



# Service Component Architecture Assembly Model Specification Version 1.1

Committee Draft 01 Revision **8**

23rd December 2008

**Specification URIs:**

**This Version:**

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev8.html>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev8.doc>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev8.pdf>  
(Authoritative)

**Previous Version:**

**Latest Version:**

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

**Latest Approved Version:**

**Technical Committee:**

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

**Chair(s):**

Martin Chapman, Oracle  
Mike Edwards, IBM

**Editor(s):**

Michael Beisiegel, IBM  
Khanderao Khand, Oracle  
Anish Karmarkar, Oracle  
Sanjay Patil, SAP  
Michael Rowley, BEA Systems

**Related work:**

This specification replaces or supercedes:

- Service Component Architecture Assembly Model Specification Version 1.00, March 15, 2007

This specification is related to:

- Service Component Architecture Policy Framework Specification Version 1.1

**Declared XML Namespace(s):**

<http://docs.oasis-open.org/ns/opencsa/sca/200712>

**Abstract:**

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

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

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

This document describes the SCA Assembly Model, which covers

- A model for the assembly of services, both tightly coupled and loosely coupled
- A model for applying infrastructure capabilities to services and to service interactions, including Security and Transactions

**Status:**

This document was last revised or approved by the OASIS Service Component Architecture / Assembly (SCA-Assembly) TC on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

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

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (<http://www.oasis-open.org/committees/sca-assembly/ipr.php>).

---

**The non-normative errata page for this specification is located at**  
**<http://www.oasis-open.org/committees/sca-assembly/>**

---

## Notices

Copyright © OASIS® 2005, 2008. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

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

## Table of Contents

Committee Draft 01 Revision 8 .....	1
The non-normative errata page for this specification is located at <a href="http://www.oasis-open.org/committees/sca-assembly/">http://www.oasis-open.org/committees/sca-assembly/</a> .....	3
Notices .....	4
Table of Contents .....	5
1 Introduction .....	8
1.1 Terminology .....	8
1.2 Normative References .....	8
1.3 Naming Conventions .....	9
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 .....	21
5 Component .....	24
5.1 Implementation .....	25
5.2 Service .....	26
5.3 Reference .....	28
5.3.1 Specifying the Target Service(s) for a Reference .....	30
5.4 Property .....	31
5.5 Example Component .....	34
6 Composite .....	38
6.1 Service .....	40
6.1.1 Service Examples .....	41
6.2 Reference .....	43
6.2.1 Example Reference .....	45
6.3 Property .....	47
6.3.1 Property Examples .....	48
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 .....	59
6.6 Using Composites through Inclusion .....	59
6.6.1 Included Composite Examples .....	60
6.7 Composites which Include Component Implementations of Multiple Types .....	63

6.8	Structural URI of Components .....	63
7	ConstrainingType .....	65
7.1	Example constrainingType .....	66
8	Interface.....	68
8.1	Local and Remotable Interfaces .....	69
8.2	Bidirectional Interfaces .....	70
8.3	Conversational Interfaces.....	71
8.4	Long-running Request-Response Operations .....	73
8.4.1	Background .....	73
8.4.2	Definition of "long-running".....	73
8.4.3	The asyncInvocation Intent .....	74
8.4.4	Requirements on Bindings .....	74
8.4.5	Implementation Type Support.....	74
8.5	SCA-Specific Aspects for WSDL Interfaces .....	74
8.6	WSDL Interface Type .....	76
8.6.1	Example of interface.wsdl .....	76
9	Binding.....	77
9.1	Messages containing Data not defined in the Service Interface .....	79
9.2	WireFormat .....	79
9.3	OperationSelector.....	80
9.4	Form of the URI of a Deployed Binding.....	80
9.4.1	Non-hierarchical URIs .....	80
9.4.2	Determining the URI scheme of a deployed binding.....	80
9.5	SCA Binding .....	81
9.5.1	Example SCA Binding .....	82
9.6	Web Service Binding .....	82
9.7	JMS Binding.....	82
10	SCA Definitions .....	83
11	Extension Model .....	84
11.1	Defining an Interface Type.....	84
11.2	Defining an Implementation Type.....	86
11.3	Defining a Binding Type.....	87
11.4	Defining an Import Type .....	89
11.5	Defining an Export Type .....	91
12	Packaging and Deployment .....	93
12.1	Domains.....	93
12.2	Contributions.....	93
12.2.1	SCA Artifact Resolution.....	94
12.2.2	SCA Contribution Metadata Document .....	96
12.2.3	Contribution Packaging using ZIP .....	98
12.3	Installed Contribution .....	98
12.3.1	Installed Artifact URIs.....	99
12.4	Operations for Contributions.....	99
12.4.1	install Contribution & update Contribution.....	99
12.4.2	add Deployment Composite & update Deployment Composite.....	100

12.4.3 remove Contribution .....	100
12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts .....	100
12.6 Domain-Level Composite .....	100
12.6.1 add To Domain-Level Composite.....	101
12.6.2 remove From Domain-Level Composite .....	101
12.6.3 get Domain-Level Composite.....	101
12.6.4 get QName Definition .....	101
12.7 Dynamic Behaviour of Wires in the SCA Domain .....	101
12.8 Dynamic Behaviour of Component Property Values .....	102
13 Conformance .....	103
A. XML Schemas .....	104
A.1 sca.xsd .....	104
A.2 sca-core.xsd .....	104
A.3 sca-binding-sca.xsd.....	112
A.4 sca-interface-java.xsd .....	113
A.5 sca-interface-wsdl.xsd.....	113
A.6 sca-implementation-java.xsd.....	114
A.7 sca-implementation-composite.xsd.....	115
A.8 sca-definitions.xsd .....	115
A.9 sca-binding-webservice.xsd .....	116
A.10 sca-binding-jms.xsd.....	116
A.11 sca-policy.xsd .....	116
A.12 sca-contribution.xsd .....	116
B. SCA Concepts .....	119
B.1 Binding.....	119
B.2 Component.....	119
B.3 Service.....	119
B.3.1 Remotable Service.....	119
B.3.2 Local Service.....	120
B.4 Reference .....	120
B.5 Implementation .....	120
B.6 Interface.....	120
B.7 Composite .....	121
B.8 Composite inclusion .....	121
B.9 Property.....	121
B.10 Domain .....	121
B.11 Wire .....	121
C. Conformance Items .....	123
D. Acknowledgements .....	134
E. Non-Normative Text .....	135
F. Revision History.....	136

# 1 Introduction

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

- A model for the assembly of services, both tightly coupled and loosely coupled
- A model for applying infrastructure capabilities to services and to service interactions, including Security and Transactions

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

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

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

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

## 1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

## 1.2 Normative References

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [SCA-Java] SCA Java Component Implementation Specification
- [SCA-Common-Java] 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)
- [SCA BPEL] SCA BPEL Client and Implementation Specification
- <http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec-cd-01.pdf>
- [SDO] SDO Specification
- <http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf>
- [3] SCA Example Code document
- [http://www.osoa.org/download/attachments/28/SCA\\_BuildingYourFirstApplication\\_V09.pdf](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

Formatted: French (France)

Formatted: French (France)

Field Code Changed

Formatted: French (France)

Field Code Changed

Formatted: French (France)

Formatted: French (France)

Field Code Changed

Formatted: French (France)

Formatted: French (France)

Deleted: 2

40 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>  
41  
42 [7] Business Process Execution Language (BPEL)  
43 [http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel)  
44  
45 [8] WSDL Specification  
46 WSDL 1.1: <http://www.w3.org/TR/wsdl>  
47 WSDL 2.0: <http://www.w3.org/TR/wsdl20/>  
48  
49 [9] SCA Web Services Binding Specification  
50 [http://www.osoa.org/download/attachments/35/SCA\\_WebServiceBindings\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_WebServiceBindings_V100.pdf)  
51  
52 [10] SCA Policy Framework Specification  
53 [http://www.osoa.org/download/attachments/35/SCA\\_Policy\\_Framework\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf)  
54  
55 [11] SCA JMS Binding Specification  
56 [http://www.osoa.org/download/attachments/35/SCA\\_JMSBinding\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JMSBinding_V100.pdf)  
57  
58 [12] ZIP Format Definition  
59 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>  
60  
61 [13] Infoset Specification  
62 <http://www.w3.org/TR/xml-infoset/>  
63  
64 [\[WSDL11 Identifiers\] WSDL 1.1 Element Identifiers](#)  
65 <http://www.w3.org/TR/wsdl11elementidentifiers/>  
66

## 67 1.3 Naming Conventions

68  
69 This specification follows some naming conventions for artifacts defined by the specification,  
70 as follows:  
71

- 72 • For the names of elements and the names of attributes within XSD files, the names follow the  
73 CamelCase convention, with all names starting with a lower case letter.  
74 eg <element name="componentType" type="sca:ComponentType"/>
- 75 • For the names of types within XSD files, the names follow the CamelCase convention with all  
76 names starting with an upper case letter.  
77 eg. <complexType name="ComponentService">
- 78 • For the names of intents, the names follow the CamelCase convention, with all names starting  
79 with a lower case letter, EXCEPT for cases where the intent represents an established acronym,  
80 in which case the entire name is in upper case.  
81 An example of an intent which is an acronym is the "SOAP" intent.

---

## 82 2 Overview

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

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

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

99 One basic artifact of SCA is the **component**, which is the unit of construction for SCA. A component  
100 consists of a configured instance of an implementation, where an implementation is the piece of program  
101 code providing business functions. The business function is offered for use by other components as  
102 **services**. Implementations can depend on services provided by other components – these dependencies  
103 are called **references**. Implementations can have settable **properties**, which are data values which  
104 influence the operation of the business function. The component **configures** the implementation by  
105 providing values for the properties and by wiring the references to services provided by other  
106 components.

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

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

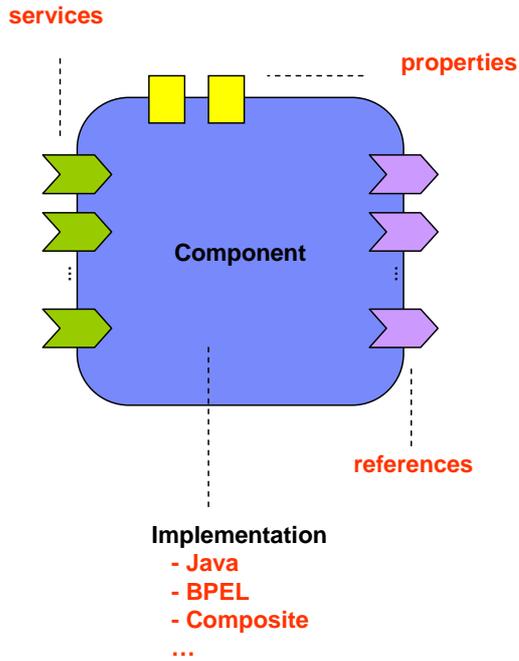
120 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services  
121 providing an area of business functionality that is controlled by a single organization. As an example, for  
122 the accounts department in a business, the SCA Domain might cover all financial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

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

132 **2.1 Diagram used to Represent SCA Artifacts**

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

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



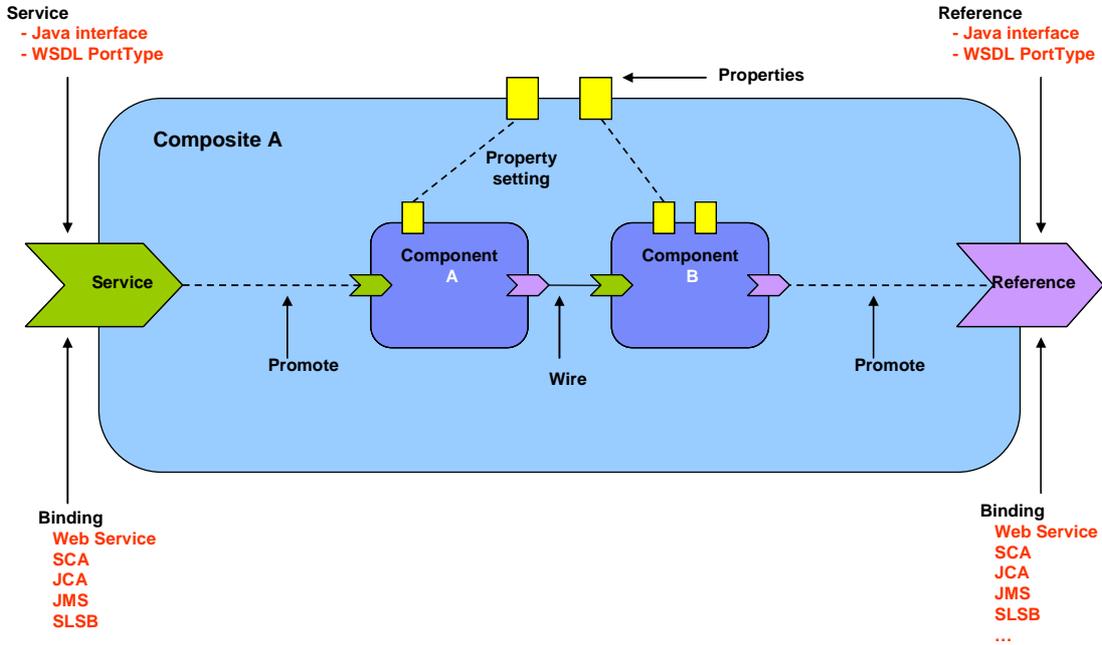
137

138 *Figure 1: SCA Component Diagram*

139

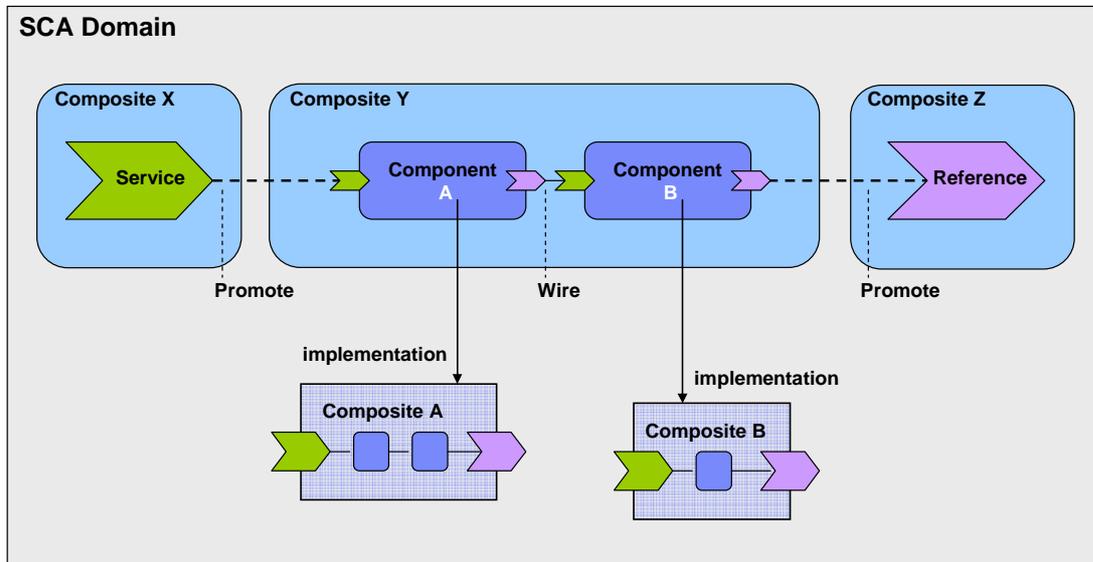
140 The following picture illustrates some of the features of a composite assembled using a set of  
141 components:

142



143  
144 Figure 2: SCA Composite Diagram

145  
146 The following picture illustrates an SCA Domain assembled from a series of high-level composites, some  
147 of which are in turn implemented by lower-level composites:



148  
149 Figure 3: SCA Domain Diagram

---

150 **3 Quick Tour by Sample**

151 To be completed.

152

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

154

155

---

## 156 4 Implementation and ComponentType

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

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

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

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

170 So, for example:

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

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

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

185

### 186 4.1 Component Type

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

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

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

201 The implementation type specification defines whether introspection is allowed, whether a side file  
202 is allowed, both are allowed or some other mechanism specifies the component type. The  
203 component type information derived through introspection is called the **introspected component**  
204 **type**. In any case, the implementation type specification specifies how multiple sources of  
205 information are combined to produce the **effective component type**. The effective component  
206 type is the component type metadata that is presented to the using Component for configuration.

