



# Service Component Architecture Assembly Model Specification Version 1.1

Committee Draft 01 Revision 3 **+ Issue 90**

18th November 2008

**Specification URIs:**

**This Version:**

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01.html>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01.doc>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01.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 01 Revision 3 .....	1
The non-normative errata page for this specification is located at <a href="http://www.oasis-open.org/committees/sca-assembly/">http://www.oasis-open.org/committees/sca-assembly/</a> .....	3
Notices .....	4
Table of Contents .....	5
1 Introduction .....	8
1.1 Terminology .....	8
1.2 Normative References .....	8
2 Overview .....	10
2.1 Diagram used to Represent SCA Artifacts .....	11
3 Quick Tour by Sample .....	13
4 Implementation and ComponentType .....	14
4.1 Component Type .....	14
4.1.1 Service .....	15
4.1.2 Reference .....	16
4.1.3 Property .....	18
4.1.4 Implementation .....	19
4.2 Example ComponentType .....	20
4.3 Example Implementation .....	20
5 Component .....	23
5.1 Implementation .....	24
5.2 Service .....	25
5.3 Reference .....	27
5.3.1 Specifying the Target Service(s) for a Reference .....	29
5.4 Property .....	30
5.5 Example Component .....	33
6 Composite .....	37
6.1 Service .....	39
6.1.1 Service Examples .....	40
6.2 Reference .....	42
6.2.1 Example Reference .....	44
6.3 Property .....	46
6.3.1 Property Examples .....	47
6.4 Wire .....	<del>51</del>
6.4.1 Wire Examples .....	<del>53</del>
6.4.2 Autowire .....	54
6.4.3 Autowire Examples .....	55
6.5 Using Composites as Component Implementations .....	58
6.5.1 Example of Composite used as a Component Implementation .....	60
6.6 Using Composites through Inclusion .....	61
6.6.1 Included Composite Examples .....	62
6.7 Composites which Include Component Implementations of Multiple Types .....	65
7 ConstrainingType .....	66

Deleted: 50

Deleted: 52

7.1	Example constrainingType .....	67
8	Interface.....	69
8.1	Local and Remotable Interfaces.....	70
8.2	Bidirectional Interfaces .....	71
8.3	Conversational Interfaces .....	71
8.4	SCA-Specific Aspects for WSDL Interfaces .....	73
9	Binding.....	75
9.1	Messages containing Data not defined in the Service Interface .....	77
9.2	Form of the URI of a Deployed Binding.....	77
9.2.1	Constructing Hierarchical URIs .....	77
9.2.2	Non-hierarchical URIs .....	78
9.2.3	Determining the URI scheme of a deployed binding.....	78
9.3	SCA Binding .....	79
9.3.1	Example SCA Binding .....	79
9.4	Web Service Binding .....	80
9.5	JMS Binding.....	80
10	SCA Definitions .....	81
11	Extension Model.....	82
11.1	Defining an Interface Type.....	82
11.2	Defining an Implementation Type.....	84
11.3	Defining a Binding Type.....	85
12	Packaging and Deployment .....	88
12.1	Domains.....	88
12.2	Contributions.....	88
12.2.1	SCA Artifact Resolution.....	89
12.2.2	SCA Contribution Metadata Document .....	90
12.2.3	Contribution Packaging using ZIP .....	92
12.3	Installed Contribution .....	92
12.3.1	Installed Artifact URIs.....	92
12.4	Operations for Contributions.....	93
12.4.1	install Contribution & update Contribution.....	93
12.4.2	add Deployment Composite & update Deployment Composite.....	93
12.4.3	remove Contribution .....	93
12.5	Use of Existing (non-SCA) Mechanisms for Resolving Artifacts .....	93
12.6	Domain-Level Composite .....	94
12.6.1	add To Domain-Level Composite.....	94
12.6.2	remove From Domain-Level Composite .....	95
12.6.3	get Domain-Level Composite .....	95
12.6.4	get QName Definition .....	95
13	Conformance .....	96
A.	Pseudo Schema .....	97
A.1	ComponentType.....	97
A.2	Composite .....	98
B.	XML Schemas .....	100
B.1	sca.xsd .....	100

B.2 sca-core.xsd .....	100
B.3 sca-binding-sca.xsd.....	109
B.4 sca-interface-java.xsd .....	110
B.5 sca-interface-wsdl.xsd .....	110
B.6 sca-implementation-java.xsd.....	111
B.7 sca-implementation-composite.xsd .....	112
B.8 sca-definitions.xsd .....	112
B.9 sca-binding-webservice.xsd .....	113
B.10 sca-binding-jms.xsd.....	113
B.11 sca-policy.xsd.....	113
B.12 sca-contribution.xsd .....	113
C. SCA Concepts .....	115
C.1 Binding .....	115
C.2 Component.....	115
C.3 Service .....	115
C.3.1 Remotable Service .....	115
C.3.2 Local Service .....	116
C.4 Reference.....	116
C.5 Implementation.....	116
C.6 Interface .....	116
C.7 Composite .....	117
C.8 Composite inclusion.....	117
C.9 Property.....	117
C.10 Domain .....	117
C.11 Wire .....	117
D. Conformance Items .....	118
E. Acknowledgements .....	125
F. Non-Normative Text .....	126
G. Revision History.....	127

---

# 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.

[1] SCA Java Component Implementation Specification

SCA Java Common Annotations and APIs Specification

[http://www.osoa.org/download/attachments/35/SCA\\_JavaComponentImplementation\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf)

[http://www.osoa.org/download/attachments/35/SCA\\_JavaAnnotationsAndAPIs\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf)

[2] 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](http://www.osoa.org/download/attachments/28/SCA_BuildingYourFirstApplication_V09.pdf)

[4] JAX-WS Specification

<http://jcp.org/en/jsr/detail?id=101>

[5] WS-I Basic Profile

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

[6] WS-I Basic Security Profile

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

- 40
- 41 [7] Business Process Execution Language (BPEL)
- 42 [http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel)
- 43
- 44 [8] WSDL Specification
- 45 WSDL 1.1: <http://www.w3.org/TR/wsdl>
- 46 WSDL 2.0: <http://www.w3.org/TR/wsdl20/>
- 47
- 48 [9] SCA Web Services Binding Specification
- 49 [http://www.osoa.org/download/attachments/35/SCA\\_WebServiceBindings\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_WebServiceBindings_V100.pdf)
- 50
- 51 [10] SCA Policy Framework Specification
- 52 [http://www.osoa.org/download/attachments/35/SCA\\_Policy\\_Framework\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf)
- 53
- 54 [11] SCA JMS Binding Specification
- 55 [http://www.osoa.org/download/attachments/35/SCA\\_JMSBinding\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JMSBinding_V100.pdf)
- 56
- 57 [12] ZIP Format Definition
- 58 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>
- 59
- 60 [13] Infoset Specification
- 61 <http://www.w3.org/TR/xml-infoset/>
- 62

---

## 63 2 Overview

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

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

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

80 One basic artifact of SCA is the **component**, which is the unit of construction for SCA. A component  
81 consists of a configured instance of an implementation, where an implementation is the piece of program  
82 code providing business functions. The business function is offered for use by other components as  
83 **services**. Implementations can depend on services provided by other components – these dependencies  
84 are called **references**. Implementations can have settable **properties**, which are data values which  
85 influence the operation of the business function. The component **configures** the implementation by  
86 providing values for the properties and by wiring the references to services provided by other  
87 components.

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

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

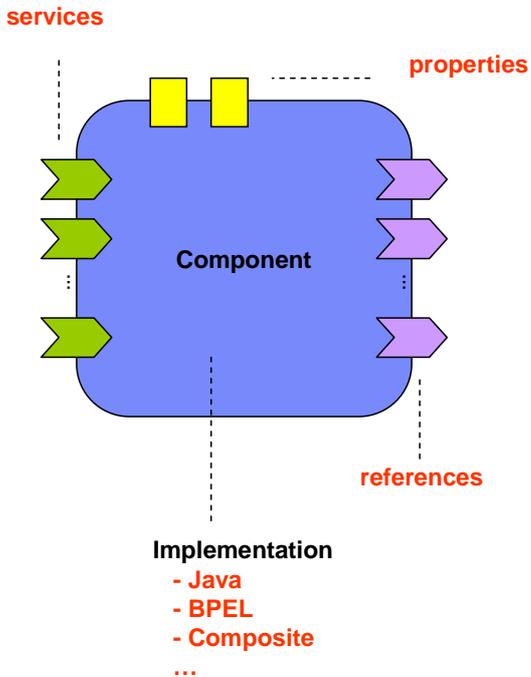
101 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services  
102 providing an area of business functionality that is controlled by a single organization. As an example, for  
103 the accounts department in a business, the SCA Domain might cover all financial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

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

113 **2.1 Diagram used to Represent SCA Artifacts**

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

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



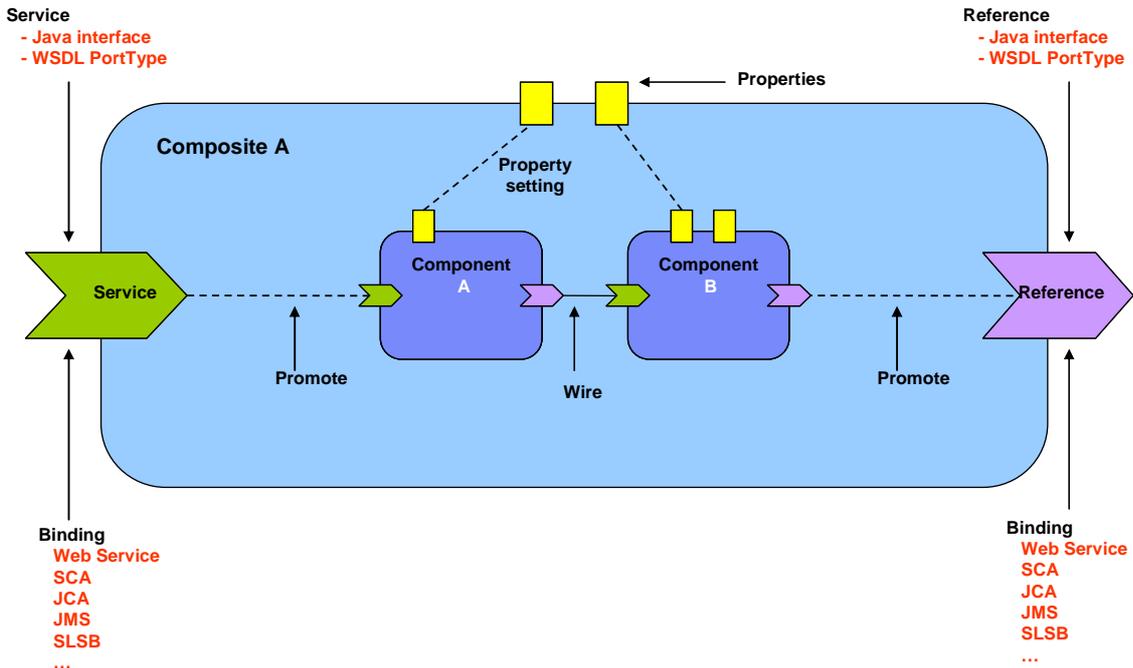
118

119 *Figure 1: SCA Component Diagram*

120

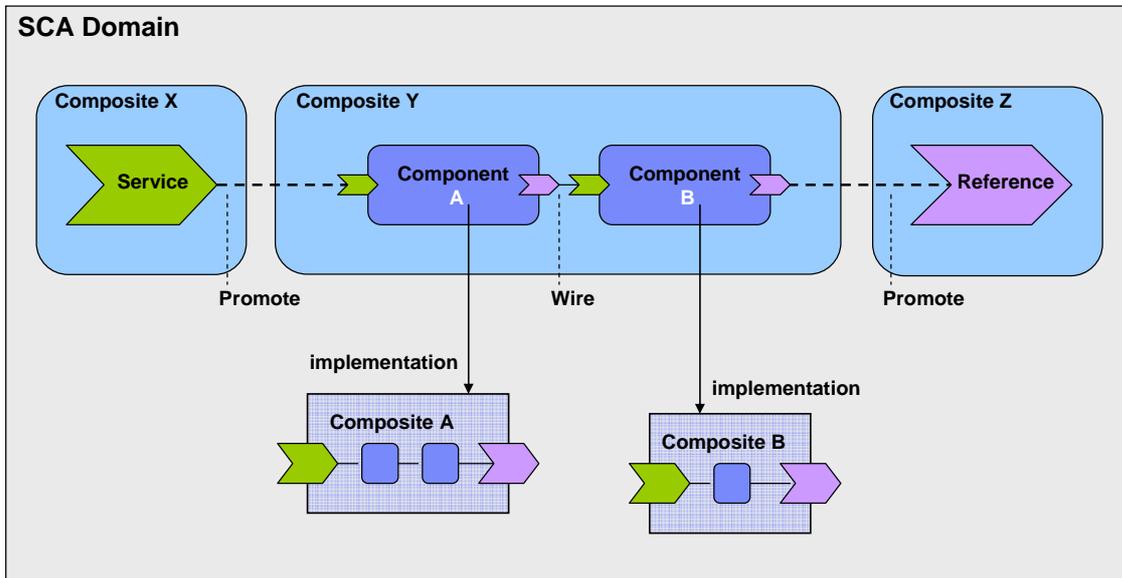
121 The following picture illustrates some of the features of a composite assembled using a set of  
122 components:

123



124  
125 *Figure 2: SCA Composite Diagram*

126  
127 The following picture illustrates an SCA Domain assembled from a series of high-level composites, some  
128 of which are in turn implemented by lower-level composites:



129  
130 *Figure 3: SCA Domain Diagram*

---

131 **3 Quick Tour by Sample**

132 To be completed.

133

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

135

136

---

## 137 4 Implementation and ComponentType

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

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

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

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

151 So, for example:

- 152 • for a service, the implementation might define the interface, binding(s), a URI, intents,  
153 and policy sets, including details of the bindings
- 154 • for a reference, the implementation might define the interface, binding(s), target URI(s),  
155 intents, policy sets, including details of the bindings
- 156 • for a property the implementation might define its type and a default value
- 157 • the implementation itself might define policy intents or concrete policy sets

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

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

166

### 167 4.1 Component Type

168 **Component type** represents the configurable aspects of an implementation. A component type  
169 consists of services that are offered, references to other services that can be wired and properties  
170 that can be set. The settable properties and the settable references to services are configured by a  
171 component which uses the implementation.

172 The **component type is calculated in two steps** where the second step adds to the information  
173 found in the first step. Step one is introspecting the implementation (if possible), including the  
174 inspection of implementation annotations (if available). Step two covers the cases where  
175 introspection of the implementation is not possible or where it does not provide complete  
176 information and it involves looking for an SCA **component type file**. Component type  
177 information found in the component type file must be compatible with the equivalent information  
178 found from inspection of the implementation. The component type file can specify partial  
179 information, with the remainder being derived from the implementation.

180 In the ideal case, the component type information is determined by inspecting the  
181 implementation, for example as code annotations. The component type file provides a mechanism  
182 for the provision of component type information for implementation types where the information  
183 cannot be determined by inspecting the implementation.

184 The component type is defined by a `componentType` element in the `componentType` file. The  
 185 extension of a `componentType` file MUST be `.componentType` and its name and location depends  
 186 on the type of the component implementation: the specifics are described in the respective client  
 187 and implementation model specification for the implementation type.

188

189 The following snippet shows the `componentType` schema.

190

```
191 <?xml version="1.0" encoding="ASCII"?>
192 <!-- Component type schema snippet -->
193 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
194               constrainingType="QName"? >
195
196     <service ... /*>
197     <reference ... /*>
198     <property ... /*>
199     <implementation ... /*?
200
201 </componentType>
202
```

203 The **`componentType`** element has the following **attribute**:

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

209

210 The **`componentType`** element has the following **child elements**:

- 211 • **`service : Service (0..n)`** – see [component type service section](#).
- 212 • **`reference : Reference (0..n)`** – see [component type reference section](#).
- 213 • **`property : Property (0..n)`** – see [component type property section](#).
- 214 • **`implementation : Implementation (0..1)`** – see [component type implementation](#)  
 215 [section](#).

216

## 217 4.1.1 Service

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

222

```
223 <?xml version="1.0" encoding="ASCII"?>
224 <!-- Component type service schema snippet -->
225 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
226 >
227
228     <service name="xs:NCName"
229             requires="list of xs:QName"? policySets="list of xs:QName"?>*
230         <interface ... /*>
231         <binding ... /*>
232         <callback?>
```

```

233         <binding ... />+
234     </callback>
235 </service>
236
237 <reference ... />*
238 <property ... />*
239 <implementation ... />?
240
241 </componentType>
242

```

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

- 244 • **name : NCName (1..1)** - the name of the service. The @name attribute of a <service/>  
245 child element of a <componentType/> MUST be unique amongst the service elements of  
246 that <componentType/>. [ASM40003]
- 247 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
248 [10] for a description of this attribute.
- 249 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
250 [10] for a description of this attribute.

251

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

- 253 • **interface : Interface (1..1)** - A service has **one interface**, which describes the  
254 operations provided by the service. For details on the interface element see [the Interface](#)  
255 [section](#).
- 256 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as  
257 children. If the binding element is not present it defaults to <binding.sca>. Details of the  
258 binding element are described in [the Bindings section](#).
- 259 • **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback**  
260 element used if the interface has a callback defined, which has one or more **binding**  
261 elements as children. The **callback** and its binding child elements are specified if there is  
262 a need to have binding details used to handle callbacks. If the callback element is not  
263 present, the behaviour is runtime implementation dependent. For details on callbacks, see  
264 [the Bidirectional Interfaces section](#).

265

## 266 4.1.2 Reference

267 A **Reference** represents a requirement that the implementation has on a service provided by  
268 another component. The reference is represented by a **reference element** which is a child of the  
269 componentType element. There can be **zero or more** reference elements in a component type  
270 definition. The following snippet shows the component type schema with the schema for a  
271 reference child element:

272

```

273 <?xml version="1.0" encoding="ASCII"?>
274 <!-- Component type reference schema snippet -->
275 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
276 >
277
278     <service ... />*
279
280     <reference name="xs:NCName"
281               target="list of xs:anyURI"? autowire="xs:boolean"?
282               multiplicity="0..1 or 1..1 or 0..n or 1..n"?
283               wiredByImpl="xs:boolean"?

```

```

284         requires="list of xs:QName"? policySets="list of xs:QName"?>*
285     <interface ... />
286     <binding ... />*
287     <callback?
288         <binding ... />+
289     </callback>
290 </reference>
291
292 <property ... />*
293 <implementation ... />?
294
295 </componentType>
296

```

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

- 298 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a  
299 <reference/> child element of a <componentType/> MUST be unique amongst the  
300 reference elements of that <componentType/>. [ASM40004]
- 301 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect  
302 the reference to target services. The multiplicity can have the following values
  - 303 ○ 0..1 – zero or one wire can have the reference as a source
  - 304 ○ 1..1 – one wire can have the reference as a source
  - 305 ○ 0..n - zero or more wires can have the reference as a source
  - 306 ○ 1..n – one or more wires can have the reference as a source
- 307 If @multiplicity is not specified, the default value is "1..1".
- 308 • **target : anyURI (0..n)** - a list of one or more of target service URI's, depending on  
309 multiplicity setting. Each value wires the reference to a component service that resolves  
310 the reference. For more details on wiring see [the section on Wires](#).
- 311 • **autowire : boolean (0..1)** - whether the reference should be autowired, as described in  
312 [the Autowire section](#). Default is false.
- 313 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the  
314 reference is wired to the target(s) configured on the reference. If set to "true" it indicates  
315 that the target of the reference is set at runtime by the implementation code (eg by the  
316 code obtaining an endpoint reference by some means and setting this as the target of the  
317 reference through the use of programming interfaces defined by the relevant Client and  
318 Implementation specification). If @wiredByImpl is set to "true", then any reference  
319 targets configured for this reference MUST be ignored by the runtime. [ASM40006] It is  
320 recommended that any references with @wiredByImpl = "true" are left unwired.
- 321 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
322 [10] for a description of this attribute.
- 323 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
324 [10] for a description of this attribute.

325

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

- 327 • **interface : Interface (1..1)** - A reference has **one interface**, which describes the  
328 operations required by the reference. The interface is described by an **interface element**  
329 which is a child element of the reference element. For details on the interface element see  
330 [the Interface section](#).
- 331 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
332 children. Details of the binding element are described in the [Bindings section](#).

333 Note that a binding element may specify an endpoint which is the target of that binding. A  
334 reference must not mix the use of endpoints specified via binding elements with target  
335 endpoints specified via the target attribute. If the target attribute is set, then binding  
336 elements can only list one or more binding types that can be used for the wires identified  
337 by the target attribute. All the binding types identified are available for use on each wire  
338 in this case. If endpoints are specified in the binding elements, each endpoint must use  
339 the binding type of the binding element in which it is defined. In addition, each binding  
340 element needs to specify an endpoint in this case.

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

347

### 348 4.1.3 Property

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

353

```
354 <?xml version="1.0" encoding="ASCII"?>
355 <!-- Component type property schema snippet -->
356 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
357 >
358
359     <service ... />*
360     <reference ... >*
361
362     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
363         many="xs:boolean"? mustSupply="xs:boolean"?
364         requires="list of xs:QName"?
365         policySets="list of xs:QName"?>*
366         default-property-value?
367     </property>
368
369     <implementation ... />?
370
371 </componentType>
372
```

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

- 374 ▪ **name : NCName (1..1)** - the name of the property. The @name attribute of a  
375 <property/> child element of a <componentType/> MUST be unique amongst the  
376 property elements of that <componentType/>. [ASM40005]
- 377 ▪ one of **(1..1)**:
  - 378 ○ **type : QName** - the type of the property defined as the qualified name of an XML  
379 schema type. The value of the property @type attribute MUST be the QName of  
380 an XML schema type. [ASM40007]
  - 381 ○ **element : QName** - the type of the property defined as the qualified name of an  
382 XML schema global element – the type is the type of the global element. The value  
383 of the property @element attribute MUST be the QName of an XSD global  
384 element. [ASM40008]

- 385
- 386
- 387
- 388
- 389
- 390
- 391
- 392
- 393
- 394
- **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-valued (true). In the case of a multi-valued property, it is presented to the implementation as a collection of property values.
  - **mustSupply : boolean (0..1)** - whether the property value must be supplied by the component that uses the implementation – when mustSupply="true" the component must supply a value since the implementation has no default value for the property. A default-property-value should only be supplied 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.
  - **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the `property`.

Comment [mbgl2]: Issue 68

395 The value for a property is supplied to the implementation of a component at the time that the

396 implementation is started. The implementation can choose to use the supplied value in any way

397 that it chooses. In particular, the implementation can alter the internal value of the property at

398 any time. However, if the implementation queries the SCA system for the value of the property,

399 the value as defined in the SCA composite is the value returned.

400 The `componentType` property element can contain an SCA default value for the property declared

401 by the implementation. However, the implementation can have a property which has an

402 implementation defined default value, where the default value is not represented in the

403 componentType. An example of such a default value is where the default value is computed at

404 runtime by some code contained in the implementation. If a using component needs to control the

405 value of a property used by an implementation, the component sets the value explicitly. The SCA

406 runtime MUST ensure that any implementation default property value is replaced by a value for

407 that property explicitly set by a component using that implementation. [ASM40009]

Comment [mbgl3]: Issue 38

408

#### 409 4.1.4 Implementation

410 **Implementation** represents characteristics inherent to the implementation itself, in particular

411 intents and policies. See the [Policy Framework specification \[10\]](#) for a description of intents and

412 policies. The following snippet shows the component type schema with the schema for a

413 implementation child element:

414

```
415 <?xml version="1.0" encoding="ASCII"?>
416 <!-- Component type implementation schema snippet -->
417 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
418 >
419
420     <service ... />*
421     <reference ... >*
422     <property ... />*
423
424     <implementation requires="list of xs:QName"?
425                 policySets="list of xs:QName"?/>?
426
427 </componentType>
```

428

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

- 430
- 431
- **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.

