by a space:

2493 • `<component-name>[/<service-name> /<binding-name>]?]?`

2494 ○ `<component-name>` is the name of the target component.

2495 ○ `<service-name>` is the name of the target service within the component.

2496 If `<service-name>` is present, the component service with `@name` corresponding
2497 to `<service-name>` MUST be used for the wire. [ASM60046]

2498 If there is no component service with `@name` corresponding to `<service-name>`,
2499 the SCA runtime MUST raise an error. [ASM60047]

2500 If `<service-name>` is not present, the target component MUST have one and only
2501 one service with an interface that is a compatible superset of the wire source's
2502 interface and satisfies the policy requirements of the wire source, and the SCA
2503 runtime MUST use this service for the wire. [ASM60048]

2504 ○ `<binding-name>` is the name of the service's binding to use. The `<binding-name>`
2505 can be the default name of a binding element (see section 8 "Binding").

2506

2507 If `<binding-name>` is present, the `<binding/>` subelement of the target service
2508 with `@name` corresponding to `<binding-name>` MUST be used for the wire.
2509 [ASM60049] If there is no `<binding/>` subelement of the target service with
2510 `@name` corresponding to `<binding-name>`, the SCA runtime MUST raise an error.
2511 [ASM60050] If `<binding-name>` is not present and the target service has multiple
2512 `<binding/>` subelements, the SCA runtime MUST choose one and only one of the
2513 `<binding/>` elements which satisfies the mutual policy requirements of the
2514 reference and the service, and the SCA runtime MUST use this binding for the
2515 wire. [ASM60051]

2516

2517 The **wire element** has the attributes:

2518 • **source (1..1)** – names the source component reference. The valid URI scheme is:

2519 – `<component-name>[/<reference-name>]?`

2520 • where the source is a component reference. The reference name can be omitted if the
2521 source component only has one reference

2522 • **target (1..1)** – names the target component service. The valid URI scheme is the same as the one
2523 defined for component references above.

2524 • **replace (0..1)** - a boolean value, with the default of "false". When a wire element has
2525 `@replace="false"`, the wire is added to the set of wires which apply to the reference identified by the
2526 `@source` attribute. When a wire element has `@replace="true"`, the wire is added to the set of wires
2527 which apply to the reference identified by the `@source` attribute - but any wires for that reference
2528 specified by means of the `@target` attribute of the reference are removed from the set of wires which
2529 apply to the reference.

2530 In other words, if any `<wire/>` element with `@replace="true"` is used for a particular reference, the
2531 value of the `@target` attribute on the reference is ignored - and this permits existing wires on the
2532 reference to be overridden by separate configuration, where the reference is on a component at the
2533 Domain level.

2534 `<include/>` processing MUST take place before the `@source` and `@target` attributes of a wire are
2535 resolved. [ASM60039]

2536 For a composite used as a component implementation, wires can only link sources and targets that are
2537 contained in the same composite (irrespective of which file or files are used to describe the composite).

2538 Wiring to entities outside the composite is done through services and references of the composite with
 2539 wiring defined by the next higher composite.

2540 The interface declared by the target of a wire **MUST** be a compatible superset of the interface declared by
 2541 the source of the wire. [ASM60043] See the section on Interface Compatibility for a definition of
 2542 "compatible superset".

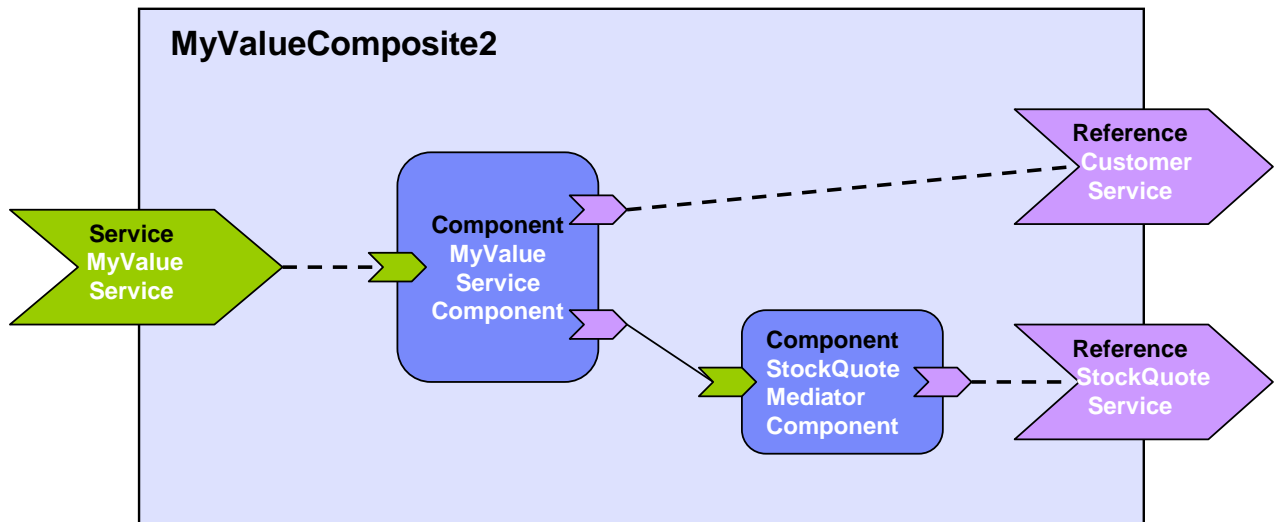
2543 A Wire can connect between different interface languages (e.g. Java interfaces and WSDL portTypes) in
 2544 either direction, as long as the operations defined by the two interface types are equivalent. They are
 2545 equivalent if the operation(s), parameter(s), return value(s) and faults/exceptions map to each other.

2546 Service clients cannot (portably) ask questions at runtime about additional interfaces that are provided by
 2547 the implementation of the service (e.g. the result of "instance of" in Java is non portable). It is valid for an
 2548 SCA implementation to have proxies for all wires, so that, for example, a reference object passed to an
 2549 implementation might only have the business interface of the reference and might not be an instance of
 2550 the (Java) class which is used to implement the target service, even where the interface is local and the
 2551 target service is running in the same process.

2552 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of an un-
 2553 wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA runtime is
 2554 encouraged to issue a warning.

2555 5.4.1 Wire Examples

2556 Figure 5-5 shows the assembly diagram for the MyValueComposite2 containing wires between service,
 2557 components and references.



2558
 2559 *Figure 5-5: MyValueComposite2 showing Wires*

2560

2561 Snippet 5-14 shows the MyValueComposite2.composite file for the MyValueComposite2 containing the
 2562 configured component and service references. The service MyValueService is wired to the
 2563 MyValueServiceComponent, using an explicit <wire/> element. The MyValueServiceComponent's
 2564 customerService reference is wired to the composite's CustomerService reference. The
 2565 MyValueServiceComponent's stockQuoteService reference is wired to the
 2566 StockQuoteMediatorComponent, which in turn has its reference wired to the StockQuoteService
 2567 reference of the composite.

```
2568
2569 <?xml version="1.0" encoding="ASCII"?>
2570 <!-- MyValueComposite Wires examples -->
2571 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2572           targetNamespace="http://foo.com"
2573           name="MyValueComposite2" >
```

```

2574 <service name="MyValueService" promote="MyValueServiceComponent">
2575   <interface.java interface="services.myvalue.MyValueService"/>
2576   <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
2577     wsdl.port(MyValueService/MyValueServiceSOAP)"/>
2578 </service>
2579
2580
2581 <component name="MyValueServiceComponent">
2582   <implementation.java
2583     class="services.myvalue.MyValueServiceImpl"/>
2584   <property name="currency">EURO</property>
2585   <service name="MyValueService"/>
2586   <reference name="customerService"/>
2587   <reference name="stockQuoteService"/>
2588 </component>
2589
2590 <wire source="MyValueServiceComponent/stockQuoteService"
2591   target="StockQuoteMediatorComponent"/>
2592
2593 <component name="StockQuoteMediatorComponent">
2594   <implementation.java class="services.myvalue.SQMediatorImpl"/>
2595   <property name="currency">EURO</property>
2596   <reference name="stockQuoteService"/>
2597 </component>
2598
2599 <reference name="CustomerService"
2600   promote="MyValueServiceComponent/customerService">
2601   <interface.java interface="services.customer.CustomerService"/>
2602   <binding.sca/>
2603 </reference>
2604
2605 <reference name="StockQuoteService"
2606   promote="StockQuoteMediatorComponent">
2607   <interface.java
2608     interface="services.stockquote.StockQuoteService"/>
2609   <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
2610     wsdl.port(StockQuoteService/StockQuoteServiceSOAP)"/>
2611 </reference>
2612
2613 </composite>

```

2614 *Snippet 5-14: Example composite with a wire*

2615 5.4.2 Autowire

2616 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.
2617 Autowire enables component references to be automatically wired to component services which will
2618 satisfy those references, without the need to create explicit wires between the references and the
2619 services. When the autowire feature is used, a component reference which is not promoted and which is
2620 not explicitly wired to a service within a composite is automatically wired to a target service within the
2621 same composite. Autowire works by searching within the composite for a service interface which
2622 matches the interface of the references.

2623 The autowire feature is not used by default. Autowire is enabled by the setting of an @autowire attribute
2624 to "true". Autowire is disabled by setting of the @autowire attribute to "false" The @autowire attribute can
2625 be applied to any of the following elements within a composite:

- 2626 • reference
- 2627 • component
- 2628 • composite

2629 Where an element does not have an explicit setting for the @autowire attribute, it inherits the setting from
2630 its parent element. Thus a reference element inherits the setting from its containing component. A

2631 component element inherits the setting from its containing composite. Where there is no setting on any
2632 level, `autowire="false"` is the default.

2633 As an example, if a composite element has `autowire="true"` set, this means that autowiring is enabled for
2634 all component references within that composite. In this example, autowiring can be turned off for specific
2635 components and specific references through setting `autowire="false"` on the components and references
2636 concerned.

2637 For each component reference for which `autowire` is enabled, the SCA runtime **MUST** search within the
2638 composite for target services which have an interface that is a compatible superset of the interface of the
2639 reference. [ASM60022]

2640 The intents, and policies applied to the service **MUST** be compatible with those on the reference when
2641 using `autowire` to wire a reference – so that wiring the reference to the service will not cause an error due
2642 to policy mismatch [ASM60024] (see the [Policy Framework specification \[SCA-POLICY\]](#) for details)

2643 If the search finds **1 or more** valid target service for a particular reference, the action taken depends on
2644 the multiplicity of the reference:

- 2645 • for an `autowire` reference with multiplicity `0..1` or `1..1`, the SCA runtime **MUST** wire the reference to
2646 one of the set of valid target services chosen from the set in a runtime-dependent fashion
2647 [ASM60025]
- 2648 • for an `autowire` reference with multiplicity `0..n` or `1..n`, the reference **MUST** be wired to all of the set of
2649 valid target services [ASM60026]

2650 If the search finds **no** valid target services for a particular reference, the action taken depends on the
2651 multiplicity of the reference:

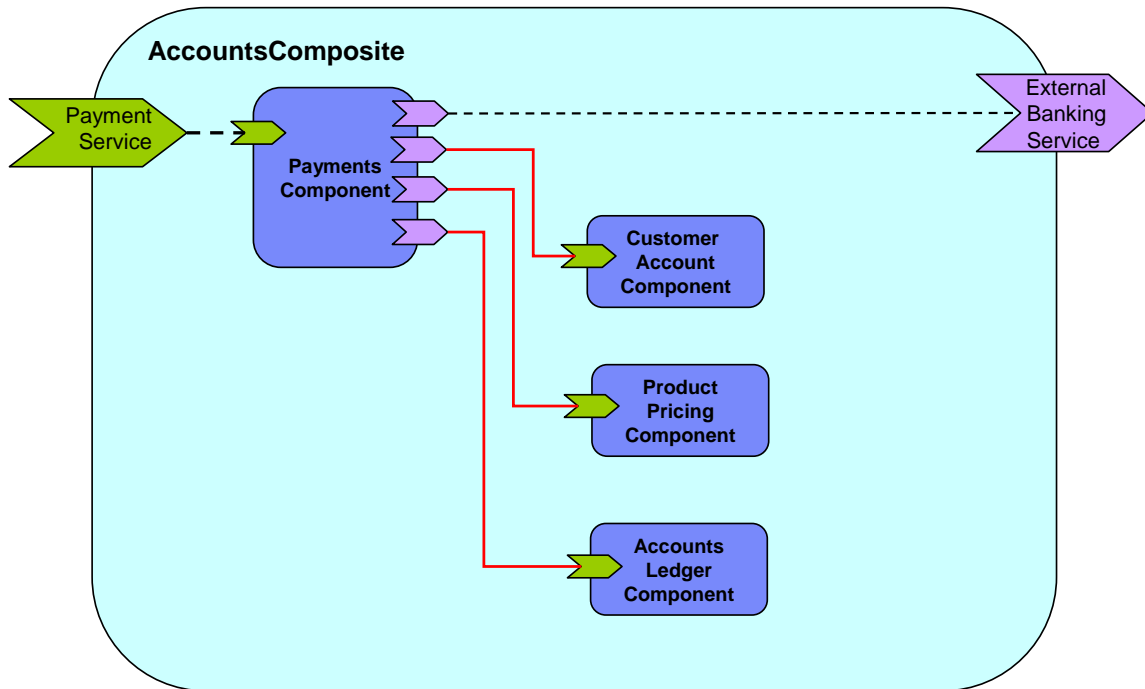
- 2652 • for an `autowire` reference with multiplicity `0..1` or `0..n`, if the SCA runtime finds no valid target service,
2653 there is no problem – no services are wired and the SCA runtime **MUST NOT** raise an error
2654 [ASM60027]
- 2655 • for an `autowire` reference with multiplicity `1..1` or `1..n`, if the SCA runtime finds no valid target services
2656 an error **MUST** be raised by the SCA runtime since the reference is intended to be wired [ASM60028]

2657 5.4.3 Autowire Examples

2658 Snippet 5-15 and Snippet 5-16 demonstrate two versions of the same composite – the first version is
2659 done using explicit wires, with no autowiring used, the second version is done using `autowire`. In both
2660 cases the end result is the same – the same wires connect the references to the services.

2661 Figure 5-6 is a diagram for the composite:

2662



2663
 2664 *Figure 5-6: Example Composite for Autowire*

2665
 2666 Snippet 5-15 is the composite using explicit wires:

2667
 2668
 2669
 2670
 2671
 2672
 2673
 2674
 2675
 2676
 2677
 2678
 2679
 2680
 2681
 2682
 2683
 2684
 2685
 2686
 2687
 2688
 2689
 2690
 2691
 2692
 2693
 2694
 2695
 2696
 2697
 2698
 2699

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Autowire Example - No autowire -->
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  xmlns:foo="http://foo.com"
  targetNamespace="http://foo.com"
  name="AccountComposite">

  <service name="PaymentService" promote="PaymentsComponent"/>

  <component name="PaymentsComponent">
    <implementation.java class="com.foo.accounts.Payments"/>
    <service name="PaymentService"/>
    <reference name="CustomerAccountService"
      target="CustomerAccountComponent"/>
    <reference name="ProductPricingService"
      target="ProductPricingComponent"/>
    <reference name="AccountsLedgerService"
      target="AccountsLedgerComponent"/>
    <reference name="ExternalBankingService"/>
  </component>

  <component name="CustomerAccountComponent">
    <implementation.java class="com.foo.accounts.CustomerAccount"/>
  </component>

  <component name="ProductPricingComponent">
    <implementation.java class="com.foo.accounts.ProductPricing"/>
  </component>

  <component name="AccountsLedgerComponent">
    <implementation.composite name="foo:AccountsLedgerComposite"/>
  </component>

```

```

2700     </component>
2701
2702     <reference name="ExternalBankingService"
2703         promote="PaymentsComponent/ExternalBankingService"/>
2704
2705 </composite>

```

2706 *Snippet 5-15: Example composite with Explicit wires*

2707

2708 Snippet 5-16 is the composite using autowire:

2709

```

2710 <?xml version="1.0" encoding="UTF-8"?>
2711 <!-- Autowire Example - With autowire -->
2712 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2713     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2714     xmlns:foo="http://foo.com"
2715     targetNamespace="http://foo.com"
2716     name="AccountComposite">
2717
2718     <service name="PaymentService" promote="PaymentsComponent">
2719         <interface.java class="com.foo.PaymentServiceInterface"/>
2720     </service>
2721
2722     <component name="PaymentsComponent" autowire="true">
2723         <implementation.java class="com.foo.accounts.Payments"/>
2724         <service name="PaymentService"/>
2725         <reference name="CustomerAccountService"/>
2726         <reference name="ProductPricingService"/>
2727         <reference name="AccountsLedgerService"/>
2728         <reference name="ExternalBankingService"/>
2729     </component>
2730
2731     <component name="CustomerAccountComponent">
2732         <implementation.java class="com.foo.accounts.CustomerAccount"/>
2733     </component>
2734
2735     <component name="ProductPricingComponent">
2736         <implementation.java class="com.foo.accounts.ProductPricing"/>
2737     </component>
2738
2739     <component name="AccountsLedgerComponent">
2740         <implementation.composite name="foo:AccountsLedgerComposite"/>
2741     </component>
2742
2743     <reference name="ExternalBankingService"
2744         promote="PaymentsComponent/ExternalBankingService"/>
2745
2746 </composite>

```

2747 *Snippet 5-16: composite of Snippet 5-15 Using autowire*

2748

2749 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires for
2750 any of its references – the wires are created automatically through autowire.

2751 **Note:** In the second example, it would be possible to omit all of the service and reference elements from
2752 the PaymentsComponent. They are left in for clarity, but if they are omitted, the component service and
2753 references still exist, since they are provided by the implementation used by the component.

2754 5.5 Consumer

2755 The **consumers of a composite** are defined by **promoting** consumers defined by components
2756 contained in the composite. Consumers are promoted by means of a composite **consumer element**,
2757 which is a child element of the composite element. Promotion of the component consumer allows the
2758 configuration of the composite consumer set by a higher level component to override the configuration of
2759 the lower component consumer. There can be **zero or more** consumer elements in a composite.

2760 Every event received by the composite consumer is sent on to all of the promoted consumers.

2761 Snippet 5-17 shows the pseudo-schema for a composite consumer element:

2762

```
2763 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
2764 ...  
2765  
2766 <consumer name="xs:NCName"  
2767     promote="list of xs:anyURI"  
2768     requires="list of xs:QName"?  
2769     policySets="list of xs:QName"?>*<br>  
2770     <filters/>?  
2771     <requires/>*<br>  
2772     <policySetAttachment/>*<br>  
2773 </consumer>  
2774 ...  
2775 </composite>
```

2776 Snippet 5-17: composite Pseudo-Schema with consumer Child Element

2777

2778 The consumer element has the following **attributes**:

- 2779 • **name: NCName (1..1)** – the name of the consumer. The name of the consumer **MUST** be
2780 **unique amongst the consumer elements of the composite**. [ASM60101] The name the
2781 composite consumer can be different from the name of the promoted component consumer.
- 2782 • **promote: listOfAnyURI (1..1)** – identifies the promoted consumers. The value is a list
2783 containing entries of the form componentName/consumerName. The consumer name is
2784 optional if the component only has one consumer. The same component consumer can be
2785 promoted by more than one composite consumer. A composite <consumer/> element's
2786 @promote attribute **MUST** identify one of the component consumers within that composite.
2787 [ASM60102] <include/> processing **MUST** take place before the processing of the @promote
2788 attribute of a composite service is performed. [ASM60103]
- 2789 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework](#)
2790 [specification \[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or
2791 further qualify the required intents defined by the promoted component consumer.
- 2792 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework](#)
2793 [specification \[SCA-POLICY\]](#) for a description of this attribute.

2794 The consumer element has the following **child elements** :

- 2795 • **filters: Filters (0..1)** – filter elements. See the section Filters: Selecting Subsets of Events for a
2796 detailed description of filters.
- 2797 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2798 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 2799 • **policySetAttachment : policySetAttachment (0..n)** - A producer element has **zero or more**
2800 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for
2801 a description of this element.

2802 5.6 Producer

2803 The **producers of a composite** are defined by **promoting** producers defined by components contained
2804 in the composite. Producers are promoted by means of a composite **producer element**, which is a child
2805 element of the composite element. Promotion of the component producer allows the configuration of the
2806 composite producer set by a higher level component to override the configuration of the lower component
2807 producer. There can be **zero or more** producer elements in a composite.

2808 Every event sent by any of the composite producer is sent out by the promoted producer.

2809 Snippet 5-18 shows the pseudo-schema for a composite producer element:

2810

```
2811 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
2812 ...  
2813  
2814 <producer name="xs:NCName"  
2815     promote="list of xs:anyURI"  
2816     requires="list of xs:QName"?  
2817     policySets="list of xs:QName"?>*<br>  
2818     <eventType/?<br>  
2819     <requires/?<br>  
2820     <policySetAttachment/?<br>  
2821 </producer>  
2822  
2823 ...  
2824 </composite>
```

2825 *Snippet 5-18: composite Pseudo-Schema with producer Child Element*

2826

2827 The producer element has the following **attributes**:

- 2828 • **name: NCName (1..1)** – the name of the producer. The name of the producer **MUST be unique**
2829 **amongst the producer elements of the composite.** [ASM60104] The name the composite
2830 producer can be different from the name of the promoted component producer.
- 2831 • **promote: listOfAnyURI (1..1)** – identifies the promoted producers. The value is a list containing
2832 entries of the form componentName/producerName. The producer name is optional if the
2833 component only has one producer. The same component producer can be promoted by more
2834 than one composite producer. A composite <producer/> element's @promote attribute **MUST**
2835 **identify one of the component producers within that composite.** [ASM60105] <include/>
2836 **processing MUST take place before the processing of the @promote attribute of a composite**
2837 **service is performed.** [ASM60106]
- 2838 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
2839 [\[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or further qualify the
2840 required intents defined by the promoted component producer.
- 2841 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
2842 [\[SCA-POLICY\]](#) for a description of this attribute.
- 2843 • **promotes (required)** – identifies the promoted producers. The value is a list containing entries of
2844 the form componentName/producerName. The producer name is optional if the component only
2845 has one producer.

2846 The producer element has the following **child elements**:

- 2847 • **eventType : EventType (0..1)** - A producer has **zero or one eventType** child subelement. See
2848 Section Use of <eventType> on a Producer.
- 2849 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2850 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

- 2851 • **policySetAttachment : policySetAttachment (0..n)** - A producer element has **zero or more**
2852 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for
2853 a description of this element.

2854 5.7 Channels

2855 A **channel** is an SCA artifact that is used to connect a set of event producers to a set of event
2856 consumers. The channel can accept events sent by many producers and it can send all of these events
2857 to each of the set of consumers, which are subscribed to the channel.

2858 One role of the channel is to act as an intermediary between the set of producers and the set of
2859 consumers. The channel exists separately from any individual producer or consumer.

2860 A channel acts as if it has a single consumer element with the name "in", to which producers can send
2861 events. A channel acts as if it has a single producer element with the name "out", from which subscribers
2862 receive events.

2863 A channel may be configured with filters, which defines the set of events that the channel accepts. If an
2864 event does not match the filters defined, the event is discarded. See section Filters: Selecting Subsets of
2865 Events for more details.

2866 The pseudo-schema for Channels is shown in Snippet 5-19.

2867

```
2868 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
2869   ...
2870
2871   <channel name="xs:NCName"
2872     requires="list of xs:QName"?
2873     policySets="list of xs:QName"?>*
2874     <filters/?>
2875     <binding ... />?
2876     <requires/>*
2877     <policySetAttachment/>*
2878   </channel>
2879   ...
2880 </composite>
2881
```

2882 *Snippet 5-19: composite Pseudo-Schema with channel Child Element*

2883

2884 The channel element has the following **attributes**:

- 2885 • **name: NCName (1..1)** – the name of the channel. The name of the channel MUST be unique
2886 amongst the channel elements of the composite. [\[ASM60107\]](#)
- 2887 • **requires : listOfQNames (0..1)** – a list of policy intents that apply to the handling of events by
2888 this channel. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this
2889 attribute.
- 2890 • **policySets : listOfQNames (0..1)** – a list of policy sets that apply to the handling of events by
2891 this channel. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this
2892 attribute.

2893 The channel element has the following **child elements**:

- 2894 • **filters: Filters (0..1)** – filter elements. See the section Filters: Selecting Subsets of Events for a
2895 detailed description of filters.
- 2896 • **binding : Binding (0..1)** - A channel element has zero or one **binding element** as children. Each
2897 element defines the mechanism used for transmission of events from/to this channel. Details of
2898 the binding element are described in the [Bindings section](#).
- 2899 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2900 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

- 2901
- **policySetAttachment : policySetAttachment (0..n)** - A channel element has **zero or more policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 2902
- 2903

2904 5.8 Scopes of Channels

2905 Channels can exist either at the Domain level or they can exist within a composite used as an
2906 implementation.

2907 Channels at the Domain level (i.e., channels that are present in the domain-level composite) are termed
2908 **domain channels**. They can be used as targets for producers at any level within the composition
2909 hierarchy. They can be used as sources for consumers at any level within the composition hierarchy. An
2910 SCA runtimes **MUST support the use of domain channels**. [ASM60108]. To create a Domain Channel,
2911 deploy a composite containing a channel directly to the SCA Domain (i.e., do not use that composite as
2912 the implementation of some component in the Domain).

2913 Channels within a composite used as an implementation are private to the components within that
2914 composite. These **private channels** can only be the targets for producers existing within the same
2915 composite as the channel. Private channels can only be sources for consumers existing within the
2916 same composite as the channel. An SCA runtime **MAY support the use of private channels**. [ASM60109].

2917 This division of Channels into global channels and private channels permits the assembler of an
2918 application to control the set of components involved in event exchange, if required. Producers and
2919 consumers of global channels are effectively uncontrolled – they exist at the Domain and they can be
2920 added or removed at any time through deployment actions. Private channels have restricted sets of
2921 producers and consumers – these sets are decided by the assembler when the composite containing
2922 them is created.

2923 5.9 The Default Domain Channel

2924 In SCA Event processing, there is a special **default channel** which is a domain channel.

2925 The default channel always exists, even if it not declared explicitly in the configuration of the Domain. The
2926 default channel has the URI "/".

2927 Producers and consumers at any level in the Domain can communicate using the default channel by
2928 using the URI "/" in their target or source attribute respectively.

2929 5.10 The URI of a Channel

2930 When used for the source of a consumer or for the target of a producer, a channel is referenced by a URI.
2931 The URI of a channel is built from the name of the channel.

2932 The URI of a private channel is the name of the channel. For example, "local-weather"

2933 The URI of a domain channel is "/" followed by the name of the channel. For example, "/cyclones"

2934 The URI of the default domain channel is simply "/".

2935

2936 5.11 Using Composites as Component Implementations

2937 Composites can be used as **component implementations** in higher-level composites – in other words
2938 the higher-level composites can have components which are implemented by composites.

2939 When a composite is used as a component implementation, it defines a boundary of visibility.

2940 Components within the composite cannot be referenced directly by the using component. The using
2941 component can only connect wires to the services and references of the used composite, connect
2942 consumers and producers of the composite to channels, and set values for any properties of the
2943 composite. The internal construction of the composite is invisible to the using component. The boundary
2944 of visibility, sometimes called encapsulation, can be enforced when assembling components and
2945 composites, but such encapsulation structures might not be enforceable in a particular implementation
2946 language.

2947 A composite used as a component implementation also needs to honor a completeness contract. The
2948 services, references and properties of the composite form a contract (represented by the component type
2949 of the composite) which is relied upon by the using component. The concept of completeness of the
2950 composite implies that, once all <include/> element processing is performed on the composite:

- 2951 1. For a composite used as a component implementation, each composite service
2952 offered by the composite MUST promote a component service of a component
2953 that is within the composite. [ASM60032]
- 2954 2. For a composite used as a component implementation, every component
2955 reference of components within the composite with a multiplicity of 1..1 or 1..n
2956 MUST be wired or promoted. [ASM60033] (according to the various rules for
2957 specifying target services for a component reference described in the section "
2958 Specifying the Target Service(s) for a Reference").
- 2959 3. For a composite used as a component implementation, all properties of
2960 components within the composite, where the underlying component
2961 implementation specifies "mustSupply=true" for the property, MUST either
2962 specify a value for the property or source the value from a composite property.
2963 [ASM60034]

2964 The component type of a composite is defined by the set of composite service elements, composite
2965 reference elements, composite consumer elements, composite producer elements, and composite
2966 property elements that are the children of the composite element.

2967 Composites are used as component implementations through the use of the **implementation.composite**
2968 element as a child element of the component. Snippet 5-20 shows the pseudo-schema for the
2969 implementation.composite element:

2970

