



---

# Service Component Architecture Assembly Model Specification Version 1.1

**Committee Draft 01 Revision 2 + Conformance**

**27th October 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

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 .....	50
6.4.1	Wire Examples .....	52
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
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
Binding .....	75
8.5 Messages containing Data not defined in the Service Interface .....	77
8.6 Form of the URI of a Deployed Binding .....	77
8.6.1 Constructing Hierarchical URIs .....	77
8.6.2 Non-hierarchical URIs.....	78
8.6.3 Determining the URI scheme of a deployed binding .....	78
8.7 SCA Binding.....	79
8.7.1 Example SCA Binding .....	79
8.8 Web Service Binding .....	80
8.9 JMS Binding .....	80
9 SCA Definitions .....	81
10 Extension Model.....	82
10.1 Defining an Interface Type.....	82
10.2 Defining an Implementation Type.....	84
10.3 Defining a Binding Type.....	85
11 Packaging and Deployment.....	88
11.1 Domains.....	88
11.2 Contributions .....	88
11.2.1 SCA Artifact Resolution .....	89
11.2.2 SCA Contribution Metadata Document.....	90
11.2.3 Contribution Packaging using ZIP .....	92
11.3 Installed Contribution .....	92
11.3.1 Installed Artifact URIs .....	92
11.4 Operations for Contributions .....	93
11.4.1 install Contribution & update Contribution .....	93
11.4.2 add Deployment Composite & update Deployment Composite.....	93
11.4.3 remove Contribution .....	93
11.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts.....	93
11.6 Domain-Level Composite.....	94
11.6.1 add To Domain-Level Composite.....	94
11.6.2 remove From Domain-Level Composite .....	95
11.6.3 get Domain-Level Composite.....	95
11.6.4 get QName Definition .....	95
12 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.	Acknowledgements .....	118
E.	Non-Normative Text .....	126
F.	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

## 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.

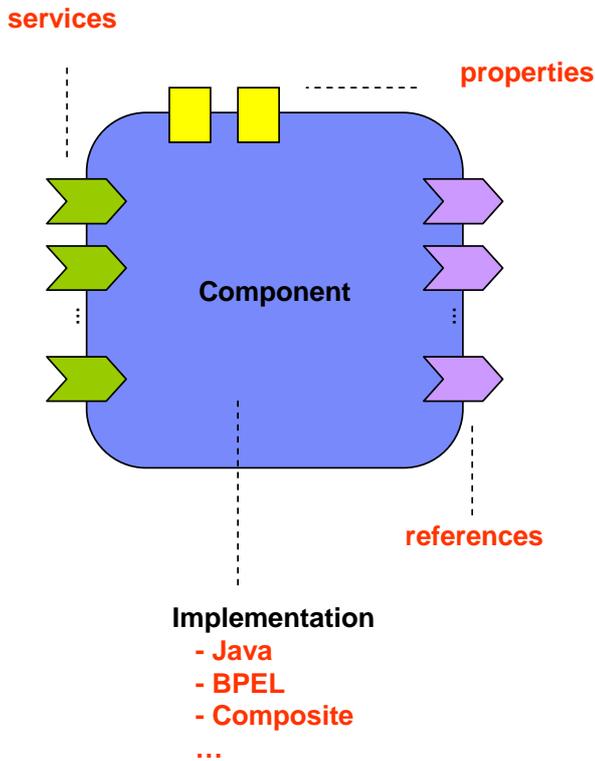
101 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services  
102 providing an area of business functionality that is controlled by a single organization. As an example, for  
103 the accounts department in a business, the SCA Domain might cover all financial 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:



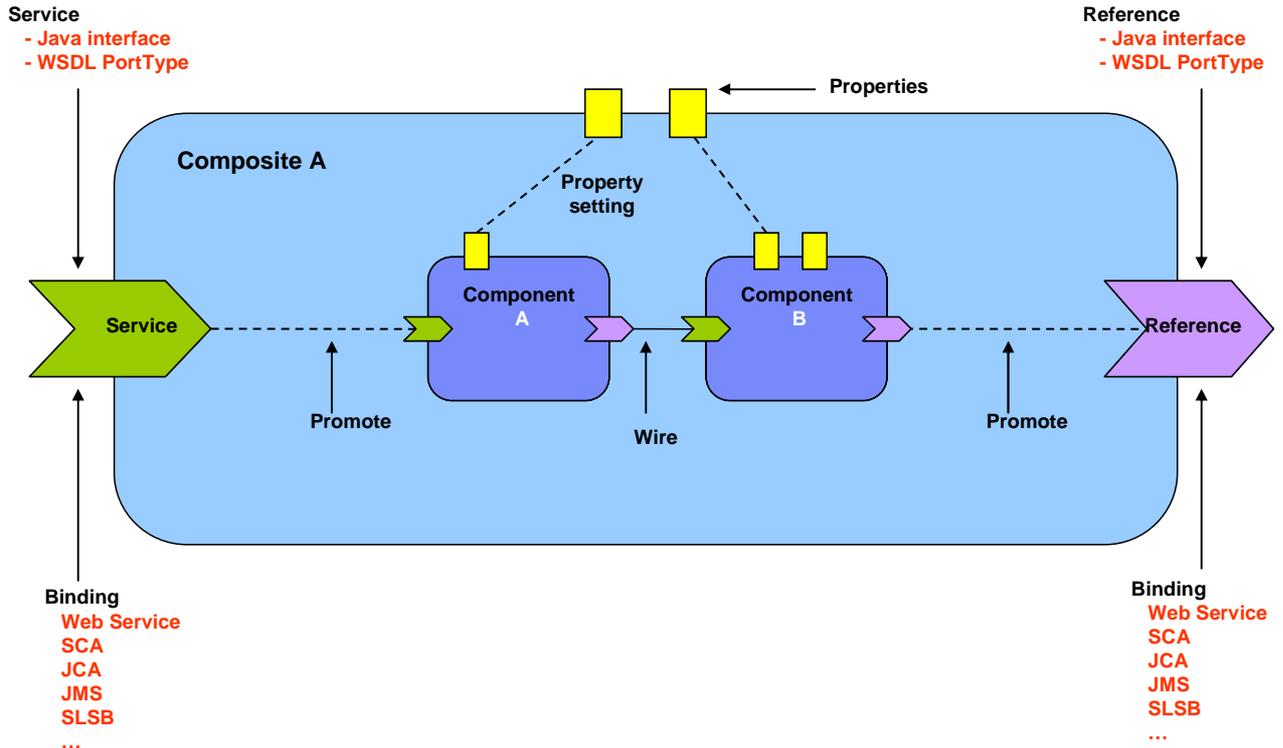
118

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:

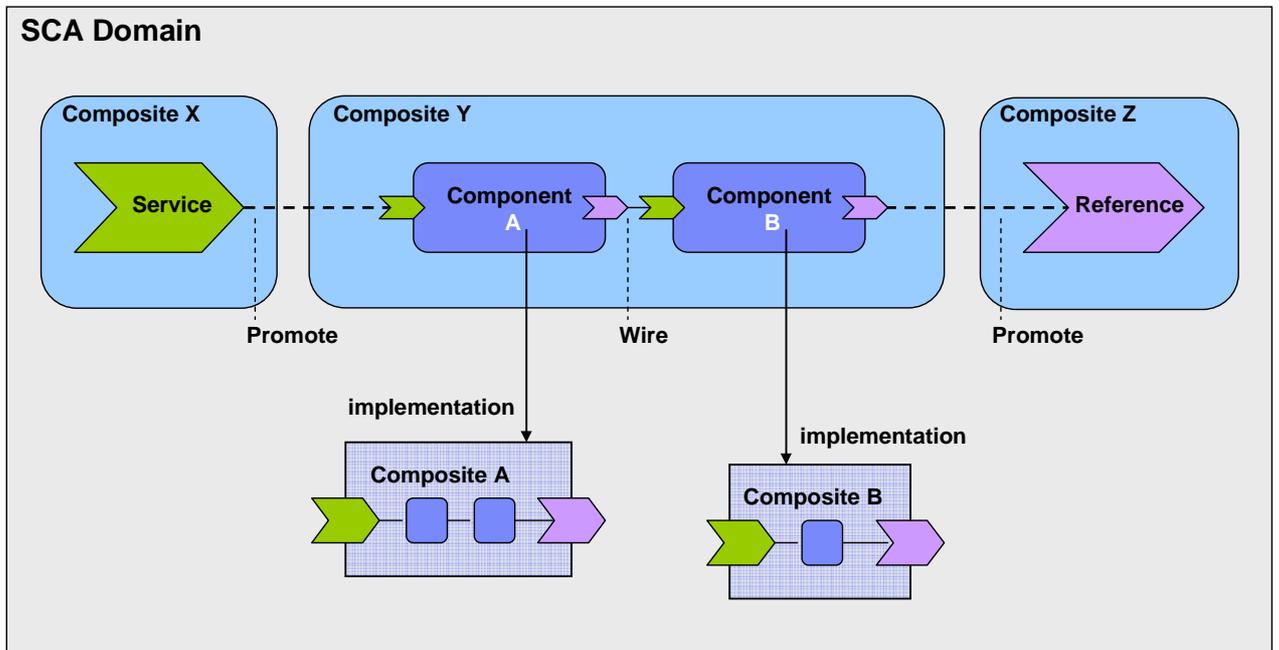
123



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  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183

---

## 4 Implementation and ComponentType

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

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

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

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

So, for example:

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

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

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

### 4.1 Component Type

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

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

In the ideal case, the component type information is determined by inspecting the implementation, for example as code annotations. The component type file provides a mechanism for the provision of component type information for implementation types where the information 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       ▪ **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-  
386       valued (true). In the case of a multi-valued property, it is presented to the implementation  
387       as a collection of property values.
- 388       ▪ **mustSupply : boolean (0..1)** - whether the property value must be supplied by the  
389       component that uses the implementation – when mustSupply="true" the component must  
390       supply a value since the implementation has no default value for the property. A default-  
391       property-value should only be supplied when mustSupply="false" (the default setting for  
392       the mustSupply attribute), since the implication of a default value is that it is used only  
393       when a value is not supplied by the using component.
- 394       ▪ **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property.

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]

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               <service ... />*
420               <reference ... >*
421               <property ... />*
422               
423               <implementation requires="list of xs:QName"?
424                               policySets="list of xs:QName"?/>?
425               
426       </componentType>
```

427

428

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

- 430       • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
431       [\[10\]](#) for a description of this attribute.
- 432       • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
433       [\[10\]](#) for a description of this attribute.

434

## 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
444     <service name="MyValueService">
445         <interface.java interface="services.myvalue.MyValueService"/>
446     </service>
447
448     <reference name="customerService">
449         <interface.java interface="services.customer.CustomerService"/>
450     </reference>
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
```

## 4.3 Example Implementation

460

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.

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" />
```

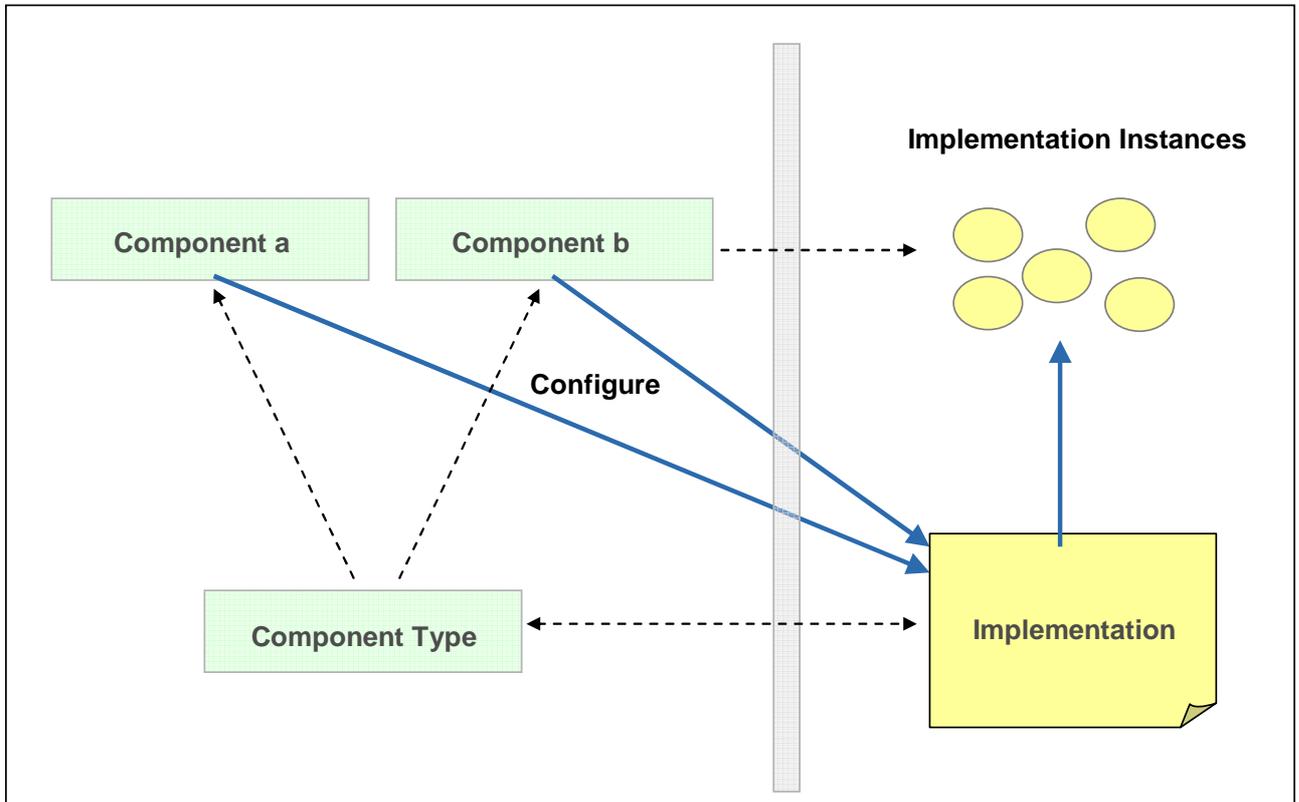
656

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.

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

788 [ASM50009]

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

- 793
- 794
- 795
- 796
- **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on multiplicity setting. Each value wires the reference to a component service that resolves the reference. For more details on wiring see [the section on Wires](#). Overrides any target specified for this reference on the implementation.
  - **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that the implementation wires this reference dynamically. If set to "true" it indicates that the target of the reference is set at runtime by the implementation code (eg by the code obtaining an endpoint reference by some means and setting this as the target of the reference through the use of programming interfaces defined by the relevant Client and Implementation specification). If @wiredByImpl="true" is set for a reference, then the 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 the operations required by the reference. The interface is described by an **interface element** which is a child element of the reference element. If no interface is specified, then the interface specified for the reference in the componentType of the implementation is in effect. 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. [ASM50011] For details on the interface element see [the Interface section](#).

- 815
- **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as children. 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. [ASM50012] Details of the binding element are described in the [Bindings section](#). The binding, combined with any PolicySets in effect for the binding, needs to satisfy the set of policy intents for the reference, as described in [the Policy Framework specification \[10\]](#).

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

- 826
- 827
- **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if

830

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

979

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

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

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

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

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

991 The property type specified must be compatible with the type of the property declared in the  
992 component type of the implementation. If no type is declared in the component property, the type of  
993 the property declared by the implementation is used.

994

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

996

```
997 <?xml version="1.0" encoding="UTF-8"?>
998 <!-- Component Property schema snippet -->
999 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1000   ...
1001   <component ... >*
1002     <implementation ... />?
1003     <service ... />*
1004     <reference ... />*
1005     <property name="xs:NCName"
1006       (type="xs:QName" | element="xs:QName")?
1007       mustSupply="xs:boolean"? many="xs:boolean"?
1008       source="xs:string"? file="xs:anyURI"?
1009       value="xs:string"?>*
1010       [<value>+ | xs:any+ ]?
1011     </property>
1012   </component>
1013   ...
1014 </composite>
```

1015

1016 The **component property** element has the following **attributes**:

- 1017
- 1018
- 1019
- 1020
- 1021
- 1022
- 1023
- 1024
- 1025
- 1026
- 1027
- 1028
- 1029
- 1030
- 1031
- 1032
- 1033
- 1034
- **name : NCName (1..1)** – the name of the property. The name attribute of a component property MUST match the name of a property element in the component type of the component implementation. [\[ASM50031\]](#)
  - zero or one of **(0..1)**:
    - **type : QName** – the type of the property defined as the qualified name of an XML schema type
    - **element : QName** – the type of the property defined as the qualified name of an XML schema global element – the type is the type of the global element
  - **source : string (0..1)** – an XPath expression pointing to a property of the containing composite from which the value of this component property is obtained.
  - **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
  - **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or multi-valued (true). Overrides the many specified for this property on the implementation. The value can only be equal or further restrict, i.e. if the implementation specifies many true, then the component can say false. In the case of a multi-valued property, it is presented to the implementation as a Collection of property values.
  - **value : string (0..1)** - the value of the property if the property is defined using a simple type.

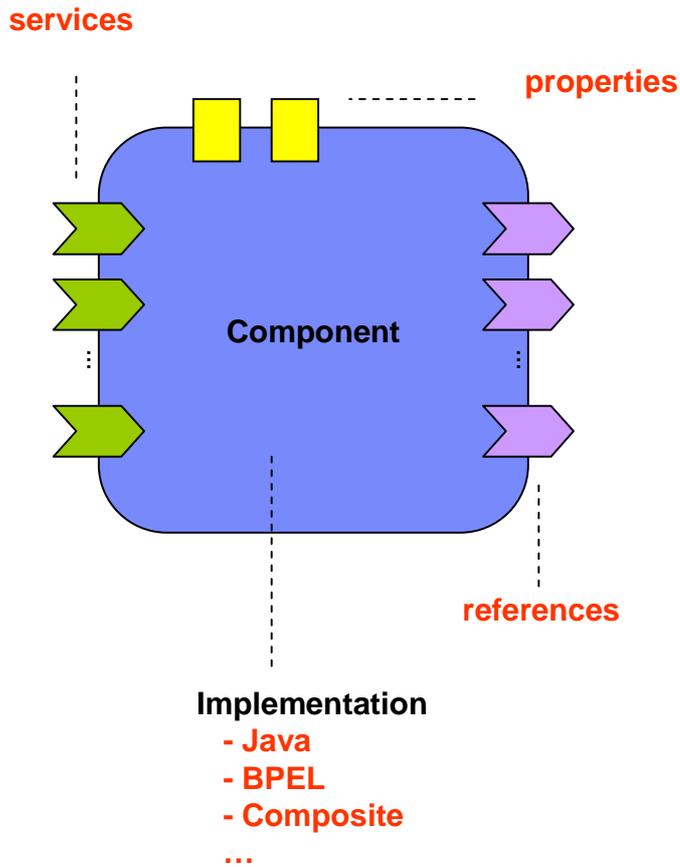
1035 The **component property** element has the following **child element**:

1036 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1037 property that is defined using a XML Schema type. If a property is single-valued, the <value/>  
1038 subelement MUST NOT occur more than once. [\[ASM50032\]](#) A property <value/> subelement MUST  
1039 NOT be used when the @value attribute is used to specify the value for that property. [\[ASM50033\]](#).

## 1040 5.5 Example Component

1041

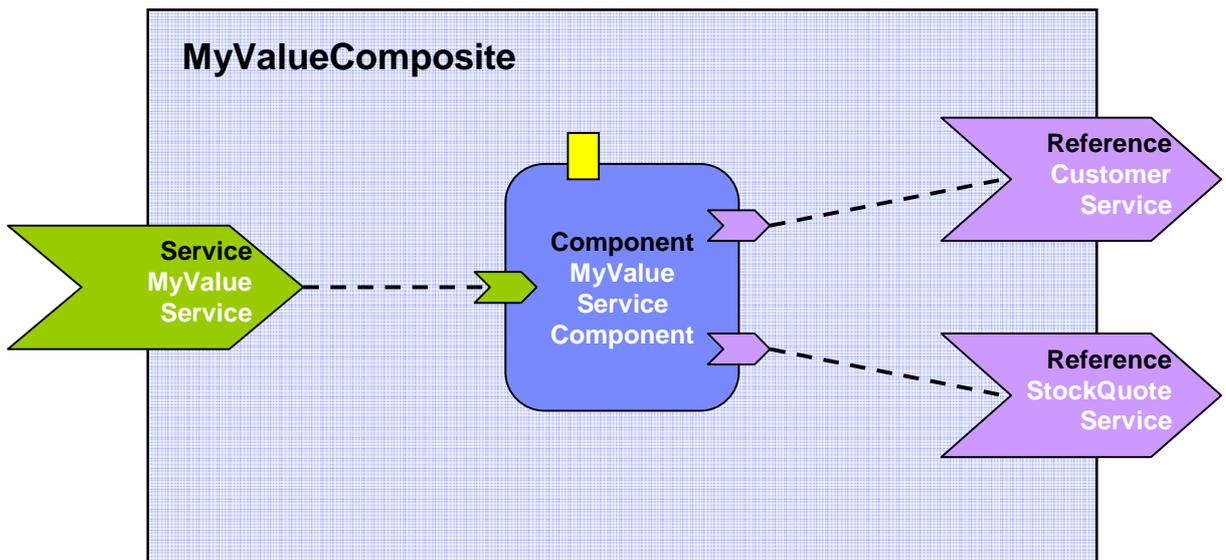
1042 The following figure shows the **component symbol** that is used to represent a component in an  
1043 assembly diagram.



1044  
 1045  
 1046  
 1047  
 1048

Figure 5: Component symbol

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



1049  
 1050  
 1051

Figure 6: Assembly diagram for MyValueComposite

1052

1053 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1054 containing the component element for the MyValueServiceComponent. A value is set for the  
1055 property named currency, and the customerService and stockQuoteService references are  
1056 promoted:

1057

```
1058 <?xml version="1.0" encoding="ASCII"?>
1059 <!-- MyValueComposite_1 example -->
1060 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1061           targetNamespace="http://foo.com"
1062           name="MyValueComposite" >
1063
1064     <service name="MyValueService" promote="MyValueServiceComponent" />
1065
1066     <component name="MyValueServiceComponent">
1067       <implementation.java
1068 class="services.myvalue.MyValueServiceImpl" />
1069       <property name="currency">EURO</property>
1070       <reference name="customerService" />
1071       <reference name="stockQuoteService" />
1072     </component>
1073
1074     <reference name="CustomerService"
1075               promote="MyValueServiceComponent/customerService" />
1076
1077     <reference name="StockQuoteService"
1078               promote="MyValueServiceComponent/stockQuoteService" />
1079
1080 </composite>
```

1081

1082 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of  
1083 clarity – the references are defined by the MyValueServiceImpl implementation and there is no  
1084 need to redeclare them on the component unless the intention is to wire them or to override some  
1085 aspect of them.

1086 The following snippet gives an example of the layout of a composite file if both the currency  
1087 property and the customerService reference of the MyValueServiceComponent are declared to be  
1088 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1089 <?xml version="1.0" encoding="ASCII"?>
1090 <!-- MyValueComposite_2 example -->
1091 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1092           targetNamespace="http://foo.com"
1093           name="MyValueComposite" >
1094
1095     <service name="MyValueService" promote="MyValueServiceComponent" />
```

1096

```
1097     <component name="MyValueServiceComponent">
1098         <implementation.java
1099 class="services.myvalue.MyValueServiceImpl"/>
1100         <property name="currency">EURO</property>
1101         <property name="currency">Yen</property>
1102         <property name="currency">USDollar</property>
1103         <reference name="customerService"
1104             target="InternalCustomer/customerService"/>
1105         <reference name="StockQuoteService"/>
1106     </component>
1107
1108     ...
1109
1110     <reference name="CustomerService"
1111         promote="MyValueServiceComponent/customerService"/>
1112
1113     <reference name="StockQuoteService"
1114         promote="MyValueServiceComponent/StockQuoteService"/>
1115
1116 </composite>
```

1117  
1118 ...this assumes that the composite has another component called InternalCustomer (not shown)  
1119 which has a service to which the customerService reference of the MyValueServiceComponent is  
1120 wired as well as being promoted externally through the composite reference CustomerService.

