



Service Component Architecture Assembly Model Specification

Version 1.1

Committee Specification Draft 07

18 January 2011

Specification URIs:

This Version:

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-csd07.html>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-csd07.doc>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-csd07.pdf>
(Authoritative)

Previous Version:

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.html>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.doc>
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd06.pdf> (Authoritative)

Latest Version:

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

Technical Committee:

OASIS Service Component Architecture / Assembly (SCA-Assembly) TC

Chair(s):

Martin Chapman, Oracle
Mike Edwards, IBM

Editor(s):

Michael Beisiegel, IBM
Khanderao Khand, Oracle
Anish Karmarkar, Oracle
Sanjay Patil, SAP
Michael Rowley, Active Endpoints

Related work:

This specification replaces or supercedes:

- [Service Component Architecture Assembly Model Specification Version 1.00, March 15, 2007](#)

This specification is related to:

- [Service Component Architecture Policy Framework Specification Version 1.1](#)

Declared XML Namespace(s):

<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.

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

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

Service Component Architecture Assembly Model Specification Version 1.1. 18 January 2011. OASIS Committee Specification Draft. <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-csd07.pdf>

Notices

Copyright © OASIS® 2005-2011. 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 names "OASIS", "SCA" and "Service Component Architecture" are trademarks of [OASIS](http://www.oasis-open.org), 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
2	Overview.....	11
2.1	Diagram used to Represent SCA Artifacts.....	12
3	Implementation and ComponentType.....	14
3.1	Component Type.....	14
3.1.1	Service.....	15
3.1.2	Reference.....	16
3.1.3	Property.....	18
3.1.4	Implementation.....	19
3.2	Example ComponentType.....	20
3.3	Example Implementation.....	20
4	Component.....	23
4.1	Implementation.....	24
4.2	Service.....	25
4.3	Reference.....	26
4.3.1	Specifying the Target Service(s) for a Reference.....	29
4.4	Property.....	30
4.4.1	Property Type Compatibility.....	33
4.4.2	Property Value File Format.....	34
4.5	Example Component.....	34
5	Composite.....	38
5.1	Service.....	39
5.1.1	Service Examples.....	41
5.2	Reference.....	42
5.2.1	Example Reference.....	45
5.3	Property.....	46
5.3.1	Property Examples.....	47
5.4	Wire.....	49
5.4.1	Wire Examples.....	51
5.4.2	Autowire.....	53
5.4.3	Autowire Examples.....	54
5.5	Using Composites as Component Implementations.....	56
5.5.1	Component Type of a Composite used as a Component Implementation.....	57
5.5.2	Example of Composite used as a Component Implementation.....	58
5.6	Using Composites through Inclusion.....	59
5.6.1	Included Composite Examples.....	60
5.7	Composites which Contain Component Implementations of Multiple Types.....	63
5.8	Structural URI of Components.....	63
6	Interface.....	64

6.1	Local and Remotable Interfaces	65
6.2	Interface Compatibility	65
6.2.1	Compatible Interfaces.....	66
6.2.2	Compatible Subset	66
6.2.3	Compatible Superset	67
6.3	Bidirectional Interfaces	67
6.4	Long-running Request-Response Operations	68
6.4.1	Background	68
6.4.2	Definition of "long-running"	69
6.4.3	The asyncInvocation Intent.....	69
6.4.4	Requirements on Bindings	69
6.4.5	Implementation Type Support	69
6.5	SCA-Specific Aspects for WSDL Interfaces.....	69
6.6	WSDL Interface Type	70
6.6.1	Example of interface.wsdl.....	71
7	Binding	72
7.1	Messages containing Data not defined in the Service Interface	74
7.2	WireFormat	74
7.3	OperationSelector	74
7.4	Form of the URI of a Deployed Binding	75
7.4.1	Non-hierarchical URIs	75
7.4.2	Determining the URI scheme of a deployed binding	75
7.5	SCA Binding	76
7.5.1	Example SCA Binding	77
7.6	Web Service Binding	77
7.7	JMS Binding.....	78
8	SCA Definitions	79
9	Extension Model.....	80
9.1	Defining an Interface Type.....	80
9.2	Defining an Implementation Type	81
9.3	Defining a Binding Type.....	83
9.4	Defining an Import Type	85
9.5	Defining an Export Type	86
10	Packaging and Deployment.....	89
10.1	Domains.....	89
10.2	Contributions.....	89
10.2.1	SCA Artifact Resolution	90
10.2.2	SCA Contribution Metadata Document.....	92
10.2.3	Contribution Packaging using ZIP	94
10.3	States of Artifacts in the Domain.....	94
10.4	Installed Contribution	95
10.4.1	Installed Artifact URIs	95
10.5	Operations for Contributions.....	95
10.5.1	install Contribution & update Contribution	95
10.5.2	add Deployment Composite & update Deployment Composite.....	96

10.5.3	remove Contribution	96
10.6	Use of Existing (non-SCA) Mechanisms for Resolving Artifacts	96
10.7	Domain-Level Composite	97
10.7.1	add To Domain-Level Composite	97
10.7.2	remove From Domain-Level Composite	97
10.7.3	get Domain-Level Composite	97
10.7.4	get QName Definition	97
10.8	Dynamic Behaviour of Wires in the SCA Domain	98
10.9	Dynamic Behaviour of Component Property Values	98
11	SCA Runtime Considerations	99
11.1	Error Handling	99
11.1.1	Errors which can be Detected at Deployment Time	99
11.1.2	Errors which are Detected at Runtime	99
12	Conformance	100
12.1	SCA Documents	100
12.2	SCA Runtime	100
12.2.1	Optional Items	101
A.	XML Schemas	102
A.1	sca.xsd	102
A.2	sca-core.xsd	102
A.3	sca-binding-sca.xsd	109
A.4	sca-interface-java.xsd	109
A.5	sca-interface-wsdl.xsd	110
A.6	sca-implementation-java.xsd	110
A.7	sca-implementation-composite.xsd	110
A.8	sca-binding-webservice.xsd	111
A.9	sca-binding-jms.xsd	111
A.10	sca-policy.xsd	111
A.11	sca-contribution.xsd	111
A.12	sca-definitions.xsd	112
B.	SCA Concepts	114
B.1	Binding	114
B.2	Component	114
B.3	Service	114
B.3.1	Remotable Service	114
B.3.2	Local Service	115
B.4	Reference	115
B.5	Implementation	115
B.6	Interface	115
B.7	Composite	116
B.8	Composite inclusion	116
B.9	Property	116
B.10	Domain	116
B.11	Wire	116
B.12	SCA Runtime	116

C.	Conformance Items	118
C.1	Mandatory Items	118
C.2	Non-mandatory Items.....	128
D.	Acknowledgements	130
E.	Revision History	132

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

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]

WSDL Specification version 1.1
<http://www.w3.org/TR/wsd>

41
42 **[SCA-WSBINDING]**
43 OASIS Committee Draft 04, "SCA Web Services Binding Specification Version 1.1", May 2010
44 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd04.pdf>
45
46 **[SCA-POLICY]**
47 OASIS Committee Draft 04, "SCA Policy Framework Specification Version 1.1", September 2010
48 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1.pdf>
49
50 **[SCA-JMSBINDING]**
51 OASIS Committee Draft 05, "SCA JMS Binding Specification Version 1.1 Version 1.1", November
52 2010
53 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-jmsbinding-1.1-spec-csprd03.pdf>
54
55 **[SCA-CPP-Client]**
56 OASIS Committee Draft 06, "SCA Client and Implementation for C++ Specification Version 1.1",
57 October 2010
58 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd06.pdf>
59
60 **[SCA-C-Client]**
61 OASIS Committee Draft 06, "SCA Client and Implementation for C Specification Version 1.1",
62 October 2010
63 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd06.pdf>
64
65 **[ZIP-FORMAT]**
66 ZIP Format Definition
67 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>
68
69 **[XML-INFOSET]**
70 Infoset Specification
71 <http://www.w3.org/TR/xml-infoset/>
72
73 **[WSDL11_Identifiers]**
74 WSDL 1.1 Element Identifiers
75 <http://www.w3.org/TR/wsdl11elementidentifiers/>
76
77 **[SCA-TSA]**
78 OASIS Committee Draft 01, " Test Suite Adaptation for SCA Assembly Model Version 1.1
79 Specification", July 2010
80 <http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-testsuite-adaptation-cd01.pdf>
81
82 **[SCA-IMPLTYPDOC]**

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

87

88 1.3 Non-Normative References

89 [SDO]

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

92 [JAX-WS]

93 JAX-WS Specification

94 <http://jcp.org/en/jsr/detail?id=224>

95

96 [WSI-BP]

97 WS-I Basic Profile

98 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>

99

100 [WSI-BSP]

101 WS-I Basic Security Profile

102 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>

103

104 [WS-BPEL]

105 OASIS Standard, "Web Services Business Process Execution Language Version 2.0", April 2007

106 <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>

107

108 1.4 Naming Conventions

109 This specification follows naming conventions for artifacts defined by the specification:

- 110 • For the names of elements and the names of attributes within XSD files, the names follow the
111 CamelCase convention, with all names starting with a lower case letter.
112 e.g. <element name="componentType" type="sca:ComponentType"/>
- 113 • For the names of types within XSD files, the names follow the CamelCase convention with all names
114 starting with an upper case letter.
115 eg. <complexType name="ComponentService">
- 116 • For the names of intents, the names follow the CamelCase convention, with all names starting with a
117 lower case letter, EXCEPT for cases where the intent represents an established acronym, in which
118 case the entire name is in upper case.
119 An example of an intent which is an acronym is the "SOAP" intent.

120 2 Overview

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

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

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

137 One basic artifact of SCA is the **component**, which is the unit of construction for SCA. A component
138 consists of a configured instance of an implementation, where an implementation is the piece of program
139 code providing business functions. The business function is offered for use by other components as
140 **services**. Implementations can depend on services provided by other components – these dependencies
141 are called **references**. Implementations can have settable **properties**, which are data values which
142 influence the operation of the business function. The component **configures** the implementation by
143 providing values for the properties and by wiring the references to services provided by other
144 components.

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

148 SCA describes the content and linkage of an application in assemblies called **composites**. Composites
149 can contain components, services, references, property declarations, plus the wiring that describes the
150 connections between these elements. Composites can group and link components built from different
151 implementation technologies, allowing appropriate technologies to be used for each business task. In
152 turn, composites can be used as complete component implementations: providing services, depending on
153 references and with settable property values. Such composite implementations can be used in
154 components within other composites, allowing for a hierarchical construction of business solutions, where
155 high-level services are implemented internally by sets of lower-level services. The content of composites
156 can also be used as groupings of elements which are contributed by inclusion into higher-level
157 compositions.

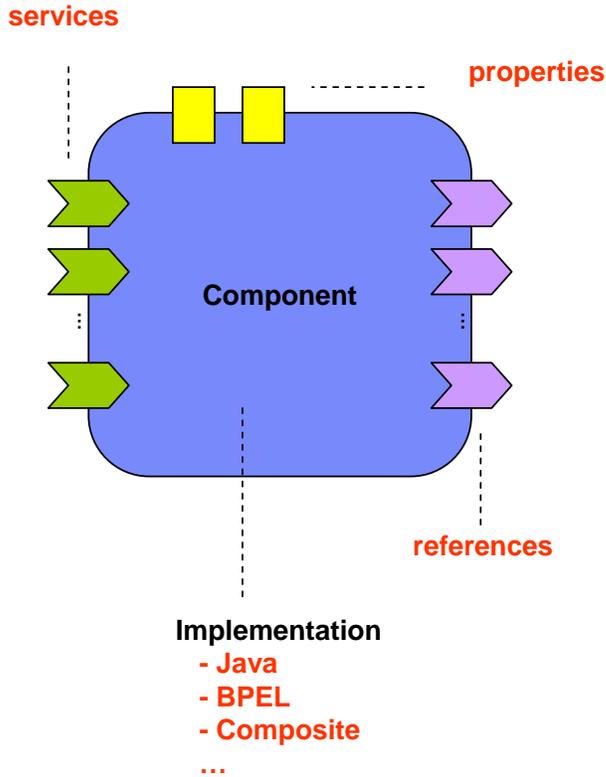
158 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services
159 providing an area of business functionality that is controlled by a single organization. As an example, for
160 the accounts department in a business, the SCA Domain might cover all financial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

164 SCA defines an XML file format for its artifacts. These XML files define the portable representation of the
165 SCA artifacts. An SCA runtime might have other representations of the artifacts represented by these
166 XML files. In particular, component implementations in some programming languages might have
167 attributes or properties or annotations which can specify some of the elements of the SCA Assembly
168 model. The XML files define a static format for the configuration of an SCA Domain. An SCA runtime
169 might also allow for the configuration of the Domain to be modified dynamically.

170 **2.1 Diagram used to Represent SCA Artifacts**

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

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

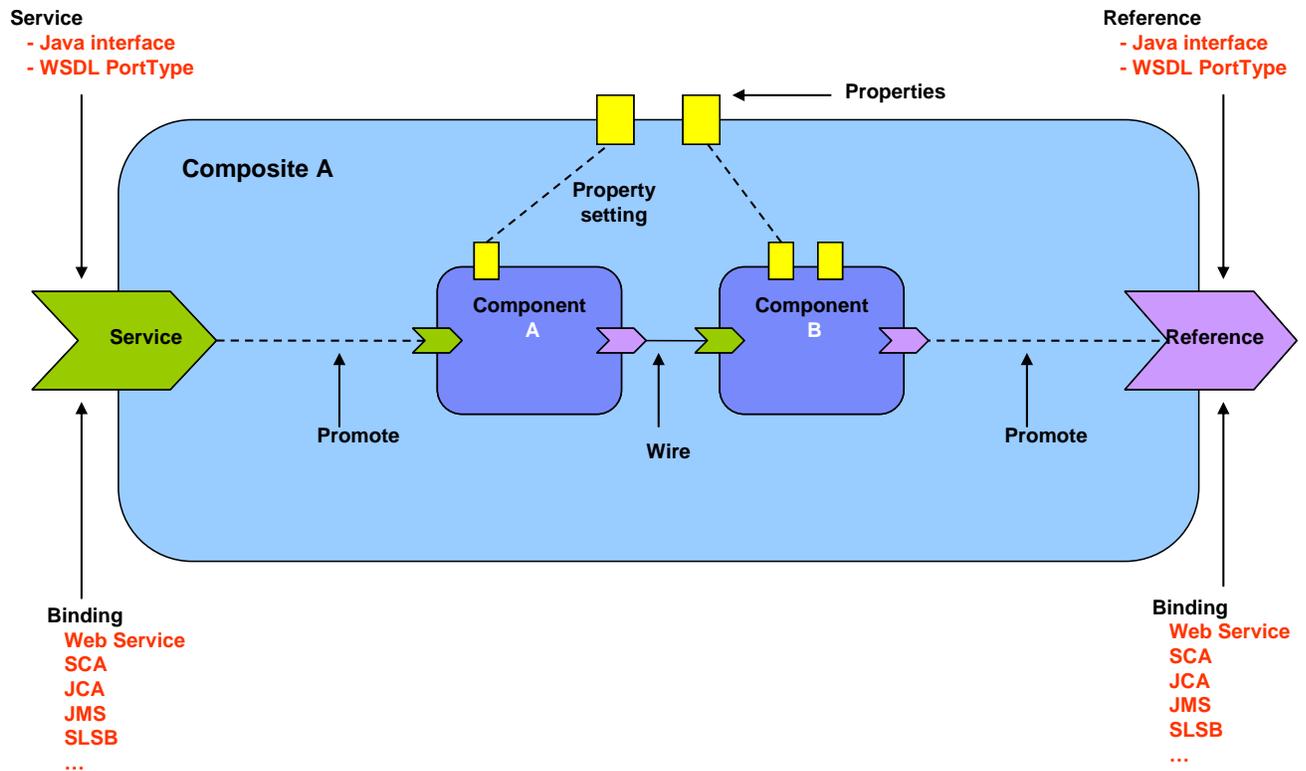


176

177 *Figure 2-1: SCA Component Diagram*

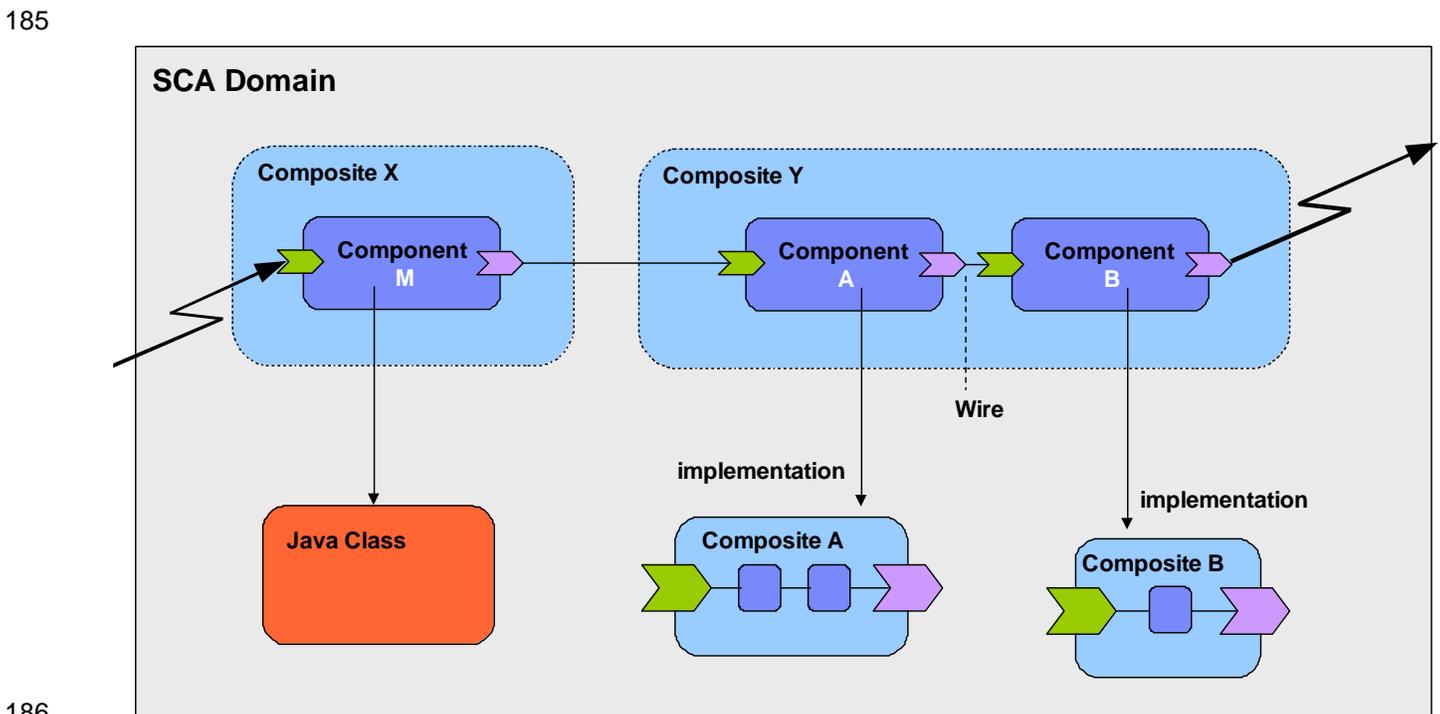
178 Figure 2-2 illustrates some of the features of a composite assembled using a set of components:

179



180
181 *Figure 2-2: SCA Composite Diagram*

182
183 Figure 2-3 illustrates an SCA Domain assembled from a series of high-level composites, some of which
184 are in turn implemented by lower-level composites:



186
187 *Figure 2-3: SCA Domain Diagram*

188

3 Implementation and ComponentType

189 Component **implementations** are concrete implementations of business function which provide services
190 and/or which make references to services provided elsewhere. In addition, an implementation can have
191 some settable property values.

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

197 **Services, references and properties** are the **configurable aspects of an implementation**. SCA refers
198 to them collectively as the **component type**.

199 Depending on the implementation type, the implementation can declare the services, references and
200 properties that it has and it also might be able to set values for all the characteristics of those services,
201 references and properties.

202 So, for example:

- 203 • for a service, the implementation might define the interface, binding(s), a URI, intents, and policy sets,
204 including details of the bindings
- 205 • for a reference, the implementation might define the interface, binding(s), target URI(s), intents, policy
206 sets, including details of the bindings
- 207 • for a property the implementation might define its type and a default value
- 208 • the implementation itself might define policy intents or concrete policy sets

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

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

3.1 Component Type

217 **Component type** represents the configurable aspects of an implementation. A component type consists
218 of services that are offered, references to other services that can be wired and properties that can be set.
219 The settable properties and the settable references to services are configured by a component that uses
220 the implementation.
221

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

225 Since SCA allows a broad range of implementation technologies, it is expected that some implementation
226 technologies (for example, the Java Component Implementation Specification Version 1.1 [SCA-Java])
227 allow for introspecting the implementation artifact(s) (for example, a Java class) to derive the component
228 type information. Other implementation technologies might not allow for introspection of the
229 implementation artifact(s). In those cases where introspection is not allowed, SCA encourages the use of
230 a SCA component type side file. A **component type side file** is an XML file whose document root
231 element is `sca:componentType`.

232 The implementation type specification defines whether introspection is allowed, whether a side file is
233 allowed, both are allowed or some other mechanism specifies the component type. The component type
234 information derived through introspection is called the **introspected component type**. In any case, the

235 implementation type specification specifies how multiple sources of information are combined to produce
236 the **effective component type**. The effective component type is the component type metadata that is
237 presented to the using component for configuration.

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

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

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

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

245

```
246 <?xml version="1.0" encoding="ASCII"?>  
247 <!-- Component type schema snippet -->  
248 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
249  
250     <service ... />*  
251     <reference ... />*  
252     <property ... />*  
253     <implementation ... />?  
254  
255 </componentType>
```

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

257

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

- 259 • **service : Service (0..n)** – see component type service section.
- 260 • **reference : Reference (0..n)** – see component type reference section.
- 261 • **property : Property (0..n)** – see component type property section.
- 262 • **implementation : Implementation (0..1)** – see component type implementation
263 section.

264 3.1.1 Service

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

269

```
270 <?xml version="1.0" encoding="ASCII"?>  
271 <!-- Component type service schema snippet -->  
272 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
273  
274     <service name="xs:NCName"  
275         requires="list of xs:QName"? policySets="list of xs:QName"?>*  
276         <interface ... />  
277         <binding ... />*  
278         <callback?>  
279             <binding ... />+  
280         </callback>  
281         <requires/>*  
282         <policySetAttachment/>*  
283     </service>  
284  
285     <reference ... />*
```

```

286     <property ... />*
287     <implementation ... />?
288
289 </componentType>

```

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

291

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

- 293 • **name** : **NCName (1..1)** - the name of the service. The @name attribute of a <service/> child element
294 of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
295 [ASM40003]
- 296 • **requires** : **listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
297 [SCA-POLICY] for a description of this attribute.
- 298 • **policySets** : **listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
299 [SCA-POLICY] for a description of this attribute.

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

- 301 • **interface** : **Interface (1..1)** - A service has **one interface**, which describes the operations provided
302 by the service. For details on the interface element see [the Interface section](#).
- 303 • **binding** : **Binding (0..n)** - A service element has **zero or more binding elements** as children. If the
304 binding element is not present it defaults to <binding.sca>. Details of the binding element are
305 described in [the Bindings section](#).
- 306 • **callback (0..1) / binding** : **Binding (1..n)** - A **callback** element is used if the interface has a callback
307 defined, and the callback element has one or more **binding** elements as subelements. The **callback**
308 and its binding subelements are specified if there is a need to have binding details used to handle
309 callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
310 For details on callbacks, see [the Bidirectional Interfaces section](#).
- 311 • **requires** : **requires (0..n)** - A service element has **zero or more requires subelements**. See the
312 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 313 • **policySetAttachment** : **policySetAttachment (0..n)** - A service element has **zero or more**
314 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
315 description of this element.

316 3.1.2 Reference

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

321

```

322 <?xml version="1.0" encoding="ASCII"?>
323 <!-- Component type reference schema snippet -->
324 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
325
326     <service ... />*
327
328     <reference name="xs:NCName"
329         autowire="xs:boolean"?
330         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
331         wiredByImpl="xs:boolean"? requires="list of xs:QName"?
332         policySets="list of xs:QName"?>*
333     <interface ... />
334     <binding ... />*
335     <callback?
336         <binding ... />+

```

```

337     </callback>
338     <requires/>*
339     <policySetAttachment/>*
340 </reference>
341
342 <property ... />*
343 <implementation ... />?
344
345 </componentType>

```

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

347

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

- 349 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a <reference/> child
350 element of a <componentType/> MUST be unique amongst the reference elements of that
351 <componentType/>. [ASM40004]
- 352 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to
353 target services. The multiplicity can have the following values
 - 354 – 0..1 – zero or one wire can have the reference as a source
 - 355 – 1..1 – one wire can have the reference as a source
 - 356 – 0..n - zero or more wires can have the reference as a source
 - 357 – 1..n – one or more wires can have the reference as a source
- 358 If @multiplicity is not specified, the default value is "1..1".
- 359 • **autowire : boolean (0..1)** - whether the reference is autowired, as described in [the Autowire section](#).
360 Default is false.
- 361 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the reference is
362 wired to the target(s) configured on the reference. If set to "true" it indicates that the target of the
363 reference is set at runtime by the implementation code (e.g. by the code obtaining an endpoint
364 reference by some means and setting this as the target of the reference through the use of
365 programming interfaces defined by the relevant Client and Implementation specification). If
366 @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be
367 ignored by the runtime. [ASM40006]
- 368 • **requires : listOfQNames (0..1)** - a list of policy intents. See the [Policy Framework specification](#)
369 [SCA-POLICY] for a description of this attribute.
- 370 • **policySets : listOfQNames (0..1)** - a list of policy sets. See the [Policy Framework specification](#)
371 [SCA-POLICY] for a description of this attribute.

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

- 373 • **interface : Interface (1..1)** - A reference has **one interface**, which describes the operations used by
374 the reference. The interface is described by an **interface element** which is a child element of the
375 reference element. For details on the interface element see [the Interface section](#).
- 376 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as children.
377 Details of the binding element are described in the [Bindings section](#).

378 When used with a reference element, a binding element specifies an endpoint which is the target of
379 that binding. A reference cannot mix the use of endpoints specified via binding elements with target
380 endpoints specified via the @target attribute. If the @target attribute is set, the reference cannot also
381 have binding subelements. If binding elements with endpoints are specified, each endpoint uses the
382 binding type of the binding element in which it is defined.

- 383 • **callback (0..1) / binding : Binding (1..n)** - a **callback** element is used if the interface has a callback
384 defined and the callback element has one or more **binding** elements as subelements. The **callback**
385 and its binding subelements are specified if there is a need to have binding details used to handle

386 callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
387 For details on callbacks, see [the Bidirectional Interfaces section](#).

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

393 For a full description of the setting of target service(s) for a reference, see the section "[Specifying the](#)
394 [Target Service\(s\) for a Reference](#)".

395 3.1.3 Property

396 **Properties** allow for the configuration of an implementation with externally set values. Each Property is
397 defined as a property element. The componentType element can have **zero or more property elements**
398 as its children. Snippet 3-4 shows the componentType pseudo-schema with the pseudo-schema for a
399 reference child element:

400

```
401 <?xml version="1.0" encoding="ASCII"?>  
402 <!-- Component type property schema snippet -->  
403 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
404  
405     <service ... />*  
406     <reference ... >*  
407  
408     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")  
409         many="xs:boolean"? mustSupply="xs:boolean"?>*  
410         default-property-value?  
411     </property>  
412  
413     <implementation ... />?  
414  
415 </componentType>
```

416 *Snippet 3-4: componentType Pseudo-Schema with property Child Element*

417

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

- 419 • **name : NCName (1..1)** - the name of the property. The @name attribute of a <property/> child
420 element of a <componentType/> MUST be unique amongst the property elements of that
421 <componentType/>. [\[ASM40005\]](#)
- 422 • one of (1..1):
 - 423 – **type : QName** - the type of the property defined as the qualified name of an XML schema type.
424 The value of the property @type attribute MUST be the QName of an XML schema type.
425 [\[ASM40007\]](#)
 - 426 – **element : QName** - the type of the property defined as the qualified name of an XML schema
427 global element – the type is the type of the global element. The value of the property @element
428 attribute MUST be the QName of an XSD global element. [\[ASM40008\]](#)

429 A single property element MUST NOT contain both a @type attribute and an @element attribute.
430 [\[ASM40010\]](#)

- 431 • **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). In the
432 case of a multi-valued property, it is presented to the implementation as a collection of property
433 values. If many is not specified, it takes a default value of false.
- 434 • **mustSupply : boolean (0..1)** - whether the property value needs to be supplied by the component
435 that uses the implementation. Default value is "false". When the componentType has

436 @mustSupply="true" for a property element, a component using the implementation MUST supply a
437 value for the property since the implementation has no default value for the property. [ASM40011] If
438 the implementation has a default-property-value then @mustSupply="false" is appropriate, since the
439 implication of a default value is that it is used when a value is not supplied by the using component.

- 440 • **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property. The value of
441 the property @file attribute MUST be a dereferencable URI to a file containing the value for the
442 property. [ASM40012] The URI can be an absolute URI or a relative URI. For a relative URI, it is
443 taken relative to the base of the contribution containing the implementation. For a description of the
444 format of the file, see the section on Property Value File Format.

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

447 The value for a property is supplied to the implementation of a component at the time that the
448 implementation is started. The implementation can use the supplied value in any way that it chooses. In
449 particular, the implementation can alter the internal value of the property at any time. However, if the
450 implementation queries the SCA system for the value of the property, the value as defined in the SCA
451 composite is the value returned.

452 The componentType property element can contain an SCA default value for the property declared by the
453 implementation. However, the implementation can have a property which has an implementation defined
454 default value, where the default value is not represented in the componentType. An example of such a
455 default value is where the default value is computed at runtime by some code contained in the
456 implementation. If a using component needs to control the value of a property used by an implementation,
457 the component sets the value explicitly. The SCA runtime MUST ensure that any implementation default
458 property value is replaced by a value for that property explicitly set by a component using that
459 implementation. [ASM40009]

460 3.1.4 Implementation

461 **Implementation** represents characteristics inherent to the implementation itself, in particular intents and
462 policies. See the [Policy Framework specification \[SCA-POLICY\]](#) for a description of intents and policies.
463 Snippet 3-5 shows the componentType pseudo-schema with the pseudo-schema for a implementation
464 child element:

465

```
466 <?xml version="1.0" encoding="ASCII"?>  
467 <!-- Component type implementation schema snippet -->  
468 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
469  
470   <service ... /> *  
471   <reference ... > *  
472   <property ... /> *  
473  
474   <implementation requires="list of xs:QName"?  
475     policySets="list of xs:QName"?>  
476     <requires/> *  
477     <policySetAttachment/> *  
478   </implementation?>  
479  
480 </componentType>
```

481 *Snippet 3-5: componentType Pseudo-Schema with implementation Child Element*

482

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

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

487

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

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

494 3.2 Example ComponentType

495 Snippet 3-6 shows the contents of the componentType file for the MyValueServiceImpl implementation.
496 The componentType file shows the services, references, and properties of the MyValueServiceImpl
497 implementation. In this case, Java is used to define interfaces:

```
498 <?xml version="1.0" encoding="ASCII"?>
499 <componentType xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200912
500     xmlns:xsd="http://www.w3.org/2001/XMLSchema">
501
502     <service name="MyValueService">
503         <interface.java interface="services.myvalue.MyValueService"/>
504     </service>
505
506     <reference name="customerService">
507         <interface.java interface="services.customer.CustomerService"/>
508     </reference>
509     <reference name="stockQuoteService">
510         <interface.java
511             interface="services.stockquote.StockQuoteService"/>
512     </reference>
513
514     <property name="currency" type="xsd:string">USD</property>
515
516 </componentType>
```

517 *Snippet 3-6: Example componentType*

518 3.3 Example Implementation

519 Snippet 3-7 and Snippet 3-8 are an example implementation, written in Java.

520 **AccountServiceImpl** implements the **AccountService** interface, which is defined via a Java interface:

```
521 package services.account;
522
523 @Remotable
524 public interface AccountService {
525
526     AccountReport getAccountReport(String customerID);
527 }
```

528 *Snippet 3-7: Example Interface in Java*

529

530 Snippet 3-8 is a full listing of the AccountServiceImpl class, showing the Service it implements, plus the
531 service references it makes and the settable properties that it has. Notice the use of Java annotations to
532 mark SCA aspects of the code, including the @Property, @Reference and @Service annotations:

```
533 package services.account;
534
535 import java.util.List;
536
537 import commonj.sdo.DataFactory;
538
539 import org.oasisopen.sca.annotation.Property;
540 import org.oasisopen.sca.annotation.Reference;
541 import org.oasisopen.sca.annotation.Service;
```

```

542
543 import services.accountdata.AccountDataService;
544 import services.accountdata.CheckingAccount;
545 import services.accountdata.SavingsAccount;
546 import services.accountdata.StockAccount;
547 import services.stockquote.StockQuoteService;
548
549 @Service(AccountService.class)
550 public class AccountServiceImpl implements AccountService {
551
552     @Property
553     private String currency = "USD";
554
555     @Reference
556     private AccountDataService accountDataService;
557     @Reference
558     private StockQuoteService stockQuoteService;
559
560     public AccountReport getAccountReport(String customerID) {
561
562         DataFactory dataFactory = DataFactory.INSTANCE;
563         AccountReport accountReport =
564             (AccountReport) dataFactory.create(AccountReport.class);
565         List accountSummaries = accountReport.getAccountSummaries();
566
567         CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
568         AccountSummary checkingAccountSummary =
569             (AccountSummary) dataFactory.create(AccountSummary.class);
570         checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
571         checkingAccountSummary.setAccountType("checking");
572
573         checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
574         accountSummaries.add(checkingAccountSummary);
575
576         SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
577         AccountSummary savingsAccountSummary =
578             (AccountSummary) dataFactory.create(AccountSummary.class);
579         savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
580         savingsAccountSummary.setAccountType("savings");
581
582         savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
583         accountSummaries.add(savingsAccountSummary);
584
585         StockAccount stockAccount = accountDataService.getStockAccount(customerID);
586         AccountSummary stockAccountSummary =
587             (AccountSummary) dataFactory.create(AccountSummary.class);
588         stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
589         stockAccountSummary.setAccountType("stock");
590         float balance =
591
592         (stockQuoteService.getQuote(stockAccount.getSymbol())) * stockAccount.getQuantity();
593         stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
594         accountSummaries.add(stockAccountSummary);
595
596         return accountReport;
597     }
598
599     private float fromUSDollarToCurrency(float value) {
600
601         if (currency.equals("USD")) return value; else
602         if (currency.equals("EURO")) return value * 0.8f; else
603         return 0.0f;
604     }
605 }

```

606 *Snippet 3-8: Example Component Implementation in Java*

607
608 The following is the SCA componentType definition for the AccountServiceImpl, derived by introspection
609 of the code above:

```

610 <?xml version="1.0" encoding="ASCII"?>

```

```
611 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
612             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
613
614     <service name="AccountService">
615         <interface.java interface="services.account.AccountService"/>
616     </service>
617     <reference name="accountDataService">
618         <interface.java
619             interface="services.accountdata.AccountDataService"/>
620     </reference>
621     <reference name="stockQuoteService">
622         <interface.java
623             interface="services.stockquote.StockQuoteService"/>
624     </reference>
625
626     <property name="currency" type="xsd:string"/>
627
628 </componentType>
```

629 *Snippet 3-9: Example componentType for Implementation in Snippet 3-8*

630

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

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

636 4 Component

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

639 **Components** are configured **instances** of **implementations**. Components provide and consume
640 services. More than one component can use and configure the same implementation, where each
641 component configures the implementation differently.