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

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

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

215 The following snippet shows the componentType pseudo-schema.

```
216  
217 <?xml version="1.0" encoding="ASCII"?>  
218 <!-- Component type schema snippet -->  
219 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
220     constrainingType="QName"? >  
221  
222     <service ... />*  
223     <reference ... />*  
224     <property ... />*  
225     <implementation ... />?  
226  
227 </componentType>  
228
```

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

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

235  
236 The **componentType** element has the following **child elements**:

- 237 • **service : Service (0..n)** – see [component type service section](#).
- 238 • **reference : Reference (0..n)** – see [component type reference section](#).
- 239 • **property : Property (0..n)** – see [component type property section](#).
- 240 • **implementation : Implementation (0..1)** – see [component type implementation](#)  
241 [section](#).

## 243 4.1.1 Service

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

248

Comment [ME1]: Issue 36

Comment [mbgl2]: Issue 67

**Deleted: 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. ¶  
The component type is defined by a ... [1]

**Deleted:** .

```

249 <?xml version="1.0" encoding="ASCII"?>
250 <!-- Component type service schema snippet -->
251 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
252 >
253
254     <service name="xs:NCName"
255           requires="list of xs:QName"? policySets="list of xs:QName"?>*
256     <interface ... />
257     <operation name="xs:NCName" requires="list of xs:QName"?
258           policySets="list of xs:QName"?/>*
259     <binding ... />*
260     <callback?
261         <binding ... />+
262     </callback>
263 </service>
264
265 <reference ... />*
266 <property ... />*
267 <implementation ... />?
268
269 </componentType>
270

```

Comment [ME3]: Issue 30

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

- 272 • **name : NCName (1..1)** - the name of the service. The @name attribute of a <service/>  
273 child element of a <componentType/> MUST be unique amongst the service elements of  
274 that <componentType/>. [ASM40003]
- 275 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
276 [10] for a description of this attribute.
- 277 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
278 [10] for a description of this attribute.

279

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

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

Comment [ME4]: Issue 30

296

## 297 4.1.2 Reference

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

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

```
303  
304 <?xml version="1.0" encoding="ASCII"?>  
305 <!-- Component type reference schema snippet -->  
306 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...  
307 >  
308  
309     <service ... />*  
310  
311     <reference name="xs:NCName"  
312         autowire="xs:boolean"?  
313         multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
314         wiredByImpl="xs:boolean"?  
315         requires="list of xs:QName"? policySets="list of xs:QName"?>*  
316     <interface ... />  
317     <operation name="xs:NCName" requires="list of xs:QName"?  
318         policySets="list of xs:QName"?/>*  
319     <binding ... />*  
320     <callback?>  
321         <binding ... />+  
322     </callback>  
323 </reference>  
324  
325     <property ... />*  
326     <implementation ... />?  
327  
328 </componentType>  
329
```

Comment [ME5]: Issue 5

Comment [ME6]: Issue 30

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

- 331 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a  
332 <reference/> child element of a <componentType/> MUST be unique amongst the  
333 reference elements of that <componentType/>. [ASM40004]
  - 334 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect  
335 the reference to target services. The multiplicity can have the following values
    - 336 ○ 0..1 – zero or one wire can have the reference as a source
    - 337 ○ 1..1 – one wire can have the reference as a source
    - 338 ○ 0..n - zero or more wires can have the reference as a source
    - 339 ○ 1..n – one or more wires can have the reference as a source
- 340 If @multiplicity is not specified, the default value is "1..1".
- 341 • **autowire : boolean (0..1)** - whether the reference should be autowired, as described in  
342 the [Autowire section](#). Default is false. Comment [ME7]: Issue 5
  - 343 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the  
344 reference is wired to the target(s) configured on the reference. If set to "true" it indicates  
345 that the target of the reference is set at runtime by the implementation code (eg by the  
346 code obtaining an endpoint reference by some means and setting this as the target of the  
347 reference through the use of programming interfaces defined by the relevant Client and  
348 Implementation specification). If @wiredByImpl is set to "true", then any reference  
349 targets configured for this reference MUST be ignored by the runtime. [ASM40006] It is  
350 recommended that any references with @wiredByImpl = "true" are left unwired.

- 351 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
352 [\[10\]](#) for a description of this attribute.
- 353 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
354 [\[10\]](#) for a description of this attribute.

355

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

- 357 • **interface : Interface (1..1)** - A reference has **one interface**, which describes the  
358 operations required by the reference. The interface is described by an **interface element**  
359 which is a child element of the reference element. For details on the interface element see  
360 [the Interface section](#).

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

Comment [ME8]: Issue 30

- 364 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
365 children. Details of the binding element are described in the [Bindings section](#).

366 Note that a binding element may specify an endpoint which is the target of that binding. A  
367 reference must not mix the use of endpoints specified via binding elements with target  
368 endpoints specified via the target attribute. If the target attribute is set, then binding  
369 elements can only list one or more binding types that can be used for the wires identified  
370 by the target attribute. All the binding types identified are available for use on each wire  
371 in this case. If endpoints are specified in the binding elements, each endpoint must use  
372 the binding type of the binding element in which it is defined. In addition, each binding  
373 element needs to specify an endpoint in this case.

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

380

### 381 4.1.3 Property

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

386

```
387 <?xml version="1.0" encoding="ASCII"?>
388 <!-- Component type property schema snippet -->
389 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
390 >
391
392 <service ... />*
393 <reference ... >*
394
395 <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
396     many="xs:boolean"? mustSupply="xs:boolean"?
397     requires="list of xs:QName"?
398     policySets="list of xs:QName"?>*
399     default-property-value?
400 </property>
401
```

402 <implementation ... />?  
403  
404 </componentType>  
405

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

- 407 ▪ **name : NCName (1..1)** - the name of the property. The @name attribute of a  
408 <property/> child element of a <componentType/> MUST be unique amongst the  
409 property elements of that <componentType/>. [ASM40005]
- 410 ▪ one of (1..1):
  - 411 ○ **type : QName** - the type of the property defined as the qualified name of an XML  
412 schema type. The value of the property @type attribute MUST be the QName of  
413 an XML schema type. [ASM40007]
  - 414 ○ **element : QName** - the type of the property defined as the qualified name of an  
415 XML schema global element – the type is the type of the global element. The value  
416 of the property @element attribute MUST be the QName of an XSD global  
417 element. [ASM40008]
- 418 ▪ **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-  
419 valued (true). In the case of a multi-valued property, it is presented to the implementation  
420 as a collection of property values. If many is not specified, it takes a default value of false. Comment [ME9]: Issue 62
- 421 ▪ **mustSupply : boolean (0..1)** - whether the property value must be supplied by the  
422 component that uses the implementation – when mustSupply="true" the component must  
423 supply a value since the implementation has no default value for the property. A default-  
424 property-value should only be supplied when mustSupply="false" (the default setting for  
425 the mustSupply attribute), since the implication of a default value is that it is used only  
426 when a value is not supplied by the using component. If mustSupply is not specified, it  
427 takes a default value of false. Comment [ME10]: Issue 62
- 428 ▪ **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the `property`. Comment [mbgl11]: Issue 68
- 429 ▪ **requires : QName (0..n)** - a list of policy intents. See the Policy Framework specification  
430 [10] for a description of this attribute.
- 431 ▪ **policySets : QName (0..n)** - a list of policy sets. See the Policy Framework specification  
432 [10] for a description of this attribute. Comment [ME12]: Issue 45

433 The property element can contain a default property value as its content. The form of the default  
434 `property` value is as described in the section on Component Property. Comment [ME13]: Issue 14

435 The value for a property is supplied to the implementation of a component at the time that the  
436 implementation is started. The implementation can choose to use the supplied value in any way  
437 that it chooses. In particular, the implementation can alter the internal value of the property at  
438 any time. However, if the implementation queries the SCA system for the value of the property,  
439 the value as defined in the SCA composite is the value returned.

440 The componentType property element can contain an SCA default value for the property declared  
441 by the implementation. However, the implementation can have a property which has an  
442 implementation defined default value, where the default value is not represented in the  
443 componentType. An example of such a default value is where the default value is computed at  
444 runtime by some code contained in the implementation. If a using component needs to control the  
445 value of a property used by an implementation, the component sets the value explicitly. The SCA  
446 runtime MUST ensure that any implementation default property value is replaced by a value for  
447 that property explicitly set by a component using that implementation. [ASM40009] Comment [mbgl14]: Issue 38

448

#### 449 4.1.4 Implementation

450 **Implementation** represents characteristics inherent to the implementation itself, in particular  
451 intents and policies. See the Policy Framework specification [10] for a description of intents and

452 policies. The following snippet shows the component type schema with the schema for a  
453 implementation child element:

```
454  
455 <?xml version="1.0" encoding="ASCII"?>  
456 <!-- Component type implementation schema snippet -->  
457 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...  
458 >  
459     <service ... /*>  
460     <reference ... /*>  
461     <property ... /*>  
462  
463     <implementation requires="list of xs:QName"?  
464         _____policySets="list of xs:QName"?/*>  
465  
466 </componentType>
```

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

- 470 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
471 [\[10\]](#) for a description of this attribute.
- 472 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
473 [\[10\]](#) for a description of this attribute.

474

## 475 4.2 Example ComponentType

476

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

480

```
481 <?xml version="1.0" encoding="ASCII"?>  
482 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
483  
484     <service name="MyValueService">  
485         <interface.java interface="services.myvalue.MyValueService" /*>  
486     </service>  
487  
488     <reference name="customerService">  
489         <interface.java interface="services.customer.CustomerService" /*>  
490     </reference>  
491     <reference name="stockQuoteService">  
492         <interface.java  
493             interface="services.stockquote.StockQuoteService" /*>  
494     </reference>  
495  
496     <property name="currency" type="xsd:string">USD</property>  
497  
498 </componentType>  
499
```

### 500 4.3 Example Implementation

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

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

```
505  
506 package services.account;  
507  
508 @Remotable  
509 public interface AccountService {  
510  
511     AccountReport getAccountReport(String customerID);  
512 }  
513
```

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

```
517  
518 package services.account;  
519  
520 import java.util.List;  
521  
522 import commonj.sdo.DataFactory;  
523  
524 import org.osoa.sca.annotations.Property;  
525 import org.osoa.sca.annotations.Reference;  
526  
527 import services.accountdata.AccountDataService;  
528 import services.accountdata.CheckingAccount;  
529 import services.accountdata.SavingsAccount;  
530 import services.accountdata.StockAccount;  
531 import services.stockquote.StockQuoteService;  
532  
533 public class AccountServiceImpl implements AccountService {  
534  
535     @Property  
536     private String currency = "USD";  
537  
538     @Reference  
539     private AccountDataService accountDataService;  
540     @Reference  
541     private StockQuoteService stockQuoteService;  
542  
543     public AccountReport getAccountReport(String customerID) {  
544  
545         DataFactory dataFactory = DataFactory.INSTANCE;  
546         AccountReport accountReport = (AccountReport)dataFactory.create(AccountReport.class);  
547         List accountSummaries = accountReport.getAccountSummaries();  
548
```

```

549     CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
550     AccountSummary checkingAccountSummary =
551 (AccountSummary)dataFactory.create(AccountSummary.class);
552     checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
553     checkingAccountSummary.setAccountType("checking");
554     checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
555     accountSummaries.add(checkingAccountSummary);
556
557     SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
558     AccountSummary savingsAccountSummary =
559 (AccountSummary)dataFactory.create(AccountSummary.class);
560     savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
561     savingsAccountSummary.setAccountType("savings");
562     savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
563     accountSummaries.add(savingsAccountSummary);
564
565     StockAccount stockAccount = accountDataService.getStockAccount(customerID);
566     AccountSummary stockAccountSummary =
567 (AccountSummary)dataFactory.create(AccountSummary.class);
568     stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
569     stockAccountSummary.setAccountType("stock");
570     float balance=
571 (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
572     stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
573     accountSummaries.add(stockAccountSummary);
574
575     return accountReport;
576 }
577
578 private float fromUSDollarToCurrency(float value){
579
580     if (currency.equals("USD")) return value; else
581     if (currency.equals("EURO")) return value * 0.8f; else
582     return 0.0f;
583 }
584 }

```

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

```

588
589 <?xml version="1.0" encoding="ASCII"?>
590 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
591               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
592
593     <service name="AccountService">
594         <interface.java interface="services.account.AccountService"/>
595     </service>
596     <reference name="accountDataService">
597         <interface.java
598 interface="services.accountdata.AccountDataService"/>

```

```
599     </reference>
600     <reference name="stockQuoteService">
601         <interface.java
602 interface="services.stockquote.StockQuoteService"/>
603     </reference>
604
605     <property name="currency" type="xsd:string">USD</property>
606
607 </componentType>
```

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

## 612 5 Component

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

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

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

```
622  
623 <?xml version="1.0" encoding="UTF-8"?>  
624 <!-- Component schema snippet -->  
625 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
626 ...  
627 <component name="xs:NCName" autowire="xs:boolean"?  
628 <requires="list of xs:QName"? policySets="list of xs:QName"?  
629 <constrainingType="xs:QName"?>*<br>  
630 <implementation ... />?<br>  
631 <service ... />*<br>  
632 <reference ... />*<br>  
633 <property ... />*<br>  
634 </component><br>  
635 ...<br>  
636 </composite><br>  
637
```

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

- 639 • **name : NCName (1..1)** – the name of the component. [The @name attribute of a](#)  
640 [<component/> child element of a <composite/> MUST be unique amongst the component](#)  
641 [elements of that <composite/>. \[ASM50001\]](#)
- 642 • **autowire : boolean (0..1)** – whether contained component references should be  
643 autowired, as described in [the Autowire section](#). Default is false.
- 644 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
645 [\[10\]](#) for a description of this attribute.
- 646 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
647 [\[10\]](#) for a description of this attribute.
- 648 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
649 the set of services, references and properties of the component, plus related intents, is  
650 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
651 for more details.

**Deleted:** The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>. The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/>

652  
653 The **component** element has the following **child elements**:

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

- 656 • **service** : *ComponentService (0..n)* – see component service section.
- 657 • **reference** : *ComponentReference (0..n)* – see component reference section.
- 658 • **property** : *ComponentProperty (0..n)* – see component property section.

659

## 660 5.1 Implementation

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

666

```
667 <?xml version="1.0" encoding="UTF-8"?>
668 <!-- Component Implementation schema snippet -->
669 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
670   ...
671   <component ... > *
672     <implementation ... />?
673     <service ... /> *
674     <reference ... /> *
675     <property ... /> *
676   </component>
677   ...
678 </composite>
679
```

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

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

Comment [mbgl15]: Issue 69 part 1

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

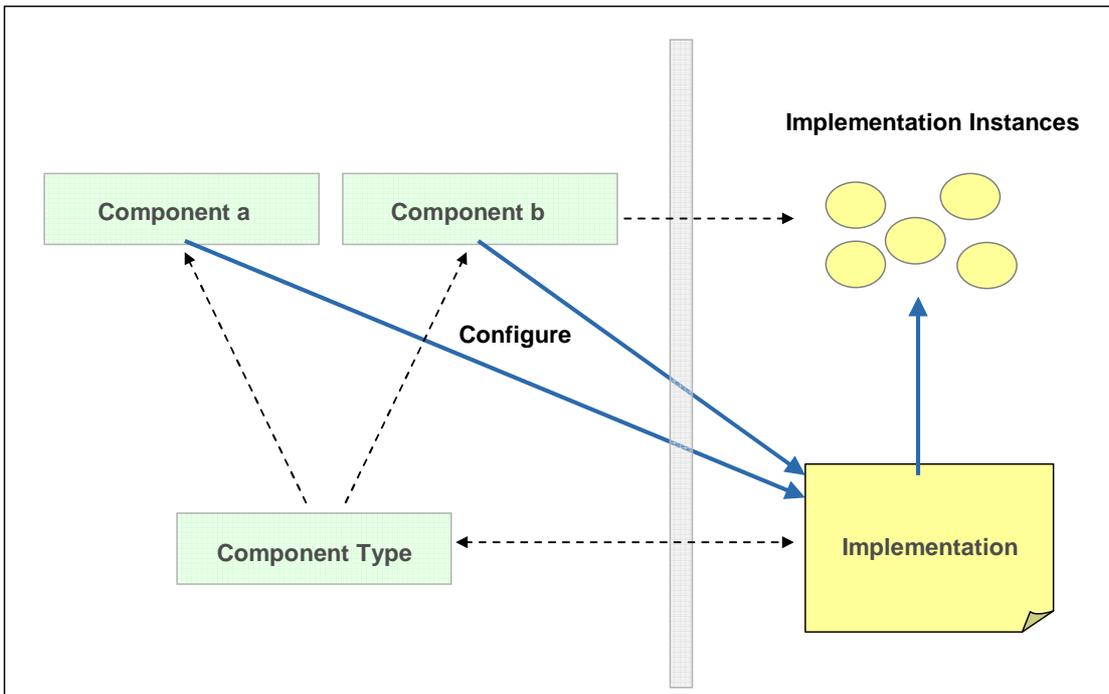
690

```
691 <implementation.java class="services.myvalue.MyValueServiceImpl" />
692
693 <implementation.bpel process="ans:MoneyTransferProcess" />
694
695 <implementation.composite name="bns:MyValueComposite" />
696
697
```

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

699

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



705 Figure 4: Relationship of Component and Implementation

706  
707

## 708 5.2 Service

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

713

```
714 <?xml version="1.0" encoding="UTF-8"?>
715 <!-- Component Service schema snippet -->
716 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
717   ...
718   <component ... >*
719     <implementation ... />?
720     <service name="xs:NCName" requires="list of xs:QName"?
721       __policySets="list of xs:QName"?>*
722     <interface ... />?
723     <operation name="xs:NCName" requires="list of xs:QName"?
724       policySets="list of xs:QName"?/>*
725     <binding ... />*
726     <callback?>
```

Comment [ME16]: Issue 30

```

727         <binding ... />+
728     </callback>
729 </service>
730 <reference ... />*
731 <property ... />*
732 </component>
733 ...
734 </composite>
735

```

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

- 737 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service  
738 element of a <component/> MUST be unique amongst the service elements of that  
739 <component/> [ASM50002] The @name attribute of a service element of a  
740 <component/> MUST match the @name attribute of a service element of the  
741 componentType of the <implementation/> child element of the component. [ASM50003]
- 742 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
743 [10] for a description of this attribute.  
744 Note: The effective set of policy intents for the service consists of any intents explicitly  
745 stated in this requires attribute, combined with any intents specified for the service by the  
746 implementation.
- 747 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
748 [10] for a description of this attribute.