---

## 6 Composite

1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164

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

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

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

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

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Composite schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
    targetNamespace="xs:anyURI"
    name="xs:NCName" local="xs:boolean"?
    autowire="xs:boolean"? constrainingType="QName"?
    requires="list of xs:QName"? policySets="list of xs:QName"?>

    <include ... />*

    <service ... />*
    <reference ... />*
    <property ... />*

    <component ... />*

    <wire ... />*

</composite>
```

1165 The **composite** element has the following **attributes**:

- 1166 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is  
1167 an XML QName, in the namespace identified by the targetNamespace attribute. A  
1168 composite name must be unique within the namespace of the composite. [ASM60001]
- 1169 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the  
1170 composite is declared
- 1171 • **local : boolean (0..1)** – whether all the components within the composite all run in the  
1172 same operating system process. @local="true" for a composite means that all the  
1173 components within the composite MUST run in the same operating system process.  
1174 [ASM60002] local="false", which is the default, means that different components within  
1175 the composite can run in different operating system processes and they can even run on  
1176 different nodes on a network.
- 1177 • **autowire : boolean (0..1)** – whether contained component references should be  
1178 autowired, as described in the [Autowire section](#). Default is false.
- 1179 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
1180 the set of services, references and properties of the composite, plus related intents, is  
1181 constrained to the set defined by the constrainingType. See the [ConstrainingType Section](#)  
1182 for more details.
- 1183 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)  
1184 [specification \[10\]](#) for a description of this attribute.
- 1185 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1186 [\[10\]](#) for a description of this attribute.

1187

1188 The **composite** element has the following **child elements**:

- 1189 • **service : CompositeService (0..n)** – see composite service section.
- 1190 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1191 • **property : CompositeProperty (0..n)** – see composite property section.
- 1192 • **component : Component (0..n)** – see component section.
- 1193 • **wire : Wire (0..n)** – see composite wire section.
- 1194 • **include : Include (0..n)** – see composite include section

1195

1196 Components contain configured implementations which hold the business logic of the composite.  
1197 The components offer services and require references to other services. **Composite services**  
1198 define the public services provided by the composite, which can be accessed from outside the  
1199 composite. **Composite references** represent dependencies which the composite has on services  
1200 provided elsewhere, outside the composite. Wires describe the connections between component  
1201 services and component references within the composite. Included composites contribute the  
1202 elements they contain to the using composite.

1203 Composite services involve the **promotion** of one service of one of the components within the  
1204 composite, which means that the composite service is actually provided by one of the components  
1205 within the composite. Composite references involve the **promotion** of one or more references of  
1206 one or more components. Multiple component references can be promoted to the same composite  
1207 reference, as long as all the component references are compatible with one another. Where  
1208 multiple component references are promoted to the same composite reference, then they all share  
1209 the same configuration, including the same target service(s).

1210 Composite services and composite references can use the configuration of their promoted services  
1211 and references respectively (such as Bindings and Policy Sets). Alternatively composite services  
1212 and composite references can override some or all of the configuration of the promoted services  
1213 and references, through the configuration of bindings and other aspects of the composite service  
1214 or reference.

1215 Component services and component references can be promoted to composite services and  
1216 references and also be wired internally within the composite at the same time. For a reference,  
1217 this only makes sense if the reference supports a multiplicity greater than 1.  
1218

## 1219 6.1 Service

1220 The **services of a composite** are defined by promoting services defined by components  
1221 contained in the composite. A component service is promoted by means of a composite **service**  
1222 **element**.

1223 A composite service is represented by a **service element** which is a child of the composite  
1224 element. There can be **zero or more** service elements in a composite. The following snippet  
1225 shows the composite schema with the schema for a service child element:

```
1226  
1227 <?xml version="1.0" encoding="ASCII"?>  
1228 <!-- Composite Service schema snippet -->  
1229 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1230   ...  
1231   <service name="xs:NCName" promote="xs:anyURI"  
1232     requires="list of xs:QName"? policySets="list of xs:QName"?>*  
1233     <interface ... />?  
1234     <binding ... />*  
1235     <callback>?  
1236       <binding ... />+  
1237     </callback>  
1238   </service>  
1239   ...  
1240 </composite>  
1241
```

1242 The **composite service** element has the following **attributes**:

- 1243 • **name : NCName (1..1)** – the name of the service. The name of a composite <service/>  
1244 element MUST be unique across all the composite services in the composite. [ASM60003]  
1245 The name of the composite service can be different from the name of the promoted  
1246 component service.
- 1247 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form  
1248 <component-name>/<service-name>. The service name is optional if the target  
1249 component only has one service. The same component service can be promoted by more  
1250 than one composite service. A composite <service/> element's promote attribute MUST  
1251 identify one of the component services within that composite. [ASM60004]
- 1252 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1253 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1254 further qualify the required intents defined by the promoted component service.
- 1255 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1256 [\[10\]](#) for a description of this attribute.

1257

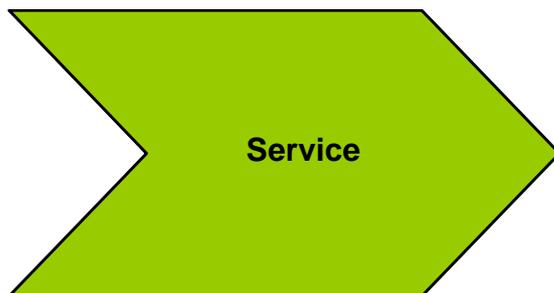
1258 The **composite service** element has the following **child elements**, whatever is not specified is  
1259 defaulted from the promoted component service.

- 1260
- 1261
- 1262
- 1263
- 1264
- 1265
- **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](#).
  - **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the promoted component service from the composite service perspective. The bindings defined on the component service are still in effect for local wires within the composite that target the component service. A service element has zero or more **binding elements** as children. Details of the binding element are described in the [Bindings section](#). For more details on wiring see [the Wiring section](#).
  - **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
- 1272
- 1273
- 1274
- 1275
- 1276
- 1277

## 1278 6.1.1 Service Examples

1279

1280 The following figure shows the service symbol that used to represent a service in an assembly  
1281 diagram:



1282

1283 *Figure 7: Service symbol*

1284

1285 The following figure shows the assembly diagram for the MyValueComposite containing the service  
1286 MyValueService.

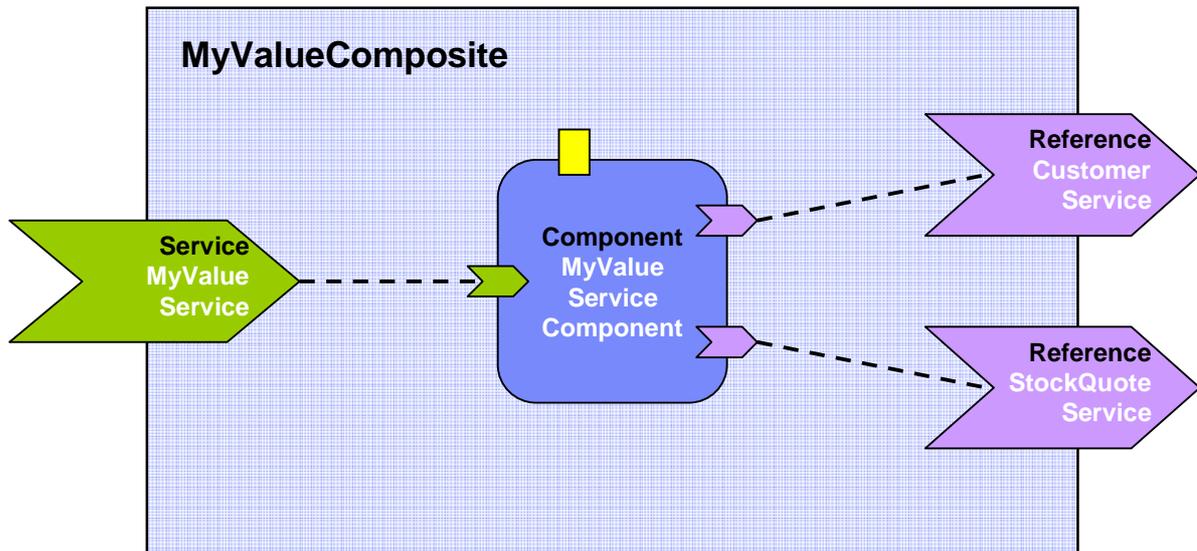


Figure 8: MyValueComposite showing Service

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.

```

1296 <?xml version="1.0" encoding="ASCII"?>
1297 <!-- MyValueComposite_4 example -->
1298 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1299           targetNamespace="http://foo.com"
1300           name="MyValueComposite" >
1301
1302     ...
1303
1304     <service name="MyValueService" promote="MyValueServiceComponent">
1305       <interface.java interface="services.myvalue.MyValueService"/>
1306       <binding.ws port="http://www.myvalue.org/MyValueService#
1307                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1308     </service>
1309
1310     <component name="MyValueServiceComponent">
1311       <implementation.java
1312       class="services.myvalue.MyValueServiceImpl"/>
1313       <property name="currency">EURO</property>
1314       <service name="MyValueService"/>
1315       <reference name="customerService"/>
1316       <reference name="StockQuoteService"/>
1317     </component>

```

1318  
1319       ...  
1320  
1321       </composite>

1322

## 1323 6.2 Reference

1324       The **references of a composite** are defined by **promoting** references defined by components  
1325       contained in the composite. Each promoted reference indicates that the component reference  
1326       needs to be resolved by services outside the composite. A component reference is promoted using  
1327       a composite **reference element**.

1328       A composite reference is represented by a **reference element** which is a child of a composite  
1329       element. There can be **zero or more reference** elements in a composite. The following snippet  
1330       shows the composite schema with the schema for a **reference** element.

1331

```
1332 <?xml version="1.0" encoding="ASCII"?>
1333 <!-- Composite Reference schema snippet -->
1334 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1335   ...
1336   <reference name="xs:NCName" target="list of xs:anyURI"?
1337     promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
1338     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
1339     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1340     <interface ... />?
1341     <binding ... />*
1342     <callback>?
1343       <binding ... />+
1344     </callback>
1345   </reference>
1346   ...
1347 </composite>
```

1348

1349

1350       The **composite reference** element has the following **attributes**:

- 1351       • **name : NCName (1..1)** – the name of the reference. The name of a composite  
1352       <reference/> element MUST be unique across all the composite references in the  
1353       composite. [ASM60006] The name of the composite reference can be different than the  
1354       name of the promoted component reference.
- 1355       • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
1356       value is a list of values of the form <component-name>/<reference-name> separated by  
1357       spaces. The specification of the reference name is optional if the component has only one  
1358       reference. Each of the URIs declared by a composite reference's @promote attribute MUST  
1359       identify a component reference within the composite. [ASM60007]

1360       The same component reference can be promoted more than once, using different  
1361       composite references, but only if the multiplicity defined on the component reference is  
1362       0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1363       Where a composite reference promotes two or more component references:

- 1364 • the interfaces of the component references promoted by a composite reference  
1365 MUST be the same, or if the composite reference itself declares an interface then  
1366 all the component reference interfaces must be compatible with the composite  
1367 reference interface. Compatible means that the component reference interface is  
1368 the same or is a strict subset of the composite reference interface. [ASM60008]
  - 1369 • the intents declared on a composite reference and on the component references  
1370 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents  
1371 which apply to the composite reference in this case are the union of the required  
1372 intents specified for each of the promoted component references plus any intents  
1373 declared on the composite reference itself. If any intents in the set which apply to  
1374 a composite reference are mutually exclusive then the SCA runtime MUST raise an  
1375 error. [ASM60010]
  - 1376 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1377 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1378 further qualify the required intents defined for the promoted component reference.
  - 1379 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1380 [\[10\]](#) for a description of this attribute.
  - 1381 • **multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can  
1382 connect the reference to target services. The multiplicity can have the following values
    - 1383 ○ 0..1 – zero or one wire can have the reference as a source
    - 1384 ○ 1..1 – one wire can have the reference as a source
    - 1385 ○ 0..n - zero or more wires can have the reference as a source
    - 1386 ○ 1..n – one or more wires can have the reference as a source
- 1387 The value specified for the **multiplicity** attribute of a composite reference MUST be  
1388 compatible with the multiplicity specified on each of the promoted component references,  
1389 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used  
1390 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be  
1391 used where the promoted component reference has multiplicity 0..n or 1..n and  
1392 multiplicity 1..n can be used where the promoted component reference has multiplicity  
1393 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to  
1394 promote a component reference of multiplicity 0..1 or 1..1 respectively. [ASM60011]
- 1395 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
1396 multiplicity setting. Each value wires the reference to a service in a composite that uses  
1397 the composite containing the reference as an implementation for one of its components. For  
1398 more details on wiring see [the section on Wires](#).
  - 1399 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
1400 the implementation wires this reference dynamically. If set to "true" it indicates that the  
1401 target of the reference is set at runtime by the implementation code (eg by the code  
1402 obtaining an endpoint reference by some means and setting this as the target of the  
1403 reference through the use of programming interfaces defined by the relevant Client and  
1404 Implementation specification). If "true" is set, then the reference should not be wired  
1405 statically within a using composite, but left unwired.

1406

1407 The **composite reference** element has the following **child elements**, whatever is not specified is  
1408 defaulted from the promoted component reference(s).

- 1409 • **interface : Interface (0..1) - zero or one interface element** which declares an  
1410 interface for the composite reference. If a composite reference has an **interface** specified,  
1411 it MUST provide an interface which is the same or which is a compatible superset of the  
1412 interface(s) declared by the promoted component reference(s), i.e. provide a superset of  
1413 the operations in the interface defined by the component for the reference. [ASM60012] If  
1414 no interface is declared on a composite reference, the interface from one of its promoted  
1415 component references is used, which MUST be the same as or a compatible superset of

1416 the interface(s) declared by the promoted component reference(s).  
1417 [ASM60013] For details on the interface element see the [Interface](#) section.

1418 • **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as  
1419 children. If one or more **bindings** are specified they **override** any and all of the bindings  
1420 defined for the promoted component reference from the composite reference perspective.  
1421 The bindings defined on the component reference are still in effect for local wires within  
1422 the composite that have the component reference as their source. Details of the binding  
1423 element are described in the [Bindings](#) section. For more details on wiring see the [section](#)  
1424 [on Wires](#).

1425 A reference identifies zero or more target services which satisfy the reference. This can be  
1426 done in a number of ways, which are fully described in section "5.3.1 [Specifying the](#)  
1427 [Target Service\(s\) for a Reference](#)".

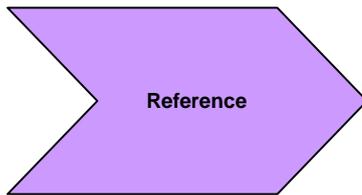
1428 • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional  
1429 **callback** element used if the interface has a callback defined, which has one or more  
1430 **binding** elements as children. The **callback** and its binding child elements are specified if  
1431 there is a need to have binding details used to handle callbacks. If the callback element is  
1432 not present, the behaviour is runtime implementation dependent.

1433

## 1434 6.2.1 Example Reference

1435

1436 The following figure shows the reference symbol that is used to represent a reference in an  
1437 assembly diagram.



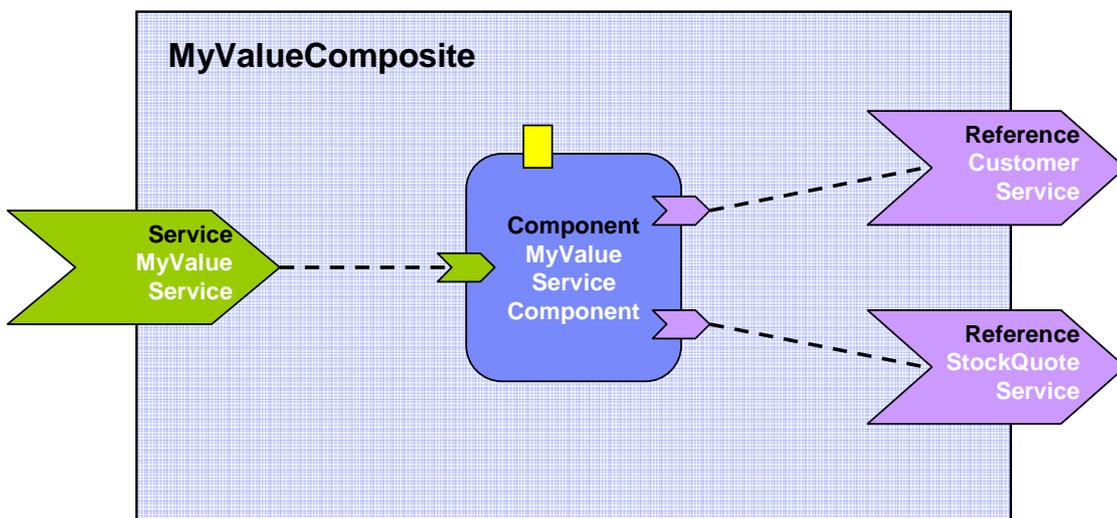
1438

1439 *Figure 9: Reference symbol*

1440

1441 The following figure shows the assembly diagram for the MyValueComposite containing the  
1442 reference CustomerService and the reference StockQuoteService.

1443



1444

1445 *Figure 10: MyValueComposite showing References*

1446

1447 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1448 containing the reference elements for the CustomerService and the StockQuoteService. The  
1449 reference CustomerService is bound using the SCA binding. The reference StockQuoteService is  
1450 bound using the Web service binding. The endpoint addresses of the bindings can be specified, for  
1451 example using the binding *uri* attribute (for details see the [Bindings](#) section), or overridden in an  
1452 enclosing composite. Although in this case the reference StockQuoteService is bound to a Web  
1453 service, its interface is defined by a Java interface, which was created from the WSDL portType of  
1454 the target web service.

1455

```
1456 <?xml version="1.0" encoding="ASCII"?>
```

```
1457 <!-- MyValueComposite_3 example -->
```

```
1458 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
```

```
1459 targetNamespace="http://foo.com"
```

```
1460 name="MyValueComposite" >
```

1461

```
1462 ...
```

1463

```
1464 <component name="MyValueServiceComponent">
```

```
1465 <implementation.java
```

```
1466 class="services.myvalue.MyValueServiceImpl"/>
```

```
1467 <property name="currency">EURO</property>
```

```
1468 <reference name="customerService"/>
```

```
1469 <reference name="StockQuoteService"/>
```

```
1470 </component>
```

1471

```
1472 <reference name="CustomerService"
```

```
1473 promote="MyValueServiceComponent/customerService">
```

```
1474 <interface.java interface="services.customer.CustomerService"/>
```

```
1475 <!-- The following forces the binding to be binding.sca whatever
```

```
1476 is -->
```

```
1477 <!-- specified by the component reference or by the underlying
```

```
1478 -->
```

```
1479 <!-- implementation
```

```
1480 -->
```

```
1481 <binding.sca/>
```

```
1482 </reference>
```

1483

```
1484 <reference name="StockQuoteService"
```

```
1485 promote="MyValueServiceComponent/StockQuoteService">
```

```
1486 <interface.java
```

```
1487 interface="services.stockquote.StockQuoteService"/>
```

```
1488 <binding.ws port="http://www.stockquote.org/StockQuoteService#
```

1489

```
1490 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
```

```
1491     </reference>
1492
1493     ...
1494
1495 </composite>
1496
```

## 1497 6.3 Property

1498 **Properties** allow for the configuration of an implementation with externally set data values. A  
1499 composite can declare zero or more properties. Each property has a type, which may be either  
1500 simple or complex. An implementation can also define a default value for a property. Properties  
1501 can be configured with values in the components that use the implementation.

1502 The declaration of a property in a composite follows the form described in the following schema  
1503 snippet:

```
1504
1505 <?xml version="1.0" encoding="ASCII"?>
1506 <!-- Composite Property schema snippet -->
1507 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1508     ...
1509     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1510             many="xs:boolean"? mustSupply="xs:boolean"?>*
1511             default-property-value?
1512     </property>
1513     ...
1514 </composite>
1515
```

1516 The **composite property** element has the following **attributes**:

- 1517 ▪ **name : NCName (1..1)** - the name of the property. The name attribute of a composite  
1518 property MUST be unique amongst the properties of the same composite. [ASM60014]
- 1519 ▪ one of **(1..1)**:
  - 1520 ○ **type : QName** – the type of the property - the qualified name of an XML schema  
1521 type
  - 1522 ○ **element : QName** – the type of the property defined as the qualified name of an  
1523 XML schema global element – the type is the type of the global element
- 1524 ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued  
1525 (true). The default is **false**. In the case of a multi-valued property, it is presented to the  
1526 implementation as a collection of property values.
- 1527 ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the  
1528 component that uses the composite – when mustSupply="true" the component has to  
1529 supply a value since the composite has no default value for the property. A default-  
1530 property-value is only worth declaring when mustSupply="false" (the default setting for  
1531 the mustSupply attribute), since the implication of a default value is that it is used only  
1532 when a value is not supplied by the using component.

1533

1534 The property element may contain an optional **default-property-value**, which provides default  
1535 value for the property. The default value must match the type declared for the property:

- 1536 ○ a string, if **type** is a simple type (matching the **type** declared)
- 1537 ○ a complex type value matching the type declared by **type**
- 1538 ○ an element matching the element named by **element**
- 1539 ○ multiple values are permitted if many="true" is specified
- 1540

1541 Implementation types other than **composite** can declare properties in an implementation-  
 1542 dependent form (eg annotations within a Java class), or through a property declaration of exactly  
 1543 the form described above in a componentType file.

1544 Property values can be configured when an implementation is used by a component. The form of  
 1545 the property configuration is shown in [the section on Components](#).

### 1546 6.3.1 Property Examples

1547  
 1548 For the following example of Property declaration and value setting, the following complex type is  
 1549 used as an example:

```

1550 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1551             targetNamespace="http://foo.com/"
1552             xmlns:tns="http://foo.com/">
1553   <!-- ComplexProperty schema -->
1554   <xsd:element name="fooElement" type="MyComplexType"/>
1555   <xsd:complexType name="MyComplexType">
1556     <xsd:sequence>
1557       <xsd:element name="a" type="xsd:string"/>
1558       <xsd:element name="b" type="anyURI"/>
1559     </xsd:sequence>
1560     <attribute name="attr" type="xsd:string" use="optional"/>
1561   </xsd:complexType>
1562 </xsd:schema>
1563
  
```

1564 The following composite demonstrates the declaration of a property of a complex type, with a  
 1565 default value, plus it demonstrates the setting of a property value of a complex type within a  
 1566 component:

```

1567 <?xml version="1.0" encoding="ASCII"?>
1568
1569 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1570           xmlns:foo="http://foo.com"
1571           targetNamespace="http://foo.com"
1572           name="AccountServices">
1573 <!-- AccountServices Example1 -->
1574
1575   ...
1576
1577   <property name="complexFoo" type="foo:MyComplexType">
1578     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1579       <foo:a>AValue</foo:a>
  
```

```

1580         <foo:b>InterestingURI</foo:b>
1581     </MyComplexPropertyValue>
1582 </property>
1583
1584     <component name="AccountServiceComponent">
1585         <implementation.java class="foo.AccountServiceImpl"/>
1586         <property name="complexBar" source="$complexFoo"/>
1587         <reference name="accountDataService"
1588             target="AccountDataServiceComponent"/>
1589         <reference name="stockQuoteService" target="StockQuoteService"/>
1590     </component>
1591
1592     ...
1593
1594 </composite>

```

1595 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the  
 1596 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the  
 1597 composite and it references the example XSD, where MyComplexType is defined. The declaration  
 1598 of complexFoo contains a default value. This is declared as the content of the property element.  
 1599 In this example, the default value consists of the element **MyComplexPropertyValue** of type  
 1600 foo:MyComplexType and its two child elements <foo:a> and <foo:b>, following the definition of  
 1601 MyComplexType.

1602 In the component **AccountServiceComponent**, the component sets the value of the property  
 1603 **complexBar**, declared by the implementation configured by the component. In this case, the  
 1604 type of complexBar is foo:MyComplexType. The example shows that the value of the complexBar  
 1605 property is set from the value of the complexFoo property – the **source** attribute of the property  
 1606 element for complexBar declares that the value of the property is set from the value of a property  
 1607 of the containing composite. The value of the source attribute is **\$complexFoo**, where  
 1608 complexFoo is the name of a property of the composite. This value implies that the whole of the  
 1609 value of the source property is used to set the value of the component property.

1610 The following example illustrates the setting of the value of a property of a simple type (a string)  
 1611 from **part** of the value of a property of the containing composite which has a complex type:

```

1612 <?xml version="1.0" encoding="ASCII"?>
1613
1614 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1615           xmlns:foo="http://foo.com"
1616           targetNamespace="http://foo.com"
1617           name="AccountServices">
1618 <!-- AccountServices Example2 -->
1619
1620     ...
1621
1622     <property name="complexFoo" type="foo:MyComplexType">
1623         <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1624             <foo:a>AValue</foo:a>
1625             <foo:b>InterestingURI</foo:b>
1626         </MyComplexPropertyValue>

```

```

1627     </property>
1628
1629     <component name="AccountServiceComponent">
1630         <implementation.java class="foo.AccountServiceImpl"/>
1631         <property name="currency" source="$complexFoo/a"/>
1632         <reference name="accountDataService"
1633             target="AccountDataServiceComponent"/>
1634         <reference name="stockQuoteService" target="StockQuoteService"/>
1635     </component>
1636
1637     ...
1638
1639 </composite>

```

1640 In this example, the component **AccountServiceComponent** sets the value of a property called  
1641 **currency**, which is of type string. The value is set from a property of the composite  
1642 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression  
1643 that selects the property name **complexFoo** and then selects the value of the **a** subelement of  
1644 complexFoo. The "a" subelement is a string, matching the type of the currency property.

1645 Further examples of declaring properties and setting property values in a component follow:

1646 Declaration of a property with a simple type and a default value:

```

1647 <property name="SimpleTypeProperty" type="xsd:string">
1648 MyValue
1649 </property>

```

1650

1651 Declaration of a property with a complex type and a default value:

```

1652 <property name="complexFoo" type="foo:MyComplexType">
1653     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1654         <foo:a>AValue</foo:a>
1655         <foo:b>InterestingURI</foo:b>
1656     </MyComplexPropertyValue>
1657 </property>

```

1658

1659 Declaration of a property with an element type:

```

1660 <property name="elementFoo" element="foo:fooElement">
1661     <foo:fooElement>
1662         <foo:a>AValue</foo:a>
1663         <foo:b>InterestingURI</foo:b>
1664     </foo:fooElement>
1665 </property>

```

1666

1667 Property value for a simple type:

```

1668 <property name="SimpleTypeProperty">
1669 MyValue
1670 </property>

```

1671  
1672  
1673 Property value for a complex type, also showing the setting of an attribute value of the complex  
1674 type:

```
1675 <property name="complexFoo">  
1676   <MyComplexPropertyValue xsi:type="foo:MyComplexType" attr="bar">  
1677     <foo:a>AValue</foo:a>  
1678     <foo:b>InterestingURI</foo:b>  
1679   </MyComplexPropertyValue>  
1680 </property>
```

1681  
1682 Property value for an element type:

```
1683 <property name="elementFoo">  
1684   <foo:fooElement attr="bar">  
1685     <foo:a>AValue</foo:a>  
1686     <foo:b>InterestingURI</foo:b>  
1687   </foo:fooElement>  
1688 </property>
```

1689  
1690 Declaration of a property with a complex type where multiple values are supported:

```
1691 <property name="complexFoo" type="foo:MyComplexType" many="true"/>  
1692
```

1693 Setting of a value for that property where multiple values are supplied:

```
1694 <property name="complexFoo">  
1695   <MyComplexPropertyValue1 xsi:type="foo:MyComplexType" attr="bar">  
1696     <foo:a>AValue</foo:a>  
1697     <foo:b>InterestingURI</foo:b>  
1698   </MyComplexPropertyValue1>  
1699   <MyComplexPropertyValue2 xsi:type="foo:MyComplexType" attr="zing">  
1700     <foo:a>BValue</foo:a>  
1701     <foo:b>BoringURI</foo:b>  
1702   </MyComplexPropertyValue2>  
1703 </property>  
1704
```

## 1705 6.4 Wire

1706 **SCA wires** within a composite connect **source component references** to **target component**  
1707 **services**.

1708 One way of defining a wire is by **configuring a reference of a component using its target**  
1709 **attribute**. The reference element is configured with the wire-target-URI of the service(s) that  
1710 resolve the reference. Multiple target services are valid when the reference has a multiplicity of  
1711 0..n or 1..n.

1712 An alternative way of defining a Wire is by means of a **wire element** which is a child of the  
1713 composite element. There can be **zero or more** wire elements in a composite. This alternative  
1714 method for defining wires is useful in circumstances where separation of the wiring from the

1715 elements the wires connect helps simplify development or operational activities. An example is  
1716 where the components used to build a domain are relatively static but where new or changed  
1717 applications are created regularly from those components, through the creation of new assemblies  
1718 with different wiring. Deploying the wiring separately from the components allows the wiring to  
1719 be created or modified with minimum effort.

1720 Note that a Wire specified via a wire element is equivalent to a wire specified via the target  
1721 attribute of a reference. The rule which forbids mixing of wires specified with the target attribute  
1722 with the specification of endpoints in binding subelements of the reference also applies to wires  
1723 specified via separate wire elements.

1724 The following snippet shows the composite schema with the schema for the reference elements of  
1725 components and composite services and the wire child element:

1726

```
1727 <?xml version="1.0" encoding="ASCII"?>
1728 <!-- Wires schema snippet -->
1729 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1730           targetNamespace="xs:anyURI"
1731           name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
1732           constrainingType="QName"?
1733           requires="list of xs:QName"? policySets="list of
1734 xs:QName"?>
1735
1736     ...
1737
1738     <wire source="xs:anyURI" target="xs:anyURI" /*>
1739
1740 </composite>
```

1741

1742

1743 The **reference element of a component** and the **reference element of a service** has a list of  
1744 one or more of the following **wire-target-URI** values for the target, with multiple values  
1745 separated by a space:

- 1746 • `<component-name>/<service-name>`
  - 1747 ○ where the target is a service of a component. The specification of the service  
1748 name is optional if the target component only has one service with a compatible  
1749 interface

1750

1751 The **wire element** has the following attributes:

- 1752 • **source (1..1)** – names the source component reference. Valid URI schemes are:
  - 1753 ○ `<component-name>/<reference-name>`
    - 1754 ▪ where the source is a component reference. The specification of the  
1755 reference name is optional if the source component only has one reference
- 1756 • **target (1..1)** – names the target component service. Valid URI schemes are
  - 1757 ○ `<component-name>/<service-name>`
    - 1758 ▪ where the target is a service of a component. The specification of the  
1759 service name is optional if the target component only has one service with  
1760 a compatible interface

1761 For a composite used as a component implementation, wires can only link sources and targets  
1762 that are contained in the same composite (irrespective of which file or files are used to describe  
1763 the composite). Wiring to entities outside the composite is done through services and references  
1764 of the composite with wiring defined by the next higher composite.

1765 A wire may only connect a source to a target if the target implements an interface that is  
1766 compatible with the interface required by the source. The source and the target are compatible if:

- 1767 1. the source interface and the target interface of a wire MUST either both be remotable or  
1768 else both be local [ASM60015]
- 1769 2. the operations on the target interface of a wire MUST be the same as or be a superset of  
1770 the operations in the interface specified on the source [ASM60016]
- 1771 3. compatibility between the source interface and the target interface for a wire for the  
1772 individual operations is defined as compatibility of the signature, that is operation name,  
1773 input types, and output types MUST be the same. [ASM60017]
- 1774 4. the order of the input and output types for operations in the source interface and the  
1775 target interface of a wire also MUST be the same. [ASM60018]
- 1776 5. the set of Faults and Exceptions expected by each operation in the source interface MUST  
1777 be the same or be a superset of those specified by the target interface. [ASM60019]
- 1778 6. other specified attributes of the source interface and the target interface of a wire MUST  
1779 match, including Scope and Callback interface [ASM60020]

1780 A Wire can connect between different interface languages (eg. Java interfaces and WSDL  
1781 portTypes) in either direction, as long as the operations defined by the two interface types are  
1782 equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and  
1783 faults/exceptions map to each other.

1784 Service clients cannot (portably) ask questions at runtime about additional interfaces that are  
1785 provided by the implementation of the service (e.g. the result of "instance of" in Java is non  
1786 portable). It is valid for an SCA implementation to have proxies for all wires, so that, for example,  
1787 a reference object passed to an implementation may only have the business interface of the  
1788 reference and may not be an instance of the (Java) class which is used to implement the target  
1789 service, even where the interface is local and the target service is running in the same process.

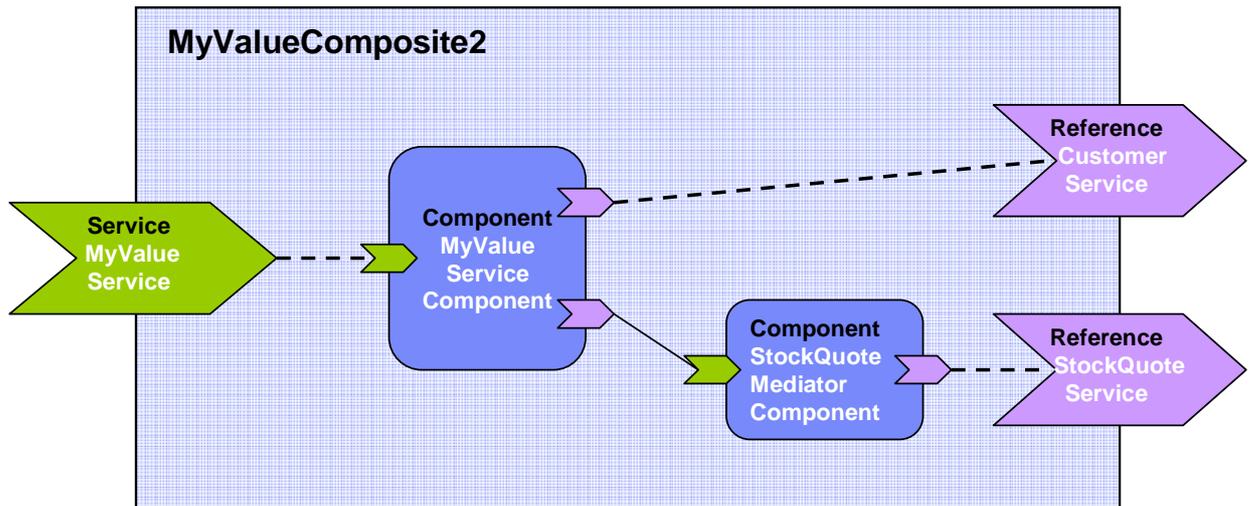
1790 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of  
1791 an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA  
1792 runtime SHOULD issue a warning. [ASM60021]

1793

## 1794 6.4.1 Wire Examples

1795

1796 The following figure shows the assembly diagram for the MyValueComposite2 containing wires  
1797 between service, components and references.



1798

1799 *Figure 11: MyValueComposite2 showing Wires*

1800

1801 The following snippet shows the `MyValueComposite2.composite` file for the `MyValueComposite2`  
 1802 containing the configured component and service references. The service `MyValueService` is wired  
 1803 to the `MyValueServiceComponent`, using an explicit `<wire/>` element. The  
 1804 `MyValueServiceComponent`'s `customerService` reference is wired to the composite's  
 1805 `CustomerService` reference. The `MyValueServiceComponent`'s `stockQuoteService` reference is  
 1806 wired to the `StockQuoteMediatorComponent`, which in turn has its reference wired to the  
 1807 `StockQuoteService` reference of the composite.

1808

```

1809 <?xml version="1.0" encoding="ASCII"?>
1810 <!-- MyValueComposite Wires examples -->
1811 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1812           targetNamespace="http://foo.com"
1813           name="MyValueComposite2" >
1814
1815     <service name="MyValueService" promote="MyValueServiceComponent">
1816       <interface.java interface="services.myvalue.MyValueService"/>
1817       <binding.ws port="http://www.myvalue.org/MyValueService#
1818                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1819     </service>
1820
1821     <component name="MyValueServiceComponent">
1822       <implementation.java
1823         class="services.myvalue.MyValueServiceImpl"/>
1824       <property name="currency">EURO</property>
1825       <service name="MyValueService"/>
1826       <reference name="customerService"/>
1827       <reference name="stockQuoteService"/>
1828     </component>
  
```

1829

```

1830     <wire source="MyValueServiceComponent/stockQuoteService"
1831         target="StockQuoteMediatorComponent" />
1832
1833     <component name="StockQuoteMediatorComponent">
1834         <implementation.java class="services.myvalue.SQMediatorImpl" />
1835         <property name="currency">EURO</property>
1836         <reference name="stockQuoteService" />
1837     </component>
1838
1839     <reference name="CustomerService"
1840         promote="MyValueServiceComponent/customerService">
1841         <interface.java interface="services.customer.CustomerService" />
1842         <binding.sca />
1843     </reference>
1844
1845     <reference name="StockQuoteService"
1846         promote="StockQuoteMediatorComponent">
1847         <interface.java
1848             interface="services.stockquote.StockQuoteService" />
1849         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1850             wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)" />
1851     </reference>
1852
1853 </composite>
1854

```

## 1855 6.4.2 Autowire

1856 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.  
1857 Autowire enables component references to be automatically wired to component services which  
1858 will satisfy those references, without the need to create explicit wires between the references and  
1859 the services. When the autowire feature is used, a component reference which is not promoted  
1860 and which is not explicitly wired to a service within a composite is automatically wired to a target  
1861 service within the same composite. Autowire works by searching within the composite for a  
1862 service interface which matches the interface of the references.

1863 The autowire feature is not used by default. Autowire is enabled by the setting of an autowire  
1864 attribute to "true". Autowire is disabled by setting of the autowire attribute to "false" The autowire  
1865 attribute can be applied to any of the following elements within a composite:

- 1866 • reference
- 1867 • component
- 1868 • composite

1869 Where an element does not have an explicit setting for the autowire attribute, it inherits the  
1870 setting from its parent element. Thus a reference element inherits the setting from its containing  
1871 component. A component element inherits the setting from its containing composite. Where  
1872 there is no setting on any level, autowire="false" is the default.

1873 As an example, if a composite element has autowire="true" set, this means that autowiring is  
1874 enabled for all component references within that composite. In this example, autowiring can be

1875 turned off for specific components and specific references through setting autowire="false" on the  
1876 components and references concerned.

1877 For each component reference for which autowire is enabled, the the SCA runtime MUST search  
1878 within the composite for target services which are compatible with the reference. [ASM60022]  
1879 "Compatible" here means:

- 1880 • the target service interface MUST be a compatible superset of the reference interface  
1881 when using autowire to wire a reference (as defined in the section on Wires) [ASM60023]
- 1882 • the intents, and policies applied to the service MUST be compatible with those on the  
1883 reference when using autowire to wire a reference – so that wiring the reference to the  
1884 service will not cause an error due to policy mismatch [ASM60024] (see the Policy  
1885 Framework specification [10] for details)

1886 If the search finds **1 or more** valid target service for a particular reference, the action taken  
1887 depends on the multiplicity of the reference:

- 1888 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
1889 reference to one of the set of valid target services chosen from the set in a runtime-  
1890 dependent fashion [ASM60025]
- 1891 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
1892 of the set of valid target services [ASM60026]

1893 If the search finds **no** valid target services for a particular reference, the action taken depends on  
1894 the multiplicity of the reference:

- 1895 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
1896 target service, there is no problem – no services are wired and the SCA runtime MUST  
1897 NOT raise an error [ASM60027]
- 1898 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
1899 target services an error MUST be raised by the SCA runtime since the reference is  
1900 intended to be wired [ASM60028]

1901

## 1902 6.4.3 Autowire Examples

1903 This example demonstrates two versions of the same composite – the first version is done using  
1904 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
1905 the end result is the same – the same wires connect the references to the services.

1906 First, here is a diagram for the composite:

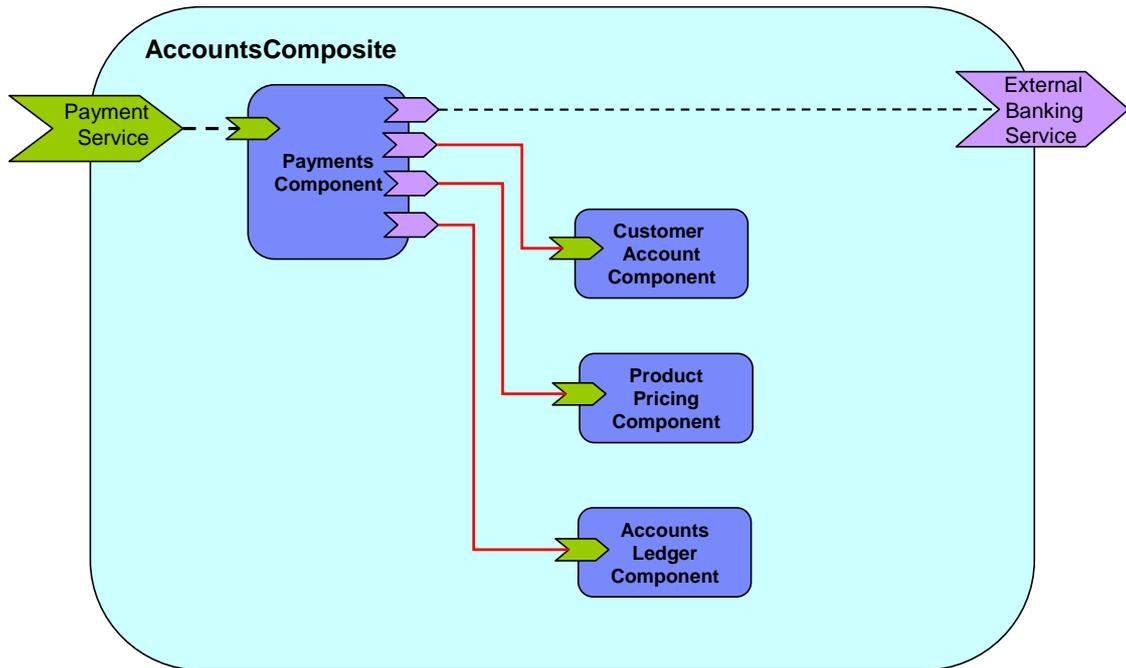


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1910 <?xml version="1.0" encoding="UTF-8"?>
1911 <!-- Autowire Example - No autowire -->
1912 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1913     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1914     xmlns:foo="http://foo.com"
1915     targetNamespace="http://foo.com"
1916     name="AccountComposite">
1917
1918     <service name="PaymentService" promote="PaymentsComponent"/>
1919
1920     <component name="PaymentsComponent">
1921         <implementation.java class="com.foo.accounts.Payments"/>
1922         <service name="PaymentService"/>
1923         <reference name="CustomerAccountService"
1924             target="CustomerAccountComponent"/>
1925         <reference name="ProductPricingService"
1926             target="ProductPricingComponent"/>
1927         <reference name="AccountsLedgerService"
1928             target="AccountsLedgerComponent"/>
1929         <reference name="ExternalBankingService"/>
1930     </component>
1931
1932     <component name="CustomerAccountComponent">

```

```

1933         <implementation.java class="com.foo.accounts.CustomerAccount" />
1934     </component>
1935
1936     <component name="ProductPricingComponent">
1937         <implementation.java class="com.foo.accounts.ProductPricing" />
1938     </component>
1939
1940     <component name="AccountsLedgerComponent">
1941         <implementation.composite name="foo:AccountsLedgerComposite" />
1942     </component>
1943
1944     <reference name="ExternalBankingService"
1945         promote="PaymentsComponent/ExternalBankingService" />
1946
1947 </composite>
1948
1949 Secondly, the composite using autowire:
1950 <?xml version="1.0" encoding="UTF-8"?>
1951 <!-- Autowire Example - With autowire -->
1952 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1953     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1954     xmlns:foo="http://foo.com"
1955     targetNamespace="http://foo.com"
1956     name="AccountComposite">
1957
1958     <service name="PaymentService" promote="PaymentsComponent">
1959         <interface.java class="com.foo.PaymentServiceInterface" />
1960     </service>
1961
1962     <component name="PaymentsComponent" autowire="true">
1963         <implementation.java class="com.foo.accounts.Payments" />
1964         <service name="PaymentService" />
1965         <reference name="CustomerAccountService" />
1966         <reference name="ProductPricingService" />
1967         <reference name="AccountsLedgerService" />
1968         <reference name="ExternalBankingService" />
1969     </component>
1970
1971     <component name="CustomerAccountComponent">
1972         <implementation.java class="com.foo.accounts.CustomerAccount" />
1973     </component>
1974
1975     <component name="ProductPricingComponent">

```

```

1976         <implementation.java class="com.foo.accounts.ProductPricing"/>
1977     </component>
1978
1979     <component name="AccountsLedgerComponent">
1980         <implementation.composite name="foo:AccountsLedgerComposite"/>
1981     </component>
1982
1983     <reference name="ExternalBankingService"
1984         promote="PaymentsComponent/ExternalBankingService"/>
1985
1986 </composite>

```

1987 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires  
1988 for any of its references – the wires are created automatically through autowire.