```
2971 <!-- implementation.composite pseudo-schema -->  
2972 <implementation.composite name="xs:QName" requires="list of xs:QName"?  
2973 policySets="list of xs:QName"?>
```

2974 *Snippet 5-20: implementation.composite Pseudo-Schema*

2975

2976 The **implementation.composite** element has the attributes:

- 2977 • **name (1..1)** – the name of the composite used as an implementation. The @name attribute of an
2978 <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.
2979 [ASM60030]
- 2980 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
2981 [\[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or further qualify the required
2982 intents defined for the promoted component reference.
- 2983 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
2984 [\[SCA-POLICY\]](#) for a description of this attribute.

2985 5.11.1 Component Type of a Composite used as a Component 2986 Implementation

2987 An SCA runtime MUST introspect the componentType of a Composite used as a Component
2988 Implementation following the rules defined in the section "Component Type of a Composite used as a
2989 Component Implementation" [ASM60045]

2990 The componentType of a Composite used as a Component Implementation is introspected from the
2991 Composite document as follows:

2992 A <service/> element exists for each direct <service/> subelement of the <composite/> element

- 2993 • @name attribute set to the value of the @name attribute of the <service/> in the composite

- 2994 • @requires attribute set to the value of the @requires attribute of the <service/> in the composite,
 2995 if present (the value of the @requires attribute contains the intents which apply to the promoted
 2996 component service, as defined in the Policy Framework specification [SCA_POLICY]). If no
 2997 intents apply to the <service/> in the composite, the @requires attribute is omitted.
- 2998 • @policySets attribute set to the value of the @policySets attribute of the <service/> in the
 2999 composite, if it is present. If the @policySets attribute of the <service/> element in the composite
 3000 is absent, the @policySets attribute is omitted.
- 3001 • <interface/> subelement set to the <interface/> subelement of the <service/> element in the
 3002 composite. If not declared on the composite service, it is set to the <interface/> subelement which
 3003 applies to the component service which is promoted by the composite service (this is either an
 3004 explicit <interface/> subelement of the component <service/>, or the <interface/> element of the
 3005 corresponding <service/> in the componentType of the implementation used by the component).
- 3006 • <binding/> subelements set to the <binding/> subelements of the <service/> element in the
 3007 composite. If not declared on the composite service, the <binding/> subelements which apply to
 3008 the component service promoted by the composite service are used, if any are present. If none
 3009 are present in both of these locations, <binding/> subelements are omitted.
- 3010 • <callback/> subelement is set to the <callback/> subelement of the <service/> element in the
 3011 composite. If no <callback/> subelement is present on the composite <service/> element, the
 3012 <callback/> subelement is omitted.
- 3013 A <reference/> element exists for each direct <reference/> subelement of the <composite/> element.
- 3014 • @name attribute set to the value of the @name attribute of the <reference/> in the composite
- 3015 • @requires attribute set to the value of the @requires attribute of the <reference/> in the
 3016 composite, if present (the value of the @requires attribute contains the intents which apply to the
 3017 promoted component references, as defined in the Policy Framework specification
 3018 [SCA_POLICY]). If no intents apply to the <reference/> in the composite, the @requires attribute
 3019 is omitted.
- 3020 • @policySets attribute set to the value of the @policySets attribute of the <reference/> in the
 3021 composite, if present. If the @policySets attribute of the <reference/> element in the composite is
 3022 absent, the @policySets attribute is omitted.
- 3023 • @target attribute is set to the value of the @target attribute of the <reference/> in the composite,
 3024 if present, otherwise the @target attribute is omitted.
- 3025 • @wiredByImpl attribute is set to the value of the @wiredByImpl attribute of the <reference/> in
 3026 the composite, if present. If it is not declared on the composite reference, it is set to the value of
 3027 the @wiredByImpl attribute of the promoted reference(s).
- 3028 • @multiplicity attribute is set to the value of the @multiplicity attribute of the <reference/> in the
 3029 composite
- 3030 • <interface/> subelement set to the <interface/> subelement of the <reference/> element in the
 3031 composite. If not declared on the composite reference, it is set to the <interface/> subelement
 3032 which applies to one of the component reference(s) which are promoted by the composite
 3033 reference (this is either an explicit <interface/> subelement of the component <reference/>, or the
 3034 <interface/> element of the corresponding <reference/> in the componentType of the
 3035 implementation used by the component).
- 3036 • <binding/> subelements set to the <binding/> subelements of the <reference/> element in the
 3037 composite. Otherwise, <binding/> subelements are omitted.
- 3038 • <callback/> subelement is set to the <callback/> subelement of the <reference/> element in the
 3039 composite. Otherwise, <callback/> subelements are omitted.
- 3040 A <consumer/> element exists for each direct <consumer/> subelement of the <composite/> element
- 3041 • @name attribute set to the value of the @name attribute of the <consumer/> in the composite
- 3042 • @requires attribute set to the value of the @requires attribute of the <consumer/> in the
 3043 composite, if present (the value of the @requires attribute contains the intents which apply to the

3044 promoted component consumer, as defined in the Policy Framework specification
3045 [SCA_POLICY]). If no intents apply to the <consumer/> in the composite, the @requires attribute
3046 is omitted.

3047 • @policySets attribute set to the value of the @policySets attribute of the <consumer/> in the
3048 composite, if it is present. If the @policySets attribute of the <consumer/> element in the
3049 composite is absent, the @policySets attribute is omitted.

3050 • <filters/> subelement set to the <filters/> subelement of the <consumer/> element in the
3051 composite, if present (the value of the <filters> element contains the filters that apply to the
3052 promoted component consumer, as defined by the Section Filters: Selecting Subsets of Events).
3053 If no filters apply to the <consumer/> in the composite, the <filters> subelement is omitted.

3054 • <binding/> subelements set to the <binding/> subelements of the <consumer/> element in the
3055 composite. If not declared on the composite consumer, the <binding/> subelements which apply
3056 to the component consumer promoted by the composite consumer are used, if any are present.
3057 If none are present in both of these locations, <binding/> subelements are omitted.

3058 A <producer/> element exists for each direct <producer/> subelement of the <composite/> element

3059 • @name attribute set to the value of the @name attribute of the <producer/> in the composite

3060 • @requires attribute set to the value of the @requires attribute of the <producer/> in the
3061 composite, if present (the value of the @requires attribute contains the intents which apply to the
3062 promoted component producer, as defined in the Policy Framework specification
3063 [SCA_POLICY]). If no intents apply to the <producer/> in the composite, the @requires attribute
3064 is omitted.

3065 • @policySets attribute set to the value of the @policySets attribute of the <producer/> in the
3066 composite, if it is present. If the @policySets attribute of the <producer/> element in the
3067 composite is absent, the @policySets attribute is omitted.

3068 • <eventType/> subelement set to the <eventType/> subelement of the <producer/> element in the
3069 composite, if present (the value of the <eventType> element contains the event types that apply
3070 to the promoted component producer, as defined by the Section Representation of Events and
3071 Event Types. If no event types apply to the <producer/> in the composite, the <eventType>
3072 subelement is omitted.

3073 • <binding/> subelements set to the <binding/> subelements of the <producer/> element in the
3074 composite. If not declared on the composite producer, the <binding/> subelements which apply
3075 to the component producer promoted by the composite producer are used, if any are present. If
3076 none are present in both of these locations, <binding/> subelements are omitted.

3077 A <property/> element exists for each direct <property/> subelement of the <composite/> element.

3078 • @name attribute set to the value of the @name attribute of the <property/> in the composite

3079 • @type attribute set to the value of the @type attribute of the <property/> in the composite, if
3080 present

3081 • @element attribute set to the value of the @element attribute of the <property/> in the composite,
3082 if present
3083 (Note: either a @type attribute is present or an @element attribute is present - one of them has to
3084 be present, but both are not allowed)

3085 • @many attribute set to the value of the @many attribute of the <property/> in the composite, if
3086 present, otherwise omitted.

3087 • @mustSupply attribute set to the value of the @mustSupply attribute of the <property/> in the
3088 composite, if present, otherwise omitted.

3089 • @requires attribute set to the value of the @requires attribute of the <property/> in the composite,
3090 if present, otherwise omitted.

3091 • @policySets attribute set to the value of the @policySets attribute of the <property/> in the
3092 composite, if present, otherwise omitted.

3093 A <implementation/> element exists if the <composite/> element has either of the @requires or
3094 @policySets attributes declared, with:

- 3095 • @requires attribute set to the value of the @requires attribute of the composite, if present,
3096 otherwise omitted.
- 3097 • @policySets attribute set to the value of the @policySets attribute of the composite, if present,
3098 otherwise omitted.

3099

3100 5.11.2 Example of Composite used as a Component Implementation

3101 Snippet 5-21 shows an example of a composite which contains two components, each of which is
3102 implemented by a composite:
3103

```
3104 <?xml version="1.0" encoding="UTF-8"?>  
3105 <!-- CompositeComponent example -->  
3106 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"  
3107   xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200912  
3108   file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"  
3109   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
3110   targetNamespace="http://foo.com"  
3111   xmlns:foo="http://foo.com"  
3112   name="AccountComposite">  
3113  
3114   <service name="AccountService" promote="AccountServiceComponent">  
3115     <interface.java interface="services.account.AccountService"/>  
3116     <binding.ws wsdlElement="AccountService#  
3117       wsdl.port (AccountService/AccountServiceSOAP) "/>  
3118   </service>  
3119  
3120   <reference name="stockQuoteService"  
3121     promote="AccountServiceComponent/StockQuoteService">  
3122     <interface.java  
3123       interface="services.stockquote.StockQuoteService"/>  
3124     <binding.ws  
3125       wsdlElement="http://www.quickstockquote.com/StockQuoteService#  
3126       wsdl.port (StockQuoteService/StockQuoteServiceSOAP) "/>  
3127   </reference>  
3128  
3129   <consumer name="StockQuoteListener" promote="AccountServiceComponent" />  
3130  
3131   <producer name="AccountServiceEvents" promote="AccountServiceComponent" />  
3132  
3133   <property name="currency" type="xsd:string">EURO</property>  
3134  
3135   <component name="AccountServiceComponent">  
3136     <implementation.composite name="foo:AccountServiceComposite1"/>  
3137  
3138     <reference name="AccountDataService" target="AccountDataService"/>  
3139     <reference name="StockQuoteService"/>  
3140  
3141     <property name="currency" source="$currency"/>  
3142   </component>  
3143  
3144   <component name="AccountDataService">  
3145     <implementation.composite name="foo:AccountDataServiceComposite"/>  
3146  
3147     <property name="currency" source="$currency"/>  
3148   </component>  
3149 </composite>  
3150
```

3151 *Snippet 5-21: Example of a composite Using implementation.composite*

3152 5.12 Using Composites through Inclusion

3153 In order to assist team development, composites can be developed in the form of multiple physical
3154 artifacts that are merged into a single logical unit.

3155 A composite can include another composite by using the **include** element. This provides a recursive
3156 inclusion capability. The semantics of included composites are that the element content children of the
3157 included composite are inlined, with certain modification, into the using composite. This is done
3158 recursively till the resulting composite does not contain an **include** element. The outer included
3159 composite element itself is discarded in this process – only its contents are included as described below:

- 3160 1. All the element content children of the included composite are inlined in the
3161 including composite.
- 3162 2. The attributes **@targetNamespace**, **@name** and **@local** of the included
3163 composites are discarded.
- 3164 3. All the namespace declaration on the included composite element are added to
3165 the inlined element content children unless the namespace binding is overridden
3166 by the element content children.
- 3167 4. The attribute **@autowire**, if specified on the included composite, is included on
3168 all inlined component element children unless the component child already
3169 specifies that attribute.
- 3170 5. The attribute values of **@requires** and **@policySet**, if specified on the included
3171 composite, are merged with corresponding attribute on the inlined component,
3172 service and reference children elements. Merge in this context means a set union.
- 3173 6. Extension attributes ,if present on the included composite, follow the rules
3174 defined for that extension. Authors of attribute extensions on the composite
3175 element define the rules applying to those attributes for inclusion.

3176 If the included composite has the value *true* for the attribute **@local** then the including composite **MUST**
3177 have the same value for the **@local** attribute, else it is an error. [\[ASM60041\]](#)

3178 The composite file used for inclusion can have any contents. The composite element can contain any of
3179 the elements which are valid as child elements of a composite element, namely components, services,
3180 references, wires and includes. There is no need for the content of an included composite to be complete,
3181 so that artifacts defined within the using composite or in another associated included composite file can
3182 be referenced. For example, it is permissible to have two components in one composite file while a wire
3183 specifying one component as the source and the other as the target can be defined in a second included
3184 composite file.

3185 The SCA runtime **MUST** raise an error if the composite resulting from the inclusion of one composite into
3186 another is invalid. [\[ASM60031\]](#) For example, it is an error if there are duplicated elements in the using
3187 composite (e.g. two services with the same uri contributed by different included composites). It is not
3188 considered an error if the (using) composite resulting from the inclusion is incomplete (eg. wires with non-
3189 existent source or target). Such incomplete resulting composites are permitted to allow recursive
3190 composition.

3191 Snippet 5-22 snippet shows the pseudo-schema for the include element:

3192

```
3193 <?xml version="1.0" encoding="UTF-8"?>  
3194 <!-- Include snippet -->  
3195 <composite ...>  
3196   ...  
3197   <include name="xs:QName"/> *  
3198   ...  
3199 </composite>
```

3200 *Snippet 5-22: include Pseudo-Schema*

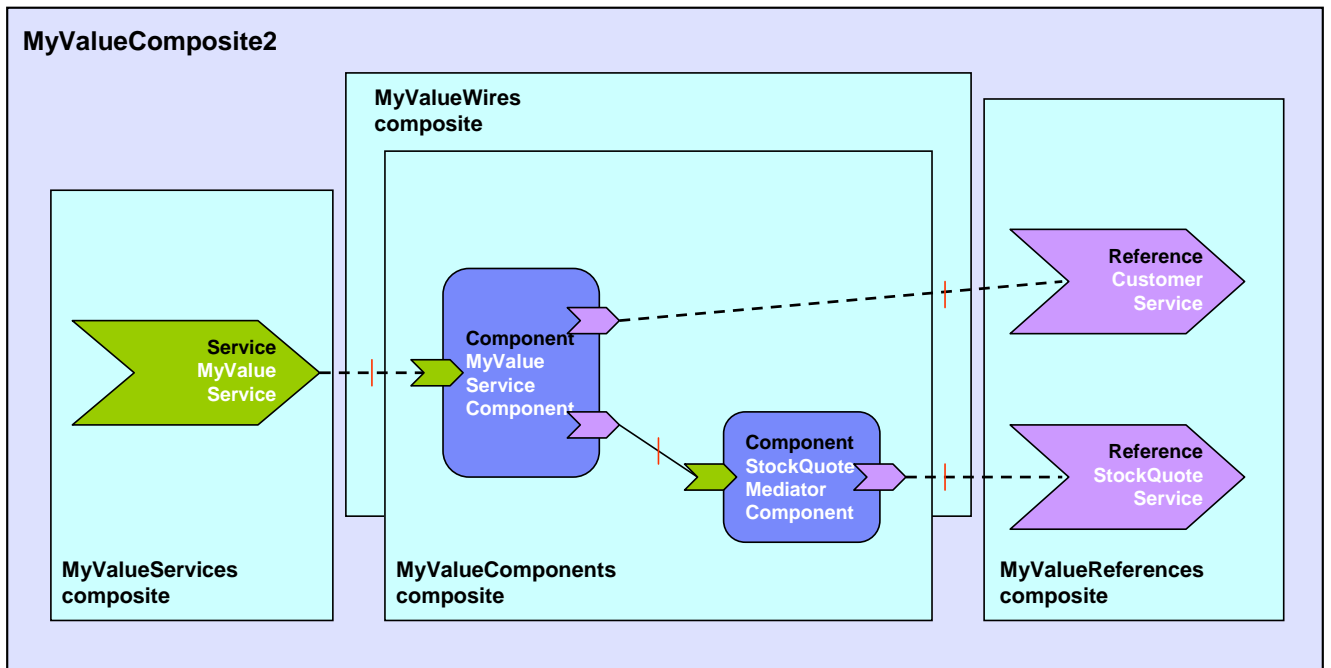
3201
3202
3203
3204
3205

The **include** element has the **attribute**:

- **name: QName (1..1)** – the name of the composite that is included. The @name attribute of an include element MUST be the QName of a composite in the SCA Domain. [ASM60042]

3206 5.12.1 Included Composite Examples

3207 Figure 5-7 shows the assembly diagram for the MyValueComposite2 containing four included
3208 composites. The **MyValueServices composite** contains the MyValueService service. The
3209 **MyValueComponents composite** contains the MyValueServiceComponent and the
3210 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences composite**
3211 contains the CustomerService and StockQuoteService references. The **MyValueWires composite**
3212 contains the wires that connect the MyValueService service to the MyValueServiceComponent, that
3213 connect the customerService reference of the MyValueServiceComponent to the CustomerService
3214 reference, and that connect the stockQuoteService reference of the StockQuoteMediatorComponent
3215 to the StockQuoteService reference. Note that this is just one possible way of building the
3216 MyValueComposite2 from a set of included composites. (TODO: include new example with pub-sub)
3217



3218
3219 Figure 5-7 MyValueComposite2 built from 4 included composites

3220
3221 Snippet 5-23 shows the contents of the MyValueComposite2.composite file for the MyValueComposite2
3222 built using included composites. In this sample it only provides the name of the composite. The composite
3223 file itself could be used in a scenario using included composites to define components, services,
3224 references and wires.

3225
3226
3227
3228
3229
3230
3231
3232

```
<?xml version="1.0" encoding="ASCII"?>  
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
targetNamespace="http://foo.com"  
xmlns:foo="http://foo.com"  
name="MyValueComposite2" >  
  
<include name="foo:MyValueServices"/>
```

```
3233     <include name="foo:MyValueComponents"/>
3234     <include name="foo:MyValueReferences"/>
3235     <include name="foo:MyValueWires"/>
3236
3237 </composite>
```

3238 *Snippet 5-23: Example composite with includes*

3239
3240 Snippet 5-24 shows the content of the MyValueServices.composite file.

```
3241
3242 <?xml version="1.0" encoding="ASCII"?>
3243 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3244           targetNamespace="http://foo.com"
3245           xmlns:foo="http://foo.com"
3246           name="MyValueServices" >
3247
3248     <service name="MyValueService" promote="MyValueServiceComponent">
3249       <interface.java interface="services.myvalue.MyValueService"/>
3250       <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
3251                 wsdl.port(MyValueService/MyValueServiceSOAP)"/>
3252     </service>
3253
3254 </composite>
```

3255 *Snippet 5-24: Example Partial composite with Only a service*

3256
3257 Snippet 5-25 shows the content of the MyValueComponents.composite file.

```
3258
3259 <?xml version="1.0" encoding="ASCII"?>
3260 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3261           targetNamespace="http://foo.com"
3262           xmlns:foo="http://foo.com"
3263           name="MyValueComponents" >
3264
3265     <component name="MyValueServiceComponent">
3266       <implementation.java
3267         class="services.myvalue.MyValueServiceImpl"/>
3268       <property name="currency">EURO</property>
3269     </component>
3270
3271     <component name="StockQuoteMediatorComponent">
3272       <implementation.java class="services.myvalue.SQMediatorImpl"/>
3273       <property name="currency">EURO</property>
3274     </component>
3275
3276 </composite>
```

3277 *Snippet 5-25: Example Partial composite with Only components*

3278
3279 Snippet 5-26 shows the content of the MyValueReferences.composite file.

```
3280
3281 <?xml version="1.0" encoding="ASCII"?>
3282 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3283           targetNamespace="http://foo.com"
3284           xmlns:foo="http://foo.com"
3285           name="MyValueReferences" >
3286
3287     <reference name="CustomerService"
```

```

3288     promote="MyValueServiceComponent/CustomerService">
3289     <interface.java interface="services.customer.CustomerService"/>
3290     <binding.sca/>
3291 </reference>
3292
3293     <reference name="StockQuoteService"
3294     promote="StockQuoteMediatorComponent">
3295     <interface.java
3296     interface="services.stockquote.StockQuoteService"/>
3297     <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
3298     wsdl.port (StockQuoteService/StockQuoteServiceSOAP)"/>
3299     </reference>
3300
3301 </composite>

```

3302 *Snippet 5-26: Example Partial composite with Only references*

3303

3304 Snippet 5-27 shows the content of the MyValueWires.composite file.

3305

```

3306 <?xml version="1.0" encoding="ASCII"?>
3307 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3308     targetNamespace="http://foo.com"
3309     xmlns:foo="http://foo.com"
3310     name="MyValueWires" >
3311
3312     <wire source="MyValueServiceComponent/stockQuoteService"
3313     target="StockQuoteMediatorComponent"/>
3314
3315 </composite>

```

3316 *Snippet 5-27: Example Partial composite with Only a wire*

3317 5.13 Composites which Contain Component Implementations of 3318 Multiple Types

3319 A Composite containing multiple components can have multiple component implementation types. For
3320 example, a Composite can contain one component with a Java POJO as its implementation and another
3321 component with a BPEL process as its implementation.

3322 5.14 Structural URI of Components

3323 The **structural URI** is a relative URI that describes each use of a given component in the Domain,
3324 relative to the URI of the Domain itself. It is never specified explicitly, but it calculated from the
3325 configuration of the components configured into the Domain.

3326 A component in a composite can be used more than once in the Domain, if its containing composite is
3327 used as the implementation of more than one higher-level component. The structural URI is used to
3328 separately identify each use of a component - for example, the structural URI can be used to attach
3329 different policies to each separate use of a component.

3330 For components directly deployed into the Domain, the structural URI is simply the name of the
3331 component.

3332 Where components are nested within a composite which is used as the implementation of a higher level
3333 component, the structural URI consists of the name of the nested component prepended with each of the
3334 names of the components upto and including the Domain level component.

3335 For example, consider a component named Component1 at the Domain level, where its implementation is
3336 Composite1 which in turn contains a component named Component2, which is implemented by
3337 Composite2 which contains a component named Component3. The three components in this example
3338 have the following structural URIs:

- 3339 1. Component1: Component1
3340 2. Component2: Component1/Component2
3341 3. Component3: Component1/Component2/Component3

3342 The structural URI can also be extended to refer to specific parts of a component, such as a service or a
3343 reference, by appending an appropriate fragment identifier to the component's structural URI, as follows:

- 3344 • Service:
3345 #service(servicename)
- 3346 • Reference:
3347 #reference(referencename)
- 3348 • Service binding:
3349 #service-binding(servicename/bindingname)
- 3350 • Reference binding:
3351 #reference-binding(referencename/bindingname)
- 3352 • Consumer:
3353 #consumer(consumername)
- 3354 • Producer:
3355 #producer(producername)
- 3356 • Consumer binding:
3357 #consumer-binding(consumername/bindingname)
- 3358 • Producer binding:
3359 #producer-binding(producername/bindingname)

3360

3361 So, for example, the structural URI of the service named "testservice" of component "Component1" is
3362 Component1#service(testservice).

3363 6 Interface

3364 **Interfaces** define one or more business functions. These business functions are provided by Services
3365 and are used by References. A Service offers the business functionality of exactly one interface for use
3366 by other components. Each interface defines one or more service **operations** and each operation has
3367 zero or one **request (input) message** and zero or one **response (output) message**. The request and
3368 response messages can be simple types such as a string value or they can be complex types.

3369 SCA currently supports the following interface type systems:

- 3370 • Java interfaces
- 3371 • WSDL 1.1 portTypes ([Web Services Definition Language \[WSDL-11\]](#))
- 3372 • C++ classes
- 3373 • Collections of 'C' functions

3374 SCA is also extensible in terms of interface types. Support for other interface type systems can be added
3375 through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

3376 Snippet 6-1 shows the pseudo-schema for the **interface** base element:

3377

```
3378 <interface remotable="boolean"? requires="list of xs:QName"?  
3379     policySets="list of xs:QName"?  
3380     <requires/*>  
3381     <policySetAttachment/*>  
3382 </interface>
```

3383 *Snippet 6-1: interface Pseudo-Schema*

3384

3385 The **interface** base element has the **attributes**:

- 3386 • **remotable : boolean (0..1)** – indicates whether an interface is remotable or not (see [the section on](#)
3387 [Local and Remotable Interfaces](#)). A value of “true” means the interface is remotable, and a value of
3388 “false” means it is not. The @remotable attribute has no default value. This attribute is used as an
3389 alternative to interface type specific mechanisms such as the @Remotable annotation on a Java
3390 interface. The remotable nature of an interface in the absence of this attribute is interface type
3391 specific. The rules governing how this attribute relates to interface type specific mechanisms are
3392 defined by each interface type. When specified on an interface definition which includes a callback,
3393 this attribute also applies to the callback interface (see [the section on Bidirectional Interfaces](#)).
- 3394 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
3395 [\[SCA-POLICY\]](#) for a description of this attribute
- 3396 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
3397 [\[SCA-POLICY\]](#) for a description of this attribute.

3398 The **interface** element has the following **subelements**:

- 3399 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
3400 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 3401 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
3402 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
3403 description of this element.

3404 For information about Java interfaces, including details of SCA-specific annotations, see the [SCA Java](#)
3405 [Common Annotations and APIs specification \[SCA-Common-Java\]](#).

3406 For information about WSDL interfaces, including details of SCA-specific extensions, see [SCA-Specific](#)
3407 [Aspects for WSDL Interfaces and WSDL Interface Type](#).

3408 For information about C++ interfaces, see the [SCA C++ Client and Implementation Model specification](#)
3409 [\[SCA-CPP-Client\]](#).
3410 For information about C interfaces, see the [SCA C Client and Implementation Model specification](#) [\[SCA-](#)
3411 [C-Client\]](#).

3412 6.1 Local and Remotable Interfaces

3413 A remotable service is one which can be called by a client which is running in an operating system
3414 process different from that of the service itself (this also applies to clients running on different machines
3415 from the service). Whether a service of a component implementation is remotable is defined by the
3416 interface of the service. WSDL defined interfaces are always remotable. See the relevant specifications
3417 for details of interfaces defined using other languages.

3418 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**
3419 interactions. **Remotable service Interfaces MUST NOT make use of *method or operation overloading*.**
3420 **[ASM80002]** This restriction on operation overloading for remotable services aligns with the WSDL 2.0
3421 specification, which disallows operation overloading, and also with the [WS-I Basic Profile 1.1 \[WSI-BP\]](#)
3422 (section 4.5.3 - R2304) which has a constraint which disallows operation overloading when using WSDL
3423 1.1.

3424 Independent of whether the remotable service is called remotely from outside the process where the
3425 service runs or from another component running in the same process, the data exchange semantics are
3426 **by-value**.

3427 Implementations of remotable services can modify input messages (parameters) during or after an
3428 invocation and can modify return messages (results) after the invocation. **If a remotable service is called**
3429 **locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the**
3430 **service or post-invocation modifications to return messages are seen by the caller.** **[ASM80003]**

3431 Snippet 6-2 shows an example of a remotable java interface:

3432

```
3433 package services.hello;  
3434  
3435 @Remotable  
3436 public interface HelloService {  
3437  
3438     String hello(String message);  
3439 }
```

3440 *Snippet 6-2: Example remotable interface*

3441

3442 It is possible for the implementation of a remotable service to indicate that it can be called using by-
3443 reference data exchange semantics when it is called from a component in the same process. This can be
3444 used to improve performance for service invocations between components that run in the same process.
3445 This can be done using the [@AllowsPassByReference](#) annotation (see the [Java Client and](#)
3446 [Implementation Specification](#)).

3447 A service typed by a local interface can only be called by clients that are running in the same process as
3448 the component that implements the local service. Local services cannot be published via remotable
3449 services of a containing composite. In the case of Java a local service is defined by a Java interface
3450 definition without a **@Remotable** annotation.

3451 The style of local interfaces is typically **fine grained** and intended for **tightly coupled** interactions. Local
3452 service interfaces can make use of **method or operation overloading**.

3453 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

3454 6.2 Interface Compatibility

3455 The **compatibility** of two interfaces is defined in this section and these definitions are used throughout
3456 this specification. Three forms of compatibility are defined:

- 3457 • Compatible interfaces
- 3458 • Compatible subset
- 3459 • Compatible superset

3460 Note that WSDL 1.1 message parts can point to an XML Schema element declaration or to an XML
3461 Schema types. When determining compatibility between two WSDL operations, a message part that
3462 points to an XML Schema element declaration is considered to be incompatible with a message part that
3463 points to an XML Schema type.

3464 6.2.1 Compatible Interfaces

3465 An interface A is **Compatible** with a second interface B if and only if all of points 1 through 7 in the
3466 following list apply:

- 3467 1. interfaces A and B are either both remotable or else both local
- 3468 2. the set of operations in interface A is the same as the set of operations in
3469 interface B
- 3470 3. compatibility for individual operations of the interfaces A and B is defined as
3471 compatibility of the signature, i.e., the operation name, the input types, and the
3472 output types are the same
- 3473 4. the order of the input and output types for each operation in interface A is the
3474 same as the order of the input and output types for the corresponding operation
3475 in interface B
- 3476 5. the set of Faults and Exceptions expected by each operation in interface A is the
3477 same as the set of Faults and Exceptions specified by the corresponding
3478 operation in interface B
- 3479 6. for checking the compatibility of 2 remotable interfaces which are in different
3480 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
3481 compatibility checking is done between the WSDL 1.1 mapped interfaces.
3482
3483 For checking the compatibility of 2 local interfaces which are in different interface
3484 languages, the method of checking compatibility is defined by the specifications
3485 which define those interface types, which must define mapping rules for the 2
3486 interface types concerned.
- 3487 7. if either interface A or interface B declares a callback interface then both interface
3488 A and interface B declare callback interfaces and the callback interface declared
3489 on interface A is compatible with the callback interface declared on interface B,
3490 according to points 1 through 6 above

3491 6.2.2 Compatible Subset

3492 An interface A is a **Compatible Subset** of a second interface B if and only if all of points 1 through 7 in
3493 the following list apply:

- 3494 1. interfaces A and B are either both remotable or else both local
- 3495 2. the set of operations in interface A is the same as or is a subset of the set of
3496 operations in interface B
- 3497 3. compatibility for individual operations of the interfaces A and B is defined as
3498 compatibility of the signature, i.e., the operation name, the input types, and the
3499 output types are the same

- 3500 4. the order of the input and output types for each operation in interface A is the
3501 same as the order of the input and output types for the corresponding operation
3502 in interface B
- 3503 5. the set of Faults and Exceptions expected by each operation in interface A is the
3504 same as or is a superset of the set of Faults and Exceptions specified by the
3505 corresponding operation in interface B
- 3506 6. for checking the compatibility of 2 remotable interfaces which are in different
3507 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
3508 compatibility checking is done between the WSDL 1.1 mapped interfaces.
3509
- 3510 For checking the compatibility of 2 local interfaces which are in different interface
3511 languages, the method of checking compatibility is defined by the specifications
3512 which define those interface types, which must define mapping rules for the 2
3513 interface types concerned.
- 3514 7. if either interface A or interface B declares a callback interface then both interface
3515 A and interface B declare callback interfaces and the callback interface declared
3516 on interface B is a compatible subset of the callback interface declared on
3517 interface A, according to points 1 through 6 above

3518 6.2.3 Compatible Superset

3519 An interface A is a **Compatible Superset** of a second interface B if and only if all of points 1 through 7 in
3520 the following list apply:

- 3521 1. interfaces A and B are either both remotable or else both local
- 3522 2. the set of operations in interface A is the same as or is a superset of the set of
3523 operations in interface B
- 3524 3. compatibility for individual operations of the interfaces A and B is defined as
3525 compatibility of the signature, i.e., the operation name, the input types, and the
3526 output types are the same
- 3527 4. the order of the input and output types for each operation in interface B is the
3528 same as the order of the input and output types for the corresponding operation
3529 in interface A
- 3530 5. the set of Faults and Exceptions expected by each operation in interface A is the
3531 same as or is a subset of the set of Faults and Exceptions specified by the
3532 corresponding operation in interface B
- 3533 6. for checking the compatibility of 2 remotable interfaces which are in different
3534 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
3535 compatibility checking is done between the WSDL 1.1 mapped interfaces.
3536
- 3537 For checking the compatibility of 2 local interfaces which are in different interface
3538 languages, the method of checking compatibility is defined by the specifications
3539 which define those interface types, which must define mapping rules for the 2
3540 interface types concerned.
- 3541 7. if either interface A or interface B declares a callback interface then both interface
3542 A and interface B declare callback interfaces and the callback interface declared
3543 on interface B is a compatible superset of the callback interface declared on
3544 interface A, according to points 1 through 6 above

3545 6.3 Bidirectional Interfaces

3546 The relationship of a business service to another business service is often peer-to-peer, requiring a two-
3547 way dependency at the service level. In other words, a business service represents both a consumer of a
3548 service provided by a partner business service and a provider of a service to the partner business
3549 service. This is especially the case when the interactions are based on asynchronous messaging rather
3550 than on remote procedure calls. The notion of **bidirectional interfaces** is used in SCA to directly model
3551 peer-to-peer bidirectional business service relationships.

3552 An interface element for a particular interface type system needs to allow the specification of a callback
3553 interface. If a callback interface is specified, SCA refers to the interface as a whole as a bidirectional
3554 interface.

3555 Snippet 6-3 shows the interface element defined using Java interfaces with a @callbackInterface
3556 attribute.

3557