749

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

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

Comment [ME17]: Issue 30

778

## 779 5.3 Reference

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

```
784  
785 <?xml version="1.0" encoding="UTF-8"?>  
786 <!-- Component Reference schema snippet -->  
787 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
788   ...  
789   <component ... >*  
790     <implementation ... />?  
791     <service ... />*  
792     <reference name="xs:NCName"  
793       target="list of xs:anyURI"? autowire="xs:boolean"?  
794       multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
795       wiredByImpl="xs:boolean"? requires="list of xs:QName"?  
796       policySets="list of xs:QName"?>*  
797     <interface ... />?  
798     <operation name="xs:NCName" requires="list of xs:QName"?  
799       policySets="list of xs:QName"?/>*  
800     <binding uri="xs:anyURI"? requires="list of xs:QName"?  
801       policySets="list of xs:QName"?/>*  
802     <callback>?  
803       <binding ... />+  
804     </callback>  
805   </reference>  
806   <property ... />*  
807 </component>  
808   ...  
809 </composite>  
810
```

Comment [ME18]: Issue 30

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

- 812 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service  
813 element of a <component/> MUST be unique amongst the service elements of that  
814 <component/> [ASM50007] [The @name attribute of a reference element of a](#)  
815 [<component/> MUST match the @name attribute of a reference element of the](#)  
816 [componentType of the <implementation/> child element of the component.](#) [ASM50008]
- 817 • **autowire : boolean (0..1)** – whether the reference should be autowired, as described in  
818 [the Autowire section](#). Default is false.
- 819 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
820 [\[10\]](#) for a description of this attribute.  
821 Note: The effective set of policy intents for the reference consists of any intents explicitly  
822 stated in this requires attribute, combined with any intents specified for the reference by  
823 the implementation.
- 824 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
825 [\[10\]](#) for a description of this attribute.
- 826 • **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect  
827 the reference to target services. Overrides the multiplicity specified for this reference in  
828 the componentType of the implementation. The multiplicity can have the following values  
829     o 0..1 – zero or one wire can have the reference as a source  
830     o 1..1 – one wire can have the reference as a source

Deleted: The @name attribute of a reference element of a <component/> MUST match the @name attribute of a reference element of the componentType of the <implementation/> child element of the component.

- 831           o 0..n - zero or more wires can have the reference as a source
- 832           o 1..n - one or more wires can have the reference as a source

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

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

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

- 840           • **target : anyURI (0..n)** – a list of one or more of target service URI’s, depending on  
 841           multiplicity setting. Each value wires the reference to a component service that resolves  
 842           the reference. For more details on wiring see [the section on Wires](#). Overrides any target  
 843           specified for this reference on the implementation.
- 844           • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
 845           the implementation wires this reference dynamically. If set to "true" it indicates that the  
 846           target of the reference is set at runtime by the implementation code (eg by the code  
 847           obtaining an endpoint reference by some means and setting this as the target of the  
 848           reference through the use of programming interfaces defined by the relevant Client and  
 849           Implementation specification). If @wiredByImpl="true" is set for a reference, then the  
 850           reference MUST NOT be wired statically within a composite, but left unwired. [ASM50010]

851  
 852           The **component reference** element has the following **child elements**:

- 853           • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes  
 854           the operations required by the reference. The interface is described by an **interface**  
 855           **element** which is a child element of the reference element. If no interface is specified,  
 856           then the interface specified for the reference in the componentType of the implementation  
 857           is in effect. If an interface is declared for a component reference it MUST provide a  
 858           compatible superset of the interface declared for the equivalent reference in the  
 859           componentType of the implementation, i.e. provide the same operations or a superset of  
 860           the operations defined by the implementation for the reference. [ASM50011] For details  
 861           on the interface element see [the Interface section](#).

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

**Comment [ME19]:** Issue 30

- 865           • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
 866           children. If no binding elements are specified for the reference, then the bindings specified  
 867           for the equivalent reference in the componentType of the implementation MUST be used,  
 868           but if the componentType also has no bindings specified, then <binding.sca/> MUST be  
 869           used as the binding. If binding elements are specified for the reference, then those  
 870           bindings MUST be used and they override any bindings specified for the equivalent  
 871           reference in the componentType of the implementation. [ASM50012] Details of the binding  
 872           element are described in the [Bindings section](#). The binding, combined with any PolicySets  
 873           in effect for the binding, needs to satisfy the set of policy intents for the reference, as  
 874           described in [the Policy Framework specification \[10\]](#).

**Deleted:** If no binding elements are specified for the reference, then the bindings specified for the equivalent reference in the componentType of the implementation MUST be used, but if the componentType also has no bindings specified, then <binding.sca/> MUST be used as the binding. If binding elements are specified for the reference, then those bindings MUST be used and they override any bindings specified for the equivalent reference in the componentType of the implementation.

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

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

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

### 885 5.3.1 Specifying the Target Service(s) for a Reference

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

- 888 1. Through a value specified in the @target attribute of the reference element
- 889 2. Through a target URI specified in the @uri attribute of a binding element which is a child  
890 of the reference element
- 891 3. Through the setting of one or more values for binding-specific attributes and/or child  
892 elements of a binding element that is a child of the reference element
- 893 4. Through the specification of @autowire="true" for the reference (or through inheritance  
894 of that value from the component or composite containing the reference)
- 895 5. Through the specification of @wiredByImpl="true" for the reference
- 896 6. Through the promotion of a component reference by a composite reference of the  
897 composite containing the component (the target service is then identified by the  
898 configuration of the composite reference)
- 899 7. Through the presence of a <wire/> element which has the reference specified in its  
900 @source attribute.

Comment [ME20]: Issue 41

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

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

Comment [ME21]: Issue 41

#### 922 5.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

- 924 • A reference with multiplicity 0..1 or 0..n MAY have no target service defined. [ASM50018]
- 925 • A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service  
926 defined. [ASM50019]
- 927 • A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.  
928 [ASM50020]

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

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

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

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

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

## 953   5.4 Property

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

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

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

965           For example,

966           

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

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

971           For example,

- 972           • property defined using a XML Schema simple type and which contains a single  
973           value

974           

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

- 977           • property defined using a XML Schema simple type and which contains multiple  
978           values

```

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

```

1023 This form allows a specific property of the composite to be addressed by name. Where the  
1024 composite property is of a complex type, the XPath expression can be extended to refer to  
1025 a sub-part of the complex property value.  
1026

1027 So, for example, `source="$currency"` is used to reference a property of the composite  
1028 called "currency", while `source="$currency/a"` references the sub-part "a" of the  
1029 complex composite property with the name "currency".

- 1030 • By specifying a dereferencable URI to a file containing the property value through the **file**  
1031 attribute. The contents of the referenced file are used as the value of the property.

1032

1033 If more than one property value specification is present, the source attribute takes precedence, then  
1034 the file attribute.

1035 For a property defined using a XML Schema simple type and for which a single value is desired, can  
1036 be set either using the @value attribute or the <value> child element. The two forms in such a case  
1037 are equivalent.

1038 When a property has multiple values set, they MUST all be contained within the same property  
1039 element. A <component/> element MUST NOT contain two <property/> subelements with the same  
1040 value of the @name attribute. [ASM50030]

1041 Optionally, the type of the property can be specified in **one** of two ways:

- 1042 • by the qualified name of a type defined in an XML schema, using the **type** attribute
- 1043 • by the qualified name of a global element in an XML schema, using the **element** attribute

1044 The property type specified must be compatible with the type of the property declared in the  
1045 component type of the implementation. If no type is declared in the component property, the type of  
1046 the property declared by the implementation is used.

1047

1048 The following snippet shows the component schema with the schema for a property child element:

1049

```
1050 <?xml version="1.0" encoding="UTF-8"?>
1051 <!-- Component Property schema snippet -->
1052 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1053   ...
1054   <component ... > *
1055     <implementation ... />?
1056     <service ... /> *
1057     <reference ... /> *
1058     <property name="xs:NCName"
1059       (type="xs:QName" | element="xs:QName")?
1060       mustSupply="xs:boolean"? many="xs:boolean"?
1061       source="xs:string"? file="xs:anyURI"?
1062       requires="list of xs:QName"?
1063       policySets="list of xs:QName"?
1064       value="xs:string"? > *
1065       [<value>+ | xs:any+ ]?
1066     </property>
1067   </component>
1068   ...
```

Comment [ME22]: Issue 45

1069 </composite>

1070  
1071

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

- 1072
- 1073 ▪ **name : NCName (1..1)** – the name of the property. The name attribute of a component  
1074 property MUST match the name of a property element in the component type of the  
component implementation. [ASM50031]
  - 1075 ▪ zero or one of **(0..1)**:
    - 1076 ○ **type : QName** – the type of the property defined as the qualified name of an XML  
1077 schema type
    - 1078 ○ **element : QName** – the type of the property defined as the qualified name of an  
1079 XML schema global element – the type is the type of the global element
  - 1080 ▪ **source : string (0..1)** – an XPath expression pointing to a property of the containing  
1081 composite from which the value of this component property is obtained.
  - 1082 ▪ **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
  - 1083 ▪ **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or  
1084 multi-valued (true). Overrides the many specified for this property on the implementation.  
1085 The value can only be equal or further restrict, i.e. if the implementation specifies many  
1086 true, then the component can say false. In the case of a multi-valued property, it is  
1087 presented to the implementation as a Collection of property values. If many is not  
1088 specified, it takes the value defined by the component type of the implementation used by  
1089 the component.
  - 1090 ▪ **value : string (0..1)** - the value of the property if the property is defined using a simple  
1091 type.
  - 1092 ▪ **requires : QName (0..n)** - a list of policy intents. See the Policy Framework specification  
1093 [10] for a description of this attribute.
  - 1094 ▪ **policySets : QName (0..n)** - a list of policy sets. See the Policy Framework specification  
1095 [10] for a description of this attribute.

Comment [ME23]: Issue 62

Comment [ME24]: Issue 62  
- mustSupply removed in  
accordance with the  
resolution

Comment [ME25]: Issue 45

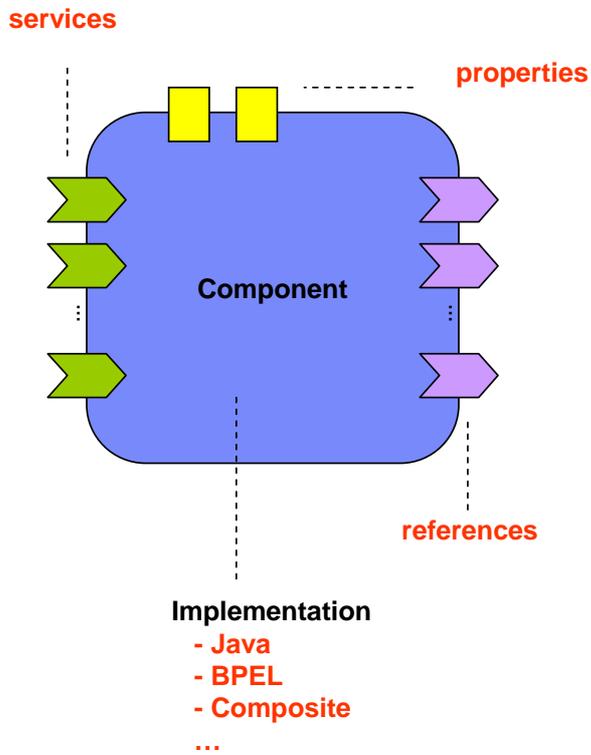
1096 The **component property** element has the following **child element**:

1097 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1098 property that is defined using a XML Schema type. If a property is single-valued, the <value/>  
1099 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST  
1100 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

## 1101 5.5 Example Component

1102

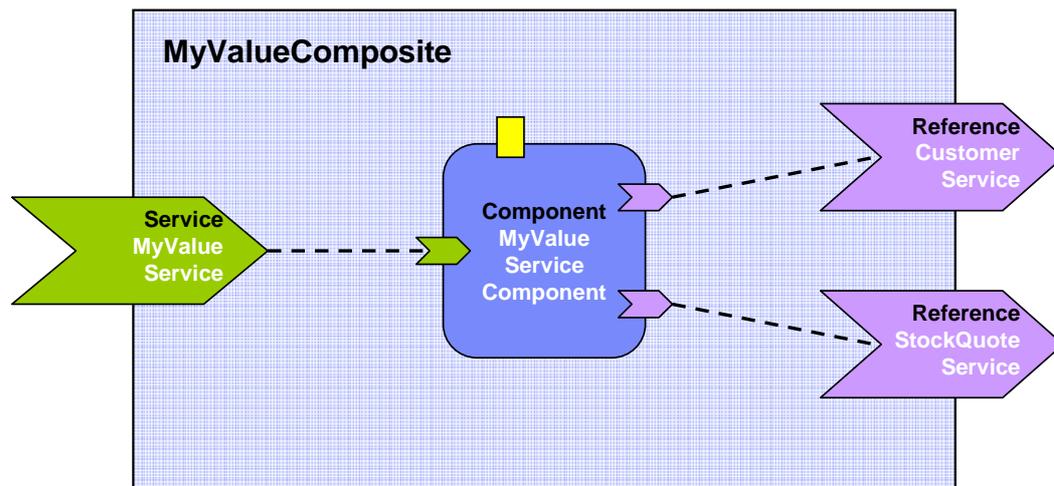
1103 The following figure shows the **component symbol** that is used to represent a component in an  
1104 assembly diagram.



1105  
1106  
1107  
1108  
1109

Figure 5: Component symbol

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



1110  
1111  
1112

Figure 6: Assembly diagram for MyValueComposite

1113

1114 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1115 containing the component element for the MyValueServiceComponent. A value is set for the  
1116 property named currency, and the customerService and stockQuoteService references are  
1117 promoted:

1118

```
1119 <?xml version="1.0" encoding="ASCII"?>
1120 <!-- MyValueComposite_1 example -->
1121 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1122           targetNamespace="http://foo.com"
1123           name="MyValueComposite" >
1124
1125     <service name="MyValueService" promote="MyValueServiceComponent"/>
1126
1127     <component name="MyValueServiceComponent">
1128       <implementation.java
1129 class="services.myvalue.MyValueServiceImpl"/>
1130       <property name="currency">EURO</property>
1131       <reference name="customerService"/>
1132       <reference name="stockQuoteService"/>
1133     </component>
1134
1135     <reference name="CustomerService"
1136       promote="MyValueServiceComponent/customerService"/>
1137
1138     <reference name="StockQuoteService"
1139       promote="MyValueServiceComponent/stockQuoteService"/>
1140
1141 </composite>
```

1142

1143 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of  
1144 clarity – the references are defined by the MyValueServiceImpl implementation and there is no  
1145 need to redeclare them on the component unless the intention is to wire them or to override some  
1146 aspect of them.