1989 **Note:** In the second example, it would be possible to omit all of the service and reference  
1990 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the  
1991 component service and references still exist, since they are provided by the implementation used  
1992 by the component.

1993

## 1994 6.5 Using Composites as Component Implementations

1995 Composites may form **component implementations** in higher-level composites – in other words  
1996 the higher-level composites can have components which are implemented by composites.

1997 When a composite is used as a component implementation, it defines a boundary of visibility.  
1998 Components within the composite cannot be referenced directly by the using component. The  
1999 using component can only connect wires to the services and references of the used composite and  
2000 set values for any properties of the composite. The internal construction of the composite is  
2001 invisible to the using component.

2002 A composite used as a component implementation needs to also honor a **completeness**  
2003 **contract**. The services, references and properties of the composite form a contract which is relied  
2004 upon by the using component. The concept of completeness of the composite implies:

- 2005 • the composite must have at least one service or at least one reference.  
2006 A component with no services and no references is not meaningful in terms of SCA, since  
2007 it cannot be wired to anything – it neither provides nor consumes any services  
2008
- 2009 • each service offered by the composite must be wired to a service of a component or to a  
2010 composite reference.  
2011 If services are left unwired, the implication is that some exception will occur at runtime if  
2012 the service is invoked.

2013 The component type of a composite is defined by the set of service elements, reference elements  
2014 and property elements that are the children of the composite element.

2015 Composites are used as component implementations through the use of the  
2016 **implementation.composite** element as a child element of the component. The schema snippet  
2017 for the implementation.composite element is:

```

2018
2019 <?xml version="1.0" encoding="ASCII"?>
2020 <!-- Composite Implementation schema snippet -->
2021 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2022         targetNamespace="xs:anyURI"

```

```

2023         name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2024         constrainingType="QName"?
2025         requires="list of xs:QName"? policySets="list of
2026 xs:QName"?>
2027
2028     ...
2029
2030     <component name="xs:NCName" autowire="xs:boolean"?
2031         requires="list of xs:QName"? policySets="list of xs:QName"?>*
2032         <implementation.composite name="xs:QName"/>?
2033         <service name="xs:NCName" requires="list of xs:QName"?
2034             policySets="list of xs:QName"?>*
2035             <interface ... />?
2036             <binding uri="xs:anyURI" name="xs:QName"?
2037                 requires="list of xs:QName"
2038                 policySets="list of xs:QName"?/>*
2039             <callback>?
2040                 <binding uri="xs:anyURI"? name="xs:QName"?
2041                     requires="list of xs:QName"?
2042                     policySets="list of xs:QName"?/>+
2043             </callback>
2044         </service>
2045         <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
2046             source="xs:string"? file="xs:anyURI"?>*
2047             property-value
2048         </property>
2049         <reference name="xs:NCName" target="list of xs:anyURI"?
2050             autowire="xs:boolean"? wiredByImpl="xs:boolean"?
2051             requires="list of xs:QName"? policySets="list of xs:QName"?
2052             multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
2053             <interface ... />?
2054             <binding uri="xs:anyURI"? name="xs:QName"?
2055                 requires="list of xs:QName" policySets="list of
2056 xs:QName"?/>*
2057             <callback>?
2058                 <binding uri="xs:anyURI"? name="xs:QName"?
2059                     requires="list of xs:QName"?
2060                     policySets="list of xs:QName"?/>+
2061             </callback>
2062         </reference>
2063     </component>
2064
2065     ...

```

2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074

```
</composite>
```

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]

## 2075 6.5.1 Example of Composite used as a Component Implementation

2076

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

2077

2078

2079

2080

2081

2082

2083

2084

2085

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

2099

2100

2101

2102

2103

2104

2105

2106

2107

2108

2109

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- CompositeComponent example -->
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200712
file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  targetNamespace="http://foo.com"
  xmlns:foo="http://foo.com"
  name="AccountComposite">
  <service name="AccountService" promote="AccountServiceComponent">
    <interface.java interface="services.account.AccountService"/>
    <binding.ws port="AccountService#"
      wsdl.endpoint(AccountService/AccountServiceSOAP) />
  </service>
  <reference name="stockQuoteService"
    promote="AccountServiceComponent/StockQuoteService">
    <interface.java
interface="services.stockquote.StockQuoteService"/>
    <binding.ws
port="http://www.quickstockquote.com/StockQuoteService#"
      wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP) />
  </reference>
  <property name="currency" type="xsd:string">EURO</property>
  <component name="AccountServiceComponent">
    <implementation.composite name="foo:AccountServiceComposite1"/>
  </component>
</composite>
```

```

2110     <reference name="AccountDataService" target="AccountDataService"/>
2111     <reference name="StockQuoteService"/>
2112
2113     <property name="currency" source="$currency"/>
2114 </component>
2115
2116 <component name="AccountDataService">
2117     <implementation.composite name="foo:AccountDataServiceComposite"/>
2118
2119     <property name="currency" source="$currency"/>
2120 </component>
2121
2122 </composite>
2123

```

## 2124 6.6 Using Composites through Inclusion

2125 In order to assist team development, composites may be developed in the form of multiple  
 2126 physical artifacts that are merged into a single logical unit.

2127 A composite is defined in an **xxx.composite** file and the composite may receive additional  
 2128 content through the **inclusion of other composite** files.

2129 The semantics of included composites are that the content of the included composite is inlined into  
 2130 the using composite **xxx.composite** file through **include** elements in the using composite. The  
 2131 effect is one of **textual inclusion** – that is, the text content of the included composite is placed  
 2132 into the using composite in place of the include statement. The included composite element itself  
 2133 is discarded in this process – only its contents are included.

2134 The composite file used for inclusion can have any contents, but always contains a single  
 2135 **composite** element. The composite element can contain any of the elements which are valid as  
 2136 child elements of a composite element, namely components, services, references, wires and  
 2137 includes. There is no need for the content of an included composite to be complete, so that  
 2138 artifacts defined within the using composite or in another associated included composite file may  
 2139 be referenced. For example, it is permissible to have two components in one composite file while a  
 2140 wire specifying one component as the source and the other as the target can be defined in a  
 2141 second included composite file.

2142 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one  
 2143 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated  
 2144 elements in the using composite (eg. two services with the same uri contributed by different  
 2145 included composites), or if there are wires with non-existent source or target.

2146 The following snippet shows the partial schema for the include element.

```

2147
2148 <?xml version="1.0" encoding="UTF-8"?>
2149 <!-- Include snippet -->
2150 <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2151                targetNamespace="xs:anyURI"
2152                name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2153                constrainingType="QName"?
2154                requires="list of xs:QName"? policySets="list of
2155 xs:QName"?>

```

```

2156
2157     ...
2158
2159     <include name="xs:QName" />*
2160
2161     ...
2162
2163 </composite>
2164

```

2165 The include element has the following **attribute**:

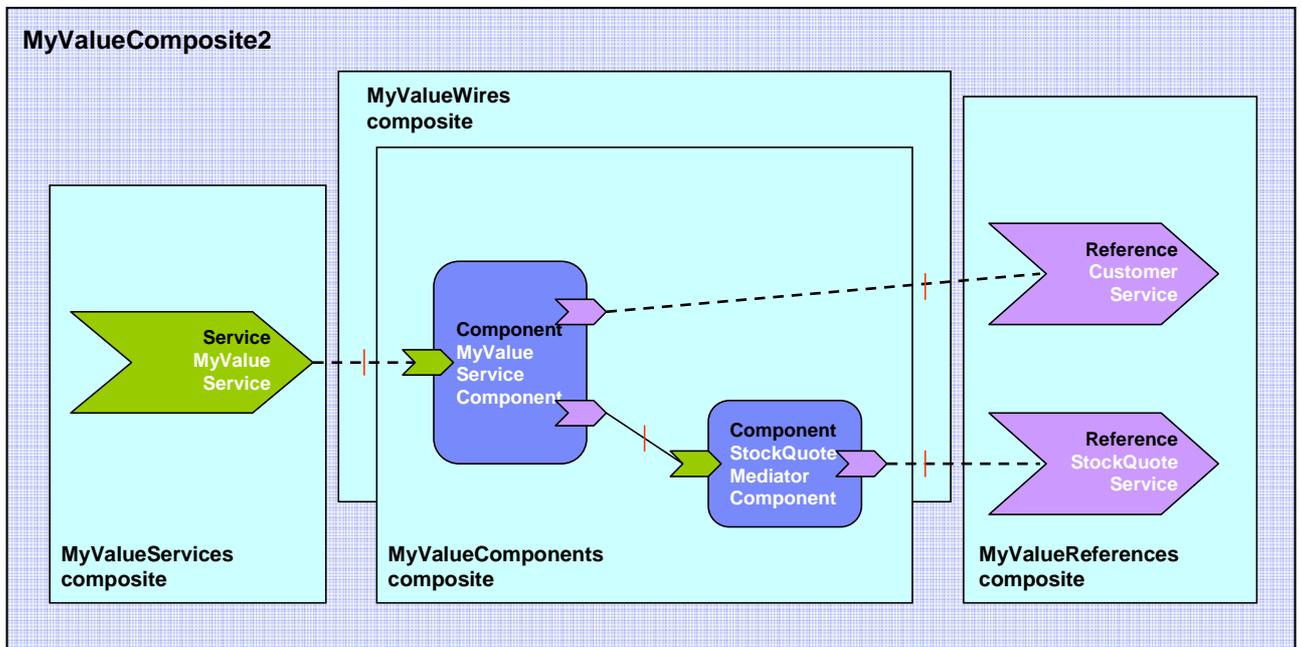
- 2166 • **name (required)** – the name of the composite that is included.

2167

### 2168 6.6.1 Included Composite Examples

2169

2170 The following figure shows the assembly diagram for the MyValueComposite2 containing four  
2171 included composites. The **MyValueServices composite** contains the MyValueService service. The  
2172 **MyValueComponents composite** contains the MyValueServiceComponent and the  
2173 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences**  
2174 **composite** contains the CustomerService and StockQuoteService references. The **MyValueWires**  
2175 **composite** contains the wires that connect the MyValueService service to the  
2176 MyValueServiceComponent, that connect the customerService reference of the  
2177 MyValueServiceComponent to the CustomerService reference, and that connect the  
2178 stockQuoteService reference of the StockQuoteMediatorComponent to the StockQuoteService  
2179 reference. Note that this is just one possible way of building the MyValueComposite2 from a set of  
2180 included composites.



2181

2182

2183 *Figure 13 MyValueComposite2 built from 4 included composites*

2184

2185 The following snippet shows the contents of the MyValueComposite2.composite file for the  
2186 MyValueComposite2 built using included composites. In this sample it only provides the name of  
2187 the composite. The composite file itself could be used in a scenario using included composites to  
2188 define components, services, references and wires.

```
2189  
2190 <?xml version="1.0" encoding="ASCII"?>  
2191 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2192 targetNamespace="http://foo.com"  
2193 xmlns:foo="http://foo.com"  
2194 name="MyValueComposite2" >  
2195  
2196 <include name="foo:MyValueServices"/>  
2197 <include name="foo:MyValueComponents"/>  
2198 <include name="foo:MyValueReferences"/>  
2199 <include name="foo:MyValueWires"/>  
2200  
2201 </composite>
```

2202  
2203 The following snippet shows the content of the MyValueServices.composite file.

```
2204  
2205 <?xml version="1.0" encoding="ASCII"?>  
2206 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2207 targetNamespace="http://foo.com"  
2208 xmlns:foo="http://foo.com"  
2209 name="MyValueServices" >  
2210  
2211 <service name="MyValueService" promote="MyValueServiceComponent">  
2212 <interface.java interface="services.myvalue.MyValueService"/>  
2213 <binding.ws port="http://www.myvalue.org/MyValueService#"  
2214 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>  
2215 </service>  
2216  
2217 </composite>
```

2218  
2219 The following snippet shows the content of the MyValueComponents.composite file.

```
2220  
2221 <?xml version="1.0" encoding="ASCII"?>  
2222 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
2223 targetNamespace="http://foo.com"  
2224 xmlns:foo="http://foo.com"  
2225 name="MyValueComponents" >  
2226  
2227 <component name="MyValueServiceComponent" >
```

```

2228         <implementation.java
2229         class="services.myvalue.MyValueServiceImpl"/>
2230         <property name="currency">EURO</property>
2231     </component>
2232
2233     <component name="StockQuoteMediatorComponent">
2234         <implementation.java class="services.myvalue.SQMediatorImpl"/>
2235         <property name="currency">EURO</property>
2236     </component>
2237
2238 </composite>
2239

```

2240 The following snippet shows the content of the MyValueReferences.composite file.

```

2241
2242 <?xml version="1.0" encoding="ASCII"?>
2243 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2244           targetNamespace="http://foo.com"
2245           xmlns:foo="http://foo.com"
2246           name="MyValueReferences" >
2247
2248     <reference name="CustomerService"
2249             promote="MyValueServiceComponent/CustomerService">
2250         <interface.java interface="services.customer.CustomerService"/>
2251         <binding.sca/>
2252     </reference>
2253
2254     <reference name="StockQuoteService"
2255             promote="StockQuoteMediatorComponent">
2256         <interface.java
2257             interface="services.stockquote.StockQuoteService"/>
2258         <binding.ws port="http://www.stockquote.org/StockQuoteService#
2259                 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2260     </reference>
2261
2262 </composite>

```

2263 The following snippet shows the content of the MyValueWires.composite file.

```

2264
2265 <?xml version="1.0" encoding="ASCII"?>
2266 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2267           targetNamespace="http://foo.com"
2268           xmlns:foo="http://foo.com"
2269           name="MyValueWires" >
2270
2271     <wire source="MyValueServiceComponent/stockQuoteService"

```

2272 `target="StockQuoteMediatorComponent"/>`

2273

2274 `</composite>`

## 2275 **6.7 Composites which Include Component Implementations of**

### 2276 **Multiple Types**

2277

2278 A Composite containing multiple components can have multiple component implementation types.  
2279 For example, a Composite may include one component with a Java POJO as its implementation  
2280 and another component with a BPEL process as its implementation.

2281

2282

## 7 ConstrainingType

2283

2284

2285

2286

2287

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.

2288

2289

2290

2291

2292

2293

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.

2294

2295

2296

2297

2298

2299

2300

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`.

2301

2302

2303

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]

2304

2305

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

2306

2307

2308

2309

2310

2311

2312

2313

2314

2315

2316

2317

2318

2319

2320

2321

2322

2323

2324

2325

2326

```
<?xml version="1.0" encoding="ASCII"?>
<!-- ConstrainingType schema snippet -->
<constrainingType xmlns="http://docs.oasis-
open.org/ns/opencsa/sca/200712"
    targetNamespace="xs:anyURI"?
    name="xs:NCName" requires="list of xs:QName"?>
    <service name="xs:NCName" requires="list of xs:QName"?>*
        <interface ... />?
    </service>
    <reference name="xs:NCName"
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?
        requires="list of xs:QName"?>*
        <interface ... />?
    </reference>
    <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
        many="xs:boolean"? mustSupply="xs:boolean"?>*
```

```
2327         default-property-value?
2328     </property>
2329
2330 </constrainingType>
2331
```

2332 The constrainingType element has the following **attributes**:

- 2333 • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is  
2334 an XML QName, in the namespace identified by the targetNamespace attribute. The name  
2335 attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2336 • **targetNamespace (0..1)** – an identifier for a target namespace into which the  
2337 constrainingType is declared
- 2338 • **requires (0..1)** – a list of policy intents. See [the Policy Framework specification \[10\]](#) for  
2339 a description of this attribute.

2340 ConstrainingType contains **zero or more properties, services, references**.

2341

2342 When an implementation is constrained by a constrainingType its component type MUST contain  
2343 all the services, references and properties specified in the constrainingType. [ASM70004] The  
2344 constraining type's references and services will have interfaces specified and can have intents  
2345 specified. An implementation MAY contain additional services, additional optional references  
2346 (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the  
2347 constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or  
2348 1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

2349 When a component is constrained by a constrainingType via the "constrainingType" attribute, the  
2350 entire componentType associated with the component and its implementation is not visible to the  
2351 containing composite. The containing composite can only see a projection of the componentType  
2352 associated with the component and implementation as scoped by the constrainingType of the  
2353 component. Additional services, references and properties provided by the implementation which  
2354 are not declared in the constrainingType associated with a component MUST NOT be configured in  
2355 any way by the containing composite. [ASM70006] This requirement ensures that the  
2356 constrainingType contract cannot be violated by the composite.

2357 The constrainingType can include required intents on any element. Those intents are applied to  
2358 any component that uses that constrainingType. In other words, if requires="reliability" exists on  
2359 a constrainingType, or its child service or reference elements, then a constrained component or its  
2360 implementation must include requires="reliability" on the component or implementation or on its  
2361 corresponding service or reference. A component or implementation can use a qualified form of  
2362 an intent specified in unqualified form in the constrainingType, but if the constrainingType uses  
2363 the qualified form of an intent, then the component or implementation MUST also use the qualified  
2364 form, otherwise there is an error. [ASM70007]

2365 A constrainingType can be applied to an implementation. In this case, the implementation's  
2366 componentType has a constrainingType attribute set to the QName of the constrainingType.

2367

## 2368 7.1 Example constrainingType

2369

2370 The following snippet shows the contents of the component called "MyValueServiceComponent"  
2371 which is constrained by the constrainingType myns:CT. The componentType associated with the  
2372 implementation is also shown.

2373

```
2374     <component name="MyValueServiceComponent" constrainingType="myns:CT">
2375         <implementation.java class="services.myvalue.MyValueServiceImpl" />
```

```

2376     <property name="currency">EURO</property>
2377     <reference name="customerService" target="CustomerService">
2378         <binding.ws ...>
2379     <reference name="StockQuoteService"
2380         target="StockQuoteMediatorComponent"/>
2381 </component>
2382
2383 <constrainingType name="CT"
2384     targetNamespace="http://mysns.com">
2385     <service name="MyValueService">
2386         <interface.java interface="services.myvalue.MyValueService"/>
2387     </service>
2388     <reference name="customerService">
2389         <interface.java interface="services.customer.CustomerService"/>
2390     </reference>
2391     <reference name="stockQuoteService">
2392         <interface.java interface="services.stockquote.StockQuoteService"/>
2393     </reference>
2394     <property name="currency" type="xsd:string"/>
2395 </constrainingType>

```

2396 The component MyValueServiceComponent is constrained by the constrainingType CT which  
2397 means that it must provide:

- 2398 • service **MyValueService** with the interface services.myvalue.MyValueService
- 2399 • reference **customerService** with the interface services.stockquote.StockQuoteService
- 2400 • reference **stockQuoteService** with the interface services.stockquote.StockQuoteService
- 2401 • property **currency** of type xsd:string.

---

## 2402 8 Interface

2403 **Interfaces** define one or more business functions. These business functions are provided by  
2404 Services and are used by References. A Service offers the business functionality of exactly one  
2405 interface for use by other components. Each interface defines one or more service **operations**  
2406 and each operation has zero or one **request (input) message** and zero or one **response**  
2407 **(output) message**. The request and response messages can be simple types such as a string  
2408 value or they can be complex types.

2409 SCA currently supports the following interface type systems:

- 2410 • Java interfaces
- 2411 • WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))
- 2412 • WSDL 2.0 interfaces ([Web Services Definition Language \[8\]](#))
- 2413 • C++ classes

2414 SCA is also extensible in terms of interface types. Support for other interface type systems can be  
2415 added through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2416

2417 The following snippet shows the definition for the **interface** base element.

2418

```
2419 <interface requires="list of xs:QName"? policySets="list of xs:QName"?/>
```

2420

2421 The **interface** base element has the following **attributes**:

- 2422 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
2423 [\[10\]](#) for a description of this attribute
- 2424 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2425 [\[10\]](#) for a description of this attribute.

2426

2427 The following snippet shows a sample for the WSDL portType (WSDL 1.1) or WSDL interface  
2428 (WSDL 2.0) element.

2429

```
2430 <interface.wSDL interface="xs:anyURI" ... />
```

2431

2432 The interface.wSDL element has the following attributes:

- 2433 • **interface** – URI of the portType/interface with the following format.
  - 2434 ○ `<WSDL-namespace-URI>#wsdl.interface(<portTypeOrInterface-name>)`

2435 The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.0  
2436 document OR an interface element of a WSDL 2.0 document. [\[ASM80001\]](#)

2437

2438 The following snippet shows a sample for the WSDL portType/interface element.

2439