432

433

434

## 435 4.2 Example ComponentType

436

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

440

```
441 <?xml version="1.0" encoding="ASCII"?>
442 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">
443     <service name="MyValueService">
444         <interface.java interface="services.myvalue.MyValueService" />
445     </service>
446
447     <reference name="customerService">
448         <interface.java interface="services.customer.CustomerService" />
449     </reference>
450
451     <reference name="stockQuoteService">
452         <interface.java
453             interface="services.stockquote.StockQuoteService" />
454     </reference>
455
456     <property name="currency" type="xsd:string">USD</property>
457
458 </componentType>
459
```

## 460 4.3 Example Implementation

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

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

465

```
466 package services.account;
467
468 @Remotable
469 public interface AccountService {
470
471     AccountReport getAccountReport(String customerID);
472 }
473
```

473

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

477

```
478 package services.account;
479
480 import java.util.List;
481
482 import commonj.sdo.DataFactory;
483
```

```

484     import org.osoa.sca.annotations.Property;
485     import org.osoa.sca.annotations.Reference;
486
487     import services.accountdata.AccountDataService;
488     import services.accountdata.CheckingAccount;
489     import services.accountdata.SavingsAccount;
490     import services.accountdata.StockAccount;
491     import services.stockquote.StockQuoteService;
492
493     public class AccountServiceImpl implements AccountService {
494
495         @Property
496         private String currency = "USD";
497
498         @Reference
499         private AccountDataService accountDataService;
500         @Reference
501         private StockQuoteService stockQuoteService;
502
503         public AccountReport getAccountReport(String customerID) {
504
505             DataFactory dataFactory = DataFactory.INSTANCE;
506             AccountReport accountReport = (AccountReport)dataFactory.create(AccountReport.class);
507             List accountSummaries = accountReport.getAccountSummaries();
508
509             CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
510             AccountSummary checkingAccountSummary =
511 (AccountSummary)dataFactory.create(AccountSummary.class);
512             checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
513             checkingAccountSummary.setAccountType("checking");
514             checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
515             accountSummaries.add(checkingAccountSummary);
516
517             SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
518             AccountSummary savingsAccountSummary =
519 (AccountSummary)dataFactory.create(AccountSummary.class);
520             savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
521             savingsAccountSummary.setAccountType("savings");
522             savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
523             accountSummaries.add(savingsAccountSummary);
524
525             StockAccount stockAccount = accountDataService.getStockAccount(customerID);
526             AccountSummary stockAccountSummary =
527 (AccountSummary)dataFactory.create(AccountSummary.class);
528             stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
529             stockAccountSummary.setAccountType("stock");
530             float balance=
531 (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
532             stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
533             accountSummaries.add(stockAccountSummary);
534
535             return accountReport;

```

```

536     }
537
538     private float fromUSDollarToCurrency(float value){
539
540         if (currency.equals("USD")) return value; else
541         if (currency.equals("EURO")) return value * 0.8f; else
542         return 0.0f;
543     }
544 }
545

```

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

```

548
549 <?xml version="1.0" encoding="ASCII"?>
550 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
551               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
552
553     <service name="AccountService">
554         <interface.java interface="services.account.AccountService"/>
555     </service>
556     <reference name="accountDataService">
557         <interface.java
558 interface="services.accountdata.AccountDataService"/>
559     </reference>
560     <reference name="stockQuoteService">
561         <interface.java
562 interface="services.stockquote.StockQuoteService"/>
563     </reference>
564
565     <property name="currency" type="xsd:string">USD</property>
566
567 </componentType>
568

```

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

## 572 5 Component

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

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

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

```
582  
583 <?xml version="1.0" encoding="UTF-8"?>  
584 <!-- Component schema snippet -->  
585 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
586   ...  
587   <component name="xs:NCName" autowire="xs:boolean"?  
588     requires="list of xs:QName"? policySets="list of xs:QName"?  
589     constrainingType="xs:QName"?> *  
590     <implementation ... />?  
591     <service ... /> *  
592     <reference ... /> *  
593     <property ... /> *  
594   </component>  
595   ...  
596 </composite>  
597
```

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

- 599 • **name : NCName (1..1)** – the name of the component. The @name attribute of a  
600 <service/> child element of a <componentType/> MUST be unique amongst the service  
601 elements of that <componentType/>. [ASM50001]
- 602 • **autowire : boolean (0..1)** – whether contained component references should be  
603 autowired, as described in [the Autowire section](#). Default is false.
- 604 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
605 [10] for a description of this attribute.
- 606 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
607 [10] for a description of this attribute.
- 608 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
609 the set of services, references and properties of the component, plus related intents, is  
610 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
611 for more details.

612

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

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

- 616 • **service** : *ComponentService (0..n)* – see component service section.
- 617 • **reference** : *ComponentReference (0..n)* – see component reference section.
- 618 • **property** : *ComponentProperty (0..n)* – see component property section.
- 619

## 620 5.1 Implementation

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

```
626  
627 <?xml version="1.0" encoding="UTF-8"?>  
628 <!-- Component Implementation schema snippet -->  
629 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
630   ...  
631   <component ... >*  
632     <implementation ... />?  
633     <service ... />*  
634     <reference ... />*  
635     <property ... />*  
636   </component>  
637   ...  
638 </composite>  
639
```

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

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

Comment [mbgl4]: Issue  
69 part 1

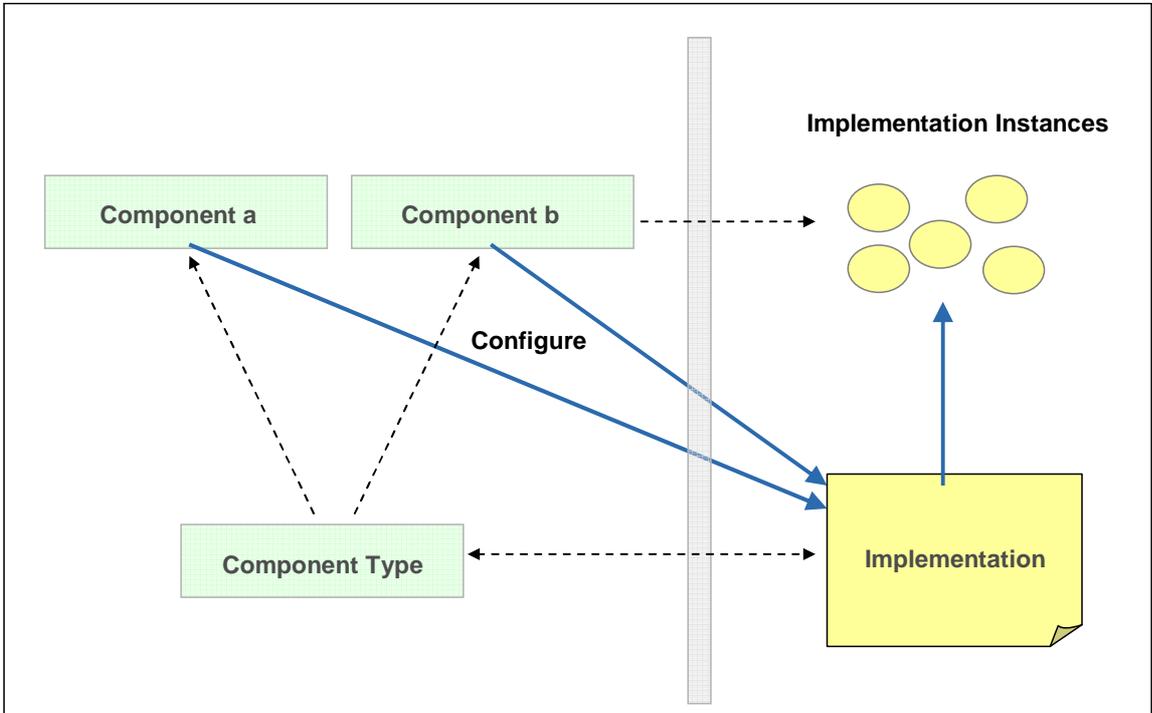
648 The following snippets show implementation elements for the Java and BPEL implementation types  
649 and for the use of a composite as an implementation:

```
650  
651 <implementation.java class="services.myvalue.MyValueServiceImpl"/>  
652  
653 <implementation.bpel process="ans:MoneyTransferProcess"/>  
654  
655 <implementation.composite name="bns:MyValueComposite"/>  
656  
657
```

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

659

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



665  
 666 *Figure 4: Relationship of Component and Implementation*  
 667

## 668 5.2 Service

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

```
673
674 <?xml version="1.0" encoding="UTF-8"?>
675 <!-- Component Service schema snippet -->
676 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
677   ...
678   <component ... >*
679     <implementation ... />?
680     <service name="xs:NCName" requires="list of xs:QName"?
681       policySets="list of xs:QName"?>*
682     <interface ... />?
683     <binding ... />*
```

```

684         <callback>?
685             <binding ... />+
686         </callback>
687     </service>
688     <reference ... />*
689     <property ... />*
690 </component>
691 ...
692 </composite>
693

```

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

- 695 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service  
696 element of a <component/> MUST be unique amongst the service elements of that  
697 <component/> [ASM50002] The @name attribute of a service element of a  
698 <component/> MUST match the @name attribute of a service element of the  
699 componentType of the <implementation/> child element of the component. [ASM50003]
- 700 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
701 [\[10\]](#) for a description of this attribute.  
702 Note: The effective set of policy intents for the service consists of any intents explicitly  
703 stated in this requires attribute, combined with any intents specified for the service by the  
704 implementation.
- 705 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
706 [\[10\]](#) for a description of this attribute.

707

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

- 709 • **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the  
710 operations provided by the service. The interface is described by an **interface element**  
711 which is a child element of the service element. If no interface is specified, then the  
712 interface specified for the service in the componentType of the implementation is in effect.  
713 If a <service/> element has an interface subelement specified, the interface MUST provide  
714 a compatible subset of the interface declared on the componentType of the  
715 implementation [ASM50004] For details on the interface element see [the Interface section](#).
- 716 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as  
717 children. If no binding elements are specified for the service, then the bindings specified  
718 for the equivalent service in the componentType of the implementation MUST be used, but  
719 if the componentType also has no bindings specified, then <binding.sca/> MUST be used  
720 as the binding. If binding elements are specified for the service, then those bindings MUST  
721 be used and they override any bindings specified for the equivalent service in the  
722 componentType of the implementation. [ASM50005] Details of the binding element are  
723 described in [the Bindings section](#). The binding, combined with any PolicySets in effect for  
724 the binding, needs to satisfy the set of policy intents for the service, as described in [the](#)  
725 [Policy Framework specification \[10\]](#).
- 726 • **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback**  
727 element used if the interface has a callback defined, which has one or more **binding**  
728 elements as children. The **callback** and its binding child elements are specified if there is  
729 a need to have binding details used to handle callbacks. If the callback element is present  
730 and contains one or more binding child elements, then those bindings MUST be used for  
731 the callback. [ASM50006] If the callback element is not present, the behaviour is runtime  
732 implementation dependent.

733

## 734 5.3 Reference

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

```
739  
740 <?xml version="1.0" encoding="UTF-8"?>  
741 <!-- Component Reference schema snippet -->  
742 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
743   ...  
744   <component ... >*  
745     <implementation ... />?  
746     <service ... />*  
747     <reference name="xs:NCName"  
748       target="list of xs:anyURI"? autowire="xs:boolean"?  
749       multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
750       wiredByImpl="xs:boolean"? requires="list of xs:QName"?  
751       policySets="list of xs:QName"?>*  
752     <interface ... />?  
753     <binding uri="xs:anyURI"? requires="list of xs:QName"?  
754       policySets="list of xs:QName"? />*  
755     <callback?>  
756       <binding ... />+  
757     </callback>  
758   </reference>  
759   <property ... />*  
760 </component>  
761   ...  
762 </composite>  
763
```

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

- 765 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service  
766 element of a <component/> MUST be unique amongst the service elements of that  
767 <component/> [ASM50007] The @name attribute of a reference element of a  
768 <component/> MUST match the @name attribute of a reference element of the  
769 componentType of the <implementation/> child element of the component, [ASM50008]
- 770 • **autowire : boolean (0..1)** – whether the reference should be autowired, as described in  
771 the [Autowire section](#). Default is false.
- 772 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
773 [\[10\]](#) for a description of this attribute.  
774 Note: The effective set of policy intents for the reference consists of any intents explicitly  
775 stated in this requires attribute, combined with any intents specified for the reference by  
776 the implementation.
- 777 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
778 [\[10\]](#) for a description of this attribute.

**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.

- 779 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect  
780 the reference to target services. Overrides the multiplicity specified for this reference in  
781 the componentType of the implementation. The multiplicity can have the following values
- 782     o 0..1 – zero or one wire can have the reference as a source
- 783     o 1..1 – one wire can have the reference as a source
- 784     o 0..n - zero or more wires can have the reference as a source
- 785     o 1..n – one or more wires can have the reference as a source

786 The value of multiplicity for a component reference MUST only be equal or further restrict  
787 any value for the multiplicity of the reference with the same name in the componentType  
788 of the implementation, where further restriction means 0..n to 0..1 or 1..n to 1..1.  
789 [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.

790 If not present, the value of multiplicity is equal to the multiplicity specified for this  
791 reference in the componentType of the implementation - if not present in the  
792 componentType, the value defaults to 1..1.

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

804

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

- 806 • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes  
807 the operations required by the reference. The interface is described by an **interface**  
808 **element** which is a child element of the reference element. If no interface is specified,  
809 then the interface specified for the reference in the componentType of the implementation  
810 is in effect. If an interface is declared for a component reference it MUST provide a  
811 compatible superset of the interface declared for the equivalent reference in the  
812 componentType of the implementation, i.e. provide the same operations or a superset of  
813 the operations defined by the implementation for the reference. [\[ASM50011\]](#) For details  
814 on the interface element see [the Interface section](#).

- 815 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
816 children. If no binding elements are specified for the reference, then the bindings specified  
817 for the equivalent reference in the componentType of the implementation MUST be used,  
818 but if the componentType also has no bindings specified, then <binding.sca/> MUST be  
819 used as the binding. If binding elements are specified for the reference, then those  
820 bindings MUST be used and they override any bindings specified for the equivalent  
821 reference in the componentType of the implementation. [\[ASM50012\]](#) Details of the binding  
822 element are described in the [Bindings section](#). The binding, combined with any PolicySets  
823 in effect for the binding, needs to satisfy the set of policy intents for the reference, as  
824 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.

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

- 828 • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional  
829 **callback** element used if the interface has a callback defined, which has one or more  
830 **binding** elements as children. The **callback** and its binding child elements are specified if

831 there is a need to have binding details used to handle callbacks. If the callback element is  
832 present and contains one or more binding child elements, then those bindings MUST be  
833 used for the callback. [ASM50006] If the callback element is not present, the behaviour is  
834 runtime implementation dependent.

### 835 5.3.1 Specifying the Target Service(s) for a Reference

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

- 838 1. Through a value specified in the @target attribute of the reference element
- 839 2. Through a target URI specified in the @uri attribute of a binding element which is a child  
840 of the reference element
- 841 3. Through the setting of one or more values for binding-specific attributes and/or child  
842 elements of a binding element that is a child of the reference element
- 843 4. Through the specification of @autowire="true" for the reference (or through inheritance  
844 of that value from the component or composite containing the reference)
- 845 5. Through the specification of @wiredByImpl="true" for the reference
- 846 6. Through the promotion of a component reference by a composite reference of the  
847 composite containing the component (the target service is then identified by the  
848 configuration of the composite reference)

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

- 850 • If @wiredByImpl="true", other methods of specifying the target service MUST NOT be  
851 used. [ASM50013]
- 852 • If @autowire="true", the autowire procedure MUST only be used if no target is identified  
853 by any of the other ways listed above. It is not an error if @autowire="true" and a target  
854 is also defined through some other means, however in this case the autowire procedure  
855 MUST NOT be used. [ASM50014]
- 856 • If a reference has a value specified for one or more target services in its @target attribute,  
857 the child binding elements of that reference MUST NOT identify target services using the  
858 @uri attribute or using binding specific attributes or elements. [ASM50026]
- 859 • If a binding element has a value specified for a target service using its @uri attribute, the  
860 binding element MUST NOT identify target services using binding specific attributes or  
861 elements. [ASM50015]
- 862 • It is possible that a particular binding type MAY require that the address of a target service  
863 uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to  
864 identify the target service - instead, binding specific attributes and/or child elements must  
865 be used. [ASM50016]
- 866 • When the reference has a value specified in its @target attribute, one of the child binding  
867 elements MUST be used on each wire created by the @target attribute, or the sca binding,  
868 if no binding is specified. [ASM50017]

#### 869 5.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

- 871 • A reference with multiplicity 0..1 or 0..n MAY have no target service defined. [ASM50018]
- 872 • A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service  
873 defined. [ASM50019]
- 874 • A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.  
875 [ASM50020]
- 876 • A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.  
877 [ASM50021]

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

881 Some reference multiplicity errors can be detected at deployment time. In these cases, an error  
882 SHOULD be generated by the SCA runtime at deployment time. [ASM50023] For example, where  
883 a composite is used as a component implementation, wires and target services cannot be added to  
884 the composite after deployment. As a result, for components which are part of the composite,  
885 both missing wires and wires with a non-existent target can be detected at deployment time  
886 through a scan of the contents of the composite.

887 Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA  
888 runtime MUST generate an error no later than when the reference is invoked by the component  
889 implementation. [ASM50024] Examples include cases of components deployed to the SCA  
890 Domain. At the Domain level, the target of a wire, or even the wire itself, may form part of a  
891 separate deployed contribution and as a result these may be deployed after the original  
892 component is deployed. For the cases where it is valid for the reference to have no target service  
893 specified, the component implementation language specification needs to define the programming  
894 model for interacting with an untargetted reference.

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

## 900 5.4 Property

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

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

- 908 • As a value, supplied in the **value** attribute of the property element.  
909 If the @value attribute of a component property element is declared, the type of the  
910 property MUST be an XML Schema simple type and the @value attribute MUST contain a  
911 single value of that type. [ASM50027]

912 For example,

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

- 914 • As a value, supplied as the content of the **value** element(s) children of the property  
915 element.  
916 If the value subelement of a component property is specified, the type of the property  
917 MUST be an XML Schema simple type or an XML schema complex type. [ASM50028]

918 For example,

- 919 • property defined using a XML Schema simple type and which contains a single  
920 value

```
921 <property name="pi">  
922 <value>3.14159265</value>  
923 </property>
```

- 924 • property defined using a XML Schema simple type and which contains multiple  
925 values

```
926 <property name="currency">
```

927                   <value>EURO</value>

928                   <value>USDollar</value>

929                   </property>

- 930                   • property defined using a XML Schema complex type and which contains a single
 931                   value
 932                   <property name="complexFoo">
 933                    <value attr="bar">
 934                     <foo:a>TheValue</foo:a>
 935                     <foo:b>InterestingURI</foo:b>
 936                    </value>
 937                   </property>
- 938                   • property defined using a XML Schema complex type and which contains multiple
 939                   values
 940                   <property name="complexBar">
 941                    <value anotherAttr="foo">
 942                     <bar:a>AValue</bar:a>
 943                     <bar:b>InterestingURI</bar:b>
 944                    </value>
 945                    <value attr="zing">
 946                     <bar:a>BValue</bar:a>
 947                     <bar:b>BoringURI</bar:b>
 948                    </value>
 949                   </property>
- 950                   • As a value, supplied as the content of the property element.
 951                   If a component property value is declared using a child element of the <property/>
 952                   element, the type of the property MUST be an XML Schema global element and the
 953                   declared child element MUST be an instance of that global element. [\[ASM50029\]](#)

954                   For example,

  - 955                    • property defined using a XML Schema global element declaration and which
 956                    contains a single value
 957                    <property name="foo">
 958                     <foo:SomeGED ...>...</foo:SomeGED>
 959                    </property>
  - 960                    • property defined using a XML Schema global element declaration and which
 961                    contains multiple values
 962                    <property name="bar">
 963                     <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
 964                     <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
 965                    </property>
- 966                   • By referencing a Property value of the composite which contains the component. The
 967                   reference is made using the **source** attribute of the property element.
 968                   The form of the value of the source attribute follows the form of an XPath expression.
 969                   This form allows a specific property of the composite to be addressed by name. Where the
 970

971 composite property is of a complex type, the XPath expression can be extended to refer to  
972 a sub-part of the complex property value.

973  
974 So, for example, `source="$currency"` is used to reference a property of the composite  
975 called "currency", while `source="$currency/a"` references the sub-part "a" of the  
976 complex composite property with the name "currency".

977  
978 Note that the source attribute refers to the contents of a composite after the processing of  
979 all <include/> elements. It is possible for @source to refer to a composite property that is  
980 not contained in the same physical file as the component property element. The  
981 requirement is that the composite property and the component are brought together  
982 through include processing when the component is deployed into the SCA Domain.

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

985

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

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

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

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

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

997 The property type specified must be compatible with the type of the property declared in the  
998 component type of the implementation. If no type is declared in the component property, the type of  
999 the property declared by the implementation is used.

1000

1001 The following snippet shows the component schema with the schema for a property child element:

1002

```
1003 <?xml version="1.0" encoding="UTF-8"?>
1004 <!-- Component Property schema snippet -->
1005 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1006   ...
1007   <component ... >*
1008     <implementation ... />?
1009     <service ... />*
1010     <reference ... />*
1011     <property name="xs:NCName"
1012       (type="xs:QName" | element="xs:QName")?
1013       mustSupply="xs:boolean"? many="xs:boolean"?
1014       source="xs:string"? file="xs:anyURI"?
1015       value="xs:string"?>*
1016       [<value>+ | xs:any+ ]?
1017     </property>
```

1018 </component>  
1019 ...  
1020 </composite>

1021 The **component property** element has the following **attributes**:

- 1023 ▪ **name : NCName (1..1)** – the name of the property. The name attribute of a component  
1024 property MUST match the name of a property element in the component type of the  
1025 component implementation. [ASM50031]
- 1026 ▪ zero or one of (**0..1**):
  - 1027 ○ **type : QName** – the type of the property defined as the qualified name of an XML  
1028 schema type
  - 1029 ○ **element : QName** – the type of the property defined as the qualified name of an  
1030 XML schema global element – the type is the type of the global element
- 1031 ▪ **source : string (0..1)** – an XPath expression pointing to a property of the containing  
1032 composite ([post include processing](#)) from which the value of this component property is  
1033 obtained.
- 1034 ▪ **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
- 1035 ▪ **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or  
1036 multi-valued (true). Overrides the many specified for this property on the implementation.  
1037 The value can only be equal or further restrict, i.e. if the implementation specifies many  
1038 true, then the component can say false. In the case of a multi-valued property, it is  
1039 presented to the implementation as a Collection of property values.
- 1040 ▪ **value : string (0..1)** – the value of the property if the property is defined using a simple  
1041 type.

**Comment [ME5]:** Issue 62  
- mustSupply removed in  
accordance with the  
resolution.

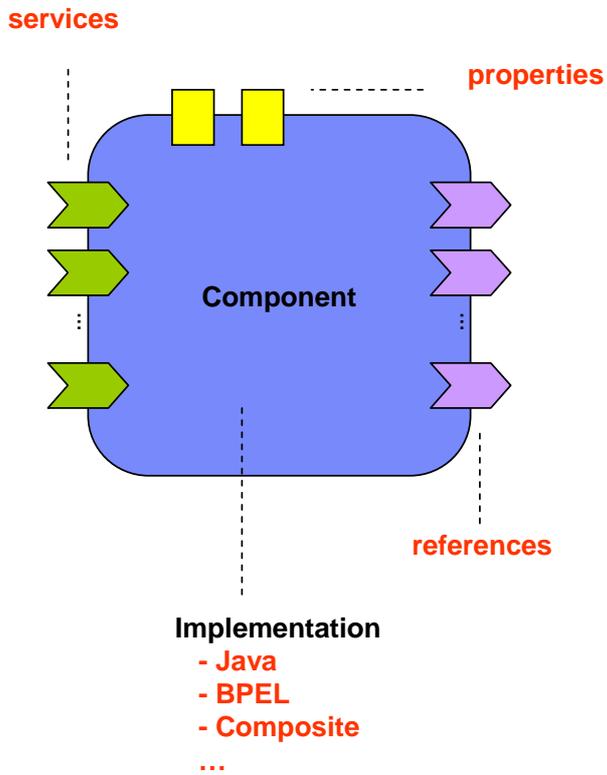
1042 The **component property** element has the following **child element**:

1043 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1044 property that is defined using a XML Schema type. If a property is single-valued, the <value/>  
1045 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST  
1046 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

## 1047 5.5 Example Component

1048

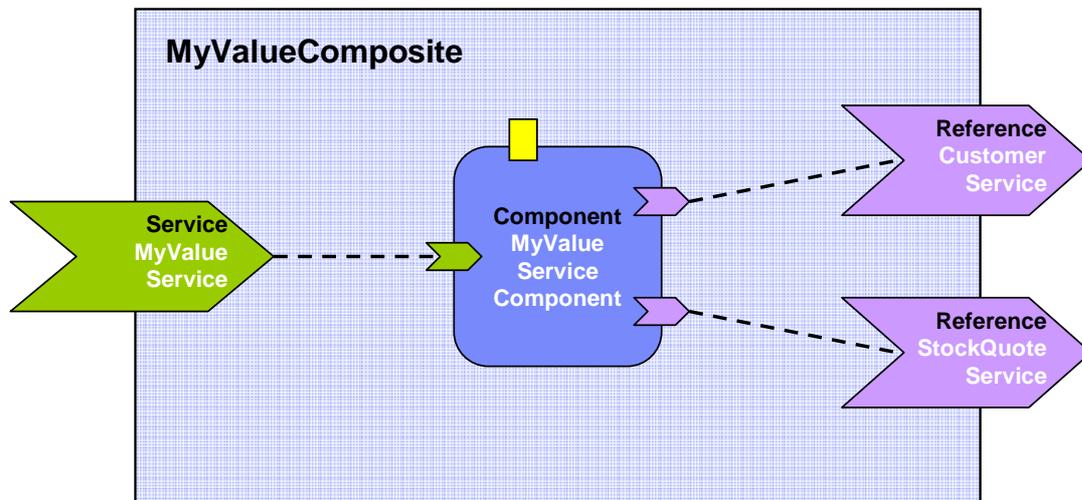
1049 The following figure shows the **component symbol** that is used to represent a component in an  
1050 assembly diagram.



1051  
1052  
1053  
1054  
1055

Figure 5: Component symbol

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



1056  
1057  
1058

Figure 6: Assembly diagram for MyValueComposite

1059

1060 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1061 containing the component element for the MyValueServiceComponent. A value is set for the  
1062 property named currency, and the customerService and stockQuoteService references are  
1063 promoted:

1064

```
1065 <?xml version="1.0" encoding="ASCII"?>
1066 <!-- MyValueComposite_1 example -->
1067 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1068           targetNamespace="http://foo.com"
1069           name="MyValueComposite" >
1070
1071     <service name="MyValueService" promote="MyValueServiceComponent" />
1072
1073     <component name="MyValueServiceComponent">
1074       <implementation.java
1075 class="services.myvalue.MyValueServiceImpl" />
1076       <property name="currency">EURO</property>
1077       <reference name="customerService" />
1078       <reference name="stockQuoteService" />
1079     </component>
1080
1081     <reference name="CustomerService"
1082           promote="MyValueServiceComponent/customerService" />
1083
1084     <reference name="StockQuoteService"
1085           promote="MyValueServiceComponent/stockQuoteService" />
1086
1087 </composite>
```

1088

1089 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of  
1090 clarity – the references are defined by the MyValueServiceImpl implementation and there is no  
1091 need to redeclare them on the component unless the intention is to wire them or to override some  
1092 aspect of them.