1147 The following snippet gives an example of the layout of a composite file if both the currency  
1148 property and the customerService reference of the MyValueServiceComponent are declared to be  
1149 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1150 <?xml version="1.0" encoding="ASCII"?>
1151 <!-- MyValueComposite_2 example -->
1152 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1153           targetNamespace="http://foo.com"
1154           name="MyValueComposite" >
1155
1156     <service name="MyValueService" promote="MyValueServiceComponent"/>
1157
```

```
1158     <component name="MyValueServiceComponent">
1159         <implementation.java
1160 class="services.myvalue.MyValueServiceImpl"/>
1161         <property name="currency">EURO</property>
1162         <property name="currency">Yen</property>
1163         <property name="currency">USDollar</property>
1164         <reference name="customerService"
1165             target="InternalCustomer/customerService"/>
1166         <reference name="StockQuoteService"/>
1167     </component>
1168
1169     ...
1170
1171     <reference name="CustomerService"
1172         promote="MyValueServiceComponent/customerService"/>
1173
1174     <reference name="StockQuoteService"
1175         promote="MyValueServiceComponent/StockQuoteService"/>
1176
1177 </composite>
```

1178 ....this assumes that the composite has another component called InternalCustomer (not shown)  
1179 which has a service to which the customerService reference of the MyValueServiceComponent is  
1180 wired as well as being promoted externally through the composite reference CustomerService.  
1181

## 1182 6 Composite

1183 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of  
1184 composition within an SCA Domain. An **SCA composite** contains a set of components, services,  
1185 references and the wires that interconnect them, plus a set of properties which can be used to  
1186 configure components.

1187 Composites can be used as **component implementations** in higher-level composites – in other  
1188 words the higher-level composites can have components that are implemented by composites.  
1189 For more detail on the use of composites as component implementations see the section [Using  
1190 Composites as Component Implementations](#).

1191 The content of a composite can be used within another composite through **inclusion**. When a  
1192 composite is included by another composite, all of its contents are made available for use within  
1193 the including composite – the contents are fully visible and can be referenced by other elements  
1194 within the including composite. For more detail on the inclusion of one composite into another see  
1195 the section [Using Composites through Inclusion](#).

1196 A composite can be used as a unit of deployment. When used in this way, composites contribute  
1197 elements to an SCA domain. A composite can be deployed to the SCA domain either by inclusion,  
1198 or a composite can be deployed to the domain as an implementation. For more detail on the  
1199 deployment of composites, see the section dealing with the [SCA Domain](#).

1200

1201 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite**  
1202 element. The following snippet shows the schema for the composite element.

1203

```
1204 <?xml version="1.0" encoding="ASCII"?>  
1205 <!-- Composite schema snippet -->  
1206 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
1207     targetNamespace="xs:anyURI"  
1208     name="xs:NCName" local="xs:boolean"?  
1209     autowire="xs:boolean"? constrainingType="QName"?  
1210     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1211  
1212     <include ... />*  
1213  
1214     <service ... />*  
1215     <reference ... />*  
1216     <property ... />*  
1217  
1218     <component ... />*  
1219  
1220     <wire ... />*  
1221  
1222 </composite>
```

1223

1224 The **composite** element has the following **attributes**:

- 1225 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is  
1226 an XML QName, in the namespace identified by the targetNamespace attribute. A  
1227 composite name must be unique within the namespace of the composite. [\[ASM60001\]](#)
- 1228 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the  
1229 composite is declared
- 1230 • **local : boolean (0..1)** – whether all the components within the composite all run in the  
1231 same operating system process. @local="true" for a composite means that all the

- 1232 components within the composite MUST run in the same operating system process.  
 1233 [\[ASM60002\]](#) local="false", which is the default, means that different components within  
 1234 the composite can run in different operating system processes and they can even run on  
 1235 different nodes on a network.
- 1236 • **autowire : boolean (0..1)** – whether contained component references should be  
 1237 autowired, as described in [the Autowire section](#). Default is false.
  - 1238 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
 1239 the set of services, references and properties of the composite, plus related intents, is  
 1240 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
 1241 for more details.
  - 1242 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)  
 1243 [specification \[10\]](#) for a description of this attribute.
  - 1244 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
 1245 [\[10\]](#) for a description of this attribute.

1246

1247 The **composite** element has the following **child elements**:

- 1248 • **service : CompositeService (0..n)** – see composite service section.
- 1249 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1250 • **property : CompositeProperty (0..n)** – see composite property section.
- 1251 • **component : Component (0..n)** – see component section.
- 1252 • **wire : Wire (0..n)** – see composite wire section.
- 1253 • **include : Include (0..n)** – see composite include section

1254

1255 Components contain configured implementations which hold the business logic of the composite.  
 1256 The components offer services and require references to other services. **Composite services**  
 1257 define the public services provided by the composite, which can be accessed from outside the  
 1258 composite. **Composite references** represent dependencies which the composite has on services  
 1259 provided elsewhere, outside the composite. Wires describe the connections between component  
 1260 services and component references within the composite. Included composites contribute the  
 1261 elements they contain to the using composite.

1262 Composite services involve the **promotion** of one service of one of the components within the  
 1263 composite, which means that the composite service is actually provided by one of the components  
 1264 within the composite. Composite references involve the **promotion** of one or more references of  
 1265 one or more components. Multiple component references can be promoted to the same composite  
 1266 reference, as long as all the component references are compatible with one another. Where  
 1267 multiple component references are promoted to the same composite reference, then they all share  
 1268 the same configuration, including the same target service(s).

1269 Composite services and composite references can use the configuration of their promoted services  
 1270 and references respectively (such as Bindings and Policy Sets). Alternatively composite services  
 1271 and composite references can override some or all of the configuration of the promoted services  
 1272 and references, through the configuration of bindings and other aspects of the composite service  
 1273 or reference.

1274 Component services and component references can be promoted to composite services and  
 1275 references and also be wired internally within the composite at the same time. For a reference,  
 1276 this only makes sense if the reference supports a multiplicity greater than 1.

1277

## 1278 6.1 Service

1279 The **services of a composite** are defined by promoting services defined by components  
1280 contained in the composite. A component service is promoted by means of a composite **service**  
1281 **element**.

1282 A composite service is represented by a **service element** which is a child of the composite  
1283 element. There can be **zero or more** service elements in a composite. The following snippet  
1284 shows the **pseudo-schema** for a **service** child element:

Deleted: composite schema with the

```
1285  
1286 <?xml version="1.0" encoding="ASCII"?>  
1287 <!-- Composite Service schema snippet -->  
1288 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1289 ...  
1290 <service name="xs:NCName" promote="xs:anyURI"  
1291 <!-- requires="list of xs:QName"? policySets="list of xs:QName"?*>  
1292 <interface ... />?  
1293 <operation name="xs:NCName" requires="list of xs:QName"?  
1294 <!-- policySets="list of xs:QName"?/>*>  
1295 <binding ... />*>  
1296 <callback?>  
1297 <!-- binding ... />+>  
1298 </callback>  
1299 </service>  
1300 ...  
1301 </composite>
```

Comment [ME26]: Issue 30

1303 The **composite service** element has the following **attributes**:

- 1304 • **name : NCName (1..1)** – the name of the service. The name of a composite <service/>  
1305 element MUST be unique across all the composite services in the composite. [ASM60003]  
1306 The name of the composite service can be different from the name of the promoted  
1307 component service.
- 1308 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form  
1309 <component-name>/<service-name>. The service name is optional if the target  
1310 component only has one service. The same component service can be promoted by more  
1311 than one composite service. A composite <service/> element's promote attribute MUST  
1312 identify one of the component services within that composite. [ASM60004]
- 1313 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1314 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1315 further qualify the required intents defined by the promoted component service.
- 1316 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1317 [\[10\]](#) for a description of this attribute.

1318

1319 The **composite service** element has the following **child elements**, whatever is not specified is  
1320 defaulted from the promoted component service.

- 1321 • **interface : Interface (0..1)** - **If a composite service interface is specified it must be the**  
1322 **same or a compatible subset of the interface provided by the promoted component**  
1323 **service, i.e. provide a subset of the operations defined by the component service,**  
1324 **[ASM60005]** The interface is described by **zero or one interface element** which is a  
1325 child element of the service element. For details on the interface element see [the Interface](#)  
1326 [section](#).

Deleted: If a composite service **interface** is specified it must be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service.

1327  
1328  
1329

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

Comment [ME27]: Issue 30

1330  
1331  
1332  
1333  
1334  
1335

- **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](#).

1336  
1337  
1338  
1339  
1340

- **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.

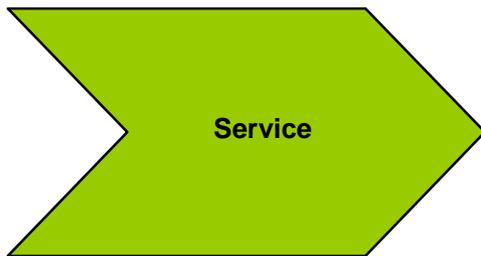
1341

## 1342 6.1.1 Service Examples

1343

1344  
1345

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



1346

1347

Figure 7: Service symbol

1348

1349

1350

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

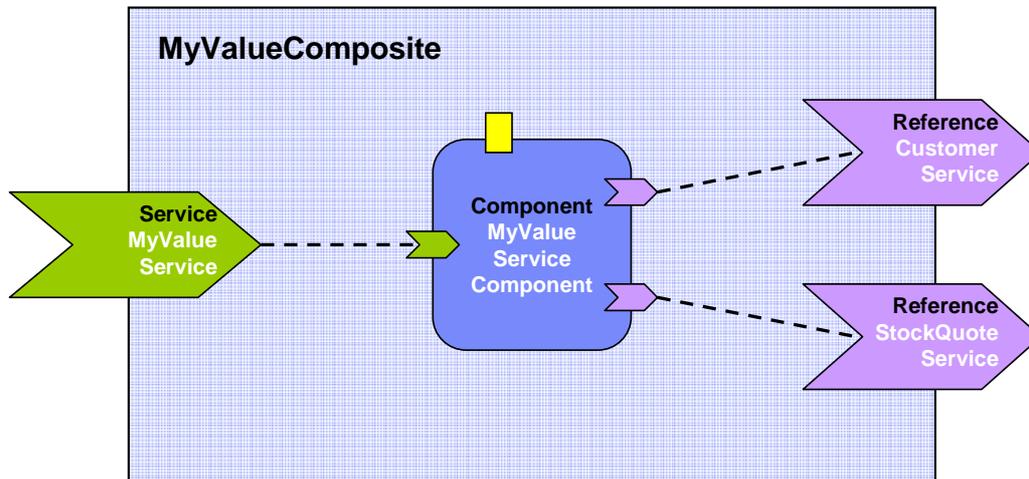


Figure 8: MyValueComposite showing Service

1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381

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

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

1382  
1383       ...  
1384  
1385       </composite>  
1386

## 1387 6.2 Reference

1388       The **references of a composite** are defined by **promoting** references defined by components  
1389       contained in the composite. Each promoted reference indicates that the component reference  
1390       needs to be resolved by services outside the composite. A component reference is promoted using  
1391       a composite **reference element**.

1392       A composite reference is represented by a **reference element** which is a child of a composite  
1393       element. There can be **zero or more** reference elements in a composite. The following snippet  
1394       shows the composite schema with the schema for a **reference** element.

```
1395  
1396 <?xml version="1.0" encoding="ASCII"?>  
1397 <!-- Composite Reference schema snippet -->  
1398 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1399   ...  
1400   <reference name="xs:NCName" target="list of xs:anyURI"?  
1401     promote="list of xs:anyURI" wiredByImpl="xs:boolean"?  
1402     multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
1403     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1404     <interface ... />?  
1405     <operation name="xs:NCName" requires="list of xs:QName"?  
1406       policySets="list of xs:QName"?/>*  
1407     <binding ... />?  
1408     <callback?>  
1409       <binding ... />+  
1410     </callback>  
1411   </reference>  
1412   ...  
1413 </composite>
```

Comment [ME28]: Issue 30

1416       The **composite reference** element has the following **attributes**:

- 1417       • **name : NCName (1..1)** – the name of the reference. The name of a composite  
1418       <reference/> element MUST be unique across all the composite references in the  
1419       composite. [ASM60006] The name of the composite reference can be different then the  
1420       name of the promoted component reference.
- 1421       • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
1422       value is a list of values of the form <component-name>/<reference-name> separated by  
1423       spaces. The specification of the reference name is optional if the component has only one  
1424       reference. Each of the URIs declared by a composite reference's @promote attribute MUST  
1425       identify a component reference within the composite. [ASM60007]

1426       The same component reference can be promoted more than once, using different  
1427       composite references, but only if the multiplicity defined on the component reference is  
1428       0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1429       Where a composite reference promotes two or more component references:

- 1430       • the interfaces of the component references promoted by a composite reference  
1431       MUST be the same, or if the composite reference itself declares an interface then  
1432       all the component reference interfaces must be compatible with the composite

1433 reference interface. Compatible means that the component reference interface is  
1434 the same or is a strict subset of the composite reference interface. [ASM60008]

- 1435 • the intents declared on a composite reference and on the component references  
1436 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents  
1437 which apply to the composite reference in this case are the union of the required  
1438 intents specified for each of the promoted component references plus any intents  
1439 declared on the composite reference itself. If any intents in the set which apply to  
1440 a composite reference are mutually exclusive then the SCA runtime MUST raise an  
1441 error. [ASM60010]
- 1442 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1443 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1444 further qualify the required intents defined for the promoted component reference.
- 1445 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1446 [\[10\]](#) for a description of this attribute.
- 1447 • **multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can  
1448 connect the reference to target services. The multiplicity can have the following values
  - 1449 ○ 0..1 – zero or one wire can have the reference as a source
  - 1450 ○ 1..1 – one wire can have the reference as a source
  - 1451 ○ 0..n - zero or more wires can have the reference as a source
  - 1452 ○ 1..n – one or more wires can have the reference as a source

1453 The value specified for the **multiplicity** attribute of a composite reference MUST be  
1454 compatible with the multiplicity specified on each of the promoted component references,  
1455 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used  
1456 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be  
1457 used where the promoted component reference has multiplicity 0..n or 1..n and  
1458 multiplicity 1..n can be used where the promoted component reference has multiplicity  
1459 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to  
1460 promote a component reference of multiplicity 0..1 or 1..1 respectively. [ASM60011]

- 1461 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
1462 multiplicity setting. Each value wires the reference to a service in a composite that uses  
1463 the composite containing the reference as an implementation for one of its components. For  
1464 more details on wiring see [the section on Wires](#).
- 1465 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
1466 the implementation wires this reference dynamically. If set to "true" it indicates that the  
1467 target of the reference is set at runtime by the implementation code (eg by the code  
1468 obtaining an endpoint reference by some means and setting this as the target of the  
1469 reference through the use of programming interfaces defined by the relevant Client and  
1470 Implementation specification). If "true" is set, then the reference should not be wired  
1471 statically within a using composite, but left **unwired**.

1472  
1473 The **composite reference** element has the following **child elements**, whatever is not specified is  
1474 defaulted from the promoted component reference(s).

- 1475 • **interface : Interface (0..1)** - **zero or one interface element** which declares an  
1476 interface for the composite reference. If a composite reference has an **interface** specified,  
1477 it MUST provide an interface which is the same or which is a compatible superset of the  
1478 interface(s) declared by the promoted component reference(s), i.e. provide a superset of  
1479 the operations in the interface defined by the component for the reference. [ASM60012] If  
1480 no interface is declared on a composite reference, the interface from one of its promoted  
1481 component references is used, which MUST be the same as or a compatible superset of  
1482 the interface(s) declared by the promoted component reference(s).  
1483 [ASM60013] For details on the interface element see [the Interface section](#).

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

**Comment [ME29]:** Need to consider this as a normative statement

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

1484  
1485  
1486

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

Comment [ME30]: Issue 30

1487  
1488  
1489  
1490  
1491  
1492  
1493

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

1494  
1495  
1496

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

1497  
1498  
1499  
1500  
1501

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

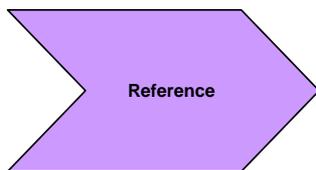
1502

## 1503 6.2.1 Example Reference

1504

1505  
1506

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



1507

1508

Figure 9: Reference symbol

1509

1510

1511

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

1512

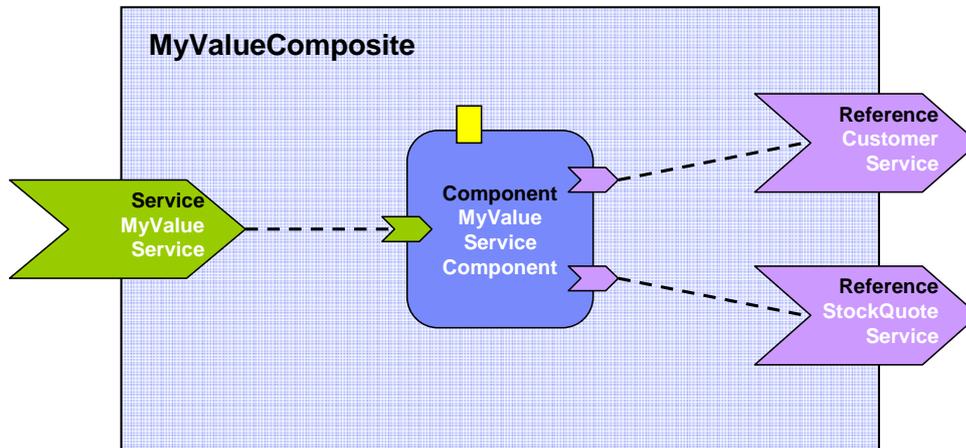


Figure 10: MyValueComposite showing References

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