```
3558 <interface.java interface="services.invoicing.ComputePrice"  
3559 callbackInterface="services.invoicing.InvoiceCallback"/>
```

3560 *Snippet 6-3: Example interface with a callback*

3561

3562 If a service is defined using a bidirectional interface element then its implementation implements the
3563 interface, and its implementation uses the callback interface to converse with the client that called the
3564 service interface.

3565 If a reference is defined using a bidirectional interface element, the client component implementation
3566 using the reference calls the referenced service using the interface. The client **MUST** provide an
3567 implementation of the callback interface. [ASM80004]

3568 Callbacks can be used for both remotable and local services. **Either both interfaces of a bidirectional
3569 service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and
3570 remote services.** [ASM80005]

3571 Note that an interface document such as a WSDL file or a Java interface can contain annotations that
3572 declare a callback interface for a particular interface (see [the section on WSDL Interface type](#) and the
3573 [Java Common Annotations and APIs specification \[SCA-Common-Java\]](#)). **Whenever an interface
3574 document declaring a callback interface is used in the declaration of an <interface/> element in SCA, it
3575 MUST be treated as being bidirectional with the declared callback interface.** [ASM80010] In such cases,
3576 there is no requirement for the <interface/> element to declare the callback interface explicitly.

3577 If an <interface/> element references an interface document which declares a callback interface and also
3578 itself contains a declaration of a callback interface, the two callback interfaces **MUST** be compatible.
3579 [ASM80011]

3580 See [the section on Interface Compatibility](#) for a definition of "compatible interfaces".

3581 In a bidirectional interface, the service interface can have more than one operation defined, and the
3582 callback interface can also have more than one operation defined. SCA runtimes **MUST** allow an
3583 invocation of any operation on the service interface to be followed by zero, one or many invocations of
3584 any of the operations on the callback interface. [ASM80009] These callback operations can be invoked
3585 either before or after the operation on the service interface has returned a response message, if there is
3586 one.

3587 For a given invocation of a service operation, which operations are invoked on the callback interface,
3588 when these are invoked, the number of operations invoked, and their sequence are not described by
3589 SCA. It is possible that this metadata about the bidirectional interface can be supplied through
3590 mechanisms outside SCA. For example, it might be provided as a written description attached to the
3591 callback interface.

3592 6.4 Long-running Request-Response Operations

3593 6.4.1 Background

3594 A service offering one or more operations which map to a WSDL request-response pattern might be
3595 implemented in a long-running, potentially interruptible, way. Consider a BPEL process with receive and
3596 reply activities referencing the WSDL request-response operation. Between the two activities, the
3597 business process logic could be a long-running sequence of steps, including activities causing the
3598 process to be interrupted. Typical examples are steps where the process waits for another message to
3599 arrive or a specified time interval to expire, or the process performs asynchronous interactions such as
3600 service invocations bound to asynchronous protocols or user interactions. This is a common situation in
3601 business processes, and it causes the implementation of the WSDL request-response operation to run for
3602 a very long time, e.g., several months (!). In this case, it is not meaningful for any caller to remain in a
3603 synchronous wait for the response while blocking system resources or holding database locks.

3604 Note that it is possible to model long-running interactions as a pair of two independent operations as
3605 described in the section on bidirectional interfaces. However, it is a common practice (and in fact much
3606 more convenient) to model a request-response operation and let the infrastructure deal with the
3607 asynchronous message delivery and correlation aspects instead of putting this burden on the application
3608 developer.

3609 6.4.2 Definition of "long-running"

3610 A request-response operation is considered long-running if the implementation does not guarantee the
3611 delivery of the response within any specified time interval. Clients invoking such request-response
3612 operations are strongly discouraged from making assumptions about when the response can be
3613 expected.

3614 6.4.3 The asyncInvocation Intent

3615 This specification permits a long-running request-response operation or a complete interface containing
3616 such operations to be marked using a policy intent with the name **asyncInvocation**. It is also possible for
3617 a service to set the `asyncInvocation` intent when using an interface which is not marked with the
3618 `asyncInvocation` intent. This can be useful when reusing an existing interface definition that does not
3619 contain SCA information.

3620 6.4.4 Requirements on Bindings

3621 In order to support a service operation which is marked with the `asyncInvocation` intent, it is necessary for
3622 the binding (and its associated policies) to support separate handling of the request message and the
3623 response message. Bindings which only support a synchronous style of message handling, such as a
3624 conventional HTTP binding, cannot be used to support long-running operations.

3625 The requirements on a binding to support the `asyncInvocation` intent are the same as those to support
3626 services with bidirectional interfaces - namely that the binding needs to be able to treat the transmission
3627 of the request message separately from the transmission of the response message, with an arbitrarily
3628 large time interval between the two transmissions.

3629 An example of a binding/policy combination that supports long-running request-response operations is a
3630 Web service binding used in conjunction with the WS-Addressing "wsam:NonAnonymousResponses"
3631 assertion.

3632 6.4.5 Implementation Type Support

3633 SCA implementation types can provide special asynchronous client-side and asynchronous server-side
3634 mappings to assist in the development of services and clients for long-running request-response
3635 operations.

3636 6.5 SCA-Specific Aspects for WSDL Interfaces

3637 There are a number of aspects that SCA applies to interfaces in general, such as marking them as having
3638 a callback interface. These aspects apply to the interfaces themselves, rather than their use in a specific
3639 place within SCA. There is thus a need to provide appropriate ways of marking the interface definitions
3640 themselves, which go beyond the basic facilities provided by the interface definition language.

3641 For WSDL interfaces, there is an extension mechanism that permits additional information to be included
3642 within the WSDL document. SCA takes advantage of this extension mechanism. In order to use the SCA
3643 extension mechanism, the SCA namespace (<http://docs.oasis-open.org/ns/opencsa/sca/200912>) needs
3644 to be declared within the WSDL document.

3645 First, SCA defines a global element in the SCA namespace which provides a mechanism to attach policy
3646 intents - **requires**. Snippet 6-4 shows the definition of the requires element:

3647

```
3648 <element name="requires">  
3649   <complexType>  
3650     <sequence minOccurs="0" maxOccurs="unbounded">  
3651       <any namespace="##other" processContents="lax"/>  
3652     </sequence>  
3653     <attribute name="intents" type="sca:listOfQNames" use="required"/>  
3654     <anyAttribute namespace="##other" processContents="lax"/>  
3655   </complexType>  
3656 </element>  
  
3657  
3658 <simpleType name="listOfQNames">  
3659   <list itemType="QName"/>  
3660 </simpleType>
```

3661 *Snippet 6-4: requires WSDL extension definition*

3662

3663 The requires element can be used as a subelement of the WSDL portType and operation elements. The
3664 element contains one or more intent names, as defined by the [Policy Framework specification \[SCA-](#)
3665 [POLICY\]](#). Any service or reference that uses an interface marked with intents MUST implicitly add those
3666 intents to its own @requires list. [ASM80008]

3667 SCA defines an attribute which is used to indicate that a given WSDL portType element (WSDL 1.1) has
3668 an associated callback interface. This is the @callback attribute, which applies to a WSDL portType
3669 element.

3670 Snippet 6-5 shows the definition of the @callback attribute:

3671

```
3672 <attribute name="callback" type="QName"/>
```

3673 *Snippet 6-5: callback WSDL extension definition*

3674

3675 The value of the @callback attribute is the QName of a portType. The portType declared by the
3676 @callback attribute is the callback interface to use for the portType which is annotated by the
3677 @callback attribute.

3678 Snippet 6-6 is an example of a portType element with a @callback attribute:

3679

```
3680 <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
3681   <operation name="apply">  
3682     <input message="tns:ApplicationInput"/>  
3683     <output message="tns:ApplicationOutput"/>  
3684   </operation>  
3685   ...  
3686 </portType>
```

3687 *Snippet 6-6: Example use of @callback*

3688 6.6 WSDL Interface Type

3689 The WSDL interface type is used to declare interfaces for services and for references, where the interface
3690 is defined in terms of a WSDL document. An interface is defined in terms of a WSDL 1.1 portType with
3691 the arguments and return of the service operations described using XML schema.

3692 A WSDL interface is declared by an *interface.wSDL* element. Snippet 6-7 shows the pseudo-schema for
3693 the interface.wSDL element:

3694

```
3695 <!-- WSDL Interface schema snippet -->  
3696 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?  
3697     remotable="xs:boolean"?  
3698     requires="listOfQNames"?  
3699     policySets="listOfQNames">  
3700     <requires/>*  
3701     <policySetAttachment/>*  
3702 </interface.wSDL>
```

3703 *Snippet 6-7: interface.wSDL Pseudo-Schema*

3704

3705 The *interface.wSDL* element has the *attributes*:

- 3706 • **interface : uri (1..1)** - the URI of a WSDL portType

3707 The *interface.wSDL* @interface attribute MUST reference a portType of a WSDL 1.1 document.
3708 [ASM80001]

- 3709 • **callbackInterface : uri (0..1)** - a callback interface, which is the URI of a WSDL portType

3710 The *interface.wSDL* @callbackInterface attribute, if present, MUST reference a portType of a WSDL
3711 1.1 document. [ASM80016]

- 3712 • **remotable : boolean (0..1)** – indicates whether the interface is remotable or not. @remotable has a
3713 default value of true. WSDL interfaces are always remotable and therefore an <interface.wSDL/>
3714 element MUST NOT contain remotable="false". [ASM80017]

- 3715 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
3716 [SCA-POLICY] for a description of this attribute.

- 3717 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
3718 [SCA-POLICY] for a description of this attribute.

3719 The form of the URI for WSDL portTypes follows the syntax described in the [WSDL 1.1 Element](#)
3720 [Identifiers specification](#) [WSDL11_Identifiers]

3721 The *interface.wSDL* element has the following *subelements*:

- 3722 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
3723 [Policy Framework specification](#) [SCA-POLICY] for a description of this element.

- 3724 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
3725 **policySetAttachment subelements**. See the [Policy Framework specification](#) [SCA-POLICY] for a
3726 description of this element.

3727 6.6.1 Example of interface.wSDL

3728 Snippet 6-8 shows an interface defined by the WSDL portType "StockQuote" with a callback interface
3729 defined by the "StockQuoteCallback" portType.

3730

```
3731 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
3732     wsdl.porttype (StockQuote) "  
3733     callbackInterface="http://www.stockquote.org/StockQuoteService#  
3734     wsdl.porttype (StockQuoteCallback) "/>
```

3735 *Snippet 6-8: Example interface.wsdl*

3736

7 Binding

3737 Bindings are used by services, references, channels, consumers, and producers. References use
3738 bindings to describe the access mechanism used to call a service (which can be a service provided by
3739 another SCA composite). Services use bindings to describe the access mechanism that clients (which
3740 can be a client from another SCA composite) have to use to call the service. Producers, consumers and
3741 channels use bindings to describe the mechanism used to send and receive events.

3742 SCA supports the use of multiple different types of bindings. Examples include **SCA service**, **Web**
3743 **service**, **stateless session EJB**, **database stored procedure**, **EIS service**. SCA provides an
3744 extensibility mechanism by which an SCA runtime can add support for additional binding types. For
3745 details on how additional binding types are defined, see the section on the Extension Model.

3746 A binding is defined by a **binding element** which is a child element of a service, a reference, a channel,
3747 a consumer, or a producer element in a composite. Snippet 7-1 shows the composite pseudo-schema
3748 with the pseudo-schema for the binding element.

```
3749 <?xml version="1.0" encoding="ASCII"?>
3750 <!-- Bindings schema snippet -->
3751 <composite ... >
3752   ...
3753     <service ... >*
3754     <interface ... />?
3755     <binding uri="xs:anyURI"? name="xs:NCName"?
3756       requires="list of xs:QName"?
3757       policySets="list of xs:QName"?>*
3758       <wireFormat/>?
3759       <operationSelector/>?
3760       <requires/>*
3761       <policySetAttachment/>*
3762     </binding>
3763     <callback>?
3764       <binding uri="xs:anyURI"? name="xs:NCName"?
3765         requires="list of xs:QName"?
3766         policySets="list of xs:QName"?>+
3767         <wireFormat/>?
3768         <operationSelector/>?
3769         <requires/>*
3770         <policySetAttachment/>*
3771       </binding>
3772     </callback>
3773   </service>
3774   ...
3775   <reference ... >*
3776   <interface ... />?
3777   <binding uri="xs:anyURI"? name="xs:NCName"?
3778     requires="list of xs:QName"?
3779     policySets="list of xs:QName"?>*
3780     <wireFormat/>?
3781     <operationSelector/>?
3782     <requires/>*
3783     <policySetAttachment/>*
3784   </binding>
3785   <callback>?
3786     <binding uri="xs:anyURI"? name="xs:NCName"?
3787       requires="list of xs:QName"?
3788       policySets="list of xs:QName"?>+
3789       <wireFormat/>?
3790       <operationSelector/>?
3791       <requires/>*
3792     <policySetAttachment/>*
```

```

3793     </binding>
3794   </callback>
3795 </reference>
3796   ...
3797
3798 <channel ... >
3799   <filters/>?
3800   <binding uri="xs:anyURI"? name="xs:NCName"?
3801     requires="list of xs:QName"?
3802     policySets="list of xs:QName"?>*
3803   <requires/>*
3804   <policySetAttachment/>*
3805   <filters/>*
3806 </binding>?
3807 <requires/>*
3808 <policySetAttachment/>*
3809 </channel>*
3810
3811 <consumer ... >
3812   <filters/>?
3813   <binding uri="xs:anyURI"? name="xs:NCName"?
3814     requires="list of xs:QName"?
3815     policySets="list of xs:QName"?>*
3816   <requires/>*
3817   <policySetAttachment/>*
3818   <filters/>*
3819 </binding>*
3820 <requires/>*
3821 <policySetAttachment/>*
3822 </consumer>*
3823
3824 <producer ... >
3825   <eventType/>?
3826   <binding uri="xs:anyURI"? name="xs:NCName"?
3827     requires="list of xs:QName"?
3828     policySets="list of xs:QName"?>*
3829   <requires/>*
3830   <policySetAttachment/>*
3831   <filters/>*
3832 </binding>*
3833 <requires/>*
3834 <policySetAttachment/>*
3835 </producer>*
3836
3837 </composite>

```

3838 *Snippet 7-1: composite Pseudo-Schema with binding Child element*

3839

3840 The element name of the binding element is architected; it is in itself a qualified name. The first qualifier is
3841 always named “binding”, and the second qualifier names the respective binding-type (e.g. binding.sca,
3842 binding.ws, binding.ejb, binding.eis).

3843 A **binding** element has the attributes:

3844 • **uri (0..1)** - has the semantic:

3845 – The @uri attribute can be omitted.

3846 – For a binding of a **reference** the @uri attribute defines the target URI of the reference. This
3847 MUST be either the componentName/serviceName/bindingName for a wire to an endpoint within
3848 the SCA Domain, or the accessible address of some service endpoint either inside or outside the
3849 SCA Domain (where the addressing scheme is defined by the type of the binding). [ASM90001]

3850 – The circumstances under which the @uri attribute can be used are defined in section "[Specifying](#)
3851 [the Target Service\(s\) for a Reference.](#)"

3852 – For a binding of a **service** the @uri attribute defines the bindingURI. If present, the bindingURI
3853 can be used by the binding as described in the [section "Form of the URI of a Deployed Binding"](#).

3854 • **name (0..1)** – a name for the binding instance (an NCName). The @name attribute allows distinction
3855 between multiple binding elements on a single service or reference. The default value of the @name
3856 attribute is the service or reference name. **When a service or reference has multiple bindings, all non-**
3857 **callback bindings of the service or reference MUST have unique names, and all callback bindings of**
3858 **the service or reference MUST have unique names.** [ASM90002] This uniqueness requirement
3859 implies that only one non-callback binding of a service or reference can have the default @name
3860 value, and only one callback binding of a service or reference can have the default @name value.
3861

3862 The @name also permits the binding instance to be referenced from elsewhere – particularly useful
3863 for some types of binding, which can be declared in a definitions document as a template and
3864 referenced from other binding instances, simplifying the definition of more complex binding instances
3865 (see the [JMS Binding specification \[SCA-JMSBINDING\]](#) for examples of this referencing).

3866 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
3867 description of this attribute.

3868 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
3869 description of this attribute.

3870 A **binding** element has the child elements:

3871 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See [the wireFormat](#)
3872 [section](#) for details.

3873 • **operationSelector(0..1)** - an operationSelector element that is used to match a particular message to
3874 a particular operation in the interface. See [the operationSelector section](#) for details

3875 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
3876 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

3877 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
3878 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
3879 description of this element.

3880 • **filters : Filters (0..1)** – a binding-specific **zero or more filters subelement**. For more details on the
3881 filters subelement see Section Filters: Selecting Subsets of Events. The filters subelement of the
3882 bindings element contains filters that are binding-specific.

3883 When multiple bindings exist for a service, channel, consumer, or producer, it means that the service,
3884 channel, consumer, or producer is available through any of the specified bindings. The technique that
3885 the SCA runtime uses to choose among available bindings is left to the implementation and it might
3886 include additional (nonstandard) configuration. Whatever technique is used needs to be documented by
3887 the runtime.

3888 Services, References, consumers, and producers can always have their bindings overridden at the SCA
3889 Domain level, unless restricted by Intents applied to them.

3890 **If a reference has any bindings, they MUST be resolved, which means that each binding MUST include a**
3891 **value for the @uri attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired**
3892 **using other SCA mechanisms.** [ASM90003] To specify constraints on the kinds of bindings that are
3893 acceptable for use with a reference, the user specifies either policy intents or policy sets.
3894

3895 Users can also specifically wire, not just to a component service, but to a specific binding offered by that
3896 target service. **To wire to a specific binding of a target service the syntax**
3897 **"componentName/serviceName/bindingName" MUST be used.** [ASM90004]

3898 The following sections describe the SCA and Web service binding type in detail.

3899 **7.1 Messages containing Data not defined in the Service Interface**

3900 It is possible for a message to include information that is not defined in the interface used to define the
3901 service, for instance information can be contained in SOAP headers or as MIME attachments.

3902 Implementation types can make this information available to component implementations in their
3903 execution context. The specifications for these implementation types describe how this information is
3904 accessed and in what form it is presented.

3905 **7.2 WireFormat**

3906 A wireFormat is the form that a data structure takes when it is transmitted using some communication
3907 binding. Another way to describe this is "the form that the data takes on the wire". A wireFormat can be
3908 specific to a given communication method, or it can be general, applying to many different communication
3909 methods. An example of a general wireFormat is XML text format.

3910 Where a particular SCA binding can accommodate transmitting data in more than one format, the
3911 configuration of the binding can include a definition of the wireFormat to use. This is done using an
3912 <sca:wireFormat/> subelement of the <binding/> element.

3913 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats to be
3914 the default wireFormat which applies if no <wireFormat/> subelement is present.

3915 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a
3916 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 3917 • <sca:wireFormat.xml/>
3918 A wireFormat that transmits the data as an XML text datastructure
- 3919 • <sca:wireFormat.jms/>
3920 The "default JMS wireFormat" as described in the JMS Binding specification

3921 Specific wireFormats can have elements that include either attributes or subelements or both.

3922 For details about specific wireFormats, see the related SCA Binding specifications.

3923 **7.3 OperationSelector**

3924 An operationSelector is necessary for some types of transport binding where messages are transmitted
3925 across the transport without any explicit relationship between the message and the interface operation to
3926 which it relates. SOAP is an example of a protocol where the messages do contain explicit information
3927 that relates each message to the operation it targets. However, other transport bindings have messages
3928 where this relationship is not expressed in the message or in any related headers (pure JMS messages,
3929 for example). In cases where the messages arrive at a service without any explicit information that maps
3930 them to specific operations, it is necessary for the metadata attached to the service binding to contain the
3931 mapping information. The information is held in an operationSelector element which is a child element of
3932 the binding element.

3933 The base sca:operationSelector element is abstract and it has no attributes and no child elements. For a
3934 particular operationSelector, an extension subtype is defined, using substitution groups, for example:

- 3935 • <sca:operationSelector.XPath/>
3936 An operation selector that uses XPath to filter out specific messages and target them to
3937 particular named operations.

3938

3939 Specific operationSelectors can have elements that include either attributes or subelements or both.

3940 For details about specific operationSelectors, see the related SCA Binding specifications.

3941 7.4 Form of the URI of a Deployed Binding

3942 SCA Bindings specifications can choose to use the **structural URI** defined in the section "[Structural URI](#)
3943 [of Components](#)" above to derive a binding specific URI according to some Binding-related scheme. The
3944 relevant binding specification describes this.

3945 Alternatively, <binding/> elements have a @uri attribute, which is termed a bindingURI.

3946 If the bindingURI is specified on a given <binding/> element, the binding can use it to derive an endpoint
3947 URI relevant to the binding. The derivation is binding specific and is described by the relevant binding
3948 specification.

3949 For binding.sca, which is described in the SCA Assembly specification, this is as follows:

- 3950 • If the binding @uri attribute is specified on a reference, it identifies the target service in
3951 the SCA Domain by specifying the service's structural URI.
- 3952 • If the binding @uri attribute is specified on a service, it is ignored.

3953 7.4.1 Non-hierarchical URIs

3954 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) can make use of the @uri
3955 attribute, which is the complete representation of the URI for that service binding. Where the binding
3956 does not use the @uri attribute, the binding needs to offer a different mechanism for specifying the
3957 service address.

3958 7.4.2 Determining the URI scheme of a deployed binding

3959 One of the things that needs to be determined when building the effective URI of a deployed binding (i.e.
3960 endpoint) is the URI scheme. The process of determining the endpoint URI scheme is binding type
3961 specific.

3962 If the binding type supports a single protocol then there is only one URI scheme associated with it. In this
3963 case, that URI scheme is used.

3964 If the binding type supports multiple protocols, the binding type implementation determines the URI
3965 scheme by introspecting the binding configuration, which can include the policy sets associated with the
3966 binding.

3967 A good example of a binding type that supports multiple protocols is binding.ws, which can be configured
3968 by referencing either an "abstract" WSDL element (i.e. portType or interface) or a "concrete" WSDL
3969 element (i.e. binding or port). When the binding references a portType or Interface, the protocol and
3970 therefore the URI scheme is derived from the intents/policy sets attached to the binding. When the
3971 binding references a "concrete" WSDL element, there are two cases:

- 3972 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most
3973 common case. In this case, the URI scheme is given by the protocol/transport specified in the
3974 WSDL binding element.
- 3975 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,
3976 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"
3977 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined
3978 by looking at the policy sets attached to the binding.

3979 It is worth noting that an intent supported by a binding type can completely change the behavior of the
3980 binding. For example, when the intent "confidentiality/transport" is attached to an HTTP binding, SSL is
3981 turned on. This basically changes the URI scheme of the binding from "http" to "https".

3982

3983 7.5 SCA Binding

3984 The SCA binding element is defined by the pseudo-schema in Snippet 7-2.

```
3985 <binding.sca uri="xs:anyURI"?  
3986 name="xs:NCName"?
```

```

3987     requires="list of xs:QName"?
3988     policySets="list of xs:QName"??
3989     <wireFormat/>?
3990     <operationSelector/>?
3991     <requires/>*
3992     <policySetAttachment/>*
3993     <filters/>?
3994 </binding.sca>

```

3995 *Snippet 7-2: binding.sca pseudo-schema*

3996

3997 A **binding.sca** element has the attributes:

- 3998 • **uri (0..1)** - has the semantic:
 - 3999 – The @uri attribute can be omitted.
 - 4000 – If a <binding.sca/> element of a component reference specifies a URI via its @uri attribute, then
 - 4001 this provides a wire to a target service provided by another component. The form of the URI
 - 4002 which points to the service of a component that is in the same composite as the source
 - 4003 component is as follows:

```

4004         <component-name>/<service-name>
4005     or
4006         <component-name>/<service-name>/<binding-name>

```

4009 in cases where the service has multiple bindings present.

- 4010 – The circumstances under which the @uri attribute can be used are defined in the section
- 4011 ["Specifying the Target Service\(s\) for a Reference."](#)
- 4012 – **For a binding.sca of a component service, the @uri attribute MUST NOT be present. [ASM90005]**
- 4013 • **name (0..1)** – a name for the binding instance (an NCName), as defined for the base <binding/>
- 4014 element type.
- 4015 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 4016 description of this attribute.
- 4017 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 4018 description of this attribute.

4019 A **binding.sca** element has the child elements:

- 4020 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. binding.sca does not
- 4021 define any specific wireFormat elements.
- 4022 • **operationSelector(0..1)** - an operationSelector element that is used to match a particular message to
- 4023 a particular operation in the interface. binding.sca does not define any specific operationSelector
- 4024 elements.
- 4025 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
- 4026 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 4027 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
- 4028 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 4029 description of this element.
- 4030 • **filters: Filters (0..1)** – binding-specific filters on event instances. This subelement can only be used
- 4031 for bindings specified on channels, consumers and producers.

4032 The SCA binding can be used for (1) service interactions between references and services contained
 4033 within the SCA Domain, (2) event interaction between consumers and channels, and (3) event
 4034 interactions between producers and channels. The way in which this binding type is implemented is not
 4035 defined by the SCA specification and it can be implemented in different ways by different SCA runtimes.
 4036 The only requirement is that any specified qualities of service are implemented for the SCA binding type.

4037 Qualities of service for <binding.sca/> are expressed using intents and/or policy sets following the rules
4038 defined in the [Policy Framework specification \[SCA-POLICY\]](#).

4039 The SCA binding type is not intended to be an interoperable binding type. For interoperability, an
4040 interoperable binding type such as the Web service binding is used.

4041 An SCA runtime has to support the binding.sca binding type. See the section on [SCA Runtime
4042 conformance](#).

4043 A service definition with no binding element specified uses the SCA binding (see ASM50005 in section
4044 4.2 on Component Service). <binding.sca/> only has to be specified explicitly in override cases, or when
4045 a set of bindings is specified on a service definition and the SCA binding needs to be one of them.

4046 If a reference does not have a binding subelement specified, then the binding used is one of the bindings
4047 specified by the service provider, as long as the intents attached to the reference and the service are all
4048 honoured, as described in [the section on Component References](#).

4049 A channel, producer, or consumer with no binding element specified uses the SCA binding.

4050 If the interface of the service or reference is local, then the local variant of the SCA binding will be used. If
4051 the interface of the service or reference is remotable, then either the local or remote variant of the SCA
4052 binding will be used depending on whether source and target are co-located or not.

4053 If a <binding.sca/> element of a <component/> <reference/> specifies a URI via its @uri attribute, then
4054 this provides a wire to a target service provided by another component.

4055 The form of the URI which points to the service of a component that is in the same composite as the
4056 source component is as follows:

4057

- [<domain-component-name>/<service-name>](#)

4058 7.5.1 Example SCA Binding

4059 Snippet 7-3 shows the MyValueComposite.composite file for the MyValueComposite containing the
4060 service element for the MyValueService and a reference element for the StockQuoteService. Both the
4061 service and the reference use an SCA binding. The target for the reference is left undefined in this
4062 binding and would have to be supplied by the composite in which this composite is used.