1093 The following snippet gives an example of the layout of a composite file if both the currency  
1094 property and the customerService reference of the MyValueServiceComponent are declared to be  
1095 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1096 <?xml version="1.0" encoding="ASCII"?>
1097 <!-- MyValueComposite_2 example -->
1098 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1099           targetNamespace="http://foo.com"
1100           name="MyValueComposite" >
1101
1102     <service name="MyValueService" promote="MyValueServiceComponent" />
```

1103

```
1104     <component name="MyValueServiceComponent" >
1105         <implementation.java
1106 class="services.myvalue.MyValueServiceImpl" />
1107         <property name="currency">EURO</property>
1108         <property name="currency">Yen</property>
1109         <property name="currency">USDollar</property>
1110         <reference name="customerService"
1111             target="InternalCustomer/customerService" />
1112         <reference name="StockQuoteService" />
1113     </component>
1114
1115     ...
1116
1117     <reference name="CustomerService"
1118         promote="MyValueServiceComponent/customerService" />
1119
1120     <reference name="StockQuoteService"
1121         promote="MyValueServiceComponent/StockQuoteService" />
1122
1123 </composite>
```

1124  
1125 ....this assumes that the composite has another component called InternalCustomer (not shown)  
1126 which has a service to which the customerService reference of the MyValueServiceComponent is  
1127 wired as well as being promoted externally through the composite reference CustomerService.

## 1128 6 Composite

1129 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of  
1130 composition within an SCA Domain. An **SCA composite** contains a set of components, services,  
1131 references and the wires that interconnect them, plus a set of properties which can be used to  
1132 configure components.

1133 Composites can be used as **component implementations** in higher-level composites – in other  
1134 words the higher-level composites can have components that are implemented by composites.  
1135 For more detail on the use of composites as component implementations see the section [Using](#)  
1136 [Composites as Component Implementations](#).

1137 The content of a composite can be used within another composite through **inclusion**. When a  
1138 composite is included by another composite, all of its contents are made available for use within  
1139 the including composite – the contents are fully visible and can be referenced by other elements  
1140 within the including composite. For more detail on the inclusion of one composite into another see  
1141 the section [Using Composites through Inclusion](#).

1142 A composite can be used as a unit of deployment. When used in this way, composites contribute  
1143 elements to an SCA domain. A composite can be deployed to the SCA domain either by inclusion,  
1144 or a composite can be deployed to the domain as an implementation. For more detail on the  
1145 deployment of composites, see the section dealing with the [SCA Domain](#).

1146

1147 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite**  
1148 element. The following snippet shows the schema for the composite element.

1149

```
1150 <?xml version="1.0" encoding="ASCII"?>  
1151 <!-- Composite schema snippet -->  
1152 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
1153     targetNamespace="xs:anyURI"  
1154     name="xs:NCName" local="xs:boolean"?  
1155     autowire="xs:boolean"? constrainingType="QName"?  
1156     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1157  
1158     <include ... />*  
1159  
1160     <service ... />*  
1161     <reference ... />*  
1162     <property ... />*  
1163  
1164     <component ... />*  
1165  
1166     <wire ... />*  
1167  
1168 </composite>
```

1169

1170

1171

1172 The **composite** element has the following **attributes**:

- 1173 • **name : QName (1..1)** – the name of the composite. The form of a composite name is  
1174 an XML QName, in the namespace identified by the targetNamespace attribute. A  
1175 composite name must be unique within the namespace of the composite. [ASM60001]
- 1176 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the  
1177 composite is declared
- 1178 • **local : boolean (0..1)** – whether all the components within the composite all run in the  
1179 same operating system process. @local="true" for a composite means that all the  
1180 components within the composite MUST run in the same operating system process.  
1181 [ASM60002] local="false", which is the default, means that different components within  
1182 the composite can run in different operating system processes and they can even run on  
1183 different nodes on a network.
- 1184 • **autowire : boolean (0..1)** – whether contained component references should be  
1185 autowired, as described in the [Autowire section](#). Default is false.
- 1186 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
1187 the set of services, references and properties of the composite, plus related intents, is  
1188 constrained to the set defined by the constrainingType. See the [ConstrainingType Section](#)  
1189 for more details.
- 1190 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)  
1191 [specification \[10\]](#) for a description of this attribute.
- 1192 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1193 [\[10\]](#) for a description of this attribute.

1194

1195 The **composite** element has the following **child elements**:

- 1196 • **service : CompositeService (0..n)** – see composite service section.
- 1197 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1198 • **property : CompositeProperty (0..n)** – see composite property section.
- 1199 • **component : Component (0..n)** – see component section.
- 1200 • **wire : Wire (0..n)** – see composite wire section.
- 1201 • **include : Include (0..n)** – see composite include section

1202

1203 Components contain configured implementations which hold the business logic of the composite.  
1204 The components offer services and require references to other services. **Composite services**  
1205 define the public services provided by the composite, which can be accessed from outside the  
1206 composite. **Composite references** represent dependencies which the composite has on services  
1207 provided elsewhere, outside the composite. Wires describe the connections between component  
1208 services and component references within the composite. Included composites contribute the  
1209 elements they contain to the using composite.

1210 Composite services involve the **promotion** of one service of one of the components within the  
1211 composite, which means that the composite service is actually provided by one of the components  
1212 within the composite. Composite references involve the **promotion** of one or more references of  
1213 one or more components. Multiple component references can be promoted to the same composite  
1214 reference, as long as all the component references are compatible with one another. Where  
1215 multiple component references are promoted to the same composite reference, then they all share  
1216 the same configuration, including the same target service(s).

1217 Composite services and composite references can use the configuration of their promoted services  
1218 and references respectively (such as Bindings and Policy Sets). Alternatively composite services  
1219 and composite references can override some or all of the configuration of the promoted services  
1220 and references, through the configuration of bindings and other aspects of the composite service  
1221 or reference.

1222 Component services and component references can be promoted to composite services and  
1223 references and also be wired internally within the composite at the same time. For a reference,  
1224 this only makes sense if the reference supports a multiplicity greater than 1.

1225

## 1226 6.1 Service

1227 The **services of a composite** are defined by promoting services defined by components  
1228 contained in the composite. A component service is promoted by means of a composite **service**  
1229 **element**.

1230 A composite service is represented by a **service element** which is a child of the composite  
1231 element. There can be **zero or more** service elements in a composite. The following snippet  
1232 shows the composite schema with the schema for a service child element:

1233

```
1234 <?xml version="1.0" encoding="ASCII"?>
1235 <!-- Composite Service schema snippet -->
1236 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1237   ...
1238   <service name="xs:NCName" promote="xs:anyURI"
1239     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1240     <interface ... />?
1241     <binding ... />*
1242     <callback?
1243       <binding ... />+
1244     </callback>
1245   </service>
1246   ...
1247 </composite>
```

1248

1249 The **composite service** element has the following **attributes**:

- 1250 • **name : NCName (1..1)** – the name of the service. The name of a composite <service/>  
1251 element MUST be unique across all the composite services in the composite. [ASM60003]  
1252 The name of the composite service can be different from the name of the promoted  
1253 component service.
- 1254 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form  
1255 <component-name>/<service-name>. The service name is optional if the target  
1256 component only has one service. The same component service can be promoted by more  
1257 than one composite service. A composite <service/> element's promote attribute MUST  
1258 identify one of the component services within that composite. [ASM60004]  
1259 Note that the promote attribute refers to the contents of a composite after the processing  
1260 of all <include/> elements. It is possible for @promote to refer to a component that is not  
1261 contained in the same physical file as the composite service element. The requirement is  
1262 that the composite service and the referenced component are brought together through  
1263 include processing when the composite service is deployed into the SCA Domain.
- 1264 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1265 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1266 further qualify the required intents defined by the promoted component service.
- 1267 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1268 [\[10\]](#) for a description of this attribute.

1269

1270  
1271

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

1272  
1273  
1274  
1275  
1276  
1277

- **interface : Interface (0..1)** - 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. [ASM60005] The interface is described by **zero or one interface element** which is a child element of the service element. For details on the interface element see [the Interface 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.

1278  
1279  
1280  
1281  
1282  
1283

- **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](#).

1284  
1285  
1286  
1287  
1288

- **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.

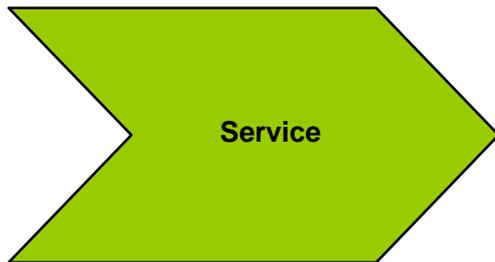
1289

## 1290 6.1.1 Service Examples

1291

1292  
1293

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



1294  
1295

Figure 7: Service symbol

1296

1297  
1298

The following figure shows the assembly diagram for the MyValueComposite containing the service MyValueService.

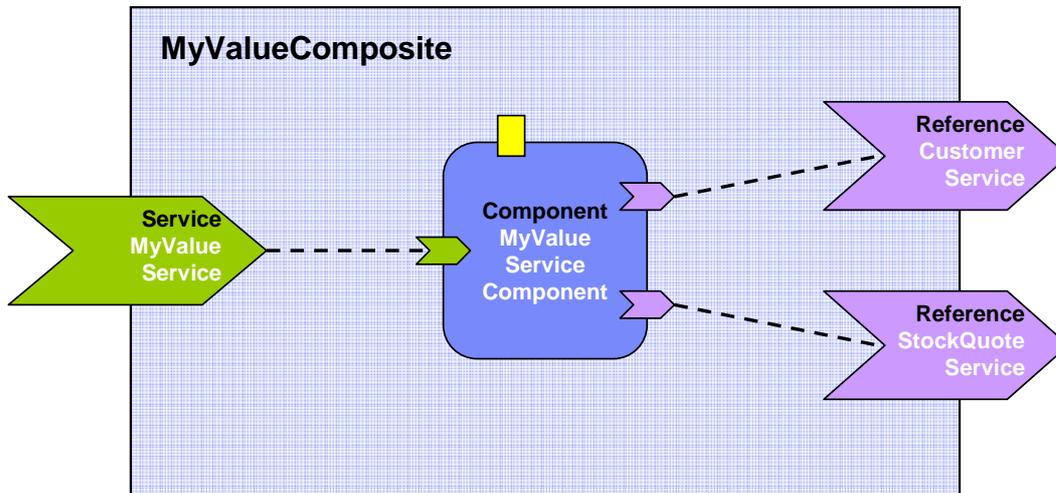


Figure 8: MyValueComposite showing Service

1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329

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>
```

1330  
1331       ...  
1332  
1333       </composite>  
1334

## 1335 6.2 Reference

1336       The **references of a composite** are defined by **promoting** references defined by components  
1337       contained in the composite. Each promoted reference indicates that the component reference  
1338       needs to be resolved by services outside the composite. A component reference is promoted using  
1339       a composite **reference element**.

1340       A composite reference is represented by a **reference element** which is a child of a composite  
1341       element. There can be **zero or more** reference elements in a composite. The following snippet  
1342       shows the composite schema with the schema for a **reference** element.

```
1343  
1344       <?xml version="1.0" encoding="ASCII"?>  
1345       <!-- Composite Reference schema snippet -->  
1346       <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1347       ...  
1348       <reference name="xs:NCName" target="list of xs:anyURI"?  
1349                promote="list of xs:anyURI" wiredByImpl="xs:boolean"?  
1350                multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
1351                requires="list of xs:QName"? policySets="list of xs:QName"?>*  
1352       <interface ... />?  
1353       <binding ... />*  
1354       <callback?>  
1355                <binding ... />+  
1356       </callback>  
1357       </reference>  
1358       ...  
1359       </composite>
```

1362       The **composite reference** element has the following **attributes**:

- 1363       • **name : NCName (1..1)** – the name of the reference. The name of a composite  
1364        <reference/> element MUST be unique across all the composite references in the  
1365        composite. [ASM60006] The name of the composite reference can be different then the  
1366        name of the promoted component reference.
- 1367       • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
1368        value is a list of values of the form <component-name>/<reference-name> separated by  
1369        spaces. The specification of the reference name is optional if the component has only one  
1370        reference. Each of the URIs declared by a composite reference's @promote attribute MUST  
1371        identify a component reference within the composite. [ASM60007]  
1372        Note that the promote attribute refers to the contents of a composite after the processing  
1373        of all <include/> elements. It is possible for @promote to refer to a component that is not  
1374        contained in the same physical file as the composite reference element. The requirement  
1375        is that the composite reference and the referenced component are brought together  
1376        through include processing when the composite service is deployed into the SCA Domain.

1377 The same component reference can be promoted more than once, using different  
1378 composite references, but only if the multiplicity defined on the component reference is  
1379 0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1380 Where a composite reference promotes two or more component references:

- 1381 • the interfaces of the component references promoted by a composite reference  
1382 MUST be the same, or if the composite reference itself declares an interface then  
1383 all the component reference interfaces must be compatible with the composite  
1384 reference interface. Compatible means that the component reference interface is  
1385 the same or is a strict subset of the composite reference interface. [ASM60008]
- 1386 • the intents declared on a composite reference and on the component references  
1387 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents  
1388 which apply to the composite reference in this case are the union of the required  
1389 intents specified for each of the promoted component references plus any intents  
1390 declared on the composite reference itself. If any intents in the set which apply to  
1391 a composite reference are mutually exclusive then the SCA runtime MUST raise an  
1392 error. [ASM60010]

1393 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1394 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1395 further qualify the required intents defined for the promoted component reference.

1396 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1397 [\[10\]](#) for a description of this attribute.

1398 • **multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can  
1399 connect the reference to target services. The multiplicity can have the following values

- 1400 ○ 0..1 – zero or one wire can have the reference as a source
- 1401 ○ 1..1 – one wire can have the reference as a source
- 1402 ○ 0..n - zero or more wires can have the reference as a source
- 1403 ○ 1..n – one or more wires can have the reference as a source

1404 The value specified for the **multiplicity** attribute of a composite reference MUST be  
1405 compatible with the multiplicity specified on each of the promoted component references,  
1406 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used  
1407 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be  
1408 used where the promoted component reference has multiplicity 0..n or 1..n and  
1409 multiplicity 1..n can be used where the promoted component reference has multiplicity  
1410 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to  
1411 promote a component reference of multiplicity 0..1 or 1..1 respectively, [ASM60011]

1412 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
1413 multiplicity setting. Each value wires the reference to a service in a composite that uses  
1414 the composite containing the reference as an implementation for one of its components. For  
1415 more details on wiring see [the section on Wires](#).

1416 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
1417 the implementation wires this reference dynamically. If set to "true" it indicates that the  
1418 target of the reference is set at runtime by the implementation code (eg by the code  
1419 obtaining an endpoint reference by some means and setting this as the target of the  
1420 reference through the use of programming interfaces defined by the relevant Client and  
1421 Implementation specification). If "true" is set, then the reference should not be wired  
1422 statically within a using composite, but left unwired.

1423  
1424 The **composite reference** element has the following **child elements**, whatever is not specified is  
1425 defaulted from the promoted component reference(s).

1426 • **interface : Interface (0..1)** - **zero or one interface element** which declares an  
1427 interface for the composite reference. If a composite reference has an **interface** specified,

**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.

**Comment [ME6]:** Need to consider this as a normative statement

**Comment [ME7]:** Need to raise an issue to remove this attribute from composite references since there is not code to do the wiring, unlike a component reference.

1428 | it MUST provide an interface which is the same or which is a compatible superset of the  
1429 | interface(s) declared by the promoted component reference(s), i.e. provide a superset of  
1430 | the operations in the interface defined by the component for the reference. [ASM60012] If  
1431 | no interface is declared on a composite reference, the interface from one of its promoted  
1432 | component references is used, which MUST be the same as or a compatible superset of  
1433 | the interface(s) declared by the promoted component reference(s).  
1434 | [ASM60013] For details on the interface element see the Interface section.

**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.

1435 | • **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as  
1436 | children. If one or more **bindings** are specified they **override** any and all of the bindings  
1437 | defined for the promoted component reference from the composite reference perspective.  
1438 | The bindings defined on the component reference are still in effect for local wires within  
1439 | the composite that have the component reference as their source. Details of the binding  
1440 | element are described in the Bindings section. For more details on wiring see the section  
1441 | on Wires.

1442 | A reference identifies zero or more target services which satisfy the reference. This can be  
1443 | done in a number of ways, which are fully described in section "5.3.1 Specifying the  
1444 | Target Service(s) for a Reference".

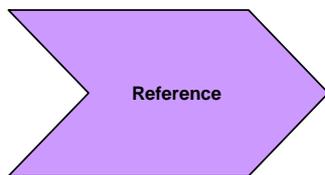
1445 | • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional  
1446 | **callback** element used if the interface has a callback defined, which has one or more  
1447 | **binding** elements as children. The **callback** and its binding child elements are specified if  
1448 | there is a need to have binding details used to handle callbacks. If the callback element is  
1449 | not present, the behaviour is runtime implementation dependent.

1450 |

## 1451 | 6.2.1 Example Reference

1452 |

1453 | The following figure shows the reference symbol that is used to represent a reference in an  
1454 | assembly diagram.



1455 |

1456 | Figure 9: Reference symbol

1457 |

1458 | The following figure shows the assembly diagram for the MyValueComposite containing the  
1459 | reference CustomerService and the reference StockQuoteService.

1460 |

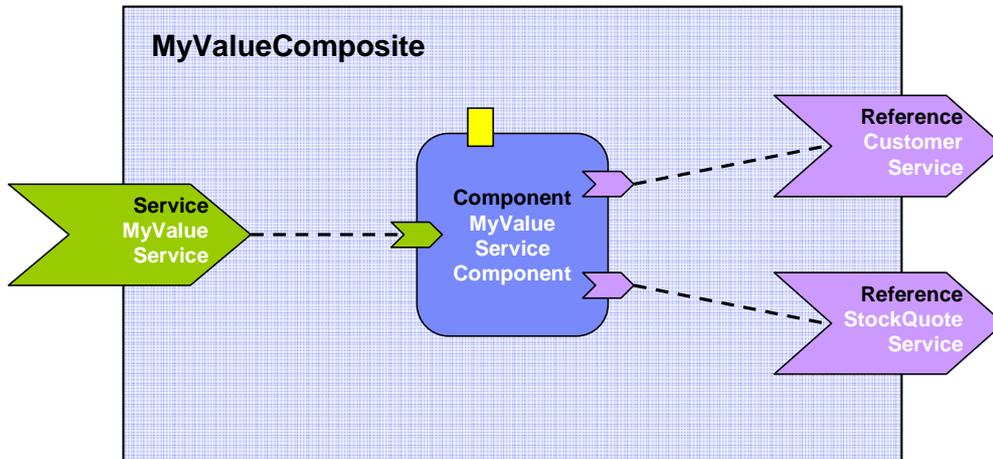


Figure 10: MyValueComposite showing References

1461

1462

1463

1464 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
 1465 containing the reference elements for the CustomerService and the StockQuoteService. The  
 1466 reference CustomerService is bound using the SCA binding. The reference StockQuoteService is  
 1467 bound using the Web service binding. The endpoint addresses of the bindings can be specified, for  
 1468 example using the binding **uri** attribute (for details see the [Bindings](#) section), or overridden in an  
 1469 enclosing composite. Although in this case the reference StockQuoteService is bound to a Web  
 1470 service, its interface is defined by a Java interface, which was created from the WSDL portType of  
 1471 the target web service.

1472

1473

```

1474 <?xml version="1.0" encoding="ASCII"?>
1475 <!-- MyValueComposite_3 example -->
1476 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1477           targetNamespace="http://foo.com"
1478           name="MyValueComposite" >
1479     ...
1480
1481     <component name="MyValueServiceComponent">
1482       <implementation.java
1483 class="services.myvalue.MyValueServiceImpl"/>
1484       <property name="currency">EURO</property>
1485       <reference name="customerService"/>
1486       <reference name="StockQuoteService"/>
1487     </component>
1488
1489     <reference name="CustomerService"
1490       promote="MyValueServiceComponent/customerService">
1491       <interface.java interface="services.customer.CustomerService"/>
1492       <!-- The following forces the binding to be binding.sca whatever
1493 is -->

```

```

1494         <!-- specified by the component reference or by the underlying
1495 -->
1496         <!-- implementation
1497 -->
1498         <binding.sca/>
1499     </reference>
1500
1501     <reference name="StockQuoteService"
1502         promote="MyValueServiceComponent/StockQuoteService">
1503         <interface.java
1504 interface="services.stockquote.StockQuoteService"/>
1505         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1506 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1507     </reference>
1508
1509     ...
1510
1511
1512 </composite>
1513

```

### 6.3 Property

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

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

```

1522 <?xml version="1.0" encoding="ASCII"?>
1523 <!-- Composite Property schema snippet -->
1524 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1525     ...
1526     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1527         many="xs:boolean"? mustSupply="xs:boolean"?>*
1528         default-property-value?
1529     </property>
1530     ...
1531 </composite>
1532

```

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

- **name : NCName (1..1)** - the name of the property. **The name attribute of a composite property MUST be unique amongst the properties of the same composite.**[ASM60014]
- one of **(1..1)**:
  - **type : QName** - the type of the property - the qualified name of an XML schema type

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

- 1539           o **element : QName** – the type of the property defined as the qualified name of an  
1540           XML schema global element – the type is the type of the global element
- 1541           ▪ **many : boolean (0..1)** – whether the property is single-valued (false) or multi-valued  
1542           (true). The default is **false**. In the case of a multi-valued property, it is presented to the  
1543           implementation as a collection of property values.
- 1544           ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the  
1545           component that uses the composite – when mustSupply="true" the component has to  
1546           supply a value since the composite has no default value for the property. A default-  
1547           property-value is only worth declaring when mustSupply="false" (the default setting for  
1548           the mustSupply attribute), since the implication of a default value is that it is used only  
1549           when a value is not supplied by the using component.

1550

1551 The property element may contain an optional **default-property-value**, which provides default  
1552 value for the property. The default value must match the type declared for the property:

- 1553           o a string, if **type** is a simple type (matching the **type** declared)
- 1554           o a complex type value matching the type declared by **type**
- 1555           o an element matching the element named by **element**
- 1556           o multiple values are permitted if many="true" is specified

1557

1558 Implementation types other than **composite** can declare properties in an implementation-  
1559 dependent form (eg annotations within a Java class), or through a property declaration of exactly  
1560 the form described above in a componentType file.

**Comment [ME8]:** I think that this paragraph should be removed.

1561 Property values can be configured when an implementation is used by a component. The form of  
1562 the property configuration is shown in [the section on Components](#).

### 1563 6.3.1 Property Examples

1564

1565 For the following example of Property declaration and value setting, the following complex type is  
1566 used as an example:

```
1567 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1568             targetNamespace="http://foo.com/"
1569             xmlns:tns="http://foo.com/">
1570   <!-- ComplexProperty schema -->
1571   <xsd:element name="fooElement" type="MyComplexType"/>
1572   <xsd:complexType name="MyComplexType">
1573     <xsd:sequence>
1574       <xsd:element name="a" type="xsd:string"/>
1575       <xsd:element name="b" type="anyURI"/>
1576     </xsd:sequence>
1577     <attribute name="attr" type="xsd:string" use="optional"/>
1578   </xsd:complexType>
1579 </xsd:schema>
```

1580

1581 The following composite demonstrates the declaration of a property of a complex type, with a  
1582 default value, plus it demonstrates the setting of a property value of a complex type within a  
1583 component:

```
1584 <?xml version="1.0" encoding="ASCII"?>
```

```

1585
1586 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1587           xmlns:foo="http://foo.com"
1588           targetNamespace="http://foo.com"
1589           name="AccountServices">
1590 <!-- AccountServices Example1 -->
1591
1592     ...
1593
1594     <property name="complexFoo" type="foo:MyComplexType">
1595         <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1596             <foo:a>AValue</foo:a>
1597             <foo:b>InterestingURI</foo:b>
1598         </MyComplexPropertyValue>
1599     </property>
1600
1601     <component name="AccountServiceComponent">
1602         <implementation.java class="foo.AccountServiceImpl"/>
1603         <property name="complexBar" source="$complexFoo"/>
1604         <reference name="accountDataService"
1605                 target="AccountDataServiceComponent"/>
1606         <reference name="stockQuoteService" target="StockQuoteService"/>
1607     </component>
1608
1609     ...
1610
1611 </composite>

```

1612 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the  
1613 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the  
1614 composite and it references the example XSD, where **MyComplexType** is defined. The declaration  
1615 of **complexFoo** contains a default value. This is declared as the content of the property element.  
1616 In this example, the default value consists of the element **MyComplexPropertyValue** of type  
1617 **foo:MyComplexType** and its two child elements **<foo:a>** and **<foo:b>**, following the definition of  
1618 **MyComplexType**.

1619 In the component **AccountServiceComponent**, the component sets the value of the property  
1620 **complexBar**, declared by the implementation configured by the component. In this case, the  
1621 type of **complexBar** is **foo:MyComplexType**. The example shows that the value of the **complexBar**  
1622 property is set from the value of the **complexFoo** property – the **source** attribute of the property  
1623 element for **complexBar** declares that the value of the property is set from the value of a property  
1624 of the containing composite. The value of the source attribute is **\$complexFoo**, where  
1625 **complexFoo** is the name of a property of the composite. This value implies that the whole of the  
1626 value of the source property is used to set the value of the component property.