```

1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
is -->

```

```

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

```

```

1546         <!-- specified by the component reference or by the underlying
1547 -->
1548         <!-- implementation
1549 -->
1550         <binding.sca/>
1551     </reference>
1552
1553     <reference name="StockQuoteService"
1554         promote="MyValueServiceComponent/StockQuoteService">
1555         <interface.java
1556 interface="services.stockquote.StockQuoteService"/>
1557         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1558 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1559     </reference>
1560
1561     ...
1562
1563 </composite>
1564
1565

```

### 1566 6.3 Property

1567 **Properties** allow for the configuration of an implementation with externally set data values. A  
1568 composite can declare zero or more properties. Each property has a type, which may be either  
1569 simple or complex. An implementation can also define a default value for a property. Properties  
1570 can be configured with values in the components that use the implementation.

1571 The declaration of a property in a composite follows the form described in the following schema  
1572 snippet:

```

1573
1574 <?xml version="1.0" encoding="ASCII"?>
1575 <!-- Composite Property schema snippet -->
1576 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1577     ...
1578     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1579         requires="list of xs:QName"?
1580         policySets="list of xs:QName"?
1581         many="xs:boolean"? mustSupply="xs:boolean"?>*
1582         default-property-value?
1583     </property>
1584     ...
1585 </composite>
1586

```

Comment [ME31]: Issue 45

1587 The **composite property** element has the following **attributes**:

- 1588 ▪ **name : NCName (1..1)** - the name of the property. **The name attribute of a composite**  
1589 **property MUST be unique amongst the properties of the same composite.** [ASM60014]

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

- 1590       ▪ one of **(1..1)**:
  - 1591           ○ **type : QName** – the type of the property - the qualified name of an XML schema type
  - 1592
  - 1593           ○ **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
  - 1594
  - 1595       ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued (true). The default is **false**. In the case of a multi-valued property, it is presented to the implementation as a collection of property values.
  - 1596
  - 1597
  - 1598       ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the component that uses the composite – when mustSupply="true" the component has to supply a value since the composite has no default value for the property. A default-property-value is only worth declaring when mustSupply="false" (the default setting for the mustSupply attribute), since the implication of a default value is that it is used only when a value is not supplied by the using component.
  - 1599
  - 1600
  - 1601
  - 1602
  - 1603
  - 1604       ▪ **requires : QName (0..n)** - a list of policy intents. See the Policy Framework specification [10] for a description of this attribute.
  - 1605
  - 1606       ▪ **policySets : QName (0..n)** - a list of policy sets. See the Policy Framework specification [10] for a description of this attribute.
  - 1607

Comment [ME32]: Issue 45

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

Comment [ME33]: Issue 14

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

Comment [ME34]: I think that this paragraph should be removed.

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

### 1618 6.3.1 Property Examples

1619  
1620 For the following example of Property declaration and value setting, the following complex type is used as an example:

```
1622 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1623             targetNamespace="http://foo.com/"
1624             xmlns:tns="http://foo.com/">
1625   <!-- ComplexProperty schema -->
1626   <xsd:element name="fooElement" type="MyComplexType"/>
1627   <xsd:complexType name="MyComplexType">
1628     <xsd:sequence>
1629       <xsd:element name="a" type="xsd:string"/>
1630       <xsd:element name="b" type="anyURI"/>
1631     </xsd:sequence>
1632     <attribute name="attr" type="xsd:string" use="optional"/>
1633   </xsd:complexType>
1634 </xsd:schema>
```

1636 The following composite demonstrates the declaration of a property of a complex type, with a default value, plus it demonstrates the setting of a property value of a complex type within a component:

```

1639 <?xml version="1.0" encoding="ASCII"?>
1640 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1641           xmlns:foo="http://foo.com"
1642           targetNamespace="http://foo.com"
1643           name="AccountServices">
1644 <!-- AccountServices Example1 -->
1645
1646     ...
1647
1648     <property name="complexFoo" type="foo:MyComplexType">
1649       <value>
1650         <foo:a>AValue</foo:a>
1651         <foo:b>InterestingURI</foo:b>
1652       </value>
1653     </property>
1654
1655     <component name="AccountServiceComponent">
1656       <implementation.java class="foo.AccountServiceImpl"/>
1657       <property name="complexBar" source="$complexFoo"/>
1658       <reference name="accountDataService"
1659         target="AccountDataServiceComponent"/>
1660       <reference name="stockQuoteService" target="StockQuoteService"/>
1661     </component>
1662
1663     ...
1664
1665 </composite>
1666

```

Comment [ME35]: Issue 14

1667 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the  
1668 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the  
1669 composite and it references the example XSD, where **MyComplexType** is defined. The declaration  
1670 of **complexFoo** contains a default value. This is declared as the content of the property element.  
1671 In this example, the default value consists of the element **value** which is required to be of type  
1672 **foo:MyComplexType** and its two child elements **<foo:a>** and **<foo:b>**, following the definition of  
1673 **MyComplexType**.

Comment [ME36]: Issue 14

1674 In the component **AccountServiceComponent**, the component sets the value of the property  
1675 **complexBar**, declared by the implementation configured by the component. In this case, the  
1676 type of **complexBar** is **foo:MyComplexType**. The example shows that the value of the **complexBar**  
1677 property is set from the value of the **complexFoo** property – the **source** attribute of the property  
1678 element for **complexBar** declares that the value of the property is set from the value of a property  
1679 of the containing composite. The value of the source attribute is **\$complexFoo**, where  
1680 **complexFoo** is the name of a property of the composite. This value implies that the whole of the  
1681 value of the source property is used to set the value of the component property.

1682 The following example illustrates the setting of the value of a property of a simple type (a string)  
1683 from **part** of the value of a property of the containing composite which has a complex type:

```

1684 <?xml version="1.0" encoding="ASCII"?>
1685 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1686           xmlns:foo="http://foo.com"
1687           targetNamespace="http://foo.com"
1688           name="AccountServices">
1689 <!-- AccountServices Example2 -->
1690
1691     ...
1692
1693     <property name="complexFoo" type="foo:MyComplexType">
1694       <value>
1695         <foo:a>AValue</foo:a>

```

Comment [ME37]: Issue 14

```

1696         <foo:b>InterestingURI</foo:b>
1697     </value>
1698 </property>
1699
1700 <component name="AccountServiceComponent">
1701     <implementation.java class="foo.AccountServiceImpl"/>
1702     <property name="currency" source="$complexFoo/a"/>
1703     <reference name="accountDataService"
1704         target="AccountDataServiceComponent"/>
1705     <reference name="stockQuoteService" target="StockQuoteService"/>
1706 </component>
1707
1708     ...
1709
1710 </composite>
1711

```

1712 In this example, the component **AccountServiceComponent** sets the value of a property called  
1713 **currency**, which is of type string. The value is set from a property of the composite  
1714 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression  
1715 that selects the property name **complexFoo** and then selects the value of the **a** subelement of  
1716 the value of complexFoo. The "a" subelement is a string, matching the type of the currency  
1717 property.

1718 Further examples of declaring properties and setting property values in a component follow:

1719 Declaration of a property with a simple type and a default value:

```

1720 <property name="SimpleTypeProperty" type="xsd:string">
1721 MyValue
1722 </property>

```

1723

1724 Declaration of a property with a complex type and a default value:

```

1725 <property name="complexFoo" type="foo:MyComplexType">
1726 <value>
1727 <foo:a>AValue</foo:a>
1728 <foo:b>InterestingURI</foo:b>
1729 </value>
1730 </property>
1731

```

Comment [ME38]: Issue 14

1732 Declaration of a property with a global element type:

```

1733 <property name="elementFoo" element="foo:fooElement">
1734 <foo:fooElement>
1735 <foo:a>AValue</foo:a>
1736 <foo:b>InterestingURI</foo:b>
1737 </foo:fooElement>
1738 </property>
1739

```

Comment [ME39]: Issue 14  
(see section 5.4 for this material)

## 1740 6.4 Wire

1741 **SCA wires** within a composite connect **source component references** to **target component**  
1742 **services**.

1743 One way of defining a wire is by **configuring a reference of a component using its target**  
1744 **attribute**. The reference element is configured with the wire-target-URI of the service(s) that  
1745 resolve the reference. Multiple target services are valid when the reference has a multiplicity of  
1746 0..n or 1..n.

1747 An alternative way of defining a Wire is by means of a **wire element** which is a child of the  
1748 composite element. There can be **zero or more** wire elements in a composite. This alternative  
1749 method for defining wires is useful in circumstances where separation of the wiring from the  
1750 elements the wires connect helps simplify development or operational activities. An example is  
1751 where the components used to build a domain are relatively static but where new or changed  
1752 applications are created regularly from those components, through the creation of new assemblies  
1753 with different wiring. Deploying the wiring separately from the components allows the wiring to  
1754 be created or modified with minimum effort.

1755 Note that a Wire specified via a wire element is equivalent to a wire specified via the target  
1756 attribute of a reference. The rule which forbids mixing of wires specified with the target attribute  
1757 with the specification of endpoints in binding subelements of the reference also applies to wires  
1758 specified via separate wire elements.

1759 The following snippet shows the composite schema with the schema for the reference elements of  
1760 components and composite services and the wire child element:

1761

```
1762 <?xml version="1.0" encoding="ASCII"?>  
1763 <!-- Wires schema snippet -->  
1764 <composite ...>  
1765   ...  
1766   <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/> *  
1767   ...  
1768 </composite>
```

1770 The **reference element of a component** and the **reference element of a service** has a list of  
1771 one or more of the following **wire-target-URI** values for the target, with multiple values  
1772 separated by a space:

- 1773 • <component-name>/<service-name>
  - 1774 ○ where the target is a service of a component. The specification of the service  
1775 name is optional if the target component only has one service with a compatible  
1776 interface

1777

1778 The **wire element** has the following attributes:

- 1779 • **source (1..1)** – names the source component reference. Valid URI schemes are:
  - 1780 ○ <component-name>/<reference-name>
    - 1781 ▪ where the source is a component reference. The specification of the  
1782 reference name is optional if the source component only has one reference
- 1783 • **target (1..1)** – names the target component service. Valid URI schemes are
  - 1784 ○ <component-name>/<service-name>
    - 1785 ▪ where the target is a service of a component. The specification of the  
1786 service name is optional if the target component only has one service with  
1787 a compatible interface

- 1788 • **replace (0..1)** - a boolean value, with the default of "false". When a wire element has  
1789 @replace="false", the wire is added to the set of wires which apply to the reference  
1790 identified by the @source attribute. When a wire element has @replace="true", the wire  
1791 is added to the set of wires which apply to the reference identified by the @source  
1792 attribute - but any wires for that reference specified by means of the @target attribute of  
1793 the reference are removed from the set of wires which apply to the reference.

1794  
1795 In other words, if any <wire/> element with @replace="true" is used for a particular  
1796 reference, the value of the @target attribute on the reference is ignored - and this permits

Deleted: ..xmlns="http://d  
ocs.oasis-  
open.org/ns/opencsa/sca  
/200712"¶  
...targetNamespace="xs:  
anyURI" ¶  
...name="xs:NCName"  
local="xs:boolean"?  
autowire="xs:boolean"? ¶  
...constrainingType="QN  
ame"? ¶  
...requires="list of  
xs:QName"?  
policySets="list of  
xs:QName"?

Comment [ME40]: This  
pseudo-schema does not  
match the wording leading  
up to it...

1797  
1798

existing wires on the reference to be overridden by separate configuration, if required, where the reference is on a component at the Domain level.

Comment [ME41]: Issue 41

1799  
1800  
1801  
1802

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

1803  
1804

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

1805  
1806

1. the source interface and the target interface of a wire MUST either both be remotable or else both be local [ASM60015]

1807  
1808

2. 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 [ASM60016]

1809  
1810  
1811

3. 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. [ASM60017]

1812  
1813

4. 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. [ASM60018]

1814  
1815

5. 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. [ASM60019]

1816  
1817

6. other specified attributes of the source interface and the target interface of a wire MUST match, including Scope and Callback interface [ASM60020]

1818  
1819  
1820  
1821

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

1822  
1823  
1824  
1825  
1826  
1827

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

1828  
1829  
1830

**Note:** It is permitted to deploy a composite that has references that are not wired. 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. [ASM60021]

1831

### 6.4.1 Wire Examples

Deleted: <#>¶

Formatted: Bullets and Numbering

1833

1834  
1835

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

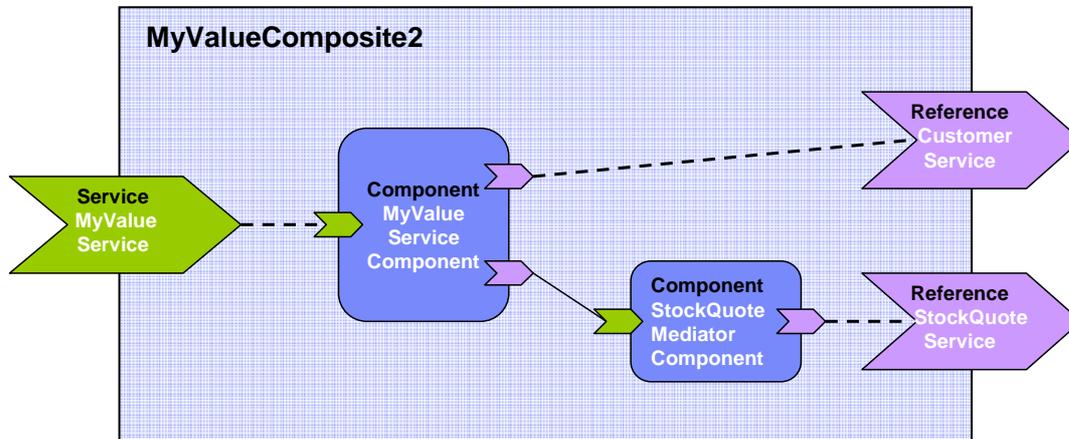


Figure 11: MyValueComposite2 showing Wires

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

```

1847 <?xml version="1.0" encoding="ASCII"?>
1848 <!-- MyValueComposite Wires examples -->
1849 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1850           targetNamespace="http://foo.com"
1851           name="MyValueComposite2" >
1852
1853     <service name="MyValueService" promote="MyValueServiceComponent">
1854       <interface.java interface="services.myvalue.MyValueService"/>
1855       <binding.ws port="http://www.myvalue.org/MyValueService#
1856                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1857     </service>
1858
1859     <component name="MyValueServiceComponent">
1860       <implementation.java
1861         class="services.myvalue.MyValueServiceImpl"/>
1862       <property name="currency">EURO</property>
1863       <service name="MyValueService"/>
1864       <reference name="customerService"/>
1865       <reference name="stockQuoteService"/>
1866     </component>

```