```
4063 <?xml version="1.0" encoding="ASCII"?>
4064 <!-- Binding SCA example -->
4065 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4066           targetNamespace="http://foo.com"
4067           name="MyValueComposite" >
4068
4069   <service name="MyValueService" promote="MyValueComponent">
4070     <interface.java interface="services.myvalue.MyValueService"/>
4071     <binding.sca/>
4072     ...
4073   </service>
4074
4075   ...
4076
4077   <reference name="StockQuoteService"
4078             promote="MyValueComponent/StockQuoteReference">
4079     <interface.java interface="services.stockquote.StockQuoteService"/>
4080     <binding.sca/>
4081   </reference>
4082
4083 </composite>
```

4084 *Snippet 7-3: Example binding.sca*

4085 7.6 Web Service Binding

4086 SCA defines a Web services binding. This is described in a separate specification document [\[SCA-
4087 WSBINDING\]](#).

4088 **7.7 JMS Binding**

4089 SCA defines a JMS binding. This is described in a separate specification document [\[SCA-JMSBINDING\]](#).

8 Representation of Events and Event Types

4090

Events in SCA MAY have an **event type** associated with them. Each event type is identified by a unique **event type QName**.

4091
4092

An event can have no event type metadata associated with it - for example, this can be the case for events which are created by pre-existing non-SCA event sources.

4093
4094

SCA has a canonical **representation** of events in terms of XML Infoset and of event shapes in terms of XML Schema (see Section [Event Type Definition Language Used by SCA](#)). SCA event shapes are **describable** using XML Schema, although they don't have to be described using XML Schema – other type systems can be used. SCA events can have a wire format that is not XML.

4095
4096
4097
4098

Events can also have programming language specific representations. The details of the mapping between language specific formats and XML Infoset are defined by the SCA implementation language specifications.

4099
4100
4101

8.1 Event Type and Metadata

4102

In SCA, **event type** definition consist of the following:

4103

1. a unique **event type QName**
2. a set of business data. This data is also called the **shape** of the event. It is possible that the same shape is used by multiple event types.
3. optional additional metadata associated with events of this type, such as creation time, and is separate from the event business data.

4104
4105
4106

4107
4108

The shape of the event is defined in terms of an existing type system. Examples include XSD and Java. For event shape defined using XSD, this is done in terms of an XML global element declaration.

4109
4110

8.2 Event Type Definition Language Used by SCA

4111

SCA uses Web Services Event Descriptions (WS-EventDescriptions) **[WS-ED]** as its interoperable event type definition language. **Any event type definition used in SCA MUST be mappable to WS-EventDescriptions. [ASM16001].**

4112
4113
4114

Snippet 8-1 provides an example of an event type definition.

4115
4116

4117

```
<EventDescriptions xmlns="http://www.w3.org/2011/03/ws-evd"
  xmlns:foo="http://example.org/ed1"
  targetNamespace="http://example.org/ed1">
  <types>
    . . .
  </types>
  <eventType id="PrinterEvent" element="foo:PrinterEvent"/>
</EventDescriptions>
```

4118
4119
4120
4121
4122
4123
4124
4125

Snippet 8-1: Canonical Event Type Definition

4126

8.3 Events with No Event Type

4127

Events MAY have no event type metadata associated with them.

4128

From an SCA perspective (and in particular, when dealing with events of this kind in Filter statements), such events are given the special event type name of `sca:NULL` (a QName). This special event type name **MUST NOT** be used in event instances for its type metadata. It is reserved for use in composite

4129
4130
4131

4132 (such as in type filters on consumer, channels and type declarations on producers) to identify event
4133 instances that do not have any type metadata.

9 Filters: Selecting Subsets of Events

4134

4135 Event filters are used to select subsets of events from an event source. Event filters can be specified on
4136 consumers and on channels, and are then applied to the event instances that would otherwise be
4137 received by those consumers or channels.

4138 Filters can operate against various sorts of data relating to an event instance:

- 4139 • Event types
- 4140 • Event business data
- 4141 • Other event metadata

4142 The mechanism for expressing filters is extensible, so that in the future filters can be added that operate
4143 against other data, such as Properties of the Event channel.

4144 Filters can be expressed in a variety of dialects of filter language. It is possible to use different filter
4145 language dialects for different types of data - eg Event Metadata vs Business Data. It is possible to
4146 specify multiple filters (of the same type or different types) on a single consumer or channel.

4147 Each filter expression must resolve to a boolean where "false" means that the event instance is discarded
4148 and "true" means that the event instance is passed by the filter. Where multiple filters are present, they
4149 are logically "AND"ed together so that only messages that pass all of the filters are passed by the
4150 collection of filters.

4151 Filters can be specified on a channel, component consumer, composite consumer, or a consumer in the
4152 Component Type of an implementation. All filter expressions specified on a consumer, regardless of
4153 where (Component Type, Component or Composite) they are specified are logically "AND"ed together.

4154 Filters have no side effects and filters have no state. They are evaluated against a particular event
4155 instance and indicate whether the event passes the filter or not – there are no other implications. This
4156 means that the order in which multiple filters are applied does not matter – the same result occurs
4157 whatever the order.

9.1 Form of Explicit Filter Elements

4158 Explicit filters can be attached to various elements in SCA, such as consumers and channels. The syntax
4159 used to express the filters conveys three things:

- 4161 1. The type of data that the filter operates against (the "subject")
- 4162 2. The language used to express the filter (the "dialect")
- 4163 3. The filter expression itself.

4164 The choice of dialect might be constrained by the choice of subject; there are some dialect/subject
4165 combinations that do not make sense.

4166 The filters, if any, that are attached to a consumer or channel are all contained in a single <sca:filters>
4167 element. The filters themselves MUST appear as child element of <sca:filters> and any element that is
4168 included as a child element of <sca:filters> MUST be a filter. The QName of the element indicates the
4169 subject of the filter and its dialect; SCA provides element declarations for all the filter subjects that it
4170 defines.

4171 The element content is used to convey the expression, and is constrained by the dialect chosen.

4172 The SCA specification defines a number of predefined filter subject/dialect elements. These are described
4173 in the following sections, but are summarized in the pseudo-schema in Snippet 9-1.

4174

```
4175 <filters>  
4176   <eventType.sca qnames="list of xs:QName"? namespaces="list of xs:anyURI"?  
4177 />*  
4178   <body.xpath1> xs:string </body.xpath1>*
```



```
4179 <any>*
4180 </filters> ?
```

4181 *Snippet 9-1: filters Pseudo-Schema*

4182
4183 Note that the event filters are extensible, allowing new filter types to be defined as an extension and to be
4184 used at the place that the <any/> subelement is shown in the pseudo-schema. An SCA runtime is not
4185 required to support filter types not defined by this specification - but if an extended filter type is declared
4186 within a <filters/> element and the SCA runtime does not support that extended filter type, then the SCA
4187 runtime **MUST** generate an error when it encounters the declaration. [ASM17001]

4188 9.2 Event Type Filters

4189 Event type filters filter events based on the Event Type metadata of the event.

4190 Only one dialect is currently defined for event type with the element name <eventType.sca>

```
4191  
4192 <filters>
4193   <eventType.sca qnames="list of xs:QName"? namespaces="list of xs:anyURI"?
4194   />*
4195   ...
4196 </filters>
```

4197 *Snippet 9-2: Pseudo-Schema for Event Type filter*

4198
4199 In this dialect, a filter expression consists of either a list of one or more QNames specified as a value of
4200 the attribute @qname or a list of one or more namespace URIs specified as a value of the attribute
4201 @namespaces or both. This dialect filters on event types described using the WS-EventDescriptions
4202 **[WS-ED]** specification as described in Section [Event Type Definition Language Used by SCA](#).

4203 Each QName in the list **MUST** be associated with a Namespace URI. This association is performed using
4204 the namespace declarations that are in-scope where the QName expression appears (e.g. in the
4205 composite document containing sca:Filter element). Unprefixed QNames are permitted, provided there is
4206 a default namespace declaration in-scope where the QName expression appears. QNames that belong to
4207 no namespace are not allowed.

4208 A filter expressed in this dialect returns true if and only if either of the following is true:

- 4209 1. at least one of the QNames specified in the @qnames attribute matches the QName of the
4210 event's Event Type. In order for a match to occur both these conditions must be true:
 - 4211 • The associated Namespace URI's must contain an identical sequence of characters
4212 when expressed as Unicode code points.
 - 4213 • The local parts of each QName must contain an identical sequence of characters when
4214 expressed as Unicode code points.
- 4215 2. at least one of the namespaces specified in the @namespace attribute matches the namespace
4216 of the event's Event Type.
 - 4217 • The Namespace URI's must contain an identical sequence of characters when expressed
4218 as Unicode code points.

4219 9.2.1 Use of <eventType> on a Producer

4220 The element <eventType> (and by extension <eventType.sca>) can also be used on a component type
4221 producer, component producer or a composite producer to declare the event types produced by the
4222 producer.

4223 9.2.1.1 Use of <eventType.sca> on a Component Producer

4224 When the element <eventType.sca> is specified on a component producer and the @qnames attribute is
4225 omitted, the value defaults to the value of the @qnames attribute for the producer of the same name in
4226 the componentType of the implementation used by the component. If the @qnames attribute is omitted
4227 and if the corresponding producer in the componentType of the implementation also does not have this
4228 attribute, then the producer is unconstrained with respect to the Event Type QNames for the events that
4229 are sent by the producer. If the componentType has a value for @typeNames then the value of
4230 @typeNames for the component producer element MUST match that in the componentType.

4231 [ASM17002]

4232 When the element <eventType.sca> is specified on a component producer and the @namespaces
4233 attribute is omitted, the value defaults to the value of the @namespaces attribute for the producer of the
4234 same name in the componentType of the implementation used by the component. If the @namespaces
4235 attribute is omitted, and if the corresponding producer in the componentType of the implementation also
4236 does not have this attribute, then the producer is unconstrained with respect to the Even Type
4237 Namespace URI for the events that are sent by the producer. If the componentType has a value for
4238 @namespaces then the value of @namespaces for the component producer element MUST match that
4239 in the componentType. [ASM17003]

4240 Note that both attributes @qnames and @namespaces can be used together. If both attributes are
4241 specified, then the component producer declares that it might send events whose Event Type is either
4242 listed in the @names attribute or whose Event Type belongs to one of the Namespaces listed in the
4243 @namespaces attribute.

4244 9.2.1.2 Use of <eventType.sca> on a Composite Producer

4245 When the element <eventType.sca> is specified on a component producer and the @qnames attribute is
4246 omitted, the value defaults to the value of the @qnames attribute, if present, of the associated component
4247 producer or the componentType of the component producer that is promoted.

4248 If the associated component producer or the componentType producer has a value for @qnames then
4249 the value of @qnames, if present, for the composite producer element MUST match that in the component
4250 propducer or the componentType producer. [ASM17004]

4251 If the @namespaces attribute is omitted, the value defaults to the value of the @namespaces attribute, if
4252 present, of the associated component producer or the componentType of the component producer that is
4253 promoted.

4254 If the associated component producer or the componentType producer has a value for @namespaces
4255 then the value of @namespaces, if present, for the composite producer element MUST match that in the
4256 component producer or the componentType producer.

4257 Note that both attributes @qnames and @namespaces can be used together. If both attributes are
4258 specified, then the producer declares that it might send events whose Event Type is either listed in the
4259 @qnames attribute or whose Event Type belongs to one of the Namespaces listed in the @namespaces
4260 attribute.

4261 9.2.2 Event Type Filter Examples

4262 A filter that expresses interest in the events of types ns1:printer or ns2:printer:

```
4263 <eventType.sca qnames="ns1:printer ns2:printer" />
```

4264 A filter that expresses interest in events that do not have a type metadata:

```
4265 <eventType.sca qnames="sca:NULL" />
```

4266 A filter that expresses interest in events that either do not have type metadata or are of type ns1:printer:

```
4267 <eventType.sca qnames="sca:NULL ns1:printer" />
```

4268 A filter that expresses interest in events whose type belongs to one of the two namespaces

4269 "http://example.org/ns1" or "http://example.org/ns2":

```
4270 <eventType.sca namespaces="http://example.org/ns1 http://example.org/ns2" />
```

4271 A filter that expresses interest in events whose type belongs to the namespaces
4272 http://example.org/ns2 or is of type ns1:printer or is untyped:
4273 <eventType.sca qnames="ns1:printer sca:NULL"
4274 namespaces="http://example.org/ns2" />

4275 **9.3 Business Data Filters**

4276 Business data filters filter events based on the business data contained within the event.
4277 The following dialects are defined - xpath1.

4278 **9.3.1 XPATH 1.0 Dialect**

4279 Filter element QName: <sca:body.xpath1>

4280 The Filter expression (content of the element <body.xpath1>) is an XPath 1.0 expression (not a
4281 predicate) whose context is:

- 4282 • Context Node: the root element of the document being searched based upon the subject. In this
4283 case (the Business Data Subject) it is the root element of the event business data.
- 4284 • Context Position: 1
- 4285 • Context Size: 1
- 4286 • Variable Binding: None
- 4287 • Function Libraries: Core function library
- 4288 • Namespace Declarations: Any namespace declarations in-scope where the XPath expression
4289 appears (e.g. in the SCDL document containing sca:Filter element)

4290 This XPath expression can evaluate to one of four possible types: a node-set, a boolean, a number or a
4291 string. These result types are converted to a boolean value as follows:

- 4292 • Node-set – false if no nodes, true otherwise
- 4293 • boolean – no conversion
- 4294 • string – false is empty string, true otherwise
- 4295 • number – false if 0, true otherwise

4296

10 SCA Definitions

4297 There are a variety of SCA artifacts which are generally useful and which are not specific to a particular
4298 composite or a particular component. These shared artifacts include intents, policy sets, bindings,
4299 binding type definitions, implementation type definitions, and external attachment definitions.

4300 All of these artifacts within an SCA Domain are defined in SCA contributions in files called META-
4301 INF/definitions.xml (relative to the contribution base URI). An SCA runtime MUST make available to the
4302 Domain all the artifacts contained within the definitions.xml files in the Domain. [ASM10002] An SCA
4303 runtime MUST reject a definitions.xml file that does not conform to the sca-definitions.xsd schema.
4304 [ASM10003]

4305 Although the definitions are specified within a single SCA contribution, the definitions are visible
4306 throughout the Domain. Because of this, all of the QNames for the definitions contained in definitions.xml
4307 files MUST be unique within the Domain.. [ASM10001] The definitions.xml file contains a definitions
4308 element that conforms to the pseudo-schema shown in Snippet 10-1:

4309

```
4310 <?xml version="1.0" encoding="ASCII"?>
4311 <!-- Composite schema snippet -->
4312 <definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4313             targetNamespace="xs:anyURI">
4314
4315     <sca:intent/>*
4316
4317     <sca:policySet/>*
4318
4319     <sca:bindingType/>*
4320
4321     <sca:implementationType/>*
4322
4323 </definitions>
```

4324 *Snippet 10-1: definitions Pseudo-Schema*

4325

4326 The definitions element has the attribute:

- 4327 • **targetNamespace (1..1)** – the namespace into which the child elements of this definitions element
4328 are placed (used for artifact resolution)

4329 The definitions element contains child elements – intent, policySet, bindingType, implementationType and
4330 externalAttachmen. These elements are described elsewhere in this specification or in the [Policy
4331 Framework specification \[SCA-POLICY\]](#).

4332

11 Extension Model

4333 The assembly model can be extended with support for new interface types, implementation types and
4334 binding types. The extension model is based on XML schema substitution groups. There are five XML
4335 Schema substitution group heads defined in the SCA namespace: **interface**, **implementation**, **binding**,
4336 **import** and **export** for interface types, implementation types, binding types, import types and export
4337 types, respectively.

4338 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [SCA-
4339 COMMON-JAVA], [SCA-JAVA], [SCA BPEL], [SCA-CPP-Client], [SCA-C-Client], [SCA-WSBINDING],
4340 [SCA-JMSBINDING] as examples) use these XML Schema substitution groups to define some basic
4341 types of interfaces, implementations and bindings, but additional types can be defined as needed, where
4342 support for these extra ones is available from the runtime. The interface type elements, implementation
4343 type elements, binding type elements, import type elements and export type elements defined by the SCA
4344 specifications are all part of the SCA namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200912"),
4345 as indicated in their respective schemas. New interface types, implementation types and binding types
4346 that are defined using this extensibility model, which are not part of these SCA specifications are defined
4347 in namespaces other than the SCA namespace.

4348 The "." notation is used in naming elements defined by the SCA specifications (e.g. <implementation.java
4349 ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel extensibility approach but as a naming
4350 convention that improves usability of the SCA assembly language.

4351

4352 A conforming implementation type, interface type, import type or export type MUST meet the
4353 requirements in "Implementation Type Documentation Requirements for SCA Assembly Model Version
4354 1.2 Specification". [ASM11001]

4355 A binding extension element MUST be declared as an element in the substitution group of the sca:binding
4356 element. [ASM11002] A binding extension element MUST be declared to be of a type which is an
4357 extension of the sca:Binding type. [ASM11003]

11.1 Defining an Interface Type

4359 Snippet 11-1 shows the base definition for the **interface** element and **Interface** type contained in **sca-
4360 core.xsd**; see sca-core.xsd for the complete schema.

4361

```
4362 <?xml version="1.0" encoding="UTF-8"?>  
4363 <!-- (c) Copyright SCA Collaboration 2006 -->  
4364 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
4365         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4366         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4367         elementFormDefault="qualified">  
4368  
4369     ...  
4370  
4371     <element name="interface" type="sca:Interface" abstract="true"/>  
4372     <complexType name="Interface" abstract="true">  
4373         <choice minOccurs="0" maxOccurs="unbounded">  
4374             <element ref="sca:requires"/>  
4375             <element ref="sca:policySetAttachment"/>  
4376         </choice>  
4377         <attribute name="remotable" type="boolean" use="optional"/>  
4378         <attribute name="requires" type="sca:listOfQNames" use="optional"/>  
4379         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>  
4380     </complexType>  
4381  
4382     ...
```

```
4383
4384 </schema>
```

4385 *Snippet 11-1: interface and Interface Schema*

4386
4387 Snippet 11-2 is an example of how the base definition is extended to support Java interfaces. The snippet
4388 shows the definition of the **interface.java** element and the **JavaInterface** type contained in **sca-**
4389 **interface-java.xsd**.

```
4390
4391 <?xml version="1.0" encoding="UTF-8"?>
4392 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4393         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4394         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
4395
4396     <element name="interface.java" type="sca:JavaInterface"
4397           substitutionGroup="sca:interface"/>
4398     <complexType name="JavaInterface">
4399       <complexContent>
4400         <extension base="sca:Interface">
4401           <attribute name="interface" type="NCName"
4402             use="required"/>
4403         </extension>
4404       </complexContent>
4405     </complexType>
4406 </schema>
```

4407 *Snippet 11-2: Extending interface to interface.java*

4408
4409 Snippet 11-3 is an example of how the base definition can be extended by other specifications to support
4410 a new interface not defined in the SCA specifications. The snippet shows the definition of the **my-**
4411 **interface-extension** element and the **my-interface-extension-type** type.

```
4412
4413 <?xml version="1.0" encoding="UTF-8"?>
4414 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4415         targetNamespace="http://www.example.org/myextension"
4416         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4417         xmlns:tns="http://www.example.org/myextension">
4418
4419     <element name="my-interface-extension"
4420           type="tns:my-interface-extension-type"
4421           substitutionGroup="sca:interface"/>
4422     <complexType name="my-interface-extension-type">
4423       <complexContent>
4424         <extension base="sca:Interface">
4425           ...
4426         </extension>
4427       </complexContent>
4428     </complexType>
4429 </schema>
```

4430 *Snippet 11-3: Example interface extension*

4431 11.2 Defining an Implementation Type

4432 Snippet 11-4 shows the base definition for the **implementation** element and **Implementation** type
4433 contained in **sca-core.xsd**; see **sca-core.xsd** for complete schema.

```
4434
4435 <?xml version="1.0" encoding="UTF-8"?>
4436 <!-- (c) Copyright SCA Collaboration 2006 -->
```

```

4437 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4438         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4439         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4440         elementFormDefault="qualified">
4441
4442     ...
4443
4444     <element name="implementation" type="sca:Implementation"
4445             abstract="true"/>
4446     <complexType name="Implementation" abstract="true">
4447         <complexContent>
4448             <extension base="sca:CommonExtensionBase">
4449                 <choice minOccurs="0" maxOccurs="unbounded">
4450                     <element ref="sca:requires"/>
4451                     <element ref="sca:policySetAttachment"/>
4452                 </choice>
4453                 <attribute name="requires" type="sca:listOfQNames"
4454                             use="optional"/>
4455                 <attribute name="policySets" type="sca:listOfQNames"
4456                             use="optional"/>
4457             </extension>
4458         </complexContent>
4459     </complexType>
4460
4461     ...
4462
4463 </schema>

```

4464 *Snippet 11-4: implementation and Implementation Schema*

4465
4466 Snippet 11-5 shows how the base definition is extended to support Java implementation. The snippet
4467 shows the definition of the **implementation.java** element and the **JavaImplementation** type contained in
4468 **sca-implementation-java.xsd**.

```

4469
4470 <?xml version="1.0" encoding="UTF-8"?>
4471 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4472         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4473         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
4474
4475     <element name="implementation.java" type="sca:JavaImplementation"
4476             substitutionGroup="sca:implementation"/>
4477     <complexType name="JavaImplementation">
4478         <complexContent>
4479             <extension base="sca:Implementation">
4480                 <attribute name="class" type="NCName"
4481                             use="required"/>
4482             </extension>
4483         </complexContent>
4484     </complexType>
4485 </schema>

```

4486 *Snippet 11-5: Extending implementation to implementation.java*

4487
4488 Snippet 11-6 is an example of how the base definition can be extended by other specifications to support
4489 a new implementation type not defined in the SCA specifications. The snippet shows the definition of the
4490 **my-impl-extension** element and the **my-impl-extension-type** type.

```

4491
4492 <?xml version="1.0" encoding="UTF-8"?>
4493 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

4494     targetNamespace="http://www.example.org/myextension"
4495     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4496     xmlns:tns="http://www.example.org/myextension">
4497
4498     <element name="my-impl-extension" type="tns:my-impl-extension-type"
4499         substitutionGroup="sca:implementation"/>
4500     <complexType name="my-impl-extension-type">
4501         <complexContent>
4502             <extension base="sca:Implementation">
4503                 ...
4504             </extension>
4505         </complexContent>
4506     </complexType>
4507 </schema>

```

4508 *Snippet 11-6: Example implementation extension*

4509

4510 In addition to the definition for the new implementation instance element, there needs to be an associated
4511 implementationType element which provides metadata about the new implementation type. The pseudo
4512 schema for the implementationType element is shown in Snippet 11-7:

4513

```

4514 <implementationType type="xs:QName"
4515     alwaysProvides="list of intent xs:QName"
4516     mayProvide="list of intent xs:QName"/>

```

4517 *Snippet 11-7: implementationType Pseudo-Schema*

4518

4519 The implementation type has the attributes:

- 4520 • **type (1..1)** – the type of the implementation to which this implementationType element applies. This
4521 is intended to be the QName of the implementation element for the implementation type, such as
4522 "sca:implementation.java"
- 4523 • **alwaysProvides (0..1)** – a set of intents which the implementation type always provides. See the
4524 [Policy Framework specification \[SCA-POLICY\]](#) for details.
- 4525 • **mayProvide (0..1)** – a set of intents which the implementation type provides only when the intent is
4526 attached to the implementation element. See the [Policy Framework specification \[SCA-POLICY\]](#) for
4527 details.

4528 11.3 Defining a Binding Type

4529 Snippet 11-8 shows the base definition for the **binding** element and **Binding** type contained in **sca-**
4530 **core.xsd**; see sca-core.xsd for complete schema.

4531

```

4532 <?xml version="1.0" encoding="UTF-8"?>
4533 <!-- binding type schema snippet -->
4534 <!-- (c) Copyright SCA Collaboration 2006, 2009 -->
4535 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4536     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4537     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4538     elementFormDefault="qualified">
4539
4540     ...
4541
4542     <element name="binding" type="sca:Binding" abstract="true"/>
4543     <complexType name="Binding">
4544         <attribute name="uri" type="anyURI" use="optional"/>
4545         <attribute name="name" type="NCName" use="optional"/>

```



```

4546     <attribute name="requires" type="sca:listOfQNames"
4547         use="optional"/>
4548     <attribute name="policySets" type="sca:listOfQNames"
4549         use="optional"/>
4550 </complexType>
4551
4552     ...
4553
4554 </schema>

```

4555 *Snippet 11-8: binding and Binding Schema*

4556

4557 Snippet 11-9 is an example of how the base definition is extended to support Web service binding. The
4558 snippet shows the definition of the **binding.ws** element and the **WebServiceBinding** type contained in
4559 **sca-binding-webservice.xsd**.

4560

```

4561 <?xml version="1.0" encoding="UTF-8"?>
4562 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4563     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4564     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
4565
4566     <element name="binding.ws" type="sca:WebServiceBinding"
4567 substitutionGroup="sca:binding"/>
4568     <complexType name="WebServiceBinding">
4569         <complexContent>
4570             <extension base="sca:Binding">
4571                 <attribute name="port" type="anyURI" use="required"/>
4572             </extension>
4573         </complexContent>
4574     </complexType>
4575 </schema>

```

4576 *Snippet 11-9: Extending binding to binding.ws*

4577

4578 Snippet 11-10 is an example of how the base definition can be extended by other specifications to
4579 support a new binding not defined in the SCA specifications. The snippet shows the definition of the **my-**
4580 **binding-extension** element and the **my-binding-extension-type** type.

4581

```

4582 <?xml version="1.0" encoding="UTF-8"?>
4583 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4584     targetNamespace="http://www.example.org/myextension"
4585     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4586     xmlns:tns="http://www.example.org/myextension">
4587
4588     <element name="my-binding-extension"
4589         type="tns:my-binding-extension-type"
4590         substitutionGroup="sca:binding"/>
4591     <complexType name="my-binding-extension-type">
4592         <complexContent>
4593             <extension base="sca:Binding">
4594                 ...
4595             </extension>
4596         </complexContent>
4597     </complexType>
4598 </schema>

```

4599 *Snippet 11-10: Example binding extension*

4600

4601 In addition to the definition for the new binding instance element, there needs to be an associated
4602 bindingType element which provides metadata about the new binding type. The pseudo schema for the
4603 bindingType element is shown in Snippet 11-11:

4604

```
4605 <bindingType type="xs:QName"  
4606     alwaysProvides="list of intent QNames"?  
4607     mayProvide = "list of intent QNames"?/>
```

4608 *Snippet 11-11: bindingType Pseudo-Schema*

4609

4610 The binding type has the following attributes:

- 4611 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is intended to be
4612 the QName of the binding element for the binding type, such as "sca:binding.ws"
- 4613 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See the [Policy](#)
4614 [Framework specification \[SCA-POLICY\]](#) for details.
- 4615 • **mayProvide (0..1)** – a set of intents which the binding type provides only when the intent is attached
4616 to the binding element. See the [Policy Framework specification \[SCA-POLICY\]](#) for details.

4617 11.4 Defining an Import Type

4618 Snippet 11-12 shows the base definition for the *import* element and *Import* type contained in *sca-*
4619 *core.xsd*; see sca-core.xsd for complete schema.

4620

```
4621 <?xml version="1.0" encoding="UTF-8"?>  
4622 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved. OASIS trademark,  
4623 IPR and other policies apply. -->  
4624 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
4625     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4626     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4627     elementFormDefault="qualified">  
4628  
4629     ...  
4630  
4631     <!-- Import -->  
4632     <element name="importBase" type="sca:Import" abstract="true" />  
4633     <complexType name="Import" abstract="true">  
4634         <complexContent>  
4635             <extension base="sca:CommonExtensionBase">  
4636                 <sequence>  
4637                     <any namespace="##other" processContents="lax" minOccurs="0"  
4638                         maxOccurs="unbounded"/>  
4639                 </sequence>  
4640             </extension>  
4641         </complexContent>  
4642     </complexType>  
4643  
4644     <element name="import" type="sca:ImportType"  
4645         substitutionGroup="sca:importBase"/>  
4646     <complexType name="ImportType">  
4647         <complexContent>  
4648             <extension base="sca:Import">  
4649                 <attribute name="namespace" type="string" use="required"/>  
4650                 <attribute name="location" type="anyURI" use="required"/>  
4651             </extension>  
4652         </complexContent>  
4653     </complexType>  
4654  
4655     ...
```

4656
4657

```
</schema>
```

4658 *Snippet 11-12: import and Import Schema*

4659

4660 Snippet 11-13 shows how the base import definition is extended to support Java imports. In the import
4661 element, the namespace is expected to be an XML namespace, an `import.java` element uses a Java
4662 package name instead. The snippet shows the definition of the ***import.java*** element and the
4663 ***JavaImportType*** type contained in ***sca-import-java.xsd***.

4664

```
4665 <?xml version="1.0" encoding="UTF-8"?>  
4666 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
4667         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4668         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
4669  
4670     <element name="import.java" type="sca:JavaImportType"  
4671             substitutionGroup="sca:importBase"/>  
4672     <complexType name="JavaImportType">  
4673         <complexContent>  
4674             <extension base="sca:Import">  
4675                 <attribute name="package" type="xs:String" use="required"/>  
4676                 <attribute name="location" type="xs:AnyURI" use="optional"/>  
4677             </extension>  
4678         </complexContent>  
4679     </complexType>  
4680 </schema>
```

4681 *Snippet 11-13: Extending import to import.java*

4682

4683 Snippet 11-14 shows an example of how the base definition can be extended by other specifications to
4684 support a new interface not defined in the SCA specifications. The snippet shows the definition of the ***my-***
4685 ***import-extension*** element and the ***my-import-extension-type*** type.

4686

