



Service Component Architecture Assembly Model Specification Version 1.1

Committee Draft 02 Issue104

21st January 2009

Deleted: 14th

Specification URIs:

This Version:

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

Previous Version:

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)

Latest Approved Version:

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, BEA Systems

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/200712>

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" or "Latest Approved Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/sca-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>).

The non-normative errata page for this specification is located at

<http://www.oasis-open.org/committees/sca-assembly/>

Notices

Copyright © OASIS® 2005, 2008. 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", [insert specific trademarked names and abbreviations here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <http://www.oasis-open.org/who/trademark.php> for above guidance.

Table of Contents

Committee Draft 02	1
The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/sca-assembly/	3
Notices	4
Table of Contents	5
1 Introduction	8
1.1 Terminology	8
1.2 Normative References	8
1.3 Naming Conventions	9
2 Overview	11
2.1 Diagram used to Represent SCA Artifacts	12
3 Quick Tour by Sample	14
4 Implementation and ComponentType	15
4.1 Component Type	15
4.1.1 Service	16
4.1.2 Reference	17
4.1.3 Property	19
4.1.4 Implementation	21
4.2 Example ComponentType	21
4.3 Example Implementation	22
5 Component	25
5.1 Implementation	26
5.2 Service	27
5.3 Reference	29
5.3.1 Specifying the Target Service(s) for a Reference	31
5.4 Property	32
5.5 Example Component	35
6 Composite	39
6.1 Service	41
6.1.1 Service Examples	42
6.2 Reference	44
6.2.1 Example Reference	46
6.3 Property	48
6.3.1 Property Examples	49
6.4 Wire	51
6.4.1 Wire Examples	53
6.4.2 Autowire	55
6.4.3 Autowire Examples	56
6.5 Using Composites as Component Implementations	59
6.5.1 Example of Composite used as a Component Implementation	60
6.6 Using Composites through Inclusion	60
6.6.1 Included Composite Examples	61
6.7 Composites which Include Component Implementations of Multiple Types	64

Deleted: 20

6.8	Structural URI of Components.....	64
7	ConstrainingType	66
7.1	Example constrainingType	67
8	Interface.....	69
8.1	Local and Remotable Interfaces.....	69
8.2	Bidirectional Interfaces	70
8.3	Conversational Interfaces	72
8.4	Long-running Request-Response Operations	74
8.4.1	Background	74
8.4.2	Definition of "long-running"	74
8.4.3	The asyncInvoation Intent	74
8.4.4	Requirements on Bindings	75
8.4.5	Implementation Type Support	75
8.5	SCA-Specific Aspects for WSDL Interfaces	75
8.6	WSDL Interface Type	77
8.6.1	Example of interface.wsdl	77
9	Binding.....	78
9.1	Messages containing Data not defined in the Service Interface	80
9.2	WireFormat	80
9.3	OperationSelector	81
9.4	Form of the URI of a Deployed Binding.....	81
9.4.1	Non-hierarchical URIs	81
9.4.2	Determining the URI scheme of a deployed binding.....	81
9.5	SCA Binding	82
9.5.1	Example SCA Binding	83
9.6	Web Service Binding	83
9.7	JMS Binding.....	83
10	SCA Definitions	84
11	Extension Model.....	85
11.1	Defining an Interface Type.....	85
11.2	Defining an Implementation Type.....	87
11.3	Defining a Binding Type.....	88
11.4	Defining an Import Type	90
11.5	Defining an Export Type	92
12	Packaging and Deployment	94
12.1	Domains.....	94
12.2	Contributions.....	94
12.2.1	SCA Artifact Resolution.....	95
12.2.2	SCA Contribution Metadata Document	97
12.2.3	Contribution Packaging using ZIP	99
12.3	Installed Contribution	99
12.3.1	Installed Artifact URIs.....	100
12.4	Operations for Contributions.....	100
12.4.1	install Contribution & update Contribution	100
12.4.2	add Deployment Composite & update Deployment Composite.....	101

12.4.3	remove Contribution	101
12.5	Use of Existing (non-SCA) Mechanisms for Resolving Artifacts	101
12.6	Domain-Level Composite	101
12.6.1	add To Domain-Level Composite.....	102
12.6.2	remove From Domain-Level Composite	102
12.6.3	get Domain-Level Composite	102
12.6.4	get QName Definition	102
12.7	Dynamic Behaviour of Wires in the SCA Domain	103
12.8	Dynamic Behaviour of Component Property Values	103
13	Conformance	105
A.	XML Schemas	106
A.1	sca.xsd	106
A.2	sca-core.xsd	106
A.3	sca-binding-sca.xsd.....	114
A.4	sca-interface-java.xsd	115
A.5	sca-interface-wsdl.xsd	115
A.6	sca-implementation-java.xsd	116
A.7	sca-implementation-composite.xsd	117
A.8	sca-definitions.xsd	117
A.9	sca-binding-webservice.xsd	118
A.10	sca-binding-jms.xsd.....	118
A.11	sca-policy.xsd	118
A.12	sca-contribution.xsd	118
B.	SCA Concepts	121
B.1	Binding.....	121
B.2	Component.....	121
B.3	Service.....	121
B.3.1	Remotable Service.....	121
B.3.2	Local Service.....	122
B.4	Reference	122
B.5	Implementation	122
B.6	Interface.....	122
B.7	Composite	123
B.8	Composite inclusion	123
B.9	Property	123
B.10	Domain	123
B.11	Wire	123
C.	Conformance Items	125
D.	Acknowledgements	133
E.	Non-Normative Text	134
F.	Revision History.....	135

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 should be 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*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [SCA-Java] SCA Java Component Implementation Specification
http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf
- [SCA-Common-Java] SCA Java Common Annotations and APIs Specification
http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf
- [SCA BPEL] SCA BPEL Client and Implementation Specification
<http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec-cd-01.pdf>
- [SDO] SDO Specification
<http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf>
- [3] SCA Example Code document
http://www.osoa.org/download/attachments/28/SCA_BuildingYourFirstApplication_V09.pdf
- [4] JAX-WS Specification
<http://jcp.org/en/jsr/detail?id=101>

- 39 [5] WS-I Basic Profile
40 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>
41
- 42 [6] WS-I Basic Security Profile
43 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>
44
- 45 [7] Business Process Execution Language (BPEL)
46 http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel
47
- 48 [8] WSDL Specification
49 WSDL 1.1: <http://www.w3.org/TR/wsd>
50 WSDL 2.0: <http://www.w3.org/TR/wsd20/>
51
- 52 [9] SCA Web Services Binding Specification
53 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd01.pdf>
54
- 55 [10] SCA Policy Framework Specification
56 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf>
57
- 58 [11] SCA JMS Binding Specification
59 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-jmsbinding-1.1-spec-cd01.pdf>
60
- 61 [SCA-CPP-Client] SCA C++ Client and Implementation Specification
62 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd-01.pdf>
63
- 64 [SCA-C-Client] SCA C Client and Implementation Specification
65 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd-01.pdf>
66
- 67 [12] ZIP Format Definition
68 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>
69
- 70 [13] Infoset Specification
71 <http://www.w3.org/TR/xml-infoset/>
72
- 73 [WSDL11_Identifiers] WSDL 1.1 Element Identifiers
74 <http://www.w3.org/TR/wsd11elementidentifiers/>
75

76 1.3 Naming Conventions

77

78 This specification follows some naming conventions for artifacts defined by the specification,

79 as follows:

80

- 81 • For the names of elements and the names of attributes within XSD files, the names follow the
82 CamelCase convention, with all names starting with a lower case letter.
83 eg <element name="componentType" type="sca:ComponentType"/>
- 84 • For the names of types within XSD files, the names follow the CamelCase convention with all
85 names starting with an upper case letter.
86 eg. <complexType name="ComponentService">
- 87 • For the names of intents, the names follow the CamelCase convention, with all names starting
88 with a lower case letter, EXCEPT for cases where the intent represents an established acronym,
89 in which case the entire name is in upper case.
90 An example of an intent which is an acronym is the "SOAP" intent.

91 2 Overview

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

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

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

108 One basic artifact of SCA is the **component**, which is the unit of construction for SCA. A component
109 consists of a configured instance of an implementation, where an implementation is the piece of program
110 code providing business functions. The business function is offered for use by other components as
111 **services**. Implementations can depend on services provided by other components – these dependencies
112 are called **references**. Implementations can have settable **properties**, which are data values which
113 influence the operation of the business function. The component **configures** the implementation by
114 providing values for the properties and by wiring the references to services provided by other
115 components.

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

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

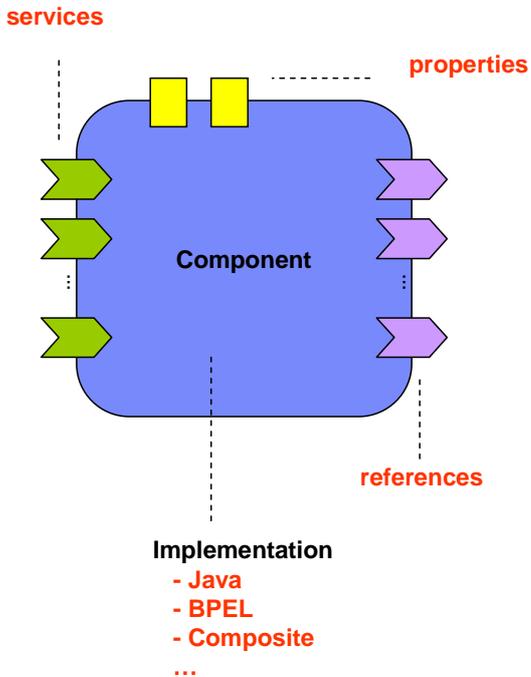
129 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services
130 providing an area of business functionality that is controlled by a single organization. As an example, for
131 the accounts department in a business, the SCA Domain might cover all financial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

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

141 **2.1 Diagram used to Represent SCA Artifacts**

142 This document introduces diagrams to represent the various SCA artifacts, as a way of visualizing the
143 relationships between the artifacts in a particular assembly. These diagrams are used in this document to
144 accompany and illuminate the examples of SCA artifacts.

145 The following picture illustrates some of the features of an SCA component:



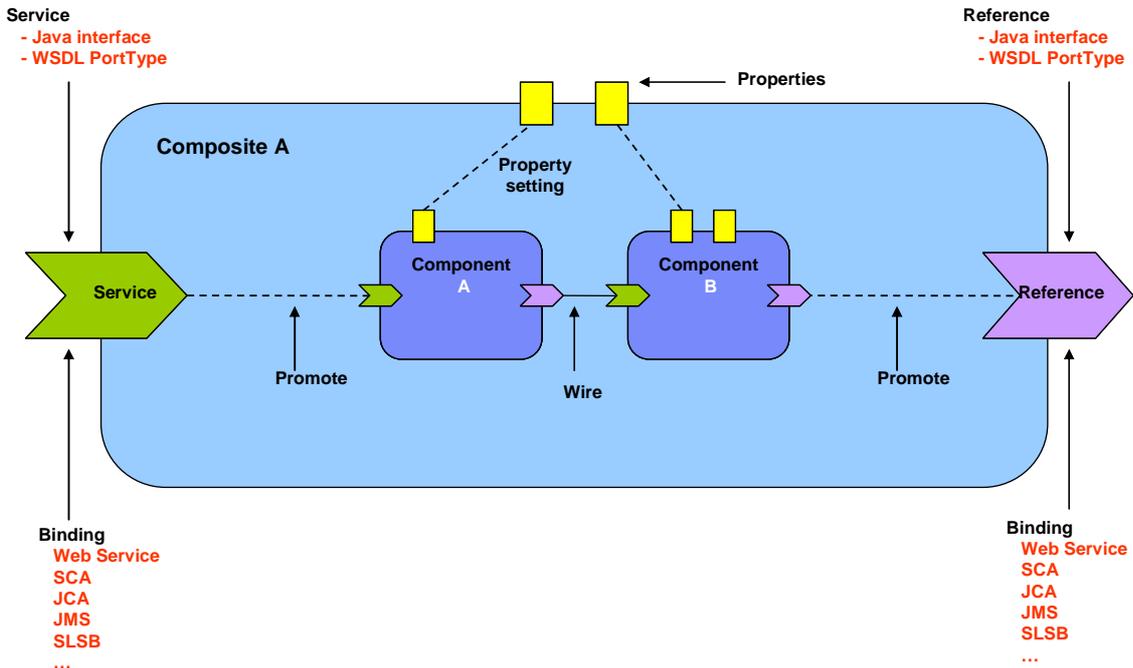
146

147 *Figure 1: SCA Component Diagram*

148

149 The following picture illustrates some of the features of a composite assembled using a set of
150 components:

151

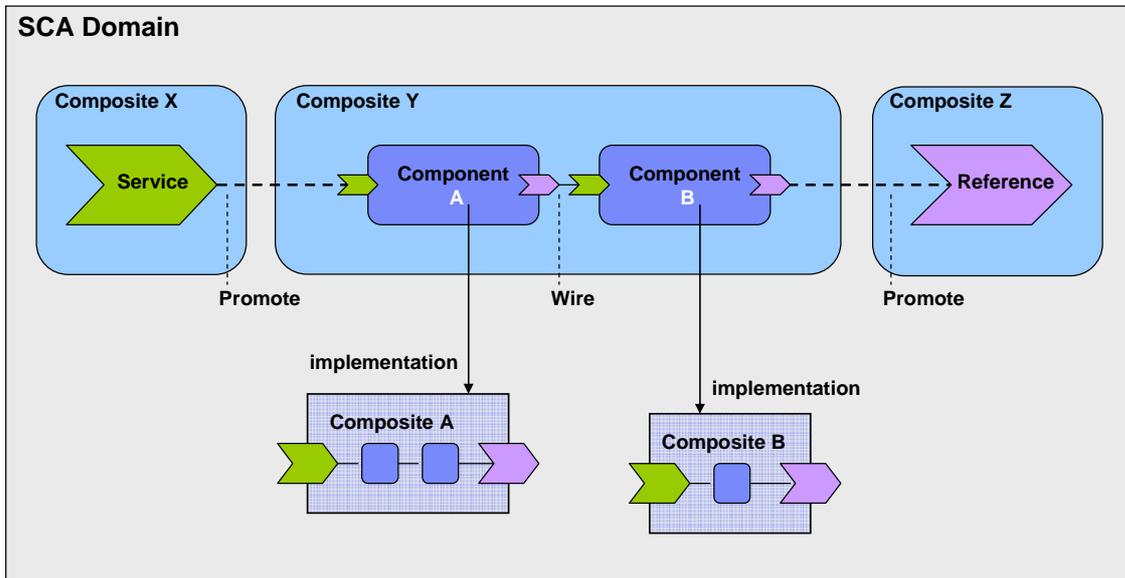


152

153 Figure 2: SCA Composite Diagram

154

155 The following picture illustrates an SCA Domain assembled from a series of high-level composites, some
 156 of which are in turn implemented by lower-level composites:



157

158 Figure 3: SCA Domain Diagram

159 **3 Quick Tour by Sample**

160 To be completed.

161

162 This section is intended to contain a sample which describes the key concepts of SCA.

163

164

165 4 Implementation and ComponentType

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

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

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

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

179 So, for example:

- 180 • for a service, the implementation might define the interface, binding(s), a URI, intents,
181 and policy sets, including details of the bindings
- 182 • for a reference, the implementation might define the interface, binding(s), target URI(s),
183 intents, policy sets, including details of the bindings
- 184 • for a property the implementation might define its type and a default value
- 185 • the implementation itself might define policy intents or concrete policy sets

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

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

194

195 4.1 Component Type

196 **Component type** represents the configurable aspects of an implementation. A component type
197 consists of services that are offered, references to other services that can be wired and properties
198 that can be set. The settable properties and the settable references to services are configured by a
199 component that uses the implementation.

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

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

210 The implementation type specification defines whether introspection is allowed, whether a side file
211 is allowed, both are allowed or some other mechanism specifies the component type. The
212 component type information derived through introspection is called the **introspected component**
213 **type**. In any case, the implementation type specification specifies how multiple sources of
214 information are combined to produce the **effective component type**. The effective component
215 type is the component type metadata that is presented to the using Component for configuration.

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

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

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

224 The following snippet shows the componentType pseudo-schema:

225

```
226 <?xml version="1.0" encoding="ASCII"?>  
227 <!-- Component type schema snippet -->  
228 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
229     constrainingType="QName"? >  
230  
231     <service ... /*>  
232     <reference ... /*>  
233     <property ... /*>  
234     <implementation ... /*?>  
235  
236 </componentType>  
237
```

238 The **componentType** element has the following **attribute**:

- 239 • **constrainingType : QName (0..1)** – If present, the @constrainingType attribute of a
240 <componentType/> element MUST reference a <constrainingType/> element in the
241 Domain through its QName. [ASM40002] When specified, the set of services, references
242 and properties of the implementation, plus related intents, is constrained to the set
243 defined by the constrainingType. See [the ConstrainingType Section](#) for more details.

244

245 The **componentType** element has the following **child elements**:

- 246 • **service : Service (0..n)** – [see component type service section](#).
- 247 • **reference : Reference (0..n)** – [see component type reference section](#).
- 248 • **property : Property (0..n)** – [see component type property section](#).
- 249 • **implementation : Implementation (0..1)** – [see component type implementation](#)
250 [section](#).

251

252 4.1.1 Service

253 **A Service** represents an addressable interface of the implementation. The service is represented
254 by a **service element** which is a child of the componentType element. There can be **zero or**
255 **more** service elements in a componentType. The following snippet shows the component type
256 schema with the schema for a service child element:

257

```

258 <?xml version="1.0" encoding="ASCII"?>
259 <!-- Component type service schema snippet -->
260 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
261 >
262
263     <service name="xs:NCName"
264         requires="list of xs:QName"? policySets="list of xs:QName"?>*
265         <interface ... />
266         <operation name="xs:NCName" requires="list of xs:QName"?
267             policySets="list of xs:QName"?/>*
268         <binding ... />*
269         <callback?
270             <binding ... />+
271         </callback>
272     </service>
273
274     <reference ... />*
275     <property ... />*
276     <implementation ... />?
277
278 </componentType>
279

```

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

- **name : NCName (1..1)** - the name of the service. The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>. [ASM40003]
- **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
- **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.

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

- **interface : Interface (1..1)** - A service has **one interface**, which describes the operations provided by the service. For details on the interface element see [the Interface section](#).
- **operation: Operation (0..n)** - Zero or more operation elements. These elements are used to describe characteristics of individual operations within the interface. For a detailed description of the operation element, see [the Policy Framework specification \[SCA Policy\]](#).
- **binding : Binding (0..n)** - A service element has **zero or more binding elements** as children. If the binding element is not present it defaults to <binding.sca>. Details of the binding element are described in [the Bindings section](#).
- **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent. For details on callbacks, see [the Bidirectional Interfaces section](#).

4.1.2 Reference

A **Reference** represents a requirement that the implementation has on a service provided by another component. The reference is represented by a **reference element** which is a child of the

309 componentType element. There can be **zero or more** reference elements in a component type
310 definition. The following snippet shows the component type schema with the schema for a
311 reference child element:

312

```
313 <?xml version="1.0" encoding="ASCII"?>
314 <!-- Component type reference schema snippet -->
315 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
316 >
317
318     <service ... />*
319
320     <reference name="xs:NCName"
321         autowire="xs:boolean"?
322         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
323         wiredByImpl="xs:boolean"?
324         requires="list of xs:QName"? policySets="list of xs:QName"?>*
325         <interface ... />
326         <operation name="xs:NCName" requires="list of xs:QName"?
327             policySets="list of xs:QName"? />*
328         <binding ... />*
329         <callback?
330             <binding ... />+
331         </callback>
332     </reference>
333
334     <property ... />*
335     <implementation ... />?
336
337 </componentType>
338
```

339 The **reference** element has the following **attributes**:

- 340 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a
341 <reference/> child element of a <componentType/> MUST be unique amongst the
342 reference elements of that <componentType/>. [ASM40004]
 - 343 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect
344 the reference to target services. The multiplicity can have the following values
 - 345 ○ 0..1 – zero or one wire can have the reference as a source
 - 346 ○ 1..1 – one wire can have the reference as a source
 - 347 ○ 0..n - zero or more wires can have the reference as a source
 - 348 ○ 1..n – one or more wires can have the reference as a source
- 349 If @multiplicity is not specified, the default value is "1..1".
- 350 • **autowire : boolean (0..1)** - whether the reference should be autowired, as described in
351 the [Autowire section](#). Default is false.
 - 352 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the
353 reference is wired to the target(s) configured on the reference. If set to "true" it indicates
354 that the target of the reference is set at runtime by the implementation code (eg by the
355 code obtaining an endpoint reference by some means and setting this as the target of the
356 reference through the use of programming interfaces defined by the relevant Client and
357 Implementation specification). If @wiredByImpl is set to "true", then any reference
358 targets configured for this reference MUST be ignored by the runtime. [ASM40006] It is
359 recommended that any references with @wiredByImpl = "true" are left unwired.

- 360 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)
361 [\[10\]](#) for a description of this attribute.
- 362 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)
363 [\[10\]](#) for a description of this attribute.

364

365 The **reference** element has the following **child elements**:

- 366 • **interface : Interface (1..1)** - A reference has **one interface**, which describes the
367 operations required by the reference. The interface is described by an **interface element**
368 which is a child element of the reference element. For details on the interface element see
369 [the Interface section](#).
- 370 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are
371 used to describe characteristics of individual operations within the interface. For a detailed
372 description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
- 373 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as
374 children. Details of the binding element are described in the [Bindings section](#).

375 Note that a binding element may specify an endpoint which is the target of that binding. A
376 reference must not mix the use of endpoints specified via binding elements with target
377 endpoints specified via the target attribute. If the target attribute is set, then binding
378 elements can only list one or more binding types that can be used for the wires identified
379 by the target attribute. All the binding types identified are available for use on each wire
380 in this case. If endpoints are specified in the binding elements, each endpoint must use
381 the binding type of the binding element in which it is defined. In addition, each binding
382 element needs to specify an endpoint in this case.

- 383 • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional
384 **callback** element used if the interface has a callback defined, which has one or more
385 **binding** elements as children. The **callback** and its binding child elements are specified if
386 there is a need to have binding details used to handle callbacks. If the callback element is
387 not present, the behaviour is runtime implementation dependent. For details on callbacks,
388 see [the Bidirectional Interfaces section](#).

389

390 4.1.3 Property

391 **Properties** allow for the configuration of an implementation with externally set values. Each
392 Property is defined as a property element. The componentType element can have zero or more
393 property elements as its children. The following snippet shows the component type schema with
394 the schema for a reference child element:

395

```
396 <?xml version="1.0" encoding="ASCII"?>
397 <!-- Component type property schema snippet -->
398 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
399 >
400
401     <service ... />*
402     <reference ... >*
403
404     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
405         many="xs:boolean"? mustSupply="xs:boolean"?
406         requires="list of xs:QName"?
407         policySets="list of xs:QName"?>*
408         default-property-value?
409     </property>
410
```

411 <implementation ... />?
412
413 </componentType>
414

415 The **property** element has the following **attributes**:

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

427 A single property element MUST NOT contain both an @type attribute and an @element
428 attribute. [ASM40010]

Formatted: Font color: Red

- 429 ▪ **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-
430 valued (true). In the case of a multi-valued property, it is presented to the implementation
431 as a collection of property values. If many is not specified, it takes a default value of false.
- 432 ▪ **mustSupply : boolean (0..1)** - whether the property value must be supplied by the
433 component that uses the implementation – when mustSupply="true" the component must
434 supply a value since the implementation has no default value for the property. A default-
435 property-value should only be supplied when mustSupply="false" (the default setting for
436 the mustSupply attribute), since the implication of a default value is that it is used only
437 when a value is not supplied by the using component. If mustSupply is not specified, it
438 takes a default value of false.
- 439 ▪ **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property.
- 440 ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)
441 [\[10\]](#) for a description of this attribute.
- 442 ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)
443 [\[10\]](#) for a description of this attribute.

444 The property element can contain a default property value as its content. The form of the default
445 property value is as described [in the section on Component Property](#).

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

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

459

460 4.1.4 Implementation

461 **Implementation** represents characteristics inherent to the implementation itself, in particular
462 intents and policies. See the [Policy Framework specification \[10\]](#) for a description of intents and
463 policies. The following snippet shows the component type schema with the schema for a
464 implementation 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/200712" ...  
469 >  
470  
471     <service ... /*>  
472     <reference ... /*>  
473     <property ... /*>  
474  
475     <implementation requires="list of xs:QName"?  
476                 policySets="list of xs:QName"?/*>  
477  
478 </componentType>  
479
```

480 The **implementationService** element has the following **attributes**:

- 481 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)
482 [\[10\]](#) for a description of this attribute.
- 483 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)
484 [\[10\]](#) for a description of this attribute.

485

486 4.2 Example ComponentType

487

488 The following snippet shows the contents of the componentType file for the MyValueServiceImpl
489 implementation. The componentType file shows the services, references, and properties of the
490 MyValueServiceImpl implementation. In this case, Java is used to define interfaces:

```
491  
492 <?xml version="1.0" encoding="ASCII"?>  
493 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
494  
495     <service name="MyValueService">  
496         <interface.java interface="services.myvalue.MyValueService" /*>  
497     </service>  
498  
499     <reference name="customerService">  
500         <interface.java interface="services.customer.CustomerService" /*>  
501     </reference>  
502     <reference name="stockQuoteService">  
503         <interface.java  
504             interface="services.stockquote.StockQuoteService" /*>  
505     </reference>  
506  
507     <property name="currency" type="xsd:string">USD</property>  
508  
509 </componentType>  
510
```

511 4.3 Example Implementation

512 The following is an example implementation, written in Java. See the [SCA Example Code](#)
513 [document](#) [3] for details.

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

```
516  
517 package services.account;  
518  
519 @Remotable  
520 public interface AccountService {  
521  
522     AccountReport getAccountReport(String customerID);  
523 }  
524
```

525 The following is a full listing of the AccountServiceImpl class, showing the Service it implements,
526 plus the service references it makes and the settable properties that it has. Notice the use of Java
527 annotations to mark SCA aspects of the code, including the @Property and @Reference tags:

```
528  
529 package services.account;  
530  
531 import java.util.List;  
532  
533 import commonj.sdo.DataFactory;  
534  
535 import org.osoa.sca.annotations.Property;  
536 import org.osoa.sca.annotations.Reference;  
537  
538 import services.accountdata.AccountDataService;  
539 import services.accountdata.CheckingAccount;  
540 import services.accountdata.SavingsAccount;  
541 import services.accountdata.StockAccount;  
542 import services.stockquote.StockQuoteService;  
543  
544 public class AccountServiceImpl implements AccountService {  
545  
546     @Property  
547     private String currency = "USD";  
548  
549     @Reference  
550     private AccountDataService accountDataService;  
551     @Reference  
552     private StockQuoteService stockQuoteService;  
553  
554     public AccountReport getAccountReport(String customerID) {  
555  
556         DataFactory dataFactory = DataFactory.INSTANCE;  
557         AccountReport accountReport = (AccountReport) dataFactory.create(AccountReport.class);  
558         List accountSummaries = accountReport.getAccountSummaries();  
559
```

```

560     CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
561     AccountSummary checkingAccountSummary =
562 (AccountSummary)dataFactory.create(AccountSummary.class);
563     checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
564     checkingAccountSummary.setAccountType("checking");
565     checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
566     accountSummaries.add(checkingAccountSummary);
567
568     SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
569     AccountSummary savingsAccountSummary =
570 (AccountSummary)dataFactory.create(AccountSummary.class);
571     savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
572     savingsAccountSummary.setAccountType("savings");
573     savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
574     accountSummaries.add(savingsAccountSummary);
575
576     StockAccount stockAccount = accountDataService.getStockAccount(customerID);
577     AccountSummary stockAccountSummary =
578 (AccountSummary)dataFactory.create(AccountSummary.class);
579     stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
580     stockAccountSummary.setAccountType("stock");
581     float balance=
582 (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
583     stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
584     accountSummaries.add(stockAccountSummary);
585
586     return accountReport;
587 }
588
589 private float fromUSDollarToCurrency(float value){
590
591     if (currency.equals("USD")) return value; else
592     if (currency.equals("EURO")) return value * 0.8f; else
593     return 0.0f;
594 }
595 }

```

597 The following is the equivalent SCA componentType definition for the AccountServiceImpl, derived
598 by reflection against the code above:

```

600 <?xml version="1.0" encoding="ASCII"?>
601 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
602     xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
603
604     <service name="AccountService">
605         <interface.java interface="services.account.AccountService"/>
606     </service>
607     <reference name="accountDataService">
608         <interface.java
609 interface="services.accountdata.AccountDataService"/>

```

```
610     </reference>
611     <reference name="stockQuoteService">
612         <interface java
613 interface="services.stockquote.StockQuoteService"/>
614     </reference>
615
616     <property name="currency" type="xsd:string">USD</property>
617
618 </componentType>
619
```

620 For full details about Java implementations, see the [Java Client and Implementation Specification](#)
621 and the [SCA Example Code](#) document. Other implementation types have their own specification
622 documents.

5 Component

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

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

Components are declared as subelements of a composite in an **xxx.composite** file. A component is represented by a **component element** which is a child of the composite element. There can be **zero or more** component elements within a composite. The following snippet shows the composite schema with the schema for the component child element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
  ...
  <component name="xs:NCName" autowire="xs:boolean"?
             requires="list of xs:QName"? policySets="list of xs:QName"?
             constrainingType="xs:QName"?>*
    <implementation ... />?
    <service ... />*
    <reference ... />*
    <property ... />*
  </component>
  ...
</composite>
```

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

- **name : NCName (1..1)** – the name of the component. The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/> [ASM50001]
- **autowire : boolean (0..1)** – whether contained component references should be autowired, as described in [the Autowire section](#). Default is false.
- **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
- **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
- **constrainingType : QName (0..1)** – the name of a constrainingType. When specified, the set of services, references and properties of the component, plus related intents, is constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#) for more details.

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

- **implementation : ComponentImplementation (0..1)** – see [component implementation section](#).