1627 The following example illustrates the setting of the value of a property of a simple type (a string)  
1628 from **part** of the value of a property of the containing composite which has a complex type:

```

1629 <?xml version="1.0" encoding="ASCII"?>
1630
1631 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"

```

```

1632         xmlns:foo="http://foo.com"
1633         targetNamespace="http://foo.com"
1634         name="AccountServices">
1635 <!-- AccountServices Example2 -->
1636
1637     ...
1638
1639     <property name="complexFoo" type="foo:MyComplexType">
1640         <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1641             <foo:a>AValue</foo:a>
1642             <foo:b>InterestingURI</foo:b>
1643         </MyComplexPropertyValue>
1644     </property>
1645
1646     <component name="AccountServiceComponent">
1647         <implementation.java class="foo.AccountServiceImpl"/>
1648         <property name="currency" source="$complexFoo/a"/>
1649         <reference name="accountDataService"
1650             target="AccountDataServiceComponent"/>
1651         <reference name="stockQuoteService" target="StockQuoteService"/>
1652     </component>
1653
1654     ...
1655 </composite>
1656

```

1657 In this example, the component **AccountServiceComponent** sets the value of a property called  
1658 **currency**, which is of type string. The value is set from a property of the composite  
1659 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression  
1660 that selects the property name **complexFoo** and then selects the value of the **a** subelement of  
1661 complexFoo. The "a" subelement is a string, matching the type of the currency property.

1662 Further examples of declaring properties and setting property values in a component follow:

1663 Declaration of a property with a simple type and a default value:

```

1664 <property name="SimpleTypeProperty" type="xsd:string">
1665     MyValue
1666 </property>
1667

```

1668 Declaration of a property with a complex type and a default value:

```

1669 <property name="complexFoo" type="foo:MyComplexType">
1670     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1671         <foo:a>AValue</foo:a>
1672         <foo:b>InterestingURI</foo:b>
1673     </MyComplexPropertyValue>
1674 </property>
1675

```

1676 Declaration of a property with an element type:

```
1677 <property name="elementFoo" element="foo:fooElement">
1678   <foo:fooElement>
1679     <foo:a>AValue</foo:a>
1680     <foo:b>InterestingURI</foo:b>
1681   </foo:fooElement>
1682 </property>
```

1683

1684 Property value for a simple type:

```
1685 <property name="SimpleTypeProperty">
1686 MyValue
1687 </property>
```

1688

1689

1690 Property value for a complex type, also showing the setting of an attribute value of the complex  
1691 type:

```
1692 <property name="complexFoo">
1693   <MyComplexPropertyValue xsi:type="foo:MyComplexType" attr="bar">
1694     <foo:a>AValue</foo:a>
1695     <foo:b>InterestingURI</foo:b>
1696   </MyComplexPropertyValue>
1697 </property>
```

1698

1699 Property value for an element type:

```
1700 <property name="elementFoo">
1701   <foo:fooElement attr="bar">
1702     <foo:a>AValue</foo:a>
1703     <foo:b>InterestingURI</foo:b>
1704   </foo:fooElement>
1705 </property>
```

1706

1707 Declaration of a property with a complex type where multiple values are supported:

```
1708 <property name="complexFoo" type="foo:MyComplexType" many="true"/>
1709
```

1710 Setting of a value for that property where multiple values are supplied:

```
1711 <property name="complexFoo">
1712   <MyComplexPropertyValue1 xsi:type="foo:MyComplexType" attr="bar">
1713     <foo:a>AValue</foo:a>
1714     <foo:b>InterestingURI</foo:b>
1715   </MyComplexPropertyValue1>
1716   <MyComplexPropertyValue2 xsi:type="foo:MyComplexType" attr="zing">
1717     <foo:a>BValue</foo:a>
1718     <foo:b>BoringURI</foo:b>
```

1719       </MyComplexPropertyValue2>  
1720       </property>  
1721

## 1722 6.4 Wire

1723       **SCA wires** within a composite connect **source component references** to **target component**  
1724       **services**.

1725       One way of defining a wire is by **configuring a reference of a component using its target**  
1726       **attribute**. The reference element is configured with the wire-target-URI of the service(s) that  
1727       resolve the reference. Multiple target services are valid when the reference has a multiplicity of  
1728       0..n or 1..n.

1729       An alternative way of defining a Wire is by means of a **wire element** which is a child of the  
1730       composite element. There can be **zero or more** wire elements in a composite. This alternative  
1731       method for defining wires is useful in circumstances where separation of the wiring from the  
1732       elements the wires connect helps simplify development or operational activities. An example is  
1733       where the components used to build a domain are relatively static but where new or changed  
1734       applications are created regularly from those components, through the creation of new assemblies  
1735       with different wiring. Deploying the wiring separately from the components allows the wiring to  
1736       be created or modified with minimum effort.

1737       Note that a Wire specified via a wire element is equivalent to a wire specified via the target  
1738       attribute of a reference. The rule which forbids mixing of wires specified with the target attribute  
1739       with the specification of endpoints in binding subelements of the reference also applies to wires  
1740       specified via separate wire elements.

1741       The following snippet shows the composite schema with the schema for the reference elements of  
1742       components and composite services and the wire child element:

1743  
1744       <?xml version="1.0" encoding="ASCII"?>  
1745       <!-- Wires schema snippet -->  
1746       <composite       xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
1747                    targetNamespace="xs:anyURI"  
1748                    name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?  
1749                    constrainingType="QName"?  
1750                    requires="list of xs:QName"? policySets="list of  
1751       xs:QName"?>  
1752  
1753                ...  
1754  
1755                <wire source="xs:anyURI" target="xs:anyURI" /\*>  
1756  
1757       </composite>

**Comment [ME9]:** This psuedo-schema does not match the wording leading up to it...

1760       The **reference element of a component** and the **reference element of a service** has a list of  
1761       one or more of the following **wire-target-URI** values for the target, with multiple values  
1762       separated by a space:

- 1763
  - <component-name>/<service-name>

1764                   o where the target is a service of a component. The specification of the service  
1765                    name is optional if the target component only has one service with a compatible  
1766                    interface

1767

1768   The **wire element** has the following attributes:

- 1769   • **source (1..1)** – names the source component reference. Valid URI schemes are:
- 1770       o `<component-name>/<reference-name>`
- 1771           ▪ where the source is a component reference. The specification of the  
1772            reference name is optional if the source component only has one reference
- 1773   • **target (1..1)** – names the target component service. Valid URI schemes are
- 1774       o `<component-name>/<service-name>`
- 1775           ▪ where the target is a service of a component. The specification of the  
1776            service name is optional if the target component only has one service with  
1777            a compatible interface

1778   For a composite used as a component implementation, wires can only link sources and targets  
1779   that are contained in the same composite (irrespective of which file or files are used to describe  
1780   the composite). Wiring to entities outside the composite is done through services and references  
1781   of the composite with wiring defined by the next higher composite.

1782   Note that the source and target attributes refer to the contents of a composite after the processing  
1783   of all <include/> elements. It is possible for these attributes to refer to a component that is not  
1784   contained in the same physical file as the wire element. The requirement is that the wire and the  
1785   referenced component(s) are brought together through include processing when the wire is  
1786   deployed in the SCA Domain.

1787   A wire may only connect a source to a target if the target implements an interface that is  
1788   compatible with the interface required by the source. The source and the target are compatible if:

- 1789   1. the source interface and the target interface of a wire MUST either both be remotable or  
1790    else both be local [ASM60015]
- 1791   2. the operations on the target interface of a wire MUST be the same as or be a superset of  
1792    the operations in the interface specified on the source [ASM60016]
- 1793   3. compatibility between the source interface and the target interface for a wire for the  
1794    individual operations is defined as compatibility of the signature, that is operation name,  
1795    input types, and output types MUST be the same. [ASM60017]
- 1796   4. the order of the input and output types for operations in the source interface and the  
1797    target interface of a wire also MUST be the same. [ASM60018]
- 1798   5. the set of Faults and Exceptions expected by each operation in the source interface MUST  
1799    be the same or be a superset of those specified by the target interface. [ASM60019]
- 1800   6. other specified attributes of the source interface and the target interface of a wire MUST  
1801    match, including Scope and Callback interface [ASM60020]

1802   A Wire can connect between different interface languages (eg. Java interfaces and WSDL  
1803   portTypes) in either direction, as long as the operations defined by the two interface types are  
1804   equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and  
1805   faults/exceptions map to each other.

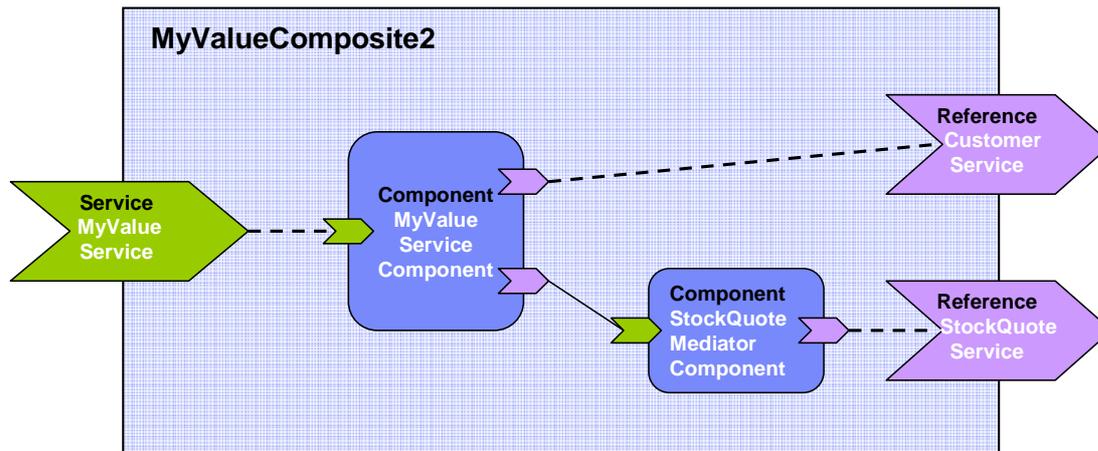
1806   Service clients cannot (portably) ask questions at runtime about additional interfaces that are  
1807   provided by the implementation of the service (e.g. the result of "instance of" in Java is non  
1808   portable). It is valid for an SCA implementation to have proxies for all wires, so that, for example,  
1809   a reference object passed to an implementation may only have the business interface of the  
1810   reference and may not be an instance of the (Java) class which is used to implement the target  
1811   service, even where the interface is local and the target service is running in the same process.

1812 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of  
1813 an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA  
1814 runtime SHOULD issue a warning. [ASM60021]  
1815

## 1816 6.4.1 Wire Examples

1817

1818 The following figure shows the assembly diagram for the MyValueComposite2 containing wires  
1819 between service, components and references.



1820

1821 *Figure 11: MyValueComposite2 showing Wires*

1822

1823 The following snippet shows the MyValueComposite2.composite file for the MyValueComposite2  
1824 containing the configured component and service references. The service MyValueService is wired  
1825 to the MyValueServiceComponent, using an explicit <wire/> element. The  
1826 MyValueServiceComponent's customerService reference is wired to the composite's  
1827 CustomerService reference. The MyValueServiceComponent's stockQuoteService reference is  
1828 wired to the StockQuoteMediatorComponent, which in turn has its reference wired to the  
1829 StockQuoteService reference of the composite.

1830

```
1831 <?xml version="1.0" encoding="ASCII"?>
1832 <!-- MyValueComposite Wires examples -->
1833 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1834           targetNamespace="http://foo.com"
1835           name="MyValueComposite2" >
1836
1837     <service name="MyValueService" promote="MyValueServiceComponent">
1838       <interface.java interface="services.myvalue.MyValueService"/>
1839       <binding.ws port="http://www.myvalue.org/MyValueService#
1840                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1841     </service>
1842
1843     <component name="MyValueServiceComponent" >
```

```

1844         <implementation.java
1845             class="services.myvalue.MyValueServiceImpl"/>
1846         <property name="currency">EURO</property>
1847         <service name="MyValueService"/>
1848         <reference name="customerService"/>
1849         <reference name="stockQuoteService"/>
1850     </component>
1851
1852     <wire source="MyValueServiceComponent/stockQuoteService"
1853         target="StockQuoteMediatorComponent"/>
1854
1855     <component name="StockQuoteMediatorComponent">
1856         <implementation.java class="services.myvalue.SQMediatorImpl"/>
1857         <property name="currency">EURO</property>
1858         <reference name="stockQuoteService"/>
1859     </component>
1860
1861     <reference name="CustomerService"
1862         promote="MyValueServiceComponent/customerService">
1863         <interface.java interface="services.customer.CustomerService"/>
1864         <binding.sca/>
1865     </reference>
1866
1867     <reference name="StockQuoteService"
1868         promote="StockQuoteMediatorComponent">
1869         <interface.java
1870             interface="services.stockquote.StockQuoteService"/>
1871         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1872             wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1873     </reference>
1874
1875 </composite>
1876

```

## 1877 6.4.2 Autowire

1878 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.  
1879 Autowire enables component references to be automatically wired to component services which  
1880 will satisfy those references, without the need to create explicit wires between the references and  
1881 the services. When the autowire feature is used, a component reference which is not promoted  
1882 and which is not explicitly wired to a service within a composite is automatically wired to a target  
1883 service within the same composite. Autowire works by searching within the composite for a  
1884 service interface which matches the interface of the references.

1885 The autowire feature is not used by default. Autowire is enabled by the setting of an autowire  
1886 attribute to "true". Autowire is disabled by setting of the autowire attribute to "false" The autowire  
1887 attribute can be applied to any of the following elements within a composite:

- 1888 • reference
- 1889 • component
- 1890 • composite

1891 Where an element does not have an explicit setting for the autowire attribute, it inherits the  
 1892 setting from its parent element. Thus a reference element inherits the setting from its containing  
 1893 component. A component element inherits the setting from its containing composite. Where  
 1894 there is no setting on any level, autowire="false" is the default.

1895 As an example, if a composite element has autowire="true" set, this means that autowiring is  
 1896 enabled for all component references within that composite. In this example, autowiring can be  
 1897 turned off for specific components and specific references through setting autowire="false" on the  
 1898 components and references concerned.

1899 For each component reference for which autowire is enabled, the the SCA runtime MUST search  
 1900 within the composite for target services which are compatible with the reference. [ASM60022]  
 1901 "Compatible" here means:

- 1902 • the target service interface MUST be a compatible superset of the reference interface  
 1903 when using autowire to wire a reference (as defined in the section on Wires), [ASM60023]
- 1904 • the intents, and policies applied to the service MUST be compatible with those on the  
 1905 reference when using autowire to wire a reference – so that wiring the reference to the  
 1906 service will not cause an error due to policy mismatch [ASM60024] (see [the Policy](#)  
 1907 [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)

1908 If the search finds **1 or more** valid target service for a particular reference, the action taken  
 1909 depends on the multiplicity of the reference:

- 1910 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
 1911 reference to one of the set of valid target services chosen from the set in a runtime-  
 1912 dependent fashion [ASM60025]
- 1913 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
 1914 of the set of valid target services [ASM60026]

1915 If the search finds **no** valid target services for a particular reference, the action taken depends on  
 1916 the multiplicity of the reference:

- 1917 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
 1918 target service, there is no problem – no services are wired and the SCA runtime MUST  
 1919 NOT raise an error [ASM60027]
- 1920 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
 1921 target services an error MUST be raised by the SCA runtime since the reference is  
 1922 intended to be wired [ASM60028]

1923

## 1924 6.4.3 Autowire Examples

1925 This example demonstrates two versions of the same composite – the first version is done using  
 1926 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
 1927 the end result is the same – the same wires connect the references to the services.

1928 First, here is a diagram for the composite:

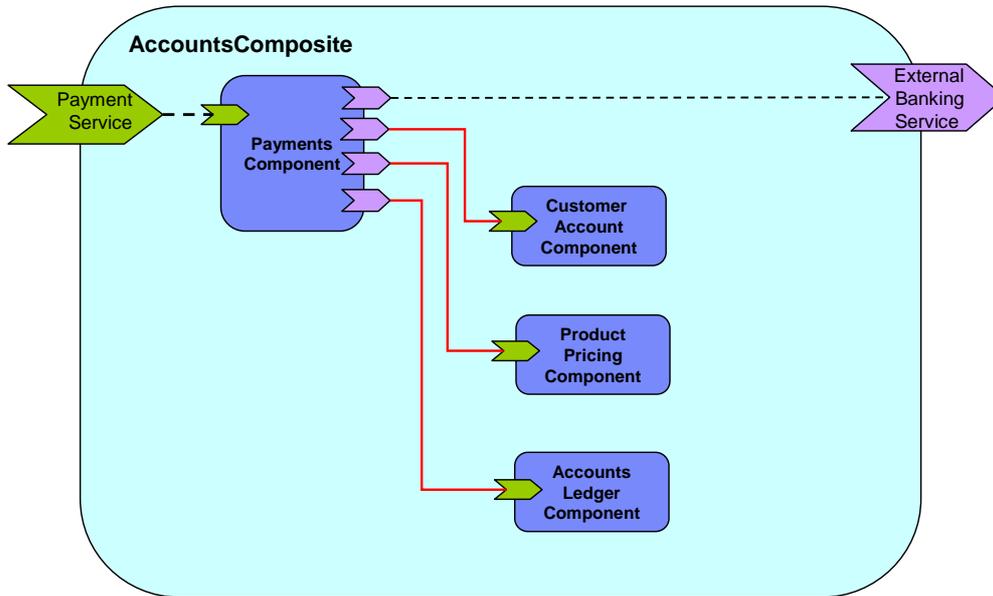


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1929 <?xml version="1.0" encoding="UTF-8"?>
1930 <!-- Autowire Example - No autowire -->
1931 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1932     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1933     xmlns:foo="http://foo.com"
1934     targetNamespace="http://foo.com"
1935     name="AccountComposite">
1936     <service name="PaymentService" promote="PaymentsComponent"/>
1937
1938     <component name="PaymentsComponent">
1939         <implementation.java class="com.foo.accounts.Payments"/>
1940         <service name="PaymentService"/>
1941         <reference name="CustomerAccountService"
1942             target="CustomerAccountComponent"/>
1943         <reference name="ProductPricingService"
1944             target="ProductPricingComponent"/>
1945         <reference name="AccountsLedgerService"
1946             target="AccountsLedgerComponent"/>
1947         <reference name="ExternalBankingService"/>
1948     </component>
1949
1950     <component name="CustomerAccountComponent">

```

```

1955         <implementation.java class="com.foo.accounts.CustomerAccount" />
1956     </component>
1957
1958     <component name="ProductPricingComponent">
1959         <implementation.java class="com.foo.accounts.ProductPricing" />
1960     </component>
1961
1962     <component name="AccountsLedgerComponent">
1963         <implementation.composite name="foo:AccountsLedgerComposite" />
1964     </component>
1965
1966     <reference name="ExternalBankingService"
1967         promote="PaymentsComponent/ExternalBankingService" />
1968
1969 </composite>

```

1971 Secondly, the composite using autowire:

```

1972 <?xml version="1.0" encoding="UTF-8"?>
1973 <!-- Autowire Example - With autowire -->
1974 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1975     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1976     xmlns:foo="http://foo.com"
1977     targetNamespace="http://foo.com"
1978     name="AccountComposite">
1979
1980     <service name="PaymentService" promote="PaymentsComponent">
1981         <interface.java class="com.foo.PaymentServiceInterface" />
1982     </service>
1983
1984     <component name="PaymentsComponent" autowire="true">
1985         <implementation.java class="com.foo.accounts.Payments" />
1986         <service name="PaymentService" />
1987         <reference name="CustomerAccountService" />
1988         <reference name="ProductPricingService" />
1989         <reference name="AccountsLedgerService" />
1990         <reference name="ExternalBankingService" />
1991     </component>
1992
1993     <component name="CustomerAccountComponent">
1994         <implementation.java class="com.foo.accounts.CustomerAccount" />
1995     </component>
1996
1997     <component name="ProductPricingComponent">

```

```

1998         <implementation.java class="com.foo.accounts.ProductPricing" />
1999     </component>
2000
2001     <component name="AccountsLedgerComponent">
2002         <implementation.composite name="foo:AccountsLedgerComposite" />
2003     </component>
2004
2005     <reference name="ExternalBankingService"
2006         promote="PaymentsComponent/ExternalBankingService" />
2007
2008 </composite>

```

2009 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires  
2010 for any of its references – the wires are created automatically through autowire.

2011 **Note:** In the second example, it would be possible to omit all of the service and reference  
2012 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the  
2013 component service and references still exist, since they are provided by the implementation used  
2014 by the component.

2015

## 2016 6.5 Using Composites as Component Implementations

2017 Composites may form **component implementations** in higher-level composites – in other words  
2018 the higher-level composites can have components which are implemented by composites.

2019 When a composite is used as a component implementation, it defines a boundary of visibility.  
2020 Components within the composite cannot be referenced directly by the using component. The  
2021 using component can only connect wires to the services and references of the used composite and  
2022 set values for any properties of the composite. The internal construction of the composite is  
2023 invisible to the using component.

2024 A composite used as a component implementation needs to also honor a **completeness**  
2025 **contract**. The services, references and properties of the composite form a contract which is relied  
2026 upon by the using component. The concept of completeness of the composite implies:

- 2027 • the composite must have at least one service or at least one reference.  
2028 A component with no services and no references is not meaningful in terms of SCA, since  
2029 it cannot be wired to anything – it neither provides nor consumes any **services**.
- 2031 • each service offered by the composite must be wired to a service of a component or to a  
2032 composite reference.  
2033 If services are left unwired, the implication is that some exception will occur at runtime if  
2034 the service is invoked.

2035 The component type of a composite is defined by the set of service elements, reference elements  
2036 and property elements that are the children of the composite element.

2037 Composites are used as component implementations through the use of the  
2038 **implementation.composite** element as a child element of the component. The schema snippet  
2039 for the implementation.composite element is:

```

2040
2041 <?xml version="1.0" encoding="ASCII"?>
2042 <!-- Composite Implementation schema snippet -->
2043 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2044         targetNamespace="xs:anyURI"

```

**Comment [ME10]:** Deliberately left unmarked due to the in-process issue that affects this section....

```

2045         name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2046         constrainingType="QName"?
2047         requires="list of xs:QName"? policySets="list of
2048 xs:QName"?>
2049
2050     ...
2051
2052     <component name="xs:NCName" autowire="xs:boolean"?
2053         requires="list of xs:QName"? policySets="list of xs:QName"?>*
2054     <implementation.composite name="xs:QName"/>?
2055     <service name="xs:NCName" requires="list of xs:QName"?
2056         policySets="list of xs:QName"?>*
2057     <interface ... />?
2058     <binding uri="xs:anyURI" name="xs:QName"?
2059         requires="list of xs:QName"
2060         policySets="list of xs:QName"?/>*
2061     <callback?>
2062         <binding uri="xs:anyURI"? name="xs:QName"?
2063             requires="list of xs:QName"?
2064             policySets="list of xs:QName"?/>+
2065     </callback>
2066     </service>
2067     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
2068         source="xs:string"? file="xs:anyURI"?>*
2069         property-value
2070     </property>
2071     <reference name="xs:NCName" target="list of xs:anyURI"?
2072         autowire="xs:boolean"? wiredByImpl="xs:boolean"?
2073         requires="list of xs:QName"? policySets="list of xs:QName"?
2074         multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
2075     <interface ... />?
2076     <binding uri="xs:anyURI"? name="xs:QName"?
2077         requires="list of xs:QName" policySets="list of
2078 xs:QName"?/>*
2079     <callback?>
2080         <binding uri="xs:anyURI"? name="xs:QName"?
2081             requires="list of xs:QName"?
2082             policySets="list of xs:QName"?/>+
2083     </callback>
2084     </reference>
2085     </component>
2086
2087     ...

```

2088  
2089 `</composite>`  
2090

2091  
2092 The implementation.composite element has the following attribute:

- **name (1..1)** – the name of the composite used as an implementation. The @name attribute of an `<implementation.composite/>` element MUST contain the QName of a composite in the SCA Domain. [ASM60030]

2096

## 2097 **6.5.1 Example of Composite used as a Component Implementation**

2098

2099 The following is an example of a composite which contains two components, each of which is  
2100 implemented by a composite:  
2101

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

```

2132         <reference name="AccountDataService" target="AccountDataService"/>
2133         <reference name="StockQuoteService"/>
2134
2135         <property name="currency" source="$currency"/>
2136     </component>
2137
2138     <component name="AccountDataService">
2139         <implementation.composite name="foo:AccountDataServiceComposite"/>
2140
2141         <property name="currency" source="$currency"/>
2142     </component>
2143
2144 </composite>
2145

```

## 2146 6.6 Using Composites through Inclusion

2147 In order to assist team development, composites may be developed in the form of multiple  
 2148 physical artifacts that are merged into a single logical unit.

2149 A composite is defined in an **xxx.composite** file and the composite may receive additional  
 2150 content through the **inclusion of other composite** files.

2151 The semantics of included composites are that the content of the included composite is inlined into  
 2152 the using composite **xxx.composite** file through **include** elements in the using composite. The  
 2153 effect is one of **textual inclusion** – that is, the text content of the included composite is placed  
 2154 into the using composite in place of the include statement. The included composite element itself  
 2155 is discarded in this process – only its contents are included.

2156 The composite file used for inclusion can have any contents, but always contains a single  
 2157 **composite** element. The composite element can contain any of the elements which are valid as  
 2158 child elements of a composite element, namely components, services, references, wires and  
 2159 includes. There is no need for the content of an included composite to be complete, so that  
 2160 artifacts defined within the using composite or in another associated included composite file may  
 2161 be referenced. For example, it is permissible to have two components in one composite file while a  
 2162 wire specifying one component as the source and the other as the target can be defined in a  
 2163 second included composite file.

2164 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one  
 2165 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated  
 2166 elements in the using composite (eg. two services with the same uri contributed by different  
 2167 included composites), or if there are wires with non-existent source or target.

2168 The following snippet shows the partial schema for the include element.