```

1868     <wire source="MyValueServiceComponent/stockQuoteService"
1869           target="StockQuoteMediatorComponent" />
1870
1871     <component name="StockQuoteMediatorComponent" >
1872       <implementation.java class="services.myvalue.SQMediatorImpl" />
1873       <property name="currency">EURO</property>
1874       <reference name="stockQuoteService" />
1875     </component>
1876
1877     <reference name="CustomerService"
1878       promote="MyValueServiceComponent/customerService">
1879       <interface.java interface="services.customer.CustomerService" />
1880       <binding.sca/>
1881     </reference>
1882
1883     <reference name="StockQuoteService"
1884       promote="StockQuoteMediatorComponent">
1885       <interface.java
1886         interface="services.stockquote.StockQuoteService" />
1887       <binding.ws port="http://www.stockquote.org/StockQuoteService#
1888         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)" />
1889     </reference>
1890
1891   </composite>
1892

```

## 1893 6.4.2 Autowire

1894 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.  
 1895 Autowire enables component references to be automatically wired to component services which  
 1896 will satisfy those references, without the need to create explicit wires between the references and  
 1897 the services. When the autowire feature is used, a component reference which is not promoted  
 1898 and which is not explicitly wired to a service within a composite is automatically wired to a target  
 1899 service within the same composite. Autowire works by searching within the composite for a  
 1900 service interface which matches the interface of the references.

1901 The autowire feature is not used by default. Autowire is enabled by the setting of an autowire  
 1902 attribute to "true". Autowire is disabled by setting of the autowire attribute to "false" The autowire  
 1903 attribute can be applied to any of the following elements within a composite:

- 1904 • reference
- 1905 • component
- 1906 • composite

1907 Where an element does not have an explicit setting for the autowire attribute, it inherits the  
 1908 setting from its parent element. Thus a reference element inherits the setting from its containing  
 1909 component. A component element inherits the setting from its containing composite. Where  
 1910 there is no setting on any level, autowire="false" is the default.

1911 As an example, if a composite element has autowire="true" set, this means that autowiring is  
 1912 enabled for all component references within that composite. In this example, autowiring can be

1913 turned off for specific components and specific references through setting autowire="false" on the  
1914 components and references concerned.

1915 For each component reference for which autowire is enabled, the the SCA runtime MUST search  
1916 within the composite for target services which are compatible with the reference. [ASM60022]  
1917 "Compatible" here means:

- 1918 • the target service interface MUST be a compatible superset of the reference interface  
1919 when using autowire to wire a reference (as defined in the section on Wires). [ASM60023]
- 1920 • the intents, and policies applied to the service MUST be compatible with those on the  
1921 reference when using autowire to wire a reference – so that wiring the reference to the  
1922 service will not cause an error due to policy mismatch [ASM60024] (see the Policy  
1923 Framework specification [10] for details)

1924 If the search finds **1 or more** valid target service for a particular reference, the action taken  
1925 depends on the multiplicity of the reference:

- 1926 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
1927 reference to one of the set of valid target services chosen from the set in a runtime-  
1928 dependent fashion [ASM60025]
- 1929 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
1930 of the set of valid target services [ASM60026]

1931 If the search finds **no** valid target services for a particular reference, the action taken depends on  
1932 the multiplicity of the reference:

- 1933 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
1934 target service, there is no problem – no services are wired and the SCA runtime MUST  
1935 NOT raise an error [ASM60027]
- 1936 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
1937 target services an error MUST be raised by the SCA runtime since the reference is  
1938 intended to be wired [ASM60028]

1939

### 1940 6.4.3 Autowire Examples

1941 This example demonstrates two versions of the same composite – the first version is done using  
1942 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
1943 the end result is the same – the same wires connect the references to the services.

1944 First, here is a diagram for the composite:

**Formatted:** Default Paragraph Font

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

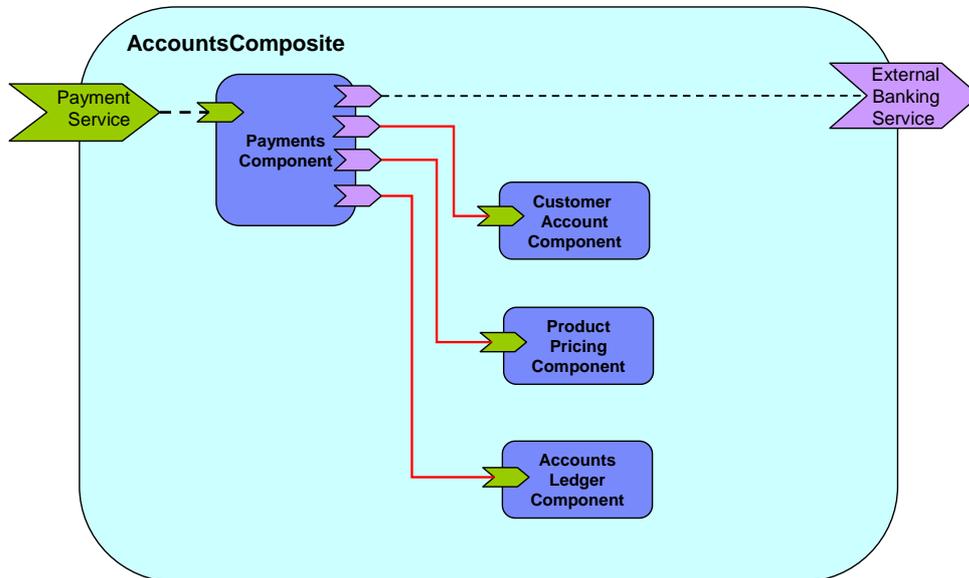


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1948 <?xml version="1.0" encoding="UTF-8"?>
1949 <!-- Autowire Example - No autowire -->
1950 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1951           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1952           xmlns:foo="http://foo.com"
1953           targetNamespace="http://foo.com"
1954           name="AccountComposite">
1955
1956     <service name="PaymentService" promote="PaymentsComponent"/>
1957
1958     <component name="PaymentsComponent">
1959       <implementation.java class="com.foo.accounts.Payments"/>
1960       <service name="PaymentService"/>
1961       <reference name="CustomerAccountService"
1962               target="CustomerAccountComponent"/>
1963       <reference name="ProductPricingService"
1964               target="ProductPricingComponent"/>
1965       <reference name="AccountsLedgerService"
1966               target="AccountsLedgerComponent"/>
1967       <reference name="ExternalBankingService"/>
1968     </component>
1969
1970     <component name="CustomerAccountComponent">
1971       <implementation.java class="com.foo.accounts.CustomerAccount"/>
1972     </component>
1973
1974     <component name="ProductPricingComponent">
1975       <implementation.java class="com.foo.accounts.ProductPricing"/>
1976     </component>
1977
1978     <component name="AccountsLedgerComponent">

```

Formatted: Indent: Left: 0.5 cm, Space Before: 0 pt

```

1979     <implementation.composite name="foo:AccountsLedgerComposite" />
1980 </component>
1981
1982     <reference name="ExternalBankingService"
1983         promote="PaymentsComponent/ExternalBankingService" />
1984
1985 </composite>
1986

```

1987 Secondly, the composite using autowire:

```

1988 <?xml version="1.0" encoding="UTF-8"?>
1989 <!-- Autowire Example - With autowire -->
1990 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1991     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1992     xmlns:foo="http://foo.com"
1993     targetNamespace="http://foo.com"
1994     name="AccountComposite">
1995
1996     <service name="PaymentService" promote="PaymentsComponent">
1997         <interface.java class="com.foo.PaymentServiceInterface" />
1998     </service>
1999
2000     <component name="PaymentsComponent" autowire="true">
2001         <implementation.java class="com.foo.accounts.Payments" />
2002         <service name="PaymentService" />
2003         <reference name="CustomerAccountService" />
2004         <reference name="ProductPricingService" />
2005         <reference name="AccountsLedgerService" />
2006         <reference name="ExternalBankingService" />
2007     </component>
2008
2009     <component name="CustomerAccountComponent">
2010         <implementation.java class="com.foo.accounts.CustomerAccount" />
2011     </component>
2012
2013     <component name="ProductPricingComponent">
2014         <implementation.java class="com.foo.accounts.ProductPricing" />
2015     </component>
2016
2017     <component name="AccountsLedgerComponent">
2018         <implementation.composite name="foo:AccountsLedgerComposite" />
2019     </component>
2020
2021     <reference name="ExternalBankingService"
2022         promote="PaymentsComponent/ExternalBankingService" />
2023
2024 </composite>

```

2025 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires  
2026 for any of its references – the wires are created automatically through autowire.

2027 **Note:** In the second example, it would be possible to omit all of the service and reference  
2028 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the  
2029 component service and references still exist, since they are provided by the implementation used  
2030 by the component.

2031

2032 **6.5 Using Composites as Component Implementations**

2033 Composites may form **component implementations** in higher-level composites – in other words  
2034 the higher-level composites can have components which are implemented by composites.

2035 When a composite is used as a component implementation, it defines a boundary of visibility.  
2036 Components within the composite cannot be referenced directly by the using component. The  
2037 using component can only connect wires to the services and references of the used composite and  
2038 set values for any properties of the composite. The internal construction of the composite is  
2039 invisible to the using component. The boundary of visibility, sometimes called encapsulation, can  
2040 be enforced when assembling components and composites, but such encapsulation structures  
2041 might not be enforceable in a particular implementation language.

Comment [ME42]: Issue 26

2042 A composite used as a component implementation must also honor a completeness contract. The  
2043 services, references and properties of the composite form a contract (represented by the  
2044 component type of the composite) which is relied upon by the using component. The concept of  
2045 completeness of the composite implies that, once all <include/> element processing is performed  
2046 on the composite:

- 2047 1. For a composite used as a component implementation, each composite service offered by  
2048 the composite MUST promote a component service of a component that is within the  
2049 composite. [ASM60032]
- 2050 2. For a composite used as a component implementation, every component reference of  
2051 components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or  
2052 promoted (according to the various rules for specifying target services for a component  
2053 reference described in section 5.3.1). [ASM60033]
- 2054 3. For a composite used as a component implementation, all properties of components within  
2055 the composite, where the underlying component implementation specifies  
2056 "mustSupply=true" for the property, MUST either specify a value for the property or  
2057 source the value from a composite property. [ASM60034]

Formatted: Font color: Red

Comment [ME43]: Issue 51

**Deleted:** A composite used as a component implementation needs to also honor a **completeness contract**. The services, references and properties of the composite form a contract which is relied upon by the using component. The concept of completeness of the composite implies:¶  
 ¶ the composite must have at least one service or at least one reference. .  
 ¶ A component with no services and no references is not meaningful in terms of SCA, since it cannot be wired to anything – it neither provides nor consumes any services .  
 ¶ each service offered by the composite must be wired to a service of a component or to a composite reference. .  
 ¶ If services are left unwired, the implication is that some exception will occur at runtime if the service is invoked.¶  
 ¶ The component type of a composite is defined by the set of service elements, reference elements and property elements that are the children of the composite element.

2058 The component type of a composite is defined by the set of composite service elements,  
2059 composite reference elements and composite property elements that are the children of the  
2060 composite element.

2061 Composites are used as component implementations through the use of the  
2062 **implementation.composite** element as a child element of the component. The schema snippet  
2063 for the implementation.composite element is:

```
2064
2065 <!-- implementation.composite pseudo-schema -->
2066 <implementation.composite name="xs:QName" requires="list of xs:QName"?
2067 policySets="list of xs:QName"?>
```

2068  
2069  
2070 The implementation.composite element has the following attributes:

- 2071 • **name (1..1)** – the name of the composite used as an implementation. The @name  
2072 attribute of an <implementation.composite/> element MUST contain the QName of a  
2073 composite in the SCA Domain. [ASM60030]
- 2074 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
2075 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
2076 further qualify the required intents defined for the promoted component reference.
- 2077 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2078 [\[10\]](#) for a description of this attribute.

Comment [ME44]: Issue 71

Formatted: English (U.S.)

Formatted: English (U.S.)

Comment [ME45]: Issue 71

## 2080 6.5.1 Example of Composite used as a Component Implementation

2081

2082 The following is an example of a composite which contains two components, each of which is  
2083 implemented by a composite:  
2084

```
2085 <?xml version="1.0" encoding="UTF-8"?>
2086 <!-- CompositeComponent example -->
2087 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2088           xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200712
2089           file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
2090           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2091           targetNamespace="http://foo.com"
2092           xmlns:foo="http://foo.com"
2093           name="AccountComposite">
2094
2095     <service name="AccountService" promote="AccountServiceComponent">
2096       <interface.java interface="services.account.AccountService"/>
2097       <binding.ws port="AccountService#
2098         wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
2099     </service>
2100
2101     <reference name="stockQuoteService"
2102       promote="AccountServiceComponent/StockQuoteService">
2103       <interface.java
2104         interface="services.stockquote.StockQuoteService"/>
2105       <binding.ws
2106         port="http://www.quickstockquote.com/StockQuoteService#
2107         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2108     </reference>
2109
2110     <property name="currency" type="xsd:string">EURO</property>
2111
2112     <component name="AccountServiceComponent">
2113       <implementation.composite name="foo:AccountServiceCompositel"/>
2114
2115       <reference name="AccountDataService" target="AccountDataService"/>
2116       <reference name="StockQuoteService"/>
2117
2118       <property name="currency" source="$currency"/>
2119     </component>
2120
2121     <component name="AccountDataService">
2122       <implementation.composite name="foo:AccountDataServiceCompositel"/>
2123
2124       <property name="currency" source="$currency"/>
2125     </component>
2126
2127   </composite>
2128
```

## 2129 6.6 Using Composites through Inclusion

2130 In order to assist team development, composites may be developed in the form of multiple  
2131 physical artifacts that are merged into a single logical unit.

2132 **A composite may include another composite by using the `include` element. This provides a**  
2133 **recursive inclusion capability. The semantics of included composites are that the element content**

2134 children of the included composite are inlined, with certain modification, into the using composite.  
2135 This is done recursively till the resulting composite does not contain an **include** element. The  
2136 outer included composite element itself is discarded in this process – only its contents are included  
2137 as described below:

- 2138 1. All the element content children of the included composite are inlined in the including  
2139 composite.
- 2140 2. The attributes **targetNamespace**, **name**, **constrainingType**, and **local** of the included  
2141 composites are discarded.
- 2142 3. All the namespace declaration on the included composite element are added to the inlined  
2143 element content children unless the namespace binding is overridden by the element  
2144 content children.
- 2145 4. The attribute **autowire**, if specified on the included composite, is included on all inlined  
2146 component element children unless the component child already specifies that attribute.
- 2147 5. The attribute values of **requires** and **policySet**, if specified on the included composite,  
2148 are merged with corresponding attribute on the inlined component, service and reference  
2149 children elements. Merge in this context means a set union.
- 2150 6. Extension attributes ,if present on the included composite, must follow the rules defined  
2151 for that extension. Authors of attribute extensions on the composite element must define  
2152 rules for inclusion.

2153 If the included composite has the value *true* for the attribute **local** then the including composite  
2154 must have the same value for the **local** attribute, else it is considered an error.

2155 The composite file used for inclusion can have any contents, but its document root element must  
2156 be **composite**. The composite element may contain any of the elements which are valid as child  
2157 elements of a composite element, namely components, services, references, wires and includes.  
2158 There is no need for the content of an included composite to be complete, so that artifacts defined  
2159 within the using composite or in another associated included composite file may be referenced. For  
2160 example, it is permissible to have two components in one composite file while a wire specifying  
2161 one component as the source and the other as the target can be defined in a second included  
2162 composite file.

2163 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one  
2164 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated  
2165 elements in the using composite (eg. two services with the same uri contributed by different  
2166 included composites). It is not considered an error if the (using) composite resulting from the  
2167 inclusion is incomplete (eg. wires with non-existent source or target). Such incomplete resulting  
2168 composites are permitted to allow recursive composition.

2169 The following snippet shows the **pseudo**-schema for the include element.

```
2170  
2171 <?xml version="1.0" encoding="UTF-8"?>  
2172 <!-- Include snippet -->  
2173 <composite ...>  
2174   ...  
2175   <include name="xs:QName"/>*  
2176   ...  
2177 </composite>
```

2179 The include element has the following **attribute**:

- 2180 • **name (required)** – the name of the composite that is included.