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

646

```
647 <?xml version="1.0" encoding="UTF-8"?>
648 <!-- Component schema snippet -->
649 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
650   ...
651   <component name="xs:NCName" autowire="xs:boolean"?
652     requires="list of xs:QName"? policySets="list of xs:QName"?>*
653     <implementation ... />?
654     <service ... />*
655     <reference ... />*
656     <property ... />*
657     <requires/>*
658     <policySetAttachment/>*
659   </component>
660   ...
661 </composite>
```

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

663

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

- 665 • **name** : **NCName (1..1)** – the name of the component. The @name attribute of a **<component/>** child
666 element of a **<composite/>** MUST be unique amongst the component elements of that **<composite/>**
667 **[ASM50001]**
- 668 • **autowire** : **boolean (0..1)** – whether contained component references are autowired, as described in
669 **the Autowire section**. Default is false.
- 670 • **requires** : **listOfQNames (0..1)** – a list of policy intents. See the **Policy Framework specification**
671 **[SCA-POLICY]** for a description of this attribute.
- 672 • **policySets** : **listOfQNames (0..1)** – a list of policy sets. See the **Policy Framework specification**
673 **[SCA-POLICY]** for a description of this attribute.

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

- 675 • **implementation** : **ComponentImplementation (0..1)** – see **component implementation section**.
- 676 • **service** : **ComponentService (0..n)** – see **component service section**.
- 677 • **reference** : **ComponentReference (0..n)** – see **component reference section**.
- 678 • **property** : **ComponentProperty (0..n)** – see **component property section**.
- 679 • **requires** : **requires (0..n)** - A service element has **zero or more requires subelements**. See the
680 **Policy Framework specification [SCA-POLICY]** for a description of this element.

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

684 4.1 Implementation

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

```
687 <?xml version="1.0" encoding="UTF-8"?>  
688 <!-- Component Implementation schema snippet -->  
689 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >  
690 ...  
691 <component ... >*<br>692 <implementation requires="list of xs:QName"?<br>693 <policySets="list of xs:QName"?><br>694 <requires/>*<br>695 <policySetAttachment/>*<br>696 </implementation><br>697 <service ... />*<br>698 <reference ... />*<br>699 <property ... />*<br>700 </component><br>701 ...<br>702 </composite>
```

703 *Snippet 4-2: component Psuedo-Schema with implementation Child Element*

704

705 The component provides the extensibility point in the assembly model for different implementation types.
706 The references to implementations of different types are expressed by implementation type specific
707 implementation elements.

708 For example the elements **implementation.java**, **implementation.bpel**, **implementation.cpp**, and
709 **implementation.c** point to Java, BPEL, C++, and C implementation types respectively.

710 **implementation.composite** points to the use of an SCA composite as an implementation.

711 **implementation.spring** and **implementation.ejb** are used for Java components written to the Spring
712 framework and the Java EE EJB technology respectively.

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

715

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

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

718

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

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

721

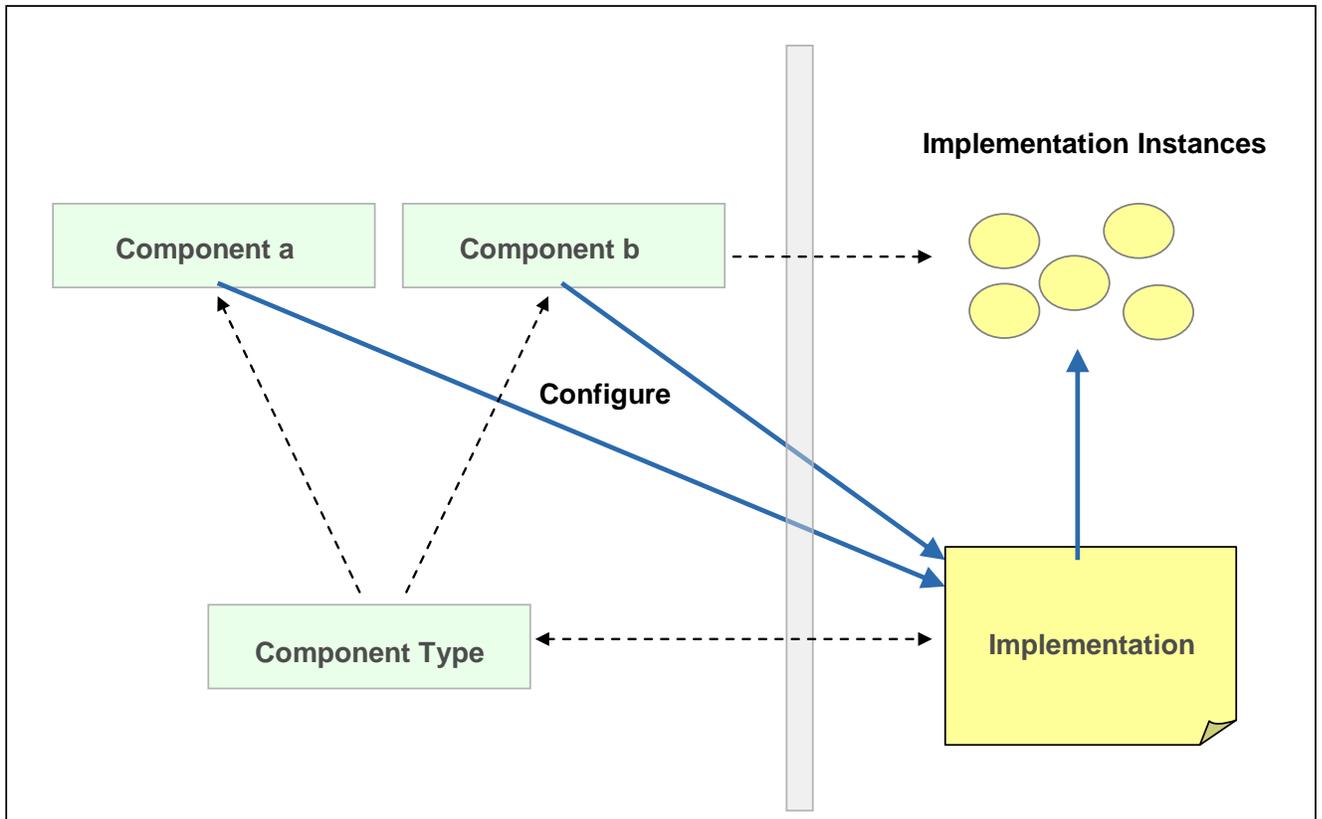
```
722 <implementation.composite name="bns:MyValueComposite"/>
```

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

724

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

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



730
731 *Figure 4-1: Relationship of Component and Implementation*

732 4.2 Service

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

737

```

738 <?xml version="1.0" encoding="UTF-8"?>
739 <!-- Component Service schema snippet -->
740 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
741 ...
742 <component ... >*
743 <implementation ... />
744 <service name="xs:NCName" requires="list of xs:QName"?
745 <policySets="list of xs:QName"?>*
746 <interface ... />?
747 <binding ... />*
748 <callback?>
749 <binding ... />+
750 </callback>
751 <requires/>*
752 <policySetAttachment/>*
753 </service>
754 <reference ... />*
755 <property ... />*
756 </component>
757 ...
758 </composite>

```

760

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

- 762 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service element of a
763 <component/> MUST be unique amongst the service elements of that <component/> [ASM50002]
764 The @name attribute of a service element of a <component/> MUST match the @name attribute of a
765 service element of the componentType of the <implementation/> child element of the component.
766 [ASM50003]
- 767 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
768 [SCA-POLICY] for a description of this attribute.
769 Note: The effective set of policy intents for the service consists of any intents explicitly stated in this
770 @requires attribute, combined with any intents specified for the service by the implementation.
- 771 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
772 [SCA-POLICY] for a description of this attribute.

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

- 774 • **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the operations
775 provided by the service. The interface is described by an **interface element** which is a child element
776 of the service element. If no interface is specified, then the interface specified for the service in the
777 componentType of the implementation is in effect. If an interface is declared for a component service,
778 the interface MUST provide a compatible subset of the interface declared for the equivalent service in
779 the componentType of the implementation [ASM50004] For details on the interface element see [the](#)
780 [Interface section](#).
- 781 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as children. If no
782 binding elements are specified for the service, then the bindings specified for the equivalent service in
783 the componentType of the implementation MUST be used, but if the componentType also has no
784 bindings specified, then <binding.sca/> MUST be used as the binding. If binding elements are
785 specified for the service, then those bindings MUST be used and they override any bindings specified
786 for the equivalent service in the componentType of the implementation. [ASM50005] Details of the
787 binding element are described in [the Bindings section](#). The binding, combined with any PolicySets in
788 effect for the binding, needs to satisfy the set of policy intents for the service, as described in [the](#)
789 [Policy Framework specification \[SCA-POLICY\]](#).
- 790 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
791 defined and the callback element has one or more **binding** elements as subelements. The **callback**
792 and its binding subelements are specified if there is a need to have binding details used to handle
793 callbacks. If the callback element is present and contains one or more binding child elements, then
794 those bindings MUST be used for the callback. [ASM50006] If the callback element is not present, the
795 behaviour is runtime implementation dependent.
- 796 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
797 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 798 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
799 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
800 description of this element.

801

4.3 Reference

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

806

```
807 <?xml version="1.0" encoding="UTF-8"?>
808 <!-- Component Reference schema snippet -->
```

```

809 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
810 ...
811 <component ... >*
812 <implementation ... />
813 <service ... />*
814 <reference name="xs:NCName"
815     target="list of xs:anyURI"? autowire="xs:boolean"?
816     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
817     nonOverridable="xs:boolean"
818     wiredByImpl="xs:boolean"? requires="list of xs:QName"?
819     policySets="list of xs:QName"?>*
820 <interface ... />?
821 <binding uri="xs:anyURI"? requires="list of xs:QName"?
822     policySets="list of xs:QName"?/>*
823 <callback>?
824     <binding ... />+
825 </callback>
826 <requires/>*
827 <policySetAttachment/>*
828 </reference>
829 <property ... />*
830 </component>
831 ...
832 </composite>

```

833 *Snippet 4-7: component Psuedo-Schema with reference Child Element*

834

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

- 836 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service element of a
837 <component/> MUST be unique amongst the service elements of that <component/> [ASM50007]
838 The @name attribute of a reference element of a <component/> MUST match the @name attribute of
839 a reference element of the componentType of the <implementation/> child element of the component.
840 [ASM50008]
- 841 • **autowire : boolean (0..1)** – whether the reference is autowired, as described in the [Autowire section](#).
842 The default value of the @autowire attribute MUST be the value of the @autowire attribute on the
843 component containing the reference, if present, or else the value of the @autowire attribute of the
844 composite containing the component, if present, and if neither is present, then it is "false".
845 [ASM50043]
- 846 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
847 [SCA-POLICY] for a description of this attribute.
848 Note: The effective set of policy intents for the reference consists of any intents explicitly stated in this
849 @requires attribute, combined with any intents specified for the reference by the implementation.
- 850 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
851 [SCA-POLICY] for a description of this attribute.
- 852 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to
853 target services. Overrides the multiplicity specified for this reference in the componentType of the
854 implementation. The multiplicity can have the following values
 - 855 – 0..1 – zero or one wire can have the reference as a source
 - 856 – 1..1 – one wire can have the reference as a source
 - 857 – 0..n - zero or more wires can have the reference as a source
 - 858 – 1..n – one or more wires can have the reference as a source

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

862 If not present, the value of multiplicity is equal to the multiplicity specified for this reference in the
863 componentType of the implementation - if not present in the componentType, the value defaults to
864 1..1.

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

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

876 • **nonOverridable : boolean (0..1)** - a boolean value, "false" by default, which indicates whether this
877 component reference can have its targets overridden by a composite reference which promotes the
878 component reference.
879 If @nonOverridable==false, if any target(s) are configured onto the composite references which
880 promote the component reference, then those targets **replace** all the targets explicitly declared on the
881 component reference for any value of @multiplicity on the component reference. If no targets are
882 defined on any of the composite references which promote the component reference, then any
883 targets explicitly declared on the component reference are used. This means in effect that any targets
884 declared on the component reference act as default targets for that reference.
885

886 If a component reference has @multiplicity 0..1 or 1..1 and @nonOverridable==true, then the
887 component reference MUST NOT be promoted by any composite reference. [ASM50042]
888

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

892 The component reference element has the child elements:

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

900 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as children. If no
901 binding elements are specified for the reference, then the bindings specified for the equivalent
902 reference in the componentType of the implementation MUST be used. If binding elements are
903 specified for the reference, then those bindings MUST be used and they override any bindings
904 specified for the equivalent reference in the componentType of the implementation. [ASM50012] It is
905 valid for there to be no binding elements on the component reference and none on the reference in
906 the componentType - the binding used for such a reference is determined by the target service. See
907 the [section on the bindings of component services](#) for a description of how the binding(s) applying to
908 a service are determined.

909 Details of the binding element are described in the [Bindings section](#). The binding, combined with any
910 PolicySets in effect for the binding, needs to satisfy the set of policy intents for the reference, as
911 described in the [Policy Framework specification \[SCA-POLICY\]](#).

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

- 915 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element used if the interface has a callback
916 defined and the callback element has one or more **binding** elements as subelements. The **callback**
917 and its binding subelements are specified if there is a need to have binding details used to handle
918 callbacks. If the callback element is present and contains one or more binding child elements, then
919 those bindings MUST be used for the callback. [ASM50006] If the callback element is not present,
920 the behaviour is runtime implementation dependent.
- 921 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
922 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 923 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
924 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
925 description of this element.

926 4.3.1 Specifying the Target Service(s) for a Reference

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

- 929 1. Through a value specified in the @target attribute of the reference element
- 930 2. Through a target URI specified in the @uri attribute of a binding element which is
931 a child of the reference element
- 932 3. Through the setting of one or more values for binding-specific attributes and/or
933 child elements of a binding element that is a child of the reference element
- 934 4. Through the specification of @autowire="true" for the reference (or through
935 inheritance of that value from the component or composite containing the
936 reference)
- 937 5. Through the specification of @wiredByImpl="true" for the reference
- 938 6. Through the promotion of a component reference by a composite reference of the
939 composite containing the component (the target service is then identified by the
940 configuration of the composite reference)
- 941 7. Through the presence of a <wire/> element which has the reference specified in
942 its @source attribute.

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

- 944 • If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.
945 [ASM50013]
- 946 • If @autowire="true", the autowire procedure MUST only be used if no target is identified by any of the
947 other ways listed above. It is not an error if @autowire="true" and a target is also defined through
948 some other means, however in this case the autowire procedure MUST NOT be used. [ASM50014]
- 949 • If a reference has a value specified for one or more target services in its @target attribute, there
950 MUST NOT be any child <binding/> elements declared for that reference. [ASM50026]
- 951 • If a binding element has a value specified for a target service using its @uri attribute, the binding
952 element MUST NOT identify target services using binding specific attributes or elements.
953 [ASM50015]
- 954 • It is possible that a particular binding type uses more than a simple URI for the address of a target
955 service. In cases where a reference element has a binding subelement that uses more than simple
956 URI, the @uri attribute of the binding element MUST NOT be used to identify the target service - in
957 this case binding specific attributes and/or child elements MUST be used. [ASM50016]
- 958 • If any <wire/> element with its @replace attribute set to "true" has a particular reference specified in
959 its @source attribute, the value of the @target attribute for that reference MUST be ignored and
960 MUST NOT be used to define target services for that reference. [ASM50034]

961 4.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

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

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

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

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

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

984 4.4 Property

985 The component element has **zero or more property elements** as its children, which are used to
986 configure data values of properties of the implementation. Each property element provides a value for the
987 named property, which is passed to the implementation. The properties that can be configured and their
988 types are defined by the component type of the implementation. An implementation can declare a
989 property as multi-valued, in which case, multiple property values can be present for a given property.

990 The property value can be specified in **one** of five ways:

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

992 If the @value attribute of a component property element is declared, the type of the property MUST
993 be an XML Schema simple type and the @value attribute MUST contain a single value of that type.
994 [ASM50027]

995 For example,

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

997 *Snippet 4-8: Example property using @value attribute*

998

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

1000 If the value subelement of a component property is specified, the type of the property MUST be an
1001 XML Schema simple type or an XML schema complex type. [ASM50028]

1002 For example,

- 1003 – property defined using a XML Schema simple type and which contains a single value

```
1004 <property name="pi">  
1005 <value>3.14159265</value>  
1006 </property>
```

1007 *Snippet 4-9: Example property with a Simple Type Containing a Single Value*

1008

1009 – property defined using a XML Schema simple type and which contains multiple values

```
1010 <property name="currency">
1011   <value>EURO</value>
1012   <value>USDollar</value>
1013 </property>
```

1014 *Snippet 4-10: Example property with a Simple Type Containing Multiple Values*

1015

1016 – property defined using a XML Schema complex type and which contains a single value

```
1017 <property name="complexFoo">
1018   <value attr="bar">
1019     <foo:a>TheValue</foo:a>
1020     <foo:b>InterestingURI</foo:b>
1021   </value>
1022 </property>
```

1023 *Snippet 4-11: Example property with a Complex Type Containing a Single Value*

1024

1025 – property defined using a XML Schema complex type and which contains multiple values

```
1026 <property name="complexBar">
1027   <value anotherAttr="foo">
1028     <bar:a>AValue</bar:a>
1029     <bar:b>InterestingURI</bar:b>
1030   </value>
1031   <value attr="zing">
1032     <bar:a>BValue</bar:a>
1033     <bar:b>BoringURI</bar:b>
1034   </value>
1035 </property>
```

1036 *Snippet 4-12: Example property with a Complex Type Containing Multiple Values*

1037

1038 • As a value, supplied as the content of the property element.

1039 If a component property value is declared using a child element of the <property/> element, the type
1040 of the property MUST be an XML Schema global element and the declared child element MUST be
1041 an instance of that global element. [\[ASM50029\]](#)

1042 For example,

1043 – property defined using a XML Schema global element declaration and which contains a single
1044 value

```
1045 <property name="foo">
1046   <foo:SomeGED ...>...</foo:SomeGED>
1047 </property>
```

1048 *Snippet 4-13: Example property with a Global Element Declaration Containing a Single Value*

1049

1050 – property defined using a XML Schema global element declaration and which contains multiple
1051 values

```
1052 <property name="bar">
1053   <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1054   <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1055 </property>
```

1056 *Snippet 4-14 Example property with a Global Element Declaration Containing Multiple Values*

1057

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

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

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

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

1069

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

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

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

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

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

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. [ASM50036]

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

Snippet 4-15 shows the component pseudo-schema with the pseudo-schema for a property child element:

1087

```
<?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>
```

Snippet 4-15: component Pseudo-Schema with property Child Element

1107

- 1108 The **component property** element has the **attributes**:
- 1109 • **name : NCName (1..1)** – the name of the property. The @name attribute of a property element of a
1110 <component/> MUST be unique amongst the property elements of that <component/>. [ASM50031]
1111 The @name attribute of a property element of a <component/> MUST match the @name attribute of
1112 a property element of the componentType of the <implementation/> child element of the component.
1113 [ASM50037]
 - 1114 • zero or one of (0..1):
 - 1115 – **type : QName** – the type of the property defined as the qualified name of an XML schema type
 - 1116 – **element : QName** – the type of the property defined as the qualified name of an XML schema
1117 global element – the type is the type of the global element
 - 1118 A single property element MUST NOT contain both a @type attribute and an @element attribute.
1119 [ASM50035]
 - 1120 • **source : string (0..1)** – an XPath expression pointing to a property of the containing composite from
1121 which the value of this component property is obtained.
 - 1122 • **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property. The value of
1123 the component property @file attribute MUST be a dereferencable URI to a file containing the value
1124 for the property. [ASM50045] The URI can be an absolute URI or a relative URI. For a relative URI, it
1125 is taken relative to the base of the contribution containing the composite in which the component is
1126 declared. For a description of the format of the file, see the [section on Property Value File Format](#).
 - 1127 • **many : boolean (0..1)** – whether the property is single-valued (false) or multi-valued (true).
1128 Overrides the many specified for this property in the componentType of the implementation. The
1129 value can only be equal or further restrict, i.e. if the implementation specifies many true, then the
1130 component can say false. In the case of a multi-valued property, it is presented to the implementation
1131 as a Collection of property values. If many is not specified, it takes the value defined by the
1132 component type of the implementation used by the component.
 - 1133 • **value : string (0..1)** - the value of the property if the property is defined using a simple type.
- 1134 The **component property** element has the **child element**:
- 1135 • **value : any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a
1136 property that is defined using a XML Schema type. If a property is single-valued, the <value/>
1137 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST
1138 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

1139 4.4.1 Property Type Compatibility

1140 There are a number of situations where the declared type of a property element is matched with the
1141 declared type of another property element. These situations include:

- 1142 • Where a component <property/> sets a value for a property of an implementation, as declared in the
1143 componentType of the implementation
- 1144 • Where a component <property/> gets its value from the value of a composite <property/> by means
1145 of its @source attribute. This situation can also involve the @source attribute referencing a
1146 subelement of the composite <property/> value, in which case it is the type of the subelement which
1147 must be matched with the type of the component <property/>
- 1148 • Where the componentType of a composite used as an implementation is calculated and
1149 componentType <property/> elements are created for each composite <property/>

1150 In these cases where the types of two property elements are matched, the types declared for the two
1151 <property/> elements MUST be compatible [ASM50038]

1152 Two property types are compatible if they have the same XSD type (where declared as XSD types) or the
1153 same XSD global element (where declared as XSD global elements). For cases where the type of a
1154 property is declared using a different type system (eg Java), then the type of the property is mapped to
1155 XSD using the mapping rules defined by the appropriate implementation type specification

1156 4.4.2 Property Value File Format

1157 The format of the file which is referenced by the @file attribute of a component property or a
1158 componentType property is that it is an XML document which MUST contain an sca:values element which
1159 in turn contains one of:

- 1160 • a set of one or more <sca:value/> elements each containing a simple string - where the property
1161 type is a simple XML type
- 1162 • a set of one or more <sca:value/> elements or a set of one or more global elements - where the
1163 property type is a complex XML type

1164 [ASM50046]

1165

```
1166 <?xml version="1.0" encoding="UTF-8"?>  
1167 <values>  
1168   <value>MyValue</value>  
1169 </values>
```

1170 *Snippet 4-16: Property Value File Content for simple property type*

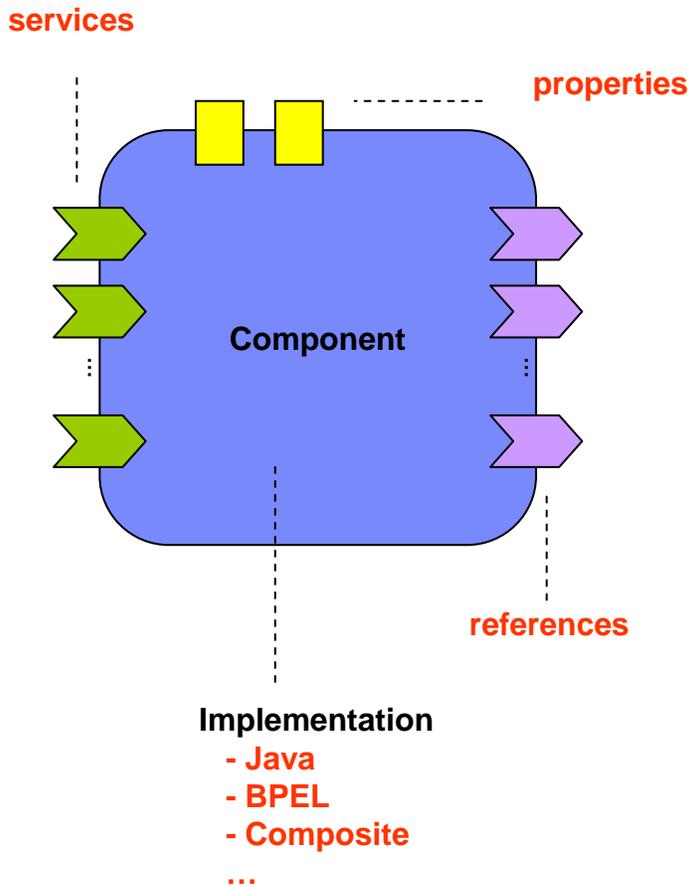
1171

```
1172 <?xml version="1.0" encoding="UTF-8"?>  
1173 <values>  
1174   <foo:fooElement>  
1175     <foo:a>AValue</foo:a>  
1176     <foo:b>InterestingURI</foo:b>  
1177   </foo:fooElement>  
1178 </values/>
```

1179 *Snippet 4-17: Property Value File Content for a complex property type*

1180 4.5 Example Component

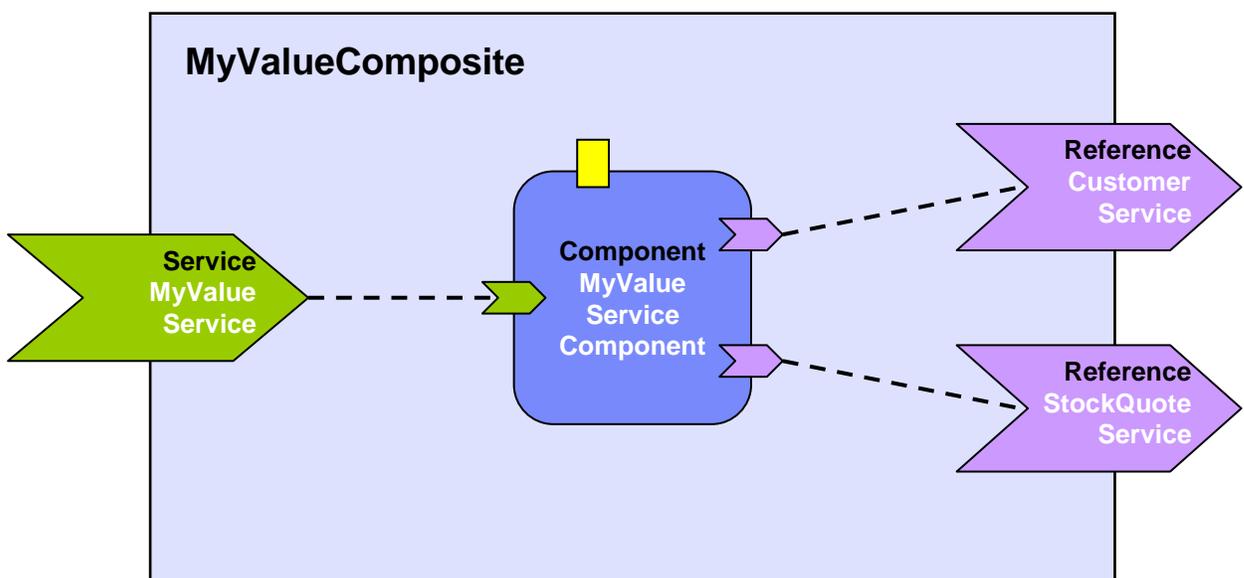
1181 Figure 4-2 shows the **component symbol** that is used to represent a component in an assembly
1182 diagram.