```

2169
2170 <?xml version="1.0" encoding="UTF-8"?>
2171 <!-- Include snippet -->
2172 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2173           targetNamespace="xs:anyURI"
2174           name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2175           constrainingType="QName"?
2176           requires="list of xs:QName"? policySets="list of
2177 xs:QName"?>

```

```

2178
2179     ...
2180
2181     <include name="xs:QName" />*
2182
2183     ...
2184
2185 </composite>
2186

```

2187 The include element has the following **attribute**:

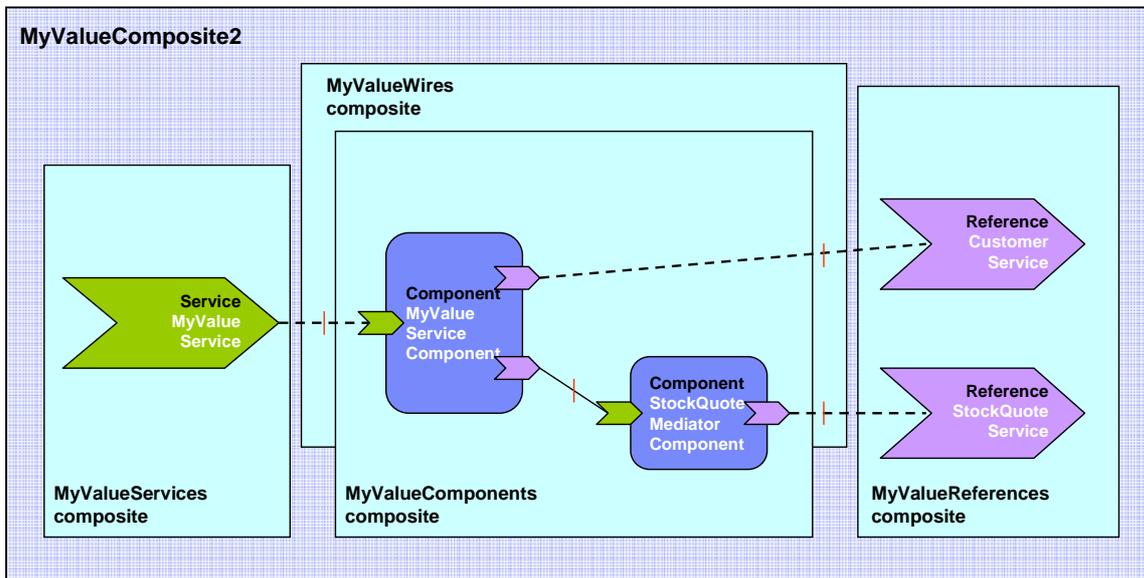
- 2188 • **name (required)** – the name of the composite that is included.

2189

## 2190 6.6.1 Included Composite Examples

2191

2192 The following figure shows the assembly diagram for the MyValueComposite2 containing four  
 2193 included composites. The **MyValueServices composite** contains the MyValueService service. The  
 2194 **MyValueComponents composite** contains the MyValueServiceComponent and the  
 2195 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences**  
 2196 **composite** contains the CustomerService and StockQuoteService references. The **MyValueWires**  
 2197 **composite** contains the wires that connect the MyValueService service to the  
 2198 MyValueServiceComponent, that connect the customerService reference of the  
 2199 MyValueServiceComponent to the CustomerService reference, and that connect the  
 2200 stockQuoteService reference of the StockQuoteMediatorComponent to the StockQuoteService  
 2201 reference. Note that this is just one possible way of building the MyValueComposite2 from a set of  
 2202 included composites.



2203

2204

2205 *Figure 13 MyValueComposite2 built from 4 included composites*

2206

2207 The following snippet shows the contents of the MyValueComposite2.composite file for the  
2208 MyValueComposite2 built using included composites. In this sample it only provides the name of  
2209 the composite. The composite file itself could be used in a scenario using included composites to  
2210 define components, services, references and wires.

```
2211  
2212 <?xml version="1.0" encoding="ASCII"?>  
2213 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2214 targetNamespace="http://foo.com"  
2215 xmlns:foo="http://foo.com"  
2216 name="MyValueComposite2" >  
2217  
2218 <include name="foo:MyValueServices"/>  
2219 <include name="foo:MyValueComponents"/>  
2220 <include name="foo:MyValueReferences"/>  
2221 <include name="foo:MyValueWires"/>  
2222  
2223 </composite>
```

2224 The following snippet shows the content of the MyValueServices.composite file.

```
2226  
2227 <?xml version="1.0" encoding="ASCII"?>  
2228 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2229 targetNamespace="http://foo.com"  
2230 xmlns:foo="http://foo.com"  
2231 name="MyValueServices" >  
2232  
2233 <service name="MyValueService" promote="MyValueServiceComponent">  
2234 <interface.java interface="services.myvalue.MyValueService"/>  
2235 <binding.ws port="http://www.myvalue.org/MyValueService#"  
2236 wsdl.endpoint(MyValueService/MyValueServicesSOAP)"/>  
2237 </service>  
2238  
2239 </composite>
```

2241 The following snippet shows the content of the MyValueComponents.composite file.

```
2242  
2243 <?xml version="1.0" encoding="ASCII"?>  
2244 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2245 targetNamespace="http://foo.com"  
2246 xmlns:foo="http://foo.com"  
2247 name="MyValueComponents" >  
2248  
2249 <component name="MyValueServiceComponent" >
```

```

2250         <implementation.java
2251 class="services.myvalue.MyValueServiceImpl"/>
2252         <property name="currency">EURO</property>
2253     </component>
2254
2255     <component name="StockQuoteMediatorComponent">
2256         <implementation.java class="services.myvalue.SQMediatorImpl"/>
2257         <property name="currency">EURO</property>
2258     </component>
2259
2260 </composite>
2261

```

2262 The following snippet shows the content of the MyValueReferences.composite file.

```

2263
2264 <?xml version="1.0" encoding="ASCII"?>
2265 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2266 targetNamespace="http://foo.com"
2267 xmlns:foo="http://foo.com"
2268 name="MyValueReferences" >
2269
2270     <reference name="CustomerService"
2271 promote="MyValueServiceComponent/CustomerService">
2272         <interface.java interface="services.customer.CustomerService"/>
2273         <binding.sca/>
2274     </reference>
2275
2276     <reference name="StockQuoteService"
2277 promote="StockQuoteMediatorComponent">
2278         <interface.java
2279 interface="services.stockquote.StockQuoteService"/>
2280         <binding.ws port="http://www.stockquote.org/StockQuoteService#
2281 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2282     </reference>
2283
2284 </composite>

```

2285 The following snippet shows the content of the MyValueWires.composite file.

```

2286
2287 <?xml version="1.0" encoding="ASCII"?>
2288 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2289 targetNamespace="http://foo.com"
2290 xmlns:foo="http://foo.com"
2291 name="MyValueWires" >
2292
2293     <wire source="MyValueServiceComponent/stockQuoteService"

```

```
2294         target="StockQuoteMediatorComponent" />
2295
2296     </composite>
```

## 2297 **6.7 Composites which Include Component Implementations of**

### 2298 **Multiple Types**

2299

2300 A Composite containing multiple components can have multiple component implementation types.  
2301 For example, a Composite may include one component with a Java POJO as its implementation  
2302 and another component with a BPEL process as its implementation.

2303

2304

## 7 ConstrainingType

2305  
2306  
2307  
2308  
2309

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.

2310  
2311  
2312  
2313  
2314  
2315

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.

2316  
2317  
2318  
2319  
2320  
2321  
2322

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**.

2323  
2324  
2325

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]

2326  
2327

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

2328

2329

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

2330

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

2331

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

2332

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

2333

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

2334

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

2335

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

2336

```
</service>
```

2337

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

2338

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

2339

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

2340

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

2341

```
</reference>
```

2342

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

2343

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

2344

2345

2346

2347

2348

2349                   default-property-value?

2350                   </property>

2351

2352                   </constrainingType>

2353

2354                   The constrainingType element has the following **attributes**:

- 2355                   • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is  
2356                    an XML QName, in the namespace identified by the targetNamespace attribute. The name  
2357                    attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2358                   • **targetNamespace (0..1)** – an identifier for a target namespace into which the  
2359                    constrainingType is declared
- 2360                   • **requires (0..1)** – a list of policy intents. See [the Policy Framework specification \[10\]](#) for  
2361                    a description of this attribute.

2362                   ConstrainingType contains **zero or more properties, services, references**.

2363

2364                   When an implementation is constrained by a constrainingType its component type MUST contain  
2365                   all the services, references and properties specified in the constrainingType. [ASM70004] The  
2366                   constraining type's references and services will have interfaces specified and can have intents  
2367                   specified. An implementation MAY contain additional services, additional optional references  
2368                   (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the  
2369                   constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or  
2370                   1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

2371                   When a component is constrained by a constrainingType via the "constrainingType" attribute, the  
2372                   entire componentType associated with the component and its implementation is not visible to the  
2373                   containing composite. The containing composite can only see a projection of the componentType  
2374                   associated with the component and implementation as scoped by the constrainingType of the  
2375                   component. Additional services, references and properties provided by the implementation which  
2376                   are not declared in the constrainingType associated with a component MUST NOT be configured in  
2377                   any way by the containing composite. [ASM70006] This requirement ensures that the  
2378                   constrainingType contract cannot be violated by the composite.

2379                   The constrainingType can include required intents on any element. Those intents are applied to  
2380                   any component that uses that constrainingType. In other words, if requires="reliability" exists on  
2381                   a constrainingType, or its child service or reference elements, then a constrained component or its  
2382                   implementation must include requires="reliability" on the component or implementation or on its  
2383                   corresponding service or reference. A component or implementation can use a qualified form of  
2384                   an intent specified in unqualified form in the constrainingType, but if the constrainingType uses  
2385                   the qualified form of an intent, then the component or implementation MUST also use the qualified  
2386                   form, otherwise there is an error. [ASM70007]

2387                   A constrainingType can be applied to an implementation. In this case, the implementation's  
2388                   componentType has a constrainingType attribute set to the QName of the constrainingType.

2389

## 2390                   7.1 Example constrainingType

2391

2392                   The following snippet shows the contents of the component called "MyValueServiceComponent"  
2393                   which is constrained by the constrainingType myns:CT. The componentType associated with the  
2394                   implementation is also shown.

2395

2396                   <component name="MyValueServiceComponent" constrainingType="myns:CT">

2397                    <implementation.java class="services.myvalue.MyValueServiceImpl" />

```

2398     <property name="currency">EURO</property>
2399     <reference name="customerService" target="CustomerService">
2400         <binding.ws ...>
2401     <reference name="StockQuoteService"
2402         target="StockQuoteMediatorComponent"/>
2403 </component>
2404
2405 <constrainingType name="CT"
2406     targetNamespace="http://myns.com">
2407     <service name="MyValueService">
2408         <interface.java interface="services.myvalue.MyValueService"/>
2409     </service>
2410     <reference name="customerService">
2411         <interface.java interface="services.customer.CustomerService"/>
2412     </reference>
2413     <reference name="stockQuoteService">
2414         <interface.java interface="services.stockquote.StockQuoteService"/>
2415     </reference>
2416     <property name="currency" type="xsd:string"/>
2417 </constrainingType>

```

2418 The component MyValueServiceComponent is constrained by the constrainingType CT which  
2419 means that it must provide:

- 2420 • service **MyValueService** with the interface services.myvalue.MyValueService
- 2421 • reference **customerService** with the interface services.stockquote.StockQuoteService
- 2422 • reference **stockQuoteService** with the interface services.stockquote.StockQuoteService
- 2423 • property **currency** of type xsd:string.

## 2424 8 Interface

2425 **Interfaces** define one or more business functions. These business functions are provided by  
2426 Services and are used by References. A Service offers the business functionality of exactly one  
2427 interface for use by other components. Each interface defines one or more service **operations**  
2428 and each operation has zero or one **request (input) message** and zero or one **response**  
2429 **(output) message**. The request and response messages can be simple types such as a string  
2430 value or they can be complex types.

2431 SCA currently supports the following interface type systems:

- 2432 • Java interfaces
- 2433 • WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))
- 2434 • WSDL 2.0 interfaces ([Web Services Definition Language \[8\]](#))
- 2435 • C++ classes

Comment [mbgl11]: Issue 69 part 2

2436 SCA is also extensible in terms of interface types. Support for other interface type systems can be  
2437 added through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2438  
2439 The following snippet shows the definition for the **interface** base element.

Comment [mbgl12]: Issue 39

```
2441 <interface requires="list of xs:QName"? policySets="list of xs:QName"?/>
```

2442

2443 The **interface** base element has the following **attributes**:

- 2444 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
2445 [\[10\]](#) for a description of this attribute
- 2446 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2447 [\[10\]](#) for a description of this attribute.

2448

2449 The following snippet shows a sample for the WSDL portType (WSDL 1.1) or WSDL interface  
2450 (WSDL 2.0) element.

2451

```
2452 <interface.wSDL interface="xs:anyURI" ... />
```

2453

2454 The interface.wSDL element has the following attributes:

- 2455 • **interface** – URI of the portType/interface with the following format.
  - 2456 ○ `<WSDL-namespace-URI>#wSDL.interface(<portTypeOrInterface-name>)`  
2457 The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.0  
2458 document OR an interface element of a WSDL 2.0 document. [\[ASM80001\]](#)

2459

2460 The following snippet shows a sample for the WSDL portType/interface element.

2461

```
2462 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2463 wSDL.interface(StockQuo  
2464 te)"/>
```

2465

2466 For WSDL 1.1, the interface attribute points to a portType in the WSDL. For WSDL 2.0, the  
2467 interface attribute points to an interface in the WSDL. For the WSDL 1.1 portType and WSDL 2.0  
2468 interface type systems, arguments and return of the service operations are described using XML  
2469 schema.

2470 For information about Java interfaces, including details of SCA-specific annotations, see the SCA  
2471 Java Common Annotations and APIs specification [1].

## 2472 8.1 Local and Remotable Interfaces

2473 A remotable service is one which may be called by a client which is running in an operating system  
2474 process different from that of the service itself (this also applies to clients running on different  
2475 machines from the service). Whether a service of a component implementation is remotable is  
2476 defined by the interface of the service. In the case of Java this is defined by adding the  
2477 **@Remotable** annotation to the Java interface (see [Client and Implementation Model Specification](#)  
2478 [for Java](#)). WSDL defined interfaces are always remotable.

2479

2480 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**  
2481 interactions. **Remotable service Interfaces MUST NOT make use of *method or operation***  
2482 ***overloading***, [ASM80002]

2483  
2484 Independent of whether the remotable service is called remotely from outside the process where  
2485 the service runs or from another component running in the same process, the data exchange  
2486 semantics are **by-value**.

2487 Implementations of remotable services can modify input messages (parameters) during or after  
2488 an invocation and can modify return messages (results) after the invocation. If a remotable  
2489 service is called locally or remotely, the SCA container MUST ensure sure that no modification of  
2490 input messages by the service or post-invocation modifications to return messages are seen by  
2491 the caller. [ASM80003]

2492 Here is a snippet which shows an example of a remotable java interface:

2493

```
2494 package services.hello;
```

2495

```
2496 @Remotable
```

```
2497 public interface HelloService {
```

2498

```
2499     String hello(String message);
```

```
2500 }
```

2501

2502 It is possible for the implementation of a remotable service to indicate that it can be called using  
2503 by-reference data exchange semantics when it is called from a component in the same process.  
2504 This can be used to improve performance for service invocations between components that run in  
2505 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java](#)  
2506 [Client and Implementation Specification](#)).

2507

2508 A service typed by a local interface can only be called by clients that are running in the same  
2509 process as the component that implements the local service. Local services cannot be published  
2510 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2511 Java interface definition without a **@Remotable** annotation.

2512

2513 The style of local interfaces is typically **fine grained** and intended for **tightly coupled**  
2514 interactions. Local service interfaces can make use of **method or operation overloading**.

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

2515 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

2516

## 2517 8.2 Bidirectional Interfaces

2518 The relationship of a business service to another business service is often peer-to-peer, requiring  
2519 a two-way dependency at the service level. In other words, a business service represents both a  
2520 consumer of a service provided by a partner business service and a provider of a service to the  
2521 partner business service. This is especially the case when the interactions are based on  
2522 asynchronous messaging rather than on remote procedure calls. The notion of **bidirectional**  
2523 **interfaces** is used in SCA to directly model peer-to-peer bidirectional business service  
2524 relationships.

2525 An interface element for a particular interface type system needs to allow the specification of an  
2526 optional callback interface. If a callback interface is specified, SCA refers to the interface as a  
2527 whole as a bidirectional interface.

2528 The following snippet shows the interface element defined using Java interfaces with an optional  
2529 callbackInterface attribute.

2530

```
2531 <interface.java interface="services.invoicing.ComputePrice"  
2532 callbackInterface="services.invoicing.InvoiceCallback"/>
```

2533

2534 If a service is defined using a bidirectional interface element then its implementation implements  
2535 the interface, and its implementation uses the callback interface to converse with the client that  
2536 called the service interface.

2537

2538 If a reference is defined using a bidirectional interface element, the client component  
2539 implementation using the reference calls the referenced service using the interface. The client  
2540 MUST provide an implementation of the callback interface. [ASM80004]

2541 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2542 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2543 mix local and remote services. [ASM80005]

2544

## 2545 8.3 Conversational Interfaces

2546 Services sometimes cannot easily be defined so that each operation stands alone and is  
2547 completely independent of the other operations of the same service. Instead, there is a sequence  
2548 of operations that must be called in order to achieve some higher level goal. SCA calls this  
2549 sequence of operations a **conversation**. If the service uses a bidirectional interface, the  
2550 conversation may include both operations and callbacks.

2551 Such **conversational services** are typically managed by using conversation identifiers that are  
2552 either (1) part of the application data (message parts or operation parameters) or 2)  
2553 communicated separately from application data (possibly in headers). SCA introduces the concept  
2554 of **conversational interfaces** for describing the interface contract for conversational services of  
2555 the second form above. With this form, it is possible for the runtime to automatically manage the  
2556 conversation, with the help of an appropriate binding specified at deployment. SCA does not  
2557 standardize any aspect of conversational services that are maintained using application data.  
2558 Such services are neither helped nor hindered by SCA's conversational service support.

2559 Conversational services typically involve state data that relates to the conversation that is taking  
2560 place. The creation and management of the state data for a conversation has a significant impact  
2561 on the development of both clients and implementations of conversational services.

2562

2563 Traditionally, application developers who have needed to write conversational services have been  
2564 required to write a lot of plumbing code. They need to:

2565

2566 - choose or define a protocol to communicate conversational (correlation) information  
2567 between the client & provider

2568 - route conversational messages in the provider to a machine that can handle that  
2569 conversation, while handling concurrent data access issues

2570 - write code in the client to use/encode the conversational information

2571 - maintain state that is specific to the conversation, sometimes persistently and  
2572 transactionally, both in the implementation and the client.

2573

2574 SCA makes it possible to divide the effort associated with conversational services between a  
2575 number of roles:

2576 - Application Developer: Declares that a service interface is conversational (leaving the  
2577 details of the protocol up to the binding). Uses lifecycle semantics, APIs or other  
2578 programmatic mechanisms (as defined by the implementation-type being used) to  
2579 manage conversational state.

2580 - Application Assembler: chooses a binding that can support conversations

2581 - Binding Provider: implements a protocol that can pass conversational information with  
2582 each operation request/response.

2583 - Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for  
2584 application developers to access conversational information. Optionally implements  
2585 instance lifecycle semantics that automatically manage implementation state based on  
2586 the binding's conversational information.

2587

2588 There is a policy intent with the name **conversational** which is used to mark an interface as being  
2589 conversational in nature. Where a service or a reference has a conversational interface, the  
2590 conversational intent MUST be attached either to the interface itself, or to the service or reference  
2591 using the interface. [ASM80006] How to attach the conversational intent to an interface depends  
2592 on the type of the interface. For a WSDL interface, this is described in section 8.4 "SCA-Specific  
2593 Aspects for WSDL Interfaces". For a Java interface, it is described in the Java Common  
2594 Annotations and APIs specification. Note that setting the conversational intent on the service or  
2595 reference element is useful when reusing an existing interface definition that contains no SCA  
2596 information, since it requires no modification of the interface artifact.

Comment [mbgl13]: Issue  
35

2597 The meaning of the conversational intent is that both the client and the provider of the interface  
2598 can assume that messages (in either direction) will be handled as part of an ongoing conversation  
2599 without depending on identifying information in the body of the message (i.e. in parameters of the  
2600 operations). In effect, the conversation interface specifies a high-level abstract protocol that must  
2601 be satisfied by any actual binding/policy combination used by the service.

2602 Examples of binding/policy combinations that support conversational interfaces are:

2603 - Web service binding with a WS-RM policy

2604 - Web service binding with a WS-Addressing policy

2605 - Web service binding with a WS-Context policy

2606 - JMS binding with a conversation policy that uses the JMS correlationID header

2607

2608 Conversations occur between one client and one target service. Consequently, requests originating  
2609 from one client to multiple target conversational services will result in multiple conversations. For  
2610 example, if a client A calls services B and C, both of which implement conversational interfaces,

2611 two conversations result, one between A and B and another between A and C. Likewise, requests  
2612 flowing through multiple implementation instances will result in multiple conversations. For  
2613 example, a request flowing from A to B and then from B to C will involve two conversations (A and  
2614 B, B and C). In the previous example, if a request was then made from C to A, a third  
2615 conversation would result (and the implementation instance for A would be different from the one  
2616 making the original request).

2617 Invocation of any operation of a conversational interface can start a conversation. The decision on  
2618 whether an operation starts a conversation depends on the component's implementation and its  
2619 implementation type. Implementation types can support components which provide conversational  
2620 services. If an implementation type does provide this support, the specification for that  
2621 implementation type defines a mechanism for determining when a new conversation should be  
2622 used for an operation (for example, in Java, the conversation is new on the first use of an injected  
2623 reference; in BPEL, the conversation is new when the client's partnerLink comes into scope).

2624

2625 One or more operations in a conversational interface can be annotated with an  
2626 **endsConversation** annotation (the mechanism for annotating the interface depends on the  
2627 interface type) which indicates that when the operation is invoked, the conversation is at an end.  
2628 Where an interface is **bidirectional**, operations may also be annotated in this way on operations  
2629 of the callback interface. When a conversation ending operation is called, it indicates to both the  
2630 client and the service provider that the conversation is complete. Once an operation marked with  
2631 endsConversation has been invoked, any subsequent attempts to call an operation or a callback  
2632 operation associated with the same conversation MUST generate a sca:ConversationViolation fault.  
2633 **[ASM80007]**

2634 A sca:ConversationViolation fault is thrown when one of the following errors occur:

- 2635 - A message is received for a particular conversation, after the conversation has ended
- 2636 - The conversation identification is invalid (not unique, out of range, etc.)
- 2637 - The conversation identification is not present in the input message of the operation that  
2638 ends the conversation
- 2639 - The client or the service attempts to send a message in a conversation, after the  
2640 conversation has ended

2641 This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI  
2642 <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2643 The lifecycle of resources and the association between unique identifiers and conversations are  
2644 determined by the service's implementation type and may not be directly affected by the  
2645 "endConversation" annotation. For example, a **WS-BPEL** process **can** outlive most of the  
2646 conversations that it is involved in.

2647 Although conversational interfaces do not require that any identifying information be passed as  
2648 part of the body of messages, there is conceptually an identity associated with the conversation.  
2649 Individual implementation types can have an API to access the ID associated with the  
2650 conversation, although no assumptions can be made about the structure of that identifier.  
2651 Implementation types can also have a means to set the conversation ID by either the client or the  
2652 service provider, although the operation may only be supported by some binding/policy  
2653 combinations.

2654 Implementation-type specifications are encouraged to define and provide conversational instance  
2655 lifecycle management for components that implement conversational interfaces. However,  
2656 implementations could also manage the conversational state manually.

2657

## 2658 8.4 SCA-Specific Aspects for WSDL Interfaces

2659 There are a number of aspects that SCA applies to interfaces in general, such as marking them  
2660 **conversational**. These aspects apply to the interfaces themselves, rather than their use in a  
2661 specific place within SCA. There is thus a need to provide appropriate ways of marking the

2662 interface definitions themselves, which go beyond the basic facilities provided by the interface  
2663 definition language.

2664 For WSDL interfaces, there is an extension mechanism that permits additional information to be  
2665 included within the WSDL document. SCA takes advantage of this extension mechanism. In order  
2666 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-  
2667 open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2668 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach  
2669 policy intents - **@requires**. The definition of this attribute is as follows:

```
2670 <attribute name="requires" type="sca:listOfQNames" />
```

2671

```
2672 <simpleType name="listOfQNames">
```

```
2673 <list itemType="QName" />
```

```
2674 </simpleType>
```

2675 The @requires attribute can be applied to WSDL Port Type elements (WSDL 1.1) and to WSDL  
2676 Interface elements (WSDL 2.0). The attribute contains one or more intent names, as defined by  
2677 the [Policy Framework specification \[10\]](#). Any service or reference that uses an interface marked  
2678 with required intents MUST implicitly add those intents to its own @requires list. [ASM80008]

2679 To specify that a WSDL interface is conversational, the following attribute setting is used on either  
2680 the WSDL Port Type or WSDL Interface:

```
2681 requires="conversational"
```

2682 SCA defines an **endsConversation** attribute that is used to mark specific operations within a  
2683 WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces  
2684 which are also marked conversational. The endsConversation attribute is a global attribute in the  
2685 SCA namespace, with the following definition:

```
2686 <attribute name="endsConversation" type="boolean" default="false" />
```

2687

2688 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2689 the portType and the **endsConversation** attribute on one of the operations:

2690

```
...
```

```
2691 <portType name="LoanService" sca:requires="conversational">
```

```
2692 <operation name="apply">
```

```
2693 <input message="tns:ApplicationInput" />
```

```
2694 <output message="tns:ApplicationOutput" />
```

```
2695 </operation>
```

```
2696 <operation name="cancel" sca:endsConversation="true">
```

```
2697 </operation>
```

2698

```
...
```

```
2699 </portType>
```

2700

```
...
```

## 2701 9 Binding

2702 Bindings are used by services and references. References use bindings to describe the access  
2703 mechanism used to call a service (which can be a service provided by another SCA composite).  
2704 Services use bindings to describe the access mechanism that clients (which can be a client from  
2705 another SCA composite) have to use to call the service.

2706 SCA supports the use of multiple different types of bindings. Examples include **SCA service, Web**  
2707 **service, stateless session EJB, data base stored procedure, EIS service**. An SCA runtime  
2708 MUST provide support for SCA service and Web service binding types. SCA provides an  
2709 extensibility mechanism by which an SCA runtime can add support for additional binding types.  
2710 For details on how additional binding types are defined, see the section on the Extension Model.

**Comment [ME14]:** Didn't we get rid of this requirement?

2711 A binding is defined by a **binding element** which is a child element of a service or of a reference  
2712 element in a composite. The following snippet shows the composite schema with the schema for  
2713 the binding element.