```
4687 <?xml version="1.0" encoding="UTF-8"?>  
4688 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
4689         targetNamespace="http://www.example.org/myextension"  
4690         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
4691         xmlns:tns="http://www.example.org/myextension">  
4692  
4693     <element name="my-import-extension"  
4694             type="tns:my-import-extension-type"  
4695             substitutionGroup="sca:importBase"/>  
4696     <complexType name="my-import-extension-type">  
4697         <complexContent>  
4698             <extension base="sca:Import">  
4699                 ...  
4700             </extension>  
4701         </complexContent>  
4702     </complexType>  
4703 </schema>
```

4704 *Snippet 11-14: Example import extension*

4705

4706 For a complete example using this extension point, see the definition of ***import.java*** in the [SCA Java](#)
4707 [Common Annotations and APIs Specification \[SCA-Common-Java\]](#).

4708 11.5 Defining an Export Type

4709 Snippet 11-15 shows the base definition for the **export** element and **ExportType** type contained in **sca-**
4710 **core.xsd**; see appendix for complete schema.

4711

```
4712 <?xml version="1.0" encoding="UTF-8"?>
4713 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved. OASIS trademark,
4714 IPR and other policies apply. -->
4715 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4716 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4717 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4718 elementFormDefault="qualified">
4719
4720 ...
4721 <!-- Export -->
4722 <element name="exportBase" type="sca:Export" abstract="true" />
4723 <complexType name="Export" abstract="true">
4724 <complexContent>
4725 <extension base="sca:CommonExtensionBase">
4726 <sequence>
4727 <any namespace="##other" processContents="lax" minOccurs="0"
4728 maxOccurs="unbounded"/>
4729 </sequence>
4730 </extension>
4731 </complexContent>
4732 </complexType>
4733
4734 <element name="export" type="sca:ExportType"
4735 substitutionGroup="sca:exportBase"/>
4736 <complexType name="ExportType">
4737 <complexContent>
4738 <extension base="sca:Export">
4739 <attribute name="namespace" type="string" use="required"/>
4740 </extension>
4741 </complexContent>
4742 </complexType>
4743 ...
4744 </schema>
```

4745 *Snippet 11-15: export and Export Schema*

4746

4747 Snippet 11-16 shows how the base definition is extended to support Java exports. In a base **export**
4748 element, the **@namespace** attribute specifies XML namespace being exported. An **export.java** element
4749 uses a **@package** attribute to specify the Java package to be exported. The snippet shows the definition
4750 of the **export.java** element and the **JavaExport** type contained in **sca-export-java.xsd**.

4751

```
4752 <?xml version="1.0" encoding="UTF-8"?>
4753 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4754 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4755 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
4756
4757 <element name="export.java" type="sca:JavaExportType"
4758 substitutionGroup="sca:exportBase"/>
4759 <complexType name="JavaExportType">
4760 <complexContent>
4761 <extension base="sca:Export">
4762 <attribute name="package" type="xs:String" use="required"/>
4763 </extension>
4764 </complexContent>
4765 </complexType>
```

4766 </schema>

4767 *Snippet 11-16: Extending export to export.java*

4768

4769 Snippet 11-17 we shows an example of how the base definition can be extended by other specifications
4770 to support a new interface not defined in the SCA specifications. The snippet shows the definition of the
4771 **my-export-extension** element and the **my-export-extension-type** type.

4772

```
4773 <?xml version="1.0" encoding="UTF-8"?>
4774 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4775         targetNamespace="http://www.example.org/myextension"
4776         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200903"
4777         xmlns:tns="http://www.example.org/myextension">
4778
4779     <element name="my-export-extension"
4780             type="tns:my-export-extension-type"
4781             substitutionGroup="sca:exportBase"/>
4782     <complexType name="my-export-extension-type">
4783         <complexContent>
4784             <extension base="sca:Export">
4785                 ...
4786             </extension>
4787         </complexContent>
4788     </complexType>
4789 </schema>
```

4790 *Snippet 11-17: Example export extension*

4791

4792 For a complete example using this extension point, see the definition of **export.java** in the [SCA Java](#)
4793 [Common Annotations and APIs Specification \[SCA-Common-Java\]](#).

4794 12 Packaging and Deployment

4795 This section describes the SCA Domain and the packaging and deployment of artifacts contributed to the
4796 Domain.

4797 12.1 Domains

4798 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series of
4799 interconnected runtime nodes.

4800 A single SCA Domain defines the boundary of visibility for all SCA mechanisms. For example, SCA wires
4801 can only be used to connect components within a single SCA Domain. Connections to services outside
4802 the Domain use binding specific mechanisms for addressing services (such as WSDL endpoint URIs).
4803 Also, SCA mechanisms such as intents and policySets can only be used in the context of a single
4804 Domain. In general, external clients of a service that is developed and deployed using SCA are not able
4805 to tell that SCA is used to implement the service – it is an implementation detail.

4806 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification and
4807 is expected to be highly variable. An SCA Domain typically represents an area of business functionality
4808 controlled by a single organization. For example, an SCA Domain might be the whole of a business, or it
4809 might be a department within a business.

4810 As an example, for the accounts department in a business, the SCA Domain might cover all finance-
4811 related functions, and it might contain a series of composites dealing with specific areas of accounting,
4812 with one for Customer accounts and another dealing with Accounts Payable.

4813 An SCA Domain has the following:

- 4814 • A virtual domain-level composite whose components are deployed and running
- 4815 • A set of *installed contributions* that contain implementations, interfaces and other artifacts necessary
4816 to execute components
- 4817 • A set of logical services for manipulating the set of contributions and the virtual domain-level
4818 composite.

4819 The information associated with an SCA Domain can be stored in many ways, including but not limited to
4820 a specific filesystem structure or a repository.

4821 12.2 Contributions

4822 An SCA Domain might need a large number of different artifacts in order to work. These artifacts include
4823 artifacts defined by SCA and other artifacts such as object code files and interface definition files. The
4824 SCA-defined artifact types are all XML documents. The root elements of the different SCA definition
4825 documents are: composite, componentType and definitions. XML artifacts that are not defined by SCA
4826 but which are needed by an SCA Domain include XML Schema documents, WSDL documents, and
4827 BPEL documents. SCA constructs, like other XML-defined constructs, use XML qualified names for their
4828 identity (i.e. namespace + local name).

4829 Non-XML artifacts are also needed within an SCA Domain. The most obvious examples of such non-
4830 XML artifacts are Java, C++ and other programming language files necessary for component
4831 implementations. Since SCA is extensible, other XML and non-XML artifacts might also be needed.

4832 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This format is
4833 not the only packaging format that an SCA runtime can use. SCA allows many different packaging
4834 formats, but it is necessary for an SCA runtime to support the ZIP contribution format. When using the
4835 ZIP format for deploying a contribution, this specification does not specify whether that format is retained
4836 after deployment. For example, a Java EE based SCA runtime could convert the ZIP package to an EAR
4837 package. SCA expects certain characteristics of any packaging:

- 4838 • For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA
4839 as a hierarchy of resources based off of a single root [ASM12001]
- 4840 • Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy
4841 named META-INF [ASM12002]
- 4842 • Within any contribution packaging a document SHOULD exist directly under the META-INF directory
4843 named sca-contribution.xml which lists the SCA Composites within the contribution that are runnable.
4844 [ASM12003]

4845 The same document can also list namespaces of constructs that are defined within the contribution
4846 and which are available for use by other contributions, through export elements.

4847 These additional elements might not be physically present in the packaging, but might be generated
4848 based on the definitions and references that are present, or they might not exist at all if there are no
4849 unresolved references.

4850 See the section "SCA Contribution Metadata Document" for details of the format of this file.

4851 To illustrate that a variety of packaging formats can be used with SCA, the following are examples of
4852 formats that might be used to package SCA artifacts and metadata (as well as other artifacts) as a
4853 contribution:

- 4854 • A filesystem directory
- 4855 • An OSGi bundle
- 4856 • A compressed directory (zip, gzip, etc)
- 4857 • A JAR file (or its variants – WAR, EAR, etc)

4858 Contributions do not contain other contributions. If the packaging format is a JAR file that contains other
4859 JAR files (or any similar nesting of other technologies), the internal files are not treated as separate SCA
4860 contributions. It is up to the implementation to determine whether the internal JAR file is represented as a
4861 single artifact in the contribution hierarchy or whether all of the contents are represented as separate
4862 artifacts.

4863 A goal of SCA's approach to deployment is that the contents of a contribution do not need to be modified
4864 in order to install and use the contents of the contribution in a Domain.

4865 12.2.1 SCA Artifact Resolution

4866 Contributions can be self-contained, in that all of the artifacts necessary to run the contents of the
4867 contribution are found within the contribution itself. However, it can also be the case that the contents of
4868 the contribution make one or many references to artifacts that are not contained within the contribution.
4869 These references can be to SCA artifacts such as composites or they can be to other artifacts such as
4870 WSDL files, XSD files or to code artifacts such as Java class files and BPEL process files. Note: This
4871 form of artifact resolution does not apply to imports of composite files, as described in Section 6.6.

4872 A contribution can use some artifact-related or packaging-related means to resolve artifact references.
4873 Examples of such mechanisms include:

- 4874 • @wsdlLocation and @schemaLocation attributes in references to WSDL and XSD schema artifacts
4875 respectively
- 4876 • OSGi bundle mechanisms for resolving Java class and related resource dependencies

4877 Where present, artifact-related or packaging-related artifact resolution mechanisms MUST be used by the
4878 SCA runtime to resolve artifact dependencies. [ASM12005] The SCA runtime MUST raise an error if an
4879 artifact cannot be resolved using these mechanisms, if present. [ASM12021]

4880 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanism is can be
4881 used where no other mechanisms are available, for example in cases where the mechanisms used by the
4882 various contributions in the same SCA Domain are different. An example of this is where an OSGi
4883 Bundle is used for one contribution but where a second contribution used by the first one is not
4884 implemented using OSGi - e.g. the second contribution relates to a mainframe COBOL service whose
4885 interfaces are declared using a WSDL which is accessed by the first contribution.

4886 The SCA artifact resolution is likely to be most useful for SCA Domains containing heterogeneous
4887 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to work
4888 across different kinds of contribution.

4889 SCA artifact resolution works on the principle that a contribution which needs to use artifacts defined
4890 elsewhere expresses these dependencies using **import** statements in metadata belonging to the
4891 contribution. A contribution controls which artifacts it makes available to other contributions through
4892 **export** statements in metadata attached to the contribution. SCA artifact resolution is a general
4893 mechanism that can be extended for the handling of specific types of artifact. The general mechanism
4894 that is described in the following paragraphs is mainly intended for the handling of XML artifacts. Other
4895 types of artifacts, for example Java classes, use an extended version of artifact resolution that is
4896 specialized to their nature (eg. instead of "namespaces", Java uses "packages"). Descriptions of these
4897 more specialized forms of artifact resolution are contained in the SCA specifications that deal with those
4898 artifact types.

4899 Import and export statements for XML artifacts work at the level of namespaces - so that an import
4900 statement declares that artifacts from a specified namespace are found in other contributions, while an
4901 export statement makes all the artifacts from a specified namespace available to other contributions.

4902 An import declaration can simply specify the namespace to import. In this case, the locations which are
4903 searched for artifacts in that namespace are the contribution(s) in the Domain which have export
4904 declarations for the same namespace, if any. Alternatively an import declaration can specify a location
4905 from which artifacts for the namespace are obtained, in which case, that specific location is searched.

4906 **There can be multiple import declarations for a given namespace. Where multiple import declarations
4907 are made for the same namespace, all the locations specified MUST be searched in lexical order.**

4908 **[ASM12022]**

4909 For an XML namespace, artifacts can be declared in multiple locations - for example a given namespace
4910 can have a WSDL declared in one contribution and have an XSD defining XML data types in a second
4911 contribution.

4912 If the same artifact is declared in multiple locations, this is not an error. The first location as defined by
4913 lexical order is chosen. If no locations are specified no order exists and the one chosen is implementation
4914 dependent.

4915 **When a contribution contains a reference to an artifact from a namespace that is declared in an import
4916 statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the
4917 SCA runtime MUST resolve artifacts in the following order:**

4918 1. **from the locations identified by the import statement(s) for the namespace.
4919 Locations MUST NOT be searched recursively in order to locate artifacts (i.e. only
4920 a one-level search is performed).**

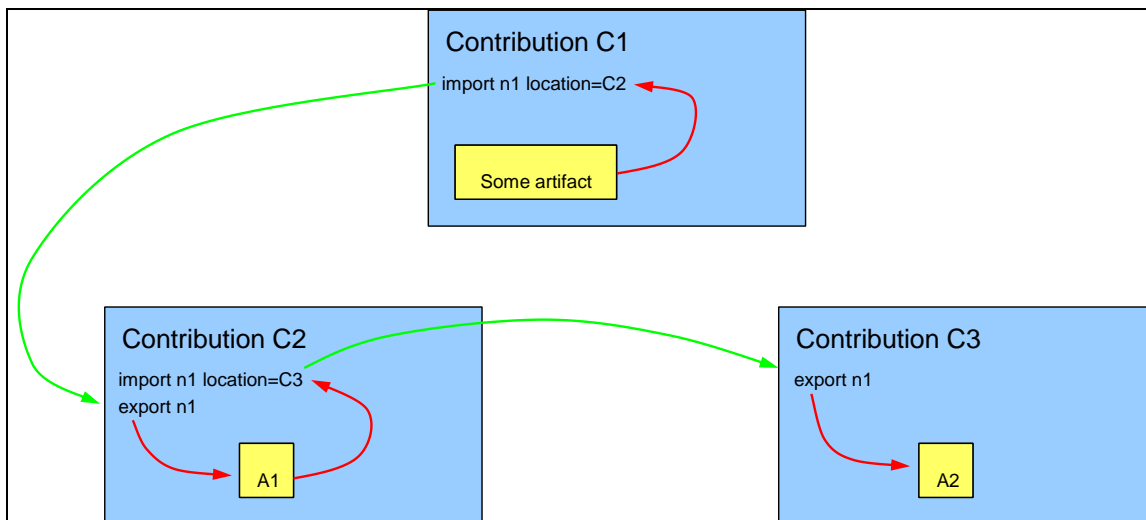
4921 2. **from the contents of the contribution itself. [ASM12023]**

4922 **Checking for errors in artifacts MUST NOT be done for artifacts in the Installed state (ie where the
4923 artifacts are simply part of installed contributions) [ASM12031]**

4924 For example:

- 4925 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the "n1"
4926 namespace from a second contribution "C2".
- 4927 • in contribution "C2" the artifact "A1" in the "n1" namespace references an artifact "A2" also in the "n1"
4928 namespace", which is resolved through an import of the "n1" namespace in "C2" which specifies the
4929 location "C3".

4930



4931

Figure 12-1: Example of SCA Artifact Resolution between Contributions

4932

4933

4934 The "A2" artifact is contained within the third contribution "C3" from which it is resolved by the contribution
4935 "C2". The "C3" contribution is never used to resolve artifacts directly for the "C1" contribution, since "C3"
4936 is not declared as an import location for "C1".

4937 For example, if for a contribution "C1", an import is used to resolve a composite "X1" contained in
4938 contribution "C2", and composite "X1" contains references to other artifacts such as WSDL files or XSDs,
4939 those references in "X1" are resolved in the context of contribution "C2" and not in the context of
4940 contribution "C1".

4941 The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through resolving an
4942 import statement. [ASM12024]

4943 The SCA runtime MUST raise an error if an artifact cannot be resolved by using artifact-related or
4944 packaging-related artifact resolution mechanisms, if present, by searching locations identified by the
4945 import statements of the contribution, if present, and by searching the contents of the contribution.
4946 [ASM12025]

4947 12.2.2 SCA Contribution Metadata Document

4948 The contribution can contain a document that declares runnable composites, exported definitions and
4949 imported definitions. The document is found at the path of META-INF/sca-contribution.xml relative to the
4950 root of the contribution. Frequently some SCA metadata needs to be specified by hand while other
4951 metadata is generated by tools (such as the <import> elements described below). To accommodate this,
4952 it is also possible to have an identically structured document at META-INF/sca-contribution-
4953 generated.xml. If this document exists (or is generated on an as-needed basis), it will be merged into the
4954 contents of sca-contribution.xml, with the entries in sca-contribution.xml taking priority if there are any
4955 conflicting declarations.

4956 An SCA runtime MUST make the <import/> and <export/> elements found in the META-INF/sca-
4957 contribution.xml and META-INF/sca-contribution-generated.xml files available for the SCA artifact
4958 resolution process. [ASM12026] An SCA runtime MUST reject files that do not conform to the schema
4959 declared in sca-contribution.xsd. [ASM12027] An SCA runtime MUST merge the contents of sca-
4960 contribution-generated.xml into the contents of sca-contribution.xml, with the entries in sca-
4961 contribution.xml taking priority if there are any conflicting declarations. [ASM12028]

4962 The format of the document is:
4963

```
4964 <?xml version="1.0" encoding="ASCII"?>  
4965 <!-- sca-contribution pseudo-schema -->
```

4966
4967
4968
4969
4970
4971
4972

```
<contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912>
  <deployable composite="xs:QName"/>*
  <import namespace="xs:String" location="xs:AnyURI"?/>*
  <export namespace="xs:String"/>*
</contribution>
```

4973 *Snippet 12-1: contribution Pseudo-Schema*

4974

4975 **deployable element:** Identifies a composite which is a composite within the contribution that is a
4976 composite intended for potential inclusion into the virtual domain-level composite. Other composites in
4977 the contribution are not intended for inclusion but only for use by other composites. New composites can
4978 be created for a contribution after it is installed, by using the [add Deployment Composite](#) capability and
4979 the add To Domain Level Composite capability. An SCA runtime MAY deploy the composites in
4980 <deployable/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-
4981 generated.xml files. [ASM12029]

4982 Attributes of the deployable element:

- 4983 • **composite (1..1)** – The QName of a composite within the contribution.

4984 **Export element:** A declaration that artifacts belonging to a particular namespace are exported and are
4985 available for use within other contributions. An export declaration in a contribution specifies a
4986 namespace, all of whose definitions are considered to be exported. By default, definitions are not
4987 exported.

4988 The SCA artifact export is useful for SCA Domains containing heterogeneous mixtures of contribution
4989 packagings and technologies, where artifact-related or packaging-related mechanisms are unlikely to
4990 work across different kinds of contribution.

4991 Attributes of the export element:

- 4992 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the @namespace attribute
4993 of the export element MUST be the namespace URI for the exported definitions. [ASM12030] For
4994 XML technologies that define multiple *symbol spaces* that can be used within one namespace (e.g.
4995 WSDL portTypes are a different symbol space from WSDL bindings), all definitions from all symbol
4996 spaces are exported.

4997 Technologies that use naming schemes other than QNames use a different export element from the
4998 same substitution group as the the SCA <export> element. The element used identifies the
4999 technology, and can use any value for the namespace that is appropriate for that technology. For
5000 example, <export.java> can be used to export java definitions, in which case the namespace is a fully
5001 qualified package name.

5002 **Import element:** Import declarations specify namespaces of definitions that are needed by the definitions
5003 and implementations within the contribution, but which are not present in the contribution. It is expected
5004 that in most cases import declarations will be generated based on introspection of the contents of the
5005 contribution. In this case, the import declarations would be found in the META-INF/ sca-contribution-
5006 generated.xml document.

5007 Attributes of the import element:

- 5008 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace is the
5009 namespace URI for the imported definitions. For XML technologies that define multiple *symbol*
5010 *spaces* that can be used within one namespace (e.g. WSDL portTypes are a different symbol space
5011 from WSDL bindings), all definitions from all symbol spaces are imported.

5012 Technologies that use naming schemes other than QNames use a different import element from the
5013 same substitution group as the the SCA <import> element. The element used identifies the
5014 technology, and can use any value for the namespace that is appropriate for that technology. For
5015 example, <import.java> can be used to import java definitions, in which case the namespace is a fully
5016 qualified package name.

- **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific requirements for the form of this URI, nor the means by which it is resolved. It can point to another contribution (through its URI) or it can point to some location entirely outside the SCA Domain. It is expected that SCA runtimes can define implementation specific ways of resolving location information for artifact resolution between contributions. These mechanisms will however usually be limited to sets of contributions of one runtime technology and one hosting environment.

In order to accommodate imports of artifacts between contributions of disparate runtime technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location specification.

SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact location specifications.

The order in which the import statements are specified can play a role in this mechanism. Since definitions of one namespace can be distributed across several artifacts, multiple import declarations can be made for one namespace.

The location value is only a default, and dependent contributions listed in the call to installContribution can override the value if there is a conflict. However, the specific mechanism for resolving conflicts between contributions that define conflicting definitions is implementation specific.

If the value of the @location attribute is an SCA contribution URI, then the contribution packaging can become dependent on the deployment environment. In order to avoid such a dependency, it is recommended that dependent contributions are specified only when deploying or updating contributions as specified in the section 'Operations for Contributions' below.

12.2.3 Contribution Packaging using ZIP

SCA allows many different packaging formats that SCA runtimes can support, but SCA requires that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This format allows that metadata specified by the section 'SCA Contribution Metadata Document' be present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-contribution.xml" file and there can also be a "META-INF/sca-contribution-generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

A definition of the ZIP file format is published by PKWARE in an [Application Note on the .ZIP file format \[ZIP-FORMAT\]](#).

12.3 States of Artifacts in the Domain

Artifacts in the SCA domain are in one of 3 states:

5050

1. Installed
2. Deployed
3. Running

5054

Installed artifacts are artifacts that are part of a Contribution that is installed into the Domain. Installed artifacts are available for use by other artifacts that are deployed, See "install Contribution" and "remove Contribution" to understand how artifacts are installed and uninstalled.

Deployed artifacts are artifacts that are available to the SCA runtime to be run.. Artifacts are deployed either through explicit deployment actions or through the presence of <deployable/> elements in sca-contribution.xml files within a Contribution. If an artifact is deployed which has dependencies on other artifacts, then those dependent artifacts are also deployed.

When the SCA runtime has one or more deployable artifacts, the runtime attempts to put those artifacts and any artifacts they depend on into the Running state. This can fail due to errors in one or more of the artifacts or the process can be delayed until all dependencies are available.

5065 Checking for errors in artifacts MUST NOT be done for artifacts in the Installed state (ie where the
5066 artifacts are simply part of installed contributions) [ASM12032]
5067 Errors in artifacts MUST be detected either during the Deployment of the artifacts, or during the process
5068 of putting the artifacts into the Running state, [ASM12033]

5069 12.4 Installed Contribution

5070 As noted in the section above, the contents of a contribution do not need to be modified in order to install
5071 and use it within a Domain. An *installed contribution* is a contribution with all of the associated
5072 information necessary in order to execute *deployable composites* within the contribution.

5073 An installed contribution is made up of the following things:

- 5074 • Contribution Packaging – the contribution that will be used as the starting point for resolving all
5075 references
- 5076 • Contribution base URI
- 5077 • Dependent contributions: a set of snapshots of other contributions that are used to resolve the import
5078 statements from the root composite and from other dependent contributions
 - 5079 – Dependent contributions might or might not be shared with other installed contributions.
 - 5080 – When the snapshot of any contribution is taken is implementation defined, ranging from the time
5081 the contribution is installed to the time of execution
- 5082 • Deployment-time composites.
5083 These are composites that are added into an installed contribution after it has been deployed. This
5084 makes it possible to provide final configuration and access to implementations within a contribution
5085 without having to modify the contribution. These do not have to be provided as composites that
5086 already exist within the contribution can also be used for deployment.

5087 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML, fully
5088 qualified class names in Java).

5089 If multiple dependent contributions have exported definitions with conflicting qualified names, the
5090 algorithm used to determine the qualified name to use is implementation dependent. Implementations of
5091 SCA MAY also raise an error if there are conflicting names exported from multiple contributions.
5092 [ASM12007]

5093 12.4.1 Installed Artifact URIs

5094 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are
5095 constructed by starting with the base URI of the contribution and adding the relative URI of each artifact
5096 (recalling that SCA demands that any packaging format be able to offer up its artifacts in a single
5097 hierarchy).

5098 12.5 Operations for Contributions

5099 SCA Runtimes provide the following conceptual functionality associated with contributions to the Domain
5100 (meaning the function might not be represented as addressable services and also meaning that
5101 equivalent functionality might be provided in other ways). It is strongly encouraged that an SCA runtime
5102 provides the contribution operation functions (install Contribution, update Contribution, add Deployment
5103 Composite, update Deployment Composite, remove Contribution); how these are provided is
5104 implementation specific.

5105 12.5.1 install Contribution & update Contribution

5106 Creates or updates an installed contribution with a supplied root contribution, and installed at a supplied
5107 base URI. A supplied dependent contribution list (<export/> elements) specifies the contributions that are
5108 used to resolve the dependencies of the root contribution and other dependent contributions. These
5109 override any dependent contributions explicitly listed via the @location attribute in the import statements
5110 of the contribution.

5111 SCA follows the simplifying assumption that the use of a contribution for resolving anything also means
5112 that all other exported artifacts can be used from that contribution. Because of this, the dependent
5113 contribution list is just a list of installed contribution URIs. There is no need to specify what is being used
5114 from each one.

5115 Each dependent contribution is also an installed contribution, with its own dependent contributions. By
5116 default these dependent contributions of the dependent contributions (which we will call *indirect*
5117 *dependent contributions*) are included as dependent contributions of the installed contribution. However,
5118 if a contribution in the dependent contribution list exports any conflicting definitions with an indirect
5119 dependent contribution, then the indirect dependent contribution is not included (i.e. the explicit list
5120 overrides the default inclusion of indirect dependent contributions). Also, **if there is ever a conflict**
5121 **between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in**
5122 **the dependent contribution list. [ASM12009]**

5123 Note that in many cases, the dependent contribution list can be generated. In particular, if the creator of
5124 a Domain is careful to avoid creating duplicate definitions for the same qualified name, then it is easy for
5125 this list to be generated by tooling.

5126 12.5.2 add Deployment Composite & update Deployment Composite

5127 Adds or updates a deployment composite using a supplied composite ("composite by value" – a data
5128 structure, not an existing resource in the Domain) to the contribution identified by a supplied contribution
5129 URI. The added or updated deployment composite is given a relative URI that matches the @name
5130 attribute of the composite, with a ".composite" suffix. Since all composites run within the context of a
5131 installed contribution (any component implementations or other definitions are resolved within that
5132 contribution), this functionality makes it possible for the deployer to create a composite with final
5133 configuration and wiring decisions and add it to an installed contribution without having to modify the
5134 contents of the root contribution.

5135 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts). It is
5136 then possible for those to be given component names by a (possibly generated) composite that is added
5137 into the installed contribution, without having to modify the packaging.

5138 12.5.3 remove Contribution

5139 Removes the deployed contribution identified by a supplied contribution URI.

5140 12.6 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

5141 For certain types of artifact, there are existing and commonly used mechanisms for referencing a specific
5142 concrete location where the artifact can be resolved.

5143 Examples of these mechanisms include:

- 5144 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the place
5145 holding the WSDL itself.
- 5146 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a URI where
5147 the XSD is found.

5148 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does not have
5149 to be dereferenced.

5150 SCA permits the use of these mechanisms **Where present, non-SCA artifact resolution mechanisms**
5151 **MUST be used by the SCA runtime in precedence to the SCA mechanisms. [ASM12010]** However, use
5152 of these mechanisms is discouraged because tying assemblies to addresses in this way makes the
5153 assemblies less flexible and prone to errors when changes are made to the overall SCA Domain.

5154 **Note:** **If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the**
5155 **resource indicated when using the mechanism (e.g. the URI is incorrect or invalid, say) the SCA runtime**
5156 **MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an alternative.**
5157 **[ASM12011]**

5158 12.7 Domain-Level Composite

5159 The domain-level composite is a virtual composite, in that it is not defined by a composite definition
5160 document. Rather, it is built up and modified through operations on the Domain. However, in other
5161 respects it is very much like a composite, since it contains components, wires, services and references.

5162 **The value of @autowire for the logical Domain composite MUST be autowire="false". [ASM12012]**

5163 For components at the Domain level, with references for which @autowire="true" applies, the behaviour
5164 of the SCA runtime for a given Domain is implementation specific although it is expected that ONE of the
5165 3 behaviours below is followed:

- 5166 1) The SCA runtime disallows deployment of any components with autowire references. In this case,
5167 the SCA runtime can raise an exception at the point where the component is deployed.
- 5168 2) The SCA runtime evaluates the target(s) for the reference at the time that the component is
5169 deployed and does not update those targets when later deployment actions occur.
- 5170 3) The SCA runtime re-evaluates the target(s) for the reference dynamically as later deployment
5171 actions occur resulting in updated reference targets which match the new Domain configuration.
5172 How the reconfiguration of the reference takes place is described by the relevant client and
5173 implementation specifications.

5174 The abstract domain-level functionality for modifying the domain-level composite is as follows, although a
5175 runtime can supply equivalent functionality in a different form:

5176 12.7.1 add To Domain-Level Composite

5177 This functionality adds the composite identified by a supplied URI to the Domain Level Composite. The
5178 supplied composite URI refers to a composite within an installed contribution. The composite's installed
5179 contribution determines how the composite's artifacts are resolved (directly and indirectly). The supplied
5180 composite is added to the domain composite with semantics that correspond to the domain-level
5181 composite having an <include> statement that references the supplied composite. All of the composites
5182 components become top-level components and the component services become externally visible
5183 services (eg. they would be present in a WSDL description of the Domain). The meaning of any promoted
5184 services and references in the supplied composite is not defined; since there is no composite scope
5185 outside the domain composite, the usual idea of promotion has no utility.

5186 12.7.2 remove From Domain-Level Composite

5187 Removes from the Domain Level composite the elements corresponding to the composite identified by a
5188 supplied composite URI. This means that the removal of the components, wires, services and references
5189 originally added to the domain level composite by the identified composite.

5190 12.7.3 get Domain-Level Composite

5191 Returns a <composite> definition that has an <include> line for each composite that had been added to
5192 the domain level composite. It is important to note that, in dereferencing the included composites, any
5193 referenced artifacts are resolved in terms of that installed composite.

5194 12.7.4 get QName Definition

5195 In order to make sense of the domain-level composite (as returned by get Domain-Level Composite), it
5196 needs to be possible to get the definitions for named artifacts in the included composites. This
5197 functionality takes the supplied URI of an installed contribution (which provides the context), a supplied
5198 qualified name of a definition to look up, and a supplied symbol space (as a QName, e.g. wsdl:portType).
5199 The result is a single definition, in whatever form is appropriate for that definition type.

5200 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities need to
5201 exist in some form, but not necessarily as a service operation with exactly this signature.

5202 12.8 Dynamic Behaviour of Wires in the SCA Domain

5203 For components with references which are at the Domain level, there is the potential for dynamic
5204 behaviour when the wires for a component reference change (this can only apply to component
5205 references at the Domain level and not to components within composites used as implementations):

5206 The configuration of the wires for a component reference of a component at the Domain level can change
5207 by means of deployment actions:

- 5208 1. <wire/> elements can be added, removed or replaced by deployment actions
- 5209 2. Components can be updated by deployment actions (i.e. this can change the
5210 component reference configuration)
- 5211 3. Components which are the targets of reference wires can be updated or removed
- 5212 4. Components can be added that are potential targets for references which are
5213 marked with @autowire=true

5214 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose
5215 references are affected by those deployment actions can have their references updated by the SCA
5216 runtime dynamically without the need to stop and start those components. How this is achieved is
5217 implementation specific.

5218 Where components are updated by deployment actions (their configuration is changed in some way,
5219 which includes changing the wires of component references), the SCA implementation needs to ensure
5220 that the updates apply to all new instances of those components once the update is complete. An SCA
5221 runtime can choose to maintain existing instances with the old configuration of components updated by
5222 deployment actions, although an implementation of an SCA runtime can choose to stop and discard
5223 existing instances of those components.

5224 Where a component that is the target of a wire is removed, without the wire being changed, then future
5225 invocations of the reference that use that wire can fail with a fault indicating that the service is
5226 unavailable. If the wire is the result of the autowire process, the SCA runtime can attempt to update the
5227 wire if there exists an alternative target component that satisfies the autowire process.

5228 Where a component that is the target of a wire is updated, an SCA runtime can direct future invocations
5229 of that reference to the updated component.

5230 Where a component is added to the Domain that is a potential target for a domain level component
5231 reference where that reference is marked as @autowire=true, the SCA runtime can:

- 5232 • either update the references for the source component once the new component is running.
- 5233 • or alternatively, defer the updating of the references of the source component until the source
5234 component is stopped and restarted.

5235

5236 12.9 Dynamic Behaviour of Component Property Values

5237 For a domain level component with a Property whose value is obtained from a Domain-level Property
5238 through the use of the @source attribute, if the domain level property is updated by means of deployment
5239 actions, the SCA runtime MUST

- 5240 • either update the property value of the domain level component once the update of the domain
5241 property is complete
- 5242 • or defer the updating of the component property value until the component is stopped and
5243 restarted

5244 **[ASM12034]**

5245 13 SCA Runtime Considerations

5246 This section describes aspects of an SCA Runtime that are defined by this specification.

5247 13.1 Error Handling

5248 The SCA Assembly specification identifies situations where the configuration of the SCA Domain and its
5249 contents are in error. When one of these situations occurs, the specification requires that the SCA
5250 Runtime that is interacting with the SCA Domain and the artifacts it contains recognises that there is an
5251 error, raise the error in a suitable manner and also refuse to run components and services that are in
5252 error.

5253 The SCA Assembly specification is not prescriptive about the functionality of an SCA Runtime and the
5254 specification recognizes that there can be a range of design points for an SCA runtime. As a result, the
5255 SCA Assembly specification describes a range of error handling approaches which can be adopted by an
5256 SCA runtime.

5257 An SCA Runtime **MUST** raise an error for every situation where the configuration of the SCA Domain or
5258 its contents are in error. The error is either raised at deployment time or at runtime, depending on the
5259 nature of the error and the design of the SCA Runtime. [ASM14005]

5260 13.1.1 Errors which can be Detected at Deployment Time

5261 Some error situations can be detected at the point that artifacts are deployed to the Domain. An example
5262 is a composite document that is invalid in a way that can be detected by static analysis, such as
5263 containing a component with two services with the same @name attribute.

5264 An SCA runtime is expected to detect errors at deployment time where those errors can be found through
5265 static analysis. An SCA runtime has to prevent deployment of contributions that are in error, and raise an
5266 error to the process performing the deployment (e.g. write a message to an interactive console or write a
5267 message to a log file). The exact timing of checking contributions for errors is implementation specific.

5268 The SCA Assembly specification recognizes that there are reasons why a particular SCA runtime finds it
5269 desirable to deploy contributions that contain errors (e.g. to assist in the process of development and
5270 debugging) - and as a result also supports an error handling strategy that is based on detecting problems
5271 at runtime. However, it is wise to consider reporting problems at an early stage in the deployment
5272 process.

5273 13.1.2 Errors which are Detected at Runtime

5274 An SCA runtime can detect problems at runtime. These errors can include some which can be found
5275 from static analysis (e.g. the inability to wire a reference because the target service does not exist in the
5276 Domain) and others that can only be discovered dynamically (e.g. the inability to invoke some remote
5277 Web service because the remote endpoint is unavailable).

5278 Where errors can be detected through static analysis, the principle is that components that are known to
5279 be in error are not run. So, for example, if there is a component with a required reference (multiplicity 1..1
5280 or 1..n) which is not wired, best practice is that the component is not run. If an attempt is made to invoke
5281 a service operation of that component, a "ServiceUnavailable" fault is raised to the invoker. It is also
5282 regarded as best practice that errors of this kind are also raised through appropriate management
5283 interfaces, for example to the deployer or to the operator of the system.

5284

14 Conformance

5285 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
5286 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
5287 this
5288 document.

5289 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd, sca-interface-
5290 wsdl.xsd, sca-implementation-composite.xsd and sca-binding-sca.xsd schema. [ASM13001]

5291 An SCA runtime MUST reject a contribution file that does not conform to the sca-contribution.xsd schema.
5292 [ASM13002]

5293 An SCA runtime MUST reject a definitions file that does not conform to the sca-definitions.xsd schema.
5294 [ASM13003]

5295 There are two categories of artifacts that this specification defines conformance for: SCA Documents and
5296 SCA Runtimes.

14.1 SCA Documents

5297 For a document to be a valid SCA Document, it MUST comply with one of the SCA document types
5298 below:
5299

5300 SCA Composite Document:

5301 An SCA Composite Document is a file that MUST have an SCA <composite/> element as its root
5302 element and MUST conform to the sca-core-1.2.xsd schema and MUST comply with the
5303 additional constraints on the document contents as defined in Appendix C.

5304 SCA ComponentType Document:

5305 An SCA ComponentType Document is a file that MUST have an SCA <componentType/>
5306 element as its root element and MUST conform to the sca-core-1.2.xsd schema and MUST
5307 comply with the additional constraints on the document contents as defined in
5308 Appendix C.

5309 SCA Definitions Document:

5310 An SCA Definitions Document is a file that MUST have an SCA <definitions/> element as its root
5311 and MUST conform to the sca-definition-1.2.xsd schema and MUST comply with the additional
5312 constraints on the document contents as defined in Appendix C.

5313 SCA Contribution Document:

5314 An SCA Contribution Document is a file that MUST have an SCA <contribution/> element as its
5315 root element and MUST conform to the sca-contribution-1.2.xsd schema and MUST comply with
5316 the additional constraints on the document contents as defined in Appendix C.

5317 SCA Interoperable Packaging Document:

5318 A ZIP file containing SCA Documents and other related artifacts. The ZIP file SHOULD contain a
5319 top-level "META-INF" directory, and SHOULD contain a "META-INF/sca-contribution.xml" file, and
5320 MAY contain a "META-INF/sca-contribution-generated.xml" file.

5321

5322

5323 14.2 SCA Runtime

5324 An implementation that claims to conform to the requirements of an SCA Runtime defined in this
5325 specification MUST meet the following conditions:

- 5326 1. The implementation MUST comply with all mandatory statements listed in table
5327 [Mandatory Items](#) in Appendix C: Conformance Items, related to an SCA Runtime.
- 5328 2. The implementation MUST conform to the [SCA Policy Framework v 1.1 Specification \[SCA-
5329 POLICY\]](#).
- 5330 3. The implementation MUST support at least one implementation type standardized by the
5331 OpenCSA Member Section or at least one implementation type that complies with the
5332 following rules:
- 5333
- 5334 a. The implementation type is defined in compliance with the SCA Assembly Extension Model
5335 (Section 9 of the SCA Assembly Specification).
- 5336
- 5337 b. A document describing the mapping of the constructs defined in the SCA Assembly
5338 specification with those of the implementation type exists and is made available to its
5339 prospective user community. Such a document describes how SCA components can be
5340 developed using the implementation type, how these components can be configured and
5341 assembled together (as instances of Components in SCA compositions). The form and
5342 content of such a document are described in the specification "[Implementation Type
5343 Documentation Requirements for SCA Assembly Model Version 1.1 Specification](#)" [[SCA-
5344 IMPLTYPDOC](#)]. The contents outlined in this specification template MUST be provided in
5345 order for an SCA runtime to claim compliance with the SCA Assembly Specification on the
5346 basis of providing support for that implementation type. An example of a document that
5347 describes an implementation type is the "[SCA POJO Component Implementation
5348 Specification Version 1.1](#)" [[SCA-Java](#)].
- 5349
- 5350 c. An adapted version of the SCA Assembly Test Suite which uses the implementation type
5351 exists and is made available to its prospective user community. The steps required to adapt
5352 the SCA Assembly Test Suite for a new implementation type are described in the
5353 specification "[Test Suite Adaptation for SCA Assembly Model Version 1.1 Specification](#)"
5354 [[SCA-TSA](#)]. The requirements described in this specification MUST be met in order for an
5355 SCA runtime to claim compliance with the SCA Assembly Specification on the basis of
5356 providing support for that implementation type.
- 5357 4. The implementation MUST support `binding.sca` and MUST support and conform to the [SCA
5358 Web Service Binding Specification v 1.1 \[SCA-WSBINDING\]](#).

5359 **14.2.1 Optional Items**

5360 In addition to mandatory items, Appendix C: Conformance Items lists a number of non-mandatory items
5361 that can be implemented SCA Runtimes. These items are categorized into functionally related classes as
5362 follows:

- 5363 • Development – items to improve the development of SCA contributions, debugging, etc.
- 5364 • Enhancement – items that add functionality and features to the SCA Runtime.
- 5365 • Interoperation – items that improve interoperability of SCA contributions and Runtimes

5366 These classifications are not rigid and some may overlap; items are classified according to their primary
5367 intent.

5368

5369

Appendix A. XML Schemas

5370

A.1 sca.xsd

5371 sca-1.2.xsd is provided for convenience. It contains <include/> elements for each of the schema files that
5372 contribute to the <http://docs.oasis-open.org/ns/opencsa/sca/200912> namespace.

5373

A.2 sca-core.xsd