1183
 1184
 1185
 1186
 1187

Figure 4-2: Component symbol

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



1188
 1189
 1190

Figure 4-3: Assembly diagram for MyValueComposite

1191 Snippet 4-18: Example composite shows the MyValueComposite.composite file for the
1192 MyValueComposite containing the component element for the MyValueServiceComponent. A value
1193 is set for the property named currency, and the customerService and stockQuoteService
1194 references are promoted:

```
1195 <?xml version="1.0" encoding="ASCII"?>
1196 <!-- MyValueComposite_1 example -->
1197 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1198           targetNamespace="http://foo.com"
1199           name="MyValueComposite" >
1200
1201     <service name="MyValueService" promote="MyValueServiceComponent"/>
1202
1203     <component name="MyValueServiceComponent">
1204       <implementation.java
1205         class="services.myvalue.MyValueServiceImpl"/>
1206       <property name="currency">EURO</property>
1207       <reference name="customerService"/>
1208       <reference name="stockQuoteService"/>
1209     </component>
1210
1211     <reference name="CustomerService"
1212       promote="MyValueServiceComponent/customerService"/>
1213
1214     <reference name="StockQuoteService"
1215       promote="MyValueServiceComponent/stockQuoteService"/>
1216
1217 </composite>
```

1218 *Snippet 4-18: Example composite*

1219
1220 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of clarity
1221 – the references are defined by the MyValueServiceImpl implementation and there is no need to
1222 redeclare them on the component unless the intention is to wire them or to override some aspect of them.
1223 The following snippet gives an example of the layout of a composite file if both the currency property and
1224 the customerService reference of the MyValueServiceComponent are declared to be multi-valued
1225 (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1226 <?xml version="1.0" encoding="ASCII"?>
1227 <!-- MyValueComposite_2 example -->
1228 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1229           targetNamespace="http://foo.com"
1230           name="MyValueComposite" >
1231
1232     <service name="MyValueService" promote="MyValueServiceComponent"/>
1233
1234     <component name="MyValueServiceComponent">
1235       <implementation.java
1236         class="services.myvalue.MyValueServiceImpl"/>
1237       <property name="currency">
1238         <value>EURO</value>
1239         <value>Yen</value>
1240         <value>USDollar</value>
1241       </property>
1242       <reference name="customerService"
1243         target="InternalCustomer/customerService"/>
1244       <reference name="stockQuoteService"/>
1245     </component>
1246
1247     ...
1248
1249     <reference name="CustomerService"
1250       promote="MyValueServiceComponent/customerService"/>
```

1251
1252
1253
1254
1255

```
<reference name="StockQuoteService"
  promote="MyValueServiceComponent/stockQuoteService"/>
</composite>
```

1256 *Snippet 4-19: Example composite with Multi-Valued property and reference*

1257

1258 ...this assumes that the composite has another component called InternalCustomer (not shown) which
1259 has a service to which the customerService reference of the MyValueServiceComponent is wired as well
1260 as being promoted externally through the composite reference CustomerService.

5 Composite

1261

1262 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of
1263 composition within an SCA Domain. An **SCA composite** contains a set of components, services,
1264 references and the wires that interconnect them, plus a set of properties which can be used to configure
1265 components.

1266 Composites can be used as **component implementations** in higher-level composites – in other words
1267 the higher-level composites can have components that are implemented by composites. For more detail
1268 on the use of composites as component implementations see the section [Using Composites as](#)
1269 [Component Implementations](#).

1270 The content of a composite can be used within another composite through **inclusion**. When a composite
1271 is included by another composite, all of its contents are made available for use within the including
1272 composite – the contents are fully visible and can be referenced by other elements within the including
1273 composite. For more detail on the inclusion of one composite into another see the section [Using](#)
1274 [Composites through Inclusion](#).

1275 A composite can be used as a unit of deployment. When used in this way, composites contribute
1276 components and wires to an SCA Domain. A composite can be deployed to the SCA Domain either by
1277 inclusion or a composite can be deployed to the Domain as an implementation. For more detail on the
1278 deployment of composites, see the section dealing with the [SCA Domain](#).

1279 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite** element.
1280 Snippet 5-1 shows the pseudo-schema for the composite element:

1281

```
1282 <?xml version="1.0" encoding="ASCII"?>  
1283 <!-- Composite schema snippet -->  
1284 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
1285     targetNamespace="xs:anyURI"  
1286     name="xs:NCName" local="xs:boolean"?  
1287     autowire="xs:boolean"?  
1288     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1289  
1290     <include ... />*  
1291  
1292     <requires/>*  
1293     <policySetAttachment/>*  
1294  
1295     <service ... />*  
1296     <reference ... />*  
1297     <property ... />*  
1298  
1299     <component ... />*  
1300  
1301     <wire ... />*  
1302  
1303 </composite>
```

1304 *Snippet 5-1: composite Pseudo-Schema*

1305

1306 The **composite** element has the **attributes**:

- 1307 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is an XML
1308 QName, in the namespace identified by the @targetNamespace attribute. A composite @name
1309 attribute value MUST be unique within the namespace of the composite. [ASM60001]
- 1310 • **targetNamespace : anyURI (1..1)** – an identifier for a target namespace into which the composite is
1311 declared

- 1312 • **local : boolean (0..1)** – whether all the components within the composite all run in the same
1313 operating system process. @local="true" for a composite means that all the components within the
1314 composite MUST run in the same operating system process. [ASM60002] local="false", which is the
1315 default, means that different components within the composite can run in different operating system
1316 processes and they can even run on different nodes on a network.
- 1317 • **autowire : boolean (0..1)** – whether contained component references are autowired, as described in
1318 the [Autowire section](#). Default is false.
- 1319 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1320 [\[SCA-POLICY\]](#) for a description of this attribute.
- 1321 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1322 [\[SCA-POLICY\]](#) for a description of this attribute.

1323 The **composite** element has the **child elements**:

- 1324 • **service : CompositeService (0..n)** – see composite service section.
- 1325 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1326 • **property : CompositeProperty (0..n)** – see composite property section.
- 1327 • **component : Component (0..n)** – see component section.
- 1328 • **wire : Wire (0..n)** – see composite wire section.
- 1329 • **include : Include (0..n)** – see composite include section
- 1330 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1331 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1332 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1333 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
1334 description of this element.

1335 Components contain configured implementations which hold the business logic of the composite. The
1336 components offer services and use references to other services. **Composite services** define the public
1337 services provided by the composite, which can be accessed from outside the composite. **Composite**
1338 **references** represent dependencies which the composite has on services provided elsewhere, outside
1339 the composite. Wires describe the connections between component services and component references
1340 within the composite. Included composites contribute the elements they contain to the using composite.

1341 Composite services involve the **promotion** of one service of one of the components within the composite,
1342 which means that the composite service is actually provided by one of the components within the
1343 composite. Composite references involve the **promotion** of one or more references of one or more
1344 components. Multiple component references can be promoted to the same composite reference, as long
1345 as each of the component references has an interface that is a compatible subset of the interface on the
1346 composite reference. Where multiple component references are promoted to the same composite
1347 reference, then they all share the same configuration, including the same target service(s).

1348 Composite services and composite references can use the configuration of their promoted services and
1349 references respectively (such as Bindings and Policy Sets). Alternatively composite services and
1350 composite references can override some or all of the configuration of the promoted services and
1351 references, through the configuration of bindings and other aspects of the composite service or reference.

1352 Component services and component references can be promoted to composite services and references
1353 and also be wired internally within the composite at the same time. For a reference, this only makes
1354 sense if the reference supports a multiplicity greater than 1.

1355 5.1 Service

1356 The **services of a composite** are defined by promoting services defined by components contained in the
1357 composite. A component service is promoted by means of a composite **service element**.

1358 A composite service is represented by a **service element** which is a child of the composite element.
1359 There can be **zero or more** service elements in a composite. Snippet 5-2 shows the composite pseudo-
1360 schema with the pseudo-schema for a service child element:

1361

```
1362 <?xml version="1.0" encoding="ASCII"?>
1363 <!-- Composite Service schema snippet -->
1364 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1365 ...
1366   <service name="xs:NCName" promote="xs:anyURI"
1367     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1368     <interface ... />?
1369     <binding ... />*
1370     <callback?>
1371       <binding ... />+
1372     </callback>
1373     <requires/>*
1374     <policySetAttachment/>*
1375   </service>
1376   ...
1377 </composite>
```

1378 *Snippet 5-2: composite Pseudo-Schema with service Child Element*

1379

1380 The **composite service** element has the **attributes**:

- 1381 • **name : NCName (1..1)** – the name of the service. The name of a composite `<service/>` element
1382 **MUST be unique across all the composite services in the composite.** [ASM60003] The name of the
1383 composite service can be different from the name of the promoted component service.
- 1384 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form `<component-`
1385 `name>/<service-name>`. The service name can be omitted if the target component only has one
1386 service. The same component service can be promoted by more than one composite service. A
1387 composite `<service/>` element's `@promote` attribute **MUST** identify one of the component services
1388 within that composite. [ASM60004] `<include/>` processing **MUST** take place before the processing of
1389 the `@promote` attribute of a composite service is performed. [ASM60038]
- 1390 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
1391 [\[SCA-POLICY\]](#) for a description of this attribute. Specified intents add to or further qualify the required
1392 intents defined by the promoted component service.
- 1393 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
1394 [\[SCA-POLICY\]](#) for a description of this attribute.

1395 The **composite service** element has the **child elements**, whatever is not specified is defaulted from the
1396 promoted component service.

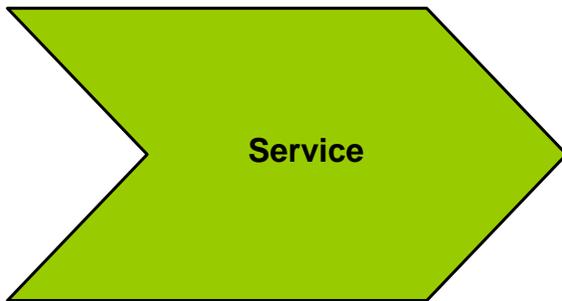
- 1397 • **interface : Interface (0..1)** - an interface which describes the operations provided by the composite
1398 service. If a composite service interface is specified it **MUST** be the same or a compatible subset of
1399 the interface provided by the promoted component service. [ASM60005] The interface is described by
1400 **zero or one interface element** which is a child element of the service element. For details on the
1401 interface element see [the Interface section](#).
- 1402 • **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the
1403 promoted component service from the composite service perspective. The bindings defined on the
1404 component service are still in effect for local wires within the composite that target the component
1405 service. A service element has zero or more **binding elements** as children. Details of the binding
1406 element are described in the [Bindings section](#). For more details on wiring see [the Wiring section](#).
- 1407 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
1408 defined and the callback has one or more **binding** elements as subelements. The **callback** and its
1409 binding subelements are specified if there is a need to have binding details used to handle callbacks.
1410 Callback binding elements attached to the composite service override any callback binding elements

1411 defined on the promoted component service. If the callback element is not present on the composite
1412 service, any callback binding elements on the promoted service are used. If the callback element is
1413 not present at all, the behaviour is runtime implementation dependent.

- 1414 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1415 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 1416 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1417 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
1418 description of this element.

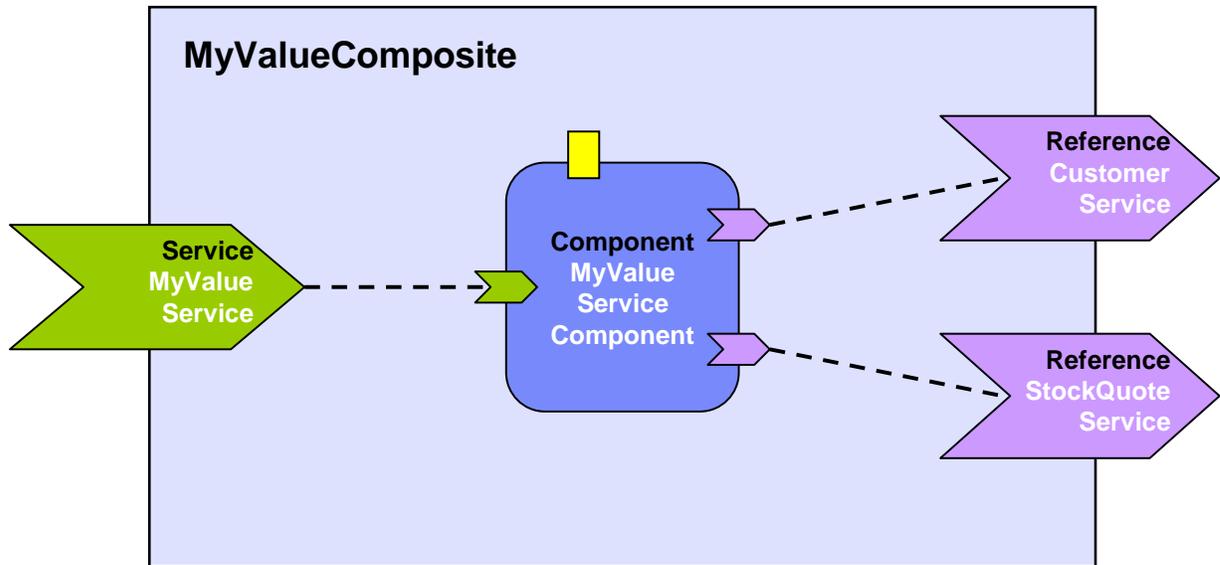
1419 5.1.1 Service Examples

1420 Figure 5-1 shows the service symbol that used to represent a service in an assembly diagram:



1421
1422 *Figure 5-1: Service symbol*

1423
1424 Figure 5-2 shows the assembly diagram for the MyValueComposite containing the service
1425 MyValueService.



1426
1427 *Figure 5-2: MyValueComposite showing Service*

1428
1429 Snippet 5-3 shows the MyValueComposite.composite file for the MyValueComposite containing the
1430 service element for the MyValueService, which is a promote of the service offered by the
1431 MyValueServiceComponent. The name of the promoted service is omitted since
1432 MyValueServiceComponent offers only one service. The composite service MyValueService is bound
1433 using a Web service binding.

1434

```

1435 <?xml version="1.0" encoding="ASCII"?>
1436 <!-- MyValueComposite_4 example -->
1437 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1438 targetNamespace="http://foo.com"
1439 name="MyValueComposite" >
1440
1441 ...
1442
1443 <service name="MyValueService" promote="MyValueServiceComponent">
1444 <interface.java interface="services.myvalue.MyValueService"/>
1445 <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
1446 wsdl.port(MyValueService/MyValueServiceSOAP)"/>
1447 </service>
1448
1449 <component name="MyValueServiceComponent">
1450 <implementation.java
1451 class="services.myvalue.MyValueServiceImpl"/>
1452 <property name="currency">EURO</property>
1453 <service name="MyValueService"/>
1454 <reference name="customerService"/>
1455 <reference name="stockQuoteService"/>
1456 </component>
1457
1458 ...
1459
1460 </composite>

```

1461 *Snippet 5-3: Example composite with a service*

1462 5.2 Reference

1463 The **references of a composite** are defined by **promoting** references defined by components contained
1464 in the composite. Each promoted reference indicates that the component reference needs to be resolved
1465 by services outside the composite. A component reference is promoted using a composite **reference**
1466 **element**.

1467 A composite reference is represented by a **reference element** which is a child of a composite element.
1468 There can be **zero or more reference** elements in a composite. Snippet 5-4 shows the composite
1469 pseudo-schema with the pseudo-schema for a **reference** element:

```

1471 <?xml version="1.0" encoding="ASCII"?>
1472 <!-- Composite Reference schema snippet -->
1473 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1474 ...
1475 <reference name="xs:NCName" target="list of xs:anyURI"?
1476 promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
1477 multiplicity="0..1 or 1..1 or 0..n or 1..n"
1478 requires="list of xs:QName"? policySets="list of xs:QName"?>*
1479 <interface ... />?
1480 <binding ... />*
1481 <callback?>
1482 <binding ... />+
1483 </callback>
1484 <requires/>*
1485 <policySetAttachment/>*
1486 </reference>
1487 ...
1488 </composite>

```

1489 *Snippet 5-4: composite Pseudo-Schema with reference Child Element*

1490
1491 The **composite reference** element has the **attributes**:

- 1492 • **name : NCName (1..1)** – the name of the reference. The name of a composite <reference/> element
 1493 MUST be unique across all the composite references in the composite. [ASM60006] The name of the
 1494 composite reference can be different than the name of the promoted component reference.
- 1495 • **promote : anyURI (1..n)** – identifies one or more promoted component references. The value is a list
 1496 of values of the form <component-name>/<reference-name> separated by spaces. The reference
 1497 name can be omitted if the component has only one reference. Each of the URIs declared by a
 1498 composite reference's @promote attribute MUST identify a component reference within the
 1499 composite. [ASM60007] <include/> processing MUST take place before the processing of the
 1500 @promote attribute of a composite reference is performed. [ASM60037]

1501 The same component reference can be promoted more than once, using different composite
 1502 references, but only if the multiplicity defined on the component reference is 0..n or 1..n. The
 1503 multiplicity on the composite reference can restrict accordingly.

1504 Where a composite reference promotes two or more component references:

- 1505 – the interfaces of the component references promoted by a composite reference MUST be the
 1506 same, or if the composite reference itself declares an interface then each of the component
 1507 reference interfaces MUST be a compatible subset of the composite reference interface..
 1508 [ASM60008]
- 1509 – the intents declared on a composite reference and on the component references which it
 1510 promotes MUST NOT be mutually exclusive. [ASM60009] The intents which apply to the
 1511 composite reference in this case are the union of the intents specified for each of the promoted
 1512 component references plus any intents declared on the composite reference itself. If any intents
 1513 in the set which apply to a composite reference are mutually exclusive then the SCA runtime
 1514 MUST raise an error. [ASM60010]

- 1515 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
 1516 [SCA-POLICY] for a description of this attribute. Specified intents add to or further qualify the intents
 1517 defined for the promoted component reference.

- 1518 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
 1519 [SCA-POLICY] for a description of this attribute.

- 1520 • **multiplicity : (1..1)** - Defines the number of wires that can connect the reference to target services.
 1521 The multiplicity of a composite reference is always specified explicitly and can have one of the
 1522 following values

- 1523 – 0..1 – zero or one wire can have the reference as a source
- 1524 – 1..1 – one wire can have the reference as a source
- 1525 – 0..n - zero or more wires can have the reference as a source
- 1526 – 1..n – one or more wires can have the reference as a source

1527 The multiplicity of a composite reference MUST be equal to or further restrict the multiplicity of each
 1528 of the component references that it promotes, with the exception that the multiplicity of the composite
 1529 reference does not have to require a target if there is already a target on the component reference.
 1530 This means that a component reference with multiplicity 1..1 and a target can be promoted by a
 1531 composite reference with multiplicity 0..1, and a component reference with multiplicity 1..n and one or
 1532 more targets can be promoted by a composite reference with multiplicity 0..n or 0..1. [ASM60011]

1533 The valid values for composite reference multiplicity are shown in the following tables:
 1534

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

1535

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

1536

1537 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on multiplicity setting.
 1538 Each value wires the reference to a service in a composite that uses the composite containing the
 1539 reference as an implementation for one of its components. For more details on wiring see [the section](#)
 1540 [on Wires](#).

1541 • **wiredByImpl : boolean (0..1)** – a boolean value. If set to "true" it indicates that the target of the
 1542 reference is set at runtime by the implementation code (for example by the code obtaining an
 1543 endpoint reference by some means and setting this as the target of the reference through the use of
 1544 programming interfaces defined by the relevant Client and Implementation specification). If "true" is
 1545 set, then the reference is not intended to be wired statically within a using composite, but left unwired.
 1546 All the component references promoted by a single composite reference MUST have the same value
 1547 for @wiredByImpl. [ASM60035] If the @wiredByImpl attribute is not specified on the composite
 1548 reference, the default value is "true" if all of the promoted component references have a wiredByImpl
 1549 value of "true", and the default value is "false" if all the promoted component references have a
 1550 wiredByImpl value of "false". If the @wiredByImpl attribute is specified, its value MUST be "true" if all
 1551 of the promoted component references have a wiredByImpl value of "true", and its value MUST be
 1552 "false" if all the promoted component references have a wiredByImpl value of "false". [ASM60036]

1553 The **composite reference** element has the **child elements**, whatever is not specified is
 1554 defaulted from the promoted component reference(s).

1555 • **interface : Interface (0..1) - zero or one interface element** which declares an interface for the
 1556 composite reference. If a composite reference has an interface specified, it MUST provide an
 1557 interface which is the same or which is a compatible superset of the interface(s) declared by the
 1558 promoted component reference(s). [ASM60012] If no interface is declared on a composite reference,
 1559 the interface from one of its promoted component references MUST be used for the component type
 1560 associated with the composite. [ASM60013] For details on the interface element see [the Interface](#)
 1561 [section](#).

1562 • **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as children. If
 1563 one or more **bindings** are specified they **override** any and all of the bindings defined for the
 1564 promoted component reference from the composite reference perspective. The bindings defined on
 1565 the component reference are still in effect for local wires within the composite that have the
 1566 component reference as their source. Details of the binding element are described in the [Bindings](#)
 1567 [section](#). For more details on wiring see [the section on Wires](#).

1568 A reference identifies zero or more target services which satisfy the reference. This can be done in a
1569 number of ways, which are fully described in section "[Specifying the Target Service\(s\) for a](#)
1570 [Reference](#)".

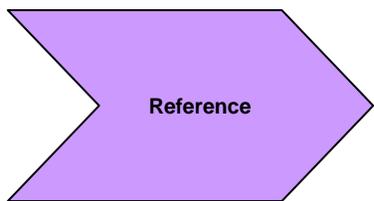
1571 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface has a callback
1572 defined and the callback element has one or more **binding** elements as subelements. The **callback**
1573 and its binding subelements are specified if there is a need to have binding details used to handle
1574 callbacks. Callback binding elements attached to the composite reference override any callback
1575 binding elements defined on any of the promoted component references. If the callback element is
1576 not present on the composite service, any callback binding elements that are declared on all the
1577 promoted references are used. If the callback element is not present at all, the behaviour is runtime
1578 implementation dependent.

1579 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
1580 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

1581 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
1582 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
1583 description of this element.

1584 5.2.1 Example Reference

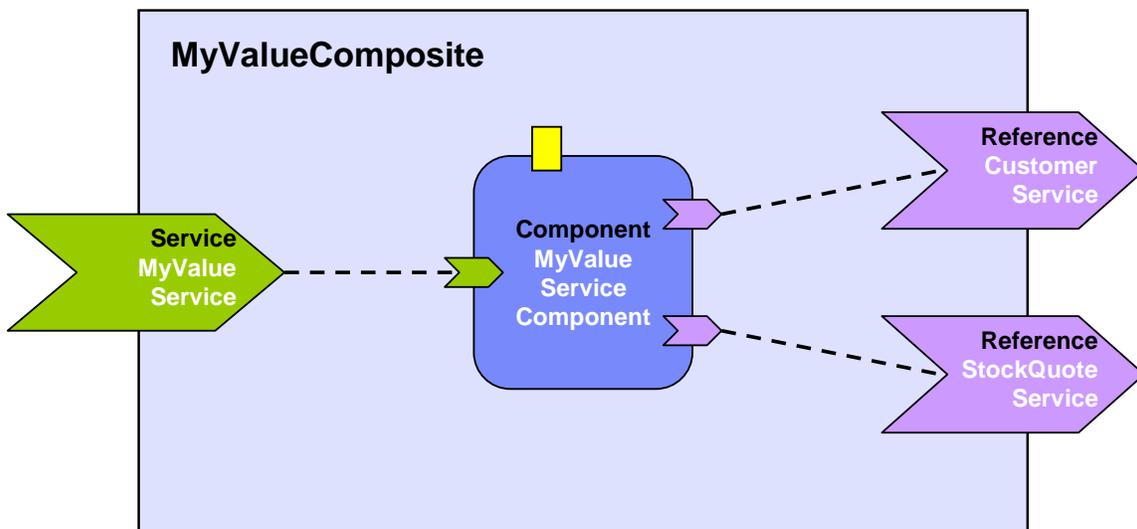
1585 Figure 5-3 shows the reference symbol that is used to represent a reference in an assembly diagram.



1586
1587 *Figure 5-3: Reference symbol*

1588
1589 Figure 5-4 shows the assembly diagram for the MyValueComposite containing the reference
1590 CustomerService and the reference StockQuoteService.

1591



1592
1593 *Figure 5-4: MyValueComposite showing References*

1594

1595 Snippet 5-5 shows the MyValueComposite.composite file for the MyValueComposite containing the
1596 reference elements for the CustomerService and the StockQuoteService. The reference CustomerService
1597 is bound using the SCA binding. The reference StockQuoteService is bound using the Web service
1598 binding. The endpoint addresses of the bindings can be specified, for example using the binding **@uri**
1599 attribute (for details see the [Bindings](#) section), or overridden in an enclosing composite. Although in this
1600 case the reference StockQuoteService is bound to a Web service, its interface is defined by a Java
1601 interface, which was created from the WSDL portType of the target web service.

1602

```
1603 <?xml version="1.0" encoding="ASCII"?>
1604 <!-- MyValueComposite_3 example -->
1605 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1606           targetNamespace="http://foo.com"
1607           name="MyValueComposite" >
1608
1609     ...
1610
1611     <component name="MyValueServiceComponent">
1612       <implementation.java
1613         class="services.myvalue.MyValueServiceImpl"/>
1614       <property name="currency">EURO</property>
1615       <reference name="customerService"/>
1616       <reference name="stockQuoteService"/>
1617     </component>
1618
1619     <reference name="CustomerService"
1620       promote="MyValueServiceComponent/customerService">
1621       <interface.java interface="services.customer.CustomerService"/>
1622       <!-- The following forces the binding to be binding.sca -->
1623       <!-- whatever is specified by the component reference or -->
1624       <!-- by the underlying implementation -->
1625       <binding.sca/>
1626     </reference>
1627
1628     <reference name="StockQuoteService"
1629       promote="MyValueServiceComponent/stockQuoteService">
1630       <interface.java
1631         interface="services.stockquote.StockQuoteService"/>
1632       <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
1633         wsdl.port(StockQuoteService/StockQuoteServiceSOAP)"/>
1634     </reference>
1635
1636     ...
1637
1638 </composite>
```

1639 *Snippet 5-5: Example composite with a reference*

1640 5.3 Property

1641 **Properties** allow for the configuration of an implementation with externally set data values. A composite
1642 can declare zero or more properties. Each property has a type, which is either simple or complex. An
1643 implementation can also define a default value for a property. Properties can be configured with values in
1644 the components that use the implementation.

1645 Snippet 5-6 shows the composite pseudo-schema with the pseudo-schema for a **reference** element:

1646

```
1647 <?xml version="1.0" encoding="ASCII"?>
1648 <!-- Composite Property schema snippet -->
1649 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912" ... >
1650   ...
1651   <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
```

```

1652     many="xs:boolean"? mustSupply="xs:boolean"?>*
1653     default-property-value?
1654 </property>
1655 ...
1656 </composite>

```

1657 *Snippet 5-6: composite Pseudo-Schema with property Child Element*

1658

1659 The **composite property** element has the **attributes**:

1660 • **name : NCName (1..1)** - the name of the property. The @name attribute of a composite property
 1661 **MUST** be unique amongst the properties of the same composite. [ASM60014]

1662 • one of (1..1):

1663 – **type : QName** – the type of the property - the qualified name of an XML schema type

1664 – **element : QName** – the type of the property defined as the qualified name of an XML schema
 1665 global element – the type is the type of the global element

1666 **A single property element MUST NOT** contain both a @type attribute and an @element
 1667 attribute. [ASM60040]

1668 • **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). The
 1669 default is **false**. In the case of a multi-valued property, it is presented to the implementation as a
 1670 collection of property values.

1671 • **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the component that
 1672 uses the composite – when mustSupply="true" the component has to supply a value since the
 1673 composite has no default value for the property. A default-property-value is only worth declaring
 1674 when mustSupply="false" (the default setting for the @mustSupply attribute), since the implication of
 1675 a default value is that it is used only when a value is not supplied by the using component.

1676 The property element can contain a **default-property-value**, which provides default value for the
 1677 property. The form of the default property value is as described in the section on [Component Property](#).

1678 Implementation types other than **composite** can declare properties in an implementation-dependent form
 1679 (e.g. annotations within a Java class), or through a property declaration of exactly the form described
 1680 above in a componentType file.

1681 Property values can be configured when an implementation is used by a component. The form of the
 1682 property configuration is shown in [the section on Components](#).

1683 5.3.1 Property Examples

1684 For the example Property declaration and value setting in Snippet 5-8, the complex type in Snippet 5-7 is
 1685 used as an example:

1686

```

1687 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1688             targetNamespace="http://foo.com/"
1689             xmlns:tns="http://foo.com/">
1690   <!-- ComplexProperty schema -->
1691   <xsd:element name="fooElement" type="tns:MyComplexType"/>
1692   <xsd:complexType name="MyComplexType">
1693     <xsd:sequence>
1694       <xsd:element name="a" type="xsd:string"/>
1695       <xsd:element name="b" type="xsd:anyURI"/>
1696     </xsd:sequence>
1697     <attribute name="attr" type="xsd:string" use="optional"/>
1698   </xsd:complexType>
1699 </xsd:schema>

```

1700 *Snippet 5-7: Complex Type for Snippet 5-8*

1701

1702 The following composite demonstrates the declaration of a property of a complex type, with a default value,
1703 plus it demonstrates the setting of a property value of a complex type within a component:

1704

```
1705 <?xml version="1.0" encoding="ASCII"?>
1706 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1707           xmlns:foo="http://foo.com"
1708           targetNamespace="http://foo.com"
1709           name="AccountServices">
1710 <!-- AccountServices Example1 -->
1711
1712   ...
1713
1714   <property name="complexFoo" type="foo:MyComplexType">
1715     <value>
1716       <foo:a>AValue</foo:a>
1717       <foo:b>InterestingURI</foo:b>
1718     </value>
1719   </property>
1720
1721   <component name="AccountServiceComponent">
1722     <implementation.java class="foo.AccountServiceImpl"/>
1723     <property name="complexBar" source="$complexFoo"/>
1724     <reference name="accountDataService"
1725               target="AccountDataServiceComponent"/>
1726     <reference name="stockQuoteService" target="StockQuoteService"/>
1727   </component>
1728
1729   ...
1730
1731 </composite>
```

1732 *Snippet 5-8: Example property with a Complex Type*

1733

1734 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the property is
1735 defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the composite and it
1736 references the example XSD, where **MyComplexType** is defined. The declaration of **complexFoo**
1737 contains a default value. This is declared as the content of the property element. In this example, the
1738 default value consists of the element **value** which is of type **foo:MyComplexType** and it has two child
1739 elements **<foo:a>** and **<foo:b>**, following the definition of **MyComplexType**.

1740 In the component **AccountServiceComponent**, the component sets the value of the property
1741 **complexBar**, declared by the implementation configured by the component. In this case, the type of
1742 **complexBar** is **foo:MyComplexType**. The example shows that the value of the **complexBar** property is set
1743 from the value of the **complexFoo** property – the **@source** attribute of the property element for
1744 **complexBar** declares that the value of the property is set from the value of a property of the containing
1745 composite. The value of the **@source** attribute is **\$complexFoo**, where **complexFoo** is the name of a
1746 property of the composite. This value implies that the whole of the value of the source property is used to
1747 set the value of the component property.

1748 Snippet 5-9 illustrates the setting of the value of a property of a simple type (a string) from **part** of the
1749 value of a property of the containing composite which has a complex type:

1750

```
1751 <?xml version="1.0" encoding="ASCII"?>
1752 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
1753           xmlns:foo="http://foo.com"
1754           targetNamespace="http://foo.com"
1755           name="AccountServices">
1756 <!-- AccountServices Example2 -->
1757
1758   ...
1759
```

```

1760 <property name="complexFoo" type="foo:MyComplexType">
1761   <value>
1762     <foo:a>AValue</foo:a>
1763     <foo:b>InterestingURI</foo:b>
1764   </value>
1765 </property>
1766
1767 <component name="AccountServiceComponent">
1768   <implementation.java class="foo.AccountServiceImpl"/>
1769   <property name="currency" source="$complexFoo/a"/>
1770   <reference name="accountDataService"
1771     target="AccountDataServiceComponent"/>
1772   <reference name="stockQuoteService" target="StockQuoteService"/>
1773 </component>
1774
1775 ...
1776
1777 </composite>

```

1778 *Snippet 5-9: Example property with a Simple Type*

1779

1780 In the example in Snippet 5-9, the component **AccountServiceComponent** sets the value of a property
1781 called **currency**, which is of type string. The value is set from a property of the composite
1782 **AccountServices** using the @source attribute set to **\$complexFoo/a**. This is an XPath expression that
1783 selects the property name **complexFoo** and then selects the value of the **a** subelement of the value of
1784 complexFoo. The "a" subelement is a string, matching the type of the currency property.

1785 Further examples of declaring properties and setting property values in a component:

- 1786 – Declaration of a property with a simple type and a default value:

```

1787 <property name="SimpleTypeProperty" type="xsd:string">
1788   <value>MyValue</value>
1789 </property>

```

1790 *Snippet 5-10: Example property with a Simple Type and Default Value*

1791

- 1792 – Declaration of a property with a complex type and a default value:

```

1793 <property name="complexFoo" type="foo:MyComplexType">
1794   <value>
1795     <foo:a>AValue</foo:a>
1796     <foo:b>InterestingURI</foo:b>
1797   </value>
1798 </property>

```

1799 *Snippet 5-11: Example property with a Complex Type and Default Value*

1800

- 1801 – Declaration of a property with a global element type:

```

1802 <property name="elementFoo" element="foo:fooElement">
1803   <foo:fooElement>
1804     <foo:a>AValue</foo:a>
1805     <foo:b>InterestingURI</foo:b>
1806   </foo:fooElement>
1807 </property>

```

1808 *Snippet 5-12: Example property with a Global Element Type*

1809 5.4 Wire

1810 SCA wires within a composite connect source component references to target component services.

1811 One way of defining a wire is by **configuring a reference of a component using its @target attribute**.
1812 The reference element is configured with the wire-target-URI of the service(s) that resolve the reference.
1813 Multiple target services are valid when the reference has a multiplicity of 0..n or 1..n.

1814 An alternative way of defining a Wire is by means of a **wire element** which is a child of the composite
1815 element. There can be **zero or more** wire elements in a composite. This alternative method for defining
1816 wires is useful in circumstances where separation of the wiring from the elements the wires connect helps
1817 simplify development or operational activities. An example is where the components used to build a
1818 Domain are relatively static but where new or changed applications are created regularly from those
1819 components, through the creation of new assemblies with different wiring. Deploying the wiring
1820 separately from the components allows the wiring to be created or modified with minimum effort.

1821 Note that a Wire specified via a wire element is equivalent to a wire specified via the @target attribute of
1822 a reference. The rule which forbids mixing of wires specified with the @target attribute with the
1823 specification of endpoints in binding subelements of the reference also applies to wires specified via
1824 separate wire elements.

1825 Snippet 5-13 shows the composite pseudo-schema with the pseudo-schema for the wire child element:

1826