- 667 • **service** : *ComponentService (0..n)* – see component service section.
 - 668 • **reference** : *ComponentReference (0..n)* – see component reference section.
 - 669 • **property** : *ComponentProperty (0..n)* – see component property section.
- 670

671 5.1 Implementation

672 A component element has **zero or one implementation element** as its child, which points to the
 673 implementation used by the component. A component with no implementation element is not
 674 runnable, but components of this kind may be useful during a "top-down" development process as
 675 a means of defining the characteristics required of the implementation before the implementation
 676 is written.

```
677
678 <?xml version="1.0" encoding="UTF-8"?>
679 <!-- Component Implementation schema snippet -->
680 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
681   ...
682   <component ... >*
683     <implementation ... />?
684     <service ... />*
685     <reference ... />*
686     <property ... />*
687   </component>
688   ...
689 </composite>
690
```

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

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

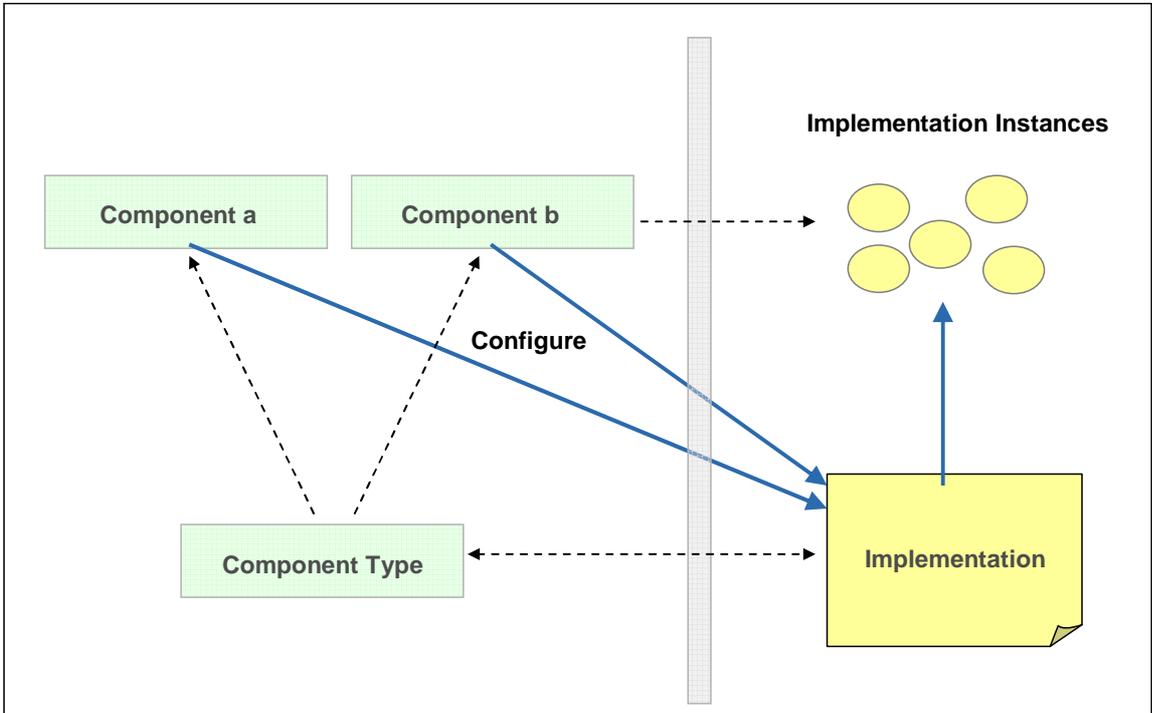
699 The following snippets show implementation elements for the Java and BPEL implementation types
 700 and for the use of a composite as an implementation:

```
701
702 <implementation.java class="services.myvalue.MyValueServiceImpl"/>
703
704 <implementation.bpel process="ans:MoneyTransferProcess"/>
705
706 <implementation.composite name="bns:MyValueComposite"/>
707
708
```

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

710

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



716
 717 *Figure 4: Relationship of Component and Implementation*
 718

719 **5.2 Service**

720 The component element can have **zero or more service elements** as children which are used to
 721 configure the services of the component. The services that can be configured are defined by the
 722 implementation. The following snippet shows the component schema with the schema for a
 723 service child element:

```
724
725 <?xml version="1.0" encoding="UTF-8"?>
726 <!-- Component Service schema snippet -->
727 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
728   ...
729   <component ... >*
730     <implementation ... />?
731     <service name="xs:NCName" requires="list of xs:QName"?
732       policySets="list of xs:QName"?>*
733       <interface ... />?
734       <operation name="xs:NCName" requires="list of xs:QName"?
735         policySets="list of xs:QName"?/>*
736       <binding ... />*
737       <callback?>
```

```

738         <binding ... />+
739     </callback>
740 </service>
741 <reference ... />*
742 <property ... />*
743 </component>
744 ...
745 </composite>
746

```

747 The **component service** element has the following **attributes**:

- 748 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service
749 element of a <component/> MUST be unique amongst the service elements of that
750 <component/> [ASM50002] The @name attribute of a service element of a
751 <component/> MUST match the @name attribute of a service element of the
752 componentType of the <implementation/> child element of the component. [ASM50003]
- 753 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)
754 [10] for a description of this attribute.
755 Note: The effective set of policy intents for the service consists of any intents explicitly
756 stated in this requires attribute, combined with any intents specified for the service by the
757 implementation.
- 758 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)
759 [10] for a description of this attribute.

760
761 The **component service** element has the following **child elements**:

- 762 • **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the
763 operations provided by the service. The interface is described by an **interface element**
764 which is a child element of the service element. If no interface is specified, then the
765 interface specified for the service in the componentType of the implementation is in effect.
766 If a <service/> element has an interface subelement specified, the interface MUST provide
767 a compatible subset of the interface declared on the componentType of the
768 implementation [ASM50004] For details on the interface element see [the Interface section](#).
- 769 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are
770 used to describe characteristics of individual operations within the interface. For a detailed
771 description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
- 772 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as
773 children. If no binding elements are specified for the service, then the bindings specified
774 for the equivalent service in the componentType of the implementation MUST be used, but
775 if the componentType also has no bindings specified, then <binding.sca/> MUST be used
776 as the binding. If binding elements are specified for the service, then those bindings MUST
777 be used and they override any bindings specified for the equivalent service in the
778 componentType of the implementation. [ASM50005] Details of the binding element are
779 described in [the Bindings section](#). The binding, combined with any PolicySets in effect for
780 the binding, needs to satisfy the set of policy intents for the service, as described in [the](#)
781 [Policy Framework specification](#) [10].
- 782 • **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback**
783 element used if the interface has a callback defined, which has one or more **binding**
784 elements as children. The **callback** and its binding child elements are specified if there is
785 a need to have binding details used to handle callbacks. If the callback element is present
786 and contains one or more binding child elements, then those bindings MUST be used for
787 the callback. [ASM50006] If the callback element is not present, the behaviour is runtime
788 implementation dependent.

790 5.3 Reference

791 The component element can have **zero or more reference elements** as children which are used
792 to configure the references of the component. The references that can be configured are defined
793 by the implementation. The following snippet shows the component schema with the schema for a
794 reference child element:

```
795  
796 <?xml version="1.0" encoding="UTF-8"?>  
797 <!-- Component Reference schema snippet -->  
798 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
799   ...  
800   <component ... >*  
801     <implementation ... />?  
802     <service ... />*  
803     <reference name="xs:NCName"  
804       target="list of xs:anyURI"? autowire="xs:boolean"?  
805       multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
806       wiredByImpl="xs:boolean"? requires="list of xs:QName"?  
807       policySets="list of xs:QName"?>*  
808     <interface ... />?  
809     <operation name="xs:NCName" requires="list of xs:QName"?  
810       policySets="list of xs:QName"?/>*  
811     <binding uri="xs:anyURI"? requires="list of xs:QName"?  
812       policySets="list of xs:QName"?/>*  
813     <callback?>  
814       <binding ... />+  
815     </callback>  
816   </reference>  
817   <property ... />*  
818 </component>  
819   ...  
820 </composite>  
821
```

822 The **component reference** element has the following **attributes**:

- 823 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service
824 element of a <component/> MUST be unique amongst the service elements of that
825 <component/> [ASM50007] The @name attribute of a reference element of a
826 <component/> MUST match the @name attribute of a reference element of the
827 componentType of the <implementation/> child element of the component. [ASM50008]
- 828 • **autowire : boolean (0..1)** – whether the reference should be autowired, as described in
829 the [Autowire section](#). Default is false.
- 830 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)
831 [\[10\]](#) for a description of this attribute.
832 Note: The effective set of policy intents for the reference consists of any intents explicitly
833 stated in this requires attribute, combined with any intents specified for the reference by
834 the implementation.
- 835 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
836 [\[10\]](#) for a description of this attribute.
- 837 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect
838 the reference to target services. Overrides the multiplicity specified for this reference in
839 the componentType of the implementation. The multiplicity can have the following values

- 840 ○ 0..1 – zero or one wire can have the reference as a source
- 841 ○ 1..1 – one wire can have the reference as a source

Deleted: 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.

- 842 ○ 0..n - zero or more wires can have the reference as a source
- 843 ○ 1..n - one or more wires can have the reference as a source

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

Deleted: 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.

848 If not present, the value of multiplicity is equal to the multiplicity specified for this
 849 reference in the componentType of the implementation - if not present in the
 850 componentType, the value defaults to 1..1.

- 851 • **target : anyURI (0..n)** - a list of one or more of target service URI's, depending on
 852 multiplicity setting. Each value wires the reference to a component service that resolves
 853 the reference. For more details on wiring see [the section on Wires](#). Overrides any target
 854 specified for this reference on the implementation.
- 855 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default, which indicates that
 856 the implementation wires this reference dynamically. If set to "true" it indicates that the
 857 target of the reference is set at runtime by the implementation code (eg by the code
 858 obtaining an endpoint reference by some means and setting this as the target of the
 859 reference through the use of programming interfaces defined by the relevant Client and
 860 Implementation specification). If @wiredByImpl="true" is set for a reference, then the
 861 reference MUST NOT be wired statically within a composite, but left unwired. [ASM50010]

862

863 The **component reference** element has the following **child elements**:

- 864 • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes
 865 the operations required by the reference. The interface is described by an **interface**
 866 **element** which is a child element of the reference element. If no interface is specified,
 867 then the interface specified for the reference in the componentType of the implementation
 868 is in effect. If an interface is declared for a component reference it MUST provide a
 869 compatible superset of the interface declared for the equivalent reference in the
 870 componentType of the implementation, i.e. provide the same operations and a superset of
 871 the operations defined by the implementation for the reference. [ASM50011] For details
 872 on the interface element see [the Interface section](#).

- 873 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are
 874 used to describe characteristics of individual operations within the interface. For a detailed
 875 description of the operation element, see [the Policy Framework specification](#) [SCA Policy].

- 876 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as
 877 children. If no binding elements are specified for the reference, then the bindings specified
 878 for the equivalent reference in the componentType of the implementation MUST be used,
 879 but if the componentType also has no bindings specified, then <binding.sca/> MUST be
 880 used as the binding. If binding elements are specified for the reference, then those
 881 bindings MUST be used and they override any bindings specified for the equivalent
 882 reference in the componentType of the implementation. [ASM50012] Details of the binding
 883 element are described in the [Bindings section](#). The binding, combined with any PolicySets
 884 in effect for the binding, needs to satisfy the set of policy intents for the reference, as
 885 described in [the Policy Framework specification](#) [10].

Deleted: 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, 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 reference, then those bindings MUST be used and they override any bindings specified for the equivalent reference in the componentType of the implementation.

886 A reference identifies zero or more target services that satisfy the reference. This can be
 887 done in a number of ways, which are fully described in section "5.3.1 Specifying the
 888 Target Service(s) for a Reference"

- 889 • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional
 890 **callback** element used if the interface has a callback defined, which has one or more
 891 **binding** elements as children. The **callback** and its binding child elements are specified if
 892 there is a need to have binding details used to handle callbacks. If the callback element is
 893 present and contains one or more binding child elements, then those bindings MUST be

894 used for the callback. [ASM50006] If the callback element is not present, the behaviour is
895 runtime implementation dependent.

896 5.3.1 Specifying the Target Service(s) for a Reference

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

- 899 1. Through a value specified in the @target attribute of the reference element
- 900 2. Through a target URI specified in the @uri attribute of a binding element which is a child
901 of the reference element
- 902 3. Through the setting of one or more values for binding-specific attributes and/or child
903 elements of a binding element that is a child of the reference element
- 904 4. Through the specification of @autowire="true" for the reference (or through inheritance
905 of that value from the component or composite containing the reference)
- 906 5. Through the specification of @wiredByImpl="true" for the reference
- 907 6. Through the promotion of a component reference by a composite reference of the
908 composite containing the component (the target service is then identified by the
909 configuration of the composite reference)
- 910 7. Through the presence of a <wire/> element which has the reference specified in its
911 @source attribute.

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

- 913 • If @wiredByImpl="true", other methods of specifying the target service MUST NOT be
914 used. [ASM50013]
- 915 • If @autowire="true", the autowire procedure MUST only be used if no target is identified
916 by any of the other ways listed above. It is not an error if @autowire="true" and a target
917 is also defined through some other means, however in this case the autowire procedure
918 MUST NOT be used. [ASM50014]
- 919 • If a reference has a value specified for one or more target services in its @target attribute,
920 there MUST NOT be any child <binding/> elements declared for that reference.
921 [ASM50026]
- 922 • If a binding element has a value specified for a target service using its @uri attribute, the
923 binding element MUST NOT identify target services using binding specific attributes or
924 elements. [ASM50015]
- 925 • It is possible that a particular binding type MAY require that the address of a target service
926 uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to
927 identify the target service - instead, binding specific attributes and/or child elements must
928 be used. [ASM50016]
- 929 • If any <wire/> element with its @replace attribute set to "true" has a particular reference
930 specified in its @source attribute, the value of the @target attribute for that reference
931 MUST be ignored and MUST NOT be used to define target services for that reference.
932 [ASM50034]

933 5.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

- 935 • A reference with multiplicity 0..1 or 0..n MAY have no target service defined. [ASM50018]
- 936 • A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service
937 defined. [ASM50019]
- 938 • A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.
939 [ASM50020]

- 940 • A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.
941 [\[ASM50021\]](#)

942 Where it is detected that the rules for the number of target services for a reference have been
943 violated, either at deployment or at execution time, an SCA Runtime MUST generate an error no
944 later than when the reference is invoked by the component implementation. [\[ASM50022\]](#)

945 Some reference multiplicity errors can be detected at deployment time. In these cases, an error
946 SHOULD be generated by the SCA runtime at deployment time. [\[ASM50023\]](#) For example, where
947 a composite is used as a component implementation, wires and target services cannot be added to
948 the composite after deployment. As a result, for components which are part of the composite,
949 both missing wires and wires with a non-existent target can be detected at deployment time
950 through a scan of the contents of the composite.

951 Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA
952 runtime MUST generate an error no later than when the reference is invoked by the component
953 implementation. [\[ASM50024\]](#) Examples include cases of components deployed to the SCA
954 Domain. At the Domain level, the target of a wire, or even the wire itself, may form part of a
955 separate deployed contribution and as a result these may be deployed after the original
956 component is deployed. For the cases where it is valid for the reference to have no target service
957 specified, the component implementation language specification needs to define the programming
958 model for interacting with an untargetted reference.

959 Where a component reference is promoted by a composite reference, the promotion MUST be
960 treated from a multiplicity perspective as providing 0 or more target services for the component
961 reference, depending upon the further configuration of the composite reference. These target
962 services are in addition to any target services identified on the component reference itself, subject
963 to the rules relating to multiplicity. [\[ASM50025\]](#)

964 5.4 Property

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

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

- 972 • As a value, supplied in the **value** attribute of the property element.
973 If the @value attribute of a component property element is declared, the type of the
974 property MUST be an XML Schema simple type and the @value attribute MUST contain a
975 single value of that type. [\[ASM50027\]](#)

976 For example,

977

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

- 978 • As a value, supplied as the content of the **value** element(s) children of the property
979 element.
980 If the value subelement of a component property is specified, the type of the property
981 MUST be an XML Schema simple type or an XML schema complex type. [\[ASM50028\]](#)

982 For example,

- 983 • property defined using a XML Schema simple type and which contains a single
984 value

985

```
<property name="pi">  
986                <value>3.14159265</value>  
987           </property>
```

- 988 • property defined using a XML Schema simple type and which contains multiple
989 values

```

990         <property name="currency">
991             <value>EURO</value>
992             <value>USDollar</value>
993         </property>
994     • property defined using a XML Schema complex type and which contains a single
995       value
996     <property name="complexFoo">
997         <value attr="bar">
998             <foo:a>TheValue</foo:a>
999             <foo:b>InterestingURI</foo:b>
1000         </value>
1001     </property>
1002     • property defined using a XML Schema complex type and which contains multiple
1003       values
1004     <property name="complexBar">
1005         <value anotherAttr="foo">
1006             <bar:a>AValue</bar:a>
1007             <bar:b>InterestingURI</bar:b>
1008         </value>
1009         <value attr="zing">
1010             <bar:a>BValue</bar:a>
1011             <bar:b>BoringURI</bar:b>
1012         </value>
1013     </property>
1014     • As a value, supplied as the content of the property element.
1015       If a component property value is declared using a child element of the <property/>
1016       element, the type of the property MUST be an XML Schema global element and the
1017       declared child element MUST be an instance of that global element. [ASM50029]
1018     For example,
1019     • property defined using a XML Schema global element declaration and which
1020       contains a single value
1021     <property name="foo">
1022         <foo:SomeGED ...>...</foo:SomeGED>
1023     </property>
1024     • property defined using a XML Schema global element declaration and which
1025       contains multiple values
1026     <property name="bar">
1027         <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1028         <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1029     </property>
1030     • By referencing a Property value of the composite which contains the component. The
1031       reference is made using the source attribute of the property element.
1032     The form of the value of the source attribute follows the form of an XPath expression.
1033

```

1034 This form allows a specific property of the composite to be addressed by name. Where the
1035 composite property is of a complex type, the XPath expression can be extended to refer to
1036 a sub-part of the complex property value.
1037

1038 So, for example, `source="$currency"` is used to reference a property of the composite
1039 called "currency", while `source="$currency/a"` references the sub-part "a" of the
1040 complex composite property with the name "currency".

- 1041 • By specifying a dereferencable URI to a file containing the property value through the **file**
1042 attribute. The contents of the referenced file are used as the value of the property.

1043

1044 If more than one property value specification is present, the source attribute takes precedence, then
1045 the file attribute.

1046 For a property defined using a XML Schema simple type and for which a single value is desired, can
1047 be set either using the @value attribute or the <value> child element. The two forms in such a case
1048 are equivalent.

1049 When a property has multiple values set, they MUST all be contained within the same property
1050 element. A <component/> element MUST NOT contain two <property/> subelements with the same
1051 value of the @name attribute. [ASM50030]

1052 Optionally, the type of the property can be specified in **one** of two ways:

- 1053 • by the qualified name of a type defined in an XML schema, using the **type** attribute
- 1054 • by the qualified name of a global element in an XML schema, using the **element** attribute

1055 The property type specified must be compatible with the type of the property declared in the
1056 component type of the implementation. If no type is declared in the component property, the type of
1057 the property declared by the implementation is used.

1058

1059 The following snippet shows the component schema with the schema for a property child element:

1060

```
1061 <?xml version="1.0" encoding="UTF-8"?>
1062 <!-- Component Property schema snippet -->
1063 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1064   ...
1065   <component ... >*
1066     <implementation ... />?
1067     <service ... />*
1068     <reference ... />*
1069     <property name="xs:NCName"
1070       (type="xs:QName" | element="xs:QName"?
1071       mustSupply="xs:boolean"? many="xs:boolean"?
1072       source="xs:string"? file="xs:anyURI"?
1073       requires="list of xs:QName"?
1074       policySets="list of xs:QName"?
1075       value="xs:string"?>*
1076       [<value>+ | xs:any+ ]?
1077     </property>
1078   </component>
1079   ...
```

1080 </composite>

1081
1082

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

1083 ▪ **name : NCName (1..1)** – the name of the property. The name attribute of a component
1084 property MUST match the name of a property element in the component type of the
1085 component implementation. [ASM50031]

1086 ▪ zero or one of **(0..1)**:

1087 ○ **type : QName** – the type of the property defined as the qualified name of an XML
1088 schema type

1089 ○ **element : QName** – the type of the property defined as the qualified name of an
1090 XML schema global element – the type is the type of the global element

1091 A single property element MUST NOT contain both an @type attribute and an @element
1092 attribute. [ASM50035]

Formatted: Indent: Before:
0.75"

1093 ▪ **source : string (0..1)** – an XPath expression pointing to a property of the containing
1094 composite from which the value of this component property is obtained.

Formatted: Font color: Red

1095 ▪ **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property

1096 ▪ **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or
1097 multi-valued (true). Overrides the many specified for this property on the implementation.
1098 The value can only be equal or further restrict, i.e. if the implementation specifies many
1099 true, then the component can say false. In the case of a multi-valued property, it is
1100 presented to the implementation as a Collection of property values. If many is not
1101 specified, it takes the value defined by the component type of the implementation used by
1102 the component.

1103 ▪ **value : string (0..1)** - the value of the property if the property is defined using a simple
1104 type.

1105 ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)
1106 [\[10\]](#) for a description of this attribute.

1107 ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)
1108 [\[10\]](#) for a description of this attribute.

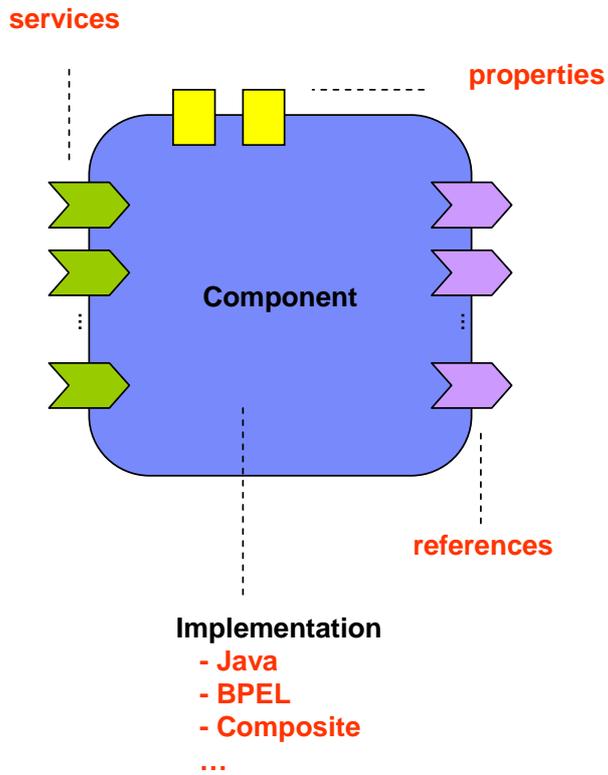
1109 The **component property** element has the following **child element**:

1110 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a
1111 property that is defined using a XML Schema type. If a property is single-valued, the <value/>
1112 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST
1113 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

1114 5.5 Example Component

1115

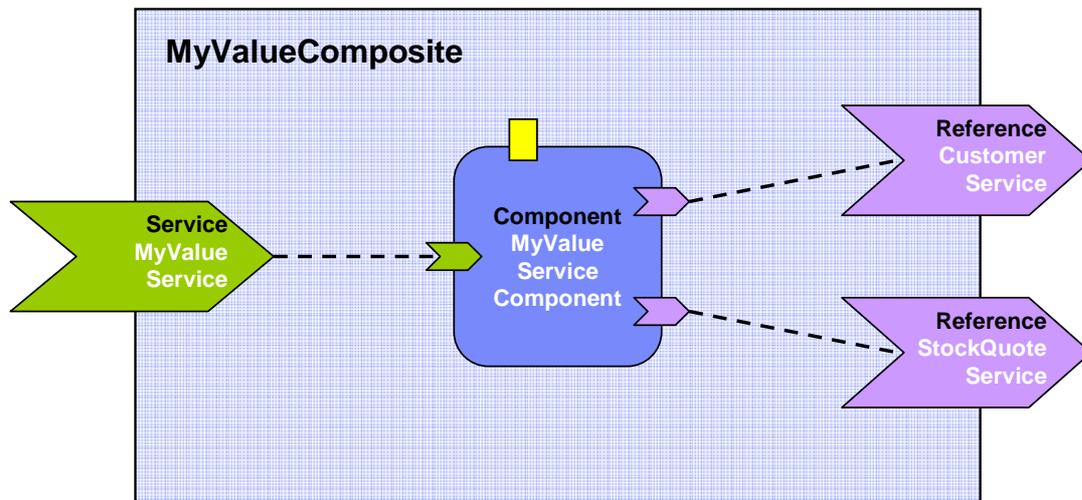
1116 The following figure shows the **component symbol** that is used to represent a component in an
1117 assembly diagram.



1118
1119
1120
1121
1122

Figure 5: Component symbol

The following figure shows the assembly diagram for the MyValueComposite containing the MyValueServiceComponent.



1123
1124
1125

Figure 6: Assembly diagram for MyValueComposite

1126

1127 The following snippet shows the MyValueComposite.composite file for the MyValueComposite
1128 containing the component element for the MyValueServiceComponent. A value is set for the
1129 property named currency, and the customerService and stockQuoteService references are
1130 promoted:

1131

```
1132 <?xml version="1.0" encoding="ASCII"?>
1133 <!-- MyValueComposite_1 example -->
1134 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1135           targetNamespace="http://foo.com"
1136           name="MyValueComposite" >
1137
1138     <service name="MyValueService" promote="MyValueServiceComponent" />
1139
1140     <component name="MyValueServiceComponent">
1141       <implementation.java
1142 class="services.myvalue.MyValueServiceImpl" />
1143       <property name="currency">EURO</property>
1144       <reference name="customerService" />
1145       <reference name="stockQuoteService" />
1146     </component>
1147
1148     <reference name="CustomerService"
1149           promote="MyValueServiceComponent/customerService" />
1150
1151     <reference name="StockQuoteService"
1152           promote="MyValueServiceComponent/stockQuoteService" />
1153
1154 </composite>
```

1155

1156 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of
1157 clarity – the references are defined by the MyValueServiceImpl implementation and there is no
1158 need to redeclare them on the component unless the intention is to wire them or to override some
1159 aspect of them.

1160 The following snippet gives an example of the layout of a composite file if both the currency
1161 property and the customerService reference of the MyValueServiceComponent are declared to be
1162 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1163 <?xml version="1.0" encoding="ASCII"?>
1164 <!-- MyValueComposite_2 example -->
1165 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1166           targetNamespace="http://foo.com"
1167           name="MyValueComposite" >
1168
1169     <service name="MyValueService" promote="MyValueServiceComponent" />
```

1170

```
1171     <component name="MyValueServiceComponent" >
1172         <implementation.java
1173 class="services.myvalue.MyValueServiceImpl" />
1174         <property name="currency">EURO</property>
1175         <property name="currency">Yen</property>
1176         <property name="currency">USDollar</property>
1177         <reference name="customerService"
1178             target="InternalCustomer/customerService" />
1179         <reference name="StockQuoteService" />
1180     </component>
1181
1182     ...
1183
1184     <reference name="CustomerService"
1185         promote="MyValueServiceComponent/customerService" />
1186
1187     <reference name="StockQuoteService"
1188         promote="MyValueServiceComponent/StockQuoteService" />
1189
1190 </composite>
1191
1192 ....this assumes that the composite has another component called InternalCustomer (not shown)
1193 which has a service to which the customerService reference of the MyValueServiceComponent is
1194 wired as well as being promoted externally through the composite reference CustomerService.
```

1195 6 Composite

1196 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of
1197 composition within an SCA Domain. An **SCA composite** contains a set of components, services,
1198 references and the wires that interconnect them, plus a set of properties which can be used to
1199 configure components.

1200 Composites can be used as **component implementations** in higher-level composites – in other
1201 words the higher-level composites can have components that are implemented by composites.
1202 For more detail on the use of composites as component implementations see the section [Using](#)
1203 [Composites as Component Implementations](#).

1204 The content of a composite can be used within another composite through **inclusion**. When a
1205 composite is included by another composite, all of its contents are made available for use within
1206 the including composite – the contents are fully visible and can be referenced by other elements
1207 within the including composite. For more detail on the inclusion of one composite into another see
1208 the section [Using Composites through Inclusion](#).

1209 A composite can be used as a unit of deployment. When used in this way, composites contribute
1210 elements to an SCA domain. A composite can be deployed to the SCA domain either by inclusion,
1211 or a composite can be deployed to the domain as an implementation. For more detail on the
1212 deployment of composites, see the section dealing with the [SCA Domain](#).

1213

1214 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite**
1215 element. The following snippet shows the schema for the composite element.

1216

```
1217 <?xml version="1.0" encoding="ASCII"?>  
1218 <!-- Composite schema snippet -->  
1219 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
1220     targetNamespace="xs:anyURI"  
1221     name="xs:NCName" local="xs:boolean"?  
1222     autowire="xs:boolean"? constrainingType="QName"?  
1223     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1224  
1225     <include ... />*  
1226  
1227     <service ... />*  
1228     <reference ... />*  
1229     <property ... />*  
1230  
1231     <component ... />*  
1232  
1233     <wire ... />*  
1234  
1235 </composite>
```

1236

1237 The **composite** element has the following **attributes**:

- 1238 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is
1239 an XML QName, in the namespace identified by the targetNamespace attribute. A
1240 composite name must be unique within the namespace of the composite. [\[ASM60001\]](#)
- 1241 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the
1242 composite is declared
- 1243 • **local : boolean (0..1)** – whether all the components within the composite all run in the
1244 same operating system process. @local="true" for a composite means that all the

- 1245 components within the composite MUST run in the same operating system process.
 1246 [\[ASM60002\]](#) local="false", which is the default, means that different components within
 1247 the composite can run in different operating system processes and they can even run on
 1248 different nodes on a network.
- 1249 • **autowire : boolean (0..1)** – whether contained component references should be
 1250 autowired, as described in [the Autowire section](#). Default is false.
 - 1251 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,
 1252 the set of services, references and properties of the composite, plus related intents, is
 1253 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)
 1254 for more details.
 - 1255 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)
 1256 [specification \[10\]](#) for a description of this attribute.
 - 1257 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
 1258 [\[10\]](#) for a description of this attribute.

1259

1260 The **composite** element has the following **child elements**:

- 1261 • **service : CompositeService (0..n)** – see composite service section.
- 1262 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1263 • **property : CompositeProperty (0..n)** – see composite property section.
- 1264 • **component : Component (0..n)** – see component section.
- 1265 • **wire : Wire (0..n)** – see composite wire section.
- 1266 • **include : Include (0..n)** – see composite include section

1267

1268 Components contain configured implementations which hold the business logic of the composite.
 1269 The components offer services and require references to other services. **Composite services**
 1270 define the public services provided by the composite, which can be accessed from outside the
 1271 composite. **Composite references** represent dependencies which the composite has on services
 1272 provided elsewhere, outside the composite. Wires describe the connections between component
 1273 services and component references within the composite. Included composites contribute the
 1274 elements they contain to the using composite.

1275 Composite services involve the **promotion** of one service of one of the components within the
 1276 composite, which means that the composite service is actually provided by one of the components
 1277 within the composite. Composite references involve the **promotion** of one or more references of
 1278 one or more components. Multiple component references can be promoted to the same composite
 1279 reference, as long as all the component references are compatible with one another. Where
 1280 multiple component references are promoted to the same composite reference, then they all share
 1281 the same configuration, including the same target service(s).

1282 Composite services and composite references can use the configuration of their promoted services
 1283 and references respectively (such as Bindings and Policy Sets). Alternatively composite services
 1284 and composite references can override some or all of the configuration of the promoted services
 1285 and references, through the configuration of bindings and other aspects of the composite service
 1286 or reference.

1287 Component services and component references can be promoted to composite services and
 1288 references and also be wired internally within the composite at the same time. For a reference,
 1289 this only makes sense if the reference supports a multiplicity greater than 1.

1290

1291 6.1 Service

1292 The **services of a composite** are defined by promoting services defined by components
1293 contained in the composite. A component service is promoted by means of a composite **service**
1294 **element**.

1295 A composite service is represented by a **service element** which is a child of the composite
1296 element. There can be **zero or more** service elements in a composite. The following snippet
1297 shows the pseudo-schema for a service child element:

```
1298  
1299 <?xml version="1.0" encoding="ASCII"?>  
1300 <!-- Composite Service schema snippet -->  
1301 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1302 ...  
1303 <service name="xs:NCName" promote="xs:anyURI"  
1304 <requires="list of xs:QName"? policySets="list of xs:QName"?>*</interface ... />?  
1305 <operation name="xs:NCName" requires="list of xs:QName"?  
1306 <policySets="list of xs:QName"?/>*</binding ... />+  
1307 <callback>?  
1308 <binding ... />+  
1309 </callback>  
1310 </service>  
1311 ...  
1312 </composite>  
1313  
1314  
1315
```

1316 The **composite service** element has the following **attributes**:

- 1317 • **name : NCName (1..1)** – the name of the service. The name of a composite <service/>
1318 element MUST be unique across all the composite services in the composite. [ASM60003]
1319 The name of the composite service can be different from the name of the promoted
1320 component service.
- 1321 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form
1322 <component-name>/<service-name>. The service name is optional if the target
1323 component only has one service. The same component service can be promoted by more
1324 than one composite service. A composite <service/> element's promote attribute MUST
1325 identify one of the component services within that composite. [ASM60004]
- 1326 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)
1327 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or
1328 further qualify the required intents defined by the promoted component service.
- 1329 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
1330 [\[10\]](#) for a description of this attribute.

1331
1332 The **composite service** element has the following **child elements**, whatever is not specified is
1333 defaulted from the promoted component service.

- 1334 • **interface : Interface (0..1)** - If a composite service interface is specified it must be the
1335 same or a compatible subset of the interface provided by the promoted component
1336 service, i.e. provide a subset of the operations defined by the component service.
1337 [ASM60005] The interface is described by **zero or one interface element** which is a
1338 child element of the service element. For details on the interface element see [the Interface](#)
1339 [section](#).

Deleted: 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, i.e. provide a subset of the operations defined by the component service.

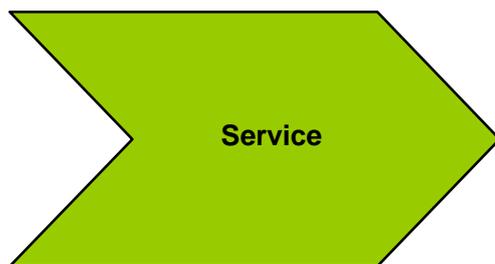
- 1340
- 1341
- 1342
- **operation: Operation (0..n)** - Zero or more operation elements. These elements are used to describe characteristics of individual operations within the interface. For a detailed description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
 - **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the promoted component service from the composite service perspective. The bindings defined on the component service are still in effect for local wires within the composite that target the component service. A service element has zero or more **binding elements** as children. Details of the binding element are described in the [Bindings section](#). For more details on wiring see [the Wiring section](#).
 - **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
- 1349
- 1350
- 1351
- 1352
- 1353
- 1354

1355 6.1.1 Service Examples

1356

1357 The following figure shows the service symbol that used to represent a service in an assembly

1358 diagram:



1359

1360 *Figure 7: Service symbol*

1361

1362 The following figure shows the assembly diagram for the MyValueComposite containing the service

1363 MyValueService.

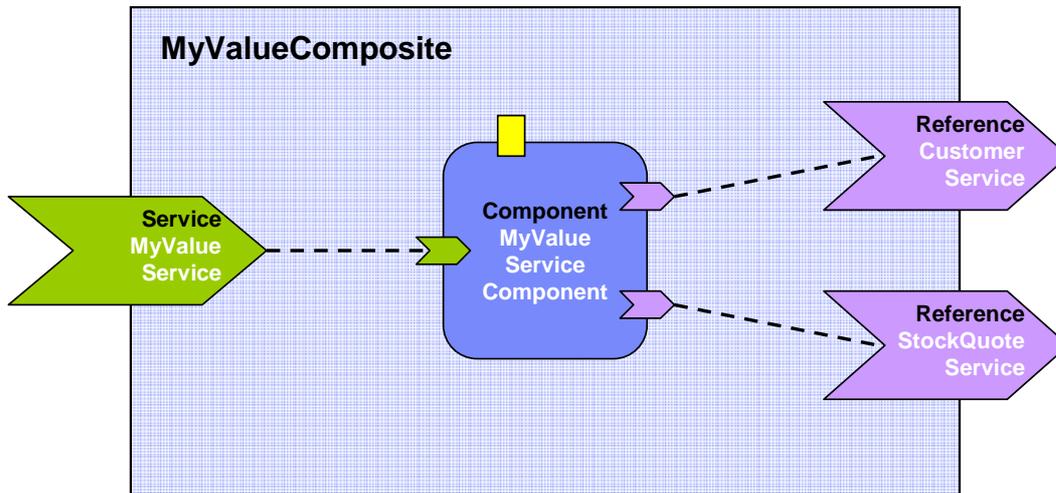


Figure 8: MyValueComposite showing Service

1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394

The following snippet shows the MyValueComposite.composite file for the MyValueComposite containing the service element for the MyValueService, which is a promote of the service offered by the MyValueServiceComponent. The name of the promoted service is omitted since MyValueServiceComponent offers only one service. The composite service MyValueService is bound using a Web service binding.

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_4 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
targetNamespace="http://foo.com"
name="MyValueComposite" >
...
<service name="MyValueService" promote="MyValueServiceComponent">
<interface.java interface="services.myvalue.MyValueService"/>
<binding.ws port="http://www.myvalue.org/MyValueService#
wsdl.endpoint(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>
```

1395
1396 ...
1397
1398 </composite>
1399

1400 6.2 Reference

1401 The **references of a composite** are defined by **promoting** references defined by components
1402 contained in the composite. Each promoted reference indicates that the component reference
1403 needs to be resolved by services outside the composite. A component reference is promoted using
1404 a composite **reference element**.

1405 A composite reference is represented by a **reference element** which is a child of a composite
1406 element. There can be **zero or more** reference elements in a composite. The following snippet
1407 shows the composite schema with the schema for a **reference** element.

1408
1409 <?xml version="1.0" encoding="ASCII"?>
1410 <!-- Composite Reference schema snippet -->
1411 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1412 ...
1413 <reference name="xs:NCName" target="list of xs:anyURI"?
1414 promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
1415 multiplicity="0..1 or 1..1 or 0..n or 1..n"?
1416 requires="list of xs:QName"? policySets="list of xs:QName"?>*</reference>
1417 <interface ... />?
1418 <operation name="xs:NCName" requires="list of xs:QName"?
1419 policySets="list of xs:QName"?/>*</operation>
1420 <binding ... />*</binding>
1421 <callback?>
1422 <binding ... />+</callback>
1423 </callback>
1424 </reference>
1425 ...
1426 </composite>
1427
1428

1429 The **composite reference** element has the following **attributes**:

- 1430 • **name : NCName (1..1)** – the name of the reference. The name of a composite
1431 <reference/> element MUST be unique across all the composite references in the
1432 composite. [ASM60006] The name of the composite reference can be different then the
1433 name of the promoted component reference.
- 1434 • **promote : anyURI (1..n)** – identifies one or more promoted component references. The
1435 value is a list of values of the form <component-name>/<reference-name> separated by
1436 spaces. The specification of the reference name is optional if the component has only one
1437 reference. Each of the URIs declared by a composite reference's @promote attribute MUST
1438 identify a component reference within the composite. [ASM60007]

1439 The same component reference can be promoted more than once, using different
1440 composite references, but only if the multiplicity defined on the component reference is
1441 0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1442 Where a composite reference promotes two or more component references:

- 1443 • the interfaces of the component references promoted by a composite reference
1444 MUST be the same, or if the composite reference itself declares an interface then
1445 all the component reference interfaces must be compatible with the composite

1446 reference interface. Compatible means that the component reference interface is
1447 the same or is a strict subset of the composite reference interface. [ASM60008]

1448 • the intents declared on a composite reference and on the component references
1449 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents
1450 which apply to the composite reference in this case are the union of the required
1451 intents specified for each of the promoted component references plus any intents
1452 declared on the composite reference itself. If any intents in the set which apply to
1453 a composite reference are mutually exclusive then the SCA runtime MUST raise an
1454 error. [ASM60010]

1455 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)
1456 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or
1457 further qualify the required intents defined for the promoted component reference.

1458 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
1459 [\[10\]](#) for a description of this attribute.

1460 • **multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can
1461 connect the reference to target services. The multiplicity can have the following values

- 1462 ○ 0..1 – zero or one wire can have the reference as a source
- 1463 ○ 1..1 – one wire can have the reference as a source
- 1464 ○ 0..n - zero or more wires can have the reference as a source
- 1465 ○ 1..n – one or more wires can have the reference as a source

1466 The value specified for the **multiplicity** attribute of a composite reference MUST be
1467 compatible with the multiplicity specified on each of the promoted component references,
1468 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used
1469 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be
1470 used where the promoted component reference has multiplicity 0..n or 1..n and
1471 multiplicity 1..n can be used where the promoted component reference has multiplicity
1472 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to
1473 promote a component reference of multiplicity 0..1 or 1..1 respectively. [ASM60011]

1474 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on
1475 multiplicity setting. Each value wires the reference to a service in a composite that uses
1476 the composite containing the reference as an implementation for one of its components. For
1477 more details on wiring see [the section on Wires](#).

1478 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that
1479 the implementation wires this reference dynamically. If set to "true" it indicates that the
1480 target of the reference is set at runtime by the implementation code (eg by the code
1481 obtaining an endpoint reference by some means and setting this as the target of the
1482 reference through the use of programming interfaces defined by the relevant Client and
1483 Implementation specification). If "true" is set, then the reference should not be wired
1484 statically within a using composite, but left unwired.

1485
1486 The **composite reference** element has the following **child elements**, whatever is not specified is
1487 defaulted from the promoted component reference(s).

1488 • **interface : Interface (0..1) - zero or one interface element** which declares an
1489 interface for the composite reference. If a composite reference has an **interface** specified,
1490 it MUST provide an interface which is the same or which is a compatible superset of the
1491 interface(s) declared by the promoted component reference(s), i.e. provide a superset of
1492 the operations in the interface defined by the component for the reference. [ASM60012] If
1493 no interface is declared on a composite reference, the interface from one of its promoted
1494 component references is used, which MUST be the same as or a compatible superset of
1495 the interface(s) declared by the promoted component reference(s).
1496 [ASM60013] For details on the interface element see [the Interface section](#).

Deleted: The value specified for the **multiplicity** attribute of a composite reference MUST be compatible with the multiplicity specified on each of the promoted component references, i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be used where the promoted component reference has multiplicity 0..n or 1..n and multiplicity 1..n can be used where the promoted component reference has multiplicity 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to promote a component reference of multiplicity 0..1 or 1..1 respectively.

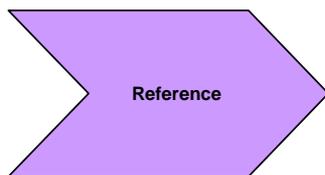
Deleted: If a composite reference has an **interface** specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s), i.e. provide a superset of the operations in the interface defined by the component for the reference.

- 1497
- 1498
- 1499
- **operation: Operation (0..n)** - Zero or more operation elements. These elements are used to describe characteristics of individual operations within the interface. For a detailed description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
 - **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as children. If one or more **bindings** are specified they **override** any and all of the bindings defined for the promoted component reference from the composite reference perspective. The bindings defined on the component reference are still in effect for local wires within the composite that have the component reference as their source. Details of the binding element are described in the [Bindings section](#). For more details on wiring see [the section on Wires](#).
- 1500
- 1501
- 1502
- 1503
- 1504
- 1505
- 1506
- 1507
- 1508
- 1509
- A reference identifies zero or more target services which satisfy the reference. This can be done in a number of ways, which are fully described in section "5.3.1 Specifying the Target Service(s) for a Reference".
- **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
- 1510
- 1511
- 1512
- 1513
- 1514
- 1515

1516 6.2.1 Example Reference

1517

1518 The following figure shows the reference symbol that is used to represent a reference in an
1519 assembly diagram.



1520

1521 *Figure 9: Reference symbol*

1522

1523 The following figure shows the assembly diagram for the MyValueComposite containing the
1524 reference CustomerService and the reference StockQuoteService.

1525

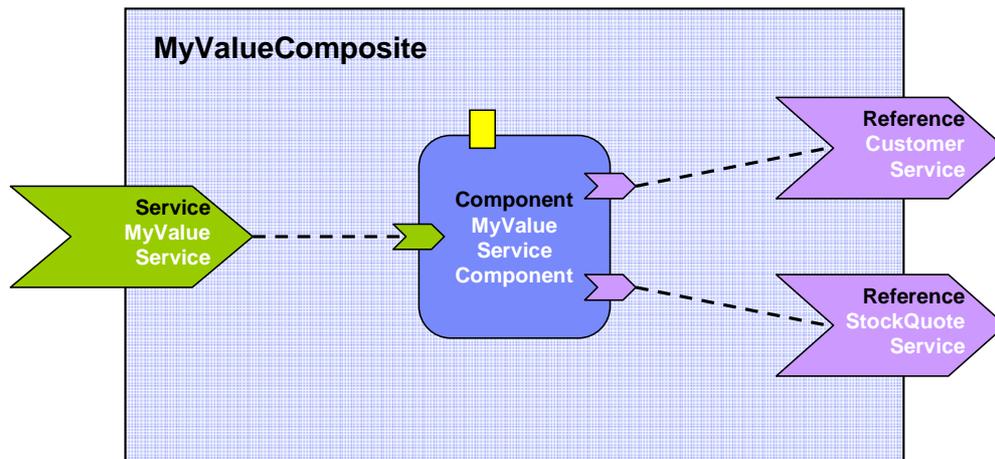


Figure 10: MyValueComposite showing References

1526

1527

1528

1529 The following snippet shows the MyValueComposite.composite file for the MyValueComposite
 1530 containing the reference elements for the CustomerService and the StockQuoteService. The
 1531 reference CustomerService is bound using the SCA binding. The reference StockQuoteService is
 1532 bound using the Web service binding. The endpoint addresses of the bindings can be specified, for
 1533 example using the binding *uri* attribute (for details see the [Bindings](#) section), or overridden in an
 1534 enclosing composite. Although in this case the reference StockQuoteService is bound to a Web
 1535 service, its interface is defined by a Java interface, which was created from the WSDL portType of
 1536 the target web service.

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

1555

1556

1557

1558

```

<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_3 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
targetNamespace="http://foo.com"
name="MyValueComposite" >
...
<component name="MyValueServiceComponent">
<implementation.java
class="services.myvalue.MyValueServiceImpl"/>
<property name="currency">EURO</property>
<reference name="customerService"/>
<reference name="StockQuoteService"/>
</component>
<reference name="CustomerService"
promote="MyValueServiceComponent/customerService">
<interface.java interface="services.customer.CustomerService"/>
<!-- The following forces the binding to be binding.sca whatever
is -->

```

```

1559         <!-- specified by the component reference or by the underlying
1560 -->
1561         <!-- implementation
1562 -->
1563         <binding.sca/>
1564     </reference>
1565
1566     <reference name="StockQuoteService"
1567         promote="MyValueServiceComponent/StockQuoteService">
1568         <interface.java
1569 interface="services.stockquote.StockQuoteService"/>
1570         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1571 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1572     </reference>
1573
1574     ...
1575
1576
1577 </composite>
1578

```

6.3 Property

1579 **Properties** allow for the configuration of an implementation with externally set data values. A
1580 composite can declare zero or more properties. Each property has a type, which may be either
1581 simple or complex. An implementation can also define a default value for a property. Properties
1582 can be configured with values in the components that use the implementation.

1584 The declaration of a property in a composite follows the form described in the following schema
1585 snippet:

```

1586
1587 <?xml version="1.0" encoding="ASCII"?>
1588 <!-- Composite Property schema snippet -->
1589 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1590     ...
1591     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1592         requires="list of xs:QName"?
1593         policySets="list of xs:QName"?
1594         many="xs:boolean"? mustSupply="xs:boolean"?>*
1595         default-property-value?
1596     </property>
1597     ...
1598 </composite>
1599

```

1600 The **composite property** element has the following **attributes**:

- 1601 ▪ **name : NCName (1..1)** - the name of the property. **The name attribute of a composite**
1602 **property MUST be unique amongst the properties of the same composite.**^[ASM60014]

Deleted: The name attribute of a composite property MUST be unique amongst the properties of the same composite.

- 1603 ▪ one of **(1..1)**:
 - 1604 ○ **type : QName** – the type of the property - the qualified name of an XML schema type
 - 1606 ○ **element : QName** – the type of the property defined as the qualified name of an XML schema global element – the type is the type of the global element
- 1608 A single property element MUST NOT contain both an @type attribute and an @element attribute. [ASM60035] ← - - - -
- 1610 ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). The default is **false**. In the case of a multi-valued property, it is presented to the implementation as a collection of property values.
- 1613 ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the component that uses the composite – when mustSupply="true" the component has to supply a value since the composite has no default value for the property. A default-property-value is only worth declaring when mustSupply="false" (the default setting for the mustSupply attribute), since the implication of a default value is that it is used only when a value is not supplied by the using component.
- 1619 ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
- 1621 ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.

Formatted: Indent: Before: 0.75"

1624 The property element may contain an optional **default-property-value**, which provides default value for the property. The form of the default property value is as described [in the section on Component Property](#).

1628 Implementation types other than **composite** can declare properties in an implementation-dependent form (eg annotations within a Java class), or through a property declaration of exactly the form described above in a componentType file.

1631 Property values can be configured when an implementation is used by a component. The form of the property configuration is shown in [the section on Components](#).

1633 6.3.1 Property Examples

1634 For the following example of Property declaration and value setting, the following complex type is used as an example:

```

1637     <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1638                targetNamespace="http://foo.com/"
1639                xmlns:tns="http://foo.com/">
1640       <!-- ComplexProperty schema -->
1641       <xsd:element name="fooElement" type="MyComplexType"/>
1642       <xsd:complexType name="MyComplexType">
1643         <xsd:sequence>
1644           <xsd:element name="a" type="xsd:string"/>
1645           <xsd:element name="b" type="anyURI"/>
1646         </xsd:sequence>
1647         <attribute name="attr" type="xsd:string" use="optional"/>
1648       </xsd:complexType>
1649     </xsd:schema>
1650
  
```

1651 The following composite demonstrates the declaration of a property of a complex type, with a
1652 default value, plus it demonstrates the setting of a property value of a complex type within a
1653 component:

```
1654 <?xml version="1.0" encoding="ASCII"?>
1655 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1656           xmlns:foo="http://foo.com"
1657           targetNamespace="http://foo.com"
1658           name="AccountServices">
1659 <!-- AccountServices Example1 -->
1660
1661     ...
1662
1663     <property name="complexFoo" type="foo:MyComplexType">
1664       <value>
1665         <foo:a>AValue</foo:a>
1666         <foo:b>InterestingURI</foo:b>
1667       </value>
1668     </property>
1669
1670     <component name="AccountServiceComponent">
1671       <implementation.java class="foo.AccountServiceImpl"/>
1672       <property name="complexBar" source="$complexFoo"/>
1673       <reference name="accountDataService"
1674               target="AccountDataServiceComponent"/>
1675       <reference name="stockQuoteService" target="StockQuoteService"/>
1676     </component>
1677
1678     ...
1679
1680 </composite>
1681
```

1682 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the
1683 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the
1684 composite and it references the example XSD, where MyComplexType is defined. The declaration
1685 of complexFoo contains a default value. This is declared as the content of the property element.
1686 In this example, the default value consists of the element **value** which is required to be of type
1687 foo:MyComplexType and its two child elements <foo:a> and <foo:b>, following the definition of
1688 MyComplexType.

1689 In the component **AccountServiceComponent**, the component sets the value of the property
1690 **complexBar**, declared by the implementation configured by the component. In this case, the
1691 type of complexBar is foo:MyComplexType. The example shows that the value of the complexBar
1692 property is set from the value of the complexFoo property – the **source** attribute of the property
1693 element for complexBar declares that the value of the property is set from the value of a property
1694 of the containing composite. The value of the source attribute is **\$complexFoo**, where
1695 complexFoo is the name of a property of the composite. This value implies that the whole of the
1696 value of the source property is used to set the value of the component property.

1697 The following example illustrates the setting of the value of a property of a simple type (a string)
1698 from **part** of the value of a property of the containing composite which has a complex type:

```
1699 <?xml version="1.0" encoding="ASCII"?>
1700 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1701           xmlns:foo="http://foo.com"
1702           targetNamespace="http://foo.com"
1703           name="AccountServices">
1704 <!-- AccountServices Example2 -->
1705
1706     ...
```

```

1707
1708     <property name="complexFoo" type="foo:MyComplexType">
1709         <value>
1710             <foo:a>AValue</foo:a>
1711             <foo:b>InterestingURI</foo:b>
1712         </value>
1713     </property>
1714
1715     <component name="AccountServiceComponent">
1716         <implementation.java class="foo.AccountServiceImpl"/>
1717         <property name="currency" source="$complexFoo/a"/>
1718         <reference name="accountDataService"
1719             target="AccountDataServiceComponent"/>
1720         <reference name="stockQuoteService" target="StockQuoteService"/>
1721     </component>
1722
1723     ...
1724
1725 </composite>
1726

```

1727 In this example, the component **AccountServiceComponent** sets the value of a property called
1728 **currency**, which is of type string. The value is set from a property of the composite
1729 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression
1730 that selects the property name **complexFoo** and then selects the value of the **a** subelement of
1731 the value of complexFoo. The "a" subelement is a string, matching the type of the currency
1732 property.

1733 Further examples of declaring properties and setting property values in a component follow:

1734 Declaration of a property with a simple type and a default value:

```

1735 <property name="SimpleTypeProperty" type="xsd:string">
1736 MyValue
1737 </property>

```

1738

1739 Declaration of a property with a complex type and a default value:

```

1740 <property name="complexFoo" type="foo:MyComplexType">
1741     <value>
1742         <foo:a>AValue</foo:a>
1743         <foo:b>InterestingURI</foo:b>
1744     </value>
1745 </property>
1746

```

1747 Declaration of a property with a global element type:

```

1748 <property name="elementFoo" element="foo:fooElement">
1749     <foo:fooElement>
1750         <foo:a>AValue</foo:a>
1751         <foo:b>InterestingURI</foo:b>
1752     </foo:fooElement>
1753 </property>
1754

```

1755 6.4 Wire

1756 **SCA wires** within a composite connect **source component references** to **target component**
1757 **services**.

1758 One way of defining a wire is by **configuring a reference of a component using its target**
1759 **attribute**. The reference element is configured with the wire-target-URI of the service(s) that
1760 resolve the reference. Multiple target services are valid when the reference has a multiplicity of
1761 0..n or 1..n.

1762 An alternative way of defining a Wire is by means of a **wire element** which is a child of the
1763 composite element. There can be **zero or more** wire elements in a composite. This alternative
1764 method for defining wires is useful in circumstances where separation of the wiring from the
1765 elements the wires connect helps simplify development or operational activities. An example is
1766 where the components used to build a domain are relatively static but where new or changed
1767 applications are created regularly from those components, through the creation of new assemblies
1768 with different wiring. Deploying the wiring separately from the components allows the wiring to
1769 be created or modified with minimum effort.

1770 Note that a Wire specified via a wire element is equivalent to a wire specified via the target
1771 attribute of a reference. The rule which forbids mixing of wires specified with the target attribute
1772 with the specification of endpoints in binding subelements of the reference also applies to wires
1773 specified via separate wire elements.

1774 The following snippet shows the composite schema with the schema for the reference elements of
1775 components and composite services and the wire child element:

```
1776  
1777 <?xml version="1.0" encoding="ASCII"?>  
1778 <!-- Wires schema snippet -->  
1779 <composite ...>  
1780   ...  
1781   <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/*>  
1782   ...  
1783 </composite>  
1784
```

1785 The **reference element of a component** and the **reference element of a service** has a list of
1786 one or more of the following **wire-target-URI** values for the target, with multiple values
1787 separated by a space:

- 1788 • `<component-name>/<service-name>`
 - 1789 ○ where the target is a service of a component. The specification of the service
1790 name is optional if the target component only has one service with a compatible
1791 interface

1792

1793 The **wire element** has the following attributes:

- 1794 • **source (1..1)** – names the source component reference. Valid URI schemes are:
 - 1795 ○ `<component-name>/<reference-name>`
 - 1796 ▪ where the source is a component reference. The specification of the
1797 reference name is optional if the source component only has one reference
- 1798 • **target (1..1)** – names the target component service. Valid URI schemes are
 - 1799 ○ `<component-name>/<service-name>`
 - 1800 ▪ where the target is a service of a component. The specification of the
1801 service name is optional if the target component only has one service with
1802 a compatible interface
- 1803 • **replace (0..1)** – a boolean value, with the default of "false". When a wire element has
1804 @replace="false", the wire is added to the set of wires which apply to the reference
1805 identified by the @source attribute. When a wire element has @replace="true", the wire
1806 is added to the set of wires which apply to the reference identified by the @source
1807 attribute - but any wires for that reference specified by means of the @target attribute of

1808 the reference are removed from the set of wires which apply to the reference.
1809
1810 In other words, if any <wire/> element with @replace="true" is used for a particular
1811 reference, the value of the @target attribute on the reference is ignored - and this permits
1812 existing wires on the reference to be overridden by separate configuration, if required,
1813 where the reference is on a component at the Domain level.

1814 For a composite used as a component implementation, wires can only link sources and targets
1815 that are contained in the same composite (irrespective of which file or files are used to describe
1816 the composite). Wiring to entities outside the composite is done through services and references
1817 of the composite with wiring defined by the next higher composite.

1818 A wire may only connect a source to a target if the target implements an interface that is
1819 compatible with the interface required by the source. The source and the target are compatible if:

- 1820 1. the source interface and the target interface of a wire MUST either both be remotable or
1821 else both be local [ASM60015]
- 1822 2. the operations on the target interface of a wire MUST be the same as or be a superset of
1823 the operations in the interface specified on the source [ASM60016]
- 1824 3. compatibility between the source interface and the target interface for a wire for the
1825 individual operations is defined as compatibility of the signature, that is operation name,
1826 input types, and output types MUST be the same. [ASM60017]
- 1827 4. the order of the input and output types for operations in the source interface and the
1828 target interface of a wire also MUST be the same. [ASM60018]
- 1829 5. the set of Faults and Exceptions expected by each operation in the source interface MUST
1830 be the same or be a superset of those specified by the target interface. [ASM60019]
- 1831 6. other specified attributes of the source interface and the target interface of a wire MUST
1832 match, including Scope and Callback interface [ASM60020]

1833 A Wire can connect between different interface languages (eg. Java interfaces and WSDL
1834 portTypes) in either direction, as long as the operations defined by the two interface types are
1835 equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and
1836 faults/exceptions map to each other.

1837 Service clients cannot (portably) ask questions at runtime about additional interfaces that are
1838 provided by the implementation of the service (e.g. the result of "instance of" in Java is non
1839 portable). It is valid for an SCA implementation to have proxies for all wires, so that, for example,
1840 a reference object passed to an implementation may only have the business interface of the
1841 reference and may not be an instance of the (Java) class which is used to implement the target
1842 service, even where the interface is local and the target service is running in the same process.

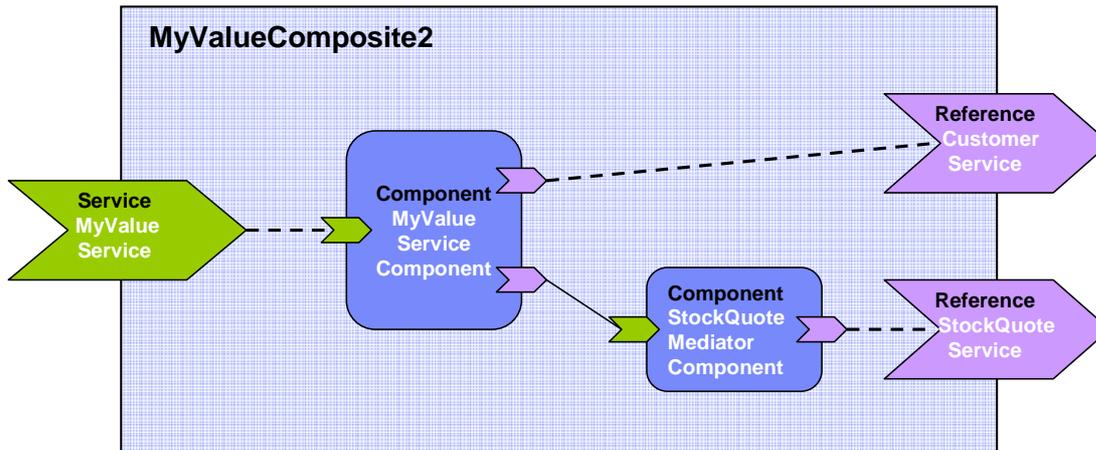
1843 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of
1844 an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA
1845 runtime SHOULD issue a warning. [ASM60021]

1846

1847 6.4.1 Wire Examples

1848

1849 The following figure shows the assembly diagram for the MyValueComposite2 containing wires
1850 between service, components and references.



1851
1852 *Figure 11: MyValueComposite2 showing Wires*

1853
1854 The following snippet shows the MyValueComposite2.composite file for the MyValueComposite2
1855 containing the configured component and service references. The service MyValueService is wired
1856 to the MyValueServiceComponent, using an explicit <wire/> element. The
1857 MyValueServiceComponent's customerService reference is wired to the composite's
1858 CustomerService reference. The MyValueServiceComponent's stockQuoteService reference is
1859 wired to the StockQuoteMediatorComponent, which in turn has its reference wired to the
1860 StockQuoteService reference of the composite.