```
2714
2715 <?xml version="1.0" encoding="ASCII"?>
2716 <!-- Bindings schema snippet -->
2717 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2718           targetNamespace="xs:anyURI"
2719           name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2720           constrainingType="QName"?
2721           requires="list of xs:QName"? policySets="list of
2722 xs:QName"?>
2723
2724     ...
2725
2726     <service name="xs:NCName" promote="xs:anyURI"
2727           requires="list of xs:QName"? policySets="list of xs:QName"?>*
2728       <interface ... />?
2729       <binding uri="xs:anyURI"? name="xs:NCName"?
2730           requires="list of xs:QName"? policySets="list of
2731 xs:QName"?/>*
2732       <callback?
2733           <binding uri="xs:anyURI"? name="xs:NCName"?
2734           requires="list of xs:QName"?
2735           policySets="list of xs:QName"?/>+
2736       </callback>
2737     </service>
2738
2739     ...
2740
2741     <reference name="xs:NCName" target="list of xs:anyURI"?
2742           promote="list of xs:anyURI"? wiredByImpl="xs:boolean"?
2743           multiplicity="0..1 or 1..1 or 0..n or 1..n"?

```

```

2744         requires="list of xs:QName"? policySets="list of xs:QName"?>*
2745     <interface ... />?
2746     <binding uri="xs:anyURI"? name="xs:NCName"?
2747         requires="list of xs:QName"? policySets="list of
2748 xs:QName"?/>*
2749     <callback?
2750         <binding uri="xs:anyURI"? name="xs:NCName"?
2751             requires="list of xs:QName"?
2752             policySets="list of xs:QName"?/>+
2753     </callback>
2754 </reference>
2755
2756     ...
2757
2758 </composite>
2759

```

2760 The element name of the binding element is architected; it is in itself a qualified name. The first  
2761 qualifier is always named "binding", and the second qualifier names the respective binding-type  
2762 (e.g. binding.composite, binding.ws, binding.ejb, binding.eis).

2763

2764 A binding element has the following attributes:

2765

- **uri (0..1)** - has the following semantic.
  - The uri attribute can be omitted.
  - 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) [ASM90001]
  - The circumstances under which the uri attribute can be used are defined in section "5.3.1 Specifying the Target Service(s) for a Reference."
  - For a binding of a **service** the URI attribute defines the URI relative to the component, which contributes the service to the SCA domain. The default value for the URI is the value of the name attribute of the binding.
- **name (0..1)** - a name for the binding instance (an NCName). The name attribute allows distinction between multiple binding elements on a single service or reference. The default value of the name attribute is the service or reference name. 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. [ASM90002] The name also permits the binding instance to be referenced from elsewhere - particularly useful for some types of binding, which can be declared in a definitions document as a template and referenced from other binding instances, simplifying the definition of more complex binding instances (see the JMS Binding specification [11] for examples of this referencing).
- **requires (optional)** - a list of policy intents. See the Policy Framework specification [10] for a description of this attribute.
- **policySets (optional)** - a list of policy sets. See the Policy Framework specification [10] for a description of this attribute.

2766

2767

2768

2769

2770

2771

2772

2773

2774

2775

2776

2777

2778

2779

2780

2781

2782

2783

2784

2785

2786

2787

2788

2789

2790

**Comment [ME15]:** This contradicts material below - is this the Issue 57 problem?

**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).

2791 When multiple bindings exist for an service, it means that the service is available by any of the  
2792 specified bindings. The technique that the SCA runtime uses to choose among available bindings  
2793 is left to the implementation and it may include additional (nonstandard) configuration. Whatever  
2794 technique is used needs to be documented by the runtime.

2795 Services and References can always have their bindings overridden at the SCA domain level,  
2796 unless restricted by Intents applied to them.

2797 If a reference has any bindings they MUST be resolved which means that each binding MUST  
2798 include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference  
2799 MUST NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds  
2800 of bindings that are acceptable for use with a reference, the user specifies either policy intents or  
2801 policy sets.

2802 Users can also specifically wire, not just to a component service, but to a specific binding offered  
2803 by that target service. To do so, a wire target MAY be specified with a syntax of  
2804 "componentName/serviceName/bindingName". [ASM90004]  
2805

2806

2807 The following sections describe the SCA and Web service binding type in detail.

2808

## 2809 9.1 Messages containing Data not defined in the Service Interface

2810 It is possible for a message to include information that is not defined in the interface used to  
2811 define the service, for instance information may be contained in SOAP headers or as MIME  
2812 attachments.

2813 Implementation types can make this information available to component implementations in their  
2814 execution context. The specifications for these implementation types describe how this  
2815 information is accessed and in what form it is presented.

2816

## 2817 9.2 Form of the URI of a Deployed Binding

2818

### 2819 9.2.1 Constructing Hierarchical URIs

2820 Bindings that use hierarchical URI schemes construct the effective URI with a combination of the  
2821 following pieces:

2822 Base System URI for a scheme / Component URI / Service Binding URI

2823

2824 Each of these components deserves addition definition:

2825 **Base Domain URI for a scheme.** An SCA domain should define a base URI for each hierarchical  
2826 URI scheme on which it intends to provide services.

2827 For example: the HTTP and HTTPS schemes would each have their own base URI defined for the  
2828 domain. An example of a scheme that is not hierarchical, and therefore will have no base URI is  
2829 the "jms:" scheme.

2830 **Component URI.** The component URI above is for a component that is deployed in the SCA  
2831 Domain. The URI of a component defaults to the name of the component, which is used as a  
2832 relative URI. The component may have a specified URI value. The specified URI value may be an  
2833 absolute URI in which case it becomes the Base URI for all the services belonging to the  
2834 component. If the specified URI value is a relative URI, it is used as the Component URI value  
2835 above.

Comment [ME16]: Issue 16  
resolution affects this  
section a lot - left for the  
moment

2836 **Service Binding URI.** The Service Binding URI is the relative URI specified in the "uri" attribute  
2837 of a binding element of the service. The default value of the attribute is value of the binding's  
2838 name attribute treated as a relative URI. If multiple bindings for a single service use the same  
2839 scheme (e.g. HTTP), then only one of the bindings may depend on the default value for the uri  
2840 attribute, i.e. only one may use the default binding name. The service binding URI may also be  
2841 absolute, in which case the absolute URI fully specifies the full URI of the service. Some  
2842 deployment environments may not support the use of absolute URIs in service bindings.

2843 Services deployed into the Domain (as opposed to services of components) have a URI that does  
2844 not include a component name, i.e.:

2845 Base Domain URI for a scheme / Service Binding URI

2846 The name of the containing composite does not contribute to the URI of any service.

2847 For example, a service where the Base URI is "http://acme.com", the component is named  
2848 "stocksComponent" and the service binding name is "getQuote", the URI would look like this:

2849 http://acme.com/stocksComponent/getQuote

2850 Allowing a binding's relative URI to be specified that differs from the name of the service allows  
2851 the URI hierarchy of services to be designed independently of the organization of the domain.

2852 It is good practice to design the URI hierarchy to be independent of the domain organization, but  
2853 there may be times when domains are initially created using the default URI hierarchy. When this  
2854 is the case, the organization of the domain can be changed, while maintaining the form of the URI  
2855 hierarchy, by giving appropriate values to the *uri* attribute of select elements. Here is an example  
2856 of a change that can be made to the organization while maintaining the existing URIs:

2857 To move a subset of the services out of one component (say "foo") to a new component (say  
2858 "bar"), the new component should have bindings for the moved services specify a URI  
2859 "../foo/MovedService"..

2860 The URI attribute may also be used in order to create shorter URIs for some endpoints, where the  
2861 component name may not be present in the URI at all. For example, if a binding has a *uri*  
2862 attribute of "../myService" the component name will not be present in the URI.

## 2863 9.2.2 Non-hierarchical URIs

2864 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) may optionally make  
2865 use of the "uri" attribute, which is the complete representation of the URI for that service  
2866 binding. Where the binding does not use the "uri" attribute, the binding must offer a different  
2867 mechanism for specifying the service address.

## 2868 9.2.3 Determining the URI scheme of a deployed binding

2869 One of the things that needs to be determined when building the effective URI of a deployed  
2870 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is  
2871 binding type specific.

2872 If the binding type supports a single protocol then there is only one URI scheme associated with it.  
2873 In this case, that URI scheme is used.

2874 If the binding type supports multiple protocols, the binding type implementation determines the  
2875 URI scheme by introspecting the binding configuration, which may include the policy sets  
2876 associated with the binding.

2877 A good example of a binding type that supports multiple protocols is binding.ws, which can be  
2878 configured by referencing either an "abstract" WSDL element (i.e. portType or interface) or a  
2879 "concrete" WSDL element (i.e. binding, port or endpoint). When the binding references a PortType  
2880 or Interface, the protocol and therefore the URI scheme is derived from the intents/policy sets  
2881 attached to the binding. When the binding references a "concrete" WSDL element, there are two  
2882 cases:

2883 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most  
2884 common case. In this case, the URI scheme is given by the protocol/transport specified in the  
2885 WSDL binding element.

2886 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,  
2887 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"  
2888 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined  
2889 by looking at the policy sets attached to the binding.

2890 It's worth noting that an intent supported by a binding type may completely change the behavior  
2891 of the binding. For example, when the intent "confidentiality/transport" is required by an HTTP  
2892 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to  
2893 "https".

2894

## 2895 9.3 SCA Binding

2896 The SCA binding element is defined by the following schema.

2897

2898 `<binding.sca />`

2899

2900 The SCA binding can be used for service interactions between references and services contained  
2901 within the SCA domain. The way in which this binding type is implemented is not defined by the  
2902 SCA specification and it can be implemented in different ways by different SCA runtimes. The only  
2903 requirement is that the required qualities of service must be implemented for the SCA binding  
2904 type. The SCA binding type is **not** intended to be an interoperable binding type. For  
2905 interoperability, an interoperable binding type such as the Web service binding should be used.

2906 A service definition with no binding element specified uses the SCA binding.

2907 `<binding.sca/>` would only have to be specified in override cases, or when you specify a  
2908 set of bindings on a service definition and the SCA binding should be one of them.

2909 If a reference does not have a binding, then the binding used can be any of the bindings  
2910 specified by the service provider, as long as the intents required by the reference and  
2911 the service are all respected.

2912 If the interface of the service or reference is local, then the local variant of the SCA  
2913 binding will be used. If the interface of the service or reference is remotable, then either  
2914 the local or remote variant of the SCA binding will be used depending on whether source  
2915 and target are co-located or not.

2916 If a reference specifies an URI via its uri attribute, then this provides the default wire to a service  
2917 provided by another domain level component. The value of the URI has to be as follows:

- 2918 • `<domain-component-name>/<service-name>`

2919

### 2920 9.3.1 Example SCA Binding

2921 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
2922 containing the service element for the MyValueService and a reference element for the  
2923 StockQuoteService. Both the service and the reference use an SCA binding. The target for the  
2924 reference is left undefined in this binding and would have to be supplied by the composite in which  
2925 this composite is used.

2926

2927 `<?xml version="1.0" encoding="ASCII"?>`

2928 `<!-- Binding SCA example -->`

```
2929     <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2930                   targetNamespace="http://foo.com"
2931                   name="MyValueComposite" >
2932
2933         <service name="MyValueService" promote="MyValueComponent">
2934             <interface.java interface="services.myvalue.MyValueService"/>
2935             <binding.sca/>
2936             ...
2937         </service>
2938
2939         ...
2940
2941         <reference name="StockQuoteService"
2942 promote="MyValueComponent/StockQuoteReference">
2943             <interface.java
2944 interface="services.stockquote.StockQuoteService"/>
2945             <binding.sca/>
2946         </reference>
2947
2948     </composite>
2949
```

## 2950 9.4 Web Service Binding

2951 SCA defines a Web services binding. This is described in a [separate specification document \[9\]](#).  
2952

## 2953 9.5 JMS Binding

2954 SCA defines a JMS binding. This is described in a [separate specification document \[11\]](#).

2955

## 10 SCA Definitions

2956  
2957  
2958

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

2959  
2960  
2961

All of these artifacts within an SCA Domain are defined in a global, SCA Domain-wide file named definitions.xml. The definitions.xml file contains a definitions element that conforms to the following pseudo-schema snippet:

2962  
2963  
2964  
2965  
2966  
2967  
2968  
2969  
2970  
2971  
2972  
2973  
2974  
2975  
2976  
2977

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Composite schema snippet -->
<definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
targetNamespace="xs:anyURI">

  <sca:intent/>*

  <sca:policySet/>*

  <sca:binding/>*

  <sca:bindingType/>*

  <sca:implementationType/>*

</definitions>
```

2978

The definitions element has the following attribute:

2979  
2980

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

2981  
2982  
2983  
2984  
2985  
2986

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

2987

## 11 Extension Model

2988

2989 The assembly model can be extended with support for new interface types, implementation types  
2990 and binding types. The extension model is based on XML schema substitution groups. There are  
2991 three XML Schema substitution group heads defined in the SCA namespace: **interface**,  
2992 **implementation** and **binding**, for interface types, implementation types and binding types,  
2993 respectively.

2994 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1],  
2995 [9], [11]) use these XML Schema substitution groups to define some basic types of interfaces,  
2996 implementations and bindings, but other types can be defined as required, where support for  
2997 these extra ones is available from the runtime. The interface type elements, implementation type  
2998 elements, and binding type elements defined by the SCA specifications are all part of the SCA  
2999 namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200712"), as indicated in their  
3000 respective schemas. New interface types, implementation types and binding types that are defined  
3001 using this extensibility model, which are not part of these SCA specifications are defined in  
3002 namespaces other than the SCA namespace.

3003 The "." notation is used in naming elements defined by the SCA specifications ( e.g.  
3004 <implementation.java ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel  
3005 extensibility approach but as a naming convention that improves usability of the SCA assembly  
3006 language.

3007

3008 **Note:** How to contribute SCA model extensions and their runtime function to an SCA runtime will  
3009 be defined by a future version of the specification.

3010

### 11.1 Defining an Interface Type

3011 The following snippet shows the base definition for the **interface** element and **Interface** type  
3012 contained in **sca-core.xsd**; see appendix for complete schema.

3013

```
3014 <?xml version="1.0" encoding="UTF-8"?>  
3015 <!-- (c) Copyright SCA Collaboration 2006 -->  
3016 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3017         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3018         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3019         elementFormDefault="qualified">
```

3020

3021 ...

3022

```
3023 <element name="interface" type="sca:Interface" abstract="true"/>
```

```
3024 <complexType name="Interface"/>
```

```
3025 <complexType name="Interface" abstract="true">
```

```
3026 <attribute name="requires" type="sca:listOfQNames" use="optional"/>
```

```
3027 <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
```

```
3028 </complexType>
```

3029

3030

Comment [mbgl17]: Issue  
39

3031  
3032     ...  
3033  
3034 </schema>

3035 In the following snippet is an example of how the base definition is extended to support Java  
3036 interfaces. The snippet shows the definition of the **interface.java** element and the  
3037 **JavaInterface** type contained in **sca-interface-java.xsd**.

```
3038  
3039 <?xml version="1.0" encoding="UTF-8"?>  
3040 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3041       targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3042       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3043  
3044     <element name="interface.java" type="sca:JavaInterface"  
3045       substitutionGroup="sca:interface"/>  
3046     <complexType name="JavaInterface">  
3047       <complexContent>  
3048         <extension base="sca:Interface">  
3049           <attribute name="interface" type="NCName"  
3050             use="required"/>  
3051         </extension>  
3052       </complexContent>  
3053     </complexType>  
3054 </schema>
```

3055 In the following snippet is an example of how the base definition can be extended by other  
3056 specifications to support a new interface not defined in the SCA specifications. The snippet shows  
3057 the definition of the **my-interface-extension** element and the **my-interface-extension-type**  
3058 type.

```
3059 <?xml version="1.0" encoding="UTF-8"?>  
3060 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3061       targetNamespace="http://www.example.org/myextension"  
3062       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3063       xmlns:tns="http://www.example.org/myextension">  
3064  
3065     <element name="my-interface-extension"  
3066       type="tns:my-interface-extension-type"  
3067       substitutionGroup="sca:interface"/>  
3068     <complexType name="my-interface-extension-type">  
3069       <complexContent>  
3070         <extension base="sca:Interface">  
3071           ...  
3072         </extension>  
3073       </complexContent>  
3074     </complexType>
```

3075 </schema>  
3076

## 3077 11.2 Defining an Implementation Type

3078 The following snippet shows the base definition for the *implementation* element and  
3079 **Implementation** type contained in *sca-core.xsd*; see appendix for complete schema.

```
3080  
3081 <?xml version="1.0" encoding="UTF-8"?>  
3082 <!-- (c) Copyright SCA Collaboration 2006 -->  
3083 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3084     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3085     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3086     elementFormDefault="qualified">  
3087  
3088     ...  
3089  
3090     <element name="implementation" type="sca:Implementation"  
3091 abstract="true"/>  
3092     <complexType name="Implementation"/>  
3093  
3094     ...  
3095 </schema>  
3096
```

3097  
3098 In the following snippet we show how the base definition is extended to support Java  
3099 implementation. The snippet shows the definition of the *implementation.java* element and the  
3100 **JavaImplementation** type contained in *sca-implementation-java.xsd*.

```
3101  
3102 <?xml version="1.0" encoding="UTF-8"?>  
3103 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3104     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3105     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3106  
3107     <element name="implementation.java" type="sca:JavaImplementation"  
3108             substitutionGroup="sca:implementation"/>  
3109     <complexType name="JavaImplementation">  
3110         <complexContent>  
3111             <extension base="sca:Implementation">  
3112                 <attribute name="class" type="NCName"  
3113                     use="required"/>  
3114             </extension>  
3115         </complexContent>  
3116     </complexType>  
3117 </schema>
```

3118 In the following snippet is an example of how the base definition can be extended by other  
3119 specifications to support a new implementation type not defined in the SCA specifications. The  
3120 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**  
3121 **type** type.

```
3122 <?xml version="1.0" encoding="UTF-8"?>
3123 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3124         targetNamespace="http://www.example.org/myextension"
3125         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3126         xmlns:tns="http://www.example.org/myextension">
3127
3128     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3129           substitutionGroup="sca:implementation"/>
3130     <complexType name="my-impl-extension-type">
3131         <complexContent>
3132             <extension base="sca:Implementation">
3133                 ...
3134             </extension>
3135         </complexContent>
3136     </complexType>
3137 </schema>
3138
```

3139 In addition to the definition for the new implementation instance element, there needs to be an  
3140 associated implementationType element which provides metadata about the new implementation  
3141 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3142 <implementationType type="xs:QName"
3143                   alwaysProvides="list of intent xs:QName"
3144                   mayProvide="list of intent xs:QName"/>
3145
```

3146 The implementation type has the following attributes:

- 3147 • **type (1..1)** – the type of the implementation to which this implementationType element  
3148 applies. This is intended to be the QName of the implementation element for the  
3149 implementation type, such as "sca:implementation.java"
- 3150 • **alwaysProvides (0..1)** – a set of intents which the implementation type always  
3151 provides. See [the Policy Framework specification \[10\]](#) for details.
- 3152 • **mayProvide (0..1)** – a set of intents which the implementation type may provide. See  
3153 [the Policy Framework specification \[10\]](#) for details.

3154

## 3155 11.3 Defining a Binding Type

3156 The following snippet shows the base definition for the **binding** element and **Binding** type  
3157 contained in **sca-core.xsd**; see appendix for complete schema.

```
3158
3159 <?xml version="1.0" encoding="UTF-8"?>
3160 <!-- binding type schema snippet -->
3161 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

```

3162 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3163         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3164         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3165         elementFormDefault="qualified">
3166
3167     ...
3168
3169     <element name="binding" type="sca:Binding" abstract="true"/>
3170     <complexType name="Binding">
3171         <attribute name="uri" type="anyURI" use="optional"/>
3172         <attribute name="name" type="NCName" use="optional"/>
3173         <attribute name="requires" type="sca:listOfQNames"
3174             use="optional"/>
3175         <attribute name="policySets" type="sca:listOfQNames"
3176             use="optional"/>
3177     </complexType>
3178
3179     ...
3180
3181 </schema>

```

3182 In the following snippet is an example of how the base definition is extended to support Web  
3183 service binding. The snippet shows the definition of the **binding.ws** element and the  
3184 **WebServiceBinding** type contained in **sca-binding-webservice.xsd**.

```

3185
3186 <?xml version="1.0" encoding="UTF-8"?>
3187 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3188         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3189         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3190
3191     <element name="binding.ws" type="sca:WebServiceBinding"
3192         substitutionGroup="sca:binding"/>
3193     <complexType name="WebServiceBinding">
3194         <complexContent>
3195             <extension base="sca:Binding">
3196                 <attribute name="port" type="anyURI" use="required"/>
3197             </extension>
3198         </complexContent>
3199     </complexType>
3200 </schema>

```

3201 In the following snippet is an example of how the base definition can be extended by other  
3202 specifications to support a new binding not defined in the SCA specifications. The snippet shows  
3203 the definition of the **my-binding-extension** element and the **my-binding-extension-type** type.

```

3204 <?xml version="1.0" encoding="UTF-8"?>
3205 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

3206         targetNamespace="http://www.example.org/myextension"
3207         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3208         xmlns:tns="http://www.example.org/myextension">
3209
3210     <element name="my-binding-extension"
3211         type="tns:my-binding-extension-type"
3212         substitutionGroup="sca:binding"/>
3213     <complexType name="my-binding-extension-type">
3214         <complexContent>
3215             <extension base="sca:Binding">
3216                 ...
3217             </extension>
3218         </complexContent>
3219     </complexType>
3220 </schema>
3221

```

3222 In addition to the definition for the new binding instance element, there needs to be an associated  
3223 bindingType element which provides metadata about the new binding type. The pseudo schema  
3224 for the bindingType element is shown in the following snippet:

```

3225 <bindingType type="xs:QName"
3226     alwaysProvides="list of intent QNames"?
3227     mayProvide = "list of intent QNames"?/>
3228

```

3229 The binding type has the following attributes:

- 3230 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3231 intended to be the QName of the binding element for the binding type, such as  
3232 "sca:binding.ws"
- 3233 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3234 [the Policy Framework specification \[10\]](#) for details.
- 3235 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)  
3236 [Policy Framework specification \[10\]](#) for details.

---

## 3237 12 Packaging and Deployment

### 3238 12.1 Domains

3239 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3240 of interconnected runtime nodes.

3241 A single SCA domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3242 wires can only be used to connect components within a single SCA domain. Connections to  
3243 services outside the domain must use binding specific mechanisms for addressing services (such  
3244 as WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used  
3245 in the context of a single domain. In general, external clients of a service that is developed and  
3246 deployed using SCA should not be able to tell that SCA was used to implement the service – it is  
3247 an implementation detail.

3248 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3249 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3250 functionality controlled by a single organization. For example, an SCA Domain may be the whole  
3251 of a business, or it may be a department within a business.

3252 As an example, for the accounts department in a business, the SCA Domain might cover all  
3253 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3254 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3255 An SCA domain has the following:

- 3256 • A virtual domain-level composite whose components are deployed and running
- 3257 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3258 necessary to execute components
- 3259 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3260 level composite.

3261 The information associated with an SCA domain can be stored in many ways, including but not  
3262 limited to a specific filesystem structure or a repository.

### 3263 12.2 Contributions

3264 An SCA domain might require a large number of different artifacts in order to work. These  
3265 artifacts include artifacts defined by SCA and other artifacts such as object code files and interface  
3266 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3267 different SCA definition documents are: `composite`, `componentType`, `constrainingType` and  
3268 definitions. XML artifacts that are not defined by SCA but which may be needed by an SCA  
3269 domain include XML Schema documents, WSDL documents, and BPEL documents. SCA  
3270 constructs, like other XML-defined constructs, use XML qualified names for their identity (i.e.  
3271 namespace + local name).