```
2440 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2441                                     wsdl.interface(StockQuo  
2442                                     te)"/>
```

2443

2444 For WSDL 1.1, the interface attribute points to a portType in the WSDL. For WSDL 2.0, the  
2445 interface attribute points to an interface in the WSDL. For the WSDL 1.1 portType and WSDL 2.0  
2446 interface type systems, arguments and return of the service operations are described using XML  
2447 schema.

2448 For information about Java interfaces, including details of SCA-specific annotations, see the SCA  
2449 Java Common Annotations and APIs specification [1].

## 2450 8.1 Local and Remotable Interfaces

2451 A remotable service is one which may be called by a client which is running in an operating system  
2452 process different from that of the service itself (this also applies to clients running on different  
2453 machines from the service). Whether a service of a component implementation is remotable is  
2454 defined by the interface of the service. In the case of Java this is defined by adding the  
2455 **@Remotable** annotation to the Java interface (see [Client and Implementation Model Specification](#)  
2456 [for Java](#)). WSDL defined interfaces are always remotable.

2457

2458 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**  
2459 interactions. Remotable service Interfaces MUST NOT make use of **method or operation**  
2460 **overloading**. [ASM80002]

2461

2462 Independent of whether the remotable service is called remotely from outside the process where  
2463 the service runs or from another component running in the same process, the data exchange  
2464 semantics are **by-value**.

2465 Implementations of remotable services can modify input messages (parameters) during or after  
2466 an invocation and can modify return messages (results) after the invocation. If a remotable  
2467 service is called locally or remotely, the SCA container MUST ensure sure that no modification of  
2468 input messages by the service or post-invocation modifications to return messages are seen by  
2469 the caller. [ASM80003]

2470 Here is a snippet which shows an example of a remotable java interface:

2471

```
2472 package services.hello;  
2473  
2474 @Remotable  
2475 public interface HelloService {  
2476  
2477     String hello(String message);  
2478 }  
2479
```

2480 It is possible for the implementation of a remotable service to indicate that it can be called using  
2481 by-reference data exchange semantics when it is called from a component in the same process.  
2482 This can be used to improve performance for service invocations between components that run in  
2483 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java](#)  
2484 [Client and Implementation Specification](#)).

2485

2486 A service typed by a local interface can only be called by clients that are running in the same  
2487 process as the component that implements the local service. Local services cannot be published  
2488 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2489 Java interface definition without a **@Remotable** annotation.

2490

2491 The style of local interfaces is typically **fine grained** and intended for **tightly coupled**  
2492 interactions. Local service interfaces can make use of **method or operation overloading**.

2493 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.  
2494

## 2495 8.2 Bidirectional Interfaces

2496 The relationship of a business service to another business service is often peer-to-peer, requiring  
2497 a two-way dependency at the service level. In other words, a business service represents both a  
2498 consumer of a service provided by a partner business service and a provider of a service to the  
2499 partner business service. This is especially the case when the interactions are based on  
2500 asynchronous messaging rather than on remote procedure calls. The notion of **bidirectional**  
2501 **interfaces** is used in SCA to directly model peer-to-peer bidirectional business service  
2502 relationships.

2503 An interface element for a particular interface type system needs to allow the specification of an  
2504 optional callback interface. If a callback interface is specified, SCA refers to the interface as a  
2505 whole as a bidirectional interface.

2506 The following snippet shows the interface element defined using Java interfaces with an optional  
2507 callbackInterface attribute.

2508

```
2509 <interface.java          interface="services.invoicing.ComputePrice"  
2510                       callbackInterface="services.invoicing.InvoiceCallback"/>
```

2511

2512 If a service is defined using a bidirectional interface element then its implementation implements  
2513 the interface, and its implementation uses the callback interface to converse with the client that  
2514 called the service interface.

2515

2516 If a reference is defined using a bidirectional interface element, the client component  
2517 implementation using the reference calls the referenced service using the interface. The client  
2518 MUST provide an implementation of the callback interface. [ASM80004]

2519 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2520 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2521 mix local and remote services. [ASM80005]

2522

## 2523 8.3 Conversational Interfaces

2524 Services sometimes cannot easily be defined so that each operation stands alone and is  
2525 completely independent of the other operations of the same service. Instead, there is a sequence  
2526 of operations that must be called in order to achieve some higher level goal. SCA calls this  
2527 sequence of operations a **conversation**. If the service uses a bidirectional interface, the  
2528 conversation may include both operations and callbacks.

2529 Such **conversational services** are typically managed by using conversation identifiers that are  
2530 either (1) part of the application data (message parts or operation parameters) or 2)  
2531 communicated separately from application data (possibly in headers). SCA introduces the concept  
2532 of **conversational interfaces** for describing the interface contract for conversational services of  
2533 the second form above. With this form, it is possible for the runtime to automatically manage the  
2534 conversation, with the help of an appropriate binding specified at deployment. SCA does not  
2535 standardize any aspect of conversational services that are maintained using application data.  
2536 Such services are neither helped nor hindered by SCA's conversational service support.

2537 Conversational services typically involve state data that relates to the conversation that is taking  
2538 place. The creation and management of the state data for a conversation has a significant impact  
2539 on the development of both clients and implementations of conversational services.

2540

2541 Traditionally, application developers who have needed to write conversational services have been  
2542 required to write a lot of plumbing code. They need to:

2543

2544 - choose or define a protocol to communicate conversational (correlation) information  
2545 between the client & provider

2546 - route conversational messages in the provider to a machine that can handle that  
2547 conversation, while handling concurrent data access issues

2548 - write code in the client to use/encode the conversational information

2549 - maintain state that is specific to the conversation, sometimes persistently and  
2550 transactionally, both in the implementation and the client.

2551

2552 SCA makes it possible to divide the effort associated with conversational services between a  
2553 number of roles:

2554 - Application Developer: Declares that a service interface is conversational (leaving the  
2555 details of the protocol up to the binding). Uses lifecycle semantics, APIs or other  
2556 programmatic mechanisms (as defined by the implementation-type being used) to  
2557 manage conversational state.

2558 - Application Assembler: chooses a binding that can support conversations

2559 - Binding Provider: implements a protocol that can pass conversational information with  
2560 each operation request/response.

2561 - Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for  
2562 application developers to access conversational information. Optionally implements  
2563 instance lifecycle semantics that automatically manage implementation state based on  
2564 the binding's conversational information.

2565

2566 There is a policy intent with the name **conversational** which is used to mark an interface as being  
2567 conversational in nature. Where a service or a reference has a conversational interface, the  
2568 conversational intent MUST be attached either to the interface itself, or to the service or reference  
2569 using the interface. [ASM80006] How to attach the conversational intent to an interface depends  
2570 on the type of the interface. For a WSDL interface, this is described in section 8.4 "SCA-Specific  
2571 Aspects for WSDL Interfaces". For a Java interface, it is described in the Java Common  
2572 Annotations and APIs specification. Note that setting the conversational intent on the service or  
2573 reference element is useful when reusing an existing interface definition that contains no SCA  
2574 information, since it requires no modification of the interface artifact.

2575 The meaning of the conversational intent is that both the client and the provider of the interface  
2576 can assume that messages (in either direction) will be handled as part of an ongoing conversation  
2577 without depending on identifying information in the body of the message (i.e. in parameters of the  
2578 operations). In effect, the conversation interface specifies a high-level abstract protocol that must  
2579 be satisfied by any actual binding/policy combination used by the service.

2580 Examples of binding/policy combinations that support conversational interfaces are:

2581 - Web service binding with a WS-RM policy

2582 - Web service binding with a WS-Addressing policy

2583 - Web service binding with a WS-Context policy

2584 - JMS binding with a conversation policy that uses the JMS correlationID header

2585

2586 Conversations occur between one client and one target service. Consequently, requests originating  
2587 from one client to multiple target conversational services will result in multiple conversations. For  
2588 example, if a client A calls services B and C, both of which implement conversational interfaces,

2589 two conversations result, one between A and B and another between A and C. Likewise, requests  
2590 flowing through multiple implementation instances will result in multiple conversations. For  
2591 example, a request flowing from A to B and then from B to C will involve two conversations (A and  
2592 B, B and C). In the previous example, if a request was then made from C to A, a third  
2593 conversation would result (and the implementation instance for A would be different from the one  
2594 making the original request).

2595 Invocation of any operation of a conversational interface can start a conversation. The decision on  
2596 whether an operation starts a conversation depends on the component's implementation and its  
2597 implementation type. Implementation types can support components which provide conversational  
2598 services. If an implementation type does provide this support, the specification for that  
2599 implementation type defines a mechanism for determining when a new conversation should be  
2600 used for an operation (for example, in Java, the conversation is new on the first use of an injected  
2601 reference; in BPEL, the conversation is new when the client's partnerLink comes into scope).

2602

2603 One or more operations in a conversational interface can be annotated with an  
2604 **endsConversation** annotation (the mechanism for annotating the interface depends on the  
2605 interface type) which indicates that when the operation is invoked, the conversation is at an end.  
2606 Where an interface is **bidirectional**, operations may also be annotated in this way on operations  
2607 of the callback interface. When a conversation ending operation is called, it indicates to both the  
2608 client and the service provider that the conversation is complete. Once an operation marked with  
2609 endsConversation has been invoked, any subsequent attempts to call an operation or a callback  
2610 operation associated with the same conversation MUST generate a sca:ConversationViolation fault.  
2611 [\[ASM80007\]](#)

2612 A sca:ConversationViolation fault is thrown when one of the following errors occur:

- 2613 - A message is received for a particular conversation, after the conversation has ended
- 2614 - The conversation identification is invalid (not unique, out of range, etc.)
- 2615 - The conversation identification is not present in the input message of the operation that
- 2616 ends the conversation
- 2617 - The client or the service attempts to send a message in a conversation, after the
- 2618 conversation has ended

2619 This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI  
2620 <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2621 The lifecycle of resources and the association between unique identifiers and conversations are  
2622 determined by the service's implementation type and may not be directly affected by the  
2623 "endConversation" annotation. For example, a **WS-BPEL** process can outlive most of the  
2624 conversations that it is involved in.

2625 Although conversational interfaces do not require that any identifying information be passed as  
2626 part of the body of messages, there is conceptually an identity associated with the conversation.  
2627 Individual implementations types can have an API to access the ID associated with the  
2628 conversation, although no assumptions can be made about the structure of that identifier.  
2629 Implementation types can also have a means to set the conversation ID by either the client or the  
2630 service provider, although the operation may only be supported by some binding/policy  
2631 combinations.

2632 Implementation-type specifications are encouraged to define and provide conversational instance  
2633 lifecycle management for components that implement conversational interfaces. However,  
2634 implementations could also manage the conversational state manually.

2635

## 2636 8.4 SCA-Specific Aspects for WSDL Interfaces

2637 There are a number of aspects that SCA applies to interfaces in general, such as marking them  
2638 **conversational**. These aspects apply to the interfaces themselves, rather than their use in a  
2639 specific place within SCA. There is thus a need to provide appropriate ways of marking the

2640 interface definitions themselves, which go beyond the basic facilities provided by the interface  
2641 definition language.

2642 For WSDL interfaces, there is an extension mechanism that permits additional information to be  
2643 included within the WSDL document. SCA takes advantage of this extension mechanism. In order  
2644 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-](http://docs.oasis-open.org/ns/opencsa/sca/200712)  
2645 [open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2646 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach  
2647 policy intents - **@requires**. The definition of this attribute is as follows:

```
2648 <attribute name="requires" type="sca:listOfQNames" />
```

2649

```
2650 <simpleType name="listOfQNames">
```

```
2651 <list itemType="QName" />
```

```
2652 </simpleType>
```

2653 The @requires attribute can be applied to WSDL Port Type elements (WSDL 1.1) and to WSDL  
2654 Interface elements (WSDL 2.0). The attribute contains one or more intent names, as defined by  
2655 the [Policy Framework specification \[10\]](#). Any service or reference that uses an interface marked  
2656 with required intents MUST implicitly add those intents to its own @requires list. [ASM80008]

2657 To specify that a WSDL interface is conversational, the following attribute setting is used on either  
2658 the WSDL Port Type or WSDL Interface:

```
2659 requires="conversational"
```

2660 SCA defines an **endsConversation** attribute that is used to mark specific operations within a  
2661 WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces  
2662 which are also marked conversational. The endsConversation attribute is a global attribute in the  
2663 SCA namespace, with the following definition:

```
2664 <attribute name="endsConversation" type="boolean" default="false" />
```

2665

2666 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2667 the portType and the **endsConversation** attribute on one of the operations:

2668

```
2669 <portType name="LoanService" sca:requires="conversational">
```

2670

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

2671

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

2672

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

2673

```
</operation>
```

2674

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

2675

```
</operation>
```

2676

```
...
```

2677

```
</portType>
```

2678

```
...
```

## 2679 9 Binding

2680 Bindings are used by services and references. References use bindings to describe the access  
2681 mechanism used to call a service (which can be a service provided by another SCA composite).  
2682 Services use bindings to describe the access mechanism that clients (which can be a client from  
2683 another SCA composite) have to use to call the service.

2684 SCA supports the use of multiple different types of bindings. Examples include **SCA service, Web**  
2685 **service, stateless session EJB, data base stored procedure, EIS service**. An SCA runtime  
2686 MUST provide support for SCA service and Web service binding types. SCA provides an  
2687 extensibility mechanism by which an SCA runtime can add support for additional binding types.  
2688 For details on how additional binding types are defined, see the section on the Extension Model.

2689 A binding is defined by a **binding element** which is a child element of a service or of a reference  
2690 element in a composite. The following snippet shows the composite schema with the schema for  
2691 the binding element.

```
2692
2693 <?xml version="1.0" encoding="ASCII"?>
2694 <!-- Bindings schema snippet -->
2695 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2696           targetNamespace="xs:anyURI"
2697           name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2698           constrainingType="QName"?
2699           requires="list of xs:QName"? policySets="list of
2700 xs:QName"?>
2701
2702   ...
2703
2704   <service name="xs:NCName" promote="xs:anyURI"
2705           requires="list of xs:QName"? policySets="list of xs:QName"?>*
2706     <interface ... />?
2707     <binding uri="xs:anyURI"? name="xs:NCName"?
2708             requires="list of xs:QName"? policySets="list of
2709 xs:QName"?/>*
2710     <callback?
2711         <binding uri="xs:anyURI"? name="xs:NCName"?
2712             requires="list of xs:QName"?
2713             policySets="list of xs:QName"?/>+
2714     </callback>
2715 </service>
2716
2717   ...
2718
2719   <reference name="xs:NCName" target="list of xs:anyURI"?
2720             promote="list of xs:anyURI"? wiredByImpl="xs:boolean"?
2721             multiplicity="0..1 or 1..1 or 0..n or 1..n"?

```

```

2722         requires="list of xs:QName"? policySets="list of xs:QName"?>*
2723     <interface ... />?
2724     <binding uri="xs:anyURI"? name="xs:NCName"?
2725         requires="list of xs:QName"? policySets="list of
2726 xs:QName"?/>*
2727     <callback?>
2728         <binding uri="xs:anyURI"? name="xs:NCName"?
2729             requires="list of xs:QName"?
2730             policySets="list of xs:QName"?/>+
2731     </callback>
2732 </reference>
2733
2734     ...
2735
2736 </composite>
2737

```

2738 The element name of the binding element is architected; it is in itself a qualified name. The first  
2739 qualifier is always named "binding", and the second qualifier names the respective binding-type  
2740 (e.g. binding.composite, binding.ws, binding.ejb, binding.eis).

2741

2742 A binding element has the following attributes:

- 2743
- 2744 • **uri (0..1)** - has the following semantic.
    - 2745 ○ The uri attribute can be omitted.
    - 2746 ○ For a binding of a **reference** the URI attribute defines the target URI of the
    - 2747 reference. This MUST be either the componentName/serviceName for a wire to an
    - 2748 endpoint within the SCA domain, or the accessible address of some service
    - 2749 endpoint either inside or outside the SCA domain (where the addressing scheme is
    - 2750 defined by the type of the binding). [ASM90001]
    - 2751 ○ The circumstances under which the uri attribute can be used are defined in
    - 2752 section "5.3.1 Specifying the Target Service(s) for a Reference."
    - 2753 ○ For a binding of a **service** the URI attribute defines the URI relative to the
    - 2754 component, which contributes the service to the SCA domain. The default value for
    - 2755 the URI is the value of the name attribute of the binding.
  - 2756 • **name (0..1)** – a name for the binding instance (an NCName). The name attribute allows
  - 2757 distinction between multiple binding elements on a single service or reference. The
  - 2758 default value of the name attribute is the service or reference name. When a service or
  - 2759 reference has multiple bindings, only one binding can have the default name value; all
  - 2760 others must have a name value specified that is unique within the service or reference.
  - 2761 [ASM90002] The name also permits the binding instance to be referenced from elsewhere
  - 2762 – particularly useful for some types of binding, which can be declared in a definitions
  - 2763 document as a template and referenced from other binding instances, simplifying the
  - 2764 definition of more complex binding instances (see [the JMS Binding specification \[11\]](#) for
  - 2765 examples of this referencing).
  - 2766 • **requires (optional)** - a list of policy intents. See the [Policy Framework specification \[10\]](#)
  - 2767 for a description of this attribute.
  - 2768 • **policySets (optional)** – a list of policy sets. See the [Policy Framework specification \[10\]](#)
  - 2769 for a description of this attribute.

2769 When multiple bindings exist for an service, it means that the service is available by any of the  
2770 specified bindings. The technique that the SCA runtime uses to choose among available bindings  
2771 is left to the implementation and it may include additional (nonstandard) configuration. Whatever  
2772 technique is used needs to be documented by the runtime.

2773 Services and References can always have their bindings overridden at the SCA domain level,  
2774 unless restricted by Intents applied to them.

2775 If a reference has any bindings they MUST be resolved which means that each binding MUST  
2776 include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference  
2777 MUST NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds  
2778 of bindings that are acceptable for use with a reference, the user specifies either policy intents or  
2779 policy sets.

2780 Users can also specifically wire, not just to a component service, but to a specific binding offered  
2781 by that target service. To do so, a wire target MAY be specified with a syntax of  
2782 "componentName/serviceName/bindingName". [ASM90004]  
2783

2784

2785 The following sections describe the SCA and Web service binding type in detail.

2786

## 2787 9.1 Messages containing Data not defined in the Service Interface

2788 It is possible for a message to include information that is not defined in the interface used to  
2789 define the service, for instance information may be contained in SOAP headers or as MIME  
2790 attachments.

2791 Implementation types can make this information available to component implementations in their  
2792 execution context. The specifications for these implementation types describe how this  
2793 information is accessed and in what form it is presented.

2794

## 2795 9.2 Form of the URI of a Deployed Binding

2796

### 2797 9.2.1 Constructing Hierarchical URIs

2798 Bindings that use hierarchical URI schemes construct the effective URI with a combination of the  
2799 following pieces:

2800 Base System URI for a scheme / Component URI / Service Binding URI

2801

2802 Each of these components deserves addition definition:

2803 **Base Domain URI for a scheme.** An SCA domain should define a base URI for each hierarchical  
2804 URI scheme on which it intends to provide services.

2805 For example: the HTTP and HTTPS schemes would each have their own base URI defined for the  
2806 domain. An example of a scheme that is not hierarchical, and therefore will have no base URI is  
2807 the "jms:" scheme.

2808 **Component URI.** The component URI above is for a component that is deployed in the SCA  
2809 Domain. The URI of a component defaults to the name of the component, which is used as a  
2810 relative URI. The component may have a specified URI value. The specified URI value may be an  
2811 absolute URI in which case it becomes the Base URI for all the services belonging to the  
2812 component. If the specified URI value is a relative URI, it is used as the Component URI value  
2813 above.

2814 **Service Binding URI.** The Service Binding URI is the relative URI specified in the "uri" attribute  
2815 of a binding element of the service. The default value of the attribute is value of the binding's  
2816 name attribute treated as a relative URI. If multiple bindings for a single service use the same  
2817 scheme (e.g. HTTP), then only one of the bindings may depend on the default value for the uri  
2818 attribute, i.e. only one may use the default binding name. The service binding URI may also be  
2819 absolute, in which case the absolute URI fully specifies the full URI of the service. Some  
2820 deployment environments may not support the use of absolute URIs in service bindings.

2821 Services deployed into the Domain (as opposed to services of components) have a URI that does  
2822 not include a component name, i.e.:

2823 Base Domain URI for a scheme / Service Binding URI

2824 The name of the containing composite does not contribute to the URI of any service.

2825 For example, a service where the Base URI is "http://acme.com", the component is named  
2826 "stocksComponent" and the service binding name is "getQuote", the URI would look like this:

2827 http://acme.com/stocksComponent/getQuote

2828 Allowing a binding's relative URI to be specified that differs from the name of the service allows  
2829 the URI hierarchy of services to be designed independently of the organization of the domain.

2830 It is good practice to design the URI hierarchy to be independent of the domain organization, but  
2831 there may be times when domains are initially created using the default URI hierarchy. When this  
2832 is the case, the organization of the domain can be changed, while maintaining the form of the URI  
2833 hierarchy, by giving appropriate values to the **uri** attribute of select elements. Here is an example  
2834 of a change that can be made to the organization while maintaining the existing URIs:

2835 To move a subset of the services out of one component (say "foo") to a new component (say  
2836 "bar"), the new component should have bindings for the moved services specify a URI  
2837 "../foo/MovedService"..

2838 The URI attribute may also be used in order to create shorter URIs for some endpoints, where the  
2839 component name may not be present in the URI at all. For example, if a binding has a **uri**  
2840 attribute of "../myService" the component name will not be present in the URI.

## 2841 9.2.2 Non-hierarchical URIs

2842 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) may optionally make  
2843 use of the "uri" attribute, which is the complete representation of the URI for that service  
2844 binding. Where the binding does not use the "uri" attribute, the binding must offer a different  
2845 mechanism for specifying the service address.

## 2846 9.2.3 Determining the URI scheme of a deployed binding

2847 One of the things that needs to be determined when building the effective URI of a deployed  
2848 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is  
2849 binding type specific.

2850 If the binding type supports a single protocol then there is only one URI scheme associated with it.  
2851 In this case, that URI scheme is used.

2852 If the binding type supports multiple protocols, the binding type implementation determines the  
2853 URI scheme by introspecting the binding configuration, which may include the policy sets  
2854 associated with the binding.

2855 A good example of a binding type that supports multiple protocols is binding.ws, which can be  
2856 configured by referencing either an "abstract" WSDL element (i.e. portType or interface) or a  
2857 "concrete" WSDL element (i.e. binding, port or endpoint). When the binding references a PortType  
2858 or Interface, the protocol and therefore the URI scheme is derived from the intents/policy sets  
2859 attached to the binding. When the binding references a "concrete" WSDL element, there are two  
2860 cases:

- 2861 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most  
2862 common case. In this case, the URI scheme is given by the protocol/transport specified in the  
2863 WSDL binding element.
- 2864 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,  
2865 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"  
2866 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined  
2867 by looking at the policy sets attached to the binding.

2868 It's worth noting that an intent supported by a binding type may completely change the behavior  
2869 of the binding. For example, when the intent "confidentiality/transport" is required by an HTTP  
2870 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to  
2871 "https".

2872

## 2873 9.3 SCA Binding

2874 The SCA binding element is defined by the following schema.

2875