1861
1862 `<?xml version="1.0" encoding="ASCII"?>`
1863 `<!-- MyValueComposite Wires examples -->`
1864 `<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"`
1865 `targetNamespace="http://foo.com"`
1866 `name="MyValueComposite2" >`
1867
1868 `<service name="MyValueService" promote="MyValueServiceComponent">`
1869 `<interface.java interface="services.myvalue.MyValueService"/>`
1870 `<binding.ws port="http://www.myvalue.org/MyValueService#`
1871 `wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>`
1872 `</service>`
1873
1874 `<component name="MyValueServiceComponent">`
1875 `<implementation.java`
1876 `class="services.myvalue.MyValueServiceImpl"/>`
1877 `<property name="currency">EURO</property>`
1878 `<service name="MyValueService"/>`
1879 `<reference name="customerService"/>`
1880 `<reference name="stockQuoteService"/>`
1881 `</component>`
1882

```

1883     <wire source="MyValueServiceComponent/stockQuoteService"
1884           target="StockQuoteMediatorComponent" />
1885
1886     <component name="StockQuoteMediatorComponent">
1887       <implementation.java class="services.myvalue.SQMediatorImpl" />
1888       <property name="currency">EURO</property>
1889       <reference name="stockQuoteService" />
1890     </component>
1891
1892     <reference name="CustomerService"
1893               promote="MyValueServiceComponent/customerService">
1894       <interface.java interface="services.customer.CustomerService" />
1895       <binding.sca/>
1896     </reference>
1897
1898     <reference name="StockQuoteService"
1899               promote="StockQuoteMediatorComponent">
1900       <interface.java
1901           interface="services.stockquote.StockQuoteService" />
1902       <binding.ws port="http://www.stockquote.org/StockQuoteService#
1903                 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)" />
1904     </reference>
1905
1906 </composite>
1907

```

1908 6.4.2 Autowire

1909 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.
1910 Autowire enables component references to be automatically wired to component services which
1911 will satisfy those references, without the need to create explicit wires between the references and
1912 the services. When the autowire feature is used, a component reference which is not promoted
1913 and which is not explicitly wired to a service within a composite is automatically wired to a target
1914 service within the same composite. Autowire works by searching within the composite for a
1915 service interface which matches the interface of the references.

1916 The autowire feature is not used by default. Autowire is enabled by the setting of an autowire
1917 attribute to "true". Autowire is disabled by setting of the autowire attribute to "false". The autowire
1918 attribute can be applied to any of the following elements within a composite:

- 1919 • reference
- 1920 • component
- 1921 • composite

1922 Where an element does not have an explicit setting for the autowire attribute, it inherits the
1923 setting from its parent element. Thus a reference element inherits the setting from its containing
1924 component. A component element inherits the setting from its containing composite. Where
1925 there is no setting on any level, autowire="false" is the default.

1926 As an example, if a composite element has autowire="true" set, this means that autowiring is
1927 enabled for all component references within that composite. In this example, autowiring can be

1928 turned off for specific components and specific references through setting `autowire="false"` on the
1929 components and references concerned.

1930 For each component reference for which `autowire` is enabled, the the SCA runtime MUST search
1931 within the composite for target services which are compatible with the reference. [ASM60022]
1932 "Compatible" here means:

- 1933 • the target service interface MUST be a compatible superset of the reference interface
1934 when using `autowire` to wire a reference (as defined in the section on Wires),[ASM60023]
- 1935 • the intents, and policies applied to the service MUST be compatible with those on the
1936 reference when using `autowire` to wire a reference – so that wiring the reference to the
1937 service will not cause an error due to policy mismatch [ASM60024] (see [the Policy](#)
1938 [Framework specification \[10\]](#) for details)

Deleted: the target service interface MUST be a compatible superset of the reference interface when using `autowire` to wire a reference (as defined in the section on Wires)

1939 If the search finds **1 or more** valid target service for a particular reference, the action taken
1940 depends on the multiplicity of the reference:

- 1941 • for an `autowire` reference with multiplicity `0..1` or `1..1`, the SCA runtime MUST wire the
1942 reference to one of the set of valid target services chosen from the set in a runtime-
1943 dependent fashion [ASM60025]
- 1944 • for an `autowire` reference with multiplicity `0..n` or `1..n`, the reference MUST be wired to all
1945 of the set of valid target services [ASM60026]

1946 If the search finds **no** valid target services for a particular reference, the action taken depends on
1947 the multiplicity of the reference:

- 1948 • for an `autowire` reference with multiplicity `0..1` or `0..n`, if the SCA runtime finds no valid
1949 target service, there is no problem – no services are wired and the SCA runtime MUST
1950 NOT raise an error [ASM60027]
- 1951 • for an `autowire` reference with multiplicity `1..1` or `1..n`, if the SCA runtime finds no valid
1952 target services an error MUST be raised by the SCA runtime since the reference is
1953 intended to be wired [ASM60028]

1954

1955 6.4.3 Autowire Examples

1956 This example demonstrates two versions of the same composite – the first version is done using
1957 explicit wires, with no autowiring used, the second version is done using `autowire`. In both cases
1958 the end result is the same – the same wires connect the references to the services.

1959 First, here is a diagram for the composite:

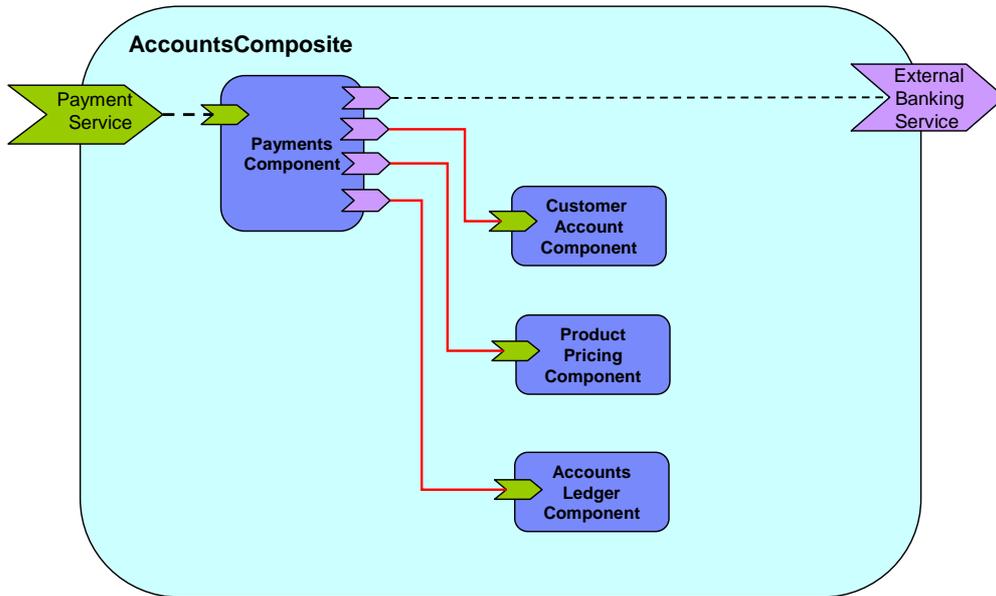


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1960
1961
1962
1963 <?xml version="1.0" encoding="UTF-8"?>
1964 <!-- Autowire Example - No autowire -->
1965 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1966   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1967   xmlns:foo="http://foo.com"
1968   targetNamespace="http://foo.com"
1969   name="AccountComposite">
1970
1971   <service name="PaymentService" promote="PaymentsComponent" />
1972
1973   <component name="PaymentsComponent">
1974     <implementation.java class="com.foo.accounts.Payments" />
1975     <service name="PaymentService" />
1976     <reference name="CustomerAccountService"
1977       target="CustomerAccountComponent" />
1978     <reference name="ProductPricingService"
1979       target="ProductPricingComponent" />
1980     <reference name="AccountsLedgerService"
1981       target="AccountsLedgerComponent" />
1982     <reference name="ExternalBankingService" />
1983   </component>
1984
1985   <component name="CustomerAccountComponent">
1986     <implementation.java class="com.foo.accounts.CustomerAccount" />
1987   </component>
1988
1989   <component name="ProductPricingComponent">
1990     <implementation.java class="com.foo.accounts.ProductPricing" />
1991   </component>
1992
1993   <component name="AccountsLedgerComponent">
  
```

```

1994         <implementation.composite name="foo:AccountsLedgerComposite"/>
1995     </component>
1996
1997     <reference name="ExternalBankingService"
1998         promote="PaymentsComponent/ExternalBankingService"/>
1999
2000 </composite>
2001

```

2002 Secondly, the composite using autowire:

```

2003 <?xml version="1.0" encoding="UTF-8"?>
2004 <!-- Autowire Example - With autowire -->
2005 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2006     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2007     xmlns:foo="http://foo.com"
2008     targetNamespace="http://foo.com"
2009     name="AccountComposite">
2010
2011     <service name="PaymentService" promote="PaymentsComponent">
2012         <interface.java class="com.foo.PaymentServiceInterface"/>
2013     </service>
2014
2015     <component name="PaymentsComponent" autowire="true">
2016         <implementation.java class="com.foo.accounts.Payments"/>
2017         <service name="PaymentService"/>
2018         <reference name="CustomerAccountService"/>
2019         <reference name="ProductPricingService"/>
2020         <reference name="AccountsLedgerService"/>
2021         <reference name="ExternalBankingService"/>
2022     </component>
2023
2024     <component name="CustomerAccountComponent">
2025         <implementation.java class="com.foo.accounts.CustomerAccount"/>
2026     </component>
2027
2028     <component name="ProductPricingComponent">
2029         <implementation.java class="com.foo.accounts.ProductPricing"/>
2030     </component>
2031
2032     <component name="AccountsLedgerComponent">
2033         <implementation.composite name="foo:AccountsLedgerComposite"/>
2034     </component>
2035
2036     <reference name="ExternalBankingService"
2037         promote="PaymentsComponent/ExternalBankingService"/>
2038
2039 </composite>

```

2040 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires
2041 for any of its references – the wires are created automatically through autowire.

2042 **Note:** In the second example, it would be possible to omit all of the service and reference
2043 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the
2044 component service and references still exist, since they are provided by the implementation used
2045 by the component.

2046

2047 6.5 Using Composites as Component Implementations

2048 Composites may form **component implementations** in higher-level composites – in other words
2049 the higher-level composites can have components which are implemented by composites.

2050 When a composite is used as a component implementation, it defines a boundary of visibility.
2051 Components within the composite cannot be referenced directly by the using component. The
2052 using component can only connect wires to the services and references of the used composite and
2053 set values for any properties of the composite. The internal construction of the composite is
2054 invisible to the using component. The boundary of visibility, sometimes called encapsulation, can
2055 be enforced when assembling components and composites, but such encapsulation structures
2056 might not be enforceable in a particular implementation language.

2057 A composite used as a component implementation must also honor a completeness contract. The
2058 services, references and properties of the composite form a contract (represented by the
2059 component type of the composite) which is relied upon by the using component. The concept of
2060 completeness of the composite implies that, once all <include/> element processing is performed
2061 on the composite:

- 2062 1. For a composite used as a component implementation, each composite service offered by
2063 the composite MUST promote a component service of a component that is within the
2064 composite. [ASM60032]
- 2065 2. For a composite used as a component implementation, every component reference of
2066 components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or
2067 promoted (according to the various rules for specifying target services for a component
2068 reference described in section 5.3.1). [ASM60033]
- 2069 3. For a composite used as a component implementation, all properties of components within
2070 the composite, where the underlying component implementation specifies
2071 "mustSupply=true" for the property, MUST either specify a value for the property or
2072 source the value from a composite property. [ASM60034]

2073 The component type of a composite is defined by the set of composite service elements,
2074 composite reference elements and composite property elements that are the children of the
2075 composite element.

2076 Composites are used as component implementations through the use of the
2077 **implementation.composite** element as a child element of the component. The schema snippet
2078 for the implementation.composite element is:

```
2079  
2080 <!-- implementation.composite pseudo-schema -->  
2081 <implementation.composite name="xs:QName" requires="list of xs:QName"?  
2082 policySets="list of xs:QName"?>
```

2083

2084

2085 The implementation.composite element has the following attributes:

- 2086 • **name (1..1)** – the name of the composite used as an implementation. The @name
2087 attribute of an <implementation.composite/> element MUST contain the QName of a
2088 composite in the SCA Domain. [ASM60030]
- 2089 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)
2090 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or
2091 further qualify the required intents defined for the promoted component reference.
- 2092 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
2093 [\[10\]](#) for a description of this attribute.

2094

2095 6.5.1 Example of Composite used as a Component Implementation

2096

2097 The following is an example of a composite which contains two components, each of which is
2098 implemented by a composite:
2099

```
2100 <?xml version="1.0" encoding="UTF-8"?>
2101 <!-- CompositeComponent example -->
2102 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2103           xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200712
2104           file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
2105           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2106           targetNamespace="http://foo.com"
2107           xmlns:foo="http://foo.com"
2108           name="AccountComposite">
2109
2110     <service name="AccountService" promote="AccountServiceComponent">
2111       <interface.java interface="services.account.AccountService"/>
2112       <binding.ws port="AccountService#
2113         wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
2114     </service>
2115
2116     <reference name="stockQuoteService"
2117       promote="AccountServiceComponent/StockQuoteService">
2118       <interface.java
2119         interface="services.stockquote.StockQuoteService"/>
2120       <binding.ws
2121         port="http://www.quickstockquote.com/StockQuoteService#
2122         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2123     </reference>
2124
2125     <property name="currency" type="xsd:string">EURO</property>
2126
2127     <component name="AccountServiceComponent">
2128       <implementation.composite name="foo:AccountServiceComposite1"/>
2129
2130       <reference name="AccountDataService" target="AccountDataService"/>
2131       <reference name="StockQuoteService"/>
2132
2133       <property name="currency" source="$currency"/>
2134     </component>
2135
2136     <component name="AccountDataService">
2137       <implementation.composite name="foo:AccountDataServiceComposite"/>
2138
2139       <property name="currency" source="$currency"/>
2140     </component>
2141 </composite>
2142
2143
```

2144 6.6 Using Composites through Inclusion

2145 In order to assist team development, composites may be developed in the form of multiple
2146 physical artifacts that are merged into a single logical unit.

2147 A composite may include another composite by using the **include** element. This provides a
2148 recursive inclusion capability. The semantics of included composites are that the element content

2149 children of the included composite are inlined, with certain modification, into the using composite.
2150 This is done recursively till the resulting composite does not contain an **include** element. The
2151 outer included composite element itself is discarded in this process – only its contents are included
2152 as described below:

- 2153 1. All the element content children of the included composite are inlined in the including
2154 composite.
- 2155 2. The attributes **targetNamespace**, **name**, **constrainingType**, and **local** of the included
2156 composites are discarded.
- 2157 3. All the namespace declaration on the included composite element are added to the inlined
2158 element content children unless the namespace binding is overridden by the element
2159 content children.
- 2160 4. The attribute **autowire**, if specified on the included composite, is included on all inlined
2161 component element children unless the component child already specifies that attribute.
- 2162 5. The attribute values of **requires** and **policySet**, if specified on the included composite,
2163 are merged with corresponding attribute on the inlined component, service and reference
2164 children elements. Merge in this context means a set union.
- 2165 6. Extension attributes ,if present on the included composite, must follow the rules defined
2166 for that extension. Authors of attribute extensions on the composite element must define
2167 rules for inclusion.

2168 If the included composite has the value *true* for the attribute **local** then the including composite
2169 must have the same value for the **local** attribute, else it is considered an error.

2170 The composite file used for inclusion can have any contents, but its document root element must
2171 be **composite**. The composite element may contain any of the elements which are valid as child
2172 elements of a composite element, namely components, services, references, wires and includes.
2173 There is no need for the content of an included composite to be complete, so that artifacts defined
2174 within the using composite or in another associated included composite file may be referenced. For
2175 example, it is permissible to have two components in one composite file while a wire specifying
2176 one component as the source and the other as the target can be defined in a second included
2177 composite file.

2178 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one
2179 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated
2180 elements in the using composite (eg. two services with the same uri contributed by different
2181 included composites). It is not considered an error if the (using) composite resulting from the
2182 inclusion is incomplete (eg. wires with non-existent source or target). Such incomplete resulting
2183 composites are permitted to allow recursive composition.

2184 The following snippet shows the pseudo-schema for the include element.

```
2185  
2186 <?xml version="1.0" encoding="UTF-8"?>  
2187 <!-- Include snippet -->  
2188 <composite ...>  
2189   ...  
2190   <include name="xs:QName" /> *  
2191   ...  
2192 </composite>  
2193
```

2194 The include element has the following **attribute**:

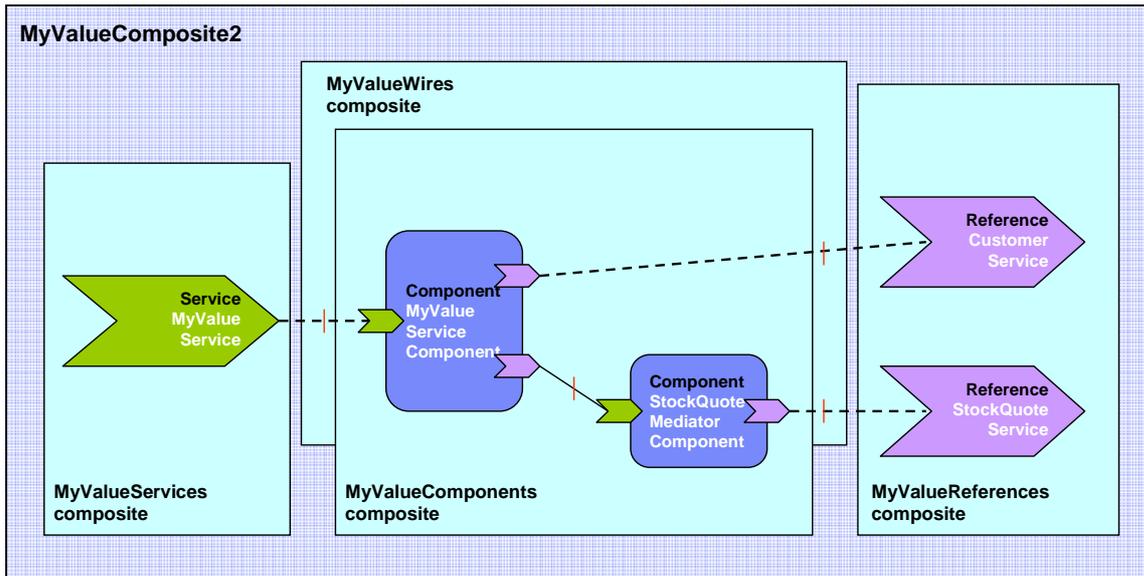
- 2195 • **name (required)** – the name of the composite that is included.

2196

2197 6.6.1 Included Composite Examples

2198

2199 The following figure shows the assembly diagram for the MyValueComposite2 containing four
 2200 included composites. The **MyValueServices composite** contains the MyValueService service. The
 2201 **MyValueComponents composite** contains the MyValueServiceComponent and the
 2202 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences**
 2203 **composite** contains the CustomerService and StockQuoteService references. The **MyValueWires**
 2204 **composite** contains the wires that connect the MyValueService service to the
 2205 MyValueServiceComponent, that connect the customerService reference of the
 2206 MyValueServiceComponent to the CustomerService reference, and that connect the
 2207 stockQuoteService reference of the StockQuoteMediatorComponent to the StockQuoteService
 2208 reference. Note that this is just one possible way of building the MyValueComposite2 from a set of
 2209 included composites.



2210
 2211
 2212 *Figure 13 MyValueComposite2 built from 4 included composites*

2213
 2214 The following snippet shows the contents of the MyValueComposite2.composite file for the
 2215 MyValueComposite2 built using included composites. In this sample it only provides the name of
 2216 the composite. The composite file itself could be used in a scenario using included composites to
 2217 define components, services, references and wires.

```
2218
2219 <?xml version="1.0" encoding="ASCII"?>
2220 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2221           targetNamespace="http://foo.com"
2222           xmlns:foo="http://foo.com"
2223           name="MyValueComposite2" >
2224
2225     <include name="foo:MyValueServices" />
2226     <include name="foo:MyValueComponents" />
2227     <include name="foo:MyValueReferences" />
2228     <include name="foo:MyValueWires" />
2229
2230 </composite>
2231
```

2232 The following snippet shows the content of the MyValueServices.composite file.

2233

```
2234 <?xml version="1.0" encoding="ASCII"?>
2235 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2236           targetNamespace="http://foo.com"
2237           xmlns:foo="http://foo.com"
2238           name="MyValueServices" >
2239
2240     <service name="MyValueService" promote="MyValueServiceComponent">
2241       <interface.java interface="services.myvalue.MyValueService" />
2242       <binding.ws port="http://www.myvalue.org/MyValueService#
2243         wsdl.endpoint(MyValueService/MyValueServicesSOAP)" />
2244     </service>
2245
2246 </composite>
2247
```

2248 The following snippet shows the content of the MyValueComponents.composite file.

2249

```
2250 <?xml version="1.0" encoding="ASCII"?>
2251 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2252           targetNamespace="http://foo.com"
2253           xmlns:foo="http://foo.com"
2254           name="MyValueComponents" >
2255
2256     <component name="MyValueServiceComponent" >
2257       <implementation.java
2258         class="services.myvalue.MyValueServiceImpl" />
2259       <property name="currency">EURO</property>
2260     </component>
2261
2262     <component name="StockQuoteMediatorComponent" >
2263       <implementation.java class="services.myvalue.SQMediatorImpl" />
2264       <property name="currency">EURO</property>
2265     </component>
2266
2267 </composite>
2268
```

2269 The following snippet shows the content of the MyValueReferences.composite file.

2270

```
2271 <?xml version="1.0" encoding="ASCII"?>
2272 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2273           targetNamespace="http://foo.com"
2274           xmlns:foo="http://foo.com"
2275           name="MyValueReferences" >
2276
2277     <reference name="CustomerService"
2278       promote="MyValueServiceComponent/CustomerService">
2279       <interface.java interface="services.customer.CustomerService" />
2280       <binding.sca/>
2281     </reference>
2282
2283     <reference name="StockQuoteService"
2284       promote="StockQuoteMediatorComponent">
2285       <interface.java
```

```

2286         interface="services.stockquote.StockQuoteService" />
2287     <binding.ws port="http://www.stockquote.org/StockQuoteService#
2288         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP) " />
2289     </reference>
2290
2291 </composite>
2292

```

2293 The following snippet shows the content of the MyValueWires.composite file.

```

2294
2295 <?xml version="1.0" encoding="ASCII" ?>
2296 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2297     targetNamespace="http://foo.com"
2298     xmlns:foo="http://foo.com"
2299     name="MyValueWires" >
2300
2301     <wire source="MyValueServiceComponent/stockQuoteService"
2302         target="StockQuoteMediatorComponent" />
2303
2304 </composite>

```

2305 6.7 Composites which Include Component Implementations of 2306 Multiple Types

2307 A Composite containing multiple components can have multiple component implementation types.
2308 For example, a Composite may include one component with a Java POJO as its implementation
2309 and another component with a BPEL process as its implementation.

2310 6.8 Structural URI of Components

2311 The **structural URI** is a relative URI that describes each use of a given component in the Domain,
2312 relative to the URI of the domain itself. It is never specified explicitly, but it calculated from the
2313 configuration of the components configured into the Domain.

2314 A component in a composite may be used more than once in the domain, if its containing
2315 composite is used as the implementation of more than one higher-level component. The structural
2316 URI may be used to separately identify each use of a component - for example, the structural URI
2317 may be used to attach different policies to each separate use of a component.

2318 For components directly deployed into the domain, the structural URI is simply the name of the
2319 component.

2320 Where components are nested within a composite which is used as the implementation of a higher
2321 level component, the structural URI consists of the name of the nested component prepended with
2322 each of the names of the components upto and including the domain level component.

2323 For example, consider a component named Component1 at the domain level, where its
2324 implementation is Composite1 which in turn contains a component named Component2, which is
2325 implemented by Composite2 which contains a component named Component3. The three
2326 components in this example have the following structural URIs:

- 2327 1. Component1: Component1
- 2328 2. Component2: Component1/Component2
- 2329 3. Component3: Component1/Component2/Component3

2330 The structural URI can also be extended to refer to specific parts of a component, such as a
2331 service or a reference, by appending an appropriate fragment identifier to the component's
2332 structural URI, as follows:

2333
2334
2335

- Service:
#service(servicename)

2336
2337
2338

- Reference:
#reference(referencename)

2339
2340
2341

- Service binding:
#service-binding(servicename/bindingname)

2342
2343

- Reference binding:
#reference-binding(referencename/bindingname)

2344
2345
2346

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

2347

7 ConstrainingType

2348

SCA allows a component, and its associated implementation, to be constrained by a **constrainingType**. The **constrainingType** element provides assistance in developing top-down usecases in SCA, where an architect or assembler can define the structure of a composite, including the required form of component implementations, before any of the implementations are developed.

2353

A **constrainingType** is expressed as an element which has services, reference and properties as child elements and which can have intents applied to it. The **constrainingType** is independent of any implementation. Since it is independent of an implementation it cannot contain any implementation-specific configuration information or defaults. Specifically, it cannot contain bindings, policySets, property values or default wiring information. The **constrainingType** is applied to a component through a **constrainingType** attribute on the component.

2354

2355

2356

2357

2358

2359

A **constrainingType** provides the "shape" for a component and its implementation. Any component configuration that points to a **constrainingType** is constrained by this shape. The **constrainingType** specifies the services, references and properties that **MUST** be implemented by the implementation of the component to which the **constrainingType** is attached. [ASM70001] This provides the ability for the implementer to program to a specific set of services, references and properties as defined by the **constrainingType**. Components are therefore configured instances of implementations and are constrained by an associated **constrainingType**.

2360

2361

2362

2363

2364

2365

2366

If the configuration of the component or its implementation do not conform to the **constrainingType** specified on the component element, the SCA runtime **MUST** raise an error. [ASM70002]

2367

2368

2369

A **constrainingType** is represented by a **constrainingType** element. The following snippet shows the pseudo-schema for the composite element.

2370

2371

2372

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

2373

```
<!-- ConstrainingType schema snippet -->
```

2374

```
<constrainingType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
```

2375

```
    targetNamespace="xs:anyURI"?
```

2376

```
    name="xs:NCName" requires="list of xs:QName"?>
```

2377

2378

2379

```
    <service name="xs:NCName" requires="list of xs:QName"?>*
```

2380

```
        <interface ... />?
```

2381

```
    </service>
```

2382

2383

```
    <reference name="xs:NCName"
```

2384

```
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?
```

2385

```
        requires="list of xs:QName"?>*
```

2386

```
        <interface ... />?
```

2387

```
    </reference>
```

2388

2389

```
    <property name="xs:NCName" ( type="xs:QName" | element="xs:QName" )
```

2390

```
        many="xs:boolean"? mustSupply="xs:boolean"?>*
```

2391

2392 default-property-value?

2393 </property>

2394

2395 </constrainingType>

2396

2397 The constrainingType element has the following **attributes**:

- 2398 • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is
2399 an XML QName, in the namespace identified by the targetNamespace attribute. The name
2400 attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2401 • **targetNamespace (0..1)** – an identifier for a target namespace into which the
2402 constrainingType is declared
- 2403 • **requires (0..1)** – a list of policy intents. See [the Policy Framework specification \[10\]](#) for
2404 a description of this attribute.

2405 ConstrainingType contains **zero or more properties, services, references**.

2406

2407 When an implementation is constrained by a constrainingType its component type MUST contain
2408 all the services, references and properties specified in the constrainingType. [ASM70004] The
2409 constraining type's references and services will have interfaces specified and can have intents
2410 specified. An implementation MAY contain additional services, additional optional references
2411 (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the
2412 constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or
2413 1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

2414 When a component is constrained by a constrainingType via the "constrainingType" attribute, the
2415 entire componentType associated with the component and its implementation is not visible to the
2416 containing composite. The containing composite can only see a projection of the componentType
2417 associated with the component and implementation as scoped by the constrainingType of the
2418 component. Additional services, references and properties provided by the implementation which
2419 are not declared in the constrainingType associated with a component MUST NOT be configured in
2420 any way by the containing composite. [ASM70006] This requirement ensures that the
2421 constrainingType contract cannot be violated by the composite.

2422 The constrainingType can include required intents on any element. Those intents are applied to
2423 any component that uses that constrainingType. In other words, if requires="reliability" exists on
2424 a constrainingType, or its child service or reference elements, then a constrained component or its
2425 implementation must include requires="reliability" on the component or implementation or on its
2426 corresponding service or reference. A component or implementation can use a qualified form of
2427 an intent specified in unqualified form in the constrainingType, but if the constrainingType uses
2428 the qualified form of an intent, then the component or implementation MUST also use the qualified
2429 form, otherwise there is an error. [ASM70007]

2430 A constrainingType can be applied to an implementation. In this case, the implementation's
2431 componentType has a constrainingType attribute set to the QName of the constrainingType.

2432

2433 7.1 Example constrainingType

2434

2435 The following snippet shows the contents of the component called "MyValueServiceComponent"
2436 which is constrained by the constrainingType myns:CT. The componentType associated with the
2437 implementation is also shown.

2438

```
2439                   <component name="MyValueServiceComponent" constrainingType="myns:CT">  
2440                    <implementation.java class="services.myvalue.MyValueServiceImpl" />
```

```

2441     <property name="currency">EURO</property>
2442     <reference name="customerService" target="CustomerService">
2443         <binding.ws ...>
2444     <reference name="StockQuoteService"
2445         target="StockQuoteMediatorComponent"/>
2446 </component>
2447
2448 <constrainingType name="CT"
2449     targetNamespace="http://myns.com">
2450     <service name="MyValueService">
2451         <interface.java interface="services.myvalue.MyValueService"/>
2452     </service>
2453     <reference name="customerService">
2454         <interface.java interface="services.customer.CustomerService"/>
2455     </reference>
2456     <reference name="stockQuoteService">
2457         <interface.java interface="services.stockquote.StockQuoteService"/>
2458     </reference>
2459     <property name="currency" type="xsd:string"/>
2460 </constrainingType>

```

2461 The component MyValueServiceComponent is constrained by the constrainingType CT which
2462 means that it must provide:

- 2463 • service **MyValueService** with the interface services.myvalue.MyValueService
- 2464 • reference **customerService** with the interface services.stockquote.StockQuoteService
- 2465 • reference **stockQuoteService** with the interface services.stockquote.StockQuoteService
- 2466 • property **currency** of type xsd:string.

2467 8 Interface

2468 **Interfaces** define one or more business functions. These business functions are provided by
2469 Services and are used by References. A Service offers the business functionality of exactly one
2470 interface for use by other components. Each interface defines one or more service **operations**
2471 and each operation has zero or one **request (input) message** and zero or one **response**
2472 **(output) message**. The request and response messages can be simple types such as a string
2473 value or they can be complex types.

2474 SCA currently supports the following interface type systems:

- 2475 • Java interfaces
- 2476 • WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))
- 2477 • C++ classes
- 2478 • Collections of 'C' functions

2479 SCA is also extensible in terms of interface types. Support for other interface type systems can be
2480 added through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2481

2482 The following snippet shows the definition for the **interface** base element.

2483