3272 Non-XML artifacts are also required within an SCA domain. The most obvious examples of such  
3273 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3274 implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3275 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3276 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3277 packaging formats, but requires that the ZIP format be supported. When using the ZIP format for  
3278 deploying a contribution, this specification does not specify whether that format is retained after  
3279 deployment. For example, a Java EE based SCA runtime may convert the ZIP package to an EAR  
3280 package. SCA expects certain characteristics of any packaging:

- 3281 • For any contribution packaging it MUST be possible to present the artifacts of the  
3282 packaging to SCA as a hierarchy of resources based off of a single root [\[ASM12001\]](#)

- 3283
- 3284
- Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy named META-INF [ASM12002]
- 3285
- 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. [ASM12003]
- 3287
- 3288
- 3289
- 3290
- 3291
- 3292
- 3293
- 3294
- 3295
- 3296
- 3297
- 3298
- 3299
- The same document also optionally lists namespaces of constructs that are defined within the contribution and which may be used by other contributions
- 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. [ASM12004] These optional elements may not be physically present in the packaging, but may be generated based on the definitions and references that are present, or they may not exist at all if there are no unresolved references.
- See the section "SCA Contribution Metadata Document" for details of the format of this file.

3300

3301

3302

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

- 3303
- A filesystem directory
- 3304
- An OSGi bundle
- 3305
- A compressed directory (zip, gzip, etc)
- 3306
- A JAR file (or its variants – WAR, EAR, etc)

3307

3308

3309

3310

3311

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

3312

3313

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

3314

## 3315 12.2.1 SCA Artifact Resolution

3316

3317

3318

3319

3320

Contributions may be self-contained, in that all of the artifacts necessary to run the contents of the contribution are found within the contribution itself. However, it can also be the case that the contents of the contribution make one or many references to artifacts that are not contained within the contribution. These references can be to SCA artifacts or they can be to other artifacts such as WSDL files, XSD files or to code artifacts such as Java class files and BPEL scripts.

3321

3322

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

- 3323
- wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema artifacts respectively
- 3324
- OSGi bundle mechanisms for resolving Java class and related resource dependencies
- 3325

3326

3327

Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact dependencies. [ASM12005]

3328

3329

3330

3331

3332

SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanisms are used either where no other mechanisms are available, or in cases where the mechanisms used by the various contributions in the same SCA Domain are different. An example of the latter case is where an OSGi Bundle is used for one contribution but where a second contribution used by the first one is not implemented using OSGi - eg the second contribution is a mainframe COBOL

3333 service whose interfaces are declared using WSDL which must be accessed by the first  
3334 contribution.

3335 The SCA artifact resolution is likely to be most useful for SCA domains containing heterogeneous  
3336 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to  
3337 work across different kinds of contribution.

3338 SCA artifact resolution works on the principle that a contribution which needs to use artifacts  
3339 defined elsewhere expresses these dependencies using **import** statements in metadata belonging  
3340 to the contribution. A contribution controls which artifacts it makes available to other  
3341 contributions through **export** statements in metadata attached to the contribution.

3342

## 3343 12.2.2 SCA Contribution Metadata Document

3344 The contribution optionally contains a document that declares runnable composites, exported  
3345 definitions and imported definitions. The document is found at the path of META-INF/sca-  
3346 contribution.xml relative to the root of the contribution. Frequently some SCA metadata needs to  
3347 be specified by hand while other metadata is generated by tools (such as the <import> elements  
3348 described below). To accommodate this, it is also possible to have an identically structured  
3349 document at META-INF/sca-contribution-generated.xml. If this document exists (or is generated  
3350 on an as-needed basis), it will be merged into the contents of sca-contribution.xml, with the  
3351 entries in sca-contribution.xml taking priority if there are any conflicting declarations.

3352 The format of the document is:

```
3353 <?xml version="1.0" encoding="ASCII"?>
3354 <!-- sca-contribution pseudo-schema -->
3355 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200712>
3356
3357     <deployable composite="xs:QName" />*
3358     <import namespace="xs:String" location="xs:AnyURI"? />*
3359     <export namespace="xs:String" />*
3360
3361 </contribution>
```

3364 **deployable element:** Identifies a composite which is a composite within the contribution that is a  
3365 composite intended for potential inclusion into the virtual domain-level composite. Other  
3366 composites in the contribution are not intended for inclusion but only for use by other composites.  
3367 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)  
3368 [Composite](#) capability and the add To Domain Level Composite capability.

3369 Attributes of the deployable element:

- 3370 • **composite (1..1)** – The QName of a composite within the contribution.

3371

3372 **Export element:** A declaration that artifacts belonging to a particular namespace are exported  
3373 and are available for use within other contributions. An export declaration in a contribution  
3374 specifies a namespace, all of whose definitions are considered to be exported. By default,  
3375 definitions are not exported.

3376 The SCA artifact export is useful for SCA domains containing heterogeneous mixtures of  
3377 contribution packagings and technologies, where artifact-related or packaging-related mechanisms  
3378 are unlikely to work across different kinds of contribution.

3379 Attributes of the export element:

- 3380  
3381  
3382  
3383  
3384  
3385  
3386  
3387  
3388  
3389  
3390
- **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace should be the namespace URI for the exported definitions. For XML technologies that define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are a different symbol space from WSDL bindings), all definitions from all symbol spaces are exported.
- Technologies that use naming schemes other than QNames must use a different export element from the same substitution group as the the SCA <export> element. The element used identifies the technology, and can use any value for the namespace that is appropriate for that technology. For example, <export.java> can be used can be used to export java definitions, in which case the namespace is a fully qualified package name.

3391  
3392  
3393  
3394  
3395  
3396

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

3397  
Attributes of the import element:

- 3398  
3399  
3400  
3401  
3402  
3403
- **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace is the namespace URI for the imported definitions. For XML technologies that define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are a different symbol space from WSDL bindings), all definitions from all symbol spaces are imported.

3404  
3405  
3406  
3407  
3408

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

- 3409  
3410  
3411  
3412  
3413
- **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific requirements for the form of this URI, nor the means by which it is resolved. It can point to another contribution (through its URI) or it can point to some location entirely outside the SCA Domain.

3414  
3415  
3416

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

3417  
3418  
3419

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

3420  
3421  
3422

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

3423  
3424  
3425  
3426

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

3427  
3428  
3429  
3430

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

3431  
3432  
3433

If the value of the location attribute is an SCA contribution URI, then the contribution packaging can become dependent on the deployment environment. In order to avoid such a dependency,

3434 dependent contributions should be specified only when deploying or updating contributions as  
3435 specified in the section 'Operations for Contributions' below.

### 3436 **12.2.3 Contribution Packaging using ZIP**

3437 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires  
3438 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This  
3439 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3440 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3441 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-  
3442 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such  
3443 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3444 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)  
3445 [.ZIP file format \[12\]](#).

3446

## 3447 **12.3 Installed Contribution**

3448 As noted in the section above, the contents of a contribution do not need to be modified in order  
3449 to install and use it within a domain. An *installed contribution* is a contribution with all of the  
3450 associated information necessary in order to execute *deployable composites* within the  
3451 contribution.

3452 An installed contribution is made up of the following things:

- 3453 • Contribution Packaging – the contribution that will be used as the starting point for  
3454 resolving all references
- 3455 • Contribution base URI
- 3456 • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3457 the import statements from the root composite and from other dependent contributions
  - 3458 ○ Dependent contributions might or might not be shared with other installed  
3459 contributions.
  - 3460 ○ When the snapshot of any contribution is taken is implementation defined, ranging  
3461 from the time the contribution is installed to the time of execution
- 3462 • Deployment-time composites.  
3463 These are composites that are added into an installed contribution after it has been  
3464 deployed. This makes it possible to provide final configuration and access to  
3465 implementations within a contribution without having to modify the contribution. These  
3466 are optional, as composites that already exist within the contribution can also be used for  
3467 deployment.

3468

3469 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3470 fully qualified class names in Java).

3471 If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3472 algorithm used to determine the qualified name to use is implementation dependent.

3473 Implementations of SCA MAY also generate an error if there are conflicting names exported from  
3474 multiple contributions. [ASM12007]

3475

### 3476 **12.3.1 Installed Artifact URIs**

3477 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3478 constructed by starting with the base URI of the contribution and adding the relative URI of each  
3479 artifact (recalling that SCA requires that any packaging format be able to offer up its artifacts in a  
3480 single hierarchy).

3481

## 3482 12.4 Operations for Contributions

3483 SCA Domains provide the following conceptual functionality associated with contributions  
3484 (meaning the function might not be represented as addressable services and also meaning that  
3485 equivalent functionality might be provided in other ways). The functionality is optional meaning  
3486 that some SCA runtimes MAY choose not to provide the contribution functions functionality in any  
3487 way. [ASM12008]

### 3488 12.4.1 install Contribution & update Contribution

3489 Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3490 supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3491 contributions that should be used to resolve the dependencies of the root contribution and other  
3492 dependent contributions. These override any dependent contributions explicitly listed via the  
3493 location attribute in the import statements of the contribution.

3494 SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3495 means that all other exported artifacts can be used from that contribution. Because of this, the  
3496 dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3497 what is being used from each one.

3498 Each dependent contribution is also an installed contribution, with its own dependent  
3499 contributions. By default these dependent contributions of the dependent contributions (which we  
3500 will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3501 contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3502 definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3503 included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3504 Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3505 MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3506 Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3507 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,  
3508 then it is easy for this list to be generated by tooling.

### 3509 12.4.2 add Deployment Composite & update Deployment Composite

3510 Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3511 data structure, not an existing resource in the domain) to the contribution identified by a supplied  
3512 contribution URI. The added or updated deployment composite is given a relative URI that  
3513 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3514 must run within the context of a installed contribution (any component implementations or other  
3515 definitions are resolved within that contribution), this functionality makes it possible for the  
3516 deployer to create a composite with final configuration and wiring decisions and add it to an  
3517 installed contribution without having to modify the contents of the root contribution.

3518 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3519 It is then possible for those to be given component names by a (possibly generated) composite  
3520 that is added into the installed contribution, without having to modify the packaging.

### 3521 12.4.3 remove Contribution

3522 Removes the deployed contribution identified by a supplied contribution URI.

3523

## 3524 12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3525

3526 For certain types of artifact, there are existing and commonly used mechanisms for referencing a  
3527 specific concrete location where the artifact can be resolved.

3528 Examples of these mechanisms include:

- 3529 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the  
3530 place holding the WSDL itself.
- 3531 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a  
3532 URI where the XSD is found.

3533 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does  
3534 not have to be dereferenced.

3535 SCA permits the use of these mechanisms. Where present, non-SCA artifact resolution  
3536 mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.  
3537 [ASM12010] However, use of these mechanisms is discouraged because tying assemblies to  
3538 addresses in this way makes the assemblies less flexible and prone to errors when changes are  
3539 made to the overall SCA Domain.

3540 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to  
3541 find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the  
3542 SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an  
3543 alternative. [ASM12011]

3544

## 3545 12.6 Domain-Level Composite

3546 The domain-level composite is a virtual composite, in that it is not defined by a composite  
3547 definition document. Rather, it is built up and modified through operations on the domain.  
3548 However, in other respects it is very much like a composite, since it contains components, wires,  
3549 services and references.

3550

3551 The value of @autowire for the logical domain composite MUST be autowire="false". [ASM12012]

3552

3553 For components at the Domain level, with References for which @autowire="true" applies, the  
3554 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

3555 1) The SCA runtime MAY disallow deployment of any components with autowire References. In  
3556 this case, the SCA runtime MUST generate an exception at the point where the component is  
3557 deployed.

3558 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component  
3559 is deployed and not update those targets when later deployment actions occur.

3560 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later  
3561 deployment actions occur resulting in updated reference targets which match the new Domain  
3562 configuration. How the new configuration of the reference takes place is described by the relevant  
3563 client and implementation specifications.

3564 [ASM12013]

3565 The abstract domain-level functionality for modifying the domain-level composite is as follows,  
3566 although a runtime may supply equivalent functionality in a different form:

### 3567 12.6.1 add To Domain-Level Composite

3568 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.  
3569 The supplied composite URI must refer to a composite within a installed contribution. The  
3570 composite's installed contribution determines how the composite's artifacts are resolved (directly  
3571 and indirectly). The supplied composite is added to the domain composite with semantics that  
3572 correspond to the domain-level composite having an <include> statement that references the

**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.

3573 supplied composite. All of the composite's components become *top-level* components and the  
3574 services become externally visible services (eg. they would be present in a WSDL description of  
3575 the domain).

### 3576 **12.6.2 remove From Domain-Level Composite**

3577 Removes from the Domain Level composite the elements corresponding to the composite  
3578 identified by a supplied composite URI. This means that the removal of the components, wires,  
3579 services and references originally added to the domain level composite by the identified  
3580 composite.

### 3581 **12.6.3 get Domain-Level Composite**

3582 Returns a <composite> definition that has an <include> line for each composite that had been  
3583 added to the domain level composite. It is important to note that, in dereferencing the included  
3584 composites, any referenced artifacts must be resolved in terms of that installed composite.

### 3585 **12.6.4 get QName Definition**

3586 In order to make sense of the domain-level composite (as returned by get Domain-Level  
3587 Composite), it must be possible to get the definitions for named artifacts in the included  
3588 composites. This functionality takes the supplied URI of an installed contribution (which provides  
3589 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
3590 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for  
3591 that definition type.

3592 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
3593 should exist in some form, but not necessarily as a service operation with exactly this signature.

---

3594 **13 Conformance**

- 3595 The XML schema available at the namespace URI, defined by this specification, is considered to be  
3596 authoritative and takes precedence over the XML Schema defined in the appendix of this document.
- 3597 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema  
3598 **[ASM10001]**

3599

## A. Pseudo Schema

### A.1 ComponentType

```
3601 <?xml version="1.0" encoding="ASCII"?>
3602 <!-- Component type schema snippet -->
3603 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3604     constrainingType="QName"? >
3605
3606     <service name="xs:NCName" requires="list of xs:QName"?
3607         policySets="list of xs:QName"?>*
3608         <interface ... />
3609         <binding uri="xs:anyURI"? name="xs:NCName"?
3610             requires="list of xs:QName"?
3611             policySets="list of xs:QName"?/>*
3612         <callback>?
3613             <binding ... />+
3614         </callback>
3615     </service>
3616
3617     <reference name="xs:NCName"
3618         target="list of xs:anyURI"? autowire="xs:boolean"?
3619         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
3620         wiredByImpl="xs:boolean"? requires="list of xs:QName"?
3621         policySets="list of xs:QName"?>*
3622         <interface ... />
3623         <binding uri="xs:anyURI"? name="xs:NCName"?
3624             requires="list of xs:QName"?
3625             policySets="list of xs:QName"?/>*
3626         <callback>?
3627             <binding ... />+
3628         </callback>
3629     </reference>
3630
3631     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3632         many="xs:boolean"? mustSupply="xs:boolean"?
3633         policySets="list of xs:QName"?>*
3634         default-property-value?
3635     </property>
3636
3637     <implementation requires="list of xs:QName"?
3638         policySets="list of xs:QName"?/>?
```

3639  
3640 </componentType>  
3641

## 3642 A.2 Composite

```
3643 <?xml version="1.0" encoding="ASCII"?>
3644 <!-- Composite schema snippet -->
3645 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3646           targetNamespace="xs:anyURI"
3647           name="xs:NCName" local="xs:boolean"?
3648           autowire="xs:boolean"? constrainingType="QName"?
3649           requires="list of xs:QName"? policySets="list of
3650 xs:QName"?>
3651
3652     <include name="xs:QName"/>*
3653
3654     <service name="xs:NCName" promote="xs:anyURI"
3655           requires="list of xs:QName"? policySets="list of xs:QName"?>*
3656       <interface ... />?
3657       <binding uri="xs:anyURI"? name="xs:NCName"?
3658           requires="list of xs:QName"? policySets="list of
3659 xs:QName"?/>*
3660       <callback?
3661           <binding uri="xs:anyURI"? name="xs:NCName"?
3662           requires="list of xs:QName"?
3663           policySets="list of xs:QName"?/>+
3664       </callback>
3665     </service>
3666
3667     <reference name="xs:NCName" target="list of xs:anyURI"?
3668           promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
3669           multiplicity="0..1 or 1..1 or 0..n or 1..n"?
3670           requires="list of xs:QName"? policySets="list of xs:QName"?>*
3671       <interface ... />?
3672       <binding uri="xs:anyURI"? name="xs:NCName"?
3673           requires="list of xs:QName"? policySets="list of
3674 xs:QName"?/>*
3675       <callback?
3676           <binding uri="xs:anyURI"? name="xs:NCName"?
3677           requires="list of xs:QName"?
3678           policySets="list of xs:QName"?/>+
3679       </callback>
3680     </reference>
3681
```

```

3682     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3683         many="xs:boolean"? mustSupply="xs:boolean"?>*
3684         default-property-value?
3685     </property>
3686
3687     <component name="xs:NCName" autowire="xs:boolean"?
3688         requires="list of xs:QName"? policySets="list of xs:QName"?>*
3689         <implementation ... />?
3690         <service name="xs:NCName" requires="list of xs:QName"?
3691             policySets="list of xs:QName"?>*
3692             <interface ... />?
3693             <binding uri="xs:anyURI"? name="xs:NCName"?
3694                 requires="list of xs:QName"?
3695                 policySets="list of xs:QName"?/>*
3696             <callback>?
3697                 <binding uri="xs:anyURI"? name="xs:NCName"?
3698                     requires="list of xs:QName"?
3699                     policySets="list of xs:QName"?/>+
3700             </callback>
3701         </service>
3702         <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3703             source="xs:string"? file="xs:anyURI"? value="xs:string"?>*
3704             [<value>+ | xs:any+]?
3705         </property>
3706         <reference name="xs:NCName" target="list of xs:anyURI"?
3707             autowire="xs:boolean"? wiredByImpl="xs:boolean"?
3708             requires="list of xs:QName"? policySets="list of xs:QName"?
3709             multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
3710             <interface ... />?
3711             <binding uri="xs:anyURI"? name="xs:NCName"?
3712                 requires="list of xs:QName"?
3713                 policySets="list of xs:QName"?/>*
3714             <callback>?
3715                 <binding uri="xs:anyURI"? name="xs:NCName"?
3716                     requires="list of xs:QName"?
3717                     policySets="list of xs:QName"?/>+
3718             </callback>
3719         </reference>
3720     </component>
3721
3722     <wire source="xs:anyURI" target="xs:anyURI" />*
3723
3724 </composite>

```

---

## 3725 B. XML Schemas

### 3726 B.1 sca.xsd

```
3727
3728 <?xml version="1.0" encoding="UTF-8"?>
3729 <!-- (c) Copyright SCA Collaboration 2006 -->
3730 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3731         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3732         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3733
3734     <include schemaLocation="sca-core.xsd"/>
3735
3736     <include schemaLocation="sca-interface-java.xsd"/>
3737     <include schemaLocation="sca-interface-wsdl.xsd"/>
3738
3739     <include schemaLocation="sca-implementation-java.xsd"/>
3740     <include schemaLocation="sca-implementation-composite.xsd"/>
3741
3742     <include schemaLocation="sca-binding-webservice.xsd"/>
3743     <include schemaLocation="sca-binding-jms.xsd"/>
3744     <include schemaLocation="sca-binding-sca.xsd"/>
3745
3746     <include schemaLocation="sca-definitions.xsd"/>
3747     <include schemaLocation="sca-policy.xsd"/>
3748
3749     <include schemaLocation="sca-contribution.xsd"/>
3750
3751 </schema>
```

Comment [mbgl18]: Issue 28

### 3753 B.2 sca-core.xsd

```
3754
3755 <?xml version="1.0" encoding="UTF-8"?>
3756 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
3757 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3758         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3759         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3760         elementFormDefault="qualified">
3761
3762     <element name="componentType" type="sca:ComponentType"/>
3763     <complexType name="ComponentType">
```

```

3764     <sequence>
3765         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
3766         <choice minOccurs="0" maxOccurs="unbounded">
3767             <element name="service" type="sca:ComponentService" />
3768             <element name="reference" type="sca:ComponentReference"/>
3769             <element name="property" type="sca:Property"/>
3770         </choice>
3771         <any namespace="##other" processContents="lax" minOccurs="0"
3772             maxOccurs="unbounded"/>
3773     </sequence>
3774     <attribute name="constrainingType" type="QName" use="optional"/>
3775     <anyAttribute namespace="##other" processContents="lax"/>
3776 </complexType>
3777
3778 <element name="composite" type="sca:Composite"/>
3779 <complexType name="Composite">
3780     <sequence>
3781         <element name="include" type="anyURI" minOccurs="0"
3782             maxOccurs="unbounded"/>
3783         <choice minOccurs="0" maxOccurs="unbounded">
3784             <element name="service" type="sca:Service"/>
3785             <element name="property" type="sca:Property"/>
3786             <element name="component" type="sca:Component"/>
3787             <element name="reference" type="sca:Reference"/>
3788             <element name="wire" type="sca:Wire"/>
3789         </choice>
3790         <any namespace="##other" processContents="lax" minOccurs="0"
3791             maxOccurs="unbounded"/>
3792     </sequence>
3793     <attribute name="name" type="NCName" use="required"/>
3794     <attribute name="targetNamespace" type="anyURI" use="required"/>
3795     <attribute name="local" type="boolean" use="optional"
3796 default="false"/>
3797     <attribute name="autowire" type="boolean" use="optional"
3798 default="false"/>
3799     <attribute name="constrainingType" type="QName" use="optional"/>
3800     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3801     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3802     <anyAttribute namespace="##other" processContents="lax"/>
3803 </complexType>
3804
3805 <complexType name="Service">
3806     <sequence>

```

```

3807     <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3808     <element name="operation" type="sca:Operation" minOccurs="0"
3809         maxOccurs="unbounded" />
3810     <choice minOccurs="0" maxOccurs="unbounded">
3811         <element ref="sca:binding" />
3812         <any namespace="##other" processContents="lax"
3813             minOccurs="0" maxOccurs="unbounded" />
3814     </choice>
3815     <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3816     <any namespace="##other" processContents="lax" minOccurs="0"
3817         maxOccurs="unbounded" />
3818 </sequence>
3819 <attribute name="name" type="NCName" use="required" />
3820 <attribute name="promote" type="anyURI" use="required" />
3821 <attribute name="requires" type="sca:listOfQNames" use="optional" />
3822 <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3823 <anyAttribute namespace="##other" processContents="lax" />
3824 </complexType>
3825
3826 <element name="interface" type="sca:Interface" abstract="true" />
3827 <complexType name="Interface" abstract="true">
3828     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3829     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3830 </complexType>
3831
3832 <complexType name="Reference">
3833     <sequence>
3834         <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3835         <element name="operation" type="sca:Operation" minOccurs="0"
3836             maxOccurs="unbounded" />
3837         <choice minOccurs="0" maxOccurs="unbounded">
3838             <element ref="sca:binding" />
3839             <any namespace="##other" processContents="lax" />
3840         </choice>
3841         <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3842         <any namespace="##other" processContents="lax" minOccurs="0"
3843             maxOccurs="unbounded" />
3844     </sequence>
3845     <attribute name="name" type="NCName" use="required" />
3846     <attribute name="target" type="sca:listOfAnyURIs" use="optional"/>
3847     <attribute name="wiredByImpl" type="boolean" use="optional"
3848         default="false"/>
3849     <attribute name="multiplicity" type="sca:Multiplicity"
3850         use="optional" default="1..1" />

```

Comment [mbgl19]: Issue  
39

```

3851     <attribute name="promote" type="sca:listOfAnyURIs" use="required" />
3852     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3853     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3854     <anyAttribute namespace="##other" processContents="lax" />
3855 </complexType>
3856
3857 <complexType name="SCAPropertyBase" mixed="true">
3858     <!-- mixed="true" to handle simple type -->
3859     <sequence>
3860         <choice minOccurs="0">
3861             <element name="value" minOccurs="1" maxOccurs="unbounded"
3862                 type="anyType" />
3863             <any namespace="##any" processContents="lax" minOccurs="1"
3864                 maxOccurs="unbounded" />
3865             <!-- NOT an extension point; This xsd:any exists
3866                 to accept the element-based or complex type
3867                 property i.e. no element-based extension point
3868                 under "sca:property" -->
3869         </choice>
3870     </sequence>
3871 </complexType>
3872
3873 <!-- complex type for sca:property declaration -->
3874 <complexType name="Property" mixed="true">
3875     <complexContent>
3876         <extension base="sca:SCAPropertyBase">
3877             <!-- extension defines the place to hold default value -->
3878             <attribute name="name" type="NCName" use="required"/>
3879             <attribute name="value" type="xs:string" use="optional"/>
3880             <attribute name="type" type="QName" use="optional"/>
3881             <attribute name="element" type="QName" use="optional"/>
3882             <attribute name="many" type="boolean" default="false"
3883                 use="optional"/>
3884             <attribute name="mustSupply" type="boolean" default="false"
3885                 use="optional"/>
3886             <anyAttribute namespace="##other" processContents="lax"/>
3887             <!-- an extension point ; attribute-based only -->
3888         </extension>
3889     </complexContent>
3890 </complexType>
3891
3892 <complexType name="PropertyValue" mixed="true">
3893     <complexContent>

```

```

3894     <extension base="sca:SCAPropertyBase">
3895         <attribute name="name" type="NCName" use="required"/>
3896         <attribute name="value" type="xs:string" use="optional"/>
3897         <attribute name="type" type="QName" use="optional"/>
3898         <attribute name="element" type="QName" use="optional"/>
3899         <attribute name="many" type="boolean" default="false"
3900             use="optional"/>
3901         <attribute name="source" type="string" use="optional"/>
3902         <attribute name="file" type="anyURI" use="optional"/>
3903         <anyAttribute namespace="##other" processContents="lax"/>
3904         <!-- an extension point ; attribute-based only -->
3905     </extension>
3906 </complexContent>
3907 </complexType>
3908
3909 <element name="binding" type="sca:Binding" abstract="true"/>
3910 <complexType name="Binding" abstract="true">
3911     <sequence>
3912         <element name="operation" type="sca:Operation" minOccurs="0"
3913             maxOccurs="unbounded" />
3914     </sequence>
3915     <attribute name="uri" type="anyURI" use="optional"/>
3916     <attribute name="name" type="NCName" use="optional"/>
3917     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3918     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3919 </complexType>
3920
3921 <element name="bindingType" type="sca:BindingType"/>
3922 <complexType name="BindingType">
3923     <sequence minOccurs="0" maxOccurs="unbounded">
3924         <any namespace="##other" processContents="lax" />
3925     </sequence>
3926     <attribute name="type" type="QName" use="required"/>
3927     <attribute name="alwaysProvides" type="sca:listOfQNames"
3928 use="optional"/>
3929     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
3930     <anyAttribute namespace="##other" processContents="lax"/>
3931 </complexType>
3932
3933 <element name="callback" type="sca:Callback"/>
3934 <complexType name="Callback">
3935     <choice minOccurs="0" maxOccurs="unbounded">
3936         <element ref="sca:binding"/>