```
1827 <!-- Wires schema snippet -->  
1828 <composite ...>  
1829   ...  
1830   <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/> *  
1831   ...  
1832 </composite>
```

1833 *Snippet 5-13: composite Pseudo-Schema with wire Child Element*

1834

1835 The **reference element of a component** has a list of one or more of the following **wire-target-URI**
1836 values for the target, with multiple values separated by a space:

1837 • **<component-name>[/<service-name> /<binding-name>]?]?**

1838 ○ <component-name> is the name of the target component.

1839 ○ <service-name> is the name of the target service within the component.

1840 If <service-name> is present, the component service with @name corresponding
1841 to <service-name> MUST be used for the wire. [ASM60046]

1842 If there is no component service with @name corresponding to <service-name>,
1843 the SCA runtime MUST raise an error. [ASM60047]

1844 If <service-name> is not present, the target component MUST have one and only
1845 one service with an interface that is a compatible superset of the wire source's
1846 interface and satisfies the policy requirements of the wire source, and the SCA
1847 runtime MUST use this service for the wire. [ASM60048]

1848 ○ <binding-name> is the name of the service's binding to use. The <binding-name>
1849 can be the default name of a binding element (see section 8 "Binding").

1850
1851 If <binding-name> is present, the <binding/> subelement of the target service
1852 with @name corresponding to <binding-name> MUST be used for the wire.
1853 [ASM60049] If there is no <binding/> subelement of the target service with
1854 @name corresponding to <binding-name>, the SCA runtime MUST raise an error.
1855 [ASM60050] If <binding-name> is not present and the target service has multiple
1856 <binding/> subelements, the SCA runtime MUST choose one and only one of the
1857 <binding/> elements which satisfies the mutual policy requirements of the
1858 reference and the service, and the SCA runtime MUST use this binding for the
1859 wire. [ASM60051]

1860

1861 The **wire element** has the attributes:

- 1862 • **source (1..1)** – names the source component reference. The valid URI scheme is:
- 1863 – `<component-name>[/<reference-name>]?`
 - 1864 • where the source is a component reference. The reference name can be omitted if the
 - 1865 source component only has one reference
- 1866 • **target (1..1)** – names the target component service. The valid URI scheme is the same as the one
- 1867 defined for component references above.
- 1868 • **replace (0..1)** - a boolean value, with the default of "false". When a wire element has
- 1869 `@replace="false"`, the wire is added to the set of wires which apply to the reference identified by the
- 1870 `@source` attribute. When a wire element has `@replace="true"`, the wire is added to the set of wires
- 1871 which apply to the reference identified by the `@source` attribute - but any wires for that reference
- 1872 specified by means of the `@target` attribute of the reference are removed from the set of wires which
- 1873 apply to the reference.
- 1874 In other words, if any `<wire/>` element with `@replace="true"` is used for a particular reference, the
- 1875 value of the `@target` attribute on the reference is ignored - and this permits existing wires on the
- 1876 reference to be overridden by separate configuration, where the reference is on a component at the
- 1877 Domain level.

1878 `<include/>` processing **MUST** take place before the `@source` and `@target` attributes of a wire are

1879 resolved. [ASM60039]

1880 For a composite used as a component implementation, wires can only link sources and targets that are

1881 contained in the same composite (irrespective of which file or files are used to describe the composite).

1882 Wiring to entities outside the composite is done through services and references of the composite with

1883 wiring defined by the next higher composite.

1884 The interface declared by the target of a wire **MUST** be a compatible superset of the interface declared by

1885 the source of the wire. [ASM60043] See the section on Interface Compatibility for a definition of

1886 "compatible superset".

1887 A Wire can connect between different interface languages (e.g. Java interfaces and WSDL portTypes) in

1888 either direction, as long as the operations defined by the two interface types are equivalent. They are

1889 equivalent if the operation(s), parameter(s), return value(s) and faults/exceptions map to each other.

1890 Service clients cannot (portably) ask questions at runtime about additional interfaces that are provided by

1891 the implementation of the service (e.g. the result of "instance of" in Java is non portable). It is valid for an

1892 SCA implementation to have proxies for all wires, so that, for example, a reference object passed to an

1893 implementation might only have the business interface of the reference and might not be an instance of

1894 the (Java) class which is used to implement the target service, even where the interface is local and the

1895 target service is running in the same process.

1896 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of an un-

1897 wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA runtime

1898 SHOULD issue a warning. [ASM60021]

1899 5.4.1 Wire Examples

1900 Figure 5-5: MyValueComposite2 showing Wires shows the assembly diagram for the

1901 MyValueComposite2 containing wires between service, components and references.

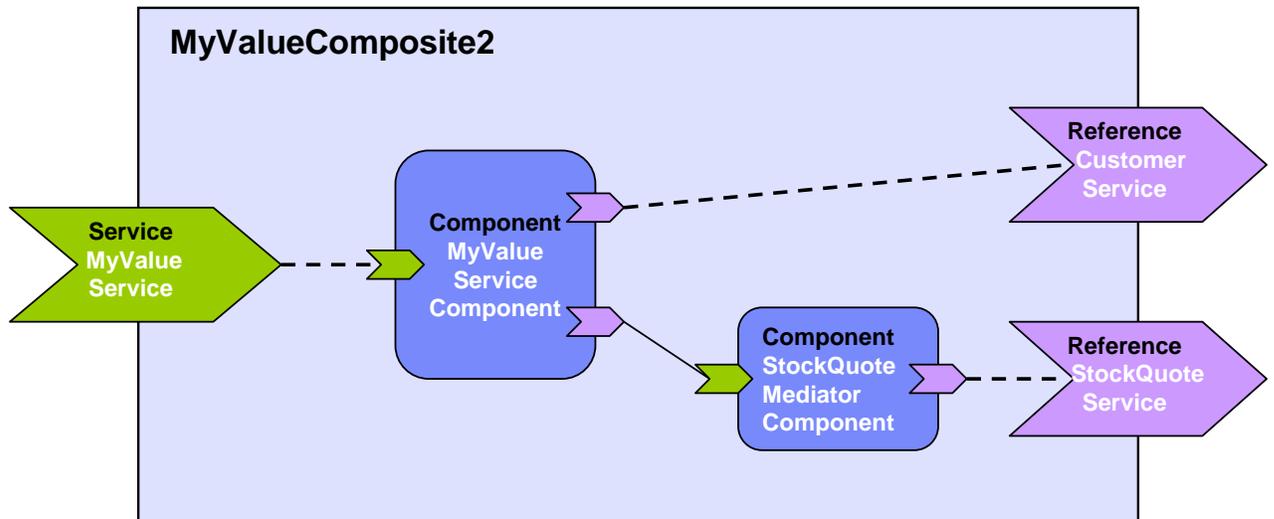


Figure 5-5: MyValueComposite2 showing Wires

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

```

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

  <service name="MyValueService" promote="MyValueServiceComponent">
    <interface.java interface="services.myvalue.MyValueService"/>
    <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
wsdl.port(MyValueService/MyValueServiceSOAP)"/>
  </service>

  <component name="MyValueServiceComponent">
    <implementation.java
class="services.myvalue.MyValueServiceImpl"/>
    <property name="currency">EURO</property>
    <service name="MyValueService"/>
    <reference name="customerService"/>
    <reference name="stockQuoteService"/>
  </component>

  <wire source="MyValueServiceComponent/stockQuoteService"
target="StockQuoteMediatorComponent"/>

  <component name="StockQuoteMediatorComponent">
    <implementation.java class="services.myvalue.SQMediatorImpl"/>
    <property name="currency">EURO</property>
    <reference name="stockQuoteService"/>
  </component>

  <reference name="CustomerService"
promote="MyValueServiceComponent/customerService">

```

```

1945     <interface.java interface="services.customer.CustomerService"/>
1946     <binding.sca/>
1947 </reference>
1948
1949     <reference name="StockQuoteService"
1950           promote="StockQuoteMediatorComponent">
1951       <interface.java
1952           interface="services.stockquote.StockQuoteService"/>
1953       <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
1954           wsdl.port (StockQuoteService/StockQuoteServiceSOAP)"/>
1955     </reference>
1956
1957 </composite>

```

1958 *Snippet 5-14: Example composite with a wire*

1959 5.4.2 Autowire

1960 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.
 1961 Autowire enables component references to be automatically wired to component services which will
 1962 satisfy those references, without the need to create explicit wires between the references and the
 1963 services. When the autowire feature is used, a component reference which is not promoted and which is
 1964 not explicitly wired to a service within a composite is automatically wired to a target service within the
 1965 same composite. Autowire works by searching within the composite for a service interface which
 1966 matches the interface of the references.

1967 The autowire feature is not used by default. Autowire is enabled by the setting of an @autowire attribute
 1968 to "true". Autowire is disabled by setting of the @autowire attribute to "false" The @autowire attribute can
 1969 be applied to any of the following elements within a composite:

- 1970 • reference
- 1971 • component
- 1972 • composite

1973 Where an element does not have an explicit setting for the @autowire attribute, it inherits the setting from
 1974 its parent element. Thus a reference element inherits the setting from its containing component. A
 1975 component element inherits the setting from its containing composite. Where there is no setting on any
 1976 level, autowire="false" is the default.

1977 As an example, if a composite element has autowire="true" set, this means that autowiring is enabled for
 1978 all component references within that composite. In this example, autowiring can be turned off for specific
 1979 components and specific references through setting autowire="false" on the components and references
 1980 concerned.

1981 For each component reference for which autowire is enabled, the SCA runtime **MUST** search within the
 1982 composite for target services which have an interface that is a compatible superset of the interface of the
 1983 reference. [ASM60022]

1984 The intents, and policies applied to the service **MUST** be compatible with those on the reference when
 1985 using autowire to wire a reference – so that wiring the reference to the service will not cause an error due
 1986 to policy mismatch [ASM60024] (see the Policy Framework specification [SCA-POLICY] for details)

1987 If the search finds **1 or more** valid target service for a particular reference, the action taken depends on
 1988 the multiplicity of the reference:

- 1989 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime **MUST** wire the reference to
 1990 one of the set of valid target services chosen from the set in a runtime-dependent fashion
 1991 [ASM60025]
- 1992 • for an autowire reference with multiplicity 0..n or 1..n, the reference **MUST** be wired to all of the set of
 1993 valid target services [ASM60026]

1994 If the search finds **no** valid target services for a particular reference, the action taken depends on the
 1995 multiplicity of the reference:

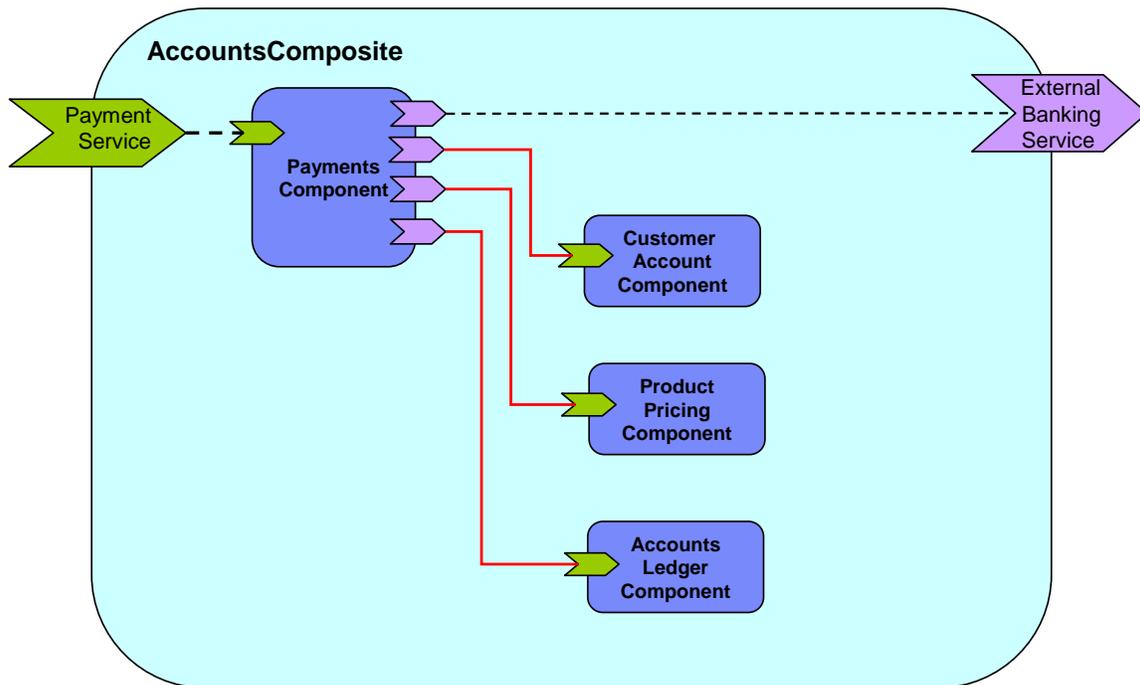
- 1996 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT raise an error [ASM60027]
- 1997
- 1998
- 1999 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid target services an error MUST be raised by the SCA runtime since the reference is intended to be wired [ASM60028]
- 2000

2001 5.4.3 Autowire Examples

2002 Snippet 5-15 and Snippet 5-16 demonstrate two versions of the same composite – the first version is
 2003 done using explicit wires, with no autowiring used, the second version is done using autowire. In both
 2004 cases the end result is the same – the same wires connect the references to the services.

2005 Figure 5-6 is a diagram for the composite:

2006



2007

2008 *Figure 5-6: Example Composite for Autowire*

2009

2010 Snippet 5-15 is the composite using explicit wires:

2011

```

2012 <?xml version="1.0" encoding="UTF-8"?>
2013 <!-- Autowire Example - No autowire -->
2014 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2015           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2016           xmlns:foo="http://foo.com"
2017           targetNamespace="http://foo.com"
2018           name="AccountComposite">
2019
2020   <service name="PaymentService" promote="PaymentsComponent"/>
2021
2022   <component name="PaymentsComponent">
2023     <implementation.java class="com.foo.accounts.Payments"/>
2024     <service name="PaymentService"/>
2025     <reference name="CustomerAccountService"
2026               target="CustomerAccountComponent"/>
2027     <reference name="ProductPricingService"
  
```

```

2028         target="ProductPricingComponent"/>
2029     <reference name="AccountsLedgerService"
2030         target="AccountsLedgerComponent"/>
2031     <reference name="ExternalBankingService"/>
2032 </component>
2033
2034 <component name="CustomerAccountComponent">
2035     <implementation.java class="com.foo.accounts.CustomerAccount"/>
2036 </component>
2037
2038 <component name="ProductPricingComponent">
2039     <implementation.java class="com.foo.accounts.ProductPricing"/>
2040 </component>
2041
2042 <component name="AccountsLedgerComponent">
2043     <implementation.composite name="foo:AccountsLedgerComposite"/>
2044 </component>
2045
2046 <reference name="ExternalBankingService"
2047     promote="PaymentsComponent/ExternalBankingService"/>
2048
2049 </composite>

```

2050 *Snippet 5-15: Example composite with Explicit wires*

2051

2052 Snippet 5-16 is the composite using autowire:

2053

```

2054 <?xml version="1.0" encoding="UTF-8"?>
2055 <!-- Autowire Example - With autowire -->
2056 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2057     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2058     xmlns:foo="http://foo.com"
2059     targetNamespace="http://foo.com"
2060     name="AccountComposite">
2061
2062     <service name="PaymentService" promote="PaymentsComponent">
2063         <interface.java class="com.foo.PaymentServiceInterface"/>
2064     </service>
2065
2066     <component name="PaymentsComponent" autowire="true">
2067         <implementation.java class="com.foo.accounts.Payments"/>
2068         <service name="PaymentService"/>
2069         <reference name="CustomerAccountService"/>
2070         <reference name="ProductPricingService"/>
2071         <reference name="AccountsLedgerService"/>
2072         <reference name="ExternalBankingService"/>
2073     </component>
2074
2075     <component name="CustomerAccountComponent">
2076         <implementation.java class="com.foo.accounts.CustomerAccount"/>
2077     </component>
2078
2079     <component name="ProductPricingComponent">
2080         <implementation.java class="com.foo.accounts.ProductPricing"/>
2081     </component>
2082
2083     <component name="AccountsLedgerComponent">
2084         <implementation.composite name="foo:AccountsLedgerComposite"/>
2085     </component>
2086
2087     <reference name="ExternalBankingService"
2088         promote="PaymentsComponent/ExternalBankingService"/>
2089

```

2090 `</composite>`

2091 *Snippet 5-16: composite of Snippet 5-15 Using autowire*

2092

2093 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires for
2094 any of its references – the wires are created automatically through autowire.

2095 **Note:** In the second example, it would be possible to omit all of the service and reference elements from
2096 the PaymentsComponent. They are left in for clarity, but if they are omitted, the component service and
2097 references still exist, since they are provided by the implementation used by the component.

2098 5.5 Using Composites as Component Implementations

2099 Composites can be used as **component implementations** in higher-level composites – in other words
2100 the higher-level composites can have components which are implemented by composites.

2101 When a composite is used as a component implementation, it defines a boundary of visibility.

2102 Components within the composite cannot be referenced directly by the using component. The using
2103 component can only connect wires to the services and references of the used composite and set values
2104 for any properties of the composite. The internal construction of the composite is invisible to the using
2105 component. The boundary of visibility, sometimes called encapsulation, can be enforced when
2106 assembling components and composites, but such encapsulation structures might not be enforceable in a
2107 particular implementation language.

2108 A composite used as a component implementation also needs to honor a completeness contract. The
2109 services, references and properties of the composite form a contract (represented by the component type
2110 of the composite) which is relied upon by the using component. The concept of completeness of the
2111 composite implies that, once all `<include/>` element processing is performed on the composite:

- 2112 1. For a composite used as a component implementation, each composite service
2113 offered by the composite **MUST** promote a component service of a component
2114 that is within the composite. [ASM60032]
- 2115 2. For a composite used as a component implementation, every component
2116 reference of components within the composite with a multiplicity of 1..1 or 1..n
2117 **MUST** be wired or promoted. [ASM60033] (according to the various rules for
2118 specifying target services for a component reference described in the section "
2119 Specifying the Target Service(s) for a Reference").
- 2120 3. For a composite used as a component implementation, all properties of
2121 components within the composite, where the underlying component
2122 implementation specifies "mustSupply=true" for the property, **MUST** either
2123 specify a value for the property or source the value from a composite property.
2124 [ASM60034]

2125 The component type of a composite is defined by the set of composite service elements, composite
2126 reference elements and composite property elements that are the children of the composite element.

2127 Composites are used as component implementations through the use of the **implementation.composite**
2128 element as a child element of the component. Snippet 5-17 shows the pseudo-schema for the
2129 `implementation.composite` element:

2130

```
2131 <!-- implementation.composite pseudo-schema -->  
2132 <implementation.composite name="xs:QName" requires="list of xs:QName"?  
2133 policySets="list of xs:QName"?>
```

2134 *Snippet 5-17: implementation.composite Pseudo-Schema*

2135

2136 The **implementation.composite** element has the attributes:

- 2137 • **name (1..1)** – the name of the composite used as an implementation. The @name attribute of an
2138 <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.
2139 [ASM60030]
- 2140 • **requires : listOfQNames (0..1)** – a list of policy intents. See the Policy Framework specification
2141 [SCA-POLICY] for a description of this attribute. Specified intents add to or further qualify the required
2142 intents defined for the promoted component reference.
- 2143 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the Policy Framework specification
2144 [SCA-POLICY] for a description of this attribute.

2145 5.5.1 Component Type of a Composite used as a Component 2146 Implementation

2147 An SCA runtime MUST introspect the componentType of a Composite used as a Component
2148 Implementation following the rules defined in the section "Component Type of a Composite used as a
2149 Component Implementation" [ASM60045]

2150 The componentType of a Composite used as a Component Implementation is introspected from the
2151 Composite document as follows:

- 2152 A <service/> element exists for each direct <service/> subelement of the <composite/> element
- 2153 • @name attribute set to the value of the @name attribute of the <service/> in the composite
 - 2154 • @requires attribute set to the value of the @requires attribute of the <service/> in the composite,
2155 if present (the value of the @requires attribute contains the intents which apply to the promoted
2156 component service, as defined in the Policy Framework specification [SCA_POLICY]). If no
2157 intents apply to the <service/> in the composite, the @requires attribute is omitted.
 - 2158 • @policySets attribute set to the value of the @policySets attribute of the <service/> in the
2159 composite, if it is present. If the @policySets attribute of the <service/> element in the composite
2160 is absent, the @policySets attribute is omitted.
 - 2161 • <interface/> subelement set to the <interface/> subelement of the <service/> element in the
2162 composite. If not declared on the composite service, it is set to the <interface/> subelement which
2163 applies to the component service which is promoted by the composite service (this is either an
2164 explicit <interface/> subelement of the component <service/>, or the <interface/> element of the
2165 corresponding <service/> in the componentType of the implementation used by the component).
 - 2166 • <binding/> subelements set to the <binding/> subelements of the <service/> element in the
2167 composite. If not declared on the composite service, the <binding/> subelements which apply to
2168 the component service promoted by the composite service are used, if any are present. If none
2169 are present in both of these locations, <binding/> subelements are omitted.
 - 2170 • <callback/> subelement is set to the <callback/> subelement of the <service/> element in the
2171 composite. If no <callback/> subelement is present on the composite <service/> element, the
2172 <callback/> subelement is omitted.
- 2173 A <reference/> element exists for each direct <reference/> subelement of the <composite/> element.
- 2174 • @name attribute set to the value of the @name attribute of the <reference/> in the composite
 - 2175 • @requires attribute set to the value of the @requires attribute of the <reference/> in the
2176 composite, if present (the value of the @requires attribute contains the intents which apply to the
2177 promoted component references, as defined in the Policy Framework specification
2178 [SCA_POLICY]). If no intents apply to the <reference/> in the composite, the @requires attribute
2179 is omitted.
 - 2180 • @policySets attribute set to the value of the @policySets attribute of the <reference/> in the
2181 composite, if present. If the @policySets attribute of the <reference/> element in the composite is
2182 absent, the @policySets attribute is omitted.
 - 2183 • @target attribute is set to the value of the @target attribute of the <reference/> in the composite,
2184 if present, otherwise the @target attribute is omitted.

- 2185 • @wiredByImpl attribute is set to the value of the @wiredByImpl attribute of the <reference/> in
2186 the composite, if present. If it is not declared on the composite reference, it is set to the value of
2187 the @wiredByImpl attribute of the promoted reference(s).
 - 2188 • @multiplicity attribute is set to the value of the @multiplicity attribute of the <reference/> in the
2189 composite
 - 2190 • <interface/> subelement set to the <interface/> subelement of the <reference/> element in the
2191 composite. If not declared on the composite reference, it is set to the <interface/> subelement
2192 which applies to one of the component reference(s) which are promoted by the composite
2193 reference (this is either an explicit <interface/> subelement of the component <reference/>, or the
2194 <interface/> element of the corresponding <reference/> in the componentType of the
2195 implementation used by the component).
 - 2196 • <binding/> subelements set to the <binding/> subelements of the <reference/> element in the
2197 composite. Otherwise, <binding/> subelements are omitted.
 - 2198 • <callback/> subelement is set to the <callback/> subelement of the <reference/> element in the
2199 composite. Otherwise, <callback/> subelements are omitted.
- 2200 A <property/> element exists for each direct <property/> subelement of the <composite/> element.
- 2201 • @name attribute set to the value of the @name attribute of the <property/> in the composite
 - 2202 • @type attribute set to the value of the @type attribute of the <property/> in the composite, if
2203 present
 - 2204 • @element attribute set to the value of the @element attribute of the <property/> in the composite,
2205 if present
2206 (Note: either a @type attribute is present or an @element attribute is present - one of them has to
2207 be present, but both are not allowed)
 - 2208 • @many attribute set to the value of the @many attribute of the <property/> in the composite, if
2209 present, otherwise omitted.
 - 2210 • @mustSupply attribute set to the value of the @mustSupply attribute of the <property/> in the
2211 composite, if present, otherwise omitted.
 - 2212 • @requires attribute set to the value of the @requires attribute of the <property/> in the composite,
2213 if present, otherwise omitted.
 - 2214 • @policySets attribute set to the value of the @policySets attribute of the <property/> in the
2215 composite, if present, otherwise omitted.
- 2216 A <implementation/> element exists if the <composite/> element has either of the @requires or
2217 @policySets attributes declared, with:
- 2218 • @requires attribute set to the value of the @requires attribute of the composite, if present,
2219 otherwise omitted.
 - 2220 • @policySets attribute set to the value of the @policySets attribute of the composite, if present,
2221 otherwise omitted.
- 2222

2223 5.5.2 Example of Composite used as a Component Implementation

2224 Snippet 5-18 shows an example of a composite which contains two components, each of which is
2225 implemented by a composite:
2226

```
2227 <?xml version="1.0" encoding="UTF-8"?>
2228 <!-- CompositeComponent example -->
2229 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2230     xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200912
2231     file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
2232     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2233     targetNamespace="http://foo.com"
```

```

2234 xmlns:foo="http://foo.com"
2235 name="AccountComposite">
2236
2237 <service name="AccountService" promote="AccountServiceComponent">
2238   <interface.java interface="services.account.AccountService"/>
2239   <binding.ws wsdlElement="AccountService#
2240     wsdl.port (AccountService/AccountServiceSOAP) "/>
2241 </service>
2242
2243 <reference name="stockQuoteService"
2244   promote="AccountServiceComponent/StockQuoteService">
2245   <interface.java
2246     interface="services.stockquote.StockQuoteService"/>
2247   <binding.ws
2248     wsdlElement="http://www.quickstockquote.com/StockQuoteService#
2249     wsdl.port (StockQuoteService/StockQuoteServiceSOAP) "/>
2250 </reference>
2251
2252 <property name="currency" type="xsd:string">EURO</property>
2253
2254 <component name="AccountServiceComponent">
2255   <implementation.composite name="foo:AccountServiceComposite1"/>
2256
2257   <reference name="AccountDataService" target="AccountDataService"/>
2258   <reference name="StockQuoteService"/>
2259
2260   <property name="currency" source="$currency"/>
2261 </component>
2262
2263 <component name="AccountDataService">
2264   <implementation.composite name="foo:AccountDataServiceComposite"/>
2265
2266   <property name="currency" source="$currency"/>
2267 </component>
2268
2269 </composite>

```

2270 *Snippet 5-18: Example of a composite Using implementation.composite*

2271 5.6 Using Composites through Inclusion

2272 In order to assist team development, composites can be developed in the form of multiple physical
 2273 artifacts that are merged into a single logical unit.

2274 A composite can include another composite by using the **include** element. This provides a recursive
 2275 inclusion capability. The semantics of included composites are that the element content children of the
 2276 included composite are inlined, with certain modification, into the using composite. This is done
 2277 recursively till the resulting composite does not contain an **include** element. The outer included
 2278 composite element itself is discarded in this process – only its contents are included as described below:

- 2279 1. All the element content children of the included composite are inlined in the
 2280 including composite.
- 2281 2. The attributes **@targetNamespace**, **@name** and **@local** of the included
 2282 composites are discarded.
- 2283 3. All the namespace declaration on the included composite element are added to
 2284 the inlined element content children unless the namespace binding is overridden
 2285 by the element content children.
- 2286 4. The attribute **@autowire**, if specified on the included composite, is included on
 2287 all inlined component element children unless the component child already
 2288 specifies that attribute.

- 2289 5. The attribute values of **@requires** and **@policySet**, if specified on the included
2290 composite, are merged with corresponding attribute on the inlined component,
2291 service and reference children elements. Merge in this context means a set union.
- 2292 6. Extension attributes ,if present on the included composite, follow the rules
2293 defined for that extension. Authors of attribute extensions on the composite
2294 element define the rules applying to those attributes for inclusion.

2295 If the included composite has the value *true* for the attribute **@local** then the including composite **MUST**
2296 have the same value for the **@local** attribute, else it is an error. [ASM60041]

2297 The composite file used for inclusion can have any contents. The composite element can contain any of
2298 the elements which are valid as child elements of a composite element, namely components, services,
2299 references, wires and includes. There is no need for the content of an included composite to be complete,
2300 so that artifacts defined within the using composite or in another associated included composite file can
2301 be referenced. For example, it is permissible to have two components in one composite file while a wire
2302 specifying one component as the source and the other as the target can be defined in a second included
2303 composite file.

2304 The SCA runtime **MUST** raise an error if the composite resulting from the inclusion of one composite into
2305 another is invalid. [ASM60031] For example, it is an error if there are duplicated elements in the using
2306 composite (e.g. two services with the same uri contributed by different included composites). It is not
2307 considered an error if the (using) composite resulting from the inclusion is incomplete (eg. wires with non-
2308 existent source or target). Such incomplete resulting composites are permitted to allow recursive
2309 composition.

2310 Snippet 5-19 snippet shows the pseudo-schema for the include element:

2311

```
2312 <?xml version="1.0" encoding="UTF-8"?>  
2313 <!-- Include snippet -->  
2314 <composite ...>  
2315   ...  
2316   <include name="xs:QName"/> *  
2317   ...  
2318 </composite>
```

2319 *Snippet 5-19: include Pseudo-Schema*

2320

2321 The **include** element has the **attribute**:

- 2322 • **name: QName (1..1)** – the name of the composite that is included. The **@name** attribute
2323 of an include element **MUST** be the QName of a composite in the SCA Domain.
2324 [ASM60042]

2325 5.6.1 Included Composite Examples

2326 Figure 5-7 shows the assembly diagram for the MyValueComposite2 containing four included
2327 composites. The **MyValueServices composite** contains the MyValueService service. The
2328 **MyValueComponents composite** contains the MyValueServiceComponent and the
2329 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences composite**
2330 contains the CustomerService and StockQuoteService references. The **MyValueWires composite**
2331 contains the wires that connect the MyValueService service to the MyValueServiceComponent, that
2332 connect the customerService reference of the MyValueServiceComponent to the CustomerService
2333 reference, and that connect the stockQuoteService reference of the StockQuoteMediatorComponent to
2334 the StockQuoteService reference. Note that this is just one possible way of building the
2335 MyValueComposite2 from a set of included composites.

2336

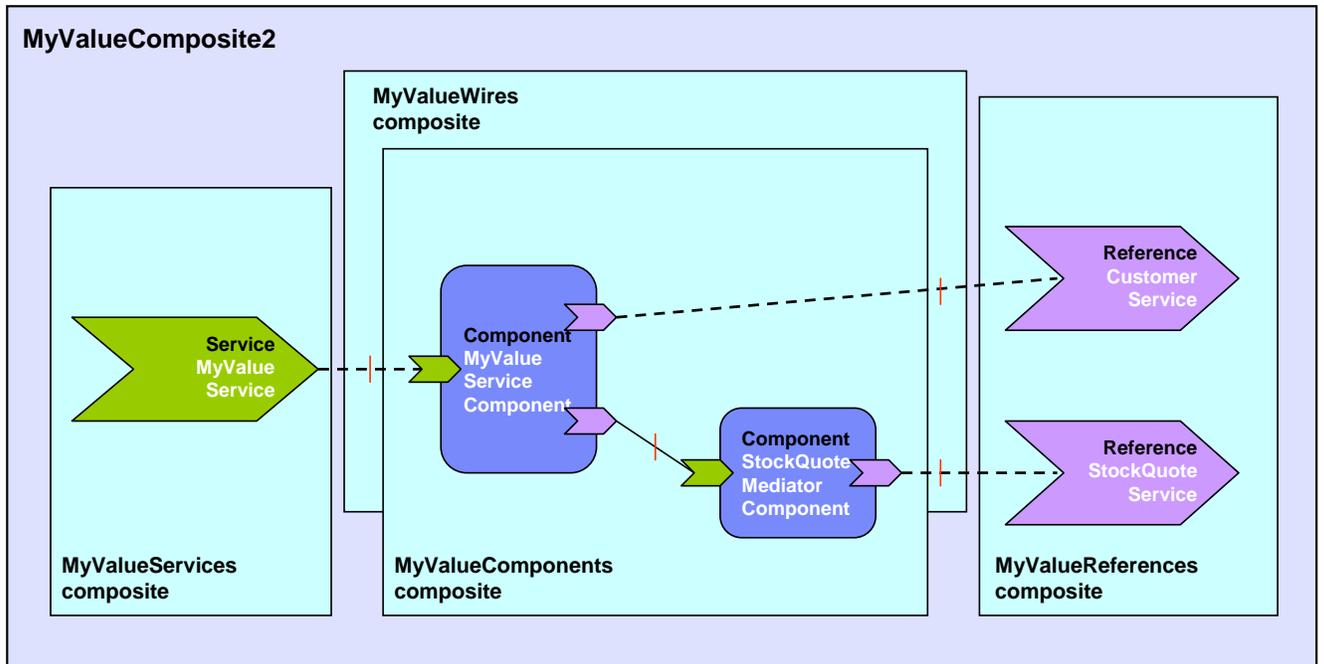


Figure 5-7 MyValueComposite2 built from 4 included composites

2337

2338

2339

2340 Snippet 5-20 shows the contents of the MyValueComposite2.composite file for the MyValueComposite2
 2341 built using included composites. In this sample it only provides the name of the composite. The composite
 2342 file itself could be used in a scenario using included composites to define components, services,
 2343 references and wires.

2344

2345

2346

2347

2348

2349

2350

2351

2352

2353

2354

2355

2356

```
<?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"/>
  <include name="foo:MyValueComponents"/>
  <include name="foo:MyValueReferences"/>
  <include name="foo:MyValueWires"/>

</composite>
```

2357

Snippet 5-20: Example composite with includes

2358

2359

Snippet 5-21 shows the content of the MyValueServices.composite file.

2360

2361

2362

2363

2364

2365

2366

2367

2368

2369

2370

2371

```
<?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="MyValueServices" >

  <service name="MyValueService" promote="MyValueServiceComponent">
    <interface.java interface="services.myvalue.MyValueService"/>
    <binding.ws wsdlElement="http://www.myvalue.org/MyValueService#
wsdl.port (MyValueService/MyValueServiceSOAP)"/>
  </service>
```

2372
2373

```
</composite>
```

2374 *Snippet 5-21: Example Partial composite with Only a service*

2375

2376 Snippet 5-22 shows the content of the MyValueComponents.composite file.

2377

```
2378 <?xml version="1.0" encoding="ASCII"?>
2379 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2380           targetNamespace="http://foo.com"
2381           xmlns:foo="http://foo.com"
2382           name="MyValueComponents" >
2383
2384   <component name="MyValueServiceComponent">
2385     <implementation.java
2386       class="services.myvalue.MyValueServiceImpl"/>
2387     <property name="currency">EURO</property>
2388   </component>
2389
2390   <component name="StockQuoteMediatorComponent">
2391     <implementation.java class="services.myvalue.SQMediatorImpl"/>
2392     <property name="currency">EURO</property>
2393   </component>
2394
2395 </composite>
```

2396 *Snippet 5-22: Example Partial composite with Only components*

2397

2398 Snippet 5-23 shows the content of the MyValueReferences.composite file.

2399

```
2400 <?xml version="1.0" encoding="ASCII"?>
2401 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
2402           targetNamespace="http://foo.com"
2403           xmlns:foo="http://foo.com"
2404           name="MyValueReferences" >
2405
2406   <reference name="CustomerService"
2407     promote="MyValueServiceComponent/CustomerService">
2408     <interface.java interface="services.customer.CustomerService"/>
2409     <binding.sca/>
2410   </reference>
2411
2412   <reference name="StockQuoteService"
2413     promote="StockQuoteMediatorComponent">
2414     <interface.java
2415       interface="services.stockquote.StockQuoteService"/>
2416     <binding.ws wsdlElement="http://www.stockquote.org/StockQuoteService#
2417       wsdl.port (StockQuoteService/StockQuoteServiceSOAP)"/>
2418   </reference>
2419
2420 </composite>
```

2421 *Snippet 5-23: Example Partial composite with Only references*

2422

2423 Snippet 5-24 shows the content of the MyValueWires.composite file.

2424