```
2484 <interface requires="list of xs:QName"? policySets="list of xs:QName"?/>
```

2485

2486 The **interface** base element has the following **attributes**:

- 2487 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)
2488 [\[10\]](#) for a description of this attribute
- 2489 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)
2490 [\[10\]](#) for a description of this attribute.

2491

2492 For information about Java interfaces, including details of SCA-specific annotations, see the SCA
2493 Java Common Annotations and APIs specification [SCA-Common-Java].

2494 For information about WSDL interfaces, including details of SCA-specific extensions, see SCA-
2495 Specific Aspects for WSDL Interfaces and WSDL Interface Type.

2496 For information about C++ interfaces, see the SCA C++ Client and Implementation Model
2497 specification [SCA-CPP-Client].

2498 For information about C interfaces, see the SCA C Client and Implementation Model specification
2499 [SCA-C-Client].

2500 8.1 Local and Remotable Interfaces

2501 A remotable service is one which may be called by a client which is running in an operating system
2502 process different from that of the service itself (this also applies to clients running on different
2503 machines from the service). Whether a service of a component implementation is remotable is
2504 defined by the interface of the service. WSDL defined interfaces are always remotable. See the
2505 relevant specifications for details of interfaces defined using other languages.

2506

2507 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**
2508 interactions. Remotable service Interfaces MUST NOT make use of **method or operation**
2509 **overloading.** [ASM80002] This restriction on operation overloading for remotable services aligns

Deleted: Remotable service Interfaces MUST NOT make use of **method or operation overloading.**

2510 with the WSDL 2.0 specification, which disallows operation overloading, and also with the WS-I
2511 Basic Profile 1.1 (section 4.5.3 - R2304) which has a constraint which disallows operation
2512 overloading when using WSDL 1.1.
2513

2514 Independent of whether the remotable service is called remotely from outside the process where
2515 the service runs or from another component running in the same process, the data exchange
2516 semantics are **by-value**.

2517 Implementations of remotable services can modify input messages (parameters) during or after
2518 an invocation and can modify return messages (results) after the invocation. If a remotable
2519 service is called locally or remotely, the SCA container MUST ensure sure that no modification of
2520 input messages by the service or post-invocation modifications to return messages are seen by
2521 the caller. [ASM80003]

2522 Here is a snippet which shows an example of a remotable java interface:

```
2523  
2524 package services.hello;  
2525  
2526 @Remotable  
2527 public interface HelloService {  
2528  
2529     String hello(String message);  
2530 }  
2531
```

2532 It is possible for the implementation of a remotable service to indicate that it can be called using
2533 by-reference data exchange semantics when it is called from a component in the same process.
2534 This can be used to improve performance for service invocations between components that run in
2535 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java
2536 Client and Implementation Specification](#)).

2537 A service typed by a local interface can only be called by clients that are running in the same
2538 process as the component that implements the local service. Local services cannot be published
2539 via remotable services of a containing composite. In the case of Java a local service is defined by a
2540 Java interface definition without a **@Remotable** annotation.

2541 The style of local interfaces is typically **fine grained** and intended for **tightly coupled**
2542 interactions. Local service interfaces can make use of **method or operation overloading**.

2543 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

2544

2545 8.2 Bidirectional Interfaces

2546 The relationship of a business service to another business service is often peer-to-peer, requiring
2547 a two-way dependency at the service level. In other words, a business service represents both a
2548 consumer of a service provided by a partner business service and a provider of a service to the
2549 partner business service. This is especially the case when the interactions are based on
2550 asynchronous messaging rather than on remote procedure calls. The notion of **bidirectional**
2551 **interfaces** is used in SCA to directly model peer-to-peer bidirectional business service
2552 relationships.

2553 An interface element for a particular interface type system needs to allow the specification of an
2554 optional callback interface. If a callback interface is specified, SCA refers to the interface as a
2555 whole as a bidirectional interface.

2556 The following snippet shows the interface element defined using Java interfaces with an optional
2557 callbackInterface attribute.

2558

2559 <interface.java interface="services.invoicing.ComputePrice"
2560 callbackInterface="services.invoicing.InvoiceCallback"/>

2561

2562 If a service is defined using a bidirectional interface element then its implementation implements
2563 the interface, and its implementation uses the callback interface to converse with the client that
2564 called the service interface.

2565

2566 If a reference is defined using a bidirectional interface element, the client component
2567 implementation using the reference calls the referenced service using the interface. The client
2568 MUST provide an implementation of the callback interface. [ASM80004]

2569 Callbacks can be used for both remotable and local services. Either both interfaces of a
2570 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT
2571 mix local and remote services. [ASM80005]

2572 Note that an interface document such as a WSDL file or a Java interface can contain annotations
2573 that declare a callback interface for a particular interface (see [the section on WSDL Interface type](#)
2574 and the Java Common Annotations and APIs specification [SCA-Common-Java]). Whenever an
2575 interface document declaring a callback interface is used in the declaration of an <interface/>
2576 element in SCA, it MUST be treated as being bidirectional with the declared callback interface.
2577 [ASM80010] In such cases, there is no requirement for the <interface/> element to declare the
2578 callback interface explicitly.

2579 If an <interface/> element references an interface document which declares a callback interface
2580 and also itself contains a declaration of a callback interface, the two callback interfaces MUST be
2581 compatible. [ASM80011]

2582 Where a component uses an implementation and the component configuration explicitly declares
2583 an interface for a service or a reference, if the matching service or reference declaration in the
2584 component type declares an interface which has a callback interface, then the component interface
2585 declaration MUST also declare a compatible interface with a compatible callback interface.
2586 [ASM80012] If the service or reference declaration in the component type declares an interface
2587 without a callback interface, then the component configuration for the corresponding service or
2588 reference MUST NOT declare an interface with a callback interface. [ASM80013]

2589 Where a composite declares an interface for a composite service or a composite reference, if the
2590 promoted service or promoted reference has an interface which has a callback interface, then the
2591 interface declaration for the composite service or the composite reference MUST also declare a
2592 compatible interface with a compatible callback interface. [ASM80014] If the promoted service or
2593 promoted reference has an interface without a callback interface, then the interface declaration for
2594 the composite service or composite reference MUST NOT declare a callback interface.
2595 [ASM80015]

2596 See Section 6.4 Wires for a definition of "compatible interfaces".

2597 In a bidirectional interface, the service interface can have more than one operation defined, and
2598 the callback interface can also have more than one operation defined. SCA runtimes MUST allow
2599 an invocation of any operation on the service interface to be followed by zero, one or many
2600 invocations of any of the operations on the callback interface. [ASM80009] These callback
2601 operations can be invoked either before or after the operation on the service interface has
2602 returned a response message, if there is one.

2603 For a given invocation of a service operation, which operations are invoked on the callback
2604 interface, when these are invoked, the number of operations invoked, and their sequence are not
2605 described by SCA. It is possible that this metadata about the bidirectional interface can be
2606 supplied through mechanisms outside SCA. For example, it might be provided as a written
2607 description attached to the callback interface.

2608 8.3 Conversational Interfaces

2609 Services sometimes cannot easily be defined so that each operation stands alone and is
2610 completely independent of the other operations of the same service. Instead, there is a sequence
2611 of operations that must be called in order to achieve some higher level goal. SCA calls this
2612 sequence of operations a **conversation**. If the service uses a bidirectional interface, the
2613 conversation may include both operations and callbacks.

2614 Such **conversational services** are typically managed by using conversation identifiers that are
2615 either (1) part of the application data (message parts or operation parameters) or 2)
2616 communicated separately from application data (possibly in headers). SCA introduces the concept
2617 of **conversational interfaces** for describing the interface contract for conversational services of
2618 the second form above. With this form, it is possible for the runtime to automatically manage the
2619 conversation, with the help of an appropriate binding specified at deployment. SCA does not
2620 standardize any aspect of conversational services that are maintained using application data.
2621 Such services are neither helped nor hindered by SCA's conversational service support.

2622 Conversational services typically involve state data that relates to the conversation that is taking
2623 place. The creation and management of the state data for a conversation has a significant impact
2624 on the development of both clients and implementations of conversational services.

2625

2626 Traditionally, application developers who have needed to write conversational services have been
2627 required to write a lot of plumbing code. They need to:

2628

- 2629 - choose or define a protocol to communicate conversational (correlation) information
2630 between the client & provider
- 2631 - route conversational messages in the provider to a machine that can handle that
2632 conversation, while handling concurrent data access issues
- 2633 - write code in the client to use/encode the conversational information
- 2634 - maintain state that is specific to the conversation, sometimes persistently and
2635 transactionally, both in the implementation and the client.

2636

2637 SCA makes it possible to divide the effort associated with conversational services between a
2638 number of roles:

- 2639 - Application Developer: Declares that a service interface is conversational (leaving the
2640 details of the protocol up to the binding). Uses lifecycle semantics, APIs or other
2641 programmatic mechanisms (as defined by the implementation-type being used) to
2642 manage conversational state.
- 2643 - Application Assembler: chooses a binding that can support conversations
- 2644 - Binding Provider: implements a protocol that can pass conversational information with
2645 each operation request/response.
- 2646 - Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for
2647 application developers to access conversational information. Optionally implements
2648 instance lifecycle semantics that automatically manage implementation state based on
2649 the binding's conversational information.

2650

2651 There is a policy intent with the name **conversational** which is used to mark an interface as being
2652 conversational in nature. Where a service or a reference has a conversational interface, the
2653 conversational intent MUST be attached either to the interface itself, or to the service or reference
2654 using the interface. [\[ASM80006\]](#) How to attach the conversational intent to an interface depends
2655 on the type of the interface. For a WSDL interface, this is described in section 8.4 "[SCA-Specific](#)
2656 [Aspects for WSDL Interfaces](#)". For a Java interface, it is described in the Java Common
2657 Annotations and APIs specification. Note that setting the conversational intent on the service or

2658 reference element is useful when reusing an existing interface definition that contains no SCA
2659 information, since it requires no modification of the interface artifact.

2660 The meaning of the conversational intent is that both the client and the provider of the interface
2661 can assume that messages (in either direction) will be handled as part of an ongoing conversation
2662 without depending on identifying information in the body of the message (i.e. in parameters of the
2663 operations). In effect, the conversation interface specifies a high-level abstract protocol that must
2664 be satisfied by any actual binding/policy combination used by the service.

2665 Examples of binding/policy combinations that support conversational interfaces are:

- 2666 - Web service binding with a WS-RM policy
- 2667 - Web service binding with a WS-Addressing policy
- 2668 - Web service binding with a WS-Context policy
- 2669 - JMS binding with a conversation policy that uses the JMS correlationID header

2670

2671 Conversations occur between one client and one target service. Consequently, requests originating
2672 from one client to multiple target conversational services will result in multiple conversations. For
2673 example, if a client A calls services B and C, both of which implement conversational interfaces,
2674 two conversations result, one between A and B and another between A and C. Likewise, requests
2675 flowing through multiple implementation instances will result in multiple conversations. For
2676 example, a request flowing from A to B and then from B to C will involve two conversations (A and
2677 B, B and C). In the previous example, if a request was then made from C to A, a third
2678 conversation would result (and the implementation instance for A would be different from the one
2679 making the original request).

2680 Invocation of any operation of a conversational interface can start a conversation. The decision on
2681 whether an operation starts a conversation depends on the component's implementation and its
2682 implementation type. Implementation types can support components which provide conversational
2683 services. If an implementation type does provide this support, the specification for that
2684 implementation type defines a mechanism for determining when a new conversation should be
2685 used for an operation (for example, in Java, the conversation is new on the first use of an injected
2686 reference; in BPEL, the conversation is new when the client's partnerLink comes into scope).

2687

2688 One or more operations in a conversational interface can be annotated with an
2689 **endsConversation** annotation (the mechanism for annotating the interface depends on the
2690 interface type) which indicates that when the operation is invoked, the conversation is at an end.
2691 Where an interface is **bidirectional**, operations may also be annotated in this way on operations
2692 of the callback interface. When a conversation ending operation is called, it indicates to both the
2693 client and the service provider that the conversation is complete. Once an operation marked with
2694 endsConversation has been invoked, any subsequent attempts to call an operation or a callback
2695 operation associated with the same conversation MUST generate a sca:ConversationViolation fault.
2696 **[ASM80007]**

2697 A sca:ConversationViolation fault is thrown when one of the following errors occur:

- 2698 - A message is received for a particular conversation, after the conversation has ended
- 2699 - The conversation identification is invalid (not unique, out of range, etc.)
- 2700 - The conversation identification is not present in the input message of the operation that
2701 ends the conversation
- 2702 - The client or the service attempts to send a message in a conversation, after the
2703 conversation has ended

2704 This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI
2705 <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2706 The lifecycle of resources and the association between unique identifiers and conversations are
2707 determined by the service's implementation type and may not be directly affected by the

2708 "endConversation" annotation. For example, a **WS-BPEL** process can outlive most of the
2709 conversations that it is involved in.

2710 Although conversational interfaces do not require that any identifying information be passed as
2711 part of the body of messages, there is conceptually an identity associated with the conversation.
2712 Individual implementations types can have an API to access the ID associated with the
2713 conversation, although no assumptions can be made about the structure of that identifier.
2714 Implementation types can also have a means to set the conversation ID by either the client or the
2715 service provider, although the operation may only be supported by some binding/policy
2716 combinations.

2717 Implementation-type specifications are encouraged to define and provide conversational instance
2718 lifecycle management for components that implement conversational interfaces. However,
2719 implementations could also manage the conversational state manually.

2720

2721 8.4 Long-running Request-Response Operations

2722 8.4.1 Background

2723 A service offering one or more operations which map to a WSDL request-response pattern may be
2724 implemented in a long-running, potentially interruptible, way. Consider a BPEL process with
2725 receive and reply activities referencing the WSDL request-response operation. Between the two
2726 activities, the business process logic may be a long-running sequence of steps, including activities
2727 causing the process to be interrupted. Typical examples are steps where the process waits for
2728 another message to arrive or a specified time interval to expire, or the process may perform
2729 asynchronous interactions such as service invocations bound to asynchronous protocols or user
2730 interactions. This is a common situation in business processes, and it causes the implementation
2731 of the WSDL request-response operation to run for a very long time, e.g., several months (!). In
2732 this case, it is not meaningful for any caller to remain in a synchronous wait for the response while
2733 blocking system resources or holding database locks.

2734 Note that it is possible to model long-running interactions as a pair of two independent operations
2735 as described in the section on bidirectional interfaces. However, it is a common practice (and in
2736 fact much more convenient) to model a request-response operation and let the infrastructure deal
2737 with the asynchronous message delivery and correlation aspects instead of putting this burden on
2738 the application developer.

2739

2740 8.4.2 Definition of "long-running"

2741 A request-response operation is considered long-running if the implementation does not guarantee
2742 the delivery of the response within any specified time interval. Clients invoking such request-
2743 response operations are strongly discouraged from making assumptions about when the response
2744 can be expected.

2745

2746 8.4.3 The asyncInvocation Intent

2747 This specification permits a long-running request-response operation or a complete interface
2748 containing such operations to be marked using a policy intent with the name **asyncInvocation**. It
2749 is also possible for a service to set the `asyncInvocation` intent when using an interface which is
2750 not marked with the `asyncInvocation` intent. This can be useful when reusing an existing interface
2751 definition that does not contain SCA information.

2752

2753 8.4.4 Requirements on Bindings

2754 In order to support a service operation which is marked with the `asyncInvocation` intent, it is
2755 necessary for the binding (and its associated policies) to support separate handling of the request
2756 message and the response message. Bindings which only support a synchronous style of message
2757 handling, such as a conventional HTTP binding, cannot be used to support long-running
2758 operations.

2759 The requirements on a binding to support the `asyncInvocation` intent are the same as those
2760 required to support services with bidirectional interfaces - namely that the binding needs to be
2761 able to treat the transmission of the request message separately from the transmission of the
2762 response message, with an arbitrarily large time interval between the two transmissions.

2763 An example of a binding/policy combination that supports long-running request-response
2764 operations is a Web service binding used in conjunction with the WS-Addressing
2765 "wsam:NonAnonymousResponses" assertion.

2766

2767 8.4.5 Implementation Type Support

2768 SCA implementation types can provide special asynchronous client-side and asynchronous server-
2769 side mappings to assist in the development of services and clients for long-running request-
2770 response operations.

2771 8.5 SCA-Specific Aspects for WSDL Interfaces

2772 There are a number of aspects that SCA applies to interfaces in general, such as marking them
2773 **conversational**. These aspects apply to the interfaces themselves, rather than their use in a
2774 specific place within SCA. There is thus a need to provide appropriate ways of marking the
2775 interface definitions themselves, which go beyond the basic facilities provided by the interface
2776 definition language.

2777 For WSDL interfaces, there is an extension mechanism that permits additional information to be
2778 included within the WSDL document. SCA takes advantage of this extension mechanism. In order
2779 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-](http://docs.oasis-open.org/ns/opencsa/sca/200712)
2780 [open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2781 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach
2782 policy intents - **@requires**. The definition of this attribute is as follows:

```
2783 <attribute name="requires" type="sca:listOfQNames"/>
```

2784

```
2785 <simpleType name="listOfQNames">  
2786 <list itemType="QName"/>  
2787 </simpleType>
```

2788 The `@requires` attribute can be applied to WSDL Port Type elements (WSDL 1.1). The attribute
2789 contains one or more intent names, as defined by [the Policy Framework specification \[10\]](#). Any
2790 service or reference that uses an interface marked with required intents MUST implicitly add those
2791 intents to its own `@requires` list. **[ASM80008]**

2792 To specify that a WSDL interface is conversational, the following attribute setting is used on either
2793 the WSDL Port Type or WSDL Interface:

```
2794 requires="conversational"
```

2795 SCA defines an **endsConversation** attribute that is used to mark specific operations within a
2796 WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces
2797 which are also marked conversational. The `endsConversation` attribute is a global attribute in the
2798 SCA namespace, with the following definition:

```
2799 <attribute name="endsConversation" type="boolean" default="false"/>
```

2800

2801 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on
2802 the portType and the **endsConversation** attribute on one of the operations:

```
2803     ...  
2804     <portType name="LoanService" sca:requires="conversational">  
2805         <operation name="apply">  
2806             <input message="tns:ApplicationInput" />  
2807             <output message="tns:ApplicationOutput" />  
2808         </operation>  
2809         <operation name="cancel" sca:endsConversation="true">  
2810             </operation>  
2811         ...  
2812     </portType>  
2813     ...
```

2814 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on
2815 the portType and the **endsConversation** attribute on one of the operations:

```
2816     ...  
2817     <portType name="LoanService" sca:requires="conversational">  
2818         <operation name="apply">  
2819             <input message="tns:ApplicationInput" />  
2820             <output message="tns:ApplicationOutput" />  
2821         </operation>  
2822         <operation name="cancel" sca:endsConversation="true">  
2823             </operation>  
2824         ...  
2825     </portType>  
2826     ...
```

2827 SCA defines an attribute which is used to indicate that a given WSDL Port Type element (WSDL
2828 1.1) has an associated callback interface. This is the @callback attribute, which applies to a WSDL
2829 <portType/> element.

2830 The @callback attribute is defined as a global attribute in the SCA namespace, as follows:

```
2831     <attribute name="callback" type="QName" />
```

2832

2833 The value of the @callback attribute is the QName of a Port Type. The port type declared by the
2834 @callback attribute is the callback interface to use for the portType which is annotated by the
2835 @callback attribute.

2836

2837 Here is an example of a portType element with a callback attribute:

```
2838     ...  
2839     <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
2840         <operation name="apply">  
2841             <input message="tns:ApplicationInput" />  
2842             <output message="tns:ApplicationOutput" />  
2843         </operation>  
2844         ...  
2845     </portType>
```

2846

2847 8.6 WSDL Interface Type

2848 The WSDL interface type is used to declare interfaces for services and for references, where the interface
2849 is defined in terms of a WSDL document. An interface is defined in terms of a WSDL 1.1 Port Type with
2850 the arguments and return of the service operations described using XML schema.

2851
2852 A WSDL interface is declared by an *interface.wSDL* element. The following shows the pseudo-schema
2853 for the interface.wSDL element:

```
2854 <!-- WSDL Interface schema snippet -->  
2855 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?>
```

2856 The interface.wSDL element has the following *attributes*:

- 2857 • **interface (1..1)** - the URI of a WSDL Port Type

2858 | The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.1
2859 | document, [ASM80001]

- 2860 • **callbackInterface(0..1)** - an optional callback interface, which is the URI of a WSDL Port
2861 Type

2862 | The interface.wSDL @callbackInterface attribute, if present, MUST reference a portType of a
2863 | WSDL 1.1 document, [ASM80016]

2864

2865 The form of the URI for WSDL port types follows the syntax described in the WSDL 1.1 Element
2866 Identifiers specification [WSDL11_Identifiers]

Deleted: The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.1 document.

Deleted: The interface.wSDL @callbackInterface attribute, if present, MUST reference a portType of a WSDL 1.1 document.

2867 8.6.1 Example of interface.wSDL

```
2868 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2869 wsdl.porttype(StockQuote)"  
2870 callbackInterface="http://www.stockquote.org/StockQuoteService#  
2871 wsdl.porttype(StockQuoteCallback)"/>  
2872
```

2873 This declares an interface in terms of the WSDL port type "StockQuote" with a callback interface defined
2874 by the "StockQuoteCallback" port type.

2875

2876 9 Binding

2877 Bindings are used by services and references. References use bindings to describe the access
2878 mechanism used to call a service (which can be a service provided by another SCA composite).
2879 Services use bindings to describe the access mechanism that clients (which can be a client from
2880 another SCA composite) have to use to call the service.

2881 SCA supports the use of multiple different types of bindings. Examples include **SCA service, Web**
2882 **service, stateless session EJB, data base stored procedure, EIS service**. An SCA runtime
2883 MUST provide support for SCA service and Web service binding types. SCA provides an
2884 extensibility mechanism by which an SCA runtime can add support for additional binding types.
2885 For details on how additional binding types are defined, see the section on the Extension Model.

2886 A binding is defined by a **binding element** which is a child element of a service or of a reference
2887 element in a composite. The following snippet shows the composite schema with the schema for
2888 the binding element.

2889

```
2890 <?xml version="1.0" encoding="ASCII"?>
2891 <!-- Bindings schema snippet -->
2892 <composite ... >
2893   ...
2894   <service ... >*
2895     <interface ... />?
2896     <binding uri="xs:anyURI"? name="xs:NCName"?
2897       requires="list of xs:QName"?
2898       policySets="list of xs:QName"?>*
2899     <operation name="xs:NCName" requires="list of xs:QName"?
2900       policySets="list of xs:QName"?/>*
2901     <wireFormat/>?
2902     <operationSelector/>?
2903   </binding>
2904   <callback?>
2905     <binding uri="xs:anyURI"? name="xs:NCName"?
2906       requires="list of xs:QName"?
2907       policySets="list of xs:QName"?>+
2908     <operation name="xs:NCName" requires="list of xs:QName"?
2909       policySets="list of xs:QName"?/>*
2910     <wireFormat/>?
2911     <operationSelector/>?
2912   </binding>
2913 </callback>
2914 </service>
2915   ...
2916 <reference ... >*
2917   <interface ... />?
2918   <binding uri="xs:anyURI"? name="xs:NCName"?
2919     requires="list of xs:QName"?
2920     policySets="list of xs:QName"?>*
2921   <operation name="xs:NCName" requires="list of xs:QName"?
2922     policySets="list of xs:QName"?/>*
2923   <wireFormat/>?
2924   <operationSelector/>?
2925 </binding>
2926 <callback?>
2927   <binding uri="xs:anyURI"? name="xs:NCName"?
2928     requires="list of xs:QName"?
```

```

2929         policySets="list of xs:QName"?>+
2930         <operation name="xs:NCName" requires="list of xs:QName"?
2931             policySets="list of xs:QName"?/*
2932             <wireFormat/?>
2933             <operationSelector/?>
2934         </binding>
2935     </callback>
2936 </reference>
2937     ...
2938 </composite>
2939

```

2940 The element name of the binding element is architected; it is in itself a qualified name. The first
 2941 qualifier is always named "binding", and the second qualifier names the respective binding-type
 2942 (e.g. binding.composite, binding.ws, binding.ejb, binding.eis).

2943

2944 A binding element has the following attributes:

- 2945 • **uri (0..1)** - has the following semantic.
 - 2946 ○ The uri attribute can be omitted.
 - 2947 ○ For a binding of a **reference** the URI attribute defines the target URI of the
 2948 reference. This MUST be either the componentName/serviceName for a wire to an
 2949 endpoint within the SCA domain, or the accessible address of some service
 2950 endpoint either inside or outside the SCA domain (where the addressing scheme is
 2951 defined by the type of the binding). [ASM90001]
 - 2952 ○ The circumstances under which the uri attribute can be used are defined in
 2953 section "5.3.1 Specifying the Target Service(s) for a Reference."
 - 2954 ○ For a binding of a **service** the URI attribute defines the URI relative to the
 2955 component, which contributes the service to the SCA domain. The default value for
 2956 the URI is the value of the name attribute of the binding.
- 2957 • **name (0..1)** – a name for the binding instance (an NCName). The name attribute allows
 2958 distinction between multiple binding elements on a single service or reference. The
 2959 default value of the name attribute is the service or reference name. When a service or
 2960 reference has multiple bindings, only one binding can have the default name value; all
 2961 others must have a name value specified that is unique within the service or reference.
 2962 [ASM90002] The name also permits the binding instance to be referenced from elsewhere
 2963 – particularly useful for some types of binding, which can be declared in a definitions
 2964 document as a template and referenced from other binding instances, simplifying the
 2965 definition of more complex binding instances (see [the JMS Binding specification \[11\]](#) for
 2966 examples of this referencing).
- 2967 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a
 2968 description of this attribute.
- 2969 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[10\]](#) for a
 2970 description of this attribute.

Deleted: For a binding of a **reference** the URI attribute defines the target URI of the reference. This MUST be either the componentName/serviceName 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).

2971 A binding element has the following child elements:

- 2972 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are
 2973 used to describe characteristics of individual operations within the interface. For a detailed
 2974 description of the operation element, see [the Policy Framework specification \[SCA Policy\]](#).
- 2975 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See [the](#)
 2976 [wireFormat section](#) for details.
- 2977 • **operationSelector(0..1)** - an operationSelector element that is used to match a
 2978 particular message to a particular operation in the interface. See [the operationSelector](#)
 2979 [section](#) for details

2980 When multiple bindings exist for an service, it means that the service is available by any of the
2981 specified bindings. The technique that the SCA runtime uses to choose among available bindings
2982 is left to the implementation and it may include additional (nonstandard) configuration. Whatever
2983 technique is used needs to be documented by the runtime.

2984 Services and References can always have their bindings overridden at the SCA domain level,
2985 unless restricted by Intents applied to them.

2986 If a reference has any bindings they MUST be resolved which means that each binding MUST
2987 include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference
2988 MUST NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds
2989 of bindings that are acceptable for use with a reference, the user specifies either policy intents or
2990 policy sets.

2991 Users can also specifically wire, not just to a component service, but to a specific binding offered
2992 by that target service. To do so, a wire target MAY be specified with a syntax of
2993 "componentName/serviceName/bindingName". [ASM90004]
2994

2995

2996 The following sections describe the SCA and Web service binding type in detail.

2997

2998 9.1 Messages containing Data not defined in the Service Interface

2999 It is possible for a message to include information that is not defined in the interface used to
3000 define the service, for instance information may be contained in SOAP headers or as MIME
3001 attachments.

3002 Implementation types can make this information available to component implementations in their
3003 execution context. The specifications for these implementation types describe how this
3004 information is accessed and in what form it is presented.

3005

3006 9.2 WireFormat

3007 A wireFormat is the form that a data structure takes when it is transmitted using some
3008 communication binding. Another way to describe this is "the form that the data takes on the wire".
3009 A wireFormat can be specific to a given communication method, or it may be general, applying to
3010 many different communication methods. An example of a general wireFormat is XML text format.

3011 Where a particular SCA binding can accommodate transmitting data in more than one format, the
3012 configuration of the binding MAY include a definition of the wireFormat to use. This is done using
3013 an optional <sca:wireFormat/> subelement of the <binding/> element.

3014 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats
3015 to be the default wireFormat which applies if no <wireFormat/> subelement is present.

3016 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a
3017 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 3018 • <sca:wireFormat.xml/>
- 3019 • A wireFormat that transmits the data as an XML text datastructure
- 3020 • <sca:wireFormat.jms/>
- 3021 • The "default JMS wireFormat" as described in the JMS Binding specification

3022

3023 Specific wireFormats can have elements that include either attributes or subelements or both.

3024 For details about specific wireFormats, see the related SCA Binding specifications.

3025

3026 9.3 OperationSelector

3027 An operationSelector is necessary for some types of transport binding where messages are
3028 transmitted across the transport without any explicit relationship between the message and the
3029 interface operation to which it relates. SOAP is an example of a protocol where the messages do
3030 contain explicit information that relates each message to the operation it targets. However, other
3031 transport bindings have messages where this relationship is not expressed in the message or in
3032 any related headers (pure JMS messages, for example). In cases where the messages arrive at a
3033 service without any explicit information that maps them to specific operations, it is necessary for
3034 the metadata attached to the service binding to contain the required mapping information. The
3035 information is held in an operationSelector element which is a child element of the binding
3036 element.

3037 The base `sca:operationSelector` element is abstract and it has no attributes and no child elements.
3038 For a particular operationSelector, an extension subtype is defined, using substitution groups, for
3039 example:

- 3040 • `<sca:operationSelector.XPath/>`
- 3041 • An operation selector that uses XPath to filter out specific messages and target them to
3042 particular named operations.

3043

3044 Specific operationSelectors can have elements that include either attributes or subelements or
3045 both.

3046 For details about specific operationSelectors, see the related SCA Binding specifications.

3047

3048 9.4 Form of the URI of a Deployed Binding

3049 SCA Bindings specifications can choose to use the **structural URI** defined in the section
3050 "[Structural URI of Components](#)" above to derive a binding specific URI according to some Binding-
3051 related scheme. The relevant binding specification describes this.

3052 Alternatively, `<binding/>` elements have an optional `@URI` attribute, which is termed a
3053 bindingURI.

3054 If the bindingURI is specified on a given `<binding/>` element, the binding can optionally use it to
3055 derive an endpoint URI relevant to the binding. The derivation is binding specific and is described
3056 by the relevant binding specification.

3057 For `binding.sca`, which is described in the SCA Assembly specification, this is as follows:

- 3058 • If the binding uri attribute is specified on a reference, it identifies the target service in the
3059 SCA domain by specifying the service's structural URI.
- 3060 • If the binding uri attribute is specified on a service, it is ignored.

3061

3062 9.4.1 Non-hierarchical URIs

3063 Bindings that use non-hierarchical URI schemes (such as `jms:` or `mailto:`) may optionally make
3064 use of the "uri" attribute, which is the complete representation of the URI for that service
3065 binding. Where the binding does not use the "uri" attribute, the binding needs to offer a different
3066 mechanism for specifying the service address.

3067 9.4.2 Determining the URI scheme of a deployed binding

3068 One of the things that needs to be determined when building the effective URI of a deployed
3069 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is
3070 binding type specific.

3071 If the binding type supports a single protocol then there is only one URI scheme associated with it.
3072 In this case, that URI scheme is used.

3073 If the binding type supports multiple protocols, the binding type implementation determines the
3074 URI scheme by introspecting the binding configuration, which may include the policy sets
3075 associated with the binding.

3076 A good example of a binding type that supports multiple protocols is binding.ws, which can be
3077 configured by referencing either an "abstract" WSDL element (i.e. portType or interface) or a
3078 "concrete" WSDL element (i.e. binding, port or endpoint). When the binding references a PortType
3079 or Interface, the protocol and therefore the URI scheme is derived from the intents/policy sets
3080 attached to the binding. When the binding references a "concrete" WSDL element, there are two
3081 cases:

3082 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most
3083 common case. In this case, the URI scheme is given by the protocol/transport specified in the
3084 WSDL binding element.

3085 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,
3086 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"
3087 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined
3088 by looking at the policy sets attached to the binding.

3089 It's worth noting that an intent supported by a binding type may completely change the behavior
3090 of the binding. For example, when the intent "confidentiality/transport" is required by an HTTP
3091 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to
3092 "https".

3093

3094 9.5 SCA Binding

3095 The SCA binding element is defined by the following schema.

3096

3097 `<binding.sca />`

3098

3099 The SCA binding can be used for service interactions between references and services contained
3100 within the SCA domain. The way in which this binding type is implemented is not defined by the
3101 SCA specification and it can be implemented in different ways by different SCA runtimes. The only
3102 requirement is that the required qualities of service must be implemented for the SCA binding
3103 type. The SCA binding type is **not** intended to be an interoperable binding type. For
3104 interoperability, an interoperable binding type such as the Web service binding should be used.

3105 A service definition with no binding element specified uses the SCA binding.

3106 `<binding.sca/>` would only have to be specified in override cases, or when you specify a
3107 set of bindings on a service definition and the SCA binding should be one of them.

3108 If a reference does not have a binding, then the binding used can be any of the bindings
3109 specified by the service provider, as long as the intents required by the reference and
3110 the service are all respected.

3111 If the interface of the service or reference is local, then the local variant of the SCA
3112 binding will be used. If the interface of the service or reference is remotable, then either
3113 the local or remote variant of the SCA binding will be used depending on whether source
3114 and target are co-located or not.

3115 If a reference specifies an URI via its uri attribute, then this provides the default wire to a service
3116 provided by another domain level component. The value of the URI has to be as follows:

3117

- `<domain-component-name>/<service-name>`

3118

3119 9.5.1 Example SCA Binding

3120 The following snippet shows the MyValueComposite.composite file for the MyValueComposite
3121 containing the service element for the MyValueService and a reference element for the
3122 StockQuoteService. Both the service and the reference use an SCA binding. The target for the
3123 reference is left undefined in this binding and would have to be supplied by the composite in which
3124 this composite is used.

3125