```
5374 <?xml version="1.0" encoding="UTF-8"?>
5375 <!-- Copyright(C) OASIS(R) 2005,2011. All Rights Reserved.
5376 OASIS trademark, IPR and other policies apply. -->
5377 <schema xmlns="http://www.w3.org/2001/XMLSchema"
5378 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
5379 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
5380 elementFormDefault="qualified">
5381
5382 <include schemaLocation="sca-policy-1.1-cd03.xsd"/>
5383 <import namespace="http://www.w3.org/XML/1998/namespace"
5384 schemaLocation="http://www.w3.org/2001/xml.xsd"/>
5385
5386 <!-- Common extension base for SCA definitions -->
5387 <complexType name="CommonExtensionBase">
5388 <sequence>
5389 <element ref="sca:documentation" minOccurs="0"
5390 maxOccurs="unbounded"/>
5391 </sequence>
5392 <anyAttribute namespace="##other" processContents="lax"/>
5393 </complexType>
5394
5395 <element name="documentation" type="sca:Documentation"/>
5396 <complexType name="Documentation" mixed="true">
5397 <sequence>
5398 <any namespace="##other" processContents="lax" minOccurs="0"
5399 maxOccurs="unbounded"/>
5400 </sequence>
5401 <attribute ref="xml:lang"/>
5402 </complexType>
5403
5404 <!-- Component Type -->
5405 <element name="componentType" type="sca:ComponentType"/>
5406 <complexType name="ComponentType">
5407 <complexContent>
5408 <extension base="sca:CommonExtensionBase">
5409 <sequence>
5410 <element ref="sca:implementation" minOccurs="0"/>
5411 <choice minOccurs="0" maxOccurs="unbounded">
5412 <element name="service" type="sca:ComponentService"/>
5413 <element name="reference"
5414 type="sca:ComponentTypeReference"/>
5415 <element name="property" type="sca:Property"/>
5416 <element name="consumer" type="sca:ComponentConsumer"/>
5417 <element name="producer" type="sca:ComponentProducer"/>
5418 </choice>
5419 <any namespace="##other" processContents="lax" minOccurs="0"
5420 maxOccurs="unbounded"/>
5421 </sequence>
5422 </extension>
5423 </complexContent>
5424 </complexType>
```

```

5425 </complexType>
5426
5427 <!-- Composite -->
5428 <element name="composite" type="sca:Composite"/>
5429 <complexType name="Composite">
5430 <complexContent>
5431 <extension base="sca:CommonExtensionBase">
5432 <sequence>
5433 <element ref="sca:include" minOccurs="0"
5434 <maxOccurs="unbounded"/>
5435 <choice minOccurs="0" maxOccurs="unbounded">
5436 <element ref="sca:requires"/>
5437 <element ref="sca:policySetAttachment"/>
5438 <element name="service" type="sca:Service"/>
5439 <element name="property" type="sca:Property"/>
5440 <element name="component" type="sca:Component"/>
5441 <element name="reference" type="sca:Reference"/>
5442 <element name="channel" type="sca:Channel"/>
5443 <element name="consumer" type="sca:Consumer"/>
5444 <element name="producer" type="sca:Producer"/>
5445
5446 <element name="wire" type="sca:Wire"/>
5447 </choice>
5448 <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
5449 </sequence>
5450 <attribute name="name" type="NCName" use="required"/>
5451 <attribute name="targetNamespace" type="anyURI" use="required"/>
5452 <attribute name="local" type="boolean" use="optional"
5453 <default="false"/>
5454 <attribute name="autowire" type="boolean" use="optional"
5455 <default="false"/>
5456 <attribute name="requires" type="sca:listOfQNames"
5457 <use="optional"/>
5458 <attribute name="policySets" type="sca:listOfQNames"
5459 <use="optional"/>
5460 </extension>
5461 </complexContent>
5462 </complexType>
5463
5464 <!-- Contract base type for Service, Reference -->
5465 <complexType name="Contract" abstract="true">
5466 <complexContent>
5467 <extension base="sca:CommonExtensionBase">
5468 <sequence>
5469 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
5470 <element ref="sca:binding" minOccurs="0"
5471 <maxOccurs="unbounded" />
5472 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
5473 <element ref="sca:requires" minOccurs="0"
5474 <maxOccurs="unbounded"/>
5475 <element ref="sca:policySetAttachment" minOccurs="0"
5476 <maxOccurs="unbounded"/>
5477 <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
5478 </sequence>
5479 <attribute name="name" type="NCName" use="required" />
5480 <attribute name="requires" type="sca:listOfQNames"
5481 <use="optional" />
5482 <attribute name="policySets" type="sca:listOfQNames"
5483 <use="optional"/>
5484 </extension>
5485 </complexContent>
5486 </complexType>
5487
5488 <!-- Service -->

```

```

5489 <complexType name="Service">
5490 <complexContent>
5491 <extension base="sca:Contract">
5492 <attribute name="promote" type="anyURI" use="required"/>
5493 </extension>
5494 </complexContent>
5495 </complexType>
5496
5497 <!-- Interface -->
5498 <element name="interface" type="sca:Interface" abstract="true"/>
5499 <complexType name="Interface" abstract="true">
5500 <complexContent>
5501 <extension base="sca:CommonExtensionBase">
5502 <choice minOccurs="0" maxOccurs="unbounded">
5503 <element ref="sca:requires"/>
5504 <element ref="sca:policySetAttachment"/>
5505 </choice>
5506 <attribute name="remotable" type="boolean" use="optional"/>
5507 <attribute name="requires" type="sca:listOfQNames"
5508 use="optional"/>
5509 <attribute name="policySets" type="sca:listOfQNames"
5510 use="optional"/>
5511 </extension>
5512 </complexContent>
5513 </complexType>
5514
5515 <!-- Reference -->
5516 <complexType name="Reference">
5517 <complexContent>
5518 <extension base="sca:Contract">
5519 <attribute name="target" type="sca:listOfAnyURIs"
5520 use="optional"/>
5521 <attribute name="wiredByImpl" type="boolean" use="optional"
5522 default="false"/>
5523 <attribute name="multiplicity" type="sca:Multiplicity"
5524 use="required"/>
5525 <attribute name="promote" type="sca:listOfAnyURIs"
5526 use="required"/>
5527 </extension>
5528 </complexContent>
5529 </complexType>
5530
5531 <complexType name="ConsumerContract">
5532 <complexContent>
5533 <extension base="sca:CommonExtensionBase">
5534 <sequence>
5535 <element ref="sca:filters" minOccurs="0" maxOccurs="1" />
5536 <element ref="sca:requires" minOccurs="0"
5537 maxOccurs="unbounded"/>
5538 <element ref="sca:policySetAttachment" minOccurs="0"
5539 maxOccurs="unbounded"/>
5540 <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
5541 </sequence>
5542 <attribute name="name" type="NCName" use="required" />
5543 <attribute name="requires" type="sca:listOfQNames"
5544 use="optional" />
5545 <attribute name="policySets" type="sca:listOfQNames"
5546 use="optional"/>
5547 </extension>
5548 </complexContent>
5549 </complexType>
5550
5551 <complexType name="Consumer">
5552 <complexContent>

```

```

5553     <extension base="sca:ConsumerContract">
5554         <sequence/>
5555         <attribute name="promote" type="sca:listOfAnyURIs"
5556             use="required"/>
5557     </extension>
5558 </complexContent>
5559 </complexType>
5560
5561 <complexType name="ProducerContract">
5562     <complexContent>
5563         <extension base="sca:CommonExtensionBase">
5564             <sequence>
5565                 <element ref="sca:eventType" minOccurs="0" maxOccurs="1" />
5566                 <element ref="sca:requires" minOccurs="0"
5567                     maxOccurs="unbounded"/>
5568                 <element ref="sca:policySetAttachment" minOccurs="0"
5569                     maxOccurs="unbounded"/>
5570                 <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
5571             </sequence>
5572             <attribute name="name" type="NCName" use="required" />
5573             <attribute name="requires" type="sca:listOfQNames"
5574                 use="optional" />
5575             <attribute name="policySets" type="sca:listOfQNames"
5576                 use="optional"/>
5577         </extension>
5578     </complexContent>
5579 </complexType>
5580
5581 <complexType name="Producer">
5582     <complexContent>
5583         <extension base="sca:ProducerContract">
5584             <sequence/>
5585             <attribute name="promote" type="sca:listOfAnyURIs"
5586                 use="required"/>
5587         </extension>
5588     </complexContent>
5589 </complexType>
5590
5591 <complexType name="Channel">
5592     <complexContent>
5593         <extension base="sca:CommonExtensionBase">
5594             <sequence>
5595                 <element ref="sca:filters" minOccurs="0" maxOccurs="1" />
5596                 <element ref="sca:binding" minOccurs="0"
5597                     maxOccurs="1" />
5598                 <element ref="sca:requires" minOccurs="0"
5599                     maxOccurs="unbounded"/>
5600                 <element ref="sca:policySetAttachment" minOccurs="0"
5601                     maxOccurs="unbounded"/>
5602                 <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
5603             </sequence>
5604             <attribute name="name" type="NCName" use="required" />
5605             <attribute name="requires" type="sca:listOfQNames"
5606                 use="optional" />
5607             <attribute name="policySets" type="sca:listOfQNames"
5608                 use="optional"/>
5609         </extension>
5610     </complexContent>
5611 </complexType>
5612
5613 <!-- Property -->
5614 <complexType name="SCAPropertyBase" mixed="true">
5615     <sequence>
5616         <any namespace="##any" processContents="lax" minOccurs="0"

```

```

5617         maxOccurs="unbounded"/>
5618     <!-- NOT an extension point; This any exists to accept
5619         the element-based or complex type property
5620         i.e. no element-based extension point under "sca:property" -->
5621 </sequence>
5622 <!-- mixed="true" to handle simple type -->
5623 <attribute name="name" type="NCName" use="required"/>
5624 <attribute name="type" type="QName" use="optional"/>
5625 <attribute name="element" type="QName" use="optional"/>
5626 <attribute name="many" type="boolean" use="optional" default="false"/>
5627 <attribute name="value" type="anySimpleType" use="optional"/>
5628 <anyAttribute namespace="##other" processContents="lax"/>
5629 </complexType>
5630
5631 <complexType name="Property" mixed="true">
5632     <complexContent mixed="true">
5633         <extension base="sca:SCAPropertyBase">
5634             <attribute name="mustSupply" type="boolean" use="optional"
5635                 default="false"/>
5636         </extension>
5637     </complexContent>
5638 </complexType>
5639
5640 <complexType name="PropertyValue" mixed="true">
5641     <complexContent mixed="true">
5642         <extension base="sca:SCAPropertyBase">
5643             <attribute name="source" type="string" use="optional"/>
5644             <attribute name="file" type="anyURI" use="optional"/>
5645         </extension>
5646     </complexContent>
5647 </complexType>
5648
5649 <!-- Binding -->
5650 <element name="binding" type="sca:Binding" abstract="true"/>
5651 <complexType name="Binding" abstract="true">
5652     <complexContent>
5653         <extension base="sca:CommonExtensionBase">
5654             <sequence>
5655                 <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
5656                 <element ref="sca:operationSelector" minOccurs="0"
5657                     maxOccurs="1" />
5658                 <element ref="sca:requires" minOccurs="0"
5659                     maxOccurs="unbounded"/>
5660                 <element ref="sca:policySetAttachment" minOccurs="0"
5661                     maxOccurs="unbounded"/>
5662                 <element ref="filters" minOccurs="0" maxOccurs="1"/>
5663             </sequence>
5664             <attribute name="uri" type="anyURI" use="optional"/>
5665             <attribute name="name" type="NCName" use="optional"/>
5666             <attribute name="requires" type="sca:listOfQNames"
5667                 use="optional"/>
5668             <attribute name="policySets" type="sca:listOfQNames"
5669                 use="optional"/>
5670         </extension>
5671     </complexContent>
5672 </complexType>
5673
5674 <!-- Binding Type -->
5675 <element name="bindingType" type="sca:BindingType"/>
5676 <complexType name="BindingType">
5677     <complexContent>
5678         <extension base="sca:CommonExtensionBase">
5679             <sequence>
5680                 <any namespace="##other" processContents="lax" minOccurs="0"

```

```

5681         maxOccurs="unbounded"/>
5682     </sequence>
5683     <attribute name="type" type="QName" use="required"/>
5684     <attribute name="alwaysProvides" type="sca:listOfQNames"
5685         use="optional"/>
5686     <attribute name="mayProvide" type="sca:listOfQNames"
5687         use="optional"/>
5688     </extension>
5689 </complexContent>
5690 </complexType>
5691
5692 <!-- WireFormat Type -->
5693 <element name="wireFormat" type="sca:WireFormatType" abstract="true"/>
5694 <complexType name="WireFormatType" abstract="true">
5695     <anyAttribute namespace="##other" processContents="lax"/>
5696 </complexType>
5697
5698 <!-- OperationSelector Type -->
5699 <element name="operationSelector" type="sca:OperationSelectorType"
5700     abstract="true"/>
5701 <complexType name="OperationSelectorType" abstract="true">
5702     <anyAttribute namespace="##other" processContents="lax"/>
5703 </complexType>
5704
5705 <!-- Callback -->
5706 <element name="callback" type="sca:Callback"/>
5707 <complexType name="Callback">
5708     <complexContent>
5709         <extension base="sca:CommonExtensionBase">
5710             <choice minOccurs="0" maxOccurs="unbounded">
5711                 <element ref="sca:binding"/>
5712                 <element ref="sca:requires"/>
5713                 <element ref="sca:policySetAttachment"/>
5714                 <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
5715             </choice>
5716             <attribute name="requires" type="sca:listOfQNames"
5717                 use="optional"/>
5718             <attribute name="policySets" type="sca:listOfQNames"
5719                 use="optional"/>
5720         </extension>
5721     </complexContent>
5722 </complexType>
5723
5724 <!-- Component -->
5725 <complexType name="Component">
5726     <complexContent>
5727         <extension base="sca:CommonExtensionBase">
5728             <sequence>
5729                 <element ref="sca:implementation" minOccurs="1"
5730                     maxOccurs="1"/>
5731                 <choice minOccurs="0" maxOccurs="unbounded">
5732                     <element name="service" type="sca:ComponentService"/>
5733                     <element name="reference" type="sca:ComponentReference"/>
5734                     <element name="property" type="sca:PropertyValue"/>
5735                     <element ref="sca:requires"/>
5736                     <element ref="sca:policySetAttachment"/>
5737                     <element name="consumer" type="sca:ComponentConsumer"/>
5738                     <element name="producer" type="sca:ComponentProducer"/>
5739                 </choice>
5740                 <any namespace="##other" processContents="lax" minOccurs="0"
5741                     maxOccurs="unbounded"/>
5742             </sequence>
5743             <attribute name="name" type="NCName" use="required"/>
5744             <attribute name="autowire" type="boolean" use="optional"/>

```



```

5745         <attribute name="requires" type="sca:listOfQNames"
5746             use="optional"/>
5747         <attribute name="policySets" type="sca:listOfQNames"
5748             use="optional"/>
5749     </extension>
5750 </complexContent>
5751 </complexType>
5752
5753 <!-- Component Service -->
5754 <complexType name="ComponentService">
5755     <complexContent>
5756         <extension base="sca:Contract">
5757             </extension>
5758         </complexContent>
5759     </complexType>
5760
5761 <!-- Component Reference -->
5762 <complexType name="ComponentReference">
5763     <complexContent>
5764         <extension base="sca:Contract">
5765             <attribute name="autowire" type="boolean" use="optional"/>
5766             <attribute name="target" type="sca:listOfAnyURIs"
5767                 use="optional"/>
5768             <attribute name="wiredByImpl" type="boolean" use="optional"
5769                 default="false"/>
5770             <attribute name="multiplicity" type="sca:Multiplicity"
5771                 use="optional" default="1..1"/>
5772             <attribute name="nonOverridable" type="boolean" use="optional"
5773                 default="false"/>
5774         </extension>
5775     </complexContent>
5776 </complexType>
5777
5778 <complexType name="ComponentConsumer">
5779     <complexContent>
5780         <extension base="sca:ConsumerContract">
5781             <sequence/>
5782             <attribute name="source" type="sca:listOfAnyURIs"
5783                 use="optional"/>
5784         </extension>
5785     </complexContent>
5786 </complexType>
5787
5788 <complexType name="ComponentProducer">
5789     <complexContent>
5790         <extension base="sca:ProducerContract">
5791             <sequence/>
5792             <attribute name="target" type="sca:listOfAnyURIs"
5793                 use="optional"/>
5794         </extension>
5795     </complexContent>
5796 </complexType>
5797
5798 <!-- Component Type Reference -->
5799 <complexType name="ComponentTypeReference">
5800     <complexContent>
5801         <restriction base="sca:ComponentReference">
5802             <sequence>
5803                 <element ref="sca:documentation" minOccurs="0"
5804                     maxOccurs="unbounded"/>
5805                 <element ref="sca:interface" minOccurs="0"/>
5806                 <element ref="sca:binding" minOccurs="0"
5807                     maxOccurs="unbounded"/>
5808                 <element ref="sca:callback" minOccurs="0"/>

```