```
2425 <?xml version="1.0" encoding="ASCII"?>
2426 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
```

```

2427         targetNamespace="http://foo.com"
2428         xmlns:foo="http://foo.com"
2429         name="MyValueWires" >
2430
2431     <wire source="MyValueServiceComponent/stockQuoteService"
2432         target="StockQuoteMediatorComponent"/>
2433
2434 </composite>

```

2435 *Snippet 5-24: Example Partial composite with Only a wire*

2436 5.7 Composites which Contain Component Implementations of 2437 Multiple Types

2438 A Composite containing multiple components can have multiple component implementation types. For
2439 example, a Composite can contain one component with a Java POJO as its implementation and another
2440 component with a BPEL process as its implementation.

2441 5.8 Structural URI of Components

2442 The **structural URI** is a relative URI that describes each use of a given component in the Domain,
2443 relative to the URI of the Domain itself. It is never specified explicitly, but it calculated from the
2444 configuration of the components configured into the Domain.

2445 A component in a composite can be used more than once in the Domain, if its containing composite is
2446 used as the implementation of more than one higher-level component. The structural URI is used to
2447 separately identify each use of a component - for example, the structural URI can be used to attach
2448 different policies to each separate use of a component.

2449 For components directly deployed into the Domain, the structural URI is simply the name of the
2450 component.

2451 Where components are nested within a composite which is used as the implementation of a higher level
2452 component, the structural URI consists of the name of the nested component prepended with each of the
2453 names of the components upto and including the Domain level component.

2454 For example, consider a component named Component1 at the Domain level, where its implementation is
2455 Composite1 which in turn contains a component named Component2, which is implemented by
2456 Composite2 which contains a component named Component3. The three components in this example
2457 have the following structural URIs:

- 2458 1. Component1: Component1
- 2459 2. Component2: Component1/Component2
- 2460 3. Component3: Component1/Component2/Component3

2461 The structural URI can also be extended to refer to specific parts of a component, such as a service or a
2462 reference, by appending an appropriate fragment identifier to the component's structural URI, as follows:

- 2463 • Service:
2464 #service(servicename)
- 2465 • Reference:
2466 #reference(referencename)
- 2467 • Service binding:
2468 #service-binding(servicename/bindingname)
- 2469 • Reference binding:
2470 #reference-binding(referencename/bindingname)

2471 So, for example, the structural URI of the service named "testservice" of component "Component1" is
2472 Component1#service(testservice).

2473

6 Interface

2474 **Interfaces** define one or more business functions. These business functions are provided by Services
2475 and are used by References. A Service offers the business functionality of exactly one interface for use
2476 by other components. Each interface defines one or more service **operations** and each operation has
2477 zero or one **request (input) message** and zero or one **response (output) message**. The request and
2478 response messages can be simple types such as a string value or they can be complex types.

2479 SCA currently supports the following interface type systems:

- 2480 • Java interfaces
- 2481 • WSDL 1.1 portTypes ([Web Services Definition Language \[WSDL-11\]](#))
- 2482 • C++ classes
- 2483 • Collections of 'C' functions

2484 SCA is also extensible in terms of interface types. Support for other interface type systems can be added
2485 through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2486 Snippet 6-1 shows the pseudo-schema for the **interface** base element:

2487

```
2488 <interface remotable="boolean"? requires="list of xs:QName"?  
2489     policySets="list of xs:QName"?  
2490     <requires/*>  
2491     <policySetAttachment/*>  
2492 </interface>
```

2493 *Snippet 6-1: interface Pseudo-Schema*

2494

2495 The **interface** base element has the **attributes**:

- 2496 • **remotable : boolean (0..1)** – indicates whether an interface is remotable or not (see [the section on](#)
2497 [Local and Remotable interfaces](#)). A value of “true” means the interface is remotable, and a value of
2498 “false” means it is not. The @remotable attribute has no default value. This attribute is used as an
2499 alternative to interface type specific mechanisms such as the @Remotable annotation on a Java
2500 interface. The remotable nature of an interface in the absence of this attribute is interface type
2501 specific. The rules governing how this attribute relates to interface type specific mechanisms are
2502 defined by each interface type. When specified on an interface definition which includes a callback,
2503 this attribute also applies to the callback interface (see [the section on Bidirectional Interfaces](#)).
- 2504 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
2505 [\[SCA-POLICY\]](#) for a description of this attribute
- 2506 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
2507 [\[SCA-POLICY\]](#) for a description of this attribute.

2508 The **interface** element has the following **subelements**:

- 2509 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2510 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 2511 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
2512 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2513 description of this element.

2514 For information about Java interfaces, including details of SCA-specific annotations, see the SCA Java
2515 Common Annotations and APIs specification [\[SCA-Common-Java\]](#).

2516 For information about WSDL interfaces, including details of SCA-specific extensions, see SCA-Specific
2517 Aspects for WSDL Interfaces and WSDL Interface Type.

2518 For information about C++ interfaces, see the SCA C++ Client and Implementation Model specification
2519 [SCA-CPP-Client].
2520 For information about C interfaces, see the SCA C Client and Implementation Model specification [SCA-
2521 C-Client].

2522 6.1 Local and Remotable Interfaces

2523 A remotable service is one which can be called by a client which is running in an operating system
2524 process different from that of the service itself (this also applies to clients running on different machines
2525 from the service). Whether a service of a component implementation is remotable is defined by the
2526 interface of the service. WSDL defined interfaces are always remotable. See the relevant specifications
2527 for details of interfaces defined using other languages.

2528 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**
2529 interactions. **Remotable service Interfaces MUST NOT make use of *method or operation overloading*.**
2530 **[ASM80002]** This restriction on operation overloading for remotable services aligns with the WSDL 2.0
2531 specification, which disallows operation overloading, and also with the WS-I Basic Profile 1.1 (section
2532 4.5.3 - R2304) which has a constraint which disallows operation overloading when using WSDL 1.1.
2533 Independent of whether the remotable service is called remotely from outside the process where the
2534 service runs or from another component running in the same process, the data exchange semantics are
2535 **by-value**.

2536 Implementations of remotable services can modify input messages (parameters) during or after an
2537 invocation and can modify return messages (results) after the invocation. **If a remotable service is called**
2538 **locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the**
2539 **service or post-invocation modifications to return messages are seen by the caller. [ASM80003]**

2540 Snippet 6-2 shows an example of a remotable java interface:

```
2541  
2542 package services.hello;  
2543  
2544 @Remotable  
2545 public interface HelloService {  
2546  
2547     String hello(String message);  
2548 }
```

2549 *Snippet 6-2: Example remotable interface*

2550

2551 It is possible for the implementation of a remotable service to indicate that it can be called using by-
2552 reference data exchange semantics when it is called from a component in the same process. This can be
2553 used to improve performance for service invocations between components that run in the same process.
2554 This can be done using the @AllowsPassByReference annotation (see the [Java Client and](#)
2555 [Implementation Specification](#)).

2556 A service typed by a local interface can only be called by clients that are running in the same process as
2557 the component that implements the local service. Local services cannot be published via remotable
2558 services of a containing composite. In the case of Java a local service is defined by a Java interface
2559 definition without a **@Remotable** annotation.

2560 The style of local interfaces is typically **fine grained** and intended for **tightly coupled** interactions. Local
2561 service interfaces can make use of **method or operation overloading**.

2562 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

2563 6.2 Interface Compatibility

2564 The **compatibility** of two interfaces is defined in this section and these definitions are used throughout
2565 this specification. Three forms of compatibility are defined:

- 2566 • Compatible interfaces

2567 • Compatible subset

2568 • Compatible superset

2569 Note that WSDL 1.1 message parts can point to an XML Schema element declaration or to an XML
2570 Schema types. When determining compatibility between two WSDL operations, a message part that
2571 points to an XML Schema element declaration is considered to be incompatible with a message part that
2572 points to an XML Schema type.

2573 6.2.1 Compatible Interfaces

2574 An interface A is **Compatible** with a second interface B if and only if all of points 1 through 7 in the
2575 following list apply:

2576 1. interfaces A and B are either both remotable or else both local

2577 2. the set of operations in interface A is the same as the set of operations in
2578 interface B

2579 3. compatibility for individual operations of the interfaces A and B is defined as
2580 compatibility of the signature, i.e., the operation name, the input types, and the
2581 output types are the same

2582 4. the order of the input and output types for each operation in interface A is the
2583 same as the order of the input and output types for the corresponding operation
2584 in interface B

2585 5. the set of Faults and Exceptions expected by each operation in interface A is the
2586 same as the set of Faults and Exceptions specified by the corresponding
2587 operation in interface B

2588 6. for checking the compatibility of 2 remotable interfaces which are in different
2589 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
2590 compatibility checking is done between the WSDL 1.1 mapped interfaces.

2591 For checking the compatibility of 2 local interfaces which are in different interface
2592 languages, the method of checking compatibility is defined by the specifications
2593 which define those interface types, which must define mapping rules for the 2
2594 interface types concerned.
2595

2596 7. if either interface A or interface B declares a callback interface then both interface
2597 A and interface B declare callback interfaces and the callback interface declared
2598 on interface A is compatible with the callback interface declared on interface B,
2599 according to points 1 through 6 above

2600 6.2.2 Compatible Subset

2601 An interface A is a **Compatible Subset** of a second interface B if and only if all of points 1 through 7 in
2602 the following list apply:

2603 1. interfaces A and B are either both remotable or else both local

2604 2. the set of operations in interface A is the same as or is a subset of the set of
2605 operations in interface B

2606 3. compatibility for individual operations of the interfaces A and B is defined as
2607 compatibility of the signature, i.e., the operation name, the input types, and the
2608 output types are the same

2609 4. the order of the input and output types for each operation in interface A is the
2610 same as the order of the input and output types for the corresponding operation
2611 in interface B

- 2612 5. the set of Faults and Exceptions expected by each operation in interface A is the
2613 same as or is a superset of the set of Faults and Exceptions specified by the
2614 corresponding operation in interface B
- 2615 6. for checking the compatibility of 2 remotable interfaces which are in different
2616 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
2617 compatibility checking is done between the WSDL 1.1 mapped interfaces.
2618
2619 For checking the compatibility of 2 local interfaces which are in different interface
2620 languages, the method of checking compatibility is defined by the specifications
2621 which define those interface types, which must define mapping rules for the 2
2622 interface types concerned.
- 2623 7. if either interface A or interface B declares a callback interface then both interface
2624 A and interface B declare callback interfaces and the callback interface declared
2625 on interface B is a compatible subset of the callback interface declared on
2626 interface A, according to points 1 through 6 above

2627 6.2.3 Compatible Superset

2628 An interface A is a **Compatible Superset** of a second interface B if and only if all of points 1 through 7 in
2629 the following list apply:

- 2630 1. interfaces A and B are either both remotable or else both local
- 2631 2. the set of operations in interface A is the same as or is a superset of the set of
2632 operations in interface B
- 2633 3. compatibility for individual operations of the interfaces A and B is defined as
2634 compatibility of the signature, i.e., the operation name, the input types, and the
2635 output types are the same
- 2636 4. the order of the input and output types for each operation in interface B is the
2637 same as the order of the input and output types for the corresponding operation
2638 in interface A
- 2639 5. the set of Faults and Exceptions expected by each operation in interface A is the
2640 same as or is a subset of the set of Faults and Exceptions specified by the
2641 corresponding operation in interface B
- 2642 6. for checking the compatibility of 2 remotable interfaces which are in different
2643 interface languages, both are mapped to WSDL 1.1 (if not already WSDL 1.1) and
2644 compatibility checking is done between the WSDL 1.1 mapped interfaces.
2645
2646 For checking the compatibility of 2 local interfaces which are in different interface
2647 languages, the method of checking compatibility is defined by the specifications
2648 which define those interface types, which must define mapping rules for the 2
2649 interface types concerned.
- 2650 7. if either interface A or interface B declares a callback interface then both interface
2651 A and interface B declare callback interfaces and the callback interface declared
2652 on interface B is a compatible superset of the callback interface declared on
2653 interface A, according to points 1 through 6 above

2654 6.3 Bidirectional Interfaces

2655 The relationship of a business service to another business service is often peer-to-peer, requiring a two-
2656 way dependency at the service level. In other words, a business service represents both a consumer of a
2657 service provided by a partner business service and a provider of a service to the partner business

2658 service. This is especially the case when the interactions are based on asynchronous messaging rather
2659 than on remote procedure calls. The notion of **bidirectional interfaces** is used in SCA to directly model
2660 peer-to-peer bidirectional business service relationships.

2661 An interface element for a particular interface type system needs to allow the specification of a callback
2662 interface. If a callback interface is specified, SCA refers to the interface as a whole as a bidirectional
2663 interface.

2664 Snippet 6-3 shows the interface element defined using Java interfaces with a @callbackInterface
2665 attribute.

2666

```
2667 <interface.java interface="services.invoicing.ComputePrice"  
2668     callbackInterface="services.invoicing.InvoiceCallback"/>
```

2669 *Snippet 6-3: Example interface with a callback*

2670

2671 If a service is defined using a bidirectional interface element then its implementation implements the
2672 interface, and its implementation uses the callback interface to converse with the client that called the
2673 service interface.

2674 If a reference is defined using a bidirectional interface element, the client component implementation
2675 using the reference calls the referenced service using the interface. The client MUST provide an
2676 implementation of the callback interface. [ASM80004]

2677 Callbacks can be used for both remotable and local services. Either both interfaces of a bidirectional
2678 service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and
2679 remote services. [ASM80005]

2680 Note that an interface document such as a WSDL file or a Java interface can contain annotations that
2681 declare a callback interface for a particular interface (see [the section on WSDL Interface type](#) and the
2682 Java Common Annotations and APIs specification [SCA-Common-Java]). Whenever an interface
2683 document declaring a callback interface is used in the declaration of an <interface/> element in SCA, it
2684 MUST be treated as being bidirectional with the declared callback interface. [ASM80010] In such cases,
2685 there is no requirement for the <interface/> element to declare the callback interface explicitly.

2686 If an <interface/> element references an interface document which declares a callback interface and also
2687 itself contains a declaration of a callback interface, the two callback interfaces MUST be compatible.
2688 [ASM80011]

2689 See [the section on Interface Compatibility](#) for a definition of "compatible interfaces".

2690 In a bidirectional interface, the service interface can have more than one operation defined, and the
2691 callback interface can also have more than one operation defined. SCA runtimes MUST allow an
2692 invocation of any operation on the service interface to be followed by zero, one or many invocations of
2693 any of the operations on the callback interface. [ASM80009] These callback operations can be invoked
2694 either before or after the operation on the service interface has returned a response message, if there is
2695 one.

2696 For a given invocation of a service operation, which operations are invoked on the callback interface,
2697 when these are invoked, the number of operations invoked, and their sequence are not described by
2698 SCA. It is possible that this metadata about the bidirectional interface can be supplied through
2699 mechanisms outside SCA. For example, it might be provided as a written description attached to the
2700 callback interface.

2701 6.4 Long-running Request-Response Operations

2702 6.4.1 Background

2703 A service offering one or more operations which map to a WSDL request-response pattern might be
2704 implemented in a long-running, potentially interruptible, way. Consider a BPEL process with receive and
2705 reply activities referencing the WSDL request-response operation. Between the two activities, the
2706 business process logic could be a long-running sequence of steps, including activities causing the

2707 process to be interrupted. Typical examples are steps where the process waits for another message to
2708 arrive or a specified time interval to expire, or the process performs asynchronous interactions such as
2709 service invocations bound to asynchronous protocols or user interactions. This is a common situation in
2710 business processes, and it causes the implementation of the WSDL request-response operation to run for
2711 a very long time, e.g., several months (!). In this case, it is not meaningful for any caller to remain in a
2712 synchronous wait for the response while blocking system resources or holding database locks.

2713 Note that it is possible to model long-running interactions as a pair of two independent operations as
2714 described in the section on bidirectional interfaces. However, it is a common practice (and in fact much
2715 more convenient) to model a request-response operation and let the infrastructure deal with the
2716 asynchronous message delivery and correlation aspects instead of putting this burden on the application
2717 developer.

2718 **6.4.2 Definition of "long-running"**

2719 A request-response operation is considered long-running if the implementation does not guarantee the
2720 delivery of the response within any specified time interval. Clients invoking such request-response
2721 operations are strongly discouraged from making assumptions about when the response can be
2722 expected.

2723 **6.4.3 The asyncInvocation Intent**

2724 This specification permits a long-running request-response operation or a complete interface containing
2725 such operations to be marked using a policy intent with the name *asyncInvocation*. It is also possible for
2726 a service to set the *asyncInvocation*. intent when using an interface which is not marked with the
2727 *asyncInvocation*. intent. This can be useful when reusing an existing interface definition that does not
2728 contain SCA information.

2729 **6.4.4 Requirements on Bindings**

2730 In order to support a service operation which is marked with the *asyncInvocation* intent, it is necessary for
2731 the binding (and its associated policies) to support separate handling of the request message and the
2732 response message. Bindings which only support a synchronous style of message handling, such as a
2733 conventional HTTP binding, cannot be used to support long-running operations.

2734 The requirements on a binding to support the *asyncInvocation* intent are the same as those to support
2735 services with bidirectional interfaces - namely that the binding needs to be able to treat the transmission
2736 of the request message separately from the transmission of the response message, with an arbitrarily
2737 large time interval between the two transmissions.

2738 An example of a binding/policy combination that supports long-running request-response operations is a
2739 Web service binding used in conjunction with the WS-Addressing "wsam:NonAnonymousResponses"
2740 assertion.

2741 **6.4.5 Implementation Type Support**

2742 SCA implementation types can provide special asynchronous client-side and asynchronous server-side
2743 mappings to assist in the development of services and clients for long-running request-response
2744 operations.

2745 **6.5 SCA-Specific Aspects for WSDL Interfaces**

2746 There are a number of aspects that SCA applies to interfaces in general, such as marking them as having
2747 a callback interface. These aspects apply to the interfaces themselves, rather than their use in a specific
2748 place within SCA. There is thus a need to provide appropriate ways of marking the interface definitions
2749 themselves, which go beyond the basic facilities provided by the interface definition language.

2750 For WSDL interfaces, there is an extension mechanism that permits additional information to be included
2751 within the WSDL document. SCA takes advantage of this extension mechanism. In order to use the SCA

2752 extension mechanism, the SCA namespace (<http://docs.oasis-open.org/ns/opencsa/sca/200912>) needs
2753 to be declared within the WSDL document.

2754 First, SCA defines a global element in the SCA namespace which provides a mechanism to attach policy
2755 intents - **requires**. Snippet 6-4 shows the definition of the requires element:

2756

```
2757 <element name="requires">  
2758   <complexType>  
2759     <sequence minOccurs="0" maxOccurs="unbounded">  
2760       <any namespace="##other" processContents="lax"/>  
2761     </sequence>  
2762     <attribute name="intents" type="sca:listOfQNames" use="required"/>  
2763     <anyAttribute namespace="##other" processContents="lax"/>  
2764   </complexType>  
2765 </element>  
  
2766  
2767 <simpleType name="listOfQNames">  
2768   <list itemType="QName"/>  
2769 </simpleType>
```

2770 *Snippet 6-4: requires WSDL extension definition*

2771

2772 The requires element can be used as a subelement of the WSDL portType and operation elements. The
2773 element contains one or more intent names, as defined by the [Policy Framework specification \[SCA-](#)
2774 [POLICY\]](#). Any service or reference that uses an interface marked with intents MUST implicitly add those
2775 intents to its own @requires list. [ASM80008]

2776 SCA defines an attribute which is used to indicate that a given WSDL portType element (WSDL 1.1) has
2777 an associated callback interface. This is the @callback attribute, which applies to a WSDL portType
2778 element.

2779 Snippet 6-5 shows the definition of the @callback attribute:

2780

```
2781 <attribute name="callback" type="QName"/>
```

2782 *Snippet 6-5: callback WSDL extension definition*

2783

2784 The value of the @callback attribute is the QName of a portType. The portType declared by the
2785 @callback attribute is the callback interface to use for the portType which is annotated by the
2786 @callback attribute.

2787 Snippet 6-6 is an example of a portType element with a @callback attribute:

2788

```
2789 <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
2790   <operation name="apply">  
2791     <input message="tns:ApplicationInput"/>  
2792     <output message="tns:ApplicationOutput"/>  
2793   </operation>  
2794   ...  
2795 </portType>
```

2796 *Snippet 6-6: Example use of @callback*

2797 6.6 WSDL Interface Type

2798 The WSDL interface type is used to declare interfaces for services and for references, where the interface
2799 is defined in terms of a WSDL document. An interface is defined in terms of a WSDL 1.1 portType with
2800 the arguments and return of the service operations described using XML schema.

2801 A WSDL interface is declared by an **interface.wsdl** element. Snippet 6-7 shows the pseudo-schema for
2802 the interface.wsdl element:

2803

```
2804 <!-- WSDL Interface schema snippet -->
2805 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?
2806     remotable="xs:boolean"?
2807     requires="listOfQNames"?
2808     policySets="listOfQNames">
2809     <requires/*>
2810     <policySetAttachment/*>
2811 </interface.wSDL>
```

2812 *Snippet 6-7: interface.wSDL Pseudo-Schema*

2813

2814 The **interface.wSDL** element has the **attributes**:

- 2815 • **interface : uri (1..1)** - the URI of a WSDL portType

2816 The **interface.wSDL** @**interface** attribute MUST reference a portType of a WSDL 1.1 document.
2817 [\[ASM80001\]](#)

- 2818 • **callbackInterface : uri (0..1)** - a callback interface, which is the URI of a WSDL portType

2819 The **interface.wSDL** @**callbackInterface** attribute, if present, MUST reference a portType of a WSDL
2820 1.1 document. [\[ASM80016\]](#)

- 2821 • **remotable : boolean (0..1)** – indicates whether the interface is remotable or not. @remotable has a
2822 default value of true. WSDL interfaces are always remotable and therefore an <interface.wSDL/>
2823 element MUST NOT contain remotable="false". [\[ASM80017\]](#)

- 2824 • **requires : listOfQNames (0..1)** – a list of policy intents. See the [Policy Framework specification](#)
2825 [\[SCA-POLICY\]](#) for a description of this attribute.

- 2826 • **policySets : listOfQNames (0..1)** – a list of policy sets. See the [Policy Framework specification](#)
2827 [\[SCA-POLICY\]](#) for a description of this attribute.

2828 The form of the URI for WSDL portTypes follows the syntax described in the WSDL 1.1 Element
2829 Identifiers specification [\[WSDL11_Identifiers\]](#)

2830 The **interface.wSDL** element has the following **subelements**:

- 2831 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2832 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.

- 2833 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
2834 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2835 description of this element.

2836 6.6.1 Example of interface.wSDL

2837 Snippet 6-8 shows an interface defined by the WSDL portType "StockQuote" with a callback interface
2838 defined by the "StockQuoteCallback" portType.

2839

```
2840 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#
2841     wSDL.porttype(StockQuote)"
2842     callbackInterface="http://www.stockquote.org/StockQuoteService#
2843     wSDL.porttype(StockQuoteCallback)"/>
```

2844 *Snippet 6-8: Example interface.wSDL*

2845 7 Binding

2846 Bindings are used by services and references. References use bindings to describe the access
2847 mechanism used to call a service (which can be a service provided by another SCA composite). Services
2848 use bindings to describe the access mechanism that clients (which can be a client from another SCA
2849 composite) have to use to call the service.

2850 SCA supports the use of multiple different types of bindings. Examples include **SCA service, Web**
2851 **service, stateless session EJB, database stored procedure, EIS service**. SCA provides an
2852 extensibility mechanism by which an SCA runtime can add support for additional binding types. For
2853 details on how additional binding types are defined, see the section on the Extension Model.

2854 A binding is defined by a **binding element** which is a child element of a service or of a reference element
2855 in a composite. Snippet 7-1 shows the composite pseudo-schema with the pseudo-schema for the
2856 binding element.

```
2857 <?xml version="1.0" encoding="ASCII"?>
2858 <!-- Bindings schema snippet -->
2859 <composite ... >
2860   ...
2861     <service ... >*
2862     <interface ... />?
2863     <binding uri="xs:anyURI"? name="xs:NCName"?
2864       requires="list of xs:QName"?
2865       policySets="list of xs:QName"?>*
2866       <wireFormat/>?
2867       <operationSelector/>?
2868       <requires/>*
2869       <policySetAttachment/>*
2870     </binding>
2871     <callback>?
2872       <binding uri="xs:anyURI"? name="xs:NCName"?
2873         requires="list of xs:QName"?
2874         policySets="list of xs:QName"?>+
2875         <wireFormat/>?
2876         <operationSelector/>?
2877         <requires/>*
2878         <policySetAttachment/>*
2879       </binding>
2880     </callback>
2881   </service>
2882   ...
2883   <reference ... >*
2884     <interface ... />?
2885     <binding uri="xs:anyURI"? name="xs:NCName"?
2886       requires="list of xs:QName"?
2887       policySets="list of xs:QName"?>*
2888       <wireFormat/>?
2889       <operationSelector/>?
2890       <requires/>*
2891       <policySetAttachment/>*
2892     </binding>
2893     <callback>?
2894       <binding uri="xs:anyURI"? name="xs:NCName"?
2895         requires="list of xs:QName"?
2896         policySets="list of xs:QName"?>+
2897         <wireFormat/>?
2898         <operationSelector/>?
2899         <requires/>*
2900         <policySetAttachment/>*
2901     </binding>
```

2902
2903
2904
2905

```
</callback>  
</reference>  
...  
</composite>
```

2906 *Snippet 7-1: composite Pseudo-Schema with binding Child element*

2907

2908 The element name of the binding element is architected; it is in itself a qualified name. The first qualifier is
2909 always named “binding”, and the second qualifier names the respective binding-type (e.g. binding.sca,
2910 binding.ws, binding.ejb, binding.eis).

2911 A **binding** element has the attributes:

- 2912 • **uri (0..1)** - has the semantic:
 - 2913 – The @uri attribute can be omitted.
 - 2914 – For a binding of a **reference** the @uri attribute defines the target URI of the reference. This
2915 MUST be either the componentName/serviceName/bindingName for a wire to an endpoint within
2916 the SCA Domain, or the accessible address of some service endpoint either inside or outside the
2917 SCA Domain (where the addressing scheme is defined by the type of the binding). [ASM90001]
 - 2918 – The circumstances under which the @uri attribute can be used are defined in section "[Specifying
2919 the Target Service\(s\) for a Reference.](#)"
 - 2920 – For a binding of a **service** the @uri attribute defines the bindingURI. If present, the bindingURI
2921 can be used by the binding as described in the section "[Form of the URI of a Deployed Binding](#)".
- 2922 • **name (0..1)** – a name for the binding instance (an NCName). The @name attribute allows distinction
2923 between multiple binding elements on a single service or reference. The default value of the @name
2924 attribute is the service or reference name. When a service or reference has multiple bindings, all non-
2925 callback bindings of the service or reference MUST have unique names, and all callback bindings of
2926 the service or reference MUST have unique names. [ASM90002] This uniqueness requirement
2927 implies that only one non-callback binding of a service or reference can have the default @name
2928 value, and only one callback binding of a service or reference can have the default @name value.
2929

2930 The @name also permits the binding instance to be referenced from elsewhere – particularly useful
2931 for some types of binding, which can be declared in a definitions document as a template and
2932 referenced from other binding instances, simplifying the definition of more complex binding instances
2933 (see the [JMS Binding specification \[SCA-JMSBINDING\]](#) for examples of this referencing).

- 2934 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2935 description of this attribute.
- 2936 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2937 description of this attribute.

2938 A **binding** element has the child elements:

- 2939 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See the [wireFormat
2940 section](#) for details.
- 2941 • **operationSelector(0..1)** - an operationSelector element that is used to match a particular message to
2942 a particular operation in the interface. See the [operationSelector section](#) for details
- 2943 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
2944 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 2945 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more
2946 policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
2947 description of this element.

2948 When multiple bindings exist for a service, it means that the service is available through any of the
2949 specified bindings. The technique that the SCA runtime uses to choose among available bindings is left
2950 to the implementation and it might include additional (nonstandard) configuration. Whatever technique is
2951 used needs to be documented by the runtime.

2952 Services and References can always have their bindings overridden at the SCA Domain level, unless
2953 restricted by Intents applied to them.

2954 If a reference has any bindings, they MUST be resolved, which means that each binding MUST include a
2955 value for the @uri attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired
2956 using other SCA mechanisms. [ASM90003] To specify constraints on the kinds of bindings that are
2957 acceptable for use with a reference, the user specifies either policy intents or policy sets.
2958

2959 Users can also specifically wire, not just to a component service, but to a specific binding offered by that
2960 target service. To wire to a specific binding of a target service the syntax
2961 "componentName/serviceName/bindingName" MUST be used. [ASM90004]

2962 The following sections describe the SCA and Web service binding type in detail.

2963 7.1 Messages containing Data not defined in the Service Interface

2964 It is possible for a message to include information that is not defined in the interface used to define the
2965 service, for instance information can be contained in SOAP headers or as MIME attachments.

2966 Implementation types can make this information available to component implementations in their
2967 execution context. The specifications for these implementation types describe how this information is
2968 accessed and in what form it is presented.

2969 7.2 WireFormat

2970 A wireFormat is the form that a data structure takes when it is transmitted using some communication
2971 binding. Another way to describe this is "the form that the data takes on the wire". A wireFormat can be
2972 specific to a given communication method, or it can be general, applying to many different communication
2973 methods. An example of a general wireFormat is XML text format.

2974 Where a particular SCA binding can accommodate transmitting data in more than one format, the
2975 configuration of the binding can include a definition of the wireFormat to use. This is done using an
2976 <sca:wireFormat/> subelement of the <binding/> element.

2977 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats to be
2978 the default wireFormat which applies if no <wireFormat/> subelement is present.

2979 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a
2980 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 2981 • <sca:wireFormat.xml/>
2982 A wireFormat that transmits the data as an XML text datastructure
- 2983 • <sca:wireFormat.jms/>
2984 The "default JMS wireFormat" as described in the JMS Binding specification

2985 Specific wireFormats can have elements that include either attributes or subelements or both.

2986 For details about specific wireFormats, see the related SCA Binding specifications.

2987 7.3 OperationSelector

2988 An operationSelector is necessary for some types of transport binding where messages are transmitted
2989 across the transport without any explicit relationship between the message and the interface operation to
2990 which it relates. SOAP is an example of a protocol where the messages do contain explicit information
2991 that relates each message to the operation it targets. However, other transport bindings have messages
2992 where this relationship is not expressed in the message or in any related headers (pure JMS messages,
2993 for example). In cases where the messages arrive at a service without any explicit information that maps
2994 them to specific operations, it is necessary for the metadata attached to the service binding to contain the
2995 mapping information. The information is held in an operationSelector element which is a child element of
2996 the binding element.

2997 The base sca:operationSelector element is abstract and it has no attributes and no child elements. For a
2998 particular operationSelector, an extension subtype is defined, using substitution groups, for example:

2999 • <sca:operationSelector.XPath/>
3000 An operation selector that uses XPath to filter out specific messages and target them to
3001 particular named operations.

3002 Specific operationSelectors can have elements that include either attributes or subelements or both.

3003 For details about specific operationSelectors, see the related SCA Binding specifications.

3004 **7.4 Form of the URI of a Deployed Binding**

3005 SCA Bindings specifications can choose to use the **structural URI** defined in the section "[Structural URI](#)
3006 [of Components](#)" above to derive a binding specific URI according to some Binding-related scheme. The
3007 relevant binding specification describes this.

3008 Alternatively, <binding/> elements have a @uri attribute, which is termed a bindingURI.

3009 If the bindingURI is specified on a given <binding/> element, the binding can use it to derive an endpoint
3010 URI relevant to the binding. The derivation is binding specific and is described by the relevant binding
3011 specification.

3012 For binding.sca, which is described in the SCA Assembly specification, this is as follows:

3013 • If the binding @uri attribute is specified on a reference, it identifies the target service in
3014 the SCA Domain by specifying the service's structural URI.

3015 • If the binding @uri attribute is specified on a service, it is ignored.

3016 **7.4.1 Non-hierarchical URIs**

3017 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) can make use of the @uri
3018 attribute, which is the complete representation of the URI for that service binding. Where the binding
3019 does not use the @uri attribute, the binding needs to offer a different mechanism for specifying the
3020 service address.

3021 **7.4.2 Determining the URI scheme of a deployed binding**

3022 One of the things that needs to be determined when building the effective URI of a deployed binding (i.e.
3023 endpoint) is the URI scheme. The process of determining the endpoint URI scheme is binding type
3024 specific.

3025 If the binding type supports a single protocol then there is only one URI scheme associated with it. In this
3026 case, that URI scheme is used.

3027 If the binding type supports multiple protocols, the binding type implementation determines the URI
3028 scheme by introspecting the binding configuration, which can include the policy sets associated with the
3029 binding.

3030 A good example of a binding type that supports multiple protocols is binding.ws, which can be configured
3031 by referencing either an "abstract" WSDL element (i.e. portType or interface) or a "concrete" WSDL
3032 element (i.e. binding or port). When the binding references a portType or Interface, the protocol and
3033 therefore the URI scheme is derived from the intents/policy sets attached to the binding. When the
3034 binding references a "concrete" WSDL element, there are two cases:

3035 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most
3036 common case. In this case, the URI scheme is given by the protocol/transport specified in the
3037 WSDL binding element.

3038 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,
3039 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"
3040 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined
3041 by looking at the policy sets attached to the binding.

3042 It is worth noting that an intent supported by a binding type can completely change the behavior of the
3043 binding. For example, when the intent "confidentiality/transport" is attached to an HTTP binding, SSL is
3044 turned on. This basically changes the URI scheme of the binding from "http" to "https".

3045

3046 7.5 SCA Binding

3047 Snippet Snippet 7-2 shows the SCA binding element pseudo-schema.