```
2876 <binding.sca />
```

2877

2878 The SCA binding can be used for service interactions between references and services contained  
2879 within the SCA domain. The way in which this binding type is implemented is not defined by the  
2880 SCA specification and it can be implemented in different ways by different SCA runtimes. The only  
2881 requirement is that the required qualities of service must be implemented for the SCA binding  
2882 type. The SCA binding type is **not** intended to be an interoperable binding type. For  
2883 interoperability, an interoperable binding type such as the Web service binding should be used.

2884 A service definition with no binding element specified uses the SCA binding.  
2885 <binding.sca/> would only have to be specified in override cases, or when you specify a  
2886 set of bindings on a service definition and the SCA binding should be one of them.

2887 If a reference does not have a binding, then the binding used can be any of the bindings  
2888 specified by the service provider, as long as the intents required by the reference and  
2889 the service are all respected.

2890 If the interface of the service or reference is local, then the local variant of the SCA  
2891 binding will be used. If the interface of the service or reference is remotable, then either  
2892 the local or remote variant of the SCA binding will be used depending on whether source  
2893 and target are co-located or not.

2894 If a reference specifies an URI via its uri attribute, then this provides the default wire to a service  
2895 provided by another domain level component. The value of the URI has to be as follows:

- 2896 • <domain-component-name>/<service-name>

2897

### 2898 9.3.1 Example SCA Binding

2899 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
2900 containing the service element for the MyValueService and a reference element for the  
2901 StockQuoteService. Both the service and the reference use an SCA binding. The target for the  
2902 reference is left undefined in this binding and would have to be supplied by the composite in which  
2903 this composite is used.

2904

```
2905 <?xml version="1.0" encoding="ASCII"?>
```

```
2906 <!-- Binding SCA example -->
```

```
2907     <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2908                   targetNamespace="http://foo.com"
2909                   name="MyValueComposite" >
2910
2911       <service name="MyValueService" promote="MyValueComponent" >
2912         <interface.java interface="services.myvalue.MyValueService" />
2913         <binding.sca/>
2914         ...
2915       </service>
2916
2917       ...
2918
2919       <reference name="StockQuoteService"
2920 promote="MyValueComponent/StockQuoteReference" >
2921         <interface.java
2922 interface="services.stockquote.StockQuoteService" />
2923         <binding.sca/>
2924       </reference>
2925
2926 </composite>
2927
```

## 2928 **9.4 Web Service Binding**

2929 SCA defines a Web services binding. This is described in [a separate specification document \[9\]](#).

2930

## 2931 **9.5 JMS Binding**

2932 SCA defines a JMS binding. This is described in [a separate specification document \[11\]](#).

2933

## 10 SCA Definitions

2934 There are a variety of SCA artifacts which are generally useful and which are not specific to a  
2935 particular composite or a particular component. These shared artifacts include intents, policy sets,  
2936 bindings, binding type definitions and implementation type definitions.

2937 All of these artifacts within an SCA Domain are defined in a global, SCA Domain-wide file named  
2938 definitions.xml. The definitions.xml file contains a definitions element that conforms to the  
2939 following pseudo-schema snippet:

```
2940 <?xml version="1.0" encoding="ASCII"?>
2941 <!-- Composite schema snippet -->
2942 <definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2943             targetNamespace="xs:anyURI">
2944
2945     <sca:intent/*>
2946
2947     <sca:policySet/*>
2948
2949     <sca:binding/*>
2950
2951     <sca:bindingType/*>
2952
2953     <sca:implementationType/*>
2954
2955 </definitions>
```

2956 The definitions element has the following attribute:

- 2957 • **targetNamespace (required)** – the namespace into which the child elements of this  
2958 definitions element are placed (used for artifact resolution)

2959 The definitions element contains optional child elements – intent, policySet, binding, bindingtype  
2960 and implementationType. These elements are described elsewhere in this specification or in [the](#)  
2961 [SCA Policy Framework specification \[10\]](#). The use of the elements declared within a definitions  
2962 element is described in the [SCA Policy Framework specification \[10\]](#) and in [the JMS Binding](#)  
2963 [specification \[11\]](#).

2964

2965  
2966  
2967  
2968  
2969  
2970  
2971  
  
2972  
2973  
2974  
2975  
2976  
2977  
2978  
2979  
2980  
  
2981  
2982  
2983  
2984  
  
2985  
  
2986  
2987  
  
2988

## 11 Extension Model

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

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

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

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

### 11.1 Defining an Interface Type

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright SCA Collaboration 2006 -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        elementFormDefault="qualified">
...
    <element name="interface" type="sca:Interface" abstract="true"/>
    <complexType name="Interface"/>
    <complexType name="Interface" abstract="true">
      <attribute name="requires" type="sca:listOfQNames" use="optional"/>
      <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
    </complexType>
```

3009

3010       ...

3011

3012       </schema>

3013       In the following snippet is an example of how the base definition is extended to support Java  
3014 interfaces. The snippet shows the definition of the **interface.java** element and the  
3015 **JavaInterface** type contained in **sca-interface-java.xsd**.

3016

3017       <?xml version="1.0" encoding="UTF-8"?>

3018       <schema xmlns="http://www.w3.org/2001/XMLSchema"

3019               targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"

3020               xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">

3021

3022               <element name="interface.java" type="sca:JavaInterface"

3023                       substitutionGroup="sca:interface"/>

3024               <complexType name="JavaInterface">

3025                       <complexContent>

3026                               <extension base="sca:Interface">

3027                                       <attribute name="interface" type="NCName"

3028   use="required"/>

3029                               </extension>

3030                       </complexContent>

3031               </complexType>

3032       </schema>

3033       In the following snippet is an example of how the base definition can be extended by other  
3034 specifications to support a new interface not defined in the SCA specifications. The snippet shows  
3035 the definition of the **my-interface-extension** element and the **my-interface-extension-type**  
3036 type.

3037       <?xml version="1.0" encoding="UTF-8"?>

3038       <schema xmlns="http://www.w3.org/2001/XMLSchema"

3039               targetNamespace="http://www.example.org/myextension"

3040               xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"

3041               xmlns:tns="http://www.example.org/myextension">

3042

3043               <element name="my-interface-extension"

3044                       type="tns:my-interface-extension-type"

3045                       substitutionGroup="sca:interface"/>

3046               <complexType name="my-interface-extension-type">

3047                       <complexContent>

3048                               <extension base="sca:Interface">

3049                                       ...

3050                               </extension>

3051                       </complexContent>

3052               </complexType>

3053 </schema>  
3054

## 3055 11.2 Defining an Implementation Type

3056 The following snippet shows the base definition for the *implementation* element and  
3057 *Implementation* type contained in *sca-core.xsd*; see appendix for complete schema.

```
3058  
3059 <?xml version="1.0" encoding="UTF-8"?>  
3060 <!-- (c) Copyright SCA Collaboration 2006 -->  
3061 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3062         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3063         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3064         elementFormDefault="qualified">  
3065  
3066     ...  
3067  
3068     <element name="implementation" type="sca:Implementation"  
3069 abstract="true"/>  
3070     <complexType name="Implementation"/>  
3071  
3072     ...  
3073 </schema>  
3074
```

3075  
3076 In the following snippet we show how the base definition is extended to support Java  
3077 implementation. The snippet shows the definition of the *implementation.java* element and the  
3078 *JavaImplementation* type contained in *sca-implementation-java.xsd*.

```
3079  
3080 <?xml version="1.0" encoding="UTF-8"?>  
3081 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3082         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3083         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3084  
3085     <element name="implementation.java" type="sca:JavaImplementation"  
3086             substitutionGroup="sca:implementation"/>  
3087     <complexType name="JavaImplementation">  
3088         <complexContent>  
3089             <extension base="sca:Implementation">  
3090                 <attribute name="class" type="NCName"  
3091                     use="required"/>  
3092             </extension>  
3093         </complexContent>  
3094     </complexType>  
3095 </schema>
```

3096 In the following snippet is an example of how the base definition can be extended by other  
3097 specifications to support a new implementation type not defined in the SCA specifications. The  
3098 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**  
3099 **type** type.

```
3100 <?xml version="1.0" encoding="UTF-8"?>
3101 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3102         targetNamespace="http://www.example.org/myextension"
3103         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3104         xmlns:tns="http://www.example.org/myextension">
3105
3106     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3107           substitutionGroup="sca:implementation"/>
3108     <complexType name="my-impl-extension-type">
3109         <complexContent>
3110             <extension base="sca:Implementation">
3111                 ...
3112             </extension>
3113         </complexContent>
3114     </complexType>
3115 </schema>
3116
```

3117 In addition to the definition for the new implementation instance element, there needs to be an  
3118 associated implementationType element which provides metadata about the new implementation  
3119 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3120 <implementationType type="xs:QName"
3121                   alwaysProvides="list of intent xs:QName"
3122                   mayProvide="list of intent xs:QName"/>
3123
```

3124 The implementation type has the following attributes:

- 3125 • **type (1..1)** – the type of the implementation to which this implementationType element  
3126 applies. This is intended to be the QName of the implementation element for the  
3127 implementation type, such as "sca:implementation.java"
- 3128 • **alwaysProvides (0..1)** – a set of intents which the implementation type always  
3129 provides. See [the Policy Framework specification \[10\]](#) for details.
- 3130 • **mayProvide (0..1)** – a set of intents which the implementation type may provide. See  
3131 [the Policy Framework specification \[10\]](#) for details.

3132

## 3133 11.3 Defining a Binding Type

3134 The following snippet shows the base definition for the **binding** element and **Binding** type  
3135 contained in **sca-core.xsd**; see appendix for complete schema.

3136

```
3137 <?xml version="1.0" encoding="UTF-8"?>
3138 <!-- binding type schema snippet -->
3139 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

```

3140 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3141         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3142         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3143         elementFormDefault="qualified">
3144
3145     ...
3146
3147     <element name="binding" type="sca:Binding" abstract="true"/>
3148     <complexType name="Binding">
3149         <attribute name="uri" type="anyURI" use="optional"/>
3150         <attribute name="name" type="NCName" use="optional"/>
3151         <attribute name="requires" type="sca:listOfQNames"
3152             use="optional"/>
3153         <attribute name="policySets" type="sca:listOfQNames"
3154             use="optional"/>
3155     </complexType>
3156
3157     ...
3158
3159 </schema>

```

3160 In the following snippet is an example of how the base definition is extended to support Web  
3161 service binding. The snippet shows the definition of the **binding.ws** element and the  
3162 **WebServiceBinding** type contained in **sca-binding-webservice.xsd**.

```

3163
3164 <?xml version="1.0" encoding="UTF-8"?>
3165 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3166         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3167         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3168
3169     <element name="binding.ws" type="sca:WebServiceBinding"
3170         substitutionGroup="sca:binding"/>
3171     <complexType name="WebServiceBinding">
3172         <complexContent>
3173             <extension base="sca:Binding">
3174                 <attribute name="port" type="anyURI" use="required"/>
3175             </extension>
3176         </complexContent>
3177     </complexType>
3178 </schema>

```

3179 In the following snippet is an example of how the base definition can be extended by other  
3180 specifications to support a new binding not defined in the SCA specifications. The snippet shows  
3181 the definition of the **my-binding-extension** element and the **my-binding-extension-type** type.

```

3182 <?xml version="1.0" encoding="UTF-8"?>
3183 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

3184         targetNamespace="http://www.example.org/myextension"
3185         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3186         xmlns:tns="http://www.example.org/myextension" >
3187
3188     <element name="my-binding-extension"
3189           type="tns:my-binding-extension-type"
3190           substitutionGroup="sca:binding"/>
3191     <complexType name="my-binding-extension-type">
3192         <complexContent>
3193             <extension base="sca:Binding">
3194                 ...
3195             </extension>
3196         </complexContent>
3197     </complexType>
3198 </schema>
3199

```

3200 In addition to the definition for the new binding instance element, there needs to be an associated  
3201 bindingType element which provides metadata about the new binding type. The pseudo schema  
3202 for the bindingType element is shown in the following snippet:

```

3203 <bindingType type="xs:QName"
3204             alwaysProvides="list of intent QNames"?
3205             mayProvide = "list of intent QNames"?/>
3206

```

3207 The binding type has the following attributes:

- 3208 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3209 intended to be the QName of the binding element for the binding type, such as  
3210 "sca:binding.ws"
- 3211 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3212 [the Policy Framework specification \[10\]](#) for details.
- 3213 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)  
3214 [Policy Framework specification \[10\]](#) for details.

---

## 3215 12 Packaging and Deployment

### 3216 12.1 Domains

3217 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3218 of interconnected runtime nodes.

3219 A single SCA domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3220 wires can only be used to connect components within a single SCA domain. Connections to  
3221 services outside the domain must use binding specific mechanisms for addressing services (such  
3222 as WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used  
3223 in the context of a single domain. In general, external clients of a service that is developed and  
3224 deployed using SCA should not be able to tell that SCA was used to implement the service – it is  
3225 an implementation detail.

3226 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3227 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3228 functionality controlled by a single organization. For example, an SCA Domain may be the whole  
3229 of a business, or it may be a department within a business.

3230 As an example, for the accounts department in a business, the SCA Domain might cover all  
3231 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3232 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3233 An SCA domain has the following:

- 3234 • A virtual domain-level composite whose components are deployed and running
- 3235 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3236 necessary to execute components
- 3237 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3238 level composite.

3239 The information associated with an SCA domain can be stored in many ways, including but not  
3240 limited to a specific filesystem structure or a repository.

### 3241 12.2 Contributions

3242 An SCA domain might require a large number of different artifacts in order to work. These  
3243 artifacts include artifacts defined by SCA and other artifacts such as object code files and interface  
3244 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3245 different SCA definition documents are: composite, componentType, constrainingType and  
3246 definitions. XML artifacts that are not defined by SCA but which may be needed by an SCA  
3247 domain include XML Schema documents, WSDL documents, and BPEL documents. SCA  
3248 constructs, like other XML-defined constructs, use XML qualified names for their identity (i.e.  
3249 namespace + local name).

3250 Non-XML artifacts are also required within an SCA domain. The most obvious examples of such  
3251 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3252 implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3253 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3254 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3255 packaging formats, but requires that the ZIP format be supported. When using the ZIP format for  
3256 deploying a contribution, this specification does not specify whether that format is retained after  
3257 deployment. For example, a Java EE based SCA runtime may convert the ZIP package to an EAR  
3258 package. SCA expects certain characteristics of any packaging:

- 3259 • For any contribution packaging it MUST be possible to present the artifacts of the  
3260 packaging to SCA as a hierarchy of resources based off of a single root [\[ASM12001\]](#)

3261 • Within any contribution packaging A directory resource SHOULD exist at the root of the  
3262 hierarchy named META-INF [ASM12002]

3263 • Within any contribution packaging a document SHOULD exist directly under the META-INF  
3264 directory named sca-contribution.xml which lists the SCA Composites within the  
3265 contribution that are runnable. [ASM12003]

3266  
3267 The same document also optionally lists namespaces of constructs that are defined within  
3268 the contribution and which may be used by other contributions  
3269 Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the  
3270 namespaces of constructs that are needed by the contribution and which are be found  
3271 elsewhere, for example in other contributions. [ASM12004] These optional elements may  
3272 not be physically present in the packaging, but may be generated based on the definitions  
3273 and references that are present, or they may not exist at all if there are no unresolved  
3274 references.

3275  
3276 See the section "SCA Contribution Metadata Document" for details of the format of this  
3277 file.

3278 To illustrate that a variety of packaging formats can be used with SCA, the following are examples  
3279 of formats that might be used to package SCA artifacts and metadata (as well as other artifacts)  
3280 as a contribution:

- 3281 • A filesystem directory
- 3282 • An OSGi bundle
- 3283 • A compressed directory (zip, gzip, etc)
- 3284 • A JAR file (or its variants – WAR, EAR, etc)

3285 Contributions do not contain other contributions. If the packaging format is a JAR file that  
3286 contains other JAR files (or any similar nesting of other technologies), the internal files are not  
3287 treated as separate SCA contributions. It is up to the implementation to determine whether the  
3288 internal JAR file should be represented as a single artifact in the contribution hierarchy or whether  
3289 all of the contents should be represented as separate artifacts.

3290 A goal of SCA's approach to deployment is that the contents of a contribution should not need to  
3291 be modified in order to install and use the contents of the contribution in a domain.

3292

## 3293 12.2.1 SCA Artifact Resolution

3294 Contributions may be self-contained, in that all of the artifacts necessary to run the contents of  
3295 the contribution are found within the contribution itself. However, it can also be the case that the  
3296 contents of the contribution make one or many references to artifacts that are not contained  
3297 within the contribution. These references can be to SCA artifacts or they can be to other artifacts  
3298 such as WSDL files, XSD files or to code artifacts such as Java class files and BPEL scripts.

3299 A contribution can use some artifact-related or packaging-related means to resolve artifact  
3300 references. Examples of such mechanisms include:

- 3301 • wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema  
3302 artifacts respectively
- 3303 • OSGi bundle mechanisms for resolving Java class and related resource dependencies

3304 Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact  
3305 dependencies. [ASM12005]

3306 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanisms are  
3307 used either where no other mechanisms are available, or in cases where the mechanisms used by  
3308 the various contributions in the same SCA Domain are different. An example of the latter case is  
3309 where an OSGi Bundle is used for one contribution but where a second contribution used by the  
3310 first one is not implemented using OSGi - eg the second contribution is a mainframe COBOL

3311 service whose interfaces are declared using WSDL which must be accessed by the first  
3312 contribution.

3313 The SCA artifact resolution is likely to be most useful for SCA domains containing heterogeneous  
3314 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to  
3315 work across different kinds of contribution.

3316 SCA artifact resolution works on the principle that a contribution which needs to use artifacts  
3317 defined elsewhere expresses these dependencies using **import** statements in metadata belonging  
3318 to the contribution. A contribution controls which artifacts it makes available to other  
3319 contributions through **export** statements in metadata attached to the contribution.

3320

## 3321 12.2.2 SCA Contribution Metadata Document

3322 The contribution optionally contains a document that declares runnable composites, exported  
3323 definitions and imported definitions. The document is found at the path of META-INF/sca-  
3324 contribution.xml relative to the root of the contribution. Frequently some SCA metadata needs to  
3325 be specified by hand while other metadata is generated by tools (such as the <import> elements  
3326 described below). To accommodate this, it is also possible to have an identically structured  
3327 document at META-INF/sca-contribution-generated.xml. If this document exists (or is generated  
3328 on an as-needed basis), it will be merged into the contents of sca-contribution.xml, with the  
3329 entries in sca-contribution.xml taking priority if there are any conflicting declarations.

3330

3331 The format of the document is:

```
3332 <?xml version="1.0" encoding="ASCII"?>
3333 <!-- sca-contribution pseudo-schema -->
3334 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200712>
3335
3336     <deployable composite="xs:QName"/>*
3337     <import namespace="xs:String" location="xs:AnyURI"?/>*
3338     <export namespace="xs:String"/>*
3339
3340 </contribution>
```

3341

3342 **deployable element:** Identifies a composite which is a composite within the contribution that is a  
3343 composite intended for potential inclusion into the virtual domain-level composite. Other  
3344 composites in the contribution are not intended for inclusion but only for use by other composites.  
3345 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)  
3346 [Composite](#) capability and the add To Domain Level Composite capability.

3347 Attributes of the deployable element:

- 3348 • **composite (1..1)** – The QName of a composite within the contribution.

3349

3350 **Export element:** A declaration that artifacts belonging to a particular namespace are exported  
3351 and are available for use within other contributions. An export declaration in a contribution  
3352 specifies a namespace, all of whose definitions are considered to be exported. By default,  
3353 definitions are not exported.

3354 The SCA artifact export is useful for SCA domains containing heterogeneous mixtures of  
3355 contribution packagings and technologies, where artifact-related or packaging-related mechanisms  
3356 are unlikely to work across different kinds of contribution.

3357 Attributes of the export element:

3358 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3359 should be the namespace URI for the exported definitions. For XML technologies that  
3360 define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port  
3361 types are a different symbol space from WSDL bindings), all definitions from all symbol  
3362 spaces are exported.

3363  
3364 Technologies that use naming schemes other than QNames must use a different export  
3365 element from the same substitution group as the the SCA <export> element. The  
3366 element used identifies the technology, and can use any value for the namespace that is  
3367 appropriate for that technology. For example, <export.java> can be used can be used to  
3368 export java definitions, in which case the namespace is a fully qualified package name.

3369  
3370 **Import element:** Import declarations specify namespaces of definitions that are needed by the  
3371 definitions and implementations within the contribution, but which are not present in the  
3372 contribution. It is expected that in most cases import declarations will be generated based on  
3373 introspection of the contents of the contribution. In this case, the import declarations would be  
3374 found in the META-INF/ sca-contribution-generated.xml document.

3375 Attributes of the import element:

3376 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3377 is the namespace URI for the imported definitions. For XML technologies that define  
3378 multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are  
3379 a different symbol space from WSDL bindings), all definitions from all symbol spaces are  
3380 imported.

3381  
3382 Technologies that use naming schemes other than QNames must use a different import  
3383 element from the same substitution group as the the SCA <import> element. The  
3384 element used identifies the technology, and can use any value for the namespace that is  
3385 appropriate for that technology. For example, <import.java> can be used can be used to  
3386 import java definitions, in which case the namespace is a fully qualified package name.

3387 • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific  
3388 requirements for the form of this URI, nor the means by which it is resolved. It can point  
3389 to another contribution (through its URI) or it can point to some location entirely outside  
3390 the SCA Domain.  
3391

3392 It is expected that SCA runtimes can define implementation specific ways of resolving location  
3393 information for artifact resolution between contributions. These mechanisms will however usually  
3394 be limited to sets of contributions of one runtime technology and one hosting environment.