## 2182 6.6.1 Included Composite Examples

2183

**Deleted:** A composite is defined in an **xxx.composite** file and the composite may receive additional content through the **inclusion of other composite** files. ¶  
The semantics of included composites are that the content of the included composite is inlined into the using composite **xxx.composite** file through **include** elements in the using composite. The effect is one of **textual inclusion** – that is, the text content of the included composite is placed into the using composite in place of the include statement. The included composite element itself is discarded in this process – only its contents are included. ¶  
The composite file used for inclusion can have any contents, but always contains a single **composite** element. The composite element can contain any of the elements which are valid as child elements of a composite element, namely components, services, references, wires and includes. There is no need for the content of an included composite to be complete, so that artifacts defined within the using composite or in another associated included composite file may be referenced. For example, it is permissible to have two components in one

[2]

**Deleted:** ,

**Comment [ME46]:** Issue 17

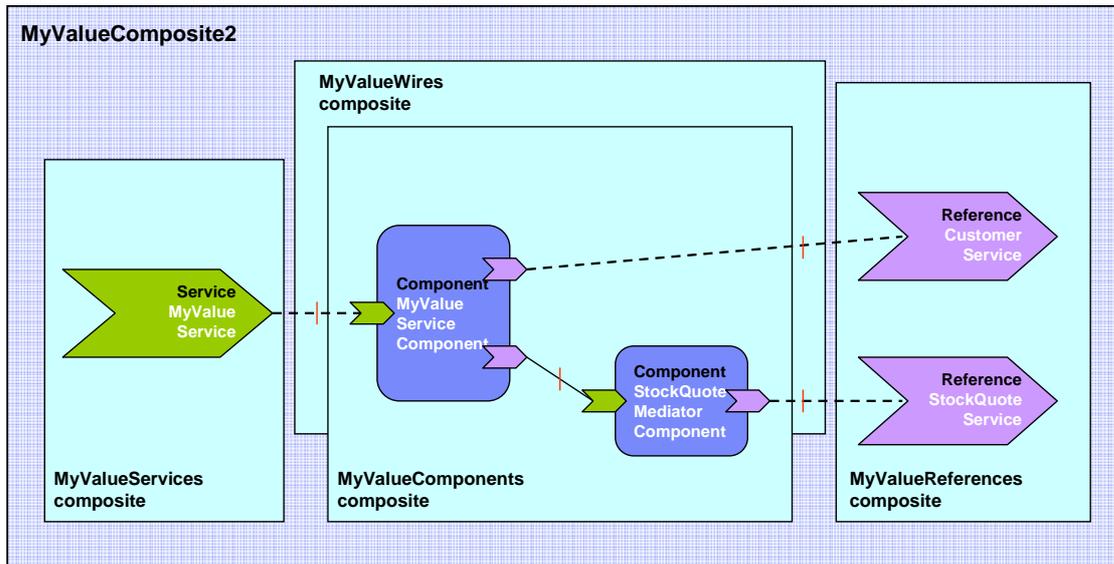
**Deleted:** or if there are wires with non-existent source or target. ¶

**Deleted:** partial

**Deleted:** .xmlns="http://d  
ocs.oasis-  
open.org/ns/opencsa/sca  
/200712" ¶  
...targetNamespace="xs:  
anyURI" ¶  
...name="xs:NCName"  
local="xs:boolean"?  
autowire="xs:boolean"? ¶  
...constrainingType="QN  
ame"? ¶

[3]

2184 The following figure shows the assembly diagram for the MyValueComposite2 containing four  
 2185 included composites. The **MyValueServices composite** contains the MyValueService service. The  
 2186 **MyValueComponents composite** contains the MyValueServiceComponent and the  
 2187 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences**  
 2188 **composite** contains the CustomerService and StockQuoteService references. The **MyValueWires**  
 2189 **composite** contains the wires that connect the MyValueService service to the  
 2190 MyValueServiceComponent, that connect the customerService reference of the  
 2191 MyValueServiceComponent to the CustomerService reference, and that connect the  
 2192 stockQuoteService reference of the StockQuoteMediatorComponent to the StockQuoteService  
 2193 reference. Note that this is just one possible way of building the MyValueComposite2 from a set of  
 2194 included composites.



2195  
 2196  
 2197 *Figure 13 MyValueComposite2 built from 4 included composites*

2198  
 2199 The following snippet shows the contents of the MyValueComposite2.composite file for the  
 2200 MyValueComposite2 built using included composites. In this sample it only provides the name of  
 2201 the composite. The composite file itself could be used in a scenario using included composites to  
 2202 define components, services, references and wires.

```
2203
2204 <?xml version="1.0" encoding="ASCII"?>
2205 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2206           targetNamespace="http://foo.com"
2207           xmlns:foo="http://foo.com"
2208           name="MyValueComposite2" >
2209
2210     <include name="foo:MyValueServices"/>
2211     <include name="foo:MyValueComponents"/>
2212     <include name="foo:MyValueReferences"/>
2213     <include name="foo:MyValueWires"/>
2214
2215 </composite>
2216
```

2217 The following snippet shows the content of the MyValueServices.composite file.

2218

```
2219 <?xml version="1.0" encoding="ASCII"?>
2220 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2221           targetNamespace="http://foo.com"
2222           xmlns:foo="http://foo.com"
2223           name="MyValueServices" >
2224
2225     <service name="MyValueService" promote="MyValueServiceComponent">
2226       <interface.java interface="services.myvalue.MyValueService"/>
2227       <binding.ws port="http://www.myvalue.org/MyValueService#
2228                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
2229     </service>
2230
2231 </composite>
2232
```

2233 The following snippet shows the content of the MyValueComponents.composite file.

2234

```
2235 <?xml version="1.0" encoding="ASCII"?>
2236 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2237           targetNamespace="http://foo.com"
2238           xmlns:foo="http://foo.com"
2239           name="MyValueComponents" >
2240
2241     <component name="MyValueServiceComponent">
2242       <implementation.java
2243         class="services.myvalue.MyValueServiceImpl"/>
2244       <property name="currency">EURO</property>
2245     </component>
2246
2247     <component name="StockQuoteMediatorComponent">
2248       <implementation.java class="services.myvalue.SQMediatorImpl"/>
2249       <property name="currency">EURO</property>
2250     </component>
2251
2252 </composite>
2253
```

2254 The following snippet shows the content of the MyValueReferences.composite file.

2255

```
2256 <?xml version="1.0" encoding="ASCII"?>
2257 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2258           targetNamespace="http://foo.com"
2259           xmlns:foo="http://foo.com"
2260           name="MyValueReferences" >
2261
2262     <reference name="CustomerService"
2263       promote="MyValueServiceComponent/CustomerService">
2264       <interface.java interface="services.customer.CustomerService"/>
2265       <binding.sca/>
2266     </reference>
2267
2268     <reference name="StockQuoteService"
2269       promote="StockQuoteMediatorComponent">
2270       <interface.java
```

```

2271 | _____ interface="services.stockquote.StockQuoteService" />
2272 |         <binding.ws port="http://www.stockquote.org/StockQuoteService#
2273 |         _____ wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP) " />
2274 |     </reference>
2275 |
2276 | </composite>
2277 |

```

2278 | The following snippet shows the content of the MyValueWires.composite file.

```

2280 | <?xml version="1.0" encoding="ASCII"?>
2281 | <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2282 |           targetNamespace="http://foo.com"
2283 |           xmlns:foo="http://foo.com"
2284 |           name="MyValueWires" >
2285 |
2286 |     <wire source="MyValueServiceComponent/stockQuoteService"
2287 |           target="StockQuoteMediatorComponent" />
2288 |
2289 | </composite>

```

## 2290 | 6.7 Composites which Include Component Implementations of 2291 | Multiple Types

2292 | A Composite containing multiple components can have multiple component implementation types.  
2293 | For example, a Composite may include one component with a Java POJO as its implementation  
2294 | and another component with a BPEL process as its implementation.

Comment [ME47]: Issue 16

## 2295 | 6.8 Structural URI of Components

2296 | The **structural URI** is a relative URI that describes each use of a given component in the Domain,  
2297 | relative to the URI of the domain itself. It is never specified explicitly, but it calculated from the  
2298 | configuration of the components configured into the Domain.

2299 | A component in a composite may be used more than once in the domain, if its containing  
2300 | composite is used as the implementation of more than one higher-level component. The structural  
2301 | URI may be used to separately identify each use of a component - for example, the structural URI  
2302 | may be used to attach different policies to each separate use of a component.

2303 | For components directly deployed into the domain, the structural URI is simply the name of the  
2304 | component.

2305 | Where components are nested within a composite which is used as the implementation of a higher  
2306 | level component, the structural URI consists of the name of the nested component prepended with  
2307 | each of the names of the components upto and including the domain level component.

2308 | For example, consider a component named Component1 at the domain level, where its  
2309 | implementation is Composite1 which in turn contains a component named Component2, which is  
2310 | implemented by Composite2 which contains a component named Component3. The three  
2311 | components in this example have the following structural URIs:

- 2312 | 1. Component1: Component1
- 2313 | 2. Component2: Component1/Component2
- 2314 | 3. Component3: Component1/Component2/Component3

2315 | The structural URI can also be extended to refer to specific parts of a component, such as a  
2316 | service or a reference, by appending an appropriate fragment identifier to the component's  
2317 | structural URI, as follows:

2318  
2319  
2320  
  
2321  
2322  
2323  
  
2324  
2325  
2326  
  
2327  
2328  
  
2329  
2330  
  
2331

- Service:  
#service(servicename)
  
- Reference:  
#reference(referencename)
  
- Service binding:  
#service-binding(servicename/bindingname)
  
- Reference binding:  
#reference-binding(referencename/bindingname)

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

## 2332 7 ConstrainingType

2333 SCA allows a component, and its associated implementation, to be constrained by a  
2334 **constrainingType**. The constrainingType element provides assistance in developing top-down  
2335 usecases in SCA, where an architect or assembler can define the structure of a composite,  
2336 including the required form of component implementations, before any of the implementations are  
2337 developed.

2338 A constrainingType is expressed as an element which has services, reference and properties as  
2339 child elements and which can have intents applied to it. The constrainingType is independent of  
2340 any implementation. Since it is independent of an implementation it cannot contain any  
2341 implementation-specific configuration information or defaults. Specifically, it cannot contain  
2342 bindings, policySets, property values or default wiring information. The constrainingType is  
2343 applied to a component through a constrainingType attribute on the component.

2344 A constrainingType provides the "shape" for a component and its implementation. Any component  
2345 configuration that points to a constrainingType is constrained by this shape. The constrainingType  
2346 specifies the services, references and properties that MUST be implemented by the  
2347 implementation of the component to which the constrainingType is attached. [ASM70001] This  
2348 provides the ability for the implementer to program to a specific set of services, references and  
2349 properties as defined by the constrainingType. Components are therefore configured instances of  
2350 implementations and are constrained by an associated constrainingType.

2351 If the configuration of the component or its implementation do not conform to the  
2352 constrainingType specified on the component element, the SCA runtime MUST raise an error.  
2353 [ASM70002]

2354 A constrainingType is represented by a **constrainingType** element. The following snippet shows  
2355 the pseudo-schema for the composite element.

```
2356  
2357 <?xml version="1.0" encoding="ASCII"?>  
2358 <!-- ConstrainingType schema snippet -->  
2359 <constrainingType xmlns="http://docs.oasis-  
2360 open.org/ns/opencsa/sca/200712"  
2361     targetNamespace="xs:anyURI" ?  
2362     name="xs:NCName" requires="list of xs:QName"?>  
2363  
2364  
2365     <service name="xs:NCName" requires="list of xs:QName"?>*  
2366         <interface ... />?  
2367     </service>  
2368  
2369     <reference name="xs:NCName"  
2370         multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
2371         requires="list of xs:QName"?>*  
2372         <interface ... />?  
2373     </reference>  
2374  
2375     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")  
2376         many="xs:boolean"? mustSupply="xs:boolean"?>*
```

2377           default-property-value?  
2378       </property>  
2379  
2380 </constrainingType>  
2381

2382 The constrainingType element has the following **attributes**:

- 2383     • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is  
2384       an XML QName, in the namespace identified by the targetNamespace attribute. The name  
2385       attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2386     • **targetNamespace (0..1)** – an identifier for a target namespace into which the  
2387       constrainingType is declared
- 2388     • **requires (0..1)** – a list of policy intents. See the Policy Framework specification [10] for  
2389       a description of this attribute.

2390 ConstrainingType contains **zero or more properties, services, references**.

2391

2392 When an implementation is constrained by a constrainingType its component type MUST contain  
2393 all the services, references and properties specified in the constrainingType. [ASM70004] The  
2394 constraining type's references and services will have interfaces specified and can have intents  
2395 specified. An implementation MAY contain additional services, additional optional references  
2396 (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the  
2397 constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or  
2398 1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

Comment [ME48]: Issue 65

2399 When a component is constrained by a constrainingType via the "constrainingType" attribute, the  
2400 entire componentType associated with the component and its implementation is not visible to the  
2401 containing composite. The containing composite can only see a projection of the componentType  
2402 associated with the component and implementation as scoped by the constrainingType of the  
2403 component. Additional services, references and properties provided by the implementation which  
2404 are not declared in the constrainingType associated with a component MUST NOT be configured in  
2405 any way by the containing composite. [ASM70006] This requirement ensures that the  
2406 constrainingType contract cannot be violated by the composite.

2407 The constrainingType can include required intents on any element. Those intents are applied to  
2408 any component that uses that constrainingType. In other words, if requires="reliability" exists on  
2409 a constrainingType, or its child service or reference elements, then a constrained component or its  
2410 implementation must include requires="reliability" on the component or implementation or on its  
2411 corresponding service or reference. A component or implementation can use a qualified form of  
2412 an intent specified in unqualified form in the constrainingType, but if the constrainingType uses  
2413 the qualified form of an intent, then the component or implementation MUST also use the qualified  
2414 form, otherwise there is an error. [ASM70007]

2415 A constrainingType can be applied to an implementation. In this case, the implementation's  
2416 componentType has a constrainingType attribute set to the QName of the constrainingType.

2417

## 2418 7.1 Example constrainingType

2419

2420 The following snippet shows the contents of the component called "MyValueServiceComponent"  
2421 which is constrained by the constrainingType myns:CT. The componentType associated with the  
2422 implementation is also shown.

2423

```
2424 <component name="MyValueServiceComponent" constrainingType="myns:CT">  
2425 <implementation.java class="services.myvalue.MyValueServiceImpl" />
```

```

2426     <property name="currency">EURO</property>
2427     <reference name="customerService" target="CustomerService">
2428         <binding.ws ...>
2429     <reference name="StockQuoteService"
2430         target="StockQuoteMediatorComponent" />
2431 </component>
2432
2433 <constrainingType name="CT"
2434     targetNamespace="http://myns.com">
2435     <service name="MyValueService">
2436         <interface.java interface="services.myvalue.MyValueService" />
2437     </service>
2438     <reference name="customerService">
2439         <interface.java interface="services.customer.CustomerService" />
2440     </reference>
2441     <reference name="stockQuoteService">
2442         <interface.java interface="services.stockquote.StockQuoteService" />
2443     </reference>
2444     <property name="currency" type="xsd:string" />
2445 </constrainingType>

```

2446 The component MyValueServiceComponent is constrained by the constrainingType CT which  
2447 means that it must provide:

- 2448 • service **MyValueService** with the interface services.myvalue.MyValueService
- 2449 • reference **customerService** with the interface services.stockquote.StockQuoteService
- 2450 • reference **stockQuoteService** with the interface services.stockquote.StockQuoteService
- 2451 • property **currency** of type xsd:string.

2452 **8 Interface**

2453 **Interfaces** define one or more business functions. These business functions are provided by  
2454 Services and are used by References. A Service offers the business functionality of exactly one  
2455 interface for use by other components. Each interface defines one or more service **operations**  
2456 and each operation has zero or one **request (input) message** and zero or one **response**  
2457 **(output) message**. The request and response messages can be simple types such as a string  
2458 value or they can be complex types.

2459 SCA currently supports the following interface type systems:

- 2460 • Java interfaces
- 2461 • WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))
- 2462 • C++ classes
- 2463 • Collections of 'C' functions

2464 SCA is also extensible in terms of interface types. Support for other interface type systems can be  
2465 added through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2466 The following snippet shows the definition for the **interface** base element.  
2467

```
2468 <interface requires="list of xs:QName"? policySets="list of xs:QName"? />
```

2471 The **interface** base element has the following **attributes**:

- 2472 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
2473 [\[10\]](#) for a description of this attribute
- 2474 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2475 [\[10\]](#) for a description of this attribute.

2476 The following snippet shows a sample for the WSDL portType (WSDL 1.1) element.  
2477

```
2478 <interface.wSDL interface="xs:anyURI" ... />
```

2481 The interface.wSDL element has the following attributes:

- 2482 • **interface** – URI of the portType/interface with the following format.  
2483 o <WSDL-namespace-URI> #wsdl.interface(<portTypeOrInterface-name>)

2484 The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.0  
2485 document OR an interface element of a WSDL 2.0 document. [ASM80001]

2486 The following snippet shows a sample for the WSDL portType/interface element.  
2487

```
2488 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2489 wsdl.interface(StockQuo  
2490 te)"/>
```

**Comment [mbgl49]:** Issue 69 part 2

**Comment [ME50]:** Issue 77

**Deleted:** <#>WSDL 2.0 interfaces ([Web Services Definition Language \[8\]](#))¶

**Comment [ME51]:** Issue 69 Part 2

**Formatted:** Bullets and Numbering

**Comment [mbgl52]:** Issue 39

**Comment [ME53]:** Issue 77

**Deleted:** or WSDL interface (WSDL 2.0

**Comment [ME54]:** Issue 77

**Deleted:** 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.

2493 For WSDL 1.1, the interface attribute points to a portType in the WSDL. For the WSDL 1.1  
2494 portType interface type `system`, arguments and return of the service operations are described  
2495 using XML schema.

**Deleted:** For WSDL 2.0, the interface attribute points to an interface in the WSDL.

2496 For information about Java interfaces, including details of SCA-specific annotations, see the SCA  
2497 Java Common Annotations and APIs specification [1].