```

3048 <binding.sca uri="xs:anyURI"?
3049     name="xs:NCName"?
3050     requires="list of xs:QName"?
3051     policySets="list of xs:QName"?>
3052   <wireFormat/>?
3053   <operationSelector/>?
3054   <requires/>*
3055   <policySetAttachment/>*
3056 </binding.sca>

```

3057 *Snippet 7-2: binding.sca pseudo-schema*

3058

3059 A **binding.sca** element has the attributes:

- 3060 • **uri (0..1)** - has the semantic:
 - 3061 – The @uri attribute can be omitted.
 - 3062 – If a <binding.sca/> element of a component reference specifies a URI via its @uri attribute, then
 - 3063 this provides a wire to a target service provided by another component. The form of the URI
 - 3064 which points to the service of a component that is in the same composite as the source
 - 3065 component is as follows:

```

3066         <component-name>/<service-name>
3067     or
3068         <component-name>/<service-name>/<binding-name>

```

3070 in cases where the service has multiple bindings present.

- 3072 – The circumstances under which the @uri attribute can be used are defined in the section
- 3073 ["Specifying the Target Service\(s\) for a Reference."](#)
- 3074 – For a binding.sca of a component service, the @uri attribute MUST NOT be present. [ASM90005]
- 3075 • **name (0..1)** – a name for the binding instance (an NCName), as defined for the base <binding/>
- 3076 element type.
- 3077 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 3078 description of this attribute.
- 3079 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 3080 description of this attribute.

3081 A **binding.sca** element has the child elements:

- 3082 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. binding.sca does not
- 3083 define any specific wireFormat elements.
- 3084 • **operationSelector(0..1)** - an operationSelector element that is used to match a particular message to
- 3085 a particular operation in the interface. binding.sca does not define any specific operationSelector
- 3086 elements.
- 3087 • **requires : requires (0..n)** - A service element has **zero or more requires subelements**. See the
- 3088 [Policy Framework specification \[SCA-POLICY\]](#) for a description of this element.
- 3089 • **policySetAttachment : policySetAttachment (0..n)** - A service element has **zero or more**
- 3090 **policySetAttachment subelements**. See the [Policy Framework specification \[SCA-POLICY\]](#) for a
- 3091 description of this element.

3092 The SCA binding can be used for service interactions between references and services contained within
3093 the SCA Domain. The way in which this binding type is implemented is not defined by the SCA

3094 specification and it can be implemented in different ways by different SCA runtimes. The only requirement
3095 is that any specified qualities of service are implemented for the SCA binding type. Qualities of service for
3096 <binding.sca/> are expressed using intents and/or policy sets following the rules defined in [the SCA](#)
3097 [Policy specification \[SCA-POLICY\]](#).

3098 The SCA binding type is not intended to be an interoperable binding type. For interoperability, an
3099 interoperable binding type such as the Web service binding is used.

3100 An SCA runtime has to support the binding.sca binding type. See the section on [SCA Runtime](#)
3101 [conformance](#).

3102 A service definition with no binding element specified uses the SCA binding (see ASM50005 in section
3103 4.2 on Component Service). <binding.sca/> only has to be specified explicitly in override cases, or when
3104 a set of bindings is specified on a service definition and the SCA binding needs to be one of them.

3105 If a reference does not have a binding subelement specified, then the binding used is one of the bindings
3106 specified by the service provider, as long as the intents attached to the reference and the service are all
3107 honoured, as described in [the section on Component References](#).

3108 If the interface of the service or reference is local, then the local variant of the SCA binding will be used. If
3109 the interface of the service or reference is remotable, then either the local or remote variant of the SCA
3110 binding will be used depending on whether source and target are co-located or not.

3111 If a <binding.sca/> element of a <component/> <reference/> specifies a URI via its @uri attribute, then
3112 this provides a wire to a target service provided by another component.

3113 The form of the URI which points to the service of a component that is in the same composite as the
3114 source component is as follows:

- 3115 • <domain-component-name>/<service-name>

3116 7.5.1 Example SCA Binding

3117 Snippet 7-3 shows the MyValueComposite.composite file for the MyValueComposite containing the
3118 service element for the MyValueService and a reference element for the StockQuoteService. Both the
3119 service and the reference use an SCA binding. The target for the reference is left undefined in this
3120 binding and would have to be supplied by the composite in which this composite is used.

```
3121 <?xml version="1.0" encoding="ASCII"?>
3122 <!-- Binding SCA example -->
3123 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3124           targetNamespace="http://foo.com"
3125           name="MyValueComposite" >
3126
3127     <service name="MyValueService" promote="MyValueComponent">
3128       <interface.java interface="services.myvalue.MyValueService"/>
3129       <binding.sca/>
3130       ...
3131     </service>
3132
3133     ...
3134
3135     <reference name="StockQuoteService"
3136               promote="MyValueComponent/StockQuoteReference">
3137       <interface.java interface="services.stockquote.StockQuoteService"/>
3138       <binding.sca/>
3139     </reference>
3140
3141 </composite>
```

3142 *Snippet 7-3: Example binding.sca*

3143 7.6 Web Service Binding

3144 SCA defines a Web services binding. This is described in [a separate specification document \[SCA-](#)
3145 [WSBINDING\]](#).

3146 **7.7 JMS Binding**

3147 SCA defines a JMS binding. This is described in [a separate specification document \[SCA-JMSBINDING\]](#).

3148 8 SCA Definitions

3149 There are a variety of SCA artifacts which are generally useful and which are not specific to a particular
3150 composite or a particular component. These shared artifacts include intents, policy sets, binding type
3151 definitions, implementation type definitions, and external attachment definitions.

3152 All of these artifacts within an SCA Domain are defined in SCA contributions in files called META-
3153 INF/definitions.xml (relative to the contribution base URI). An SCA runtime MUST make available to the
3154 Domain all the artifacts contained within the definitions.xml files in the Domain. [ASM10002] An SCA
3155 runtime MUST reject a definitions.xml file that does not conform to the sca-definitions.xsd schema.
3156 [ASM10003]

3157 Although the definitions are specified within a single SCA contribution, the definitions are visible
3158 throughout the Domain. Because of this, all of the QNames for the definitions contained in definitions.xml
3159 files MUST be unique within the Domain.. [ASM10001] The definitions.xml file contains a definitions
3160 element that conforms to the pseudo-schema shown in Snippet 8-1:

3161

```
3162 <?xml version="1.0" encoding="ASCII"?>
3163 <!-- Composite schema snippet -->
3164 <definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3165             targetNamespace="xs:anyURI">
3166
3167     <sca:intent/*>
3168
3169     <sca:policySet/*>
3170
3171     <sca:bindingType/*>
3172
3173     <sca:implementationType/*>
3174
3175     <sca:externalAttachment/*>
3176
3177 </definitions>
```

3178 *Snippet 8-1: definitions Pseudo-Schema*

3179

3180 The definitions element has the attribute:

- 3181 • **targetNamespace (1..1)** – the namespace into which the child elements of this definitions element
3182 are placed (used for artifact resolution)

3183 The definitions element contains child elements – intent, policySet, bindingType, implementationType and
3184 externalAttachment. These elements are described elsewhere in this specification or in [the SCA Policy
3185 Framework specification \[SCA-POLICY\]](#).

3186

9 Extension Model

3187 The assembly model can be extended with support for new interface types, implementation types and
3188 binding types. The extension model is based on XML schema substitution groups. There are three XML
3189 Schema substitution group heads defined in the SCA namespace: **interface**, **implementation** and
3190 **binding**, for interface types, implementation types and binding types, respectively.

3191 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1], [SCA-
3192 WSBINDING], [11]) use these XML Schema substitution groups to define some basic types of interfaces,
3193 implementations and bindings, but additional types can be defined as needed, where support for these
3194 extra ones is available from the runtime. The interface type elements, implementation type elements, and
3195 binding type elements defined by the SCA specifications are all part of the SCA namespace
3196 ("http://docs.oasis-open.org/ns/opencsa/sca/200912"), as indicated in their respective schemas. New
3197 interface types, implementation types and binding types that are defined using this extensibility model,
3198 which are not part of these SCA specifications are defined in namespaces other than the SCA
3199 namespace.

3200 The "." notation is used in naming elements defined by the SCA specifications (e.g. <implementation.java
3201 ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel extensibility approach but as a naming
3202 convention that improves usability of the SCA assembly language.

3203 **Note:** How to contribute SCA model extensions and their runtime function to an SCA runtime will be
3204 defined by a future version of the specification.

9.1 Defining an Interface Type

3206 Snippet 9-1 shows the base definition for the **interface** element and **Interface** type contained in **sca-
3207 core.xsd**; see sca-core.xsd for the complete schema.

3208

```
3209 <?xml version="1.0" encoding="UTF-8"?>
3210 <!-- (c) Copyright SCA Collaboration 2006 -->
3211 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3212         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3213         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3214         elementFormDefault="qualified">
3215
3216     ...
3217
3218     <element name="interface" type="sca:Interface" abstract="true"/>
3219     <complexType name="Interface" abstract="true">
3220         <choice minOccurs="0" maxOccurs="unbounded">
3221             <element ref="sca:requires"/>
3222             <element ref="sca:policySetAttachment"/>
3223         </choice>
3224         <attribute name="remotable" type="boolean" use="optional"/>
3225         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3226         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3227     </complexType>
3228
3229     ...
3230
3231 </schema>
```

3232 *Snippet 9-1: interface and Interface Schema*

3233

3234 Snippet 9-2 is an example of how the base definition is extended to support Java interfaces. The snippet
3235 shows the definition of the **interface.java** element and the **JavaInterface** type contained in **sca-**
3236 **interface-java.xsd**.

3237

```
3238 <?xml version="1.0" encoding="UTF-8"?>
3239 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3240         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3241         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
3242
3243     <element name="interface.java" type="sca:JavaInterface"
3244             substitutionGroup="sca:interface"/>
3245     <complexType name="JavaInterface">
3246         <complexContent>
3247             <extension base="sca:Interface">
3248                 <attribute name="interface" type="NCName"
3249                         use="required"/>
3250             </extension>
3251         </complexContent>
3252     </complexType>
3253 </schema>
```

3254 *Snippet 9-2: Extending interface to interface.java*

3255

3256 Snippet 9-3 is an example of how the base definition can be extended by other specifications to support a
3257 new interface not defined in the SCA specifications. The snippet shows the definition of the **my-interface-**
3258 **extension** element and the **my-interface-extension-type** type.

3259

```
3260 <?xml version="1.0" encoding="UTF-8"?>
3261 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3262         targetNamespace="http://www.example.org/myextension"
3263         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3264         xmlns:tns="http://www.example.org/myextension">
3265
3266     <element name="my-interface-extension"
3267             type="tns:my-interface-extension-type"
3268             substitutionGroup="sca:interface"/>
3269     <complexType name="my-interface-extension-type">
3270         <complexContent>
3271             <extension base="sca:Interface">
3272                 ...
3273             </extension>
3274         </complexContent>
3275     </complexType>
3276 </schema>
```

3277 *Snippet 9-3: Example interface extension*

3278 9.2 Defining an Implementation Type

3279 Snippet 9-4 shows the base definition for the **implementation** element and **Implementation** type
3280 contained in **sca-core.xsd**; see **sca-core.xsd** for complete schema.

3281

```
3282 <?xml version="1.0" encoding="UTF-8"?>
3283 <!-- (c) Copyright SCA Collaboration 2006 -->
3284 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3285         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3286         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3287         elementFormDefault="qualified">
3288
3289     ...
```

```

3290
3291     <element name="implementation" type="sca:Implementation"
3292         abstract="true"/>
3293     <complexType name="Implementation" abstract="true">
3294         <complexContent>
3295             <extension base="sca:CommonExtensionBase">
3296                 <choice minOccurs="0" maxOccurs="unbounded">
3297                     <element ref="sca:requires"/>
3298                     <element ref="sca:policySetAttachment"/>
3299                 </choice>
3300                 <attribute name="requires" type="sca:listOfQNames"
3301                     use="optional"/>
3302                 <attribute name="policySets" type="sca:listOfQNames"
3303                     use="optional"/>
3304             </extension>
3305         </complexContent>
3306     </complexType>
3307
3308     ...
3309
3310 </schema>

```

3311 *Snippet 9-4: implementation and Implementation Schema*

3312

3313 Snippet 9-5 shows how the base definition is extended to support Java implementation. The snippet
3314 shows the definition of the **implementation.java** element and the **JavaImplementation** type contained in
3315 **sca-implementation-java.xsd**.

3316

```

3317 <?xml version="1.0" encoding="UTF-8"?>
3318 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3319     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3320     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
3321
3322     <element name="implementation.java" type="sca:JavaImplementation"
3323         substitutionGroup="sca:implementation"/>
3324         <complexType name="JavaImplementation">
3325             <complexContent>
3326                 <extension base="sca:Implementation">
3327                     <attribute name="class" type="NCName"
3328                         use="required"/>
3329                 </extension>
3330             </complexContent>
3331         </complexType>
3332     </schema>

```

3333 *Snippet 9-5: Extending implementation to implementation.java*

3334

3335 Snippet 9-6 is an example of how the base definition can be extended by other specifications to support a
3336 new implementation type not defined in the SCA specifications. The snippet shows the definition of the
3337 **my-impl-extension** element and the **my-impl-extension-type** type.

3338

```

3339 <?xml version="1.0" encoding="UTF-8"?>
3340 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3341     targetNamespace="http://www.example.org/myextension"
3342     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3343     xmlns:tns="http://www.example.org/myextension">
3344
3345     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3346         substitutionGroup="sca:implementation"/>

```

```

3347 <complexType name="my-impl-extension-type">
3348   <complexContent>
3349     <extension base="sca:Implementation">
3350       ...
3351     </extension>
3352   </complexContent>
3353 </complexType>
3354 </schema>

```

3355 *Snippet 9-6: Example implementation extension*

3356

3357 In addition to the definition for the new implementation instance element, there needs to be an associated
3358 implementationType element which provides metadata about the new implementation type. The pseudo
3359 schema for the implementationType element is shown in Snippet 9-7:

3360

```

3361 <implementationType type="xs:QName"
3362   alwaysProvides="list of intent xs:QName"
3363   mayProvide="list of intent xs:QName"/>

```

3364 *Snippet 9-7: implementationType Pseudo-Schema*

3365

3366 The implementation type has the attributes:

- 3367 • **type (1..1)** – the type of the implementation to which this implementationType element applies. This
3368 is intended to be the QName of the implementation element for the implementation type, such as
3369 "sca:implementation.java"
- 3370 • **alwaysProvides (0..1)** – a set of intents which the implementation type always provides. See [the](#)
3371 [Policy Framework specification \[SCA-POLICY\]](#) for details.
- 3372 • **mayProvide (0..1)** – a set of intents which the implementation type provides only when the intent is
3373 attached to the implementation element. See [the Policy Framework specification \[SCA-POLICY\]](#) for
3374 details.

3375 9.3 Defining a Binding Type

3376 Snippet 9-8 shows the base definition for the **binding** element and **Binding** type contained in **sca-**
3377 **core.xsd**; see sca-core.xsd for complete schema.

3378

```

3379 <?xml version="1.0" encoding="UTF-8"?>
3380 <!-- binding type schema snippet -->
3381 <!-- (c) Copyright SCA Collaboration 2006, 2009 -->
3382 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3383   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3384   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3385   elementFormDefault="qualified">
3386   ...
3387   ...
3388   <element name="binding" type="sca:Binding" abstract="true"/>
3389   <complexType name="Binding">
3390     <attribute name="uri" type="anyURI" use="optional"/>
3391     <attribute name="name" type="NCName" use="optional"/>
3392     <attribute name="requires" type="sca:listOfQNames"
3393       use="optional"/>
3394     <attribute name="policySets" type="sca:listOfQNames"
3395       use="optional"/>
3396   </complexType>
3397
3398

```

```
3399     ...
3400
3401 </schema>
```

3402 *Snippet 9-8: binding and Binding Schema*

3403

3404 Snippet 9-9 is an example of how the base definition is extended to support Web service binding. The
3405 snippet shows the definition of the **binding.ws** element and the **WebServiceBinding** type contained in
3406 **sca-binding-webservice.xsd**.

3407

```
3408 <?xml version="1.0" encoding="UTF-8"?>
3409 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3410         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3411         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
3412
3413     <element name="binding.ws" type="sca:WebServiceBinding"
3414 substitutionGroup="sca:binding"/>
3415     <complexType name="WebServiceBinding">
3416         <complexContent>
3417             <extension base="sca:Binding">
3418                 <attribute name="port" type="anyURI" use="required"/>
3419             </extension>
3420         </complexContent>
3421     </complexType>
3422 </schema>
```

3423 *Snippet 9-9: Extending binding to binding.ws*

3424

3425 Snippet 9-10 is an example of how the base definition can be extended by other specifications to support
3426 a new binding not defined in the SCA specifications. The snippet shows the definition of the **my-binding-**
3427 **extension** element and the **my-binding-extension-type** type.

3428

```
3429 <?xml version="1.0" encoding="UTF-8"?>
3430 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3431         targetNamespace="http://www.example.org/myextension"
3432         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3433         xmlns:tns="http://www.example.org/myextension">
3434
3435     <element name="my-binding-extension"
3436 type="tns:my-binding-extension-type"
3437 substitutionGroup="sca:binding"/>
3438     <complexType name="my-binding-extension-type">
3439         <complexContent>
3440             <extension base="sca:Binding">
3441                 ...
3442             </extension>
3443         </complexContent>
3444     </complexType>
3445 </schema>
```

3446 *Snippet 9-10: Example binding extension*

3447

3448 In addition to the definition for the new binding instance element, there needs to be an associated
3449 bindingType element which provides metadata about the new binding type. The pseudo schema for the
3450 bindingType element is shown in Snippet 9-11:

3451

```
3452 <bindingType type="xs:QName"
```

```
3453 alwaysProvides="list of intent QNames"?
3454 mayProvide = "list of intent QNames"?/>
```

3455 *Snippet 9-11: bindingType Pseudo-Schema*

3456

3457 The binding type has the following attributes:

- 3458 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is intended to be
- 3459 the QName of the binding element for the binding type, such as "sca:binding.ws"
- 3460 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See [the Policy](#)
- 3461 [Framework specification \[SCA-POLICY\]](#) for details.
- 3462 • **mayProvide (0..1)** – a set of intents which the binding type provides only when the intent is attached
- 3463 to the binding element. See [the Policy Framework specification \[SCA-POLICY\]](#) for details.

3464 9.4 Defining an Import Type

3465 Snippet 9-12 shows the base definition for the *import* element and *Import* type contained in *sca-*
3466 *core.xsd*; see sca-core.xsd for complete schema.

3467

```
3468 <?xml version="1.0" encoding="UTF-8"?>
3469 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved. OASIS trademark,
3470 IPR and other policies apply. -->
3471 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3472 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3473 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3474 elementFormDefault="qualified">
3475
3476 ...
3477
3478 <!-- Import -->
3479 <element name="importBase" type="sca:Import" abstract="true" />
3480 <complexType name="Import" abstract="true">
3481 <complexContent>
3482 <extension base="sca:CommonExtensionBase">
3483 <sequence>
3484 <any namespace="##other" processContents="lax" minOccurs="0"
3485 maxOccurs="unbounded"/>
3486 </sequence>
3487 </extension>
3488 </complexContent>
3489 </complexType>
3490
3491 <element name="import" type="sca:ImportType"
3492 substitutionGroup="sca:importBase"/>
3493 <complexType name="ImportType">
3494 <complexContent>
3495 <extension base="sca:Import">
3496 <attribute name="namespace" type="string" use="required"/>
3497 <attribute name="location" type="anyURI" use="required"/>
3498 </extension>
3499 </complexContent>
3500 </complexType>
3501
3502 ...
3503
3504 </schema>
```

3505 *Snippet 9-12: import and Import Schema*

3506

3507 Snippet 9-13 shows how the base import definition is extended to support Java imports. In the import
3508 element, the namespace is expected to be an XML namespace, an `import.java` element uses a Java
3509 package name instead. The snippet shows the definition of the ***import.java*** element and the
3510 ***JavaImportType*** type contained in ***sca-import-java.xsd***.

3511

```
3512 <?xml version="1.0" encoding="UTF-8"?>  
3513 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3514         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
3515         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">  
3516  
3517     <element name="import.java" type="sca:JavaImportType"  
3518             substitutionGroup="sca:importBase"/>  
3519     <complexType name="JavaImportType">  
3520         <complexContent>  
3521             <extension base="sca:Import">  
3522                 <attribute name="package" type="xs:String" use="required"/>  
3523                 <attribute name="location" type="xs:AnyURI" use="optional"/>  
3524             </extension>  
3525         </complexContent>  
3526     </complexType>  
3527 </schema>
```

3528 *Snippet 9-13: Extending import to import.java*

3529

3530 Snippet 9-14 shows an example of how the base definition can be extended by other specifications to
3531 support a new interface not defined in the SCA specifications. The snippet shows the definition of the ***my-***
3532 ***import-extension*** element and the ***my-import-extension-type*** type.

3533

```
3534 <?xml version="1.0" encoding="UTF-8"?>  
3535 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3536         targetNamespace="http://www.example.org/myextension"  
3537         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"  
3538         xmlns:tns="http://www.example.org/myextension">  
3539  
3540     <element name="my-import-extension"  
3541             type="tns:my-import-extension-type"  
3542             substitutionGroup="sca:importBase"/>  
3543     <complexType name="my-import-extension-type">  
3544         <complexContent>  
3545             <extension base="sca:Import">  
3546                 ...  
3547             </extension>  
3548         </complexContent>  
3549     </complexType>  
3550 </schema>
```

3551 *Snippet 9-14: Example import extension*

3552

3553 For a complete example using this extension point, see the definition of ***import.java*** in the SCA Java
3554 Common Annotations and APIs Specification [SCA-Java].

3555 9.5 Defining an Export Type

3556 Snippet 9-15 shows the base definition for the ***export*** element and ***ExportType*** type contained in ***sca-***
3557 ***core.xsd***; see appendix for complete schema.

3558

```
3559 <?xml version="1.0" encoding="UTF-8"?>
```

```

3560 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved. OASIS trademark,
3561 IPR and other policies apply. -->
3562 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3563   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3564   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3565   elementFormDefault="qualified">
3566
3567   ...
3568   <!-- Export -->
3569   <element name="exportBase" type="sca:Export" abstract="true" />
3570   <complexType name="Export" abstract="true">
3571     <complexContent>
3572       <extension base="sca:CommonExtensionBase">
3573         <sequence>
3574           <any namespace="##other" processContents="lax" minOccurs="0"
3575             maxOccurs="unbounded"/>
3576         </sequence>
3577       </extension>
3578     </complexContent>
3579   </complexType>
3580
3581   <element name="export" type="sca:ExportType"
3582     substitutionGroup="sca:exportBase"/>
3583   <complexType name="ExportType">
3584     <complexContent>
3585       <extension base="sca:Export">
3586         <attribute name="namespace" type="string" use="required"/>
3587       </extension>
3588     </complexContent>
3589   </complexType>
3590   ...
3591 </schema>

```

3592 *Snippet 9-15: export and Export Schema*

3593
3594 Snippet 9-16 shows how the base definition is extended to support Java exports. In a base *export*
3595 element, the *@namespace* attribute specifies XML namespace being exported. An *export.java* element
3596 uses a *@package* attribute to specify the Java package to be exported. The snippet shows the definition
3597 of the *export.java* element and the *JavaExport* type contained in *sca-export-java.xsd*.

```

3598
3599 <?xml version="1.0" encoding="UTF-8"?>
3600 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3601   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
3602   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912">
3603
3604   <element name="export.java" type="sca:JavaExportType"
3605     substitutionGroup="sca:exportBase"/>
3606   <complexType name="JavaExportType">
3607     <complexContent>
3608       <extension base="sca:Export">
3609         <attribute name="package" type="xs:String" use="required"/>
3610       </extension>
3611     </complexContent>
3612   </complexType>
3613 </schema>

```

3614 *Snippet 9-16: Extending export to export.java*

3615

3616 Snippet 9-17 we shows an example of how the base definition can be extended by other specifications to
3617 support a new interface not defined in the SCA specifications. The snippet shows the definition of the **my-**
3618 **export-extension** element and the **my-export-extension-type** type.

3619

```
3620 <?xml version="1.0" encoding="UTF-8"?>
3621 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3622         targetNamespace="http://www.example.org/myextension"
3623         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200903"
3624         xmlns:tns="http://www.example.org/myextension">
3625
3626     <element name="my-export-extension"
3627             type="tns:my-export-extension-type"
3628             substitutionGroup="sca:exportBase"/>
3629     <complexType name="my-export-extension-type">
3630         <complexContent>
3631             <extension base="sca:Export">
3632                 ...
3633             </extension>
3634         </complexContent>
3635     </complexType>
3636 </schema>
```

3637 *Snippet 9-17: Example export extension*

3638

3639 For a complete example using this extension point, see the definition of **export.java** in the SCA Java
3640 Common Annotations and APIs Specification [SCA-Java].

3641 10 Packaging and Deployment

3642 This section describes the SCA Domain and the packaging and deployment of artifacts contributed to the
3643 Domain.

3644 10.1 Domains

3645 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series of
3646 interconnected runtime nodes.

3647 A single SCA Domain defines the boundary of visibility for all SCA mechanisms. For example, SCA wires
3648 can only be used to connect components within a single SCA Domain. Connections to services outside
3649 the Domain use binding specific mechanisms for addressing services (such as WSDL endpoint URIs).
3650 Also, SCA mechanisms such as intents and policySets can only be used in the context of a single
3651 Domain. In general, external clients of a service that is developed and deployed using SCA are not able
3652 to tell that SCA is used to implement the service – it is an implementation detail.

3653 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification and
3654 is expected to be highly variable. An SCA Domain typically represents an area of business functionality
3655 controlled by a single organization. For example, an SCA Domain might be the whole of a business, or it
3656 might be a department within a business.

3657 As an example, for the accounts department in a business, the SCA Domain might cover all finance-
3658 related functions, and it might contain a series of composites dealing with specific areas of accounting,
3659 with one for Customer accounts and another dealing with Accounts Payable.

3660 An SCA Domain has the following:

- 3661 • A virtual domain-level composite whose components are deployed and running
- 3662 • A set of *installed contributions* that contain implementations, interfaces and other artifacts necessary
3663 to execute components
- 3664 • A set of logical services for manipulating the set of contributions and the virtual domain-level
3665 composite.

3666 The information associated with an SCA Domain can be stored in many ways, including but not limited to
3667 a specific filesystem structure or a repository.

3668 10.2 Contributions

3669 An SCA Domain might need a large number of different artifacts in order to work. These artifacts include
3670 artifacts defined by SCA and other artifacts such as object code files and interface definition files. The
3671 SCA-defined artifact types are all XML documents. The root elements of the different SCA definition
3672 documents are: composite, componentType and definitions. XML artifacts that are not defined by SCA
3673 but which are needed by an SCA Domain include XML Schema documents, WSDL documents, and
3674 BPEL documents. SCA constructs, like other XML-defined constructs, use XML qualified names for their
3675 identity (i.e. namespace + local name).

3676 Non-XML artifacts are also needed within an SCA Domain. The most obvious examples of such non-
3677 XML artifacts are Java, C++ and other programming language files necessary for component
3678 implementations. Since SCA is extensible, other XML and non-XML artifacts might also be needed.

3679 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This format is
3680 not the only packaging format that an SCA runtime can use. SCA allows many different packaging
3681 formats, but it is necessary for an SCA runtime to support the ZIP contribution format. When using the
3682 ZIP format for deploying a contribution, this specification does not specify whether that format is retained
3683 after deployment. For example, a Java EE based SCA runtime could convert the ZIP package to an EAR
3684 package. SCA expects certain characteristics of any packaging:

- 3685 • For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA
3686 as a hierarchy of resources based off of a single root [ASM12001]
- 3687 • Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy
3688 named META-INF [ASM12002]
- 3689 • Within any contribution packaging a document SHOULD exist directly under the META-INF directory
3690 named sca-contribution.xml which lists the SCA Composites within the contribution that are runnable.
3691 [ASM12003]
- 3692 The same document can also list namespaces of constructs that are defined within the contribution
3693 and which are available for use by other contributions, through export elements.
- 3694 These additional elements might not be physically present in the packaging, but might be generated
3695 based on the definitions and references that are present, or they might not exist at all if there are no
3696 unresolved references.
- 3697 See the section "SCA Contribution Metadata Document" for details of the format of this file.
- 3698 To illustrate that a variety of packaging formats can be used with SCA, the following are examples of
3699 formats that might be used to package SCA artifacts and metadata (as well as other artifacts) as a
3700 contribution:
 - 3701 • A filesystem directory
 - 3702 • An OSGi bundle
 - 3703 • A compressed directory (zip, gzip, etc)
 - 3704 • A JAR file (or its variants – WAR, EAR, etc)
- 3705 Contributions do not contain other contributions. If the packaging format is a JAR file that contains other
3706 JAR files (or any similar nesting of other technologies), the internal files are not treated as separate SCA
3707 contributions. It is up to the implementation to determine whether the internal JAR file is represented as a
3708 single artifact in the contribution hierarchy or whether all of the contents are represented as separate
3709 artifacts.
- 3710 A goal of SCA's approach to deployment is that the contents of a contribution do not need to be modified
3711 in order to install and use the contents of the contribution in a Domain.

3712 10.2.1 SCA Artifact Resolution

- 3713 Contributions can be self-contained, in that all of the artifacts necessary to run the contents of the
3714 contribution are found within the contribution itself. However, it can also be the case that the contents of
3715 the contribution make one or many references to artifacts that are not contained within the contribution.
3716 These references can be to SCA artifacts such as composites or they can be to other artifacts such as
3717 WSDL files, XSD files or to code artifacts such as Java class files and BPEL process files. Note: This
3718 form of artifact resolution does not apply to imports of composite files, as described in Section 6.6.
- 3719 A contribution can use some artifact-related or packaging-related means to resolve artifact references.
3720 Examples of such mechanisms include:
 - 3721 • @wsdlLocation and @schemaLocation attributes in references to WSDL and XSD schema artifacts
3722 respectively
 - 3723 • OSGi bundle mechanisms for resolving Java class and related resource dependencies
- 3724 Where present, artifact-related or packaging-related artifact resolution mechanisms MUST be used by the
3725 SCA runtime to resolve artifact dependencies. [ASM12005] The SCA runtime MUST raise an error if an
3726 artifact cannot be resolved using these mechanisms, if present. [ASM12021]
- 3727 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanism is can be
3728 used where no other mechanisms are available, for example in cases where the mechanisms used by the
3729 various contributions in the same SCA Domain are different. An example of this is where an OSGi
3730 Bundle is used for one contribution but where a second contribution used by the first one is not
3731 implemented using OSGi - e.g. the second contribution relates to a mainframe COBOL service whose
3732 interfaces are declared using a WSDL which is accessed by the first contribution.

3733 The SCA artifact resolution is likely to be most useful for SCA Domains containing heterogeneous
3734 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to work
3735 across different kinds of contribution.

3736 SCA artifact resolution works on the principle that a contribution which needs to use artifacts defined
3737 elsewhere expresses these dependencies using *import* statements in metadata belonging to the
3738 contribution. A contribution controls which artifacts it makes available to other contributions through
3739 *export* statements in metadata attached to the contribution. SCA artifact resolution is a general
3740 mechanism that can be extended for the handling of specific types of artifact. The general mechanism
3741 that is described in the following paragraphs is mainly intended for the handling of XML artifacts. Other
3742 types of artifacts, for example Java classes, use an extended version of artifact resolution that is
3743 specialized to their nature (eg. instead of "namespaces", Java uses "packages"). Descriptions of these
3744 more specialized forms of artifact resolution are contained in the SCA specifications that deal with those
3745 artifact types.

3746 Import and export statements for XML artifacts work at the level of namespaces - so that an import
3747 statement declares that artifacts from a specified namespace are found in other contributions, while an
3748 export statement makes all the artifacts from a specified namespace available to other contributions.

3749 An import declaration can simply specify the namespace to import. In this case, the locations which are
3750 searched for artifacts in that namespace are the contribution(s) in the Domain which have export
3751 declarations for the same namespace, if any. Alternatively an import declaration can specify a location
3752 from which artifacts for the namespace are obtained, in which case, that specific location is searched.

3753 There can be multiple import declarations for a given namespace. Where multiple import declarations
3754 are made for the same namespace, all the locations specified MUST be searched in lexical order.

3755 [ASM12022]

3756 For an XML namespace, artifacts can be declared in multiple locations - for example a given namespace
3757 can have a WSDL declared in one contribution and have an XSD defining XML data types in a second
3758 contribution.

3759 If the same artifact is declared in multiple locations, this is not an error. The first location as defined by
3760 lexical order is chosen. If no locations are specified no order exists and the one chosen is implementation
3761 dependent.

3762 When a contribution contains a reference to an artifact from a namespace that is declared in an import
3763 statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the
3764 SCA runtime MUST resolve artifacts in the following order:

3765 1. from the locations identified by the import statement(s) for the namespace.
3766 Locations MUST NOT be searched recursively in order to locate artifacts (i.e. only
3767 a one-level search is performed).

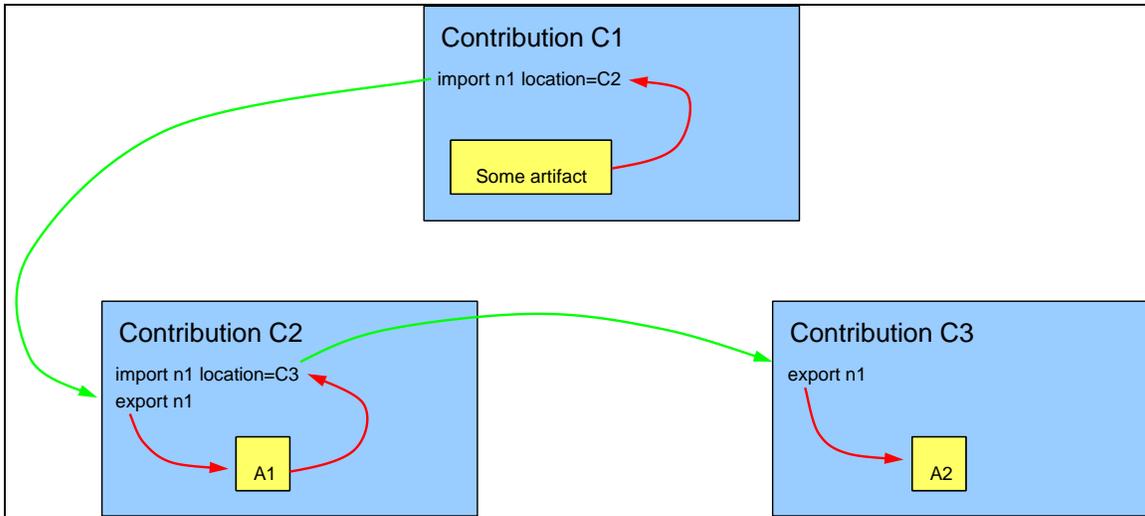
3768 2. from the contents of the contribution itself. [ASM12023]

3769 Checking for errors in artifacts MUST NOT be done for artifacts in the Installed state (ie where the
3770 artifacts are simply part of installed contributions) [ASM12031]

3771 For example:

- 3772 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the "n1"
3773 namespace from a second contribution "C2".
- 3774 • in contribution "C2" the artifact "A1" in the "n1" namespace references an artifact "A2" also in the "n1"
3775 namespace", which is resolved through an import of the "n1" namespace in "C2" which specifies the
3776 location "C3".

3777



3778

3779

Figure 10-1: Example of SCA Artifact Resolution between Contributions

3780

3781 The "A2" artifact is contained within the third contribution "C3" from which it is resolved by the contribution
3782 "C2". The "C3" contribution is never used to resolve artifacts directly for the "C1" contribution, since "C3"
3783 is not declared as an import location for "C1".

3784 For example, if for a contribution "C1", an import is used to resolve a composite "X1" contained in
3785 contribution "C2", and composite "X1" contains references to other artifacts such as WSDL files or XSDs,
3786 those references in "X1" are resolved in the context of contribution "C2" and not in the context of
3787 contribution "C1".

3788 The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through resolving an
3789 import statement. [ASM12024]

3790 The SCA runtime MUST raise an error if an artifact cannot be resolved by using artifact-related or
3791 packaging-related artifact resolution mechanisms, if present, by searching locations identified by the
3792 import statements of the contribution, if present, and by searching the contents of the contribution.
3793 [ASM12025]

3794 10.2.2 SCA Contribution Metadata Document

3795 The contribution can contain a document that declares runnable composites, exported definitions and
3796 imported definitions. The document is found at the path of META-INF/sca-contribution.xml relative to the
3797 root of the contribution. Frequently some SCA metadata needs to be specified by hand while other
3798 metadata is generated by tools (such as the <import> elements described below). To accommodate this,
3799 it is also possible to have an identically structured document at META-INF/sca-contribution-
3800 generated.xml. If this document exists (or is generated on an as-needed basis), it will be merged into the
3801 contents of sca-contribution.xml, with the entries in sca-contribution.xml taking priority if there are any
3802 conflicting declarations.

3803 An SCA runtime MUST make the <import/> and <export/> elements found in the META-INF/sca-
3804 contribution.xml and META-INF/sca-contribution-generated.xml files available for the SCA artifact
3805 resolution process. [ASM12026] An SCA runtime MUST reject files that do not conform to the schema
3806 declared in sca-contribution.xsd. [ASM12027] An SCA runtime MUST merge the contents of sca-
3807 contribution-generated.xml into the contents of sca-contribution.xml, with the entries in sca-
3808 contribution.xml taking priority if there are any conflicting declarations. [ASM12028]

3809

3810 The format of the document is:

3811

3812