3395 In order to accommodate imports of artifacts between contributions of disparate runtime  
3396 technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location  
3397 specification.

3398 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are  
3399 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact  
3400 location specifications.

3401 The order in which the import statements are specified can play a role in this mechanism. Since  
3402 definitions of one namespace can be distributed across several artifacts, multiple import  
3403 declarations can be made for one namespace.  
3404

3405 The location value is only a default, and dependent contributions listed in the call to  
3406 installContribution can override the value if there is a conflict. However, the specific mechanism  
3407 for resolving conflicts between contributions that define conflicting definitions is implementation  
3408 specific.

3409  
3410 If the value of the location attribute is an SCA contribution URI, then the contribution packaging  
3411 can become dependent on the deployment environment. In order to avoid such a dependency,

3412 dependent contributions should be specified only when deploying or updating contributions as  
3413 specified in the section 'Operations for Contributions' below.

### 3414 12.2.3 Contribution Packaging using ZIP

3415 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires  
3416 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This  
3417 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3418 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3419 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-  
3420 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such  
3421 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3422 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)  
3423 [.ZIP file format \[12\]](#).

3424

## 3425 12.3 Installed Contribution

3426 As noted in the section above, the contents of a contribution do not need to be modified in order  
3427 to install and use it within a domain. An *installed contribution* is a contribution with all of the  
3428 associated information necessary in order to execute *deployable composites* within the  
3429 contribution.

3430 An installed contribution is made up of the following things:

- 3431 • Contribution Packaging – the contribution that will be used as the starting point for  
3432 resolving all references
- 3433 • Contribution base URI
- 3434 • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3435 the import statements from the root composite and from other dependent contributions
  - 3436 ○ Dependent contributions might or might not be shared with other installed  
3437 contributions.
  - 3438 ○ When the snapshot of any contribution is taken is implementation defined, ranging  
3439 from the time the contribution is installed to the time of execution
- 3440 • Deployment-time composites.  
3441 These are composites that are added into an installed contribution after it has been  
3442 deployed. This makes it possible to provide final configuration and access to  
3443 implementations within a contribution without having to modify the contribution. These  
3444 are optional, as composites that already exist within the contribution can also be used for  
3445 deployment.

3446

3447 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3448 fully qualified class names in Java).

3449 If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3450 algorithm used to determine the qualified name to use is implementation dependent.

3451 Implementations of SCA MAY also generate an error if there are conflicting names exported from  
3452 multiple contributions. [ASM12007]

3453

### 3454 12.3.1 Installed Artifact URIs

3455 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3456 constructed by starting with the base URI of the contribution and adding the relative URI of each  
3457 artifact (recalling that SCA requires that any packaging format be able to offer up its artifacts in a  
3458 single hierarchy).

3459

## 3460 12.4 Operations for Contributions

3461 SCA Domains provide the following conceptual functionality associated with contributions  
3462 (meaning the function might not be represented as addressable services and also meaning that  
3463 equivalent functionality might be provided in other ways). The functionality is optional meaning  
3464 that some SCA runtimes MAY choose not to provide the contribution functions functionality in any  
3465 way. [ASM12008]

### 3466 12.4.1 install Contribution & update Contribution

3467 Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3468 supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3469 contributions that should be used to resolve the dependencies of the root contribution and other  
3470 dependent contributions. These override any dependent contributions explicitly listed via the  
3471 location attribute in the import statements of the contribution.

3472 SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3473 means that all other exported artifacts can be used from that contribution. Because of this, the  
3474 dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3475 what is being used from each one.

3476 Each dependent contribution is also an installed contribution, with its own dependent  
3477 contributions. By default these dependent contributions of the dependent contributions (which we  
3478 will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3479 contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3480 definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3481 included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3482 Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3483 MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3484 Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3485 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,  
3486 then it is easy for this list to be generated by tooling.

### 3487 12.4.2 add Deployment Composite & update Deployment Composite

3488 Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3489 data structure, not an existing resource in the domain) to the contribution identified by a supplied  
3490 contribution URI. The added or updated deployment composite is given a relative URI that  
3491 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3492 must run within the context of a installed contribution (any component implementations or other  
3493 definitions are resolved within that contribution), this functionality makes it possible for the  
3494 deployer to create a composite with final configuration and wiring decisions and add it to an  
3495 installed contribution without having to modify the contents of the root contribution.

3496 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3497 It is then possible for those to be given component names by a (possibly generated) composite  
3498 that is added into the installed contribution, without having to modify the packaging.

### 3499 12.4.3 remove Contribution

3500 Removes the deployed contribution identified by a supplied contribution URI.

3501

## 3502 12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3503

3504 For certain types of artifact, there are existing and commonly used mechanisms for referencing a  
3505 specific concrete location where the artifact can be resolved.

3506 Examples of these mechanisms include:

- 3507 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the  
3508 place holding the WSDL itself.
- 3509 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a  
3510 URI where the XSD is found.

3511 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does  
3512 not have to be dereferenced.

3513 SCA permits the use of these mechanisms. Where present, non-SCA artifact resolution  
3514 mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.  
3515 [ASM12010] However, use of these mechanisms is discouraged because tying assemblies to  
3516 addresses in this way makes the assemblies less flexible and prone to errors when changes are  
3517 made to the overall SCA Domain.

3518 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to  
3519 find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the  
3520 SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an  
3521 alternative. [ASM12011]

3522

## 3523 12.6 Domain-Level Composite

3524 The domain-level composite is a virtual composite, in that it is not defined by a composite  
3525 definition document. Rather, it is built up and modified through operations on the domain.  
3526 However, in other respects it is very much like a composite, since it contains components, wires,  
3527 services and references.

3528

3529 The value of @autowire for the logical domain composite MUST be autowire="false". [ASM12012]

3530

3531 For components at the Domain level, with References for which @autowire="true" applies, the  
3532 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

3533 1. The SCA runtime MAY disallow deployment of any components with autowire References. In this  
3534 case, the SCA runtime MUST generate an exception at the point where the component is  
3535 deployed.

3536 2. The SCA runtime MAY evaluate the target(s) for the reference at the time that the component is  
3537 deployed and not update those targets when later deployment actions occur.

3538 The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later  
3539 deployment actions occur resulting in updated reference targets which match the new Domain  
3540 configuration. How the new configuration of the reference takes place is described by the  
3541 relevant client and implementation specifications.

3542 [ASM12013]

3543 The abstract domain-level functionality for modifying the domain-level composite is as follows,  
3544 although a runtime may supply equivalent functionality in a different form:

### 3545 12.6.1 add To Domain-Level Composite

3546 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.  
3547 The supplied composite URI must refer to a composite within a installed contribution. The  
3548 composite's installed contribution determines how the composite's artifacts are resolved (directly  
3549 and indirectly). The supplied composite is added to the domain composite with semantics that  
3550 correspond to the domain-level composite having an <include> statement that references the

3551 supplied composite. All of the composite's components become *top-level* components and the  
3552 services become externally visible services (eg. they would be present in a WSDL description of  
3553 the domain).

### 3554 **12.6.2 remove From Domain-Level Composite**

3555 Removes from the Domain Level composite the elements corresponding to the composite  
3556 identified by a supplied composite URI. This means that the removal of the components, wires,  
3557 services and references originally added to the domain level composite by the identified  
3558 composite.

### 3559 **12.6.3 get Domain-Level Composite**

3560 Returns a <composite> definition that has an <include> line for each composite that had been  
3561 added to the domain level composite. It is important to note that, in dereferencing the included  
3562 composites, any referenced artifacts must be resolved in terms of that installed composite.

### 3563 **12.6.4 get QName Definition**

3564 In order to make sense of the domain-level composite (as returned by get Domain-Level  
3565 Composite), it must be possible to get the definitions for named artifacts in the included  
3566 composites. This functionality takes the supplied URI of an installed contribution (which provides  
3567 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
3568 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for  
3569 that definition type.

3570 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
3571 should exist in some form, but not necessarily as a service operation with exactly this signature.

---

3572 **13 Conformance**

- 3573 The XML schema available at the namespace URI, defined by this specification, is considered to be  
3574 authoritative and takes precedence over the XML Schema defined in the appendix of this document.
- 3575 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema  
3576 **[ASM10001]**

3577

## A. Pseudo Schema

3578

### A.1 ComponentType

3579

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

3580

```
<!-- Component type schema snippet -->
```

3581

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

3582

```
  constrainingType="QName"? >
```

3583

3584

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

3585

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

3586

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

3587

```
    <binding uri="xs:anyURI"? name="xs:NCName"?
```

3588

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

3589

```
      policySets="list of xs:QName"?/>*
```

3590

```
    <callback?>
```

3591

```
      <binding ... />+
```

3592

```
    </callback>
```

3593

```
  </service>
```

3594

3595

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

3596

```
    target="list of xs:anyURI"? autowire="xs:boolean"?
```

3597

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

3598

```
    wiredByImpl="xs:boolean"? requires="list of xs:QName"?
```

3599

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

3600

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

3601

```
    <binding uri="xs:anyURI"? name="xs:NCName"?
```

3602

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

3603

```
      policySets="list of xs:QName"?/>*
```

3604

```
    <callback?>
```

3605

```
      <binding ... />+
```

3606

```
    </callback>
```

3607

```
  </reference>
```

3608

3609

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

3610

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

3611

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

3612

```
    default-property-value?
```

3613

```
  </property>
```

3614

3615

```
  <implementation requires="list of xs:QName"?
```

3616

```
    policySets="list of xs:QName"?/>?
```

3617  
3618     </componentType>  
3619

## 3620 **A.2 Composite**

```
3621     <?xml version="1.0" encoding="ASCII"?>
3622     <!-- Composite schema snippet -->
3623     <composite     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3624                   targetNamespace="xs:anyURI"
3625                   name="xs:NCName" local="xs:boolean"?
3626                   autowire="xs:boolean"? constrainingType="QName"?
3627                   requires="list of xs:QName"? policySets="list of
3628     xs:QName"?>
3629
3630         <include name="xs:QName"/>*
3631
3632         <service name="xs:NCName" promote="xs:anyURI"
3633                 requires="list of xs:QName"? policySets="list of xs:QName"?>*
3634             <interface ... />?
3635             <binding uri="xs:anyURI"? name="xs:NCName"?
3636                 requires="list of xs:QName"? policySets="list of
3637     xs:QName"?/>*
3638             <callback?
3639                 <binding uri="xs:anyURI"? name="xs:NCName"?
3640                     requires="list of xs:QName"?
3641                     policySets="list of xs:QName"?/>+
3642             </callback>
3643         </service>
3644
3645         <reference name="xs:NCName" target="list of xs:anyURI"?
3646                 promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
3647                 multiplicity="0..1 or 1..1 or 0..n or 1..n"?
3648                 requires="list of xs:QName"? policySets="list of xs:QName"?>*
3649             <interface ... />?
3650             <binding uri="xs:anyURI"? name="xs:NCName"?
3651                 requires="list of xs:QName"? policySets="list of
3652     xs:QName"?/>*
3653             <callback?
3654                 <binding uri="xs:anyURI"? name="xs:NCName"?
3655                     requires="list of xs:QName"?
3656                     policySets="list of xs:QName"?/>+
3657             </callback>
3658         </reference>
3659
```

```

3660     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3661         many="xs:boolean"? mustSupply="xs:boolean"?>*
3662         default-property-value?
3663     </property>
3664
3665     <component name="xs:NCName" autowire="xs:boolean"?
3666         requires="list of xs:QName"? policySets="list of xs:QName"?>*
3667     <implementation ... />?
3668     <service name="xs:NCName" requires="list of xs:QName"?
3669         policySets="list of xs:QName"?>*
3670     <interface ... />?
3671     <binding uri="xs:anyURI"? name="xs:NCName"?
3672         requires="list of xs:QName"?
3673         policySets="list of xs:QName"?/>*
3674     <callback?
3675         <binding uri="xs:anyURI"? name="xs:NCName"?
3676             requires="list of xs:QName"?
3677             policySets="list of xs:QName"?/>+
3678     </callback>
3679 </service>
3680 <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3681     source="xs:string"? file="xs:anyURI"? value="xs:string"?>*
3682     [<value>+ | xs:any+]?
3683 </property>
3684 <reference name="xs:NCName" target="list of xs:anyURI"?
3685     autowire="xs:boolean"? wiredByImpl="xs:boolean"?
3686     requires="list of xs:QName"? policySets="list of xs:QName"?
3687     multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
3688 <interface ... />?
3689 <binding uri="xs:anyURI"? name="xs:NCName"?
3690     requires="list of xs:QName"?
3691     policySets="list of xs:QName"?/>*
3692 <callback?
3693     <binding uri="xs:anyURI"? name="xs:NCName"?
3694         requires="list of xs:QName"?
3695         policySets="list of xs:QName"?/>+
3696 </callback>
3697 </reference>
3698 </component>
3699
3700 <wire source="xs:anyURI" target="xs:anyURI" />*
3701
3702 </composite>

```

3703

## B. XML Schemas

3704

### B.1 sca.xsd

3705

3706

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

3707

```
<!-- (c) Copyright SCA Collaboration 2006 -->
```

3708

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

3709

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

3710

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

3711

3712

```
  <include schemaLocation="sca-core.xsd"/>
```

3713

3714

```
  <include schemaLocation="sca-interface-java.xsd"/>
```

3715

```
  <include schemaLocation="sca-interface-wsdl.xsd"/>
```

3716

3717

```
  <include schemaLocation="sca-implementation-java.xsd"/>
```

3718

```
  <include schemaLocation="sca-implementation-composite.xsd"/>
```

3719

3720

```
  <include schemaLocation="sca-binding-webservice.xsd"/>
```

3721

```
  <include schemaLocation="sca-binding-jms.xsd"/>
```

3722

```
  <include schemaLocation="sca-binding-sca.xsd"/>
```

3723

3724

```
  <include schemaLocation="sca-definitions.xsd"/>
```

3725

```
  <include schemaLocation="sca-policy.xsd"/>
```

3726

3727

```
  <include schemaLocation="sca-contribution.xsd"/>
```

3728

3729

```
</schema>
```

3730

3731

### B.2 sca-core.xsd

3732

3733

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

3734

```
<!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

3735

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

3736

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

3737

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

3738

```
  elementFormDefault="qualified">
```

3739

3740

```
  <element name="componentType" type="sca:ComponentType"/>
```

3741

```
  <complexType name="ComponentType">
```

```

3742     <sequence>
3743         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
3744         <choice minOccurs="0" maxOccurs="unbounded">
3745             <element name="service" type="sca:ComponentService" />
3746             <element name="reference" type="sca:ComponentReference"/>
3747             <element name="property" type="sca:Property"/>
3748         </choice>
3749         <any namespace="##other" processContents="lax" minOccurs="0"
3750             maxOccurs="unbounded"/>
3751     </sequence>
3752     <attribute name="constrainingType" type="QName" use="optional"/>
3753     <anyAttribute namespace="##other" processContents="lax"/>
3754 </complexType>
3755
3756 <element name="composite" type="sca:Composite"/>
3757 <complexType name="Composite">
3758     <sequence>
3759         <element name="include" type="anyURI" minOccurs="0"
3760             maxOccurs="unbounded"/>
3761         <choice minOccurs="0" maxOccurs="unbounded">
3762             <element name="service" type="sca:Service"/>
3763             <element name="property" type="sca:Property"/>
3764             <element name="component" type="sca:Component"/>
3765             <element name="reference" type="sca:Reference"/>
3766             <element name="wire" type="sca:Wire"/>
3767         </choice>
3768         <any namespace="##other" processContents="lax" minOccurs="0"
3769             maxOccurs="unbounded"/>
3770     </sequence>
3771     <attribute name="name" type="NCName" use="required"/>
3772     <attribute name="targetNamespace" type="anyURI" use="required"/>
3773     <attribute name="local" type="boolean" use="optional"
3774 default="false"/>
3775     <attribute name="autowire" type="boolean" use="optional"
3776 default="false"/>
3777     <attribute name="constrainingType" type="QName" use="optional"/>
3778     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3779     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3780     <anyAttribute namespace="##other" processContents="lax"/>
3781 </complexType>
3782
3783 <complexType name="Service">
3784     <sequence>

```

```

3785     <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3786     <element name="operation" type="sca:Operation" minOccurs="0"
3787         maxOccurs="unbounded" />
3788     <choice minOccurs="0" maxOccurs="unbounded">
3789         <element ref="sca:binding" />
3790         <any namespace="##other" processContents="lax"
3791             minOccurs="0" maxOccurs="unbounded" />
3792     </choice>
3793     <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3794     <any namespace="##other" processContents="lax" minOccurs="0"
3795         maxOccurs="unbounded" />
3796 </sequence>
3797 <attribute name="name" type="NCName" use="required" />
3798 <attribute name="promote" type="anyURI" use="required" />
3799 <attribute name="requires" type="sca:listOfQNames" use="optional" />
3800 <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3801 <anyAttribute namespace="##other" processContents="lax" />
3802 </complexType>
3803
3804 <element name="interface" type="sca:Interface" abstract="true" />
3805 <complexType name="Interface" abstract="true">
3806     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3807     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3808 </complexType>
3809
3810 <complexType name="Reference">
3811     <sequence>
3812         <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3813         <element name="operation" type="sca:Operation" minOccurs="0"
3814             maxOccurs="unbounded" />
3815         <choice minOccurs="0" maxOccurs="unbounded">
3816             <element ref="sca:binding" />
3817             <any namespace="##other" processContents="lax" />
3818         </choice>
3819         <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3820         <any namespace="##other" processContents="lax" minOccurs="0"
3821             maxOccurs="unbounded" />
3822     </sequence>
3823     <attribute name="name" type="NCName" use="required" />
3824     <attribute name="target" type="sca:listOfAnyURIs" use="optional"/>
3825     <attribute name="wiredByImpl" type="boolean" use="optional"
3826 default="false"/>
3827     <attribute name="multiplicity" type="sca:Multiplicity"
3828         use="optional" default="1..1" />

```

```

3829     <attribute name="promote" type="sca:listOfAnyURIs" use="required" />
3830     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3831     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3832     <anyAttribute namespace="##other" processContents="lax" />
3833 </complexType>
3834
3835 <complexType name="SCAPropertyBase" mixed="true">
3836     <!-- mixed="true" to handle simple type -->
3837     <sequence>
3838         <choice minOccurs="0">
3839             <element name="value" minOccurs="1" maxOccurs="unbounded"
3840                 type="anyType"/>
3841             <any namespace="##any" processContents="lax" minOccurs="1"
3842                 maxOccurs="unbounded" />
3843             <!-- NOT an extension point; This xsd:any exists
3844                 to accept the element-based or complex type
3845                 property i.e. no element-based extension point
3846                 under "sca:property" -->
3847         </choice>
3848     </sequence>
3849 </complexType>
3850
3851 <!-- complex type for sca:property declaration -->
3852 <complexType name="Property" mixed="true">
3853     <complexContent>
3854         <extension base="sca:SCAPropertyBase">
3855             <!-- extension defines the place to hold default value -->
3856             <attribute name="name" type="NCName" use="required"/>
3857             <attribute name="value" type="xs:string" use="optional"/>
3858             <attribute name="type" type="QName" use="optional"/>
3859             <attribute name="element" type="QName" use="optional"/>
3860             <attribute name="many" type="boolean" default="false"
3861                 use="optional"/>
3862             <attribute name="mustSupply" type="boolean" default="false"
3863                 use="optional"/>
3864             <anyAttribute namespace="##other" processContents="lax"/>
3865             <!-- an extension point ; attribute-based only -->
3866         </extension>
3867     </complexContent>
3868 </complexType>
3869
3870 <complexType name="PropertyValue" mixed="true">
3871     <complexContent>

```

```

3872     <extension base="sca:SCAPropertyBase">
3873         <attribute name="name" type="NCName" use="required"/>
3874         <attribute name="value" type="xs:string" use="optional"/>
3875         <attribute name="type" type="QName" use="optional"/>
3876         <attribute name="element" type="QName" use="optional"/>
3877         <attribute name="many" type="boolean" default="false"
3878             use="optional"/>
3879         <attribute name="source" type="string" use="optional"/>
3880         <attribute name="file" type="anyURI" use="optional"/>
3881         <anyAttribute namespace="##other" processContents="lax"/>
3882         <!-- an extension point ; attribute-based only -->
3883     </extension>
3884 </complexContent>
3885 </complexType>
3886
3887 <element name="binding" type="sca:Binding" abstract="true"/>
3888 <complexType name="Binding" abstract="true">
3889     <sequence>
3890         <element name="operation" type="sca:Operation" minOccurs="0"
3891             maxOccurs="unbounded" />
3892     </sequence>
3893     <attribute name="uri" type="anyURI" use="optional"/>
3894     <attribute name="name" type="NCName" use="optional"/>
3895     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3896     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3897 </complexType>
3898
3899 <element name="bindingType" type="sca:BindingType"/>
3900 <complexType name="BindingType">
3901     <sequence minOccurs="0" maxOccurs="unbounded">
3902         <any namespace="##other" processContents="lax" />
3903     </sequence>
3904     <attribute name="type" type="QName" use="required"/>
3905     <attribute name="alwaysProvides" type="sca:listOfQNames"
3906 use="optional"/>
3907     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
3908     <anyAttribute namespace="##other" processContents="lax"/>
3909 </complexType>
3910
3911 <element name="callback" type="sca:Callback"/>
3912 <complexType name="Callback">
3913     <choice minOccurs="0" maxOccurs="unbounded">
3914         <element ref="sca:binding"/>

```

```

3915         <any namespace="##other" processContents="lax" />
3916     </choice>
3917     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3918     <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3919     <anyAttribute namespace="##other" processContents="lax" />
3920 </complexType>
3921
3922 <complexType name="Component">
3923     <sequence>
3924         <element ref="sca:implementation" minOccurs="0" maxOccurs="1" />
3925         <choice minOccurs="0" maxOccurs="unbounded">
3926             <element name="service" type="sca:ComponentService" />
3927             <element name="reference" type="sca:ComponentReference" />
3928             <element name="property" type="sca:PropertyValue" />
3929         </choice>
3930         <any namespace="##other" processContents="lax" minOccurs="0"
3931             maxOccurs="unbounded" />
3932     </sequence>
3933     <attribute name="name" type="NCName" use="required" />
3934     <attribute name="autowire" type="boolean" use="optional" />
3935     <attribute name="constrainingType" type="QName" use="optional" />
3936     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3937     <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3938     <anyAttribute namespace="##other" processContents="lax" />
3939 </complexType>
3940
3941 <complexType name="ComponentService">
3942     <complexContent>
3943         <restriction base="sca:Service">
3944             <sequence>
3945                 <element ref="sca:interface" minOccurs="0"
3946 maxOccurs="1" />
3947                 <element name="operation" type="sca:Operation"
3948 minOccurs="0"
3949                 maxOccurs="unbounded" />
3950                 <choice minOccurs="0" maxOccurs="unbounded">
3951                     <element ref="sca:binding" />
3952                     <any namespace="##other" processContents="lax"
3953                         minOccurs="0" maxOccurs="unbounded" />
3954                 </choice>
3955                 <element ref="sca:callback" minOccurs="0"
3956 maxOccurs="1" />
3957                 <any namespace="##other" processContents="lax"
3958 minOccurs="0"

```