```

5809         <element ref="sca:requires" minOccurs="0"
5810             maxOccurs="unbounded"/>
5811         <element ref="sca:policySetAttachment" minOccurs="0"
5812             maxOccurs="unbounded"/>
5813         <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
5814     </sequence>
5815     <attribute name="name" type="NCName" use="required"/>
5816     <attribute name="autowire" type="boolean" use="optional"/>
5817     <attribute name="wiredByImpl" type="boolean" use="optional"
5818         default="false"/>
5819     <attribute name="multiplicity" type="sca:Multiplicity"
5820         use="optional" default="1..1"/>
5821     <attribute name="requires" type="sca:listOfQNames"
5822         use="optional"/>
5823     <attribute name="policySets" type="sca:listOfQNames"
5824         use="optional"/>
5825     <anyAttribute namespace="##other" processContents="lax"/>
5826 </restriction>
5827 </complexContent>
5828 </complexType>
5829
5830
5831 <!-- Implementation -->
5832 <element name="implementation" type="sca:Implementation" abstract="true"/>
5833 <complexType name="Implementation" abstract="true">
5834     <complexContent>
5835         <extension base="sca:CommonExtensionBase">
5836             <choice minOccurs="0" maxOccurs="unbounded">
5837                 <element ref="sca:requires"/>
5838                 <element ref="sca:policySetAttachment"/>
5839             </choice>
5840             <attribute name="requires" type="sca:listOfQNames"
5841                 use="optional"/>
5842             <attribute name="policySets" type="sca:listOfQNames"
5843                 use="optional"/>
5844         </extension>
5845     </complexContent>
5846 </complexType>
5847
5848 <!-- Implementation Type -->
5849 <element name="implementationType" type="sca:ImplementationType"/>
5850 <complexType name="ImplementationType">
5851     <complexContent>
5852         <extension base="sca:CommonExtensionBase">
5853             <sequence>
5854                 <any namespace="##other" processContents="lax" minOccurs="0"
5855                     maxOccurs="unbounded"/>
5856             </sequence>
5857             <attribute name="type" type="QName" use="required"/>
5858             <attribute name="alwaysProvides" type="sca:listOfQNames"
5859                 use="optional"/>
5860             <attribute name="mayProvide" type="sca:listOfQNames"
5861                 use="optional"/>
5862         </extension>
5863     </complexContent>
5864 </complexType>
5865
5866 <!-- Wire -->
5867 <complexType name="Wire">
5868     <complexContent>
5869         <extension base="sca:CommonExtensionBase">
5870             <sequence>
5871                 <any namespace="##other" processContents="lax" minOccurs="0"
5872                     maxOccurs="unbounded"/>

```

```

5873         </sequence>
5874         <attribute name="source" type="anyURI" use="required"/>
5875         <attribute name="target" type="anyURI" use="required"/>
5876         <attribute name="replace" type="boolean" use="optional"
5877             default="false"/>
5878     </extension>
5879 </complexContent>
5880 </complexType>
5881
5882 <!-- Include -->
5883 <element name="include" type="sca:Include"/>
5884 <complexType name="Include">
5885     <complexContent>
5886         <extension base="sca:CommonExtensionBase">
5887             <attribute name="name" type="QName"/>
5888         </extension>
5889     </complexContent>
5890 </complexType>
5891
5892 <!-- Extensions element -->
5893 <element name="extensions">
5894     <complexType>
5895         <sequence>
5896             <any namespace="##other" processContents="lax"
5897                 minOccurs="1" maxOccurs="unbounded"/>
5898         </sequence>
5899     </complexType>
5900 </element>
5901
5902 <!-- Intents within WSDL documents -->
5903 <attribute name="requires" type="sca:listOfQNames"/>
5904
5905 <!-- Global attribute definition for @callback to mark a WSDL port type
5906     as having a callback interface defined in terms of a second port
5907     type. -->
5908 <attribute name="callback" type="anyURI"/>
5909
5910 <!-- Value type definition for property values -->
5911 <element name="value" type="sca:ValueType"/>
5912 <complexType name="ValueType" mixed="true">
5913     <sequence>
5914         <any namespace="##any" processContents="lax" minOccurs="0"
5915             maxOccurs="unbounded"/>
5916     </sequence>
5917     <!-- mixed="true" to handle simple type -->
5918     <anyAttribute namespace="##any" processContents="lax"/>
5919 </complexType>
5920
5921 <!-- Miscellaneous simple type definitions -->
5922 <simpleType name="Multiplicity">
5923     <restriction base="string">
5924         <enumeration value="0..1"/>
5925         <enumeration value="1..1"/>
5926         <enumeration value="0..n"/>
5927         <enumeration value="1..n"/>
5928     </restriction>
5929 </simpleType>
5930
5931 <simpleType name="OverrideOptions">
5932     <restriction base="string">
5933         <enumeration value="no"/>
5934         <enumeration value="may"/>
5935         <enumeration value="must"/>
5936     </restriction>

```

```

5937 </simpleType>
5938
5939 <simpleType name="listOfQNames">
5940 <list itemType="QName"/>
5941 </simpleType>
5942
5943 <simpleType name="listOfAnyURIs">
5944 <list itemType="anyURI"/>
5945 </simpleType>
5946
5947 <simpleType name="CreateResource">
5948 <restriction base="string">
5949 <enumeration value="always" />
5950 <enumeration value="never" />
5951 <enumeration value="ifnotexist" />
5952 </restriction>
5953 </simpleType>
5954
5955 <element name="filters" type="sca:Filter"/>
5956 <complexType name="Filter">
5957 <sequence>
5958 <choice minOccurs="0" maxOccurs="unbounded">
5959 <element ref="sca:eventType" />
5960 <element ref="sca:body.xpath1" />
5961 </choice>
5962 <any namespace="##other" processContents="lax" minOccurs="0"
5963 maxOccurs="unbounded"/>
5964 </sequence>
5965 <anyAttribute namespace="##other" processContents="lax"/>
5966 </complexType>
5967
5968 <element name="eventType" abstract="true"/>
5969
5970 <element name="eventType.sca" type="sca:EventType.sca"
5971 substitutionGroup="eventType"/>
5972
5973 <complexType name="EventType.sca">
5974 <sequence>
5975 <any namespace="##other" processContents="lax" minOccurs="0"
5976 maxOccurs="unbounded"/>
5977 </sequence>
5978 <attribute name="qnames" type="sca:listOfQNames" />
5979 <attribute name="namespaces" type="sca:listOfAnyURIs" />
5980 <anyAttribute namespace="##other" processContents="lax" />
5981 </complexType>
5982
5983 <element name="body.xpath1" type="string" />
5984
5985 </schema>

```

5986 A.3 sca-binding-sca.xsd

```

5987 <?xml version="1.0" encoding="UTF-8"?>
5988 <!-- Copyright (C) OASIS (R) 2005,2011. All Rights Reserved.
5989 OASIS trademark, IPR and other policies apply. -->
5990 <schema xmlns="http://www.w3.org/2001/XMLSchema"
5991 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
5992 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
5993 elementFormDefault="qualified">
5994
5995 <include schemaLocation="sca-core-1.2-csd01.xsd"/>
5996
5997 <!-- SCA Binding -->

```

```

5998     <element name="binding.sca" type="sca:SCABinding"
5999           substitutionGroup="sca:binding"/>
6000   <complexType name="SCABinding">
6001     <complexContent>
6002       <extension base="sca:Binding"/>
6003     </complexContent>
6004   </complexType>
6005 </schema>
6006

```

6007 **A.4 sca-interface-java.xsd**

6008 Is described in the [SCA Java Common Annotations and APIs specification \[SCA-Common-Java\]](#)

6009 **A.5 sca-interface-wsdl.xsd**

```

6010 <?xml version="1.0" encoding="UTF-8"?>
6011 <!-- Copyright (C) OASIS (R) 2005,2011. All Rights Reserved.
6012       OASIS trademark, IPR and other policies apply. -->
6013 <schema xmlns="http://www.w3.org/2001/XMLSchema"
6014         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6015         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6016         elementFormDefault="qualified">
6017
6018   <include schemaLocation="sca-core-1.2-csd01.xsd"/>
6019
6020   <!-- WSDL Interface -->
6021   <element name="interface.wsdl" type="sca:WSDLPortType"
6022           substitutionGroup="sca:interface"/>
6023   <complexType name="WSDLPortType">
6024     <complexContent>
6025       <extension base="sca:Interface">
6026         <sequence>
6027           <any namespace="##other" processContents="lax" minOccurs="0"
6028               maxOccurs="unbounded"/>
6029         </sequence>
6030         <attribute name="interface" type="anyURI" use="required"/>
6031         <attribute name="callbackInterface" type="anyURI"
6032                   use="optional"/>
6033       </extension>
6034     </complexContent>
6035   </complexType>
6036 </schema>
6037

```

6038 **A.6 sca-implementation-java.xsd**

6039 Is described in the [Java Component Implementation specification \[SCA-Java\]](#)

6040 **A.7 sca-implementation-composite.xsd**

```

6041 <?xml version="1.0" encoding="UTF-8"?>
6042 <!-- Copyright (C) OASIS (R) 2005,2011. All Rights Reserved.
6043       OASIS trademark, IPR and other policies apply. -->
6044 <schema xmlns="http://www.w3.org/2001/XMLSchema"
6045         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6046         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6047         elementFormDefault="qualified">
6048
6049   <include schemaLocation="sca-core-1.2-csd01.xsd"/>
6050

```

```

6051 <!-- Composite Implementation -->
6052 <element name="implementation.composite" type="sca:SCAImplementation"
6053 substitutionGroup="sca:implementation"/>
6054 <complexType name="SCAImplementation">
6055 <complexContent>
6056 <extension base="sca:Implementation">
6057 <sequence>
6058 <any namespace="##other" processContents="lax" minOccurs="0"
6059 maxOccurs="unbounded"/>
6060 </sequence>
6061 <attribute name="name" type="QName" use="required"/>
6062 </extension>
6063 </complexContent>
6064 </complexType>
6065
6066 </schema>

```

6067 **A.8 sca-binding-webservice.xsd**

6068 Is described in [the SCA Web Services Binding specification \[SCA-WSBINDING\]](#)

6069 **A.9 sca-binding-jms.xsd**

6070 Is described in [the SCA JMS Binding specification \[SCA-JMSBINDING\]](#)

6071 **A.10 sca-policy.xsd**

6072 Is described in the [Policy Framework specification \[SCA-POLICY\]](#)

6073 **A.11 sca-contribution.xsd**

```

6074 <?xml version="1.0" encoding="UTF-8"?>
6075 <!-- Copyright (C) OASIS (R) 2005,2011. All Rights Reserved.
6076 OASIS trademark, IPR and other policies apply. -->
6077 <schema xmlns="http://www.w3.org/2001/XMLSchema"
6078 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6079 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6080 elementFormDefault="qualified">
6081
6082 <include schemaLocation="sca-core-1.2-csd01.xsd"/>
6083
6084 <!-- Contribution -->
6085 <element name="contribution" type="sca:ContributionType"/>
6086 <complexType name="ContributionType">
6087 <complexContent>
6088 <extension base="sca:CommonExtensionBase">
6089 <sequence>
6090 <element name="deployable" type="sca:DeployableType"
6091 minOccurs="0" maxOccurs="unbounded"/>
6092 <element ref="sca:importBase" minOccurs="0"
6093 maxOccurs="unbounded"/>
6094 <element ref="sca:exportBase" minOccurs="0"
6095 maxOccurs="unbounded"/>
6096 <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
6097 </sequence>
6098 </extension>
6099 </complexContent>
6100 </complexType>
6101
6102 <!-- Deployable -->
6103 <complexType name="DeployableType">
6104 <complexContent>

```

```

6105     <extension base="sca:CommonExtensionBase">
6106         <sequence>
6107             <any namespace="##other" processContents="lax" minOccurs="0"
6108                 maxOccurs="unbounded"/>
6109         </sequence>
6110         <attribute name="composite" type="QName" use="required"/>
6111     </extension>
6112 </complexContent>
6113 </complexType>
6114
6115 <!-- Import -->
6116 <element name="importBase" type="sca:Import" abstract="true" />
6117 <complexType name="Import" abstract="true">
6118     <complexContent>
6119         <extension base="sca:CommonExtensionBase">
6120             <sequence>
6121                 <any namespace="##other" processContents="lax" minOccurs="0"
6122                     maxOccurs="unbounded"/>
6123             </sequence>
6124         </extension>
6125     </complexContent>
6126 </complexType>
6127
6128 <element name="import" type="sca:ImportType"
6129         substitutionGroup="sca:importBase"/>
6130 <complexType name="ImportType">
6131     <complexContent>
6132         <extension base="sca:Import">
6133             <attribute name="namespace" type="string" use="required"/>
6134             <attribute name="location" type="anyURI" use="optional"/>
6135         </extension>
6136     </complexContent>
6137 </complexType>
6138
6139 <!-- Export -->
6140 <element name="exportBase" type="sca:Export" abstract="true" />
6141 <complexType name="Export" abstract="true">
6142     <complexContent>
6143         <extension base="sca:CommonExtensionBase">
6144             <sequence>
6145                 <any namespace="##other" processContents="lax" minOccurs="0"
6146                     maxOccurs="unbounded"/>
6147             </sequence>
6148         </extension>
6149     </complexContent>
6150 </complexType>
6151
6152 <element name="export" type="sca:ExportType"
6153         substitutionGroup="sca:exportBase"/>
6154 <complexType name="ExportType">
6155     <complexContent>
6156         <extension base="sca:Export">
6157             <attribute name="namespace" type="string" use="required"/>
6158         </extension>
6159     </complexContent>
6160 </complexType>
6161
6162 </schema>

```

6163 A.12 sca-definitions.xsd

```

6164 <?xml version="1.0" encoding="UTF-8"?>
6165 <!-- Copyright (C) OASIS (R) 2005,2011. All Rights Reserved.

```