```
<?xml version="1.0" encoding="ASCII"?>
<!-- sca-contribution pseudo-schema -->
```

3813
3814
3815
3816
3817
3818
3819

```
<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>
```

3820 *Snippet 10-1: contribution Pseudo-Schema*

3821

3822 **deployable element:** Identifies a composite which is a composite within the contribution that is a
3823 composite intended for potential inclusion into the virtual domain-level composite. Other composites in
3824 the contribution are not intended for inclusion but only for use by other composites. New composites can
3825 be created for a contribution after it is installed, by using the [add Deployment Composite](#) capability and
3826 the add To Domain Level Composite capability. An SCA runtime MAY deploy the composites in
3827 <deployable/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-
3828 generated.xml files. [ASM12029]

3829 Attributes of the deployable element:

- 3830 • **composite (1..1)** – The QName of a composite within the contribution.

3831 **Export element:** A declaration that artifacts belonging to a particular namespace are exported and are
3832 available for use within other contributions. An export declaration in a contribution specifies a
3833 namespace, all of whose definitions are considered to be exported. By default, definitions are not
3834 exported.

3835 The SCA artifact export is useful for SCA Domains containing heterogeneous mixtures of contribution
3836 packagings and technologies, where artifact-related or packaging-related mechanisms are unlikely to
3837 work across different kinds of contribution.

3838 Attributes of the export element:

- 3839 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the @namespace attribute
3840 of the export element SHOULD be the namespace URI for the exported definitions. [ASM12030] For
3841 XML technologies that define multiple *symbol spaces* that can be used within one namespace (e.g.
3842 WSDL portTypes are a different symbol space from WSDL bindings), all definitions from all symbol
3843 spaces are exported.

3844 Technologies that use naming schemes other than QNames use a different export element from the
3845 same substitution group as the the SCA <export> element. The element used identifies the
3846 technology, and can use any value for the namespace that is appropriate for that technology. For
3847 example, <export.java> can be used to export java definitions, in which case the namespace is a fully
3848 qualified package name.

3849 **Import element:** Import declarations specify namespaces of definitions that are needed by the definitions
3850 and implementations within the contribution, but which are not present in the contribution. It is expected
3851 that in most cases import declarations will be generated based on introspection of the contents of the
3852 contribution. In this case, the import declarations would be found in the META-INF/ sca-contribution-
3853 generated.xml document.

3854 Attributes of the import element:

- 3855 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace is the
3856 namespace URI for the imported definitions. For XML technologies that define multiple *symbol*
3857 *spaces* that can be used within one namespace (e.g. WSDL portTypes are a different symbol space
3858 from WSDL bindings), all definitions from all symbol spaces are imported.

3859 Technologies that use naming schemes other than QNames use a different import element from the
3860 same substitution group as the the SCA <import> element. The element used identifies the
3861 technology, and can use any value for the namespace that is appropriate for that technology. For
3862 example, <import.java> can be used to import java definitions, in which case the namespace is a fully
3863 qualified package name.

3864 • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific
3865 requirements for the form of this URI, nor the means by which it is resolved. It can point to another
3866 contribution (through its URI) or it can point to some location entirely outside the SCA Domain.
3867 It is expected that SCA runtimes can define implementation specific ways of resolving location
3868 information for artifact resolution between contributions. These mechanisms will however usually be
3869 limited to sets of contributions of one runtime technology and one hosting environment.

3870 In order to accommodate imports of artifacts between contributions of disparate runtime technologies, it is
3871 strongly suggested that SCA runtimes honor SCA contribution URIs as location specification.

3872 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are
3873 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact
3874 location specifications.

3875 The order in which the import statements are specified can play a role in this mechanism. Since
3876 definitions of one namespace can be distributed across several artifacts, multiple import declarations can
3877 be made for one namespace.

3878 The location value is only a default, and dependent contributions listed in the call to installContribution
3879 can override the value if there is a conflict. However, the specific mechanism for resolving conflicts
3880 between contributions that define conflicting definitions is implementation specific.

3881 If the value of the @location attribute is an SCA contribution URI, then the contribution packaging can
3882 become dependent on the deployment environment. In order to avoid such a dependency, it is
3883 recommended that dependent contributions are specified only when deploying or updating contributions
3884 as specified in the section 'Operations for Contributions' below.

3885 10.2.3 Contribution Packaging using ZIP

3886 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires that all
3887 runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This format allows that
3888 metadata specified by the section 'SCA Contribution Metadata Document' be present. Specifically, it can
3889 contain a top-level "META-INF" directory and a "META-INF/sca-contribution.xml" file and there can also
3890 be a "META-INF/sca-contribution-generated.xml" file in the package. SCA defined artifacts as well as
3891 non-SCA defined artifacts such as object files, WSDL definition, Java classes can be present anywhere in
3892 the ZIP archive,

3893 A definition of the ZIP file format is published by PKWARE in [an Application Note on the .ZIP file format](#)
3894 [\[ZIP-FORMAT\]](#).

3895 10.3 States of Artifacts in the Domain

3896 Artifacts in the SCA domain are in one of 3 states:

3897

- 3898 1. Installed
- 3899 2. Deployed
- 3900 3. Running

3901

3902 Installed artifacts are artifacts that are part of a Contribution that is installed into the Domain. Installed
3903 artifacts are available for use by other artifacts that are deployed, See "install Contribution" and "remove
3904 Contribution" to understand how artifacts are installed and uninstalled.

3905 Deployed artifacts are artifacts that are available to the SCA runtime to be run.. Artifacts are deployed
3906 either through explicit deployment actions or through the presence of <deployable/> elements in sca-
3907 contribution.xml files within a Contribution. If an artifact is deployed which has dependencies on other
3908 artifacts, then those dependent artifacts are also deployed.

3909 When the SCA runtime has one or more deployable artifacts, the runtime attempts to put those artifacts
3910 and any artifacts they depend on into the Running state. This can fail due to errors in one or more of the
3911 artifacts or the process can be delayed until all dependencies are available.

3912 Checking for errors in artifacts MUST NOT be done for artifacts in the Installed state (ie where the
3913 artifacts are simply part of installed contributions) [ASM12032]
3914 Errors in artifacts MUST be detected either during the Deployment of the artifacts, or during the process
3915 of putting the artifacts into the Running state, [ASM12033]

3916 10.4 Installed Contribution

3917 As noted in the section above, the contents of a contribution do not need to be modified in order to install
3918 and use it within a Domain. An *installed contribution* is a contribution with all of the associated
3919 information necessary in order to execute *deployable composites* within the contribution.

3920 An installed contribution is made up of the following things:

- 3921 • Contribution Packaging – the contribution that will be used as the starting point for resolving all
3922 references
- 3923 • Contribution base URI
- 3924 • Dependent contributions: a set of snapshots of other contributions that are used to resolve the import
3925 statements from the root composite and from other dependent contributions
 - 3926 – Dependent contributions might or might not be shared with other installed contributions.
 - 3927 – When the snapshot of any contribution is taken is implementation defined, ranging from the time
3928 the contribution is installed to the time of execution
- 3929 • Deployment-time composites.
3930 These are composites that are added into an installed contribution after it has been deployed. This
3931 makes it possible to provide final configuration and access to implementations within a contribution
3932 without having to modify the contribution. These do not have to be provided as composites that
3933 already exist within the contribution can also be used for deployment.

3934 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML, fully
3935 qualified class names in Java).

3936 If multiple dependent contributions have exported definitions with conflicting qualified names, the
3937 algorithm used to determine the qualified name to use is implementation dependent. Implementations of
3938 SCA MAY also raise an error if there are conflicting names exported from multiple contributions.
3939 [ASM12007]

3940 10.4.1 Installed Artifact URIs

3941 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are
3942 constructed by starting with the base URI of the contribution and adding the relative URI of each artifact
3943 (recalling that SCA demands that any packaging format be able to offer up its artifacts in a single
3944 hierarchy).

3945 10.5 Operations for Contributions

3946 SCA Runtimes provide the following conceptual functionality associated with contributions to the Domain
3947 (meaning the function might not be represented as addressable services and also meaning that
3948 equivalent functionality might be provided in other ways). An SCA runtime MAY provide the contribution
3949 operation functions (install Contribution, update Contribution, add Deployment Composite, update
3950 Deployment Composite, remove Contribution). [ASM12008]

3951 10.5.1 install Contribution & update Contribution

3952 Creates or updates an installed contribution with a supplied root contribution, and installed at a supplied
3953 base URI. A supplied dependent contribution list (<export/> elements) specifies the contributions that are
3954 used to resolve the dependencies of the root contribution and other dependent contributions. These
3955 override any dependent contributions explicitly listed via the @location attribute in the import statements
3956 of the contribution.

3957 SCA follows the simplifying assumption that the use of a contribution for resolving anything also means
3958 that all other exported artifacts can be used from that contribution. Because of this, the dependent
3959 contribution list is just a list of installed contribution URIs. There is no need to specify what is being used
3960 from each one.

3961 Each dependent contribution is also an installed contribution, with its own dependent contributions. By
3962 default these dependent contributions of the dependent contributions (which we will call *indirect*
3963 *dependent contributions*) are included as dependent contributions of the installed contribution. However,
3964 if a contribution in the dependent contribution list exports any conflicting definitions with an indirect
3965 dependent contribution, then the indirect dependent contribution is not included (i.e. the explicit list
3966 overrides the default inclusion of indirect dependent contributions). Also, **if there is ever a conflict**
3967 **between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in**
3968 **the dependent contribution list. [ASM12009]**

3969 Note that in many cases, the dependent contribution list can be generated. In particular, if the creator of
3970 a Domain is careful to avoid creating duplicate definitions for the same qualified name, then it is easy for
3971 this list to be generated by tooling.

3972 10.5.2 add Deployment Composite & update Deployment Composite

3973 Adds or updates a deployment composite using a supplied composite ("composite by value" – a data
3974 structure, not an existing resource in the Domain) to the contribution identified by a supplied contribution
3975 URI. The added or updated deployment composite is given a relative URI that matches the @name
3976 attribute of the composite, with a ".composite" suffix. Since all composites run within the context of a
3977 installed contribution (any component implementations or other definitions are resolved within that
3978 contribution), this functionality makes it possible for the deployer to create a composite with final
3979 configuration and wiring decisions and add it to an installed contribution without having to modify the
3980 contents of the root contribution.

3981 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts). It is
3982 then possible for those to be given component names by a (possibly generated) composite that is added
3983 into the installed contribution, without having to modify the packaging.

3984 10.5.3 remove Contribution

3985 Removes the deployed contribution identified by a supplied contribution URI.

3986 10.6 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3987 For certain types of artifact, there are existing and commonly used mechanisms for referencing a specific
3988 concrete location where the artifact can be resolved.

3989 Examples of these mechanisms include:

- 3990 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the place
3991 holding the WSDL itself.
- 3992 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a URI where
3993 the XSD is found.

3994 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does not have
3995 to be dereferenced.

3996 SCA permits the use of these mechanisms **Where present, non-SCA artifact resolution mechanisms**
3997 **MUST be used by the SCA runtime in precedence to the SCA mechanisms. [ASM12010]** However, use
3998 of these mechanisms is discouraged because tying assemblies to addresses in this way makes the
3999 assemblies less flexible and prone to errors when changes are made to the overall SCA Domain.

4000 **Note:** **If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the**
4001 **resource indicated when using the mechanism (e.g. the URI is incorrect or invalid, say) the SCA runtime**
4002 **MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an alternative.**
4003 **[ASM12011]**

4004 **10.7 Domain-Level Composite**

4005 The domain-level composite is a virtual composite, in that it is not defined by a composite definition
4006 document. Rather, it is built up and modified through operations on the Domain. However, in other
4007 respects it is very much like a composite, since it contains components, wires, services and references.

4008 The value of @autowire for the logical Domain composite MUST be autowire="false". [ASM12012]

4009 For components at the Domain level, with references for which @autowire="true" applies, the behaviour
4010 of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

4011 1) The SCA runtime disallows deployment of any components with autowire references. In this case, the
4012 SCA runtime MUST raise an exception at the point where the component is deployed.

4013 2) The SCA runtime evaluates the target(s) for the reference at the time that the component is deployed
4014 and does not update those targets when later deployment actions occur.

4015 3) The SCA runtime re-evaluates the target(s) for the reference dynamically as later deployment actions
4016 occur resulting in updated reference targets which match the new Domain configuration. How the
4017 reconfiguration of the reference takes place is described by the relevant client and implementation
4018 specifications.

4019 [ASM12013]

4020 The abstract domain-level functionality for modifying the domain-level composite is as follows, although a
4021 runtime can supply equivalent functionality in a different form:

4022 **10.7.1 add To Domain-Level Composite**

4023 This functionality adds the composite identified by a supplied URI to the Domain Level Composite. The
4024 supplied composite URI refers to a composite within an installed contribution. The composite's installed
4025 contribution determines how the composite's artifacts are resolved (directly and indirectly). The supplied
4026 composite is added to the domain composite with semantics that correspond to the domain-level
4027 composite having an <include> statement that references the supplied composite. All of the composites
4028 components become top-level components and the component services become externally visible
4029 services (eg. they would be present in a WSDL description of the Domain). The meaning of any promoted
4030 services and references in the supplied composite is not defined; since there is no composite scope
4031 outside the domain composite, the usual idea of promotion has no utility.

4032 **10.7.2 remove From Domain-Level Composite**

4033 Removes from the Domain Level composite the elements corresponding to the composite identified by a
4034 supplied composite URI. This means that the removal of the components, wires, services and references
4035 originally added to the domain level composite by the identified composite.

4036 **10.7.3 get Domain-Level Composite**

4037 Returns a <composite> definition that has an <include> line for each composite that had been added to
4038 the domain level composite. It is important to note that, in dereferencing the included composites, any
4039 referenced artifacts are resolved in terms of that installed composite.

4040 **10.7.4 get QName Definition**

4041 In order to make sense of the domain-level composite (as returned by get Domain-Level Composite), it
4042 needs to be possible to get the definitions for named artifacts in the included composites. This
4043 functionality takes the supplied URI of an installed contribution (which provides the context), a supplied
4044 qualified name of a definition to look up, and a supplied symbol space (as a QName, e.g. wsdl:portType).
4045 The result is a single definition, in whatever form is appropriate for that definition type.

4046 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities need to
4047 exist in some form, but not necessarily as a service operation with exactly this signature.

4048 10.8 Dynamic Behaviour of Wires in the SCA Domain

4049 For components with references which are at the Domain level, there is the potential for dynamic
4050 behaviour when the wires for a component reference change (this can only apply to component
4051 references at the Domain level and not to components within composites used as implementations):

4052 The configuration of the wires for a component reference of a component at the Domain level can change
4053 by means of deployment actions:

- 4054 1. <wire/> elements can be added, removed or replaced by deployment actions
- 4055 2. Components can be updated by deployment actions (i.e. this can change the
4056 component reference configuration)
- 4057 3. Components which are the targets of reference wires can be updated or removed
- 4058 4. Components can be added that are potential targets for references which are
4059 marked with @autowire=true

4060

4061 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose
4062 references are affected by those deployment actions MAY have their references updated by the SCA
4063 runtime dynamically without the need to stop and start those components. [ASM12014]

4064 Where components are updated by deployment actions (their configuration is changed in some way,
4065 which includes changing the wires of component references), the new configuration MUST apply to all
4066 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY
4067 choose to maintain existing instances with the old configuration of components updated by deployment
4068 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.
4069 [ASM12016]

4070 Where a component that is the target of a wire is removed, without the wire being changed, then future
4071 invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is
4072 the result of the autowire process, the SCA runtime MUST:

- 4073 • either cause future invocation of the target component's services to fail with a ServiceUnavailable
4074 fault
- 4075 • or alternatively, if an alternative target component is available that satisfies the autowire process,
4076 update the reference of the source component [ASM12017]

4077 Where a component that is the target of a wire is updated, future invocations of that reference SHOULD
4078 use the updated component. [ASM12018]

4079 Where a component is added to the Domain that is a potential target for a domain level component
4080 reference where that reference is marked as @autowire=true, the SCA runtime MUST:

- 4081 • either update the references for the source component once the new component is running.
- 4082 • or alternatively, defer the updating of the references of the source component until the source
4083 component is stopped and restarted. [ASM12020]

4084 10.9 Dynamic Behaviour of Component Property Values

4085 For a domain level component with a Property whose value is obtained from a Domain-level Property
4086 through the use of the @source attribute, if the domain level property is updated by means of deployment
4087 actions, the SCA runtime MUST

- 4088 • either update the property value of the domain level component once the update of the domain
4089 property is complete
- 4090 • or defer the updating of the component property value until the component is stopped and
4091 restarted

4092 [ASM12034]

4093 11 SCA Runtime Considerations

4094 This section describes aspects of an SCA Runtime that are defined by this specification.

4095 11.1 Error Handling

4096 The SCA Assembly specification identifies situations where the configuration of the SCA Domain and its
4097 contents are in error. When one of these situations occurs, the specification requires that the SCA
4098 Runtime that is interacting with the SCA Domain and the artifacts it contains recognises that there is an
4099 error, raise the error in a suitable manner and also refuse to run components and services that are in
4100 error.

4101 The SCA Assembly specification is not prescriptive about the functionality of an SCA Runtime and the
4102 specification recognizes that there can be a range of design points for an SCA runtime. As a result, the
4103 SCA Assembly specification describes a range of error handling approaches which can be adopted by an
4104 SCA runtime.

4105 An SCA Runtime **MUST** raise an error for every situation where the configuration of the SCA Domain or
4106 its contents are in error. The error is either raised at deployment time or at runtime, depending on the
4107 nature of the error and the design of the SCA Runtime. [ASM14005]

4108 11.1.1 Errors which can be Detected at Deployment Time

4109 Some error situations can be detected at the point that artifacts are deployed to the Domain. An example
4110 is a composite document that is invalid in a way that can be detected by static analysis, such as
4111 containing a component with two services with the same @name attribute.

4112 An SCA runtime **SHOULD** detect errors at deployment time where those errors can be found through
4113 static analysis. [ASM14001] The SCA runtime **SHOULD** prevent deployment of contributions that are in
4114 error, and raise the error to the process performing the deployment (e.g. write a message to an interactive
4115 console or write a message to a log file). [ASM14002]

4116 The SCA Assembly specification recognizes that there are reasons why a particular SCA runtime finds it
4117 desirable to deploy contributions that contain errors (e.g. to assist in the process of development and
4118 debugging) - and as a result also supports an error handling strategy that is based on detecting problems
4119 at runtime. However, it is wise to consider reporting problems at an early stage in the deployment
4120 process.

4121 11.1.2 Errors which are Detected at Runtime

4122 An SCA runtime can detect problems at runtime. These errors can include some which can be found
4123 from static analysis (e.g. the inability to wire a reference because the target service does not exist in the
4124 Domain) and others that can only be discovered dynamically (e.g. the inability to invoke some remote
4125 Web service because the remote endpoint is unavailable).

4126 Where errors can be detected through static analysis, the principle is that components that are known to
4127 be in error are not run. So, for example, if there is a component with a required reference (multiplicity 1..1
4128 or 1..n) which is not wired, best practice is that the component is not run. If an attempt is made to invoke
4129 a service operation of that component, a "ServiceUnavailable" fault is raised to the invoker. It is also
4130 regarded as best practice that errors of this kind are also raised through appropriate management
4131 interfaces, for example to the deployer or to the operator of the system.

4132 Where errors are only detected at runtime, when the error is detected an error **MUST** be raised to the
4133 component that is attempting the activity concerned with the error. [ASM14003] For example, if a
4134 component invokes an operation on a reference, but the target service is unavailable, a
4135 "ServiceUnavailable" fault is raised to the component. When an error that could have been detected
4136 through static analysis is detected and raised at runtime for a component, the component **SHOULD NOT**
4137 be run until the error is fixed. [ASM14004] Such errors can be fixed by redeployment or deployment of
4138 other components in the domain.

4139 12 Conformance

4140 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,
4141 are considered to be authoritative and take precedence over the XML schema defined in the appendix of
4142 this
4143 document.

4144 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd, sca-interface-
4145 wsdl.xsd, sca-implementation-composite.xsd and sca-binding-sca.xsd schema. [ASM13001]

4146 An SCA runtime MUST reject a contribution file that does not conform to the sca-contribution.xsd schema.
4147 [ASM13002]

4148 An SCA runtime MUST reject a definitions file that does not conform to the sca-definitions.xsd schema.
4149 [ASM13003]

4150 There are two categories of artifacts that this specification defines conformance for: SCA Documents and
4151 SCA Runtimes.

4152 12.1 SCA Documents

4153 For a document to be a valid SCA Document, it MUST comply with one of the SCA document types
4154 below:

4155 SCA Composite Document:

4156 An SCA Composite Document is a file that MUST have an SCA <composite/> element as its root
4157 element and MUST conform to the sca-core-1.1.xsd schema and MUST comply with the
4158 additional constraints on the document contents as defined in Appendix C.

4159 SCA ComponentType Document:

4160 An SCA ComponentType Document is a file that MUST have an SCA <componentType/>
4161 element as its root element and MUST conform to the sca-core-1.1.xsd schema and MUST
4162 comply with the additional constraints on the document contents as defined in
4163 Appendix C.

4164 SCA Definitions Document:

4165 An SCA Definitions Document is a file that MUST have an SCA <definitions/> element as its root
4166 and MUST conform to the sca-definition-1.1.xsd schema and MUST comply with the additional
4167 constraints on the document contents as defined in Appendix C.

4168 SCA Contribution Document:

4169 An SCA Contribution Document is a file that MUST have an SCA <contribution/> element as its
4170 root element and MUST conform to the sca-contribution-1.1.xsd schema and MUST comply with
4171 the additional constraints on the document contents as defined in Appendix C.

4172 SCA Interoperable Packaging Document:

4173 A ZIP file containing SCA Documents and other related artifacts. The ZIP file SHOULD contain a
4174 top-level "META-INF" directory, and SHOULD contain a "META-INF/sca-contribution.xml" file, and
4175 MAY contain a "META-INF/sca-contribution-generated.xml" file.

4176

4177

4178 12.2 SCA Runtime

4179 An implementation that claims to conform to the requirements of an SCA Runtime defined in this
4180 specification MUST meet the following conditions:

- 4181 1. The implementation MUST comply with all mandatory statements listed in table
4182 [Mandatory Items](#) in Appendix C: Conformance Items, related to an SCA Runtime.
- 4183 2. The implementation MUST conform to the SCA Policy Framework v 1.1 Specification [**SCA-**
4184 **POLICY**].
- 4185 3. The implementation MUST support at least one implementation type standardized by the
4186 OpenCSA Member Section or at least one implementation type that complies with the
4187 following rules:
- 4188 a. The implementation type is defined in compliance with the SCA Assembly Extension Model
4189 (Section 9 of the SCA Assembly Specification).
- 4190
4191 b. A document describing the mapping of the constructs defined in the SCA Assembly
4192 specification with those of the implementation type exists and is made available to its
4193 prospective user community. Such a document describes how SCA components can be
4194 developed using the implementation type, how these components can be configured and
4195 assembled together (as instances of Components in SCA compositions). The form and
4196 content of such a document are described in the specification "Implementation Type
4197 Documentation Requirements for SCA Assembly Model Version 1.1 Specification" [SCA-
4198 IMPLTYPDOC]. The contents outlined in this specification template MUST be provided in
4200 order for an SCA runtime to claim compliance with the SCA Assembly Specification on the
4201 basis of providing support for that implementation type. An example of a document that
4202 describes an implementation type is the "SCA POJO Component Implementation
4203 Specification Version 1.1" [SCA-Java].
- 4204 c. An adapted version of the SCA Assembly Test Suite which uses the implementation type
4205 exists and is made available to its prospective user community. The steps required to adapt
4206 the SCA Assembly Test Suite for a new implementation type are described in the
4207 specification "Test Suite Adaptation for SCA Assembly Model Version 1.1 Specification"
4208 [SCA-TSA]. The requirements described in this specification MUST be met in order for an
4209 SCA runtime to claim compliance with the SCA Assembly Specification on the basis of
4210 providing support for that implementation type.
- 4211
- 4212 4. The implementation MUST support binding.sca and MUST support and conform to the SCA
4213 Web Service Binding Specification v 1.1.

4214 **12.2.1 Optional Items**

4215 In addition to mandatory items, Appendix C: Conformance Items lists a number of non-mandatory items
4216 that can be implemented SCA Runtimes. These items are categorized into functionally related classes as
4217 follows:

- 4218 • Development – items to improve the development of SCA contributions, debugging, etc.
- 4219 • Enhancement – items that add functionality and features to the SCA Runtime.
- 4220 • Interoperation – items that improve interoperability of SCA contributions and Runtimes

4221 These classifications are not rigid and some may overlap; items are classified according to their primary
4222 intent.

4223

4224

A. XML Schemas

4225

A.1 sca.xsd

4226

sca-1.1.xsd is provided for convenience. It contains <include/> elements for each of the schema files that contribute to the <http://docs.oasis-open.org/ns/opencsa/sca/200912> namespace.

4227

4228

A.2 sca-core.xsd

4229

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
      OASIS trademark, IPR and other policies apply. -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
  elementFormDefault="qualified">

  <include schemaLocation="sca-policy-1.1-cd03.xsd"/>
  <import namespace="http://www.w3.org/XML/1998/namespace"
    schemaLocation="http://www.w3.org/2001/xml.xsd"/>

  <!-- Common extension base for SCA definitions -->
  <complexType name="CommonExtensionBase">
    <sequence>
      <element ref="sca:documentation" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
    <anyAttribute namespace="##other" processContents="lax"/>
  </complexType>

  <element name="documentation" type="sca:Documentation"/>
  <complexType name="Documentation" mixed="true">
    <sequence>
      <any namespace="##other" processContents="lax" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
    <attribute ref="xml:lang"/>
  </complexType>

  <!-- Component Type -->
  <element name="componentType" type="sca:ComponentType"/>
  <complexType name="ComponentType">
    <complexContent>
      <extension base="sca:CommonExtensionBase">
        <sequence>
          <element ref="sca:implementation" minOccurs="0"/>
          <choice minOccurs="0" maxOccurs="unbounded">
            <element name="service" type="sca:ComponentService"/>
            <element name="reference"
              type="sca:ComponentTypeReference"/>
            <element name="property" type="sca:Property"/>
          </choice>
          <any namespace="##other" processContents="lax" minOccurs="0"
            maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>

  <!-- Composite -->
```

4279

```

4280 <element name="composite" type="sca:Composite"/>
4281 <complexType name="Composite">
4282   <complexContent>
4283     <extension base="sca:CommonExtensionBase">
4284       <sequence>
4285         <element ref="sca:include" minOccurs="0"
4286           maxOccurs="unbounded"/>
4287         <choice minOccurs="0" maxOccurs="unbounded">
4288           <element ref="sca:requires"/>
4289           <element ref="sca:policySetAttachment"/>
4290           <element name="service" type="sca:Service"/>
4291           <element name="property" type="sca:Property"/>
4292           <element name="component" type="sca:Component"/>
4293           <element name="reference" type="sca:Reference"/>
4294           <element name="wire" type="sca:Wire"/>
4295         </choice>
4296         <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
4297       </sequence>
4298       <attribute name="name" type="NCName" use="required"/>
4299       <attribute name="targetNamespace" type="anyURI" use="required"/>
4300       <attribute name="local" type="boolean" use="optional"
4301         default="false"/>
4302       <attribute name="autowire" type="boolean" use="optional"
4303         default="false"/>
4304       <attribute name="requires" type="sca:listOfQNames"
4305         use="optional"/>
4306       <attribute name="policySets" type="sca:listOfQNames"
4307         use="optional"/>
4308     </extension>
4309   </complexContent>
4310 </complexType>
4311
4312 <!-- Contract base type for Service, Reference -->
4313 <complexType name="Contract" abstract="true">
4314   <complexContent>
4315     <extension base="sca:CommonExtensionBase">
4316       <sequence>
4317         <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4318         <element ref="sca:binding" minOccurs="0"
4319           maxOccurs="unbounded" />
4320         <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4321         <element ref="sca:requires" minOccurs="0"
4322           maxOccurs="unbounded"/>
4323         <element ref="sca:policySetAttachment" minOccurs="0"
4324           maxOccurs="unbounded"/>
4325         <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
4326       </sequence>
4327       <attribute name="name" type="NCName" use="required" />
4328       <attribute name="requires" type="sca:listOfQNames"
4329         use="optional" />
4330       <attribute name="policySets" type="sca:listOfQNames"
4331         use="optional"/>
4332     </extension>
4333   </complexContent>
4334 </complexType>
4335
4336 <!-- Service -->
4337 <complexType name="Service">
4338   <complexContent>
4339     <extension base="sca:Contract">
4340       <attribute name="promote" type="anyURI" use="required"/>
4341     </extension>
4342   </complexContent>
4343 </complexType>