```

3959         maxOccurs="unbounded" />
3960     </sequence>
3961     <attribute name="name" type="NCName" use="required" />
3962     <attribute name="requires" type="sca:listOfQNames"
3963         use="optional" />
3964     <attribute name="policySets" type="sca:listOfQNames"
3965         use="optional" />
3966     <anyAttribute namespace="##other" processContents="lax" />
3967 </restriction>
3968 </complexContent>
3969 </complexType>
3970
3971 <complexType name="ComponentReference">
3972     <complexContent>
3973         <restriction base="sca:Reference">
3974             <sequence>
3975                 <element ref="sca:interface" minOccurs="0"
3976 maxOccurs="1" />
3977                 <element name="operation" type="sca:Operation"
3978 minOccurs="0"
3979                 maxOccurs="unbounded" />
3980                 <choice minOccurs="0" maxOccurs="unbounded">
3981                     <element ref="sca:binding" />
3982                     <any namespace="##other" processContents="lax"
3983 />
3984                 </choice>
3985                 <element ref="sca:callback" minOccurs="0"
3986 maxOccurs="1" />
3987                 <any namespace="##other" processContents="lax"
3988 minOccurs="0"
3989                 maxOccurs="unbounded" />
3990             </sequence>
3991             <attribute name="name" type="NCName" use="required" />
3992             <attribute name="autowire" type="boolean" use="optional" />
3993             <attribute name="wiredByImpl" type="boolean" use="optional"
3994                 default="false" />
3995             <attribute name="target" type="sca:listOfAnyURIs"
3996 use="optional" />
3997             <attribute name="multiplicity" type="sca:Multiplicity"
3998                 use="optional" default="1..1" />
3999             <attribute name="requires" type="sca:listOfQNames"
4000 use="optional" />
4001             <attribute name="policySets" type="sca:listOfQNames"
4002                 use="optional" />
4003             <anyAttribute namespace="##other" processContents="lax" />

```

```

4004         </restriction>
4005     </complexContent>
4006 </complexType>
4007
4008 <element name="implementation" type="sca:Implementation"
4009     abstract="true" />
4010 <complexType name="Implementation" abstract="true">
4011     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4012     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4013 </complexType>
4014
4015 <element name="implementationType" type="sca:ImplementationType"/>
4016 <complexType name="ImplementationType">
4017     <sequence minOccurs="0" maxOccurs="unbounded">
4018         <any namespace="##other" processContents="lax" />
4019     </sequence>
4020     <attribute name="type" type="QName" use="required"/>
4021     <attribute name="alwaysProvides" type="sca:listOfQNames"
4022 use="optional"/>
4023     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
4024     <anyAttribute namespace="##other" processContents="lax"/>
4025 </complexType>
4026
4027 <complexType name="Wire">
4028     <sequence>
4029         <any namespace="##other" processContents="lax" minOccurs="0"
4030             maxOccurs="unbounded"/>
4031     </sequence>
4032     <attribute name="source" type="anyURI" use="required"/>
4033     <attribute name="target" type="anyURI" use="required"/>
4034     <anyAttribute namespace="##other" processContents="lax"/>
4035 </complexType>
4036
4037 <element name="include" type="sca:Include"/>
4038 <complexType name="Include">
4039     <attribute name="name" type="QName"/>
4040     <anyAttribute namespace="##other" processContents="lax"/>
4041 </complexType>
4042
4043 <complexType name="Operation">
4044     <attribute name="name" type="NCName" use="required"/>
4045     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4046     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>

```

```

4047     <anyAttribute namespace="##other" processContents="lax"/>
4048 </complexType>
4049
4050 <element name="constrainingType" type="sca:ConstrainingType"/>
4051 <complexType name="ConstrainingType">
4052     <sequence>
4053         <choice minOccurs="0" maxOccurs="unbounded">
4054             <element name="service" type="sca:ComponentService"/>
4055             <element name="reference" type="sca:ComponentReference"/>
4056             <element name="property" type="sca:Property" />
4057         </choice>
4058         <any namespace="##other" processContents="lax" minOccurs="0"
4059             maxOccurs="unbounded"/>
4060     </sequence>
4061     <attribute name="name" type="NCName" use="required"/>
4062     <attribute name="targetNamespace" type="anyURI"/>
4063     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4064     <anyAttribute namespace="##other" processContents="lax"/>
4065 </complexType>
4066
4067
4068 <simpleType name="Multiplicity">
4069     <restriction base="string">
4070         <enumeration value="0..1"/>
4071         <enumeration value="1..1"/>
4072         <enumeration value="0..n"/>
4073         <enumeration value="1..n"/>
4074     </restriction>
4075 </simpleType>
4076
4077 <simpleType name="OverrideOptions">
4078     <restriction base="string">
4079         <enumeration value="no"/>
4080         <enumeration value="may"/>
4081         <enumeration value="must"/>
4082     </restriction>
4083 </simpleType>
4084
4085 <!-- Global attribute definition for @requires to permit use of intents
4086     within WSDL documents -->
4087 <attribute name="requires" type="sca:listOfQNames"/>
4088
4089 <!-- Global attribute defintion for @endsConversation to mark operations

```

```

4090         as ending a conversation -->
4091         <attribute name="endsConversation" type="boolean" default="false"/>
4092
4093         <simpleType name="listOfQNames">
4094             <list itemType="QName"/>
4095         </simpleType>
4096
4097         <simpleType name="listOfAnyURIs">
4098             <list itemType="anyURI"/>
4099         </simpleType>
4100
4101     </schema>

```

### 4102 **B.3 sca-binding-sca.xsd**

```

4103
4104 <?xml version="1.0" encoding="UTF-8"?>
4105 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
4106 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4107         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4108         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4109         elementFormDefault="qualified">
4110
4111     <include schemaLocation="sca-core.xsd"/>
4112
4113     <element name="binding.sca" type="sca:SCABinding"
4114             substitutionGroup="sca:binding"/>
4115     <complexType name="SCABinding">
4116         <complexContent>
4117             <extension base="sca:Binding">
4118                 <sequence>
4119                     <element name="operation" type="sca:Operation"
4120 minOccurs="0"
4121                             maxOccurs="unbounded" />
4122                 </sequence>
4123                 <attribute name="uri" type="anyURI" use="optional"/>
4124                 <attribute name="name" type="QName" use="optional"/>
4125                 <attribute name="requires" type="sca:listOfQNames"
4126                             use="optional"/>
4127                 <attribute name="policySets" type="sca:listOfQNames"
4128                             use="optional"/>
4129                 <anyAttribute namespace="##other" processContents="lax"/>
4130             </extension>
4131         </complexContent>

```

```
4132     </complexType>
4133 </schema>
4134
```

## 4135 **B.4 sca-interface-java.xsd**

```
4136
4137 <?xml version="1.0" encoding="UTF-8"?>
4138 <!-- (c) Copyright SCA Collaboration 2006 -->
4139 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4140     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4141     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4142     elementFormDefault="qualified">
4143
4144     <include schemaLocation="sca-core.xsd"/>
4145
4146     <element name="interface.java" type="sca:JavaInterface"
4147         substitutionGroup="sca:interface"/>
4148     <complexType name="JavaInterface">
4149         <complexContent>
4150             <extension base="sca:Interface">
4151                 <sequence>
4152                     <any namespace="##other" processContents="lax"
4153 minOccurs="0"
4154                         maxOccurs="unbounded"/>
4155                 </sequence>
4156                 <attribute name="interface" type="NCName" use="required"/>
4157                 <attribute name="callbackInterface" type="NCName"
4158 use="optional"/>
4159                 <anyAttribute namespace="##other" processContents="lax"/>
4160             </extension>
4161         </complexContent>
4162     </complexType>
4163 </schema>
4164
```

## 4164 **B.5 sca-interface-wsdl.xsd**

```
4165
4166 <?xml version="1.0" encoding="UTF-8"?>
4167 <!-- (c) Copyright SCA Collaboration 2006 -->
4168 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4169     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4170     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4171     elementFormDefault="qualified">
4172
4173     <include schemaLocation="sca-core.xsd"/>
```

```

4174
4175     <element name="interface.wsdl" type="sca:WSDLPortType"
4176           substitutionGroup="sca:interface"/>
4177   <complexType name="WSDLPortType">
4178     <complexContent>
4179       <extension base="sca:Interface">
4180         <sequence>
4181           <any namespace="##other" processContents="lax"
4182 minOccurs="0"
4183           maxOccurs="unbounded"/>
4184         </sequence>
4185         <attribute name="interface" type="anyURI" use="required"/>
4186         <attribute name="callbackInterface" type="anyURI"
4187 use="optional"/>
4188         <anyAttribute namespace="##other" processContents="lax"/>
4189       </extension>
4190     </complexContent>
4191   </complexType>
4192 </schema>

```

## 4193 B.6 sca-implementation-java.xsd

```

4194
4195 <?xml version="1.0" encoding="UTF-8"?>
4196 <!-- (c) Copyright SCA Collaboration 2006 -->
4197 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4198   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4199   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4200   elementFormDefault="qualified">
4201
4202   <include schemaLocation="sca-core.xsd"/>
4203
4204   <element name="implementation.java" type="sca:JavaImplementation"
4205     substitutionGroup="sca:implementation"/>
4206   <complexType name="JavaImplementation">
4207     <complexContent>
4208       <extension base="sca:Implementation">
4209         <sequence>
4210           <any namespace="##other" processContents="lax"
4211 minOccurs="0" maxOccurs="unbounded"/>
4212         </sequence>
4213         <attribute name="class" type="NCName" use="required"/>
4214         <attribute name="requires" type="sca:listOfQNames"
4215 use="optional"/>
4216         <attribute name="policySets" type="sca:listOfQNames"

```

```
4217         use="optional"/>
4218         <anyAttribute namespace="##other" processContents="lax"/>
4219     </extension>
4220 </complexContent>
4221 </complexType>
4222 </schema>
```

## 4223 **B.7 sca-implementation-composite.xsd**

```
4224
4225 <?xml version="1.0" encoding="UTF-8"?>
4226 <!-- (c) Copyright SCA Collaboration 2006 -->
4227 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4228     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4229     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4230     elementFormDefault="qualified">
4231
4232     <include schemaLocation="sca-core.xsd"/>
4233     <element name="implementation.composite" type="sca:SCAImplementation"
4234         substitutionGroup="sca:implementation"/>
4235     <complexType name="SCAImplementation">
4236         <complexContent>
4237             <extension base="sca:Implementation">
4238                 <sequence>
4239                     <any namespace="##other" processContents="lax"
4240 minOccurs="0"
4241                         maxOccurs="unbounded"/>
4242                 </sequence>
4243                 <attribute name="name" type="QName" use="required"/>
4244                 <attribute name="requires" type="sca:listOfQNames"
4245 use="optional"/>
4246                 <attribute name="policySets" type="sca:listOfQNames"
4247                     use="optional"/>
4248                 <anyAttribute namespace="##other" processContents="lax"/>
4249             </extension>
4250         </complexContent>
4251     </complexType>
4252 </schema>
4253
```

## 4254 **B.8 sca-definitions.xsd**

```
4255
4256 <?xml version="1.0" encoding="UTF-8"?>
4257 <!-- (c) Copyright SCA Collaboration 2006 -->
```

```

4258 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4259     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4260     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4261     elementFormDefault="qualified">
4262
4263     <include schemaLocation="sca-core.xsd"/>
4264
4265     <element name="definitions">
4266         <complexType>
4267             <choice minOccurs="0" maxOccurs="unbounded">
4268                 <element ref="sca:intent"/>
4269                 <element ref="sca:policySet"/>
4270                 <element ref="sca:binding"/>
4271                 <element ref="sca:bindingType"/>
4272                 <element ref="sca:implementationType"/>
4273                 <any namespace="##other" processContents="lax" minOccurs="0"
4274                     maxOccurs="unbounded"/>
4275             </choice>
4276         </complexType>
4277     </element>
4278
4279 </schema>
4280

```

## 4281 **B.9 sca-binding-webservice.xsd**

4282 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4283 **B.10 sca-binding-jms.xsd**

4284 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4285 **B.11 sca-policy.xsd**

4286 Is described in [the SCA Policy Framework specification \[10\]](#)

4287

## 4288 **B.12 sca-contribution.xsd**

4289

```

4290 <?xml version="1.0" encoding="UTF-8"?>
4291 <!-- (c) Copyright SCA Collaboration 2007 -->
4292 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4293     targetNamespace="http://www.osea.org/xmlns/sca/1.0"
4294     xmlns:sca="http://www.osea.org/xmlns/sca/1.0"
4295     elementFormDefault="qualified">
4296
4297     <include schemaLocation="sca-core.xsd"/>
4298
4299

```

```

4300     <element name="contribution" type="sca:ContributionType"/>
4301     <complexType name="ContributionType">
4302         <sequence>
4303             <element name="deployable" type="sca:DeployableType"
4304 minOccurs="1" maxOccurs="unbounded"/>
4305             <element name="import" type="sca:ImportType" minOccurs="0"
4306 maxOccurs="unbounded"/>
4307             <element name="export" type="sca:ExportType" minOccurs="0"
4308 maxOccurs="unbounded"/>
4309             <any namespace="##other" processContents="lax" minOccurs="0"
4310 maxOccurs="unbounded"/>
4311         </sequence>
4312         <anyAttribute namespace="##other" processContents="lax"/>
4313     </complexType>
4314
4315
4316
4317     <complexType name="DeployableType">
4318         <sequence>
4319             <any namespace="##other" processContents="lax" minOccurs="0"
4320 maxOccurs="unbounded"/>
4321         </sequence>
4322         <attribute name="composite" type="QName" use="required"/>
4323         <anyAttribute namespace="##other" processContents="lax"/>
4324     </complexType>
4325
4326
4327     <complexType name="ImportType">
4328         <sequence>
4329             <any namespace="##other" processContents="lax" minOccurs="0"
4330 maxOccurs="unbounded"/>
4331         </sequence>
4332         <attribute name="namespace" type="string" use="required"/>
4333         <attribute name="location" type="anyURI" use="required"/>
4334         <anyAttribute namespace="##other" processContents="lax"/>
4335     </complexType>
4336
4337     <complexType name="ExportType">
4338         <sequence>
4339             <any namespace="##other" processContents="lax" minOccurs="0"
4340 maxOccurs="unbounded"/>
4341         </sequence>
4342         <attribute name="namespace" type="string" use="required"/>
4343         <anyAttribute namespace="##other" processContents="lax"/>
4344     </complexType>
4345
4346 </schema>
4347

```

---

## 4348 C. SCA Concepts

### 4349 C.1 Binding

4350 **Bindings** are used by services and references. References use bindings to describe the access  
4351 mechanism used to call the service to which they are wired. Services use bindings to describe the  
4352 access mechanism(s) that clients should use to call the service.

4353 SCA supports multiple different types of bindings. Examples include **SCA service, Web service,**  
4354 **stateless session EJB, data base stored procedure, EIS service.** SCA provides an extensibility  
4355 mechanism by which an SCA runtime can add support for additional binding types.

4356

### 4357 C.2 Component

4358 **SCA components** are configured instances of **SCA implementations**, which provide and consume  
4359 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines  
4360 an **extensibility mechanism** that allows you to introduce new implementation types. The current  
4361 specification does not mandate the implementation technologies to be supported by an SCA run-time,  
4362 vendors may choose to support the ones that are important for them. A single SCA implementation may  
4363 be used by multiple Components, each with a different configuration.

4364 The Component has a reference to an implementation of which it is an instance, a set of property values,  
4365 and a set of service reference values. Property values define the values of the properties of the  
4366 component as defined by the component's implementation. Reference values define the services that  
4367 resolve the references of the component as defined by its implementation. These values can either be a  
4368 particular service of a particular component, or a reference of the containing composite.

### 4369 C.3 Service

4370 **SCA services** are used to declare the externally accessible services of an **implementation**. For a  
4371 composite, a service is typically provided by a service of a component within the composite, or by a  
4372 reference defined by the composite. The latter case allows the republication of a service with a new  
4373 address and/or new bindings. The service can be thought of as a point at which messages from external  
4374 clients enter a composite or implementation.

4375 A service represents an addressable set of operations of an implementation that are designed to be  
4376 exposed for use by other implementations or exposed publicly for use elsewhere (eg public Web services  
4377 for use by other organizations). The operations provided by a service are specified by an Interface, as  
4378 are the operations required by the service client (if there is one). An implementation may contain  
4379 multiple services, when it is possible to address the services of the implementation separately.

4380 A service may be provided **as SCA remote services, as Web services, as stateless session EJB's, as**  
4381 **EIS services, and so on.** Services use **bindings** to describe the way in which they are published. SCA  
4382 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new  
4383 types of services.

#### 4384 C.3.1 Remotable Service

4385 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA  
4386 architecture. For example, SCA services of SCA implementations can define implementations of industry-  
4387 standard web services. Remotable services use pass-by-value semantics for parameters and returned  
4388 results.

4389 A service is remotable if it is defined by a WSDL port type or if it defined by a Java interface marked with  
4390 the @Remotable annotation.

### 4391 C.3.2 Local Service

4392 Local services are services that are designed to be only used “locally” by other implementations that are  
4393 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4394 Local services may rely on by-reference calling conventions, or may assume a very fine-grained  
4395 interaction style that is incompatible with remote distribution. They may also use technology-specific data-  
4396 types.

4397 Currently a service is local only if it defined by a Java interface not marked with the @Remotable  
4398 annotation.

4399

### 4400 C.4 Reference

4401 **SCA references** represent a dependency that an implementation has on a service that is supplied by  
4402 some other implementation, where the service to be used is specified through configuration. In other  
4403 words, a reference is a service that an implementation may call during the execution of its business  
4404 function. References are typed by an interface.

4405 For composites, composite references can be accessed by components within the composite like any  
4406 service provided by a component within the composite. Composite references can be used as the targets  
4407 of wires from component references when configuring Components.

4408 A composite reference can be used to access a service such as: an SCA service provided by another  
4409 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,  
4410 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4411 an **extensibility mechanism** that allows the introduction of new binding types to references.

4412

### 4413 C.5 Implementation

4414 An implementation is concept that is used to describe a piece of software technology such as a Java  
4415 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4416 service-oriented application. An SCA composite is also an implementation.

4417 Implementations define points of variability including properties that can be set and settable references to  
4418 other services. The points of variability are configured by a component that uses the implementation. The  
4419 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4420 C.6 Interface

4421 **Interfaces** define one or more business functions. These business functions are provided by Services  
4422 and are used by components through References. Services are defined by the Interface they implement.  
4423 SCA currently supports a number of interface type systems, for example:

- 4424 • Java interfaces
- 4425 • WSDL portTypes
- 4426 • C, C++ header files

4427

4428 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4429 interface type systems.

4430 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided  
4431 by each end of a service communication – this could be the case where a particular service requires a  
4432 “callback” interface on the client, which is calls during the process of handing service requests from the  
4433 client.

4434

## 4435 C.7 Composite

4436 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4437 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4438 be used to contribute elements to an **SCA Domain**.

4439 A **composite** has the following characteristics:

- 4440 • It may be used as a component implementation. When used in this way, it defines a boundary for  
4441 Component visibility. Components may not be directly referenced from outside of the composite  
4442 in which they are declared.
- 4443 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
4444 artifacts to an SCA domain.

4445

## 4446 C.8 Composite inclusion

4447 One composite can be used to provide part of the definition of another composite, through the process of  
4448 inclusion. This is intended to make team development of large composites easier. Included composites  
4449 are merged together into the using composite at deployment time to form a single logical composite.

4450 Composites are included into other composites through `<include.../>` elements in the using composite.  
4451 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
4452 specific location.

4453

## 4454 C.9 Property

4455 **Properties** allow for the configuration of an implementation with externally set data values. The data  
4456 value is provided through a Component, possibly sourced from the property of a containing composite.

4457 Each Property is defined by the implementation. Properties may be defined directly through the  
4458 implementation language or through annotations of implementations, where the implementation language  
4459 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
4460 type. For complex data types, XML schema is the preferred technology for defining the data types.

4461

## 4462 C.10 Domain

4463 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
4464 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
4465 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
4466 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4467 A domain specifies the instantiation, configuration and connection of a set of components, provided via  
4468 one or more composite files. The domain, like a composite, also has Services and References. Domains  
4469 also contain Wires which connect together the Components, Services and References.

4470

## 4471 C.11 Wire

4472 **SCA wires** connect **service references** to **services**.

4473 Within a composite, valid wire sources are component references and composite services. Valid wire  
4474 targets are component services and composite references.

4475 When using included composites, the sources and targets of the wires don't have to be declared in the  
4476 same composite as the composite that contains the wire. The sources and targets can be defined by  
4477 other included composites. Targets can also be external to the SCA domain.

4478

4479

## D. Conformance Items

4480

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

4481

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.
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. [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.
A property <value/> subelement MUST NOT be used when the @value attribute is used to specify the value for that property. [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 <b>multiplicity</b> 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 <b>interface</b> 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 the qualified form, otherwise there is an error.
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. [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 be found elsewhere, for example in other contributions.
[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.
Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms. [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".

<p>[ASM12013]</p>	<p>For components at the Domain level, with References for which @autowire="true" applies, the behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:</p> <ol style="list-style-type: none"> <li>3. 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.</li> <li>4. 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.</li> <li>5. 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.</li> </ol>

4482

---

4483 **E. Acknowledgements**

4484 The following individuals have participated in the creation of this specification and are gratefully  
4485 acknowledged:

4486 **Participants:**

4487 [Participant Name, Affiliation | Individual Member]

4488 [Participant Name, Affiliation | Individual Member]

4489

---

## F. Non-Normative Text

4491

## G. Revision History

4492 [optional; should not be included in OASIS Standards]

4493

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> </ul> <ul style="list-style-type: none"> <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 Combellack'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	2008-10-27	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.

4494