```

```

3937         <any namespace="##other" processContents="lax"/>
3938     </choice>
3939     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3940     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3941     <anyAttribute namespace="##other" processContents="lax"/>
3942 </complexType>
3943
3944 <complexType name="Component">
3945     <sequence>
3946         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
3947         <choice minOccurs="0" maxOccurs="unbounded">
3948             <element name="service" type="sca:ComponentService"/>
3949             <element name="reference" type="sca:ComponentReference"/>
3950             <element name="property" type="sca:PropertyValue" />
3951         </choice>
3952         <any namespace="##other" processContents="lax" minOccurs="0"
3953             maxOccurs="unbounded"/>
3954     </sequence>
3955     <attribute name="name" type="NCName" use="required"/>
3956     <attribute name="autowire" type="boolean" use="optional" />
3957     <attribute name="constrainingType" type="QName" use="optional"/>
3958     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3959     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3960     <anyAttribute namespace="##other" processContents="lax"/>
3961 </complexType>
3962
3963 <complexType name="ComponentService">
3964     <complexContent>
3965         <restriction base="sca:Service">
3966             <sequence>
3967                 <element ref="sca:interface" minOccurs="0"
3968 maxOccurs="1"/>
3969                 <element name="operation" type="sca:Operation"
3970 minOccurs="0"
3971                 maxOccurs="unbounded" />
3972                 <choice minOccurs="0" maxOccurs="unbounded">
3973                     <element ref="sca:binding"/>
3974                     <any namespace="##other" processContents="lax"
3975                         minOccurs="0" maxOccurs="unbounded"/>
3976                 </choice>
3977                 <element ref="sca:callback" minOccurs="0"
3978 maxOccurs="1"/>
3979                 <any namespace="##other" processContents="lax"
3980 minOccurs="0"

```

```

3981         maxOccurs="unbounded"/>
3982     </sequence>
3983     <attribute name="name" type="NCName" use="required" />
3984     <attribute name="requires" type="sca:listOfQNames"
3985         use="optional" />
3986     <attribute name="policySets" type="sca:listOfQNames"
3987         use="optional" />
3988     <anyAttribute namespace="##other" processContents="lax" />
3989 </restriction>
3990 </complexContent>
3991 </complexType>
3992
3993 <complexType name="ComponentReference">
3994     <complexContent>
3995         <restriction base="sca:Reference">
3996             <sequence>
3997                 <element ref="sca:interface" minOccurs="0"
3998 maxOccurs="1" />
3999                 <element name="operation" type="sca:Operation"
4000 minOccurs="0"
4001                 maxOccurs="unbounded" />
4002                 <choice minOccurs="0" maxOccurs="unbounded">
4003                     <element ref="sca:binding" />
4004                     <any namespace="##other" processContents="lax"
4005 />
4006                 </choice>
4007                 <element ref="sca:callback" minOccurs="0"
4008 maxOccurs="1" />
4009                 <any namespace="##other" processContents="lax"
4010 minOccurs="0"
4011                 maxOccurs="unbounded" />
4012             </sequence>
4013             <attribute name="name" type="NCName" use="required" />
4014             <attribute name="autowire" type="boolean" use="optional" />
4015             <attribute name="wiredByImpl" type="boolean" use="optional"
4016                 default="false" />
4017             <attribute name="target" type="sca:listOfAnyURIs"
4018 use="optional" />
4019             <attribute name="multiplicity" type="sca:Multiplicity"
4020                 use="optional" default="1..1" />
4021             <attribute name="requires" type="sca:listOfQNames"
4022 use="optional" />
4023             <attribute name="policySets" type="sca:listOfQNames"
4024                 use="optional" />
4025             <anyAttribute namespace="##other" processContents="lax" />

```

```

4026         </restriction>
4027     </complexContent>
4028 </complexType>
4029
4030 <element name="implementation" type="sca:Implementation"
4031     abstract="true" />
4032 <complexType name="Implementation" abstract="true">
4033     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4034     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4035 </complexType>
4036
4037 <element name="implementationType" type="sca:ImplementationType"/>
4038 <complexType name="ImplementationType">
4039     <sequence minOccurs="0" maxOccurs="unbounded">
4040         <any namespace="##other" processContents="lax" />
4041     </sequence>
4042     <attribute name="type" type="QName" use="required"/>
4043     <attribute name="alwaysProvides" type="sca:listOfQNames"
4044 use="optional"/>
4045     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
4046     <anyAttribute namespace="##other" processContents="lax"/>
4047 </complexType>
4048
4049 <complexType name="Wire">
4050     <sequence>
4051         <any namespace="##other" processContents="lax" minOccurs="0"
4052     maxOccurs="unbounded"/>
4053     </sequence>
4054     <attribute name="source" type="anyURI" use="required"/>
4055     <attribute name="target" type="anyURI" use="required"/>
4056     <anyAttribute namespace="##other" processContents="lax"/>
4057 </complexType>
4058
4059 <element name="include" type="sca:Include"/>
4060 <complexType name="Include">
4061     <attribute name="name" type="QName"/>
4062     <anyAttribute namespace="##other" processContents="lax"/>
4063 </complexType>
4064
4065 <complexType name="Operation">
4066     <attribute name="name" type="NCName" use="required"/>
4067     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4068     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>

```

```

4069     <anyAttribute namespace="##other" processContents="lax"/>
4070 </complexType>
4071
4072 <element name="constrainingType" type="sca:ConstrainingType"/>
4073 <complexType name="ConstrainingType">
4074     <sequence>
4075         <choice minOccurs="0" maxOccurs="unbounded">
4076             <element name="service" type="sca:ComponentService"/>
4077             <element name="reference" type="sca:ComponentReference"/>
4078             <element name="property" type="sca:Property" />
4079         </choice>
4080         <any namespace="##other" processContents="lax" minOccurs="0"
4081             maxOccurs="unbounded" />
4082     </sequence>
4083     <attribute name="name" type="NCName" use="required"/>
4084     <attribute name="targetNamespace" type="anyURI"/>
4085     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4086     <anyAttribute namespace="##other" processContents="lax"/>
4087 </complexType>
4088
4089
4090 <simpleType name="Multiplicity">
4091     <restriction base="string">
4092         <enumeration value="0..1"/>
4093         <enumeration value="1..1"/>
4094         <enumeration value="0..n"/>
4095         <enumeration value="1..n"/>
4096     </restriction>
4097 </simpleType>
4098
4099 <simpleType name="OverrideOptions">
4100     <restriction base="string">
4101         <enumeration value="no"/>
4102         <enumeration value="may"/>
4103         <enumeration value="must"/>
4104     </restriction>
4105 </simpleType>
4106
4107 <!-- Global attribute definition for @requires to permit use of intents
4108     within WSDL documents -->
4109 <attribute name="requires" type="sca:listOfQNames"/>
4110
4111 <!-- Global attribute defintion for @endsConversation to mark operations

```

```

4112         as ending a conversation -->
4113 <attribute name="endsConversation" type="boolean" default="false"/>
4114
4115 <simpleType name="listOfQNames">
4116     <list itemType="QName"/>
4117 </simpleType>
4118
4119 <simpleType name="listOfAnyURIs">
4120     <list itemType="anyURI"/>
4121 </simpleType>
4122
4123 </schema>

```

### 4124 **B.3 sca-binding-sca.xsd**

```

4125
4126 <?xml version="1.0" encoding="UTF-8"?>
4127 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
4128 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4129     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4130     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4131     elementFormDefault="qualified">
4132
4133     <include schemaLocation="sca-core.xsd"/>
4134
4135     <element name="binding.sca" type="sca:SCABinding"
4136         substitutionGroup="sca:binding"/>
4137     <complexType name="SCABinding">
4138         <complexContent>
4139             <extension base="sca:Binding">
4140                 <sequence>
4141                     <element name="operation" type="sca:Operation"
4142 minOccurs="0"
4143                         maxOccurs="unbounded" />
4144                 </sequence>
4145                 <attribute name="uri" type="anyURI" use="optional"/>
4146                 <attribute name="name" type="QName" use="optional"/>
4147                 <attribute name="requires" type="sca:listOfQNames"
4148                     use="optional"/>
4149                 <attribute name="policySets" type="sca:listOfQNames"
4150                     use="optional"/>
4151                 <anyAttribute namespace="##other" processContents="lax"/>
4152             </extension>
4153         </complexContent>

```

```
4154     </complexType>
4155 </schema>
4156
```

## 4157 **B.4 sca-interface-java.xsd**

```
4158
4159 <?xml version="1.0" encoding="UTF-8"?>
4160 <!-- (c) Copyright SCA Collaboration 2006 -->
4161 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4162     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4163     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4164     elementFormDefault="qualified">
4165
4166     <include schemaLocation="sca-core.xsd"/>
4167
4168     <element name="interface.java" type="sca:JavaInterface"
4169         substitutionGroup="sca:interface"/>
4170     <complexType name="JavaInterface">
4171         <complexContent>
4172             <extension base="sca:Interface">
4173                 <sequence>
4174                     <any namespace="##other" processContents="lax"
4175 minOccurs="0" maxOccurs="unbounded"/>
4176                 </sequence>
4177                 <attribute name="interface" type="NCName" use="required"/>
4178                 <attribute name="callbackInterface" type="NCName"
4179 use="optional"/>
4180                 <anyAttribute namespace="##other" processContents="lax"/>
4181             </extension>
4182         </complexContent>
4183     </complexType>
4184 </schema>
4185
```

## 4186 **B.5 sca-interface-wsdl.xsd**

```
4187
4188 <?xml version="1.0" encoding="UTF-8"?>
4189 <!-- (c) Copyright SCA Collaboration 2006 -->
4190 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4191     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4192     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4193     elementFormDefault="qualified">
4194
4195     <include schemaLocation="sca-core.xsd"/>
```

```

4196
4197     <element name="interface.wsdl" type="sca:WSDLPortType"
4198           substitutionGroup="sca:interface"/>
4199   <complexType name="WSDLPortType">
4200     <complexContent>
4201       <extension base="sca:Interface">
4202         <sequence>
4203           <any namespace="##other" processContents="lax"
4204 minOccurs="0"           maxOccurs="unbounded"/>
4205         </sequence>
4206         <attribute name="interface" type="anyURI" use="required"/>
4207         <attribute name="callbackInterface" type="anyURI"
4208 use="optional"/>
4209         <anyAttribute namespace="##other" processContents="lax"/>
4210       </extension>
4211     </complexContent>
4212   </complexType>
4213 </schema>
4214

```

## 4215 **B.6 sca-implementation-java.xsd**

```

4216
4217 <?xml version="1.0" encoding="UTF-8"?>
4218 <!-- (c) Copyright SCA Collaboration 2006 -->
4219 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4220   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4221   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4222   elementFormDefault="qualified">
4223
4224   <include schemaLocation="sca-core.xsd"/>
4225
4226   <element name="implementation.java" type="sca:JavaImplementation"
4227     substitutionGroup="sca:implementation"/>
4228   <complexType name="JavaImplementation">
4229     <complexContent>
4230       <extension base="sca:Implementation">
4231         <sequence>
4232           <any namespace="##other" processContents="lax"
4233 minOccurs="0" maxOccurs="unbounded"/>
4234         </sequence>
4235         <attribute name="class" type="NCName" use="required"/>
4236         <attribute name="requires" type="sca:listOfQNames"
4237 use="optional"/>
4238         <attribute name="policySets" type="sca:listOfQNames"

```

```

4239         use="optional"/>
4240     <anyAttribute namespace="##other" processContents="lax"/>
4241 </extension>
4242 </complexContent>
4243 </complexType>
4244 </schema>

```

## 4245 **B.7 sca-implementation-composite.xsd**

```

4246
4247 <?xml version="1.0" encoding="UTF-8"?>
4248 <!-- (c) Copyright SCA Collaboration 2006 -->
4249 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4250     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4251     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4252     elementFormDefault="qualified">
4253
4254     <include schemaLocation="sca-core.xsd"/>
4255     <element name="implementation.composite" type="sca:SCAImplementation"
4256         substitutionGroup="sca:implementation"/>
4257     <complexType name="SCAImplementation">
4258         <complexContent>
4259             <extension base="sca:Implementation">
4260                 <sequence>
4261                     <any namespace="##other" processContents="lax"
4262 minOccurs="0"
4263                         maxOccurs="unbounded"/>
4264                 </sequence>
4265                 <attribute name="name" type="QName" use="required"/>
4266                 <attribute name="requires" type="sca:listOfQNames"
4267 use="optional"/>
4268                 <attribute name="policySets" type="sca:listOfQNames"
4269 use="optional"/>
4270                 <anyAttribute namespace="##other" processContents="lax"/>
4271             </extension>
4272         </complexContent>
4273     </complexType>
4274 </schema>
4275

```

## 4276 **B.8 sca-definitions.xsd**

```

4277
4278 <?xml version="1.0" encoding="UTF-8"?>
4279 <!-- (c) Copyright SCA Collaboration 2006 -->

```

```

4280 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4281     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4282     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4283     elementFormDefault="qualified">
4284
4285     <include schemaLocation="sca-core.xsd"/>
4286
4287     <element name="definitions">
4288         <complexType>
4289             <choice minOccurs="0" maxOccurs="unbounded">
4290                 <element ref="sca:intent"/>
4291                 <element ref="sca:policySet"/>
4292                 <element ref="sca:binding"/>
4293                 <element ref="sca:bindingType"/>
4294                 <element ref="sca:implementationType"/>
4295                 <any namespace="##other" processContents="lax" minOccurs="0"
4296                     maxOccurs="unbounded"/>
4297             </choice>
4298         </complexType>
4299     </element>
4300 </schema>
4301
4302

```

## 4303 **B.9 sca-binding-webservice.xsd**

4304 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4305 **B.10 sca-binding-jms.xsd**

4306 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4307 **B.11 sca-policy.xsd**

4308 Is described in [the SCA Policy Framework specification \[10\]](#)

4309

## 4310 **B.12 sca-contribution.xsd**

4311

```

4312 <?xml version="1.0" encoding="UTF-8"?>
4313 <!-- (c) Copyright SCA Collaboration 2007 -->
4314 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4315     targetNamespace="http://www.osoa.org/xmlns/sca/1.0"
4316     xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
4317     elementFormDefault="qualified">
4318
4319     <include schemaLocation="sca-core.xsd"/>
4320
4321

```

**Comment [mbgl20]:** Issue 28

```

4322     <element name="contribution" type="sca:ContributionType"/>
4323     <complexType name="ContributionType">
4324         <sequence>
4325             <element name="deployable" type="sca:DeployableType"
4326 minOccurs="1" maxOccurs="unbounded"/>
4327             <element name="import" type="sca:ImportType" minOccurs="0"
4328 maxOccurs="unbounded"/>
4329             <element name="export" type="sca:ExportType" minOccurs="0"
4330 maxOccurs="unbounded"/>
4331             <any namespace="##other" processContents="lax" minOccurs="0"
4332 maxOccurs="unbounded"/>
4333         </sequence>
4334         <anyAttribute namespace="##other" processContents="lax"/>
4335     </complexType>
4336
4337
4338
4339     <complexType name="DeployableType">
4340         <sequence>
4341             <any namespace="##other" processContents="lax" minOccurs="0"
4342 maxOccurs="unbounded"/>
4343         </sequence>
4344         <attribute name="composite" type="QName" use="required"/>
4345         <anyAttribute namespace="##other" processContents="lax"/>
4346     </complexType>
4347
4348
4349     <complexType name="ImportType">
4350         <sequence>
4351             <any namespace="##other" processContents="lax" minOccurs="0"
4352 maxOccurs="unbounded"/>
4353         </sequence>
4354         <attribute name="namespace" type="string" use="required"/>
4355         <attribute name="location" type="anyURI" use="required"/>
4356         <anyAttribute namespace="##other" processContents="lax"/>
4357     </complexType>
4358
4359     <complexType name="ExportType">
4360         <sequence>
4361             <any namespace="##other" processContents="lax" minOccurs="0"
4362 maxOccurs="unbounded"/>
4363         </sequence>
4364         <attribute name="namespace" type="string" use="required"/>
4365         <anyAttribute namespace="##other" processContents="lax"/>
4366     </complexType>
4367 </schema>
4368
4369

```

---

4370

## C. SCA Concepts

4371

### C.1 Binding

4372

**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.

4373

4374

4375

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.

4376

4377

4378

4379

### C.2 Component

4380

**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.

4381

4382

4383

4384

4385

4386

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.

4387

4388

4389

4390

4391

### C.3 Service

4392

**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.

4393

4394

4395

4396

4397

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.

4398

4399

4400

4401

4402

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.

4403

4404

4405

4406

#### C.3.1 Remotable Service

4407

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.

4408

4409

4410

4411

A service is remotable if it is defined by a WSDL port type or if it defined by a Java interface marked with the **@Remotable** annotation.

4412

### 4413 C.3.2 Local Service

4414 Local services are services that are designed to be only used “locally” by other implementations that are  
4415 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4416 Local services may rely on by-reference calling conventions, or may assume a very fine-grained  
4417 interaction style that is incompatible with remote distribution. They may also use technology-specific data-  
4418 types.

4419 Currently a service is local only if it defined by a Java interface not marked with the @Remotable  
4420 annotation.

4421

### 4422 C.4 Reference

4423 **SCA references** represent a dependency that an implementation has on a service that is supplied by  
4424 some other implementation, where the service to be used is specified through configuration. In other  
4425 words, a reference is a service that an implementation may call during the execution of its business  
4426 function. References are typed by an interface.

4427 For composites, composite references can be accessed by components within the composite like any  
4428 service provided by a component within the composite. Composite references can be used as the targets  
4429 of wires from component references when configuring Components.

4430 A composite reference can be used to access a service such as: an SCA service provided by another  
4431 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,  
4432 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4433 an **extensibility mechanism** that allows the introduction of new binding types to references.

4434

### 4435 C.5 Implementation

4436 An implementation is concept that is used to describe a piece of software technology such as a Java  
4437 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4438 service-oriented application. An SCA composite is also an implementation.

4439 Implementations define points of variability including properties that can be set and settable references to  
4440 other services. The points of variability are configured by a component that uses the implementation. The  
4441 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4442 C.6 Interface

4443 **Interfaces** define one or more business functions. These business functions are provided by Services  
4444 and are used by components through References. Services are defined by the Interface they implement.  
4445 SCA currently supports a number of interface type systems, for example:

- 4446 • Java interfaces
- 4447 • WSDL portTypes
- 4448 • C, C++ header files

4449

4450 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4451 interface type systems.

4452 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided  
4453 by each end of a service communication – this could be the case where a particular service requires a  
4454 “callback” interface on the client, which is calls during the process of handing service requests from the  
4455 client.

4456

## 4457 C.7 Composite

4458 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4459 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4460 be used to contribute elements to an **SCA Domain**.

4461 A **composite** has the following characteristics:

- 4462 • It may be used as a component implementation. When used in this way, it defines a boundary for  
4463 Component visibility. Components may not be directly referenced from outside of the composite  
4464 in which they are declared.
- 4465 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
4466 artifacts to an SCA domain.

4467

## 4468 C.8 Composite inclusion

4469 One composite can be used to provide part of the definition of another composite, through the process of  
4470 inclusion. This is intended to make team development of large composites easier. Included composites  
4471 are merged together into the using composite at deployment time to form a single logical composite.

4472 Composites are included into other composites through `<include.../>` elements in the using composite.

4473 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
4474 specific location.

4475

## 4476 C.9 Property

4477 **Properties** allow for the configuration of an implementation with externally set data values. The data  
4478 value is provided through a Component, possibly sourced from the property of a containing composite.

4479 Each Property is defined by the implementation. Properties may be defined directly through the  
4480 implementation language or through annotations of implementations, where the implementation language  
4481 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
4482 type. For complex data types, XML schema is the preferred technology for defining the data types.

4483

## 4484 C.10 Domain

4485 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
4486 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
4487 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
4488 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4489 A domain specifies the instantiation, configuration and connection of a set of components, provided via  
4490 one or more composite files. The domain, like a composite, also has Services and References. Domains  
4491 also contain Wires which connect together the Components, Services and References.

4492

## 4493 C.11 Wire

4494 **SCA wires** connect **service references** to **services**.

4495 Within a composite, valid wire sources are component references and composite services. Valid wire  
4496 targets are component services and composite references.

4497 When using included composites, the sources and targets of the wires don't have to be declared in the  
4498 same composite as the composite that contains the wire. The sources and targets can be defined by  
4499 other included composites. Targets can also be external to the SCA domain.

4500

4501

## D. Conformance Items

4502

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

4503

Conformance ID	Description
[ASM10001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.
[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.
[ASM50001]	The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
[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/>
[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.
[ASM50017]	When the reference has a value specified in its @target attribute, one of the child binding elements MUST be used on each wire created by the @target attribute, or the sca binding, if no binding is specified.
[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 attribute, the child binding elements of that reference MUST NOT identify target services using the @uri attribute or using binding specific attributes or elements.
[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.
[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 <b>interface</b> 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.
[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 <i>multiplicity</i> 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 <i>interface</i> 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.
[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.
[ASM60023]	the target service interface MUST be a compatible superset of the reference interface when using autowire to wire a reference (as defined in <a href="#">the section on Wires</a> )
[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.
[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.
[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

Comment [mbgl21]: Issue 57

	the qualified form, otherwise there is an error.
[ASM80001]	The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.0 document OR an interface element of a WSDL 2.0 document.
[ASM80002]	Remotable service Interfaces MUST NOT make use of <b>method or operation overloading</b> .
[ASM80003]	If a remotable service is called locally or remotely, the SCA container MUST ensure 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.
[ASM90001]	For a binding of a <b>reference</b> 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".
[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 found elsewhere, for example in other contributions.

Comment [mbgl22]: Issue 57

[ASM12005]	Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact dependencies.
[ASM12006]	SCA requires that all runtimes MUST support the ZIP packaging format for contributions.
[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.
[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.
[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.

Comment [mbgl23]: Issue 42

Comment [mbgl24]: Issue 40

4505

---

## E. Acknowledgements

4506 The following individuals have participated in the creation of this specification and are gratefully  
4507 acknowledged:

4508 **Participants:**

4509 [Participant Name, Affiliation | Individual Member]

4510 [Participant Name, Affiliation | Individual Member]

4511

---

## F. Non-Normative Text

4513

## G. Revision History

4514 [optional; should not be included in OASIS Standards]

4515

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"> <li>- changed order of subsections from property, reference, service to service, reference, property</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> <li>- added section in appendix to contain complete pseudo schema of composite</li> </ul> <p>- moved component section after implementation section</p> <ul style="list-style-type: none"> <li>- made the ConstrainingType section a top level section</li> <li>- moved interface section to after constraining type section</li> </ul> <p>component section</p> <ul style="list-style-type: none"> <li>- added subheadings for Implementation, Service, Reference, Property</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> </ul> <p>implementation section</p> <ul style="list-style-type: none"> <li>- changed title to "Implementation and ComponentType"</li> <li>- moved implementation instance related stuff from implementation section to component implementation section</li> <li>- added subheadings for Service, Reference, Property, Implementation</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> <li>- attribute and element description still needs to be completed, all implementation statements</li> </ul>

			<p>on services, references, and properties should go here</p> <ul style="list-style-type: none"> <li>- added complete pseudo schema of componentType in appendix</li> <li>- added "Quick Tour by Sample" section, no content yet</li> <li>- added comment to introduction section that the following text needs to be added <ul style="list-style-type: none"> <li>"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."</li> </ul> </li> </ul>
3	2008-02-15	Anish Karmarkar Michael Beisiegel	<p>Incorporated resolutions from 2008 Jan f2f.</p> <ul style="list-style-type: none"> <li>- issue 9</li> <li>- issue 19</li> <li>- issue 21</li> <li>- issue 4</li> <li>- issue 1A</li> <li>- issue 27</li> <li>- in Implementation and ComponentType section added attribute and element description for service, reference, and property</li> <li>- removed comments that helped understand the initial restructuring for WD02</li> <li>- added changes for issue 43</li> <li>- added changes for issue 45, except the changes for policySet and requires attribute on property elements</li> <li>- used the NS <a href="http://docs.oasis-open.org/ns/opencsa/sca/200712">http://docs.oasis-open.org/ns/opencsa/sca/200712</a></li> <li>- updated copyright stmt</li> <li>- added wordings to make PDF normative and xml schema at the NS uri authoritative</li> </ul>
4	2008-04-22	Mike Edwards	<p>Editorial tweaks for CD01 publication:</p> <ul style="list-style-type: none"> <li>- updated URL for spec documents</li> <li>- removed comments from published CD01 version</li> <li>- removed blank pages from body of spec</li> </ul>
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: <a href="http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html">http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html</a></p>
7 CD02 - Rev3	2008-11-18	Mike Edwards	<ul style="list-style-type: none"> <li>• Specification marked for conformance statements. New Appendix (D) added</li> </ul>

			containing a table of all conformance statements. Mass of related minor editorial changes to remove the use of RFC2119 words where not appropriate.

4516