```

6166     OASIS trademark, IPR and other policies apply. -->
6167 <schema xmlns="http://www.w3.org/2001/XMLSchema"
6168     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6169     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
6170     elementFormDefault="qualified">
6171
6172     <include schemaLocation="sca-core-1.2-csd01.xsd"/>
6173     <include schemaLocation="sca-policy-1.1-cd02.xsd"/>
6174
6175     <!-- Definitions -->
6176     <element name="definitions" type="sca:tDefinitions"/>
6177     <complexType name="tDefinitions">
6178         <complexContent>
6179             <extension base="sca:CommonExtensionBase">
6180                 <choice minOccurs="0" maxOccurs="unbounded">
6181                     <element ref="sca:intent"/>
6182                     <element ref="sca:policySet"/>
6183                     <element ref="sca:bindingType"/>
6184                     <element ref="sca:implementationType"/>
6185                     <element ref="sca:externalAttachment"/>
6186                     <any namespace="##other" processContents="lax"
6187                         minOccurs="0" maxOccurs="unbounded"/>
6188                 </choice>
6189                 <attribute name="targetNamespace" type="anyURI" use="required"/>
6190             </extension>
6191         </complexContent>
6192     </complexType>
6193
6194 </schema>

```

6195 Appendix B. SCA Concepts

6196 B.1 Binding

6197 **Bindings** are used by services and references. References use bindings to describe the access
6198 mechanism used to call the service to which they are wired. Services use bindings to describe the
6199 access mechanism(s) that clients use to call the service.

6200 SCA supports multiple different types of bindings. Examples include **SCA service, Web service,**
6201 **stateless session EJB, database stored procedure, EIS service.** SCA provides an extensibility
6202 mechanism by which an SCA runtime can add support for additional binding types.

6203 B.2 Component

6204 **SCA components** are configured instances of **SCA implementations**, which provide and consume
6205 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines
6206 an **extensibility mechanism** that allows you to introduce new implementation types. The current
6207 specification does not mandate the implementation technologies to be supported by an SCA runtime,
6208 vendors can choose to support the ones that are important for them. A single SCA implementation can be
6209 used by multiple Components, each with a different configuration.

6210 The Component has a reference to an implementation of which it is an instance, a set of property values,
6211 and a set of service reference values. Property values define the values of the properties of the
6212 component as defined by the component's implementation. Reference values define the services that
6213 resolve the references of the component as defined by its implementation. These values can either be a
6214 particular service of a particular component, or a reference of the containing composite.

6215 B.3 Service

6216 **SCA services** are used to declare the externally accessible services of an **implementation**. For a
6217 composite, a service is typically provided by a service of a component within the composite, or by a
6218 reference defined by the composite. The latter case allows the republication of a service with a new
6219 address and/or new bindings. The service can be thought of as a point at which messages from external
6220 clients enter a composite or implementation.

6221 A service represents an addressable set of operations of an implementation that are designed to be
6222 exposed for use by other implementations or exposed publicly for use elsewhere (e.g. public Web
6223 services for use by other organizations). The operations provided by a service are specified by an
6224 Interface, as are the operations needed by the service client (if there is one). An implementation can
6225 contain multiple services, when it is possible to address the services of the implementation separately.

6226 A service can be provided **as SCA remote services, as Web services, as stateless session EJB's, as**
6227 **EIS services, and so on.** Services use **bindings** to describe the way in which they are published. SCA
6228 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new
6229 types of services.

6230 B.3.1 Remotable Service

6231 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA
6232 architecture. For example, SCA services of SCA implementations can define implementations of industry-
6233 standard web services. Remotable services use pass-by-value semantics for parameters and returned
6234 results.

6235 Interfaces can be identified as remotable through the <interface /> XML, but are typically specified as
6236 remotable using a component implementation technology specific mechanism, such as Java annotations.
6237 See the relevant SCA Implementation Specification for more information. As an example, to define a
6238 Remotable Service, a Component implemented in Java would have a Java Interface with the
6239 @Remotable annotation

6240 B.3.2 Local Service

6241 Local services are services that are designed to be only used “locally” by other implementations that are
6242 deployed concurrently in a tightly-coupled architecture within the same operating system process.

6243 Local services can rely on by-reference calling conventions, or can assume a very fine-grained interaction
6244 style that is incompatible with remote distribution. They can also use technology-specific data-types.

6245 How a Service is identified as local is dependant on the Component implementation technology used.

6246 See the relevant SCA Implementation Specification for more information. As an example, to define a
6247 Local Service, a Component implemented in Java would define a Java Interface that does not have the
6248 @Remotable annotation.

6249 B.4 Reference

6250 **SCA references** represent a dependency that an implementation has on a service that is provided by
6251 some other implementation, where the service to be used is specified through configuration. In other
6252 words, a reference is a service that an implementation can call during the execution of its business
6253 function. References are typed by an interface.

6254 For composites, composite references can be accessed by components within the composite like any
6255 service provided by a component within the composite. Composite references can be used as the targets
6256 of wires from component references when configuring Components.

6257 A composite reference can be used to access a service such as: an SCA service provided by another
6258 SCA composite, a Web service, a stateless session EJB, a database stored procedure or an EIS service,
6259 and so on. References use **bindings** to describe the access method used to their services. SCA provides
6260 an **extensibility mechanism** that allows the introduction of new binding types to references.

6261 B.5 Implementation

6262 An implementation is concept that is used to describe a piece of software technology such as a Java
6263 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a
6264 service-oriented application. An SCA composite is also an implementation.

6265 Implementations define points of variability including properties that can be set and settable references to
6266 other services. The points of variability are configured by a component that uses the implementation. The
6267 specification refers to the configurable aspects of an implementation as its **componentType**.

6268 B.6 Interface

6269 Interfaces define one or more business functions. These business functions are provided by Services
6270 and are used by components through References. Services are defined by the Interface they implement.
6271 SCA currently supports a number of interface type systems, for example:

- 6272 • Java interfaces
- 6273 • WSDL portTypes
- 6274 • C, C++ header files

6275

6276 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional
6277 interface type systems.

6278 Interfaces can be **bi-directional**. A bi-directional service has service operations which are provided by
6279 each end of a service communication – this could be the case where a particular service demands a
6280 “callback” interface on the client, which it calls during the process of handing service requests from the
6281 client.

6282 B.7 Composite

6283 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an
6284 assembly of Components, Services, References, and the Wires that interconnect them. Composites can
6285 be used to contribute elements to an **SCA Domain**.

6286 A **composite** has the following characteristics:

- 6287 • It can be used as a component implementation. When used in this way, it defines a boundary for
6288 Component visibility. Components cannot be directly referenced from outside of the composite in
6289 which they are declared.
- 6290 • It can be used to define a unit of deployment. Composites are used to contribute business logic
6291 artifacts to an SCA Domain.

6292 B.8 Composite inclusion

6293 One composite can be used to provide part of the definition of another composite, through the process of
6294 inclusion. This is intended to make team development of large composites easier. Included composites
6295 are merged together into the using composite at deployment time to form a single logical composite.

6296 Composites are included into other composites through `<include.../>` elements in the using composite.
6297 The SCA Domain uses composites in a similar way, through the deployment of composite files to a
6298 specific location.

6299 B.9 Property

6300 **Properties** allow for the configuration of an implementation with externally set data values. The data
6301 value is provided through a Component, possibly sourced from the property of a containing composite.

6302 Each Property is defined by the implementation. Properties can be defined directly through the
6303 implementation language or through annotations of implementations, where the implementation language
6304 permits, or through a componentType file. A Property can be either a simple data type or a complex data
6305 type. For complex data types, XML schema is the preferred technology for defining the data types.

6306 B.10 Domain

6307 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled
6308 by a single organization. As an example, for the accounts department in a business, the SCA Domain
6309 might cover all finance-related functions, and it might contain a series of composites dealing with specific
6310 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

6311 A Domain specifies the instantiation, configuration and connection of a set of components, provided via
6312 one or more composite files. A Domain also contains Wires that connect together the Components. A
6313 Domain does not contain promoted Services or promoted References, since promotion has no meaning
6314 at the Domain level.

6315 B.11 Wire

6316 **SCA wires** connect **service references** to **services**.

6317 Valid wire sources are component references. Valid wire targets are component services.

6318 When using included composites, the sources and targets of the wires don't have to be declared in the
6319 same composite as the composite that contains the wire. The sources and targets can be defined by
6320 other included composites. Targets can also be external to the SCA Domain.

6321 B.12 SCA Runtime

6322 An SCA Runtime is a set of one or more software programs which, when executed, can accept and run
6323 SCA artifacts as defined in the SCA specifications. An SCA runtime provides an implementation of the
6324 SCA Domain and an implementation of capabilities for populating the domain with artifacts and with

6325 capabilities for running specific artifacts. An SCA Runtime can vary in size and organization and can
6326 involve a single process running on a single machine, multiple processes running on a single machine or
6327 multiple processes running across multiple machines that are linked by network communications.
6328 An SCA runtime supports at least one SCA implementation type and also supports at least one binding
6329 type.
6330 SCA Runtimes can include tools provided to assist developers in creating, testing and debugging of SCA
6331 applications and can be used to host and run SCA applications that provide business capabilities.
6332 An SCA runtime can be implemented using any technologies (i.e. it is not restricted to be implemented
6333 using any particular technologies) and it can be hosted on any operating system platform.

6334 **B.13 Channel**

6335 TODO: cut-and-paste from main document once it is finalized.

6336 **B.14 Consumer**

6337 TODO: cut-and-paste from main document once it is finalized.

6338 **B.15 Producer**

6339 TODO: cut-and-paste from main document once it is finalized

6340 **B.16 Filters**

6341 TODO: cut-and-paste from main document once it is finalized

6342 **B.17 Event Types**

6343 TODO: cut-and-paste from main document once it is finalized

6344

Appendix C. Conformance Items

6345 This section contains a list of conformance items for the SCA Assembly specification.

6346 Mandatory Items

Conformance ID	Description
[ASM40001]	The extension of a componentType side file name MUST be .componentType.
[ASM40003]	The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
[ASM40004]	The @name attribute of a <reference/> child element of a <componentType/> MUST be unique amongst the reference elements of that <componentType/>.
[ASM40005]	The @name attribute of a <property/> child element of a <componentType/> MUST be unique amongst the property elements of that <componentType/>.
[ASM40006]	If @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be ignored by the runtime.
[ASM40007]	The value of the property @type attribute MUST be the QName of an XML schema type.
[ASM40008]	The value of the property @element attribute MUST be the QName of an XSD global element.
[ASM40009]	The SCA runtime MUST ensure that any implementation default property value is replaced by a value for that property explicitly set by a component using that implementation.
[ASM40010]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM40011]	When the componentType has @mustSupply="true" for a property element, a component using the implementation MUST supply a value for the property since the implementation has no default value for the property.
[ASM40012]	The value of the property @file attribute MUST be a dereferencable URI to a file containing the value for the property.
[ASM40101]	The @name attribute of a <consumer/> child element of a <componentType/> MUST be unique amongst the consumer elements of that <componentType/>.
[ASM40102]	For artifacts that have service/consumer dual manifestation, the service interface MUST <ul style="list-style-type: none"> • be non-bidirectional • have only one-way operations
[ASM40103]	The @name attribute of a <producer/> child element of a <componentType/> MUST be unique amongst the producer

	elements of that <componentType/>.
[ASM40104]	For artifacts that have reference/producer dual manifestation, the reference interface MUST <ul style="list-style-type: none"> • be non-bidirectional • have only one-way operations
[ASM50001]	The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/>
[ASM50002]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50003]	The @name attribute of a service element of a <component/> MUST match the @name attribute of a service element of the componentType of the <implementation/> child element of the component.
[ASM50004]	If an interface is declared for a component service, the interface MUST provide a compatible subset of the interface declared for the equivalent service in the componentType of the implementation
[ASM50005]	If no binding elements are specified for the service, then the bindings specified for the equivalent service in the componentType of the implementation MUST be used, but if the componentType also has no bindings specified, then <binding.sca/> MUST be used as the binding. If binding elements are specified for the service, then those bindings MUST be used and they override any bindings specified for the equivalent service in the componentType of the implementation.
[ASM50006]	If the callback element is present and contains one or more binding child elements, then those bindings MUST be used for the callback.
[ASM50007]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50008]	The @name attribute of a reference element of a <component/> MUST match the @name attribute of a reference element of the componentType of the <implementation/> child element of the component.
[ASM50009]	The value of multiplicity for a component reference MUST only be equal or further restrict any value for the multiplicity of the reference with the same name in the componentType of the implementation, where further restriction means 0..n to 0..1 or 1..n to 1..1.
[ASM50010]	If @wiredByImpl="true" is set for a reference, then the reference MUST NOT be wired statically within a composite, but left unwired.
[ASM50011]	If an interface is declared for a component reference, the interface MUST provide a compatible superset of the interface

	declared for the equivalent reference in the componentType of the implementation.
[ASM50012]	If no binding elements are specified for the reference, then the bindings specified for the equivalent reference in the componentType of the implementation MUST be used. If binding elements are specified for the reference, then those bindings MUST be used and they override any bindings specified for the equivalent reference in the componentType of the implementation.
[ASM50013]	If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.
[ASM50014]	If @autowire="true", the autowire procedure MUST only be used if no target is identified by any of the other ways listed above. It is not an error if @autowire="true" and a target is also defined through some other means, however in this case the autowire procedure MUST NOT be used.
[ASM50015]	If a binding element has a value specified for a target service using its @uri attribute, the binding element MUST NOT identify target services using binding specific attributes or elements.
[ASM50016]	It is possible that a particular binding type uses more than a simple URI for the address of a target service. In cases where a reference element has a binding subelement that uses more than simple URI, the @uri attribute of the binding element MUST NOT be used to identify the target service - in this case binding specific attributes and/or child elements MUST be used.
[ASM50022]	Where it is detected that the rules for the number of target services for a reference have been violated, either at deployment or at execution time, an SCA Runtime MUST raise an error no later than when the reference is invoked by the component implementation.
[ASM50025]	Where a component reference is promoted by a composite reference, the promotion MUST be treated from a multiplicity perspective as providing 0 or more target services for the component reference, depending upon the further configuration of the composite reference. These target services are in addition to any target services identified on the component reference itself, subject to the rules relating to multiplicity.
[ASM50026]	If a reference has a value specified for one or more target services in its @target attribute, there MUST NOT be any child <binding/> elements declared for that reference.
[ASM50027]	If the @value attribute of a component property element is declared, the type of the property MUST be an XML Schema simple type and the @value attribute MUST contain a single value of that type.
[ASM50028]	If the value subelement of a component property is specified, the type of the property MUST be an XML Schema simple type or an XML schema complex type.
[ASM50029]	If a component property value is declared using a child element of the <property/> element, the type of the property MUST be an

	XML Schema global element and the declared child element MUST be an instance of that global element.
[ASM50031]	The @name attribute of a property element of a <component/> MUST be unique amongst the property elements of that <component/>.
[ASM50032]	If a property is single-valued, the <value/> subelement MUST NOT occur more than once.
[ASM50033]	A property <value/> subelement MUST NOT be used when the @value attribute is used to specify the value for that property.
[ASM50034]	If any <wire/> element with its @replace attribute set to "true" has a particular reference specified in its @source attribute, the value of the @target attribute for that reference MUST be ignored and MUST NOT be used to define target services for that reference.
[ASM50035]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM50036]	The property type specified for the property element of a component MUST be compatible with the type of the property with the same @name declared in the component type of the implementation used by the component. If no type is declared in the component property element, the type of the property declared in the componentType of the implementation MUST be used.
[ASM50037]	The @name attribute of a property element of a <component/> MUST match the @name attribute of a property element of the componentType of the <implementation/> child element of the component.
[ASM50038]	In these cases where the types of two property elements are matched, the types declared for the two <property/> elements MUST be compatible
[ASM50039]	A reference with multiplicity 0..1 MUST have no more than one target service defined.
[ASM50040]	A reference with multiplicity 1..1 MUST have exactly one target service defined.
[ASM50041]	A reference with multiplicity 1..n MUST have at least one target service defined.
[ASM50042]	If a component reference has @multiplicity 0..1 or 1..1 and @nonOverridable==true, then the component reference MUST NOT be promoted by any composite reference.
[ASM50043]	The default value of the @autowire attribute MUST be the value of the @autowire attribute on the component containing the reference, if present, or else the value of the @autowire attribute of the composite containing the component, if present, and if neither is present, then it is "false".
[ASM50044]	When a property has multiple values set, all the values MUST be contained within a single property element.
[ASM50045]	The value of the component property @file attribute MUST be a

	dereferencable URI to a file containing the value for the property.
[ASM50046]	<p>The format of the file which is referenced by the @file attribute of a component property or a componentType property is that it is an XML document which MUST contain an sca:values element which in turn contains one of:</p> <ul style="list-style-type: none"> • a set of one or more <sca:value/> elements each containing a simple string - where the property type is a simple XML type • a set of one or more <sca:value/> elements or a set of one or more global elements - where the property type is a complex XML type
[ASM50101]	The @name attribute of a consumer element of a <component/> MUST be unique amongst the consumer elements of that <component/>.
[ASM50102]	The @name attribute of a consumer element of a <component/> MUST match the @name attribute of a consumer element of the componentType of the <implementation/> child element of the component.
[ASM50103]	The @name attribute of a producer element of a <component/> MUST be unique amongst the producer elements of that <component/>.
[ASM50104]	The @name attribute of a producer element of a <component/> MUST match the @name attribute of a producer element of the componentType of the <implementation/> child element of the component.
[ASM60001]	A composite @name attribute value MUST be unique within the namespace of the composite.
[ASM60002]	@local="true" for a composite means that all the components within the composite MUST run in the same operating system process.
[ASM60003]	The name of a composite <service/> element MUST be unique across all the composite services in the composite.
[ASM60004]	A composite <service/> element's @promote attribute MUST identify one of the component services within that composite.
[ASM60005]	If a composite service interface is specified it MUST be the same or a compatible subset of the interface provided by the promoted component service.
[ASM60006]	The name of a composite <reference/> element MUST be unique across all the composite references in the composite.
[ASM60007]	Each of the URIs declared by a composite reference's @promote attribute MUST identify a component reference within the composite.
[ASM60008]	the interfaces of the component references promoted by a composite reference MUST be the same, or if the composite reference itself declares an interface then each of the component reference interfaces MUST be a compatible subset of the

	composite reference interface..
[ASM60009]	the intents declared on a composite reference and on the component references which it promotes MUST NOT be mutually exclusive.
[ASM60010]	If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST raise an error.
[ASM60011]	The multiplicity of a composite reference MUST be equal to or further restrict the multiplicity of each of the component references that it promotes, with the exception that the multiplicity of the composite reference does not have to require a target if there is already a target on the component reference. This means that a component reference with multiplicity 1..1 and a target can be promoted by a composite reference with multiplicity 0..1, and a component reference with multiplicity 1..n and one or more targets can be promoted by a composite reference with multiplicity 0..n or 0..1.
[ASM60012]	If a composite reference has an interface specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s).
[ASM60013]	If no interface is declared on a composite reference, the interface from one of its promoted component references MUST be used for the component type associated with the composite.
[ASM60014]	The @name attribute of a composite property MUST be unique amongst the properties of the same composite.
[ASM60022]	For each component reference for which autowire is enabled, the SCA runtime MUST search within the composite for target services which have an interface that is a compatible superset of the interface of the reference.
[ASM60024]	The intents, and policies applied to the service MUST be compatible with those on the reference when using autowire to wire a reference – so that wiring the reference to the service will not cause an error due to policy mismatch
[ASM60025]	for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the reference to one of the set of valid target services chosen from the set in a runtime-dependent fashion
[ASM60026]	for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all of the set of valid target services
[ASM60027]	for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT raise an error
[ASM60028]	for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid target services an error MUST be raised by the SCA runtime since the reference is intended to be wired
[ASM60030]	The @name attribute of an <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.

[ASM60031]	The SCA runtime MUST raise an error if the composite resulting from the inclusion of one composite into another is invalid.
[ASM60032]	For a composite used as a component implementation, each composite service offered by the composite MUST promote a component service of a component that is within the composite.
[ASM60033]	For a composite used as a component implementation, every component reference of components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or promoted.
[ASM60034]	For a composite used as a component implementation, all properties of components within the composite, where the underlying component implementation specifies "mustSupply=true" for the property, MUST either specify a value for the property or source the value from a composite property.
[ASM60035]	All the component references promoted by a single composite reference MUST have the same value for @wiredByImpl.
[ASM60036]	If the @wiredByImpl attribute is not specified on the composite reference, the default value is "true" if all of the promoted component references have a wiredByImpl value of "true", and the default value is "false" if all the promoted component references have a wiredByImpl value of "false". If the @wiredByImpl attribute is specified, its value MUST be "true" if all of the promoted component references have a wiredByImpl value of "true", and its value MUST be "false" if all the promoted component references have a wiredByImpl value of "false".
[ASM60037]	<include/> processing MUST take place before the processing of the @promote attribute of a composite reference is performed.
[ASM60038]	<include/> processing MUST take place before the processing of the @promote attribute of a composite service is performed.
[ASM60039]	<include/> processing MUST take place before the @source and @target attributes of a wire are resolved.
[ASM60040]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM60041]	If the included composite has the value <i>true</i> for the attribute @local then the including composite MUST have the same value for the @local attribute, else it is an error.
[ASM60042]	The @name attribute of an include element MUST be the QName of a composite in the SCA Domain.
[ASM60043]	The interface declared by the target of a wire MUST be a compatible superset of the interface declared by the source of the wire.
[ASM60045]	An SCA runtime MUST introspect the componentType of a Composite used as a Component Implementation following the rules defined in the section "Component Type of a Composite used as a Component Implementation"
[ASM60046]	If <service-name> is present, the component service with @name corresponding to <service-name> MUST be used for the wire.

[ASM60047]	If there is no component service with @name corresponding to <service-name>, the SCA runtime MUST raise an error.
[ASM60048]	If <service-name> is not present, the target component MUST have one and only one service with an interface that is a compatible superset of the wire source's interface and satisfies the policy requirements of the wire source, and the SCA runtime MUST use this service for the wire.
[ASM60049]	If <binding-name> is present, the <binding/> subelement of the target service with @name corresponding to <binding-name> MUST be used for the wire.
[ASM60050]	If there is no <binding/> subelement of the target service with @name corresponding to <binding-name>, the SCA runtime MUST raise an error.
[ASM60051]	If <binding-name> is not present and the target service has multiple <binding/> subelements, the SCA runtime MUST choose one and only one of the <binding/> elements which satisfies the mutual policy requirements of the reference and the service, and the SCA runtime MUST use this binding for the wire.
[ASM60101]	The name of the consumer MUST be unique amongst the consumer elements of the composite.
[ASM60102]	A composite <consumer/> element's @promote attribute MUST identify one of the component consumers within that composite.
[ASM60103]	<include/> processing MUST take place before the processing of the @promote attribute of a composite service is performed.
[ASM60104]	The name of the producer MUST be unique amongst the producer elements of the composite.
[ASM60105]	A composite <producer/> element's @promote attribute MUST identify one of the component producers within that composite.
[ASM60106]	<include/> processing MUST take place before the processing of the @promote attribute of a composite service is performed.
[ASM60107]	The name of the channel MUST be unique amongst the channel elements of the composite.
[ASM60108]	An SCA runtime MUST support the use of domain channels.
[ASM80001]	The interface.wsdl @interface attribute MUST reference a portType of a WSDL 1.1 document.
[ASM80002]	Remotable service Interfaces MUST NOT make use of method or operation overloading .
[ASM80003]	If a remotable service is called locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the service or post-invocation modifications to return messages are seen by the caller.
[ASM80004]	If a reference is defined using a bidirectional interface element, the client component implementation using the reference calls the referenced service using the interface. The client MUST provide

	an implementation of the callback interface.
[ASM80005]	Either both interfaces of a bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and remote services.
[ASM80008]	Any service or reference that uses an interface marked with intents MUST implicitly add those intents to its own @requires list.
[ASM80009]	In a bidirectional interface, the service interface can have more than one operation defined, and the callback interface can also have more than one operation defined. SCA runtimes MUST allow an invocation of any operation on the service interface to be followed by zero, one or many invocations of any of the operations on the callback interface.
[ASM80010]	Whenever an interface document declaring a callback interface is used in the declaration of an <interface/> element in SCA, it MUST be treated as being bidirectional with the declared callback interface.
[ASM80011]	If an <interface/> element references an interface document which declares a callback interface and also itself contains a declaration of a callback interface, the two callback interfaces MUST be compatible.
[ASM80016]	The interface.wSDL @callbackInterface attribute, if present, MUST reference a portType of a WSDL 1.1 document.
[ASM80017]	WSDL interfaces are always remotable and therefore an <interface.wSDL/> element MUST NOT contain remotable="false".
[ASM90001]	For a binding of a reference the @uri attribute defines the target URI of the reference. This MUST be either the componentName/serviceName/bindingName for a wire to an endpoint within the SCA Domain, or the accessible address of some service endpoint either inside or outside the SCA Domain (where the addressing scheme is defined by the type of the binding).
[ASM90002]	When a service or reference has multiple bindings, all non-callback bindings of the service or reference MUST have unique names, and all callback bindings of the service or reference MUST have unique names.
[ASM90003]	If a reference has any bindings, they MUST be resolved, which means that each binding MUST include a value for the @uri attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired using other SCA mechanisms.
[ASM90004]	To wire to a specific binding of a target service the syntax "componentName/serviceName/bindingName" MUST be used.
[ASM90005]	For a binding.sca of a component service, the @uri attribute MUST NOT be present.
[ASM10001]	all of the QNames for the definitions contained in definitions.xml files MUST be unique within the Domain.
[ASM10002]	An SCA runtime MUST make available to the Domain all the

	artifacts contained within the definitions.xml files in the Domain.
[ASM10003]	An SCA runtime MUST reject a definitions.xml file that does not conform to the sca-definitions.xsd schema.
[ASM11001]	A conforming implementation type, interface type, import type or export type MUST meet the requirements in "Implementation Type Documentation Requirements for SCA Assembly Model Version 1.2 Specification".
[ASM11002]	A binding extension element MUST be declared as an element in the substitution group of the sca:binding element.
[ASM11003]	A binding extension element MUST be declared to be of a type which is an extension of the sca:Binding type.
[ASM12001]	For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA as a hierarchy of resources based off of a single root
[ASM12005]	Where present, artifact-related or packaging-related artifact resolution mechanisms MUST be used by the SCA runtime to resolve artifact dependencies.
[ASM12006]	SCA requires that all runtimes MUST support the ZIP packaging format for contributions.
[ASM12009]	if there is ever a conflict between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in the dependent contribution list.
[ASM12010]	Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.
[ASM12011]	If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the resource indicated when using the mechanism (e.g. the URI is incorrect or invalid, say) the SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an alternative.
[ASM12012]	The value of @autowire for the logical Domain composite MUST be autowire="false".
[ASM12021]	The SCA runtime MUST raise an error if an artifact cannot be resolved using these mechanisms, if present.
[ASM12022]	There can be multiple import declarations for a given namespace. Where multiple import declarations are made for the same namespace, all the locations specified MUST be searched in lexical order.
[ASM12023]	When a contribution contains a reference to an artifact from a namespace that is declared in an import statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the SCA runtime MUST resolve artifacts in the following order: <ol style="list-style-type: none"> 1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to

	<p>locate artifacts (i.e. only a one-level search is performed).</p> <p>2. from the contents of the contribution itself.</p>
[ASM12024]	The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through resolving an import statement.
[ASM12025]	The SCA runtime MUST raise an error if an artifact cannot be resolved by using artifact-related or packaging-related artifact resolution mechanisms, if present, by searching locations identified by the import statements of the contribution, if present, and by searching the contents of the contribution.
[ASM12026]	An SCA runtime MUST make the <import/> and <export/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-generated.xml files available for the SCA artifact resolution process.
[ASM12027]	An SCA runtime MUST reject files that do not conform to the schema declared in sca-contribution.xsd.
[ASM12028]	An SCA runtime MUST merge the contents of sca-contribution-generated.xml into the contents of sca-contribution.xml, with the entries in sca-contribution.xml taking priority if there are any conflicting declarations.
[ASM12030]	For XML definitions, which are identified by QNames, the @namespace attribute of the export element MUST be the namespace URI for the exported definitions.
[ASM12031]	When a contribution uses an artifact contained in another contribution through SCA artifact resolution, if that artifact itself has dependencies on other artifacts, the SCA runtime MUST resolve these dependencies in the context of the contribution containing the artifact, not in the context of the original contribution.
[ASM12032]	Checking for errors in artifacts MUST NOT be done for artifacts in the Installed state (ie where the artifacts are simply part of installed contributions)
[ASM12033]	Errors in artifacts MUST be detected either during the Deployment of the artifacts, or during the process of putting the artifacts into the Running state,
[ASM12034]	<p>For a domain level component with a Property whose value is obtained from a Domain-level Property through the use of the @source attribute, if the domain level property is updated by means of deployment actions, the SCA runtime MUST</p> <ul style="list-style-type: none"> • either update the property value of the domain level component once the update of the domain property is complete • or defer the updating of the component property value until the component is stopped and restarted
[ASM13001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd, sca-interface-wsdl.xsd, sca-implementation-composite.xsd and sca-binding-sca.xsd schema.

[ASM13002]	An SCA runtime MUST reject a contribution file that does not conform to the sca-contribution.xsd schema.
[ASM13003]	An SCA runtime MUST reject a definitions file that does not conform to the sca-definitions.xsd schema.
[ASM14005]	An SCA Runtime MUST raise an error for every situation where the configuration of the SCA Domain or its contents are in error. The error is either raised at deployment time or at runtime, depending on the nature of the error and the design of the SCA Runtime.
[ASM16001]	Any event type definition used in SCA MUST be mappable to WS-EventDescriptions.
[ASM17001]	An SCA runtime is not required to support filter types not defined by this specification - but if an extended filter type is declared within a <filters/> element and the SCA runtime does not support that extended filter type, then the SCA runtime MUST generate an error when it encounters the declaration.
[ASM17002]	If the componentType has a value for @typeNames then the value of @typeNames for the component producer element MUST match that in the componentType.
[ASM17003]	If the componentType has a value for @namespaces then the value of @namespaces for the component producer element MUST match that in the componentType.
[ASM17004]	If the associated component producer or the componentType producer has a value for @qnames then the value of @qnames, if present, for the composite producer element MUST match that in the component producer or the componentType producer.

6347

C.1 Non-mandatory Items

Conformance ID	Description	Classification
[ASM20101]	A producer SHOULD only produce event type it has declared.	Interoperation
[ASM20102]	An SCA Runtime MAY reject events of a type from a producer which does not declare that it produces events of that type.	Interoperation
[ASM60109]	An SCA runtime MAY support the use of private channels.	Enhancement
[ASM12002]	Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy named META-INF	Interoperation
[ASM12003]	Within any contribution packaging a document SHOULD exist directly under the META-INF directory named sca-contribution.xml which lists the SCA Composites within the contribution that are runnable.	Interoperation
[ASM12007]	Implementations of SCA MAY also raise an error if there are conflicting names exported from multiple	Development

	contributions.	
[ASM12029]	An SCA runtime MAY deploy the composites in <deployable/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-generated.xml files.	Interoperation

6348

6349

Appendix D. Acknowledgements

6350 The following individuals have participated in the creation of this specification and are gratefully
6351 acknowledged:

6352 **Participants:**

6353

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Megan Beynon	IBM
Vladislav Bezrukov	SAP AG*
Henning Blohm	SAP AG*
Fraser Bohm	IBM
David Booz	IBM
Fred Carter	AmberPoint
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Mark Combellack	Avaya, Inc.
Jean-Sebastien Delfino	IBM
David DiFranco	Oracle Corporation
Mike Edwards	IBM
Jeff Estefan	Jet Propulsion Laboratory:*
Raymond Feng	IBM
Billy Feng	Primeton Technologies, Inc.
Paul Fremantle	WSO2*
Robert Freund	Hitachi, Ltd.
Peter Furniss	Iris Financial Solutions Ltd.
Genadi Genov	SAP AG*
Mark Hapner	Sun Microsystems
Zahir HEZOUAT	IBM
Simon Holdsworth	IBM
Sabin Ielceanu	TIBCO Software Inc.
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Mike Kaiser	IBM
Khanderao Kand	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Nickolaos Kavantzias	Oracle Corporation
Rainer Kerth	SAP AG*
Dieter Koenig	IBM
Meeraj Kunnumpurath	Individual
Jean Baptiste Laviron	Axway Software*

Simon Laws	IBM
Rich Levinson	Oracle Corporation
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Carl Mattocks	CheckMi*
Jeff Mischkinsky	Oracle Corporation
Ian Mitchell	IBM
Dale Moberg	Axway Software*
Simon Moser	IBM
Simon Nash	Individual
Peter Niblett	IBM
Duane Nickull	Adobe Systems
Eisaku Nishiyama	Hitachi, Ltd.
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*
Gilbert Pilz	Oracle Corporation
Nilesh Rade	Deloitte Consulting LLP
Martin Raeppele	SAP AG*
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vicki Shipkowitz	SAP AG*
Ivana Trickovic	SAP AG*
Clemens Utschig - Utschig	Oracle Corporation
Scott Vorthmann	TIBCO Software Inc.
Feng Wang	Primeton Technologies, Inc.
Tim Watson	Oracle Corporation
Eric Wells	Hitachi, Ltd.
Robin Yang	Primeton Technologies, Inc.
Prasad Yendluri	Software AG, Inc.*

6354

Appendix E. Non-Normative Text

Appendix F. Revision History

Revision	Date	Editor	Changes Made
1	2007-09-24	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
2	2008-01-04	Michael Beisiegel	<p>composite section</p> <ul style="list-style-type: none"> - changed order of subsections from property, reference, service to service, reference, property - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) - added section in appendix to contain complete pseudo schema of composite <p>- moved component section after implementation section</p> <ul style="list-style-type: none"> - made the ConstrainingType section a top level section - moved interface section to after constraining type section <p>component section</p> <ul style="list-style-type: none"> - added subheadings for Implementation, Service, Reference, Property - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) <p>implementation section</p> <ul style="list-style-type: none"> - changed title to "Implementation and ComponentType" - moved implementation instance related stuff from implementation section to component implementation section - added subheadings for Service, Reference, Property, Implementation - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) - attribute and element description still needs to be completed, all implementation statements on services, references, and properties should go here

			<ul style="list-style-type: none"> - added complete pseudo schema of componentType in appendix - added "Quick Tour by Sample" section, no content yet - added comment to introduction section that the following text needs to be added <ul style="list-style-type: none"> "This specification is defined in terms of infoset and not XML 1.0, even though the spec uses XML 1.0/1.1 terminology. A mapping from XML to infoset (... link to infoset specification ...) is trivial and should be used for non-XML serializations."
3	2008-02-15	Anish Karmarkar Michael Beisiegel	<p>Incorporated resolutions from 2008 Jan f2f.</p> <ul style="list-style-type: none"> - issue 9 - issue 19 - issue 21 - issue 4 - issue 1A - issue 27 <ul style="list-style-type: none"> - in Implementation and ComponentType section added attribute and element description for service, reference, and property - removed comments that helped understand the initial restructuring for WD02 - added changes for issue 43 - added changes for issue 45, except the changes for policySet and requires attribute on property elements - used the NS http://docs.oasis-open.org/ns/opencsa/sca/200712 - updated copyright stmt - added wordings to make PDF normative and xml schema at the NS uri authoritative
4	2008-04-22	Mike Edwards	<p>Editorial tweaks for CD01 publication:</p> <ul style="list-style-type: none"> - updated URL for spec documents - removed comments from published CD01 version - removed blank pages from body of spec
5	2008-06-30	Anish Karmarkar Michael Beisiegel	<p>Incorporated resolutions of issues: 3, 6, 14 (only as it applies to the component property element), 23, 25, 28, 25, 38, 39, 40, 42, 45 (except for adding @requires and @policySets to property elements), 57, 67, 68, 69</p>
6	2008-09-23	Mike Edwards	<p>Editorial fixes in response to Mark Combella's review contained in email: http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html</p>
7 CD01 - Rev3	2008-11-18	Mike Edwards	<ul style="list-style-type: none"> • Specification marked for conformance statements. New Appendix (D) added containing a table of all conformance

			statements. Mass of related minor editorial changes to remove the use of RFC2119 words where not appropriate.
8 CD01 - Rev4	2008-12-11	Mike Edwards	<ul style="list-style-type: none"> - Fix problems of misplaced statements in Appendix D - Fixed problems in the application of Issue 57 - section 5.3.1 & Appendix D as defined in email: http://lists.oasis-open.org/archives/sca-assembly/200811/msg00045.html - Added Conventions section, 1.3, as required by resolution of Issue 96. - Issue 32 applied - section B2 - Editorial addition to section 8.1 relating to no operation overloading for remotable interfaces, as agreed at TC meeting of 16/09/2008.
9 CD01 - Rev5	2008-12-22	Mike Edwards	<ul style="list-style-type: none"> - Schemas in Appendix B updated with resolutions of Issues 32 and 60 - Schema for contributions - Appendix B12 - updated with resolutions of Issues 53 and 74. - Issues 53 and 74 incorporated - Sections 11.4, 11.5
10 CD01-Rev6	2008-12-23	Mike Edwards	<ul style="list-style-type: none"> - Issues 5, 71, 92 - Issue 14 - remaining updates applied to ComponentType (section 4.1.3) and to Composite Property (section 6.3)
11 CD01-Rev7	2008-12-23	Mike Edwards	<p>All changes accepted before revision from Rev6 started - due to changes being applied to previously changed sections in the Schemas</p> <ul style="list-style-type: none"> Issues 12 & 18 - Section B2 Issue 63 - Section C3 Issue 75 - Section C12 Issue 65 - Section 7.0 Issue 77 - Section 8 + Appendix D Issue 69 - Sections 5.1, 8 Issue 45 - Sections 4.1.3, 5.4, 6.3, B2. Issue 56 - Section 8.2, Appendix D Issue 41 - Sections 5.3.1, 6.4, 12.7, 12.8, Appendix D
12 CD01-Rev8	2008-12-30	Mike Edwards	<ul style="list-style-type: none"> Issue 72 - Removed Appendix A Issue 79 - Sections 9.0, 9.2, 9.3, Appendix A.2 Issue 62 - Sections 4.1.3, 5.4 Issue 26 - Section 6.5 Issue 51 - Section 6.5 Issue 36 - Section 4.1 Issue 44 - Section 10, Appendix C Issue 89 - Section 8.2, 8.5, Appendix A, Appendix C Issue 16 - Section 6.8, 9.4 Issue 8 - Section 11.2.1 Issue 17 - Section 6.6 Issue 30 - Sections 4.1.1, 4.1.2, 5.2, 5.3, 6.1, 6.2, 9 Issue 33 - insert new Section 8.4
12 CD01-Rev8a	2009-01-13	Bryan Aupperle Mike Edwards	Issue 99 - Section 8

13 CD02	2009-01-14	Mike Edwards	All changes accepted All comments removed
14 CD02-Rev2	2009-01-30	Mike Edwards	Issue 94 applied (removal of conversations)
15 CD02-Rev3	2009-01-30	Mike Edwards	Issue 98 - Section 5.3 Minor editorial cleanup (various locations) Removal of <operation/> element as decided at Jan 2009 F2F - various sections Issue 95 - Section 6.2 Issue 2 - Section 2.1 Issue 37 - Sections 2.1, 6, 12.6.1, B10 Issue 48 - Sections 5.3, A2 Issue 90 - Sections 6.1, 6.2, 6.4 Issue 64 - Sections 7, A2 Issue 100 - Section 6.2 Issue 103 - Sections 10, 12.2.2, A.13 Issue 104 - Sections 4.1.3, 5.4, 6.3 Section 3 (Quick Tour By Sample) removed by decision of Jan 2009 Assembly F2F meeting
16 CD02-Rev4	2009-02-06	Mike Edwards	All changes accepted Major Editorial work to clean out all RFC2119 wording and to ensure that no normative statements have been missed.
16 CD02-Rev6	2009-02-24	Mike Edwards	Issue 107 - sections 4, 5, 11, Appendix C Editorial updates resulting from Review Issue 34 - new section 12 inserted, + minor editorial changes in sections 4, 11 Issue 110 - Section 8.0 Issue 111 - Section 4.4, Appendix C Issue 112 - Section 4.5 Issue 113 - Section 3.3 Issue 108 - Section 13, Appendix C Minor editorial changes to the example in section 3.3
17 CD02-Rev7	2009-03-02	Mike Edwards	Editorial changes resulting from Vamsi's review of CD02 Rev6 Issue 109 - Section 8, Appendix A.2, Appendix B.3.1, Appendix C Added back @requires and @policySets to <interface/> as editorial correction since they were lost by accident in earlier revision Issue 101 - Section 13 Issue 120 - Section
18 CD02-Rev8	2009-03-05	Mike Edwards	XSDs corrected and given new namespace. Namespace updated throughout document.
19 CD03	2009-03-05	Mike Edwards	All Changes Accepted
20 CD03	2009-03-17	Anish Karmarkar	Changed CD03 per TC's CD03/PR01 resolution. Fixed the footer, front page.
21 CD03 Rev1	2009-06-16	Mike Edwards	Issue 115 - Sections 3.1.3, 4.4, 5.3, A.2 Editorial: Use the form "portType" in all cases when referring to WSDL portType Issue 117 - Sections 4.2, 4.3, 5.0, 5.1, 5.2, 5.4, 5.4.2, 6.0, add new 7.2, old 7.2 Note: REMOVED assertions: ASM60015 ASM60015 ASM60016 ASM60017

			<p>ASM60018 ASM60019 ASM60020 ASM60023 ASM60024 ASM80012 ASM80013 ASM80014 ASM80015 ADDED ASM70007 Issue 122 - Sections 4.3, 4.3.1, 4.3.1.1, 6.0, 8.0, 11.6 Issue 123 - Section A.2 Issue 124 - Sections A2, A5 Issue 125 - Section 7.6 Editorial - fixed broken reference links in Sections 7.0, 11.2 Issue 126 - Section 7.6 Issue 127 - Section 4.4, added Section 4.4.1 Issue 128 - Section A2 Issue 129 - Section A2 Issue 130 - multiple sections Issue 131 - Section A.11 Issue 135 - Section 8.4.2 Issue 141 - Section 4.3</p>
22 CD03 Rev2	2009-07-28	Mike Edwards	<p>Issue 151 - Section A.2 Issue 133 - Sections 7, 11.2 Issue 121 - Section 13.1, 13.2, C.1, C.2 Issue 134 - Section 5.2 Issue 153 - Section 3.2, 5.3.1</p>
23 CD03 Rev3	2009-09-23	Mike Edwards	<p>Major formatting update - all snippets and examples given a caption and consistent formatting. All references to snippets and examples updated to use the caption numbering. Issue 147 - Section 5.5.1 added Issue 136 - Section 4.3, 5.2 Issue 144 - Section 4.4 Issue 156 - Section 8 Issue 160 - Section 12.1 Issue 176 - Section A.5 Issue 180 - Section A.1 Issue 181 - Section 5.1, 5.2</p>
24 CD03 Rev4	2009-09-23	Mike Edwards	<p>All changes accepted Issue 157 - Section 6 removed, other changes scattered through many other sections, including the XSDs and normative statements. Issue 182 - Appendix A</p>
25 CD03 Rev5	2009-11-20	Mike Edwards	<p>All changes accepted Issue 138 - Section 10.3 added Issue 142 - Section 4.3 updated Issue 143 - Section 7.5 updated Issue 145 - Section 4.4 updated Issue 158 - Section 5.3.1 updated Issue 183 - Section 7.5 updated Issue 185 - Section 10.9 updated</p>
26 CD03 Rev6	2009-12-03	Mike Edwards	<p>All changes accepted Issue 175 - Section A2 updated Issue 177 - Section A2 updated Issue 188 - Sections 3.1.1, 3.1.2, 3.1.4, 4, 4.1, 4.2, 4.3, 5, 5.1, 5.2, 6, 6.6, 7, 7.5, 9, A2 updated Issue 192 - editorial fixes in Sections 5.1, 5.2,</p>

			5.4.1, 5.5, 5.6.1 SCA namespace updated to http://docs.oasis-open.org/ns/opencsa/sca/200912 as decided at Dec 1 st F2F meeting - changes scattered through the document Issue 137 - Sections 5.4, 7 updated Issue 189 - Section 6.5 updated
27 CD04	2009-12-09	Mike Edwards	All changes accepted
28 CD05	2010-01-12	Mike Edwards	All changes accepted Issue 215 – Section 8 and A.12
29 CD05 Rev1	2010-07-13	Bryan Aupperle	Issue 221 – Sections 3.1.3, 4.4 updated and 4.4.2 added Issue 222 – Section 8 and A.12 updated Issue 223 – Sections A.2 and A.11 updated Issue 225 – Section B.12 added Issue 228 – Section A.2 updated Issue 229 – Section 5 updated
30 CD05 Rev2	2010-08-10	Mike Edwards Bryan Aupperle	Issue 237 – Section A.1 updated Templated requirements – Section 1.4 added References to other SCA specifications updated to current drafts – Section 1.3 updated
31 CD06	2010-08-10	Mike Edwards	All changes accepted Editorial cleaning
32 WD061	2011-01-04	Mike Edwards	Issue 252 - Sections 1.2 & 12.2 updated
33 v1.2 wd02	2011-01-24	Anish Karmarkar	Applied resolutions of issues: 238, 241, 242
34 v1.2 wd03	2011-01-25	Bryan Aupperle	Synchronize v1.2 with v1.1 CSD07
35 v1.2 wd04	2011-05-24	Anish Karmarkar	Synchronize v1.2 with v1.1 WD074 (approved as csd08) Applied resolution of issue 256 Removed last paragraph of Section 7 because of resolution of issue 242 Labeled conformance items: 20101-20102, 40101-40104, 50101-50102, 60101-60109, 16001, 17001-17004
36 v1.2 wd05	2011-06-01	Anish Karmarkar	Sync v1.2 with v1.1 WD075 Fixed names of the schema includes to point to 1.2 XSDs
37 v1.2 wd06	2011-06-28	Anish Karmarkar	Issue 245 resolution and ed fixes to remove "????"

6358