```
3126 <?xml version="1.0" encoding="ASCII"?>
3127 <!-- Binding SCA example -->
3128 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3129           targetNamespace="http://foo.com"
3130           name="MyValueComposite" >
3131
3132     <service name="MyValueService" promote="MyValueComponent">
3133       <interface.java interface="services.myvalue.MyValueService"/>
3134       <binding.sca/>
3135       ...
3136     </service>
3137
3138     ...
3139
3140     <reference name="StockQuoteService"
3141               promote="MyValueComponent/StockQuoteReference">
3142       <interface.java interface="services.stockquote.StockQuoteService"/>
3143       <binding.sca/>
3144     </reference>
3145
3146 </composite>
3147
```

3148 9.6 Web Service Binding

3149 SCA defines a Web services binding. This is described in a [separate specification document \[9\]](#).

3150

3151 9.7 JMS Binding

3152 SCA defines a JMS binding. This is described in a [separate specification document \[11\]](#).

3153 10SCA Definitions

3154 There are a variety of SCA artifacts which are generally useful and which are not specific to a
3155 particular composite or a particular component. These shared artifacts include intents, policy sets,
3156 bindings, binding type definitions and implementation type definitions.

3157 All of these artifacts within an SCA Domain are defined in SCA contributions in files called META-
3158 INF/definitions.xml (relative to the contribution base URI). Although the definitions are specified
3159 within a single SCA contribution, the definitions are visible throughout the domain. Because of
3160 this, all of the QNames for the definitions contained in definitions.xml files MUST be unique within
3161 the domain. [ASM10001] The definitions.xml file contains a definitions element that conforms to
3162 the following pseudo-schema snippet:

```
3163 <?xml version="1.0" encoding="ASCII"?>
3164 <!-- Composite schema snippet -->
3165 <definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3166             targetNamespace="xs:anyURI">
3167
3168     <sca:intent/*>
3169
3170     <sca:policySet/*>
3171
3172     <sca:binding/*>
3173
3174     <sca:bindingType/*>
3175
3176     <sca:implementationType/*>
3177
3178 </definitions>
```

3179 The definitions element has the following attribute:

- 3180 • **targetNamespace (required)** – the namespace into which the child elements of this
3181 definitions element are placed (used for artifact resolution)

3182 The definitions element contains optional child elements – intent, policySet, binding, bindingtype
3183 and implementationType. These elements are described elsewhere in this specification or in [the](#)
3184 [SCA Policy Framework specification \[10\]](#). The use of the elements declared within a definitions
3185 element is described in the [SCA Policy Framework specification \[10\]](#) and in [the JMS Binding](#)
3186 [specification \[11\]](#).

3187

Deleted: all of the QNames
for the definitions
contained in
definitions.xml files MUST
be unique within the
domain

3188

11 Extension Model

3189

3190 The assembly model can be extended with support for new interface types, implementation types
3191 and binding types. The extension model is based on XML schema substitution groups. There are
3192 three XML Schema substitution group heads defined in the SCA namespace: **interface**,
3193 **implementation** and **binding**, for interface types, implementation types and binding types,
3194 respectively.

3195 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1],
3196 [9], [11]) use these XML Schema substitution groups to define some basic types of interfaces,
3197 implementations and bindings, but other types can be defined as required, where support for
3198 these extra ones is available from the runtime. The interface type elements, implementation type
3199 elements, and binding type elements defined by the SCA specifications are all part of the SCA
3200 namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200712"), as indicated in their
3201 respective schemas. New interface types, implementation types and binding types that are defined
3202 using this extensibility model, which are not part of these SCA specifications are defined in
3203 namespaces other than the SCA namespace.

3204 The "." notation is used in naming elements defined by the SCA specifications (e.g.
3205 <implementation.java ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel
3206 extensibility approach but as a naming convention that improves usability of the SCA assembly
3207 language.

3208

3209 **Note:** How to contribute SCA model extensions and their runtime function to an SCA runtime will
3210 be defined by a future version of the specification.

3211

11.1 Defining an Interface Type

3212 The following snippet shows the base definition for the **interface** element and **Interface** type
3213 contained in **sca-core.xsd**; see appendix for complete schema.

3214

```
3215 <?xml version="1.0" encoding="UTF-8"?>  
3216 <!-- (c) Copyright SCA Collaboration 2006 -->  
3217 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3218         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3219         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3220         elementFormDefault="qualified">
```

3221

3222 ...

3223

```
3224 <element name="interface" type="sca:Interface" abstract="true"/>  
3225 <complexType name="Interface"/>  
3226 <complexType name="Interface" abstract="true">  
3227   <attribute name="requires" type="sca:listOfQNames" use="optional"/>  
3228   <attribute name="policySets" type="sca:listOfQNames" use="optional"/>  
3229 </complexType>
```

3230

3231

3232
3233
3234
3235

```
...  
</schema>
```

3236 In the following snippet is an example of how the base definition is extended to support Java
3237 interfaces. The snippet shows the definition of the **interface.java** element and the
3238 **JavaInterface** type contained in **sca-interface-java.xsd**.

3239
3240
3241
3242
3243
3244

```
<?xml version="1.0" encoding="UTF-8"?>  
<schema xmlns="http://www.w3.org/2001/XMLSchema"  
  targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
```

3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255

```
  <element name="interface.java" type="sca:JavaInterface"  
    substitutionGroup="sca:interface"/>  
  <complexType name="JavaInterface">  
    <complexContent>  
      <extension base="sca:Interface">  
        <attribute name="interface" type="NCName"  
          use="required"/>  
      </extension>  
    </complexContent>  
  </complexType>  
</schema>
```

3256 In the following snippet is an example of how the base definition can be extended by other
3257 specifications to support a new interface not defined in the SCA specifications. The snippet shows
3258 the definition of the **my-interface-extension** element and the **my-interface-extension-type**
3259 type.

3260
3261
3262
3263
3264
3265

```
<?xml version="1.0" encoding="UTF-8"?>  
<schema xmlns="http://www.w3.org/2001/XMLSchema"  
  targetNamespace="http://www.example.org/myextension"  
  xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
  xmlns:tns="http://www.example.org/myextension">
```

3266
3267
3268
3269
3270
3271
3272
3273
3274
3275

```
  <element name="my-interface-extension"  
    type="tns:my-interface-extension-type"  
    substitutionGroup="sca:interface"/>  
  <complexType name="my-interface-extension-type">  
    <complexContent>  
      <extension base="sca:Interface">  
        ...  
      </extension>  
    </complexContent>  
  </complexType>
```

3276 </schema>
3277

3278 11.2 Defining an Implementation Type

3279 The following snippet shows the base definition for the *implementation* element and
3280 **Implementation** type contained in *sca-core.xsd*; see appendix 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/200712"  
3286         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3287         elementFormDefault="qualified">  
3288  
3289     ...  
3290  
3291     <element name="implementation" type="sca:Implementation"  
3292     abstract="true"/>  
3293     <complexType name="Implementation"/>  
3294  
3295     ...  
3296 </schema>  
3297
```

3298
3299 In the following snippet we show how the base definition is extended to support Java
3300 implementation. The snippet shows the definition of the *implementation.java* element and the
3301 **JavaImplementation** type contained in *sca-implementation-java.xsd*.

```
3302  
3303 <?xml version="1.0" encoding="UTF-8"?>  
3304 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3305         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3306         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3307  
3308     <element name="implementation.java" type="sca:JavaImplementation"  
3309             substitutionGroup="sca:implementation"/>  
3310     <complexType name="JavaImplementation">  
3311         <complexContent>  
3312             <extension base="sca:Implementation">  
3313                 <attribute name="class" type="NCName"  
3314                     use="required"/>  
3315             </extension>  
3316         </complexContent>  
3317     </complexType>  
3318 </schema>
```

3319 In the following snippet is an example of how the base definition can be extended by other
3320 specifications to support a new implementation type not defined in the SCA specifications. The
3321 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**
3322 **type** type.

```
3323 <?xml version="1.0" encoding="UTF-8"?>
3324 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3325         targetNamespace="http://www.example.org/myextension"
3326         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3327         xmlns:tns="http://www.example.org/myextension">
3328
3329     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3330           substitutionGroup="sca:implementation"/>
3331     <complexType name="my-impl-extension-type">
3332         <complexContent>
3333             <extension base="sca:Implementation">
3334                 ...
3335             </extension>
3336         </complexContent>
3337     </complexType>
3338 </schema>
3339
```

3340 In addition to the definition for the new implementation instance element, there needs to be an
3341 associated implementationType element which provides metadata about the new implementation
3342 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3343 <implementationType type="xs:QName"
3344                   alwaysProvides="list of intent xs:QName"
3345                   mayProvide="list of intent xs:QName"/>
3346
```

3347 The implementation type has the following attributes:

- 3348 • **type (1..1)** – the type of the implementation to which this implementationType element
3349 applies. This is intended to be the QName of the implementation element for the
3350 implementation type, such as "sca:implementation.java"
 - 3351 • **alwaysProvides (0..1)** – a set of intents which the implementation type always
3352 provides. See [the Policy Framework specification \[10\]](#) for details.
 - 3353 • **mayProvide (0..1)** – a set of intents which the implementation type may provide. See
3354 [the Policy Framework specification \[10\]](#) for details.
- 3355

3356 11.3 Defining a Binding Type

3357 The following snippet shows the base definition for the **binding** element and **Binding** type
3358 contained in **sca-core.xsd**; see appendix for complete schema.

```
3359
3360 <?xml version="1.0" encoding="UTF-8"?>
3361 <!-- binding type schema snippet -->
3362 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

```

3363 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3364         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3365         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3366         elementFormDefault="qualified">
3367
3368     ...
3369
3370     <element name="binding" type="sca:Binding" abstract="true"/>
3371     <complexType name="Binding">
3372         <attribute name="uri" type="anyURI" use="optional"/>
3373         <attribute name="name" type="NCName" use="optional"/>
3374         <attribute name="requires" type="sca:listOfQNames"
3375             use="optional"/>
3376         <attribute name="policySets" type="sca:listOfQNames"
3377             use="optional"/>
3378     </complexType>
3379
3380     ...
3381
3382 </schema>

```

3383 In the following snippet is an example of how the base definition is extended to support Web
3384 service binding. The snippet shows the definition of the **binding.ws** element and the
3385 **WebServiceBinding** type contained in **sca-binding-webservice.xsd**.

```

3386
3387 <?xml version="1.0" encoding="UTF-8"?>
3388 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3389         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3390         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3391
3392     <element name="binding.ws" type="sca:WebServiceBinding"
3393         substitutionGroup="sca:binding"/>
3394     <complexType name="WebServiceBinding">
3395         <complexContent>
3396             <extension base="sca:Binding">
3397                 <attribute name="port" type="anyURI" use="required"/>
3398             </extension>
3399         </complexContent>
3400     </complexType>
3401 </schema>

```

3402 In the following snippet is an example of how the base definition can be extended by other
3403 specifications to support a new binding not defined in the SCA specifications. The snippet shows
3404 the definition of the **my-binding-extension** element and the **my-binding-extension-type** type.

```

3405 <?xml version="1.0" encoding="UTF-8"?>
3406 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

3407         targetNamespace="http://www.example.org/myextension"
3408         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3409         xmlns:tns="http://www.example.org/myextension">
3410
3411     <element name="my-binding-extension"
3412         type="tns:my-binding-extension-type"
3413         substitutionGroup="sca:binding" />
3414     <complexType name="my-binding-extension-type">
3415         <complexContent>
3416             <extension base="sca:Binding">
3417                 ...
3418             </extension>
3419         </complexContent>
3420     </complexType>
3421 </schema>
3422

```

3423 In addition to the definition for the new binding instance element, there needs to be an associated
3424 bindingType element which provides metadata about the new binding type. The pseudo schema
3425 for the bindingType element is shown in the following snippet:

```

3426 <bindingType type="xs:QName"
3427     alwaysProvides="list of intent QNames"?
3428     mayProvide = "list of intent QNames"?/>
3429

```

3430 The binding type has the following attributes:

- 3431 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is
3432 intended to be the QName of the binding element for the binding type, such as
3433 "sca:binding.ws"
- 3434 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See
3435 [the Policy Framework specification \[10\]](#) for details.
- 3436 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)
3437 [Policy Framework specification \[10\]](#) for details.

3438 11.4 Defining an Import Type

3439 The following snippet shows the base definition for the *import* element and *Import* type contained in *sca-*
3440 *core.xsd*; see appendix for complete schema.

```

3441
3442 <?xml version="1.0" encoding="UTF-8"?>
3443 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3444 IPR and other policies apply. -->
3445 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3446     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3447     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3448     elementFormDefault="qualified">
3449     ...
3450
3451     <!-- Import -->
3452     <element name="importBase" type="sca:Import" abstract="true" />
3453

```

```

3454 <complexType name="Import" abstract="true">
3455 <complexContent>
3456 <extension base="sca:CommonExtensionBase">
3457 <sequence>
3458 <any namespace="##other" processContents="lax" minOccurs="0"
3459 <maxOccurs="unbounded" />
3460 </sequence>
3461 </extension>
3462 </complexContent>
3463 </complexType>
3464
3465 <element name="import" type="sca:ImportType"
3466 substitutionGroup="sca:importBase" />
3467 <complexType name="ImportType">
3468 <complexContent>
3469 <extension base="sca:Import">
3470 <attribute name="namespace" type="string" use="required" />
3471 <attribute name="location" type="anyURI" use="required" />
3472 </extension>
3473 </complexContent>
3474 </complexType>
3475
3476 ...
3477
3478 </schema>
3479

```

3480 In the following snippet we show how the base import definition is extended to support Java imports. In
3481 the import element, the namespace is expected to be an XML namespace, an import.java element uses a
3482 Java package name instead. The snippet shows the definition of the **import.java** element and the
3483 **JavaImportType** type contained in **sca-import-java.xsd**.

```

3484
3485 <?xml version="1.0" encoding="UTF-8"?>
3486 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3487 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3488 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3489
3490 <element name="import.java" type="sca:JavaImportType"
3491 substitutionGroup="sca:importBase" />
3492 <complexType name="JavaImportType">
3493 <complexContent>
3494 <extension base="sca:Import">
3495 <attribute name="package" type="xs:String" use="required" />
3496 <attribute name="location" type="xs:AnyURI" use="optional" />
3497 </extension>
3498 </complexContent>
3499 </complexType>
3500 </schema>
3501

```

3502 In the following snippet we show an example of how the base definition can be extended by other
3503 specifications to support a new interface not defined in the SCA specifications. The snippet shows the
3504 definition of the **my-import-extension** element and the **my-import-extension-type** type.

```

3505
3506 <?xml version="1.0" encoding="UTF-8"?>
3507 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3508 targetNamespace="http://www.example.org/myextension"

```

```

3509     xmlns:sca=" http://docs.oasis-open.org/ns/opencsa/sca/200712"
3510     xmlns:tns="http://www.example.org/myextension">
3511
3512     <element name="my-import-extension"
3513         type="tns:my-import-extension-type"
3514         substitutionGroup="sca:importBase"/>
3515     <complexType name="my-import-extension-type">
3516         <complexContent>
3517             <extension base="sca:Import">
3518                 ...
3519             </extension>
3520         </complexContent>
3521     </complexType>
3522 </schema>

```

3523

3524 For a complete example using this extension point, see the definition of *import.java* in the SCA Java
3525 Common Annotations and APIs Specification [SCA-Java].

3526 11.5 Defining an Export Type

3527 The following snippet shows the base definition for the *export* element and *ExportType* type contained in
3528 *sca-core.xsd*; see appendix for complete schema.

3529

```

3530 <?xml version="1.0" encoding="UTF-8"?>
3531 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3532 IPR and other policies apply. -->
3533 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3534     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3535     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3536     elementFormDefault="qualified">
3537
3538     ...
3539     <!-- Export -->
3540     <element name="exportBase" type="sca:Export" abstract="true" />
3541     <complexType name="Export" abstract="true">
3542         <complexContent>
3543             <extension base="sca:CommonExtensionBase">
3544                 <sequence>
3545                     <any namespace="##other" processContents="lax" minOccurs="0"
3546                         maxOccurs="unbounded"/>
3547                 </sequence>
3548             </extension>
3549         </complexContent>
3550     </complexType>
3551
3552     <element name="export" type="sca:ExportType"
3553         substitutionGroup="sca:exportBase"/>
3554     <complexType name="ExportType">
3555         <complexContent>
3556             <extension base="sca:Export">
3557                 <attribute name="namespace" type="string" use="required"/>
3558             </extension>
3559         </complexContent>
3560     </complexType>
3561     ...
3562 </schema>

```

3563

3564 The following snippet shows how the base definition is extended to support Java exports. In a base
3565 *export* element, the *@namespace* attribute specifies XML namespace being exported. An *export.java*
3566 element uses a *@package* attribute to specify the Java package to be exported. The snippet shows the
3567 definition of the *export.java* element and the *JavaExport* type contained in *sca-export-java.xsd*.

3568

```
3569 <?xml version="1.0" encoding="UTF-8"?>
3570 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3571         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3572         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3573
3574     <element name="export.java" type="sca:JavaExportType"
3575             substitutionGroup="sca:exportBase"/>
3576     <complexType name="JavaExportType">
3577         <complexContent>
3578             <extension base="sca:Export">
3579                 <attribute name="package" type="xs:String" use="required"/>
3580             </extension>
3581         </complexContent>
3582     </complexType>
3583 </schema>
```

3584

3585 In the following snippet we show an example of how the base definition can be extended by other
3586 specifications to support a new interface not defined in the SCA specifications. The snippet shows the
3587 definition of the *my-export-extension* element and the *my-export-extension-type* type.

3588

```
3589 <?xml version="1.0" encoding="UTF-8"?>
3590 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3591         targetNamespace="http://www.example.org/myextension"
3592         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200712"
3593         xmlns:tns="http://www.example.org/myextension">
3594
3595     <element name="my-export-extension"
3596             type="tns:my-export-extension-type"
3597             substitutionGroup="sca:exportBase"/>
3598     <complexType name="my-export-extension-type">
3599         <complexContent>
3600             <extension base="sca:Export">
3601                 ...
3602             </extension>
3603         </complexContent>
3604     </complexType>
3605 </schema>
```

3606

3607 For a complete example using this extension point, see the definition of *export.java* in the SCA Java
3608 Common Annotations and APIs Specification [SCA-Java].

3609

3610

12 Packaging and Deployment

3611

12.1 Domains

3612
3613

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

3614
3615
3616
3617
3618
3619
3620

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

3621
3622
3623
3624

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

3625
3626
3627

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

3628

An SCA domain has the following:

3629

- A virtual domain-level composite whose components are deployed and running

3630
3631

- A set of *installed contributions* that contain implementations, interfaces and other artifacts necessary to execute components

3632
3633

- A set of logical services for manipulating the set of contributions and the virtual domain-level composite.

3634
3635

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

3636

12.2 Contributions

3637
3638
3639
3640
3641
3642
3643
3644

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

3645
3646
3647

Non-XML artifacts are also required within an SCA domain. The most obvious examples of such non-XML artifacts are Java, C++ and other programming language files necessary for component implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3648
3649
3650
3651
3652
3653

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

3654
3655

- 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 [\[ASM12001\]](#)

- 3656 • Within any contribution packaging A directory resource SHOULD exist at the root of the
3657 hierarchy named META-INF [ASM12002]
- 3658 • Within any contribution packaging a document SHOULD exist directly under the META-INF
3659 directory named sca-contribution.xml which lists the SCA Composites within the
3660 contribution that are runnable. [ASM12003]
- 3661 The same document also optionally lists namespaces of constructs that are defined within
3662 the contribution and which may be used by other contributions
3663 Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the
3664 namespaces of constructs that are needed by the contribution and which are be found
3665 elsewhere, for example in other contributions. [ASM12004] These optional elements may
3666 not be physically present in the packaging, but may be generated based on the definitions
3667 and references that are present, or they may not exist at all if there are no unresolved
3668 references.
3669
- 3670 See the section "SCA Contribution Metadata Document" for details of the format of this
3671 file.
3672

3673 To illustrate that a variety of packaging formats can be used with SCA, the following are examples
3674 of formats that might be used to package SCA artifacts and metadata (as well as other artifacts)
3675 as a contribution:

- 3676 • A filesystem directory
- 3677 • An OSGi bundle
- 3678 • A compressed directory (zip, gzip, etc)
- 3679 • A JAR file (or its variants – WAR, EAR, etc)

3680 Contributions do not contain other contributions. If the packaging format is a JAR file that
3681 contains other JAR files (or any similar nesting of other technologies), the internal files are not
3682 treated as separate SCA contributions. It is up to the implementation to determine whether the
3683 internal JAR file should be represented as a single artifact in the contribution hierarchy or whether
3684 all of the contents should be represented as separate artifacts.

3685 A goal of SCA's approach to deployment is that the contents of a contribution should not need to
3686 be modified in order to install and use the contents of the contribution in a domain.

3687

3688 12.2.1 SCA Artifact Resolution

3689 Contributions can be self-contained, in that all of the artifacts necessary to run the contents of the
3690 contribution are found within the contribution itself. However, it can also be the case that the
3691 contents of the contribution make one or many references to artifacts that are not contained
3692 within the contribution. These references can be to SCA artifacts such as composites or they can
3693 be to other artifacts such as WSDL files, XSD files or to code artifacts such as Java class files and
3694 BPEL process files. Note: This form of artifact resolution does not apply to imports of composite
3695 files, as described in Section 6.6.

3696 A contribution can use some artifact-related or packaging-related means to resolve artifact
3697 references. Examples of such mechanisms include:

- 3698 • wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema
3699 artifacts respectively
- 3700 • OSGi bundle mechanisms for resolving Java class and related resource dependencies

3701 Where present, these mechanisms MUST be used by the SCA runtime to resolve artifact
3702 dependencies. [ASM12005] **The SCA runtime MUST raise an error if an artifact cannot be resolved**
3703 **using these mechanisms, if present.** [ASM12021]

3704

Deleted: The SCA runtime MUST raise an error if an artifact cannot be resolved using these mechanisms, if present.

3705 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanism is
3706 used either where no other mechanisms are available, for example in cases where the
3707 mechanisms used by the various contributions in the same SCA Domain are different. An example
3708 of the latter case is where an OSGi Bundle is used for one contribution but where a second
3709 contribution used by the first one is not implemented using OSGi - eg the second contribution
3710 relates to a mainframe COBOL service whose interfaces are declared using a WSDL which must be
3711 accessed by the first contribution.

3712 The SCA artifact resolution is likely to be most useful for SCA domains containing heterogeneous
3713 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to
3714 work across different kinds of contribution.

3715 SCA artifact resolution works on the principle that a contribution which needs to use artifacts
3716 defined elsewhere expresses these dependencies using **import** statements in metadata belonging
3717 to the contribution. A contribution controls which artifacts it makes available to other
3718 contributions through **export** statements in metadata attached to the contribution. SCA artifact
3719 resolution is a general mechanism that can be extended for the handling of specific types of
3720 artifact. The general mechanism that is described in the following paragraphs is mainly intended
3721 for the handling of XML artifacts. Other types of artifacts, for example Java classes, use an
3722 extended version of artifact resolution that is specialized to their nature (eg. instead of
3723 "namespaces", Java uses "packages"). Descriptions of these more specialized forms of artifact
3724 resolution are contained in the SCA specifications that deal with those artifact types.

3725 Import and export statements for XML artifacts work at the level of namespaces - so that an
3726 import statement declares that artifacts from a specified namespace are found in other
3727 contributions, while an export statement makes all the artifacts from a specified namespace
3728 available to other contributions.

3729 An import declaration can simply specify the namespace to import. In this case, the locations
3730 which are searched for artifacts in that namespace are the contribution(s) in the Domain which
3731 have export declarations for the same namespace, if any. Alternatively an import declaration can
3732 specify a location from which artifacts for the namespace are obtained, in which case, that specific
3733 location is searched. There can be multiple import declarations for a given namespace. Where
3734 multiple import declarations are made for the same namespace, all the locations specified MUST
3735 be searched in lexical order. [ASM12022]

3736 For an XML namespace, artifacts can be declared in multiple locations - for example a given
3737 namespace can have a WSDL declared in one contribution and have an XSD defining XML data
3738 types in a second contribution.

3739 If the same artifact is declared in multiple locations, this is not an error. The first location as
3740 defined by lexical order is chosen. If no locations are specified no order exists and the one chosen
3741 is implementation dependent.

3742 When a contribution contains a reference to an artifact from a namespace that is declared in an import
3743 statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the
3744 SCA runtime MUST resolve artifacts in the following order:

- 3745 1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT
3746 be searched recursively in order to locate artifacts (ie only a one-level search is performed).
3747 2. from the contents of the contribution itself. [ASM12023]

3748 When a contribution uses an artifact contained in another contribution through SCA artifact
3749 resolution, if that artifact itself has dependencies on other artifacts, the SCA runtime MUST resolve
3750 these dependencies in the context of the contribution containing the artifact, not in the context of
3751 the original contribution. [ASM12024]

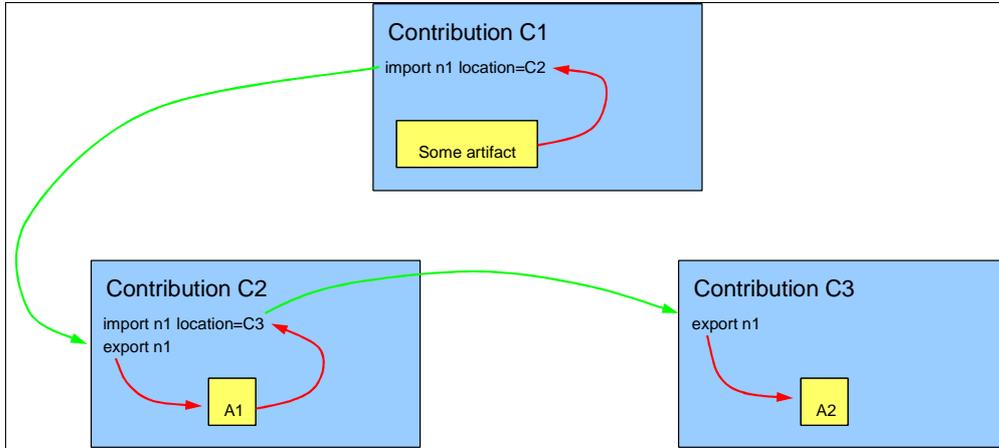
3752 For example:

- 3753 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the
3754 "n1" namespace from a second contribution "C2".
- 3755 • in contribution "C2" the artifact "A1" in the "n1" namespace references an artifact "A2"
3756 also in the "n1" namespace", which is resolved through an import of the "n1" namespace
3757 in "C2" which specifies the location "C3".

Deleted: 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.

Deleted: 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:
1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to locate artifacts (ie only a one-level search is performed).
2. from the contents of the contribution itself.