```

```

4344
4345
4346 <!-- Interface -->
4347 <element name="interface" type="sca:Interface" abstract="true"/>
4348 <complexType name="Interface" abstract="true">
4349   <complexContent>
4350     <extension base="sca:CommonExtensionBase">
4351       <choice minOccurs="0" maxOccurs="unbounded">
4352         <element ref="sca:requires"/>
4353         <element ref="sca:policySetAttachment"/>
4354       </choice>
4355       <attribute name="remotable" type="boolean" use="optional"/>
4356       <attribute name="requires" type="sca:listOfQNames"
4357         use="optional"/>
4358       <attribute name="policySets" type="sca:listOfQNames"
4359         use="optional"/>
4360     </extension>
4361   </complexContent>
4362 </complexType>
4363
4364 <!-- Reference -->
4365 <complexType name="Reference">
4366   <complexContent>
4367     <extension base="sca:Contract">
4368       <attribute name="target" type="sca:listOfAnyURIs"
4369         use="optional"/>
4370       <attribute name="wiredByImpl" type="boolean" use="optional"
4371         default="false"/>
4372       <attribute name="multiplicity" type="sca:Multiplicity"
4373         use="required"/>
4374       <attribute name="promote" type="sca:listOfAnyURIs"
4375         use="required"/>
4376     </extension>
4377   </complexContent>
4378 </complexType>
4379
4380 <!-- Property -->
4381 <complexType name="SCAPropertyBase" mixed="true">
4382   <sequence>
4383     <any namespace="##any" processContents="lax" minOccurs="0"
4384       maxOccurs="unbounded"/>
4385     <!-- NOT an extension point; This any exists to accept
4386       the element-based or complex type property
4387       i.e. no element-based extension point under "sca:property" -->
4388   </sequence>
4389   <!-- mixed="true" to handle simple type -->
4390   <attribute name="name" type="NCName" use="required"/>
4391   <attribute name="type" type="QName" use="optional"/>
4392   <attribute name="element" type="QName" use="optional"/>
4393   <attribute name="many" type="boolean" use="optional" default="false"/>
4394   <attribute name="value" type="anySimpleType" use="optional"/>
4395   <anyAttribute namespace="##other" processContents="lax"/>
4396 </complexType>
4397
4398 <complexType name="Property" mixed="true">
4399   <complexContent mixed="true">
4400     <extension base="sca:SCAPropertyBase">
4401       <attribute name="mustSupply" type="boolean" use="optional"
4402         default="false"/>
4403     </extension>
4404   </complexContent>
4405 </complexType>
4406
4407 <complexType name="PropertyValue" mixed="true">
4408   <complexContent mixed="true">

```

```

4408     <extension base="sca:SCAPropertyBase">
4409         <attribute name="source" type="string" use="optional"/>
4410         <attribute name="file" type="anyURI" use="optional"/>
4411     </extension>
4412 </complexContent>
4413 </complexType>
4414
4415 <!-- Binding -->
4416 <element name="binding" type="sca:Binding" abstract="true"/>
4417 <complexType name="Binding" abstract="true">
4418     <complexContent>
4419         <extension base="sca:CommonExtensionBase">
4420             <sequence>
4421                 <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
4422                 <element ref="sca:operationSelector" minOccurs="0"
4423                     maxOccurs="1" />
4424                 <element ref="sca:requires" minOccurs="0"
4425                     maxOccurs="unbounded"/>
4426                 <element ref="sca:policySetAttachment" minOccurs="0"
4427                     maxOccurs="unbounded"/>
4428             </sequence>
4429             <attribute name="uri" type="anyURI" use="optional"/>
4430             <attribute name="name" type="NCName" use="optional"/>
4431             <attribute name="requires" type="sca:listOfQNames"
4432                 use="optional"/>
4433             <attribute name="policySets" type="sca:listOfQNames"
4434                 use="optional"/>
4435         </extension>
4436     </complexContent>
4437 </complexType>
4438
4439 <!-- Binding Type -->
4440 <element name="bindingType" type="sca:BindingType"/>
4441 <complexType name="BindingType">
4442     <complexContent>
4443         <extension base="sca:CommonExtensionBase">
4444             <sequence>
4445                 <any namespace="##other" processContents="lax" minOccurs="0"
4446                     maxOccurs="unbounded"/>
4447             </sequence>
4448             <attribute name="type" type="QName" use="required"/>
4449             <attribute name="alwaysProvides" type="sca:listOfQNames"
4450                 use="optional"/>
4451             <attribute name="mayProvide" type="sca:listOfQNames"
4452                 use="optional"/>
4453         </extension>
4454     </complexContent>
4455 </complexType>
4456
4457 <!-- WireFormat Type -->
4458 <element name="wireFormat" type="sca:WireFormatType" abstract="true"/>
4459 <complexType name="WireFormatType" abstract="true">
4460     <anyAttribute namespace="##other" processContents="lax"/>
4461 </complexType>
4462
4463 <!-- OperationSelector Type -->
4464 <element name="operationSelector" type="sca:OperationSelectorType"
4465     abstract="true"/>
4466 <complexType name="OperationSelectorType" abstract="true">
4467     <anyAttribute namespace="##other" processContents="lax"/>
4468 </complexType>
4469
4470 <!-- Callback -->
4471 <element name="callback" type="sca:Callback"/>

```

```

4472 <complexType name="Callback">
4473   <complexContent>
4474     <extension base="sca:CommonExtensionBase">
4475       <choice minOccurs="0" maxOccurs="unbounded">
4476         <element ref="sca:binding"/>
4477         <element ref="sca:requires"/>
4478         <element ref="sca:policySetAttachment"/>
4479         <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
4480       </choice>
4481       <attribute name="requires" type="sca:listOfQNames"
4482         use="optional"/>
4483       <attribute name="policySets" type="sca:listOfQNames"
4484         use="optional"/>
4485     </extension>
4486   </complexContent>
4487 </complexType>
4488
4489 <!-- Component -->
4490 <complexType name="Component">
4491   <complexContent>
4492     <extension base="sca:CommonExtensionBase">
4493       <sequence>
4494         <element ref="sca:implementation" minOccurs="1"
4495           maxOccurs="1"/>
4496         <choice minOccurs="0" maxOccurs="unbounded">
4497           <element name="service" type="sca:ComponentService"/>
4498           <element name="reference" type="sca:ComponentReference"/>
4499           <element name="property" type="sca:PropertyValue"/>
4500           <element ref="sca:requires"/>
4501           <element ref="sca:policySetAttachment"/>
4502         </choice>
4503         <any namespace="##other" processContents="lax" minOccurs="0"
4504           maxOccurs="unbounded"/>
4505       </sequence>
4506       <attribute name="name" type="NCName" use="required"/>
4507       <attribute name="autowire" type="boolean" use="optional"/>
4508       <attribute name="requires" type="sca:listOfQNames"
4509         use="optional"/>
4510       <attribute name="policySets" type="sca:listOfQNames"
4511         use="optional"/>
4512     </extension>
4513   </complexContent>
4514 </complexType>
4515
4516 <!-- Component Service -->
4517 <complexType name="ComponentService">
4518   <complexContent>
4519     <extension base="sca:Contract">
4520     </extension>
4521   </complexContent>
4522 </complexType>
4523
4524 <!-- Component Reference -->
4525 <complexType name="ComponentReference">
4526   <complexContent>
4527     <extension base="sca:Contract">
4528       <attribute name="autowire" type="boolean" use="optional"/>
4529       <attribute name="target" type="sca:listOfAnyURIs"
4530         use="optional"/>
4531       <attribute name="wiredByImpl" type="boolean" use="optional"
4532         default="false"/>
4533       <attribute name="multiplicity" type="sca:Multiplicity"
4534         use="optional" default="1..1"/>
4535       <attribute name="nonOverridable" type="boolean" use="optional"

```

```

4536         default="false"/>
4537     </extension>
4538 </complexContent>
4539 </complexType>
4540
4541 <!-- Component Type Reference -->
4542 <complexType name="ComponentTypeReference">
4543     <complexContent>
4544         <restriction base="sca:ComponentReference">
4545             <sequence>
4546                 <element ref="sca:documentation" minOccurs="0"
4547                     maxOccurs="unbounded"/>
4548                 <element ref="sca:interface" minOccurs="0"/>
4549                 <element ref="sca:binding" minOccurs="0"
4550                     maxOccurs="unbounded"/>
4551                 <element ref="sca:callback" minOccurs="0"/>
4552                 <element ref="sca:requires" minOccurs="0"
4553                     maxOccurs="unbounded"/>
4554                 <element ref="sca:policySetAttachment" minOccurs="0"
4555                     maxOccurs="unbounded"/>
4556                 <element ref="sca:extensions" minOccurs="0" maxOccurs="1" />
4557             </sequence>
4558             <attribute name="name" type="NCName" use="required"/>
4559             <attribute name="autowire" type="boolean" use="optional"/>
4560             <attribute name="wiredByImpl" type="boolean" use="optional"
4561                 default="false"/>
4562             <attribute name="multiplicity" type="sca:Multiplicity"
4563                 use="optional" default="1..1"/>
4564             <attribute name="requires" type="sca:listOfQNames"
4565                 use="optional"/>
4566             <attribute name="policySets" type="sca:listOfQNames"
4567                 use="optional"/>
4568             <anyAttribute namespace="##other" processContents="lax"/>
4569         </restriction>
4570     </complexContent>
4571 </complexType>
4572
4573 <!-- Implementation -->
4574 <element name="implementation" type="sca:Implementation" abstract="true"/>
4575 <complexType name="Implementation" abstract="true">
4576     <complexContent>
4577         <extension base="sca:CommonExtensionBase">
4578             <choice minOccurs="0" maxOccurs="unbounded">
4579                 <element ref="sca:requires"/>
4580                 <element ref="sca:policySetAttachment"/>
4581             </choice>
4582             <attribute name="requires" type="sca:listOfQNames"
4583                 use="optional"/>
4584             <attribute name="policySets" type="sca:listOfQNames"
4585                 use="optional"/>
4586         </extension>
4587     </complexContent>
4588 </complexType>
4589 </complexType>
4590
4591 <!-- Implementation Type -->
4592 <element name="implementationType" type="sca:ImplementationType"/>
4593 <complexType name="ImplementationType">
4594     <complexContent>
4595         <extension base="sca:CommonExtensionBase">
4596             <sequence>
4597                 <any namespace="##other" processContents="lax" minOccurs="0"
4598                     maxOccurs="unbounded"/>
4599             </sequence>

```

```

4600         <attribute name="type" type="QName" use="required"/>
4601         <attribute name="alwaysProvides" type="sca:listOfQNames"
4602             use="optional"/>
4603         <attribute name="mayProvide" type="sca:listOfQNames"
4604             use="optional"/>
4605     </extension>
4606 </complexContent>
4607 </complexType>
4608
4609 <!-- Wire -->
4610 <complexType name="Wire">
4611     <complexContent>
4612         <extension base="sca:CommonExtensionBase">
4613             <sequence>
4614                 <any namespace="##other" processContents="lax" minOccurs="0"
4615                     maxOccurs="unbounded"/>
4616             </sequence>
4617             <attribute name="source" type="anyURI" use="required"/>
4618             <attribute name="target" type="anyURI" use="required"/>
4619             <attribute name="replace" type="boolean" use="optional"
4620                 default="false"/>
4621         </extension>
4622     </complexContent>
4623 </complexType>
4624
4625 <!-- Include -->
4626 <element name="include" type="sca:Include"/>
4627 <complexType name="Include">
4628     <complexContent>
4629         <extension base="sca:CommonExtensionBase">
4630             <attribute name="name" type="QName"/>
4631         </extension>
4632     </complexContent>
4633 </complexType>
4634
4635 <!-- Extensions element -->
4636 <element name="extensions">
4637     <complexType>
4638         <sequence>
4639             <any namespace="##other" processContents="lax"
4640                 minOccurs="1" maxOccurs="unbounded"/>
4641         </sequence>
4642     </complexType>
4643 </element>
4644
4645 <!-- Intents within WSDL documents -->
4646 <attribute name="requires" type="sca:listOfQNames"/>
4647
4648 <!-- Global attribute definition for @callback to mark a WSDL port type
4649     as having a callback interface defined in terms of a second port
4650     type. -->
4651 <attribute name="callback" type="anyURI"/>
4652
4653 <!-- Value type definition for property values -->
4654 <element name="value" type="sca:ValueType"/>
4655 <complexType name="ValueType" mixed="true">
4656     <sequence>
4657         <any namespace="##any" processContents="lax" minOccurs="0"
4658             maxOccurs="unbounded"/>
4659     </sequence>
4660     <!-- mixed="true" to handle simple type -->
4661     <anyAttribute namespace="##any" processContents="lax"/>
4662 </complexType>
4663

```

```

4664 <!-- Miscellaneous simple type definitions -->
4665 <simpleType name="Multiplicity">
4666   <restriction base="string">
4667     <enumeration value="0..1"/>
4668     <enumeration value="1..1"/>
4669     <enumeration value="0..n"/>
4670     <enumeration value="1..n"/>
4671   </restriction>
4672 </simpleType>
4673
4674 <simpleType name="OverrideOptions">
4675   <restriction base="string">
4676     <enumeration value="no"/>
4677     <enumeration value="may"/>
4678     <enumeration value="must"/>
4679   </restriction>
4680 </simpleType>
4681
4682 <simpleType name="listOfQNames">
4683   <list itemType="QName"/>
4684 </simpleType>
4685
4686 <simpleType name="listOfAnyURIs">
4687   <list itemType="anyURI"/>
4688 </simpleType>
4689
4690 <simpleType name="CreateResource">
4691   <restriction base="string">
4692     <enumeration value="always" />
4693     <enumeration value="never" />
4694     <enumeration value="ifnotexist" />
4695   </restriction>
4696 </simpleType>
4697 </schema>

```

4698 **A.3 sca-binding-sca.xsd**

```

4699 <?xml version="1.0" encoding="UTF-8"?>
4700 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
4701   OASIS trademark, IPR and other policies apply. -->
4702 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4703   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4704   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4705   elementFormDefault="qualified">
4706
4707   <include schemaLocation="sca-core-1.1-cd05.xsd"/>
4708
4709   <!-- SCA Binding -->
4710   <element name="binding.sca" type="sca:SCABinding"
4711     substitutionGroup="sca:binding"/>
4712   <complexType name="SCABinding">
4713     <complexContent>
4714       <extension base="sca:Binding"/>
4715     </complexContent>
4716   </complexType>
4717
4718 </schema>

```

4719 **A.4 sca-interface-java.xsd**

4720 Is described in the [SCA Java Common Annotations and APIs specification \[SCA-Common-Java\]](#).

4721 A.5 sca-interface-wsdl.xsd

```
4722 <?xml version="1.0" encoding="UTF-8"?>
4723 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
4724 OASIS trademark, IPR and other policies apply. -->
4725 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4726 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4727 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4728 elementFormDefault="qualified">
4729
4730 <include schemaLocation="sca-core-1.1-cd05.xsd"/>
4731
4732 <!-- WSDL Interface -->
4733 <element name="interface.wsdl" type="sca:WSDLPortType"
4734 substitutionGroup="sca:interface"/>
4735 <complexType name="WSDLPortType">
4736 <complexContent>
4737 <extension base="sca:Interface">
4738 <sequence>
4739 <any namespace="##other" processContents="lax" minOccurs="0"
4740 maxOccurs="unbounded"/>
4741 </sequence>
4742 <attribute name="interface" type="anyURI" use="required"/>
4743 <attribute name="callbackInterface" type="anyURI"
4744 use="optional"/>
4745 </extension>
4746 </complexContent>
4747 </complexType>
4748
4749 </schema>
```

4750 A.6 sca-implementation-java.xsd

4751 Is described in the [Java Component Implementation specification](#) [SCA-Java]

4752 A.7 sca-implementation-composite.xsd

```
4753 <?xml version="1.0" encoding="UTF-8"?>
4754 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
4755 OASIS trademark, IPR and other policies apply. -->
4756 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4757 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4758 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4759 elementFormDefault="qualified">
4760
4761 <include schemaLocation="sca-core-1.1-cd05.xsd"/>
4762
4763 <!-- Composite Implementation -->
4764 <element name="implementation.composite" type="sca:SCAImplementation"
4765 substitutionGroup="sca:implementation"/>
4766 <complexType name="SCAImplementation">
4767 <complexContent>
4768 <extension base="sca:Implementation">
4769 <sequence>
4770 <any namespace="##other" processContents="lax" minOccurs="0"
4771 maxOccurs="unbounded"/>
4772 </sequence>
4773 <attribute name="name" type="QName" use="required"/>
4774 </extension>
4775 </complexContent>
4776 </complexType>
4777
```

4778 </schema>

4779 **A.8 sca-binding-webservice.xsd**

4780 Is described in [the SCA Web Services Binding specification \[SCA-WSBINDING\]](#)

4781 **A.9 sca-binding-jms.xsd**

4782 Is described in [the SCA JMS Binding specification \[SCA-JMSBINDING\]](#)

4783 **A.10 sca-policy.xsd**

4784 Is described in [the SCA Policy Framework specification \[SCA-POLICY\]](#)

4785 **A.11 sca-contribution.xsd**

```
4786 <?xml version="1.0" encoding="UTF-8"?>
4787 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
4788 OASIS trademark, IPR and other policies apply. -->
4789 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4790 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4791 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4792 elementFormDefault="qualified">
4793
4794 <include schemaLocation="sca-core-1.1-cd05.xsd"/>
4795
4796 <!-- Contribution -->
4797 <element name="contribution" type="sca:ContributionType"/>
4798 <complexType name="ContributionType">
4799 <complexContent>
4800 <extension base="sca:CommonExtensionBase">
4801 <sequence>
4802 <element name="deployable" type="sca:DeployableType"
4803 minOccurs="0" maxOccurs="unbounded"/>
4804 <element ref="sca:importBase" minOccurs="0"
4805 maxOccurs="unbounded"/>
4806 <element ref="sca:exportBase" minOccurs="0"
4807 maxOccurs="unbounded"/>
4808 <element ref="sca:extensions" minOccurs="0" maxOccurs="1"/>
4809 </sequence>
4810 </extension>
4811 </complexContent>
4812 </complexType>
4813
4814 <!-- Deployable -->
4815 <complexType name="DeployableType">
4816 <complexContent>
4817 <extension base="sca:CommonExtensionBase">
4818 <sequence>
4819 <any namespace="##other" processContents="lax" minOccurs="0"
4820 maxOccurs="unbounded"/>
4821 </sequence>
4822 <attribute name="composite" type="QName" use="required"/>
4823 </extension>
4824 </complexContent>
4825 </complexType>
4826
4827 <!-- Import -->
4828 <element name="importBase" type="sca:Import" abstract="true" />
4829 <complexType name="Import" abstract="true">
4830 <complexContent>
```

```

4831     <extension base="sca:CommonExtensionBase">
4832         <sequence>
4833             <any namespace="##other" processContents="lax" minOccurs="0"
4834                 maxOccurs="unbounded"/>
4835         </sequence>
4836     </extension>
4837 </complexType>
4838 </complexType>
4839
4840 <element name="import" type="sca:ImportType"
4841     substitutionGroup="sca:importBase"/>
4842 <complexType name="ImportType">
4843     <complexContent>
4844         <extension base="sca:Import">
4845             <attribute name="namespace" type="string" use="required"/>
4846             <attribute name="location" type="anyURI" use="optional"/>
4847         </extension>
4848     </complexContent>
4849 </complexType>
4850
4851 <!-- Export -->
4852 <element name="exportBase" type="sca:Export" abstract="true" />
4853 <complexType name="Export" abstract="true">
4854     <complexContent>
4855         <extension base="sca:CommonExtensionBase">
4856             <sequence>
4857                 <any namespace="##other" processContents="lax" minOccurs="0"
4858                     maxOccurs="unbounded"/>
4859             </sequence>
4860         </extension>
4861     </complexContent>
4862 </complexType>
4863
4864 <element name="export" type="sca:ExportType"
4865     substitutionGroup="sca:exportBase"/>
4866 <complexType name="ExportType">
4867     <complexContent>
4868         <extension base="sca:Export">
4869             <attribute name="namespace" type="string" use="required"/>
4870         </extension>
4871     </complexContent>
4872 </complexType>
4873
4874 </schema>

```

4875 A.12 sca-definitions.xsd

```

4876 <?xml version="1.0" encoding="UTF-8"?>
4877 <!-- Copyright (C) OASIS (R) 2005,2009. All Rights Reserved.
4878     OASIS trademark, IPR and other policies apply. -->
4879 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4880     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4881     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200912"
4882     elementFormDefault="qualified">
4883
4884     <include schemaLocation="sca-core-1.1-cd05.xsd"/>
4885     <include schemaLocation="sca-policy-1.1-cd03.xsd"/>
4886
4887     <!-- Definitions -->
4888     <element name="definitions" type="sca:tDefinitions"/>
4889     <complexType name="tDefinitions">
4890         <complexContent>
4891             <extension base="sca:CommonExtensionBase">

```

```
4892     <choice minOccurs="0" maxOccurs="unbounded">
4893         <element ref="sca:intent"/>
4894         <element ref="sca:policySet"/>
4895         <element ref="sca:bindingType"/>
4896         <element ref="sca:implementationType"/>
4897         <element ref="sca:externalAttachment"/>
4898         <any namespace="##other" processContents="lax"
4899             minOccurs="0" maxOccurs="unbounded"/>
4900     </choice>
4901     <attribute name="targetNamespace" type="anyURI" use="required"/>
4902 </extension>
4903 </complexContent>
4904 </complexType>
4905
4906 </schema>
```

4907

B. SCA Concepts

4908

B.1 Binding

4909
4910
4911

Bindings are used by services and references. References use bindings to describe the access mechanism used to call the service to which they are wired. Services use bindings to describe the access mechanism(s) that clients use to call the service.

4912
4913
4914

SCA supports multiple different types of bindings. Examples include **SCA service**, **Web service**, **stateless session EJB**, **database stored procedure**, **EIS service**. SCA provides an extensibility mechanism by which an SCA runtime can add support for additional binding types.

4915

B.2 Component

4916
4917
4918
4919
4920
4921

SCA components are configured instances of **SCA implementations**, which provide and consume services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines an **extensibility mechanism** that allows you to introduce new implementation types. The current specification does not mandate the implementation technologies to be supported by an SCA runtime, vendors can choose to support the ones that are important for them. A single SCA implementation can be used by multiple Components, each with a different configuration.

4922
4923
4924
4925
4926

The Component has a reference to an implementation of which it is an instance, a set of property values, and a set of service reference values. Property values define the values of the properties of the component as defined by the component's implementation. Reference values define the services that resolve the references of the component as defined by its implementation. These values can either be a particular service of a particular component, or a reference of the containing composite.

4927

B.3 Service

4928
4929
4930
4931
4932

SCA services are used to declare the externally accessible services of an **implementation**. For a composite, a service is typically provided by a service of a component within the composite, or by a reference defined by the composite. The latter case allows the republication of a service with a new address and/or new bindings. The service can be thought of as a point at which messages from external clients enter a composite or implementation.

4933
4934
4935
4936
4937

A service represents an addressable set of operations of an implementation that are designed to be exposed for use by other implementations or exposed publicly for use elsewhere (e.g. public Web services for use by other organizations). The operations provided by a service are specified by an Interface, as are the operations needed by the service client (if there is one). An implementation can contain multiple services, when it is possible to address the services of the implementation separately.

4938
4939
4940
4941

A service can be provided **as SCA remote services, as Web services, as stateless session EJB's, as EIS services, and so on**. Services use **bindings** to describe the way in which they are published. SCA provides an **extensibility mechanism** that makes it possible to introduce new binding types for new types of services.

4942

B.3.1 Remotable Service

4943
4944
4945
4946

A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA architecture. For example, SCA services of SCA implementations can define implementations of industry-standard web services. Remotable services use pass-by-value semantics for parameters and returned results.

4947
4948
4949
4950
4951

Interfaces can be identified as remotable through the <interface /> XML, but are typically specified as remotable using a component implementation technology specific mechanism, such as Java annotations. See the relevant SCA Implementation Specification for more information. As an example, to define a Remotable Service, a Component implemented in Java would have a Java Interface with the @Remotable annotation

4952 **B.3.2 Local Service**

4953 Local services are services that are designed to be only used “locally” by other implementations that are
4954 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4955 Local services can rely on by-reference calling conventions, or can assume a very fine-grained interaction
4956 style that is incompatible with remote distribution. They can also use technology-specific data-types.

4957 How a Service is identified as local is dependant on the Component implementation technology used.

4958 See the relevant SCA Implementation Specification for more information. As an example, to define a
4959 Local Service, a Component implemented in Java would define a Java Interface that does not have the
4960 @Remotable annotation.

4961 **B.4 Reference**

4962 **SCA references** represent a dependency that an implementation has on a service that is provided by
4963 some other implementation, where the service to be used is specified through configuration. In other
4964 words, a reference is a service that an implementation can call during the execution of its business
4965 function. References are typed by an interface.

4966 For composites, composite references can be accessed by components within the composite like any
4967 service provided by a component within the composite. Composite references can be used as the targets
4968 of wires from component references when configuring Components.

4969 A composite reference can be used to access a service such as: an SCA service provided by another
4970 SCA composite, a Web service, a stateless session EJB, a database stored procedure or an EIS service,
4971 and so on. References use **bindings** to describe the access method used to their services. SCA provides
4972 an **extensibility mechanism** that allows the introduction of new binding types to references.

4973 **B.5 Implementation**

4974 An implementation is concept that is used to describe a piece of software technology such as a Java
4975 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a
4976 service-oriented application. An SCA composite is also an implementation.

4977 Implementations define points of variability including properties that can be set and settable references to
4978 other services. The points of variability are configured by a component that uses the implementation. The
4979 specification refers to the configurable aspects of an implementation as its **componentType**.

4980 **B.6 Interface**

4981 Interfaces define one or more business functions. These business functions are provided by Services
4982 and are used by components through References. Services are defined by the Interface they implement.
4983 SCA currently supports a number of interface type systems, for example:

- 4984 • Java interfaces
- 4985 • WSDL portTypes
- 4986 • C, C++ header files

4987

4988 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional
4989 interface type systems.

4990 Interfaces can be **bi-directional**. A bi-directional service has service operations which are provided by
4991 each end of a service communication – this could be the case where a particular service demands a
4992 “callback” interface on the client, which it calls during the process of handing service requests from the
4993 client.

4994 B.7 Composite

4995 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an
4996 assembly of Components, Services, References, and the Wires that interconnect them. Composites can
4997 be used to contribute elements to an **SCA Domain**.

4998 A **composite** has the following characteristics:

- 4999 • It can be used as a component implementation. When used in this way, it defines a boundary for
5000 Component visibility. Components cannot be directly referenced from outside of the composite in
5001 which they are declared.
- 5002 • It can be used to define a unit of deployment. Composites are used to contribute business logic
5003 artifacts to an SCA Domain.

5004 B.8 Composite inclusion

5005 One composite can be used to provide part of the definition of another composite, through the process of
5006 inclusion. This is intended to make team development of large composites easier. Included composites
5007 are merged together into the using composite at deployment time to form a single logical composite.

5008 Composites are included into other composites through `<include.../>` elements in the using composite.
5009 The SCA Domain uses composites in a similar way, through the deployment of composite files to a
5010 specific location.

5011 B.9 Property

5012 **Properties** allow for the configuration of an implementation with externally set data values. The data
5013 value is provided through a Component, possibly sourced from the property of a containing composite.

5014 Each Property is defined by the implementation. Properties can be defined directly through the
5015 implementation language or through annotations of implementations, where the implementation language
5016 permits, or through a componentType file. A Property can be either a simple data type or a complex data
5017 type. For complex data types, XML schema is the preferred technology for defining the data types.

5018 B.10 Domain

5019 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled
5020 by a single organization. As an example, for the accounts department in a business, the SCA Domain
5021 might cover all finance-related functions, and it might contain a series of composites dealing with specific
5022 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

5023 A Domain specifies the instantiation, configuration and connection of a set of components, provided via
5024 one or more composite files. A Domain also contains Wires that connect together the Components. A
5025 Domain does not contain promoted Services or promoted References, since promotion has no meaning
5026 at the Domain level.

5027 B.11 Wire

5028 **SCA wires** connect **service references** to **services**.

5029 Valid wire sources are component references. Valid wire targets are component services.

5030 When using included composites, the sources and targets of the wires don't have to be declared in the
5031 same composite as the composite that contains the wire. The sources and targets can be defined by
5032 other included composites. Targets can also be external to the SCA Domain.

5033 B.12 SCA Runtime

5034 An SCA Runtime is a set of one or more software programs which, when executed, can accept and run
5035 SCA artifacts as defined in the SCA specifications. An SCA runtime provides an implementation of the
5036 SCA Domain and an implementation of capabilities for populating the domain with artifacts and with

5037 capabilities for running specific artifacts. An SCA Runtime can vary in size and organization and can
5038 involve a single process running on a single machine, multiple processes running on a single machine or
5039 multiple processes running across multiple machines that are linked by network communications.

5040 An SCA runtime supports at least one SCA implementation type and also supports at least one binding
5041 type.

5042 SCA Runtimes can include tools provided to assist developers in creating, testing and debugging of SCA
5043 applications and can be used to host and run SCA applications that provide business capabilities.

5044 An SCA runtime can be implemented using any technologies (i.e. it is not restricted to be implemented
5045 using any particular technologies) and it can be hosted on any operating system platform.

5046

C. Conformance Items

5047

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

5048

C.1 Mandatory Items

Conformance ID	Description
[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.
[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.
[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]	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: <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

	<p>XML type</p> <ul style="list-style-type: none"> 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
[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.
[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 <i>method</i>

	<i>or operation overloading.</i>
[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.
[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".
[ASM12013]	<p>For components at the Domain level, with references for which @autowire="true" applies, the behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:</p> <ol style="list-style-type: none"> 1) The SCA runtime disallows deployment of any components with autowire references. In this case, the SCA runtime MUST raise an exception at the point where the component is deployed. 2) The SCA runtime evaluates the target(s) for the reference at the time that the component is deployed and does not update those targets when later deployment actions occur. 3) The SCA runtime re-evaluates the target(s) for the reference dynamically as later deployment actions occur resulting in updated reference targets which match the new Domain configuration. How the reconfiguration of the reference takes place is described by the relevant client and implementation specifications.

[ASM12015]	Where components are updated by deployment actions (their configuration is changed in some way, which includes changing the wires of component references), the new configuration MUST apply to all new instances of those components once the update is complete.
[ASM12017]	Where a component that is the target of a wire is removed, without the wire being changed, then future invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is the result of the autowire process, the SCA runtime MUST: <ul style="list-style-type: none"> • either cause future invocation of the target component's services to fail with a ServiceUnavailable fault • or alternatively, if an alternative target component is available that satisfies the autowire process, update the reference of the source component
[ASM12020]	Where a component is added to the Domain that is a potential target for a domain level component reference where that reference is marked as @autowire=true, the SCA runtime MUST: <ul style="list-style-type: none"> • either update the references for the source component once the new component is running. • or alternatively, defer the updating of the references of the source component until the source component is stopped and restarted.
[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 locate artifacts (i.e. only a one-level search is performed). 2. from the contents of the contribution itself.
[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.
[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]	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 <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
[ASM14003]	Where errors are only detected at runtime, when the error is detected an error MUST be raised to the component that is attempting the activity concerned with the error.
[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.

5049 **C.2 Non-mandatory Items**

Conformance ID	Description	Classification
[ASM60021]	For the case of an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA runtime SHOULD issue a warning.	Development
[ASM12002]	Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy	Interoperation

	named META-INF	
[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 contributions.	Development
[ASM12008]	An SCA runtime MAY provide the contribution operation functions (install Contribution, update Contribution, add Deployment Composite, update Deployment Composite, remove Contribution).	Enhancement
[ASM12014]	Where <wire/> elements are added, removed or replaced by deployment actions, the components whose references are affected by those deployment actions MAY have their references updated by the SCA runtime dynamically without the need to stop and start those components.	Enhancement
[ASM12016]	An SCA runtime MAY choose to maintain existing instances with the old configuration of components updated by deployment actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.	Enhancement
[ASM12018]	Where a component that is the target of a wire is updated, future invocations of that reference SHOULD use the updated component.	Enhancement
[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
[ASM12030]	For XML definitions, which are identified by QNames, the @namespace attribute of the export element SHOULD be the namespace URI for the exported definitions.	Interoperation
[ASM14001]	An SCA runtime SHOULD detect errors at deployment time where those errors can be found through static analysis.	Development
[ASM14002]	The SCA runtime SHOULD prevent deployment of contributions that are in error, and raise the error to the process performing the deployment (e.g. write a message to an interactive console or write a message to a log file).	Development
[ASM14004]	When an error that could have been detected through static analysis is detected and raised at runtime for a component, the component SHOULD NOT be run until the error is fixed.	Development

5051

D. Acknowledgements

5052 The following individuals have participated in the creation of this specification and are gratefully
5053 acknowledged:

5054 **Participants:**

5055

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 Kavantzas	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.*

5056

5057

E. Revision History

5058 [optional; should not be included in OASIS Standards]

5059

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

			<p>be completed, all implementation statements on services, references, and properties should go here</p> <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 - 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 Combellack'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-Rev 8	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,

			<p>5.4.2, 6.0, add new 7.2, old 7.2 Note: REMOVED assertions: ASM60015 ASM60015 ASM60016 ASM60017 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,</p>

			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, 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

5060