3758



3759

3760
3761
3762

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

3763
3764
3765
3766

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

3767
3768

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

3769
3770

The SCA runtime MUST raise an error if an artifact cannot be resolved by the precedence order above. [ASM12025]

3771

3772 12.2.2 SCA Contribution Metadata Document

3773
3774
3775
3776
3777
3778
3779
3780
3781
3782

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

The format of the document is:

3783
3784
3785
3786

```

<?xml version="1.0" encoding="ASCII"?>
<!-- sca-contribution pseudo-schema -->
<contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200712>

```

Figure 14: Example of SCA Artifact Resolution between Contributions

```

3787     <deployable composite="xs:QName" />*
3788     <import namespace="xs:String" location="xs:AnyURI"? />*
3789     <export namespace="xs:String" />*
3790
3791 </contribution>
3792

```

3793 **deployable element:** Identifies a composite which is a composite within the contribution that is a
3794 composite intended for potential inclusion into the virtual domain-level composite. Other
3795 composites in the contribution are not intended for inclusion but only for use by other composites.
3796 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)
3797 [Composite](#) capability and the add To Domain Level Composite capability.

3798 Attributes of the deployable element:

- 3799 • **composite (1..1)** – The QName of a composite within the contribution.

3800

3801 **Export element:** A declaration that artifacts belonging to a particular namespace are exported
3802 and are available for use within other contributions. An export declaration in a contribution
3803 specifies a namespace, all of whose definitions are considered to be exported. By default,
3804 definitions are not exported.

3805 The SCA artifact export is useful for SCA domains containing heterogeneous mixtures of
3806 contribution packagings and technologies, where artifact-related or packaging-related mechanisms
3807 are unlikely to work across different kinds of contribution.

3808 Attributes of the export element:

- 3809 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace
3810 should be the namespace URI for the exported definitions. For XML technologies that
3811 define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port
3812 types are a different symbol space from WSDL bindings), all definitions from all symbol
3813 spaces are exported.

3814 Technologies that use naming schemes other than QNames must use a different export
3815 element from the same substitution group as the the SCA <export> element. The
3816 element used identifies the technology, and can use any value for the namespace that is
3817 appropriate for that technology. For example, <export.java> can be used can be used to
3818 export java definitions, in which case the namespace is a fully qualified package name.
3819

3820

3821 **Import element:** Import declarations specify namespaces of definitions that are needed by the
3822 definitions and implementations within the contribution, but which are not present in the
3823 contribution. It is expected that in most cases import declarations will be generated based on
3824 introspection of the contents of the contribution. In this case, the import declarations would be
3825 found in the META-INF/ sca-contribution-generated.xml document.

3826 Attributes of the import element:

- 3827 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace
3828 is the namespace URI for the imported definitions. For XML technologies that define
3829 multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are
3830 a different symbol space from WSDL bindings), all definitions from all symbol spaces are
3831 imported.

3832 Technologies that use naming schemes other than QNames must use a different import
3833 element from the same substitution group as the the SCA <import> element. The
3834 element used identifies the technology, and can use any value for the namespace that is
3835 appropriate for that technology. For example, <import.java> can be used can be used to
3836 import java definitions, in which case the namespace is a fully qualified package name.
3837

3838 • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific
3839 requirements for the form of this URI, nor the means by which it is resolved. It can point
3840 to another contribution (through its URI) or it can point to some location entirely outside
3841 the SCA Domain.
3842

3843 It is expected that SCA runtimes can define implementation specific ways of resolving location
3844 information for artifact resolution between contributions. These mechanisms will however usually
3845 be limited to sets of contributions of one runtime technology and one hosting environment.

3846 In order to accommodate imports of artifacts between contributions of disparate runtime
3847 technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location
3848 specification.

3849 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are
3850 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact
3851 location specifications.

3852 The order in which the import statements are specified can play a role in this mechanism. Since
3853 definitions of one namespace can be distributed across several artifacts, multiple import
3854 declarations can be made for one namespace.
3855

3856 The location value is only a default, and dependent contributions listed in the call to
3857 installContribution can override the value if there is a conflict. However, the specific mechanism
3858 for resolving conflicts between contributions that define conflicting definitions is implementation
3859 specific.

3860 If the value of the location attribute is an SCA contribution URI, then the contribution packaging
3861 can become dependent on the deployment environment. In order to avoid such a dependency,
3862 dependent contributions should be specified only when deploying or updating contributions as
3863 specified in the section 'Operations for Contributions' below.
3864

3865 12.2.3 Contribution Packaging using ZIP

3866 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires
3867 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This
3868 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be
3869 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-
3870 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-
3871 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such
3872 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3873 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)
3874 [.ZIP file format \[12\]](#).

3875

3876 12.3 Installed Contribution

3877 As noted in the section above, the contents of a contribution do not need to be modified in order
3878 to install and use it within a domain. An *installed contribution* is a contribution with all of the
3879 associated information necessary in order to execute *deployable composites* within the
3880 contribution.

3881 An installed contribution is made up of the following things:

- 3882 • Contribution Packaging – the contribution that will be used as the starting point for
3883 resolving all references
- 3884 • Contribution base URI
- 3885 • Dependent contributions: a set of snapshots of other contributions that are used to resolve
3886 the import statements from the root composite and from other dependent contributions

- 3887
3888
- Dependent contributions might or might not be shared with other installed contributions.
- 3889
3890
- When the snapshot of any contribution is taken is implementation defined, ranging from the time the contribution is installed to the time of execution
- 3891
- Deployment-time composites.
- 3892
3893
3894
3895
3896
- These are composites that are added into an installed contribution after it has been deployed. This makes it possible to provide final configuration and access to implementations within a contribution without having to modify the contribution. These are optional, as composites that already exist within the contribution can also be used for deployment.

3897

3898
3899

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

3900
3901
3902
3903

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

3904

3905 12.3.1 Installed Artifact URIs

3906
3907
3908
3909

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

3910

3911 12.4 Operations for Contributions

3912
3913
3914
3915
3916

SCA Domains provide the following conceptual functionality associated with contributions (meaning the function might not be represented as addressable services and also meaning that equivalent functionality might be provided in other ways). The functionality is optional meaning that some SCA runtimes MAY choose not to provide the contribution functions functionality in any way. [ASM12008]

3917 12.4.1 install Contribution & update Contribution

3918
3919
3920
3921
3922

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

3923
3924
3925
3926

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

3927
3928
3929
3930
3931
3932
3933
3934

Each dependent contribution is also an installed contribution, with its own dependent contributions. By default these dependent contributions of the dependent contributions (which we will call *indirect dependent contributions*) are included as dependent contributions of the installed contribution. However, if a contribution in the dependent contribution list exports any conflicting definitions with an indirect dependent contribution, then the indirect dependent contribution is not included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions). Also, 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. [ASM12009]

3935 Note that in many cases, the dependent contribution list can be generated. In particular, if the
3936 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,
3937 then it is easy for this list to be generated by tooling.

3938 **12.4.2 add Deployment Composite & update Deployment Composite**

3939 Adds or updates a deployment composite using a supplied composite ("composite by value" – a
3940 data structure, not an existing resource in the domain) to the contribution identified by a supplied
3941 contribution URI. The added or updated deployment composite is given a relative URI that
3942 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites
3943 must run within the context of a installed contribution (any component implementations or other
3944 definitions are resolved within that contribution), this functionality makes it possible for the
3945 deployer to create a composite with final configuration and wiring decisions and add it to an
3946 installed contribution without having to modify the contents of the root contribution.

3947 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).
3948 It is then possible for those to be given component names by a (possibly generated) composite
3949 that is added into the installed contribution, without having to modify the packaging.

3950 **12.4.3 remove Contribution**

3951 Removes the deployed contribution identified by a supplied contribution URI.

3952

3953 **12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts**

3954

3955 For certain types of artifact, there are existing and commonly used mechanisms for referencing a
3956 specific concrete location where the artifact can be resolved.

3957 Examples of these mechanisms include:

- 3958 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the
3959 place holding the WSDL itself.
- 3960 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a
3961 URI where the XSD is found.

3962 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does
3963 not have to be dereferenced.

3964 SCA permits the use of these mechanisms. Where present, non-SCA artifact resolution
3965 mechanisms **MUST** be used by the SCA runtime in precedence to the SCA mechanisms.
3966 **[ASM12010]** However, use of these mechanisms is discouraged because tying assemblies to
3967 addresses in this way makes the assemblies less flexible and prone to errors when changes are
3968 made to the overall SCA Domain.

3969 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to
3970 find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the
3971 SCA runtime **MUST** raise an error and **MUST NOT** attempt to use SCA resolution mechanisms as an
3972 alternative. **[ASM12011]**

3973

3974 **12.6 Domain-Level Composite**

3975 The domain-level composite is a virtual composite, in that it is not defined by a composite
3976 definition document. Rather, it is built up and modified through operations on the domain.
3977 However, in other respects it is very much like a composite, since it contains components, wires,
3978 services and references.

3979

3980 The value of @autowire for the logical domain composite MUST be autowire="false". [ASM12012]

3981

3982 For components at the Domain level, with References for which @autowire="true" applies, the
3983 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

3984 1) The SCA runtime MAY disallow deployment of any components with autowire References. In
3985 this case, the SCA runtime MUST generate an exception at the point where the component is
3986 deployed.

3987 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component
3988 is deployed and not update those targets when later deployment actions occur.

3989 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later
3990 deployment actions occur resulting in updated reference targets which match the new Domain
3991 configuration. How the new configuration of the reference takes place is described by the relevant
3992 client and implementation specifications.

3993 [ASM12013]

3994 The abstract domain-level functionality for modifying the domain-level composite is as follows,
3995 although a runtime may supply equivalent functionality in a different form:

3996 12.6.1 add To Domain-Level Composite

3997 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.
3998 The supplied composite URI must refer to a composite within a installed contribution. The
3999 composite's installed contribution determines how the composite's artifacts are resolved (directly
4000 and indirectly). The supplied composite is added to the domain composite with semantics that
4001 correspond to the domain-level composite having an <include> statement that references the
4002 supplied composite. All of the composite's components become *top-level* components and the
4003 services become externally visible services (eg. they would be present in a WSDL description of
4004 the domain).

4005 12.6.2 remove From Domain-Level Composite

4006 Removes from the Domain Level composite the elements corresponding to the composite
4007 identified by a supplied composite URI. This means that the removal of the components, wires,
4008 services and references originally added to the domain level composite by the identified
4009 composite.

4010 12.6.3 get Domain-Level Composite

4011 Returns a <composite> definition that has an <include> line for each composite that had been
4012 added to the domain level composite. It is important to note that, in dereferencing the included
4013 composites, any referenced artifacts must be resolved in terms of that installed composite.

4014 12.6.4 get QName Definition

4015 In order to make sense of the domain-level composite (as returned by get Domain-Level
4016 Composite), it must be possible to get the definitions for named artifacts in the included
4017 composites. This functionality takes the supplied URI of an installed contribution (which provides
4018 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as
4019 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for
4020 that definition type.

4021 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities
4022 should exist in some form, but not necessarily as a service operation with exactly this signature.

Formatted: Body Text,Body
Text Char,Body Text Char1
Char1,Body Text Char Char
Char1,Body Text Char1 Char1
Char Char,Body Text Char
Char Char1 Char Char,Body
Text Char1 Char1 Char Char
Char Char,Body Text Char
Char Char1 Char Char Char
Char,Body Text Char1

Deleted: 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:¶
1) The SCA runtime MAY
disallow deployment of any
components with autowire
References. In this case,
the SCA runtime MUST
generate an exception at
the point where the
component is deployed.¶
2) The SCA runtime MAY
evaluate the target(s) for
the reference at the time
that the component is
deployed and not update
those targets when later
deployment actions occur.
3) The SCA runtime MAY
re-evaluate 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 new
configuration of the
reference takes place is
described by the relevant
client and implementation
specifications.

4023 12.7 Dynamic Behaviour of Wires in the SCA Domain

4024 For components with references which are at the Domain level, there is the potential for dynamic
4025 behaviour when the wires for a component reference change (this can only apply to component
4026 references at the Domain level and not to components within composites used as implementations):

4027 The configuration of the wires for a component reference of a component at the Domain level can change
4028 by means of deployment actions:

- 4029 1. <wire/> elements can be added, removed or replaced by deployment actions
- 4030 2. Components can be updated by deployment actions (ie this may change the component reference
4031 configuration)
- 4032 3. Components which are the targets of reference wires can be updated or removed
- 4033 4. Components can be added that are potential targets for references which are marked with
4034 @autowire=true

4035
4036 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose
4037 references are affected by those deployment actions MAY have their references updated by the SCA
4038 runtime dynamically without the need to stop and start those components. [ASM12014]

4039 Where components are updated by deployment actions (their configuration is changed in some way,
4040 which may include changing the wires of component references), the new configuration MUST apply to all
4041 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY
4042 choose to maintain existing instances with the old configuration of components updated by deployment
4043 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.
4044 [ASM12016]

4045 Where a component that is the target of a wire is removed, without the wire being changed, then future
4046 invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is
4047 the result of the autowire process, the SCA runtime MUST:

- 4048 • either cause future invocation of the target component's services to fail with a
4049 ServiceUnavailable fault
- 4050 • or alternatively, if an alternative target component is available that satisfies the autowire
4051 process, update the reference of the source component. [ASM12017]

4052 Where a component that is the target of a wire is updated, future invocations of that reference SHOULD
4053 use the updated component. [ASM12018] Where an existing domain level component is updated, an
4054 SCA runtime MAY maintain a copy of a component offering a conversational service until all existing
4055 conversations complete - alternatively all existing conversations MAY be terminated. [ASM12019]

4056 Where a component is added to the domain that is a potential target for a domain level component
4057 reference where that reference is marked as @autowire=true, the SCA runtime MUST:

- 4058 - either update the references for the source component once the new component is running.
- 4059 - or alternatively, defer the updating of the references of the source component until the source
4060 component is stopped and restarted. [ASM12020]

4062 12.8 Dynamic Behaviour of Component Property Values

4063 For a domain level component with a Property whose value is obtained from a Domain-level Property
4064 through the use of the @source attribute, if the domain level property is updated by means of deployment
4065 actions, the SCA runtime MUST

- 4066 - either update the property value of the domain level component. once the update of the domain
4067 property is complete
- 4068 - or alternative defer the updating of the component property value until the compoennt is stopped
4069 and restarted

Formatted: Bullets and
Numbering

Deleted: 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:¶
<#>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

Formatted: Bullets and
Numbering

Deleted: 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:¶
<#>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.

4071 **13 Conformance**

4072 The XML schema available at the namespace URI, defined by this specification, is considered to be
4073 authoritative and takes precedence over the XML Schema defined in the appendix of this document.

4074 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema..
4075 [\[ASM13001\]](#)

4076

4077 A. XML Schemas

4078 A.1 sca.xsd

```
4079
4080 <?xml version="1.0" encoding="UTF-8"?>
4081 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4082 IPR and other policies apply. -->
4083 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4084   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4085   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
4086
4087   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4088
4089   <include schemaLocation="sca-interface-java-1.1-schema-200803.xsd"/>
4090   <include schemaLocation="sca-interface-wsdl-1.1-schema-200803.xsd"/>
4091   <include schemaLocation="sca-interface-cpp-1.1-schema-200803.xsd"/>
4092   <include schemaLocation="sca-interface-c-1.1-schema-200803.xsd"/>
4093
4094   <include schemaLocation="sca-implementation-java-1.1-schema-200803.xsd"/>
4095   <include schemaLocation=
4096     "sca-implementation-composite-1.1-schema-200803.xsd"/>
4097   <include schemaLocation="sca-implementation-cpp-1.1-schema-200803.xsd"/>
4098   <include schemaLocation="sca-implementation-c-1.1-schema-200803.xsd"/>
4099   <include schemaLocation="sca-implementation-bpel-1.1-schema-200803.xsd"/>
4100
4101   <include schemaLocation="sca-binding-webservice-1.1-schema-200803.xsd"/>
4102   <include schemaLocation="sca-binding-jms-1.1-schema-200803.xsd"/>
4103   <include schemaLocation="sca-binding-sca-1.1-schema-200803.xsd"/>
4104
4105   <include schemaLocation="sca-definitions-1.1-schema-200803.xsd"/>
4106   <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4107
4108   <include schemaLocation="sca-contribution-1.1-schema-200803.xsd"/>
4109
4110 </schema>
```

4111 A.2 sca-core.xsd

```
4112
4113 <?xml version="1.0" encoding="UTF-8"?>
4114 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4115 IPR and other policies apply. -->
4116 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4117   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4118   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4119   elementFormDefault="qualified">
4120
4121   <import namespace="http://www.w3.org/XML/1998/namespace"
4122     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
4123
4124   <!-- Common extension base for SCA definitions -->
4125   <complexType name="CommonExtensionBase">
4126     <sequence>
```

```

4127         <element ref="sca:documentation" minOccurs="0"
4128             maxOccurs="unbounded" />
4129     </sequence>
4130     <anyAttribute namespace="##other" processContents="lax" />
4131 </complexType>
4132
4133 <element name="documentation" type="sca:Documentation" />
4134 <complexType name="Documentation" mixed="true">
4135     <sequence>
4136         <any namespace="##other" processContents="lax" minOccurs="0"
4137             maxOccurs="unbounded" />
4138     </sequence>
4139     <attribute ref="xml:lang" />
4140 </complexType>
4141
4142 <!-- Component Type -->
4143 <element name="componentType" type="sca:ComponentType" />
4144 <complexType name="ComponentType">
4145     <complexContent>
4146         <extension base="sca:CommonExtensionBase">
4147             <sequence>
4148                 <element ref="sca:implementation" minOccurs="0" />
4149                 <choice minOccurs="0" maxOccurs="unbounded">
4150                     <element name="service" type="sca:ComponentService" />
4151                     <element name="reference"
4152                         type="sca:ComponentTypeReference" />
4153                     <element name="property" type="sca:Property" />
4154                 </choice>
4155                 <any namespace="##other" processContents="lax" minOccurs="0"
4156                     maxOccurs="unbounded" />
4157             </sequence>
4158             <attribute name="constrainingType" type="QName" use="optional" />
4159         </extension>
4160     </complexContent>
4161 </complexType>
4162
4163 <!-- Composite -->
4164 <element name="composite" type="sca:Composite" />
4165 <complexType name="Composite">
4166     <complexContent>
4167         <extension base="sca:CommonExtensionBase">
4168             <sequence>
4169                 <element name="include" type="anyURI" minOccurs="0"
4170                     maxOccurs="unbounded" />
4171                 <choice minOccurs="0" maxOccurs="unbounded">
4172                     <element name="service" type="sca:Service" />
4173                     <element name="property" type="sca:Property" />
4174                     <element name="component" type="sca:Component" />
4175                     <element name="reference" type="sca:Reference" />
4176                     <element name="wire" type="sca:Wire" />
4177                 </choice>
4178                 <any namespace="##other" processContents="lax" minOccurs="0"
4179                     maxOccurs="unbounded" />
4180             </sequence>
4181             <attribute name="name" type="NCName" use="required" />
4182             <attribute name="targetNamespace" type="anyURI" use="required" />
4183             <attribute name="local" type="boolean" use="optional"
4184                 default="false" />

```

```

4185         <attribute name="autowire" type="boolean" use="optional"
4186             default="false"/>
4187         <attribute name="constrainingType" type="QName" use="optional"/>
4188         <attribute name="requires" type="sca:listOfQNames"
4189             use="optional"/>
4190         <attribute name="policySets" type="sca:listOfQNames"
4191             use="optional"/>
4192     </extension>
4193 </complexContent>
4194 </complexType>
4195
4196 <!-- Contract base type for Service, Reference -->
4197 <complexType name="Contract" abstract="true">
4198     <complexContent>
4199         <extension base="sca:CommonExtensionBase">
4200             <sequence>
4201                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4202                 <element name="operation" type="sca:Operation" minOccurs="0"
4203                     maxOccurs="unbounded" />
4204                 <element ref="sca:binding" minOccurs="0"
4205                     maxOccurs="unbounded"/>
4206                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4207                 <any namespace="##other" processContents="lax" minOccurs="0"
4208                     maxOccurs="unbounded" />
4209             </sequence>
4210             <attribute name="name" type="NCName" use="required" />
4211             <attribute name="requires" type="sca:listOfQNames"
4212                 use="optional"/>
4213             <attribute name="policySets" type="sca:listOfQNames"
4214                 use="optional"/>
4215         </extension>
4216     </complexContent>
4217 </complexType>
4218
4219 <!-- Service -->
4220 <complexType name="Service">
4221     <complexContent>
4222         <extension base="sca:Contract">
4223             <attribute name="promote" type="anyURI" use="required"/>
4224         </extension>
4225     </complexContent>
4226 </complexType>
4227
4228 <!-- Interface -->
4229 <element name="interface" type="sca:Interface" abstract="true"/>
4230 <complexType name="Interface" abstract="true">
4231     <complexContent>
4232         <extension base="sca:CommonExtensionBase"/>
4233     </complexContent>
4234 </complexType>
4235
4236 <!-- Reference -->
4237 <complexType name="Reference">
4238     <complexContent>
4239         <extension base="sca:Contract">
4240             <attribute name="autowire" type="boolean" use="optional"/>
4241             <attribute name="target" type="sca:listOfAnyURIs"
4242                 use="optional"/>

```

```

4243         <attribute name="wiredByImpl" type="boolean" use="optional"
4244             default="false"/>
4245         <attribute name="multiplicity" type="sca:Multiplicity"
4246             use="optional" default="1..1"/>
4247         <attribute name="promote" type="sca:listOfAnyURIs"
4248             use="required"/>
4249     </extension>
4250 </complexContent>
4251 </complexType>
4252
4253 <!-- Property -->
4254 <complexType name="SCAPropertyBase" mixed="true">
4255     <sequence>
4256         <any namespace="##any" processContents="lax" minOccurs="0"/>
4257         <!-- NOT an extension point; This any exists to accept
4258             the element-based or complex type property
4259             i.e. no element-based extension point under "sca:property" -->
4260     </sequence>
4261     <!-- mixed="true" to handle simple type -->
4262     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4263     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4264 </complexType>
4265
4266 <complexType name="Property" mixed="true">
4267     <complexContent mixed="true">
4268         <extension base="sca:SCAPropertyBase">
4269             <attribute name="name" type="NCName" use="required"/>
4270             <attribute name="type" type="QName" use="optional"/>
4271             <attribute name="element" type="QName" use="optional"/>
4272             <attribute name="many" type="boolean" use="optional"
4273                 default="false"/>
4274             <attribute name="mustSupply" type="boolean" use="optional"
4275                 default="false"/>
4276             <anyAttribute namespace="##any" processContents="lax"/>
4277         </extension>
4278         <!-- extension defines the place to hold default value -->
4279         <!-- an extension point ; attribute-based only -->
4280     </complexContent>
4281 </complexType>
4282
4283 <complexType name="PropertyValue" mixed="true">
4284     <complexContent mixed="true">
4285         <extension base="sca:SCAPropertyBase">
4286             <attribute name="name" type="NCName" use="required"/>
4287             <attribute name="type" type="QName" use="optional"/>
4288             <attribute name="element" type="QName" use="optional"/>
4289             <attribute name="many" type="boolean" use="optional"
4290                 default="false"/>
4291             <attribute name="source" type="string" use="optional"/>
4292             <attribute name="file" type="anyURI" use="optional"/>
4293             <anyAttribute namespace="##any" processContents="lax"/>
4294         </extension>
4295         <!-- an extension point ; attribute-based only -->
4296     </complexContent>
4297 </complexType>
4298
4299 <!-- Binding -->
4300 <element name="binding" type="sca:Binding" abstract="true"/>

```

```

4301 <complexType name="Binding" abstract="true">
4302   <complexContent>
4303     <extension base="sca:CommonExtensionBase">
4304       <sequence>
4305         <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
4306         <element ref="sca:operationSelector"
4307           minOccurs="0" maxOccurs="1" />
4308         <element name="operation" type="sca:Operation" minOccurs="0"
4309           maxOccurs="unbounded" />
4310       </sequence>
4311       <attribute name="uri" type="anyURI" use="optional" />
4312       <attribute name="name" type="NCName" use="optional" />
4313       <attribute name="requires" type="sca:listOfQNames"
4314         use="optional" />
4315       <attribute name="policySets" type="sca:listOfQNames"
4316         use="optional" />
4317     </extension>
4318   </complexContent>
4319 </complexType>
4320
4321 <!-- Binding Type -->
4322 <element name="bindingType" type="sca:BindingType" />
4323 <complexType name="BindingType">
4324   <complexContent>
4325     <extension base="sca:CommonExtensionBase">
4326       <sequence>
4327         <any namespace="##other" processContents="lax" minOccurs="0"
4328           maxOccurs="unbounded" />
4329       </sequence>
4330       <attribute name="type" type="QName" use="required" />
4331       <attribute name="alwaysProvides" type="sca:listOfQNames"
4332         use="optional" />
4333       <attribute name="mayProvide" type="sca:listOfQNames"
4334         use="optional" />
4335     </extension>
4336   </complexContent>
4337 </complexType>
4338
4339 <!-- WireFormat Type -->
4340 <element name="wireFormat" type="sca:WireFormatType" />
4341 <complexType name="WireFormatType" abstract="true">
4342   <sequence>
4343     <any namespace="##other" processContents="lax" minOccurs="0"
4344       maxOccurs="unbounded" />
4345   </sequence>
4346   <anyAttribute namespace="##other" processContents="lax" />
4347 </complexType>
4348
4349 <!-- OperationSelector Type -->
4350 <element name="operationSelector" type="sca:OperationSelectorType" />
4351 <complexType name="OperationSelectorType" abstract="true">
4352   <sequence>
4353     <any namespace="##other" processContents="lax" minOccurs="0"
4354       maxOccurs="unbounded" />
4355   </sequence>
4356   <anyAttribute namespace="##other" processContents="lax" />
4357 </complexType>
4358 <!-- Callback -->

```

```

4359 <element name="callback" type="sca:Callback"/>
4360 <complexType name="Callback">
4361   <complexContent>
4362     <extension base="sca:CommonExtensionBase">
4363       <choice minOccurs="0" maxOccurs="unbounded">
4364         <element ref="sca:binding"/>
4365         <any namespace="##other" processContents="lax"/>
4366       </choice>
4367       <attribute name="requires" type="sca:listOfQNames"
4368         use="optional"/>
4369       <attribute name="policySets" type="sca:listOfQNames"
4370         use="optional"/>
4371     </extension>
4372   </complexContent>
4373 </complexType>
4374
4375 <!-- Component -->
4376 <complexType name="Component">
4377   <complexContent>
4378     <extension base="sca:CommonExtensionBase">
4379       <sequence>
4380         <element ref="sca:implementation" minOccurs="0"/>
4381         <choice minOccurs="0" maxOccurs="unbounded">
4382           <element name="service" type="sca:ComponentService"/>
4383           <element name="reference" type="sca:ComponentReference"/>
4384           <element name="property" type="sca:PropertyValue"/>
4385         </choice>
4386         <any namespace="##other" processContents="lax" minOccurs="0"
4387           maxOccurs="unbounded"/>
4388       </sequence>
4389       <attribute name="name" type="NCName" use="required"/>
4390       <attribute name="autowire" type="boolean" use="optional"/>
4391       <attribute name="constrainingType" type="QName" use="optional"/>
4392       <attribute name="requires" type="sca:listOfQNames"
4393         use="optional"/>
4394       <attribute name="policySets" type="sca:listOfQNames"
4395         use="optional"/>
4396     </extension>
4397   </complexContent>
4398 </complexType>
4399
4400 <!-- Component Service -->
4401 <complexType name="ComponentService">
4402   <complexContent>
4403     <extension base="sca:Contract">
4404     </extension>
4405   </complexContent>
4406 </complexType>
4407
4408 <!-- Component Reference -->
4409 <complexType name="ComponentReference">
4410   <complexContent>
4411     <extension base="sca:Contract">
4412       <attribute name="autowire" type="boolean" use="optional"/>
4413       <attribute name="target" type="sca:listOfAnyURIs"
4414         use="optional"/>
4415       <attribute name="wiredByImpl" type="boolean" use="optional"
4416         default="false"/>

```

```

4417         <attribute name="multiplicity" type="sca:Multiplicity"
4418             use="optional" default="1..1"/>
4419     </extension>
4420 </complexContent>
4421 </complexType>
4422
4423 <!-- Component Type Reference -->
4424 <complexType name="ComponentTypeReference">
4425     <complexContent>
4426         <restriction base="sca:ComponentReference">
4427             <sequence>
4428                 <element ref="sca:documentation" minOccurs="0"
4429                     maxOccurs="unbounded"/>
4430                 <element ref="sca:interface" minOccurs="0"/>
4431                 <element name="operation" type="sca:Operation" minOccurs="0"
4432                     maxOccurs="unbounded"/>
4433                 <element ref="sca:binding" minOccurs="0"
4434                     maxOccurs="unbounded"/>
4435                 <element ref="sca:callback" minOccurs="0"/>
4436                 <any namespace="##other" processContents="lax" minOccurs="0"
4437                     maxOccurs="unbounded"/>
4438             </sequence>
4439             <attribute name="name" type="NCName" use="required"/>
4440             <attribute name="autowire" type="boolean" use="optional"/>
4441             <attribute name="wiredByImpl" type="boolean" use="optional"
4442                 default="false"/>
4443             <attribute name="multiplicity" type="sca:Multiplicity"
4444                 use="optional" default="1..1"/>
4445             <attribute name="requires" type="sca:listOfQNames"
4446                 use="optional"/>
4447             <attribute name="policySets" type="sca:listOfQNames"
4448                 use="optional"/>
4449             <anyAttribute namespace="##other" processContents="lax"/>
4450         </restriction>
4451     </complexContent>
4452 </complexType>
4453
4454 <!-- Implementation -->
4455 <element name="implementation" type="sca:Implementation" abstract="true"/>
4456 <complexType name="Implementation" abstract="true">
4457     <complexContent>
4458         <extension base="sca:CommonExtensionBase">
4459             <attribute name="requires" type="sca:listOfQNames"
4460                 use="optional"/>
4461             <attribute name="policySets" type="sca:listOfQNames"
4462                 use="optional"/>
4463         </extension>
4464     </complexContent>
4465 </complexType>
4466
4467 <!-- Implementation Type -->
4468 <element name="implementationType" type="sca:ImplementationType"/>
4469 <complexType name="ImplementationType">
4470     <complexContent>
4471         <extension base="sca:CommonExtensionBase">
4472             <sequence>
4473                 <any namespace="##other" processContents="lax" minOccurs="0"
4474                     maxOccurs="unbounded"/>

```

```

4475         </sequence>
4476         <attribute name="type" type="QName" use="required"/>
4477         <attribute name="alwaysProvides" type="sca:listOfQNames"
4478             use="optional"/>
4479         <attribute name="mayProvide" type="sca:listOfQNames"
4480             use="optional"/>
4481     </extension>
4482 </complexContent>
4483 </complexType>
4484
4485 <!-- Wire -->
4486 <complexType name="Wire">
4487     <complexContent>
4488         <extension base="sca:CommonExtensionBase">
4489             <sequence>
4490                 <any namespace="##other" processContents="lax" minOccurs="0"
4491                     maxOccurs="unbounded"/>
4492             </sequence>
4493             <attribute name="source" type="anyURI" use="required"/>
4494             <attribute name="target" type="anyURI" use="required"/>
4495         </extension>
4496     </complexContent>
4497 </complexType>
4498
4499 <!-- Include -->
4500 <element name="include" type="sca:Include"/>
4501 <complexType name="Include">
4502     <complexContent>
4503         <extension base="sca:CommonExtensionBase">
4504             <attribute name="name" type="QName"/>
4505         </extension>
4506     </complexContent>
4507 </complexType>
4508
4509 <!-- Operation -->
4510 <complexType name="Operation">
4511     <complexContent>
4512         <extension base="sca:CommonExtensionBase">
4513             <attribute name="name" type="NCName" use="required"/>
4514             <attribute name="requires" type="sca:listOfQNames"
4515                 use="optional"/>
4516             <attribute name="policySets" type="sca:listOfQNames"
4517                 use="optional"/>
4518         </extension>
4519     </complexContent>
4520 </complexType>
4521
4522 <!-- Constraining Type -->
4523 <element name="constrainingType" type="sca:ConstrainingType"/>
4524 <complexType name="ConstrainingType">
4525     <complexContent>
4526         <extension base="sca:CommonExtensionBase">
4527             <sequence>
4528                 <choice minOccurs="0" maxOccurs="unbounded">
4529                     <element name="service" type="sca:ComponentService"/>
4530                     <element name="reference" type="sca:ComponentReference"/>
4531                     <element name="property" type="sca:Property"/>
4532                 </choice>

```

```

4533         <any namespace="##other" processContents="lax" minOccurs="0"
4534             maxOccurs="unbounded" />
4535     </sequence>
4536     <attribute name="name" type="NCName" use="required" />
4537     <attribute name="targetNamespace" type="anyURI" />
4538     <attribute name="requires" type="sca:listOfQNames"
4539         use="optional" />
4540 </extension>
4541 </complexContent>
4542 </complexType>
4543
4544 <!-- Intents within WSDL documents -->
4545 <attribute name="requires" type="sca:listOfQNames" />
4546
4547 <!-- Marker for operations ending a conversation -->
4548 <attribute name="endsConversation" type="boolean" default="false" />
4549
4550 <!-- Global attribute definition for @callback to mark a WSDL port type
4551     as having a callback interface defined in terms of a second port
4552     type. -->
4553 <attribute name="callback" type="anyURI" />
4554
4555 <!-- Miscellaneous simple type definitions -->
4556 <simpleType name="Multiplicity">
4557     <restriction base="string">
4558         <enumeration value="0..1" />
4559         <enumeration value="1..1" />
4560         <enumeration value="0..n" />
4561         <enumeration value="1..n" />
4562     </restriction>
4563 </simpleType>
4564
4565 <simpleType name="OverrideOptions">
4566     <restriction base="string">
4567         <enumeration value="no" />
4568         <enumeration value="may" />
4569         <enumeration value="must" />
4570     </restriction>
4571 </simpleType>
4572
4573 <simpleType name="listOfQNames">
4574     <list itemType="QName" />
4575 </simpleType>
4576
4577 <simpleType name="listOfAnyURIs">
4578     <list itemType="anyURI" />
4579 </simpleType>
4580
4581 </schema>
4582

```

4583 **A.3 sca-binding-sca.xsd**

```

4584
4585 <?xml version="1.0" encoding="UTF-8"?>
4586 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4587 IPR and other policies apply. -->
4588 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

4589     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4590     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4591     elementFormDefault="qualified">
4592
4593     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4594
4595     <!-- SCA Binding -->
4596     <element name="binding.sca" type="sca:SCABinding"
4597         substitutionGroup="sca:binding"/>
4598     <complexType name="SCABinding">
4599         <complexContent>
4600             <extension base="sca:Binding"/>
4601         </complexContent>
4602     </complexType>
4603 </schema>
4604
4605

```

4606 **A.4 sca-interface-java.xsd**

```

4607
4608 <?xml version="1.0" encoding="UTF-8"?>
4609 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4610 IPR and other policies apply. -->
4611 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4612     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4613     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4614     elementFormDefault="qualified">
4615
4616     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4617
4618     <!-- Java Interface -->
4619     <element name="interface.java" type="sca:JavaInterface"
4620         substitutionGroup="sca:interface"/>
4621     <complexType name="JavaInterface">
4622         <complexContent>
4623             <extension base="sca:Interface">
4624                 <sequence>
4625                     <any namespace="##other" processContents="lax" minOccurs="0"
4626                         maxOccurs="unbounded"/>
4627                 </sequence>
4628                 <attribute name="interface" type="NCName" use="required"/>
4629                 <attribute name="callbackInterface" type="NCName"
4630                     use="optional"/>
4631                 <anyAttribute namespace="##any" processContents="lax"/>
4632             </extension>
4633         </complexContent>
4634     </complexType>
4635 </schema>
4636
4637
4638

```

4639 **A.5 sca-interface-wsdl.xsd**

4640

```

4641 <?xml version="1.0" encoding="UTF-8"?>
4642 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4643 IPR and other policies apply. -->
4644 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4645   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4646   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4647   elementFormDefault="qualified">
4648
4649   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4650
4651   <!-- WSDL Interface -->
4652   <element name="interface.wsdl" type="sca:WSDLPortType"
4653     substitutionGroup="sca:interface"/>
4654   <complexType name="WSDLPortType">
4655     <complexContent>
4656       <extension base="sca:Interface">
4657         <sequence>
4658           <any namespace="##other" processContents="lax" minOccurs="0"
4659             maxOccurs="unbounded"/>
4660         </sequence>
4661         <attribute name="interface" type="anyURI" use="required"/>
4662         <attribute name="callbackInterface" type="anyURI"
4663           use="optional"/>
4664         <anyAttribute namespace="##any" processContents="lax"/>
4665       </extension>
4666     </complexContent>
4667   </complexType>
4668
4669 </schema>
4670
4671

```

4672 **A.6 sca-implementation-java.xsd**

```

4673
4674 <?xml version="1.0" encoding="UTF-8"?>
4675 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4676 IPR and other policies apply. -->
4677 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4678   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4679   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4680   elementFormDefault="qualified">
4681
4682   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4683
4684   <!-- Java Implementation -->
4685   <element name="implementation.java" type="sca:JavaImplementation"
4686     substitutionGroup="sca:implementation"/>
4687   <complexType name="JavaImplementation">
4688     <complexContent>
4689       <extension base="sca:Implementation">
4690         <sequence>
4691           <any namespace="##other" processContents="lax" minOccurs="0"
4692             maxOccurs="unbounded"/>
4693         </sequence>
4694         <attribute name="class" type="NCName" use="required"/>
4695         <anyAttribute namespace="##any" processContents="lax"/>

```

```
4696         </extension>
4697     </complexContent>
4698 </complexType>
4699
4700 </schema>
```

4701 **A.7 sca-implementation-composite.xsd**

```
4702
4703 <?xml version="1.0" encoding="UTF-8"?>
4704 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4705 IPR and other policies apply. -->
4706 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4707     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4708     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4709     elementFormDefault="qualified">
4710
4711     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4712
4713     <!-- Composite Implementation -->
4714     <element name="implementation.composite" type="sca:SCAImplementation"
4715         substitutionGroup="sca:implementation"/>
4716     <complexType name="SCAImplementation">
4717         <complexContent>
4718             <extension base="sca:Implementation">
4719                 <sequence>
4720                     <any namespace="##other" processContents="lax" minOccurs="0"
4721                         maxOccurs="unbounded"/>
4722                 </sequence>
4723                 <attribute name="name" type="QName" use="required"/>
4724             </extension>
4725         </complexContent>
4726     </complexType>
4727 </schema>
4728
4729
```

4730 **A.8 sca-definitions.xsd**

```
4731
4732 <?xml version="1.0" encoding="UTF-8"?>
4733 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4734 IPR and other policies apply. -->
4735 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4736     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4737     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4738     elementFormDefault="qualified">
4739
4740     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4741     <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4742
4743     <!-- Definitions -->
4744     <element name="definitions" type="sca:tDefinitions"/>
4745     <complexType name="tDefinitions">
4746         <complexContent>
4747             <extension base="sca:CommonExtensionBase">
4748                 <choice minOccurs="0" maxOccurs="unbounded">
```

```

4749         <element ref="sca:intent"/>
4750     <element ref="sca:policySet"/>
4751 <element ref="sca:binding"/>
4752 <element ref="sca:bindingType"/>
4753 <element ref="sca:implementationType"/>
4754     <any namespace="##other" processContents="lax" minOccurs="0"
4755         maxOccurs="unbounded"/>
4756 </choice>
4757 </extension>
4758 </complexContent>
4759 </complexType>
4760
4761 </schema>
4762
4763

```

4764 **A.9 sca-binding-webservice.xsd**

4765 Is described in [the SCA Web Services Binding specification \[9\]](#)

4766 **A.10 sca-binding-jms.xsd**

4767 Is described in [the SCA JMS Binding specification \[11\]](#)

4768 **A.11 sca-policy.xsd**

4769 Is described in [the SCA Policy Framework specification \[10\]](#)

4770

4771 **A.12 sca-contribution.xsd**

4772

```

4773 <?xml version="1.0" encoding="UTF-8"?>
4774 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4775 IPR and other policies apply. -->
4776 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4777     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4778     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4779     elementFormDefault="qualified">
4780
4781     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4782
4783     <!-- Contribution -->
4784     <element name="contribution" type="sca:ContributionType"/>
4785     <complexType name="ContributionType">
4786         <complexContent>
4787             <extension base="sca:CommonExtensionBase">
4788                 <sequence>
4789                     <element name="deployable" type="sca:DeployableType"
4790                         maxOccurs="unbounded"/>
4791                     <element name="import" type="sca:ImportType" minOccurs="0"
4792                         maxOccurs="unbounded"/>
4793                     <element name="export" type="sca:ExportType" minOccurs="0"
4794                         maxOccurs="unbounded"/>
4795                     <any namespace="##other" processContents="lax" minOccurs="0"
4796                         maxOccurs="unbounded"/>
4797                 </sequence>

```

```

4798         </extension>
4799     </complexContent>
4800 </complexType>
4801
4802 <!-- Deployable -->
4803 <complexType name="DeployableType">
4804     <complexContent>
4805         <extension base="sca:CommonExtensionBase">
4806             <sequence>
4807                 <any namespace="##other" processContents="lax" minOccurs="0"
4808                     maxOccurs="unbounded" />
4809             </sequence>
4810             <attribute name="composite" type="QName" use="required" />
4811         </extension>
4812     </complexContent>
4813 </complexType>
4814
4815 <!-- Import -->
4816 <element name="importBase" type="sca:Import" abstract="true" />
4817 <complexType name="Import" abstract="true">
4818     <complexContent>
4819         <extension base="sca:CommonExtensionBase">
4820             <sequence>
4821                 <any namespace="##other" processContents="lax" minOccurs="0"
4822                     maxOccurs="unbounded" />
4823             </sequence>
4824         </extension>
4825     </complexContent>
4826 </complexType>
4827
4828 <element name="import" type="sca:ImportType" />
4829 <complexType name="ImportType">
4830     <complexContent>
4831         <extension base="sca:Import">
4832             <attribute name="namespace" type="string" use="required" />
4833             <attribute name="location" type="anyURI" use="optional" />
4834         </extension>
4835     </complexContent>
4836 </complexType>
4837
4838 <!-- Export -->
4839 <element name="exportBase" type="sca:Export" abstract="true" />
4840 <complexType name="Export" abstract="true">
4841     <complexContent>
4842         <extension base="sca:CommonExtensionBase">
4843             <sequence>
4844                 <any namespace="##other" processContents="lax" minOccurs="0"
4845                     maxOccurs="unbounded" />
4846             </sequence>
4847         </extension>
4848     </complexContent>
4849 </complexType>
4850
4851 <element name="export" type="sca:ExportType" />
4852 <complexType name="ExportType">
4853     <complexContent>
4854         <extension base="sca:Export">
4855             <attribute name="namespace" type="string" use="required" />

```

```
4856         </extension>
4857     </complexContent>
4858 </complexType>
4859
4860 </schema>
4861
4862
```

4863

B. SCA Concepts

4864

B.1 Binding

4865

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 should use to call the service.

4866

4867

4868

SCA supports multiple different types of bindings. Examples include **SCA service, Web service, stateless session EJB, data base stored procedure, EIS service**. SCA provides an extensibility mechanism by which an SCA runtime can add support for additional binding types.

4869

4870

4871

4872

B.2 Component

4873

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 run-time, vendors may choose to support the ones that are important for them. A single SCA implementation may be used by multiple Components, each with a different configuration.

4874

4875

4876

4877

4878

4879

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.

4880

4881

4882

4883

4884

B.3 Service

4885

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.

4886

4887

4888

4889

4890

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 (eg public Web services for use by other organizations). The operations provided by a service are specified by an Interface, as are the operations required by the service client (if there is one). An implementation may contain multiple services, when it is possible to address the services of the implementation separately.

4891

4892

4893

4894

4895

A service may 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.

4896

4897

4898

4899

B.3.1 Remotable Service

4900

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.

4901

4902

4903

4904

How a Service is identified as remotable is dependant on the Component implementation technology used. 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

4905

4906

4907

4908 **B.3.2 Local Service**

4909 Local services are services that are designed to be only used “locally” by other implementations that are
4910 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4911 Local services may rely on by-reference calling conventions, or may assume a very fine-grained
4912 interaction style that is incompatible with remote distribution. They may also use technology-specific data-
4913 types.

4914 How a Service is identified as local is dependant on the Component implementation technology used.
4915 See the relevant SCA Implementation Specification for more information. As an example, to define a
4916 Local Service, a Component implemented in Java would define a Java Interface that does not have the
4917 @Remotable annotation.

4918

4919 **B.4 Reference**

4920 **SCA references** represent a dependency that an implementation has on a service that is supplied by
4921 some other implementation, where the service to be used is specified through configuration. In other
4922 words, a reference is a service that an implementation may call during the execution of its business
4923 function. References are typed by an interface.

4924 For composites, composite references can be accessed by components within the composite like any
4925 service provided by a component within the composite. Composite references can be used as the targets
4926 of wires from component references when configuring Components.

4927 A composite reference can be used to access a service such as: an SCA service provided by another
4928 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,
4929 and so on. References use **bindings** to describe the access method used to their services. SCA provides
4930 an **extensibility mechanism** that allows the introduction of new binding types to references.

4931

4932 **B.5 Implementation**

4933 An implementation is concept that is used to describe a piece of software technology such as a Java
4934 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a
4935 service-oriented application. An SCA composite is also an implementation.

4936 Implementations define points of variability including properties that can be set and settable references to
4937 other services. The points of variability are configured by a component that uses the implementation. The
4938 specification refers to the configurable aspects of an implementation as its **componentType**.

4939 **B.6 Interface**

4940 **Interfaces** define one or more business functions. These business functions are provided by Services
4941 and are used by components through References. Services are defined by the Interface they implement.
4942 SCA currently supports a number of interface type systems, for example:

- 4943 • Java interfaces
- 4944 • WSDL portTypes
- 4945 • C, C++ header files

4946

4947 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional
4948 interface type systems.

4949 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided
4950 by each end of a service communication – this could be the case where a particular service requires a
4951 “callback” interface on the client, which is calls during the process of handing service requests from the
4952 client.

4953

4954 **B.7 Composite**

4955 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an
4956 assembly of Components, Services, References, and the Wires that interconnect them. Composites can
4957 be used to contribute elements to an **SCA Domain**.

4958 A **composite** has the following characteristics:

- 4959 • It may be used as a component implementation. When used in this way, it defines a boundary for
4960 Component visibility. Components may not be directly referenced from outside of the composite
4961 in which they are declared.
- 4962 • It can be used to define a unit of deployment. Composites are used to contribute business logic
4963 artifacts to an SCA domain.

4964

4965 **B.8 Composite inclusion**

4966 One composite can be used to provide part of the definition of another composite, through the process of
4967 inclusion. This is intended to make team development of large composites easier. Included composites
4968 are merged together into the using composite at deployment time to form a single logical composite.

4969 Composites are included into other composites through `<include.../>` elements in the using composite.

4970 The SCA Domain uses composites in a similar way, through the deployment of composite files to a
4971 specific location.

4972

4973 **B.9 Property**

4974 **Properties** allow for the configuration of an implementation with externally set data values. The data
4975 value is provided through a Component, possibly sourced from the property of a containing composite.

4976 Each Property is defined by the implementation. Properties may be defined directly through the
4977 implementation language or through annotations of implementations, where the implementation language
4978 permits, or through a componentType file. A Property can be either a simple data type or a complex data
4979 type. For complex data types, XML schema is the preferred technology for defining the data types.

4980

4981 **B.10 Domain**

4982 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled
4983 by a single organization. As an example, for the accounts department in a business, the SCA Domain
4984 might cover all finance-related functions, and it might contain a series of composites dealing with specific
4985 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4986 A domain specifies the instantiation, configuration and connection of a set of components, provided via
4987 one or more composite files. The domain, like a composite, also has Services and References. Domains
4988 also contain Wires which connect together the Components, Services and References.

4989

4990 **B.11 Wire**

4991 **SCA wires** connect **service references** to **services**.

4992 Valid wire sources are component references. Valid wire targets are component services.

4993 When using included composites, the sources and targets of the wires don't have to be declared in the
4994 same composite as the composite that contains the wire. The sources and targets can be defined by
4995 other included composites. Targets can also be external to the SCA domain.

4997

C. Conformance Items

4998 This section contains a list of conformance items for the SCA Assembly specification.

4999

Conformance ID	Description
[ASM13001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.
[ASM40001]	The extension of a componentType side file name MUST be .componentType.
[ASM40002]	If present, the @constrainingType attribute of a <componentType/> element MUST reference a <constrainingType/> element in the Domain through its QName.
[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 an @type attribute and an @element attribute.
[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 a <service/> element has an interface subelement specified, the interface MUST provide a compatible subset of the interface declared on 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/>

Deleted: [ASM40003]

Deleted: [ASM40004]

Formatted: Font color: Red

Deleted: [ASM50004]

Deleted: [ASM50005]

Deleted: [ASM50006]

- [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 it MUST provide a compatible superset of the interface declared for the equivalent reference in the componentType of the implementation, i.e. provide the same operations or a superset of the operations defined by the implementation for the reference.
- [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, 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 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 MAY require that the address of a target service uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to identify the target service - instead, binding specific attributes and/or child elements must be used.
- [ASM50018] A reference with multiplicity 0..1 or 0..n MAY have no target service defined.
- [ASM50019] A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service defined.
- [ASM50020] A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.
- [ASM50021] A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.
- [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 generate an error no later than when the reference is invoked by the component implementation.
- [ASM50023] Some reference multiplicity errors can be detected at deployment time. In these cases, an error SHOULD be generated by the SCA runtime at deployment time.
- [ASM50024] Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA runtime MUST generate 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

Deleted: [ASM50011]

Deleted: [ASM50012]

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.

[ASM50030]

A <component/> element MUST NOT contain two <property/> subelements with the same value of the @name attribute.

[ASM50031]

The name attribute of a component property MUST match the name of a property element in the component type of the component implementation.

[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 an @type attribute and an @element attribute.

Formatted: Font color: Red

[ASM60001]

A composite name 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, i.e. provide a subset of the operations defined by the component service.

Deleted: [ASM60005]

[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 all the component reference interfaces must be compatible with the composite reference interface. Compatible means that the component reference interface is the same or is a strict 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 value specified for the **multiplicity** attribute of a composite reference MUST be compatible with the multiplicity specified on each of the promoted component references, i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be used where the promoted component reference has

multiplicity 0..n or 1..n and multiplicity 1..n can be used where the promoted component reference has multiplicity 0..n.. However, a composite reference of multiplicity 0..n or 1..n cannot be used to promote a component reference of multiplicity 0..1 or 1..1 respectively.

[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), i.e. provide a superset of the operations in the interface defined by the component for the reference.

Deleted: [ASM60012]

[ASM60013]

If no interface is declared on a composite reference, the interface from one of its promoted component references is used, which MUST be the same as or a compatible superset of the interface(s) declared by the promoted component reference(s).

[ASM60014]

The name attribute of a composite property MUST be unique amongst the properties of the same composite.

[ASM60015]

the source interface and the target interface of a wire MUST either both be remotable or else both be local

[ASM60016]

the operations on the target interface of a wire MUST be the same as or be a superset of the operations in the interface specified on the source

[ASM60017]

compatibility between the source interface and the target interface for a wire for the individual operations is defined as compatibility of the signature, that is operation name, input types, and output types MUST be the same.

[ASM60018]

the order of the input and output types for operations in the source interface and the target interface of a wire also MUST be the same.

[ASM60019]

the set of Faults and Exceptions expected by each operation in the source interface MUST be the same or be a superset of those specified by the target interface.

[ASM60020]

other specified attributes of the source interface and the target interface of a wire MUST match, including Scope and Callback interface

[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.

[ASM60022]

For each component reference for which autowire is enabled, the the SCA runtime MUST search within the composite for target services which are compatible with the reference.

Deleted: [ASM60022]

[ASM60023]

the target service interface MUST be a compatible superset of the reference interface when using autowire to wire a reference (as defined in [the section on Wires](#))

[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 (according to the various rules for specifying target services for a component reference described in section 5.3.1).

Deleted: [ASM60033]

~~[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.

Deleted: [ASM60034]

[ASM60035]

A single property element MUST NOT contain both an @type attribute and an @element attribute.

[ASM70001]

The constrainingType specifies the services, references and properties that MUST be implemented by the implementation of the component to which the constrainingType is attached.

[ASM70002]

If the configuration of the component or its implementation do not conform to the constrainingType specified on the component element, the SCA runtime MUST raise an error.

[ASM70003]

The name attribute of the constraining type MUST be unique in the SCA domain.

[ASM70004]

When an implementation is constrained by a constrainingType its component type MUST contain all the services, references and properties specified in the constrainingType.

[ASM70005]

An implementation MAY contain additional services, additional optional references (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or 1..n) or additional non-optional properties (a property with mustSupply=true).

~~[ASM70006]~~

Additional services, references and properties provided by the implementation which are not declared in the constrainingType associated with a component MUST NOT be configured in any way by the containing composite.

Deleted: [ASM70006]

~~[ASM70007]~~

A component or implementation can use a qualified form of an intent specified in unqualified form in the constrainingType, but if the constrainingType uses the qualified form of an intent, then the component or implementation MUST also use the qualified form, otherwise there is an error.

Deleted: [ASM70007]

[ASM80001]

The interface.wsdl @interface attribute MUST reference a portType of a WSDL 1.1 document.

[ASM80002]

Remotable service Interfaces MUST NOT make use of **method or operation overloading**.

[ASM80003]

If a remotable service is called locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the service or post-invocation modifications to return messages are seen by the caller.

[ASM80004]

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

[ASM80005]

Either both interfaces of a bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and remote services.

[ASM80006]

Where a service or a reference has a conversational interface, the conversational intent MUST be attached either to the interface itself, or to the service or reference using the interface.

[ASM80007]

Once an operation marked with endsConversation has been invoked, any subsequent attempts to call an operation or a callback operation associated with the same conversation MUST generate a sca:ConversationViolation fault.

- [ASM80008] Any service or reference that uses an interface marked with required 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.
- [ASM80012] Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface.
- [ASM80013] If the service or reference declaration in the component type declares an interface without a callback interface, then the component configuration for the corresponding service or reference MUST NOT declare an interface with a callback interface.
- [ASM80014] Where a composite declares an interface for a composite service or a composite reference, if the promoted service or promoted reference has an interface which has a callback interface, then the interface declaration for the composite service or the composite reference MUST also declare a compatible interface with a compatible callback interface.
- [ASM80015] If the promoted service or promoted reference has an interface without a callback interface, then the interface declaration for the composite service or composite reference MUST NOT declare a callback interface.
- ~~[ASM80016]~~ The interface.wsdl @callbackInterface attribute, if present, MUST reference a portType of a WSDL 1.1 document. Deleted: [ASM80016]
- [ASM90001] For a binding of a **reference** the URI attribute defines the target URI of the reference. This MUST be either the componentName/serviceName 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, only one binding can have the default name value; all others must have a name value specified that is unique within the service or reference.
- [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]~~ a wire target MAY be specified with a syntax of "componentName/serviceName/bindingName". Deleted: [ASM90004]
- [ASM10001] all of the QNames for the definitions contained in definitions.xml files MUST be unique within the domain.
- [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
- [ASM12002] Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy 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.
- [ASM12004] Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the namespaces of constructs that are needed by the contribution and which are

be found elsewhere, for example in other contributions.

[ASM12005]

Where present, these 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.~~

Deleted: [ASM12006]

[ASM12007]

Implementations of SCA MAY also generate an error if there are conflicting names exported from multiple contributions.

[ASM12008]

SCA runtimes MAY choose not to provide the contribution functions functionality in any way.

[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.~~

Deleted: [ASM12010]

~~[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 (eg 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.~~

Deleted: [ASM12011]

[ASM12012]

The value of @autowire for the logical domain composite MUST be autowire="false".

[ASM12013]

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:

- 1) The SCA runtime MAY disallow deployment of any components with autowire References. In this case, the SCA runtime MUST generate an exception at the point where the component is deployed.
- 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component is deployed and not update those targets when later deployment actions occur.
- 3) The SCA runtime MAY re-evaluate 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 new configuration of the reference takes place is described by the relevant client and implementation specifications.

[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.

~~[ASM12015]~~

~~Where components are updated by deployment actions (their configuration is changed in some way, which may include changing the wires of component references), the new configuration MUST apply to all new instances of those components once the update is complete.~~

Deleted: [ASM12015]

~~[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.~~

Deleted: [ASM12016]

~~[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:~~

- 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

Deleted: [ASM12017]

~~[ASM12018]~~

~~Where a component that is the target of a wire is updated, future invocations of that reference SHOULD use the updated component.~~

Deleted: [ASM12018]

~~[ASM12019]~~

~~Where an existing domain level component is updated, an SCA runtime MAY maintain a copy of a component offering a conversational service until all existing~~

Deleted: [ASM12019]

conversations complete - alternatively all existing conversations MAY be terminated.

~~[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:

- 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:

1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to locate artifacts (ie 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 the precedence order above.

Deleted: [ASM12020]

5001

D. Acknowledgements

5002 The following individuals have participated in the creation of this specification and are gratefully
5003 acknowledged:

5004 **Participants:**

5005 [Participant Name, Affiliation | Individual Member]

5006 [Participant Name, Affiliation | Individual Member]

5007

E. Non-Normative Text

5009

F. Revision History

5010 [optional; should not be included in OASIS Standards]

5011

Revision	Date	Editor	Changes Made
1	2007-09-24	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
2	2008-01-04	Michael Beisiegel	<p>composite section</p> <ul style="list-style-type: none"> - changed order of subsections from property, reference, service to service, reference, property - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) - added section in appendix to contain complete pseudo schema of composite <p>- moved component section after implementation section</p> <ul style="list-style-type: none"> - made the ConstrainingType section a top level section - moved interface section to after constraining type section <p>component section</p> <ul style="list-style-type: none"> - added subheadings for Implementation, Service, Reference, Property - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) <p>implementation section</p> <ul style="list-style-type: none"> - changed title to "Implementation and ComponentType" - moved implementation instance related stuff from implementation section to component implementation section - added subheadings for Service, Reference, Property, Implementation - progressive disclosure of pseudo schemas, each section only shows what is described - attributes description now starts with name : type (cardinality) - child element description as list, each item starting with name : type (cardinality) - attribute and element description still needs to be completed, all implementation statements

			<p>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 Combella's review contained in email: http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html</p>
7 CD01 - Rev3	2008-11-18	Mike Edwards	<ul style="list-style-type: none"> • Specification marked for conformance statements. New Appendix (D) added

			containing a table of all conformance statements. Mass of related minor editorial changes to remove the use of RFC2119 words where not appropriate.
8 CD01 - Rev4	2008-12-11	Mike Edwards	<ul style="list-style-type: none"> - Fix problems of misplaced statements in Appendix D - Fixed problems in the application of Issue 57 - section 5.3.1 & Appendix D as defined in email: http://lists.oasis-open.org/archives/sca-assembly/200811/msg00045.html - Added Conventions section, 1.3, as required by resolution of Issue 96. - Issue 32 applied - section B2 - Editorial addition to section 8.1 relating to no operation overloading for remotable interfaces, as agreed at TC meeting of 16/09/2008.
9 CD01 - Rev5	2008-12-22	Mike Edwards	<ul style="list-style-type: none"> - Schemas in Appendix B updated with resolutions of Issues 32 and 60 - Schema for contributions - Appendix B12 - updated with resolutions of Issues 53 and 74. - Issues 53 and 74 incorporated - Sections 11.4, 11.5
10 CD01-Rev6	2008-12-23	Mike Edwards	<ul style="list-style-type: none"> - Issues 5, 71, 92 - Issue 14 - remaining updates applied to ComponentType (section 4.1.3) and to Composite Property (section 6.3)
11 CD01-Rev7	2008-12-23	Mike Edwards	<p>All changes accepted before revision from Rev6 started - due to changes being applied to previously changed sections in the Schemas</p> <ul style="list-style-type: none"> Issues 12 & 18 - Section B2 Issue 63 - Section C3 Issue 75 - Section C12 Issue 65 - Section 7.0 Issue 77 - Section 8 + Appendix D Issue 69 - Sections 5.1, 8 Issue 45 - Sections 4.1.3, 5.4, 6.3, B2. Issue 56 - Section 8.2, Appendix D Issue 41 - Sections 5.3.1, 6.4, 12.7, 12.8, Appendix D
12 CD01-Rev8	2008-12-30	Mike Edwards	<ul style="list-style-type: none"> Issue 72 - Removed Appendix A Issue 79 - Sections 9.0, 9.2, 9.3, Appendix A.2 Issue 62 - Sections 4.1.3, 5.4 Issue 26 - Section 6.5 Issue 51 - Section 6.5 Issue 36 - Section 4.1 Issue 44 - Section 10, Appendix C Issue 89 - Section 8.2, 8.5, Appendix A, Appendix C Issue 16 - Section 6.8, 9.4 Issue 8 - Section 11.2.1 Issue 17 - Section 6.6 Issue 30 - Sections 4.1.1, 4.1.2, 5.2, 5.3, 6.1, 6.2, 9 Issue 33 - insert new Section 8.4
12 CD01-Rev8a	2009-01-13	Bryan Aupperle Mike Edwards	Issue 99 - Section 8

13 CD02	2009-01-14	Mike Edwards	All changes accepted All comments removed
---------	------------	--------------	--

5012