**Comment [ME55]:** Issue 77

**Deleted:** and WSDL 2.0

**Deleted:** s

## 2498 8.1 Local and Remotable Interfaces

2499 A remotable service is one which may be called by a client which is running in an operating system  
2500 process different from that of the service itself (this also applies to clients running on different  
2501 machines from the service). Whether a service of a component implementation is remotable is  
2502 defined by the interface of the service. WSDL defined interfaces are always remotable. See the  
2503 relevant specifications for details of interfaces defined using other languages.

**Comment [ME56]:** Issue 92

2504  
2505 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**  
2506 interactions. **Remotable service Interfaces MUST NOT make use of *method or operation***  
2507 ***overloading***. [ASM80002] This restriction on operation overloading for remotable services aligns  
2508 with the WSDL 2.0 specification, which disallows operation overloading, and also with the WS-I  
2509 Basic Profile 1.1 (section 4.5.3 - R2304) which has a constraint which disallows operation  
2510 overloading when using WSDL 1.1.

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

2511 Independent of whether the remotable service is called remotely from outside the process where  
2512 the service runs or from another component running in the same process, the data exchange  
2513 semantics are **by-value**.  
2514

2515 Implementations of remotable services can modify input messages (parameters) during or after  
2516 an invocation and can modify return messages (results) after the invocation. If a remotable  
2517 service is called locally or remotely, the SCA container MUST ensure sure that no modification of  
2518 input messages by the service or post-invocation modifications to return messages are seen by  
2519 the caller. [ASM80003]

2520 Here is a snippet which shows an example of a remotable java interface:

```
2521  
2522 package services.hello;  
2523  
2524 @Remotable  
2525 public interface HelloService {  
2526  
2527     String hello(String message);  
2528 }  
2529
```

2530 It is possible for the implementation of a remotable service to indicate that it can be called using  
2531 by-reference data exchange semantics when it is called from a component in the same process.  
2532 This can be used to improve performance for service invocations between components that run in  
2533 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java  
2534 Client and Implementation Specification](#)).

2535 A service typed by a local interface can only be called by clients that are running in the same  
2536 process as the component that implements the local service. Local services cannot be published  
2537 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2538 Java interface definition without a **@Remotable** annotation.

**Deleted:** ¶

2539 The style of local interfaces is typically **fine grained** and intended for **tightly coupled**  
2540 interactions. Local service interfaces can make use of ***method or operation overloading***.

**Deleted:** ¶

2541 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

2542

## 2543 8.2 Bidirectional Interfaces

2544 The relationship of a business service to another business service is often peer-to-peer, requiring  
2545 a two-way dependency at the service level. In other words, a business service represents both a  
2546 consumer of a service provided by a partner business service and a provider of a service to the  
2547 partner business service. This is especially the case when the interactions are based on  
2548 asynchronous messaging rather than on remote procedure calls. The notion of **bidirectional**  
2549 **interfaces** is used in SCA to directly model peer-to-peer bidirectional business service  
2550 relationships.

2551 An interface element for a particular interface type system needs to allow the specification of an  
2552 optional callback interface. If a callback interface is specified, SCA refers to the interface as a  
2553 whole as a bidirectional interface.

2554 The following snippet shows the interface element defined using Java interfaces with an optional  
2555 callbackInterface attribute.

2556

```
2557 <interface.java      interface="services.invoicing.ComputePrice"  
2558                   callbackInterface="services.invoicing.InvoiceCallback"/>
```

2559

2560 If a service is defined using a bidirectional interface element then its implementation implements  
2561 the interface, and its implementation uses the callback interface to converse with the client that  
2562 called the service interface.

2563

2564 If a reference is defined using a bidirectional interface element, the client component  
2565 implementation using the reference calls the referenced service using the interface. The client  
2566 MUST provide an implementation of the callback interface. [ASM80004]

2567 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2568 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2569 mix local and remote services. [ASM80005]

2570 Note that an interface document such as a WSDL file or a Java interface can contain annotations  
2571 that declare a callback interface for a particular interface (see the section on WSDL Interface type  
2572 and the Java Common Annotations and APIs specification [SCA-Common-Java]). Whenever an  
2573 interface document declaring a callback interface is used in the declaration of an <interface/>  
2574 element in SCA, it MUST be treated as being bidirectional with the declared callback interface.  
2575 [ASM80010] In such cases, there is no requirement for the <interface/> element to declare the  
2576 callback interface explicitly.

2577 If an <interface/> element references an interface document which declares a callback interface  
2578 and also itself contains a declaration of a callback interface, the two callback interfaces MUST be  
2579 compatible. [ASM80011]

2580 Where a component uses an implementation and the component configuration explicitly declares  
2581 an interface for a service or a reference, if the matching service or reference declaration in the  
2582 component type declares an interface which has a callback interface, then the component interface  
2583 declaration MUST also declare a compatible interface with a compatible callback interface.  
2584 [ASM80012] If the service or reference declaration in the component type declares an interface  
2585 without a callback interface, then the component configuration for the corresponding service or  
2586 reference MUST NOT declare an interface with a callback interface. [ASM80013]

2587 Where a composite declares an interface for a composite service or a composite reference, if the  
2588 promoted service or promoted reference has an interface which has a callback interface, then the  
2589 interface declaration for the composite service or the composite reference MUST also declare a  
2590 compatible interface with a compatible callback interface. [ASM80014] If the promoted service or  
2591 promoted reference has an interface without a callback interface, then the interface declaration for

2592 the composite service or composite reference MUST NOT declare a callback interface.  
2593 [ASM80015]

2594 See Section 6.4 Wires for a definition of "compatible interfaces".

Comment [ME57]: Issue 89

2595 In a bidirectional interface, the service interface can have more than one operation defined, and  
2596 the callback interface can also have more than one operation defined. SCA runtimes MUST allow  
2597 an invocation of any operation on the service interface to be followed by zero, one or many  
2598 invocations of any of the operations on the callback interface. [ASM80009] These callback  
2599 operations can be invoked either before or after the operation on the service interface has  
2600 returned a response message, if there is one.

2601 For a given invocation of a service operation, which operations are invoked on the callback  
2602 interface, when these are invoked, the number of operations invoked, and their sequence are not  
2603 described by SCA. It is possible that this metadata about the bidirectional interface can be  
2604 supplied through mechanisms outside SCA. For example, it might be provided as a written  
2605 description attached to the callback interface.

Comment [ME58]: Issue 56

### 2606 8.3 Conversational Interfaces

2607 Services sometimes cannot easily be defined so that each operation stands alone and is  
2608 completely independent of the other operations of the same service. Instead, there is a sequence  
2609 of operations that must be called in order to achieve some higher level goal. SCA calls this  
2610 sequence of operations a **conversation**. If the service uses a bidirectional interface, the  
2611 conversation may include both operations and callbacks.

2612 Such **conversational services** are typically managed by using conversation identifiers that are  
2613 either (1) part of the application data (message parts or operation parameters) or 2)  
2614 communicated separately from application data (possibly in headers). SCA introduces the concept  
2615 of **conversational interfaces** for describing the interface contract for conversational services of  
2616 the second form above. With this form, it is possible for the runtime to automatically manage the  
2617 conversation, with the help of an appropriate binding specified at deployment. SCA does not  
2618 standardize any aspect of conversational services that are maintained using application data.  
2619 Such services are neither helped nor hindered by SCA's conversational service support.

2620 Conversational services typically involve state data that relates to the conversation that is taking  
2621 place. The creation and management of the state data for a conversation has a significant impact  
2622 on the development of both clients and implementations of conversational services.

2623

2624 Traditionally, application developers who have needed to write conversational services have been  
2625 required to write a lot of plumbing code. They need to:

2626

2627 - choose or define a protocol to communicate conversational (correlation) information  
2628 between the client & provider

2629 - route conversational messages in the provider to a machine that can handle that  
2630 conversation, while handling concurrent data access issues

2631 - write code in the client to use/encode the conversational information

2632 - maintain state that is specific to the conversation, sometimes persistently and  
2633 transactionally, both in the implementation and the client.

2634

2635 SCA makes it possible to divide the effort associated with conversational services between a  
2636 number of roles:

2637 - Application Developer: Declares that a service interface is conversational (leaving the  
2638 details of the protocol up to the binding). Uses lifecycle semantics, APIs or other  
2639 programmatic mechanisms (as defined by the implementation-type being used) to  
2640 manage conversational state.

- 2641 - Application Assembler: chooses a binding that can support conversations
- 2642 - Binding Provider: implements a protocol that can pass conversational information with  
2643 each operation request/response.
- 2644 - Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for  
2645 application developers to access conversational information. Optionally implements  
2646 instance lifecycle semantics that automatically manage implementation state based on  
2647 the binding's conversational information.

2648

2649 **There** is a policy intent with the name **conversational** which is used to mark an interface as being  
2650 conversational in nature. Where a service or a reference has a conversational interface, the  
2651 conversational intent MUST be attached either to the interface itself, or to the service or reference  
2652 using the interface. [ASM80006] How to attach the conversational intent to an interface depends  
2653 on the type of the interface. For a WSDL interface, this is described in section 8.4 "SCA-Specific  
2654 Aspects for WSDL Interfaces". For a Java interface, it is described in the Java Common  
2655 Annotations and APIs specification. Note that setting the conversational intent on the service or  
2656 reference element is useful when reusing an existing interface definition that contains no SCA  
2657 information, since it requires no modification of the interface artifact.

Comment [mbgl59]: Issue  
35

2658 The meaning of the conversational intent is that both the client and the provider of the interface  
2659 can assume that messages (in either direction) will be handled as part of an ongoing conversation  
2660 without depending on identifying information in the body of the message (i.e. in parameters of the  
2661 operations). In effect, the conversation interface specifies a high-level abstract protocol that must  
2662 be satisfied by any actual binding/policy combination used by the service.

2663 Examples of binding/policy combinations that support conversational interfaces are:

- 2664 - Web service binding with a WS-RM policy
- 2665 - Web service binding with a WS-Addressing policy
- 2666 - Web service binding with a WS-Context policy
- 2667 - JMS binding with a conversation policy that uses the JMS correlationID header

2668

2669 Conversations occur between one client and one target service. Consequently, requests originating  
2670 from one client to multiple target conversational services will result in multiple conversations. For  
2671 example, if a client A calls services B and C, both of which implement conversational interfaces,  
2672 two conversations result, one between A and B and another between A and C. Likewise, requests  
2673 flowing through multiple implementation instances will result in multiple conversations. For  
2674 example, a request flowing from A to B and then from B to C will involve two conversations (A and  
2675 B, B and C). In the previous example, if a request was then made from C to A, a third  
2676 conversation would result (and the implementation instance for A would be different from the one  
2677 making the original request).

2678 Invocation of any operation of a conversational interface can start a conversation. The decision on  
2679 whether an operation starts a conversation depends on the component's implementation and its  
2680 implementation type. Implementation types can support components which provide conversational  
2681 services. If an implementation type does provide this support, the specification for that  
2682 implementation type defines a mechanism for determining when a new conversation should be  
2683 used for an operation (for example, in Java, the conversation is new on the first use of an injected  
2684 reference; in BPEL, the conversation is new when the client's partnerLink comes into scope).

2685

2686 One or more operations in a conversational interface can be annotated with an  
2687 **endsConversation** annotation (the mechanism for annotating the interface depends on the  
2688 interface type) which indicates that when the operation is invoked, the conversation is at an end.  
2689 Where an interface is **bidirectional**, operations may also be annotated in this way on operations  
2690 of the callback interface. When a conversation ending operation is called, it indicates to both the  
2691 client and the service provider that the conversation is complete. Once an operation marked with  
2692 endsConversation has been invoked, any subsequent attempts to call an operation or a callback

2693 operation associated with the same conversation MUST generate a sca:ConversationViolation fault.  
2694 [\[ASM80007\]](#)

2695 A sca:ConversationViolation fault is thrown when one of the following errors occur:

- 2696 - A message is received for a particular conversation, after the conversation has ended
- 2697 - The conversation identification is invalid (not unique, out of range, etc.)
- 2698 - The conversation identification is not present in the input message of the operation that  
2699 ends the conversation
- 2700 - The client or the service attempts to send a message in a conversation, after the  
2701 conversation has ended

2702 This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI  
2703 <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2704 The lifecycle of resources and the association between unique identifiers and conversations are  
2705 determined by the service's implementation type and may not be directly affected by the  
2706 "endConversation" annotation. For example, a WS-BPEL process can outlive most of the  
2707 conversations that it is involved in.

2708 Although conversational interfaces do not require that any identifying information be passed as  
2709 part of the body of messages, there is conceptually an identity associated with the conversation.  
2710 Individual implementations types can have an API to access the ID associated with the  
2711 conversation, although no assumptions can be made about the structure of that identifier.  
2712 Implementation types can also have a means to set the conversation ID by either the client or the  
2713 service provider, although the operation may only be supported by some binding/policy  
2714 combinations.

2715 Implementation-type specifications are encouraged to define and provide conversational instance  
2716 lifecycle management for components that implement conversational interfaces. However,  
2717 implementations could also manage the conversational state manually.

2718

Comment [ME60]: Issue 33

## 2719 **8.4 Long-running Request-Response Operations**

### 2720 **8.4.1 Background**

2721 A service offering one or more operations which map to a WSDL request-response pattern may be  
2722 implemented in a long-running, potentially interruptible, way. Consider a BPEL process with  
2723 receive and reply activities referencing the WSDL request-response operation. Between the two  
2724 activities, the business process logic may be a long-running sequence of steps, including activities  
2725 causing the process to be interrupted. Typical examples are steps where the process waits for  
2726 another message to arrive or a specified time interval to expire, or the process may perform  
2727 asynchronous interactions such as service invocations bound to asynchronous protocols or user  
2728 interactions. This is a common situation in business processes, and it causes the implementation  
2729 of the WSDL request-response operation to run for a very long time, e.g., several months (!). In  
2730 this case, it is not meaningful for any caller to remain in a synchronous wait for the response while  
2731 blocking system resources or holding database locks.

2732 Note that it is possible to model long-running interactions as a pair of two independent operations  
2733 as described in the section on bidirectional interfaces. However, it is a common practice (and in  
2734 fact much more convenient) to model a request-response operation and let the infrastructure deal  
2735 with the asynchronous message delivery and correlation aspects instead of putting this burden on  
2736 the application developer.

2737

### 2738 **8.4.2 Definition of "long-running"**

2739 A request-response operation is considered long-running if the implementation does not guarantee  
2740 the delivery of the response within any specified time interval. Clients invoking such request-

2741 response operations are strongly discouraged from making assumptions about when the response  
2742 can be expected.

2743

### 2744 **8.4.3 The asyncInvocation Intent**

2745 This specification permits a long-running request-response operation or a complete interface  
2746 containing such operations to be marked using a policy intent with the name **asyncInvocation**. It  
2747 is also possible for a service to set the asyncInvocation. intent when using an interface which is  
2748 not marked with the asyncInvocation. intent. This can be useful when reusing an existing interface  
2749 definition that does not contain SCA information.

2750

### 2751 **8.4.4 Requirements on Bindings**

2752 In order to support a service operation which is marked with the asyncInvocation intent, it is  
2753 necessary for the binding (and its associated policies) to support separate handling of the request  
2754 message and the response message. Bindings which only support a synchronous style of message  
2755 handling, such as a conventional HTTP binding, cannot be used to support long-running  
2756 operations.

2757 The requirements on a binding to support the asyncInvocation intent are the same as those  
2758 required to support services with bidirectional interfaces - namely that the binding needs to be  
2759 able to treat the transmission of the request message separately from the transmission of the  
2760 response message, with an arbitrarily large time interval between the two transmissions.

2761 An example of a binding/policy combination that supports long-running request-response  
2762 operations is a Web service binding used in conjunction with the WS-Addressing  
2763 "wsam:NonAnonymousResponses" assertion.

2764

### 2765 **8.4.5 Implementation Type Support**

2766 SCA implementation types can provide special asynchronous client-side and asynchronous server-  
2767 side mappings to assist in the development of services and clients for long-running request-  
2768 response operations.

## 2769 **8.5 SCA-Specific Aspects for WSDL Interfaces**

2770 There are a number of aspects that SCA applies to interfaces in general, such as marking them  
2771 **conversational**. These aspects apply to the interfaces themselves, rather than their use in a  
2772 specific place within SCA. There is thus a need to provide appropriate ways of marking the  
2773 interface definitions themselves, which go beyond the basic facilities provided by the interface  
2774 definition language.

2775 For WSDL interfaces, there is an extension mechanism that permits additional information to be  
2776 included within the WSDL document. SCA takes advantage of this extension mechanism. In order  
2777 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-](http://docs.oasis-open.org/ns/opencsa/sca/200712)  
2778 [open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2779 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach  
2780 policy intents - **@requires**. The definition of this attribute is as follows:

2781 `<attribute name="requires" type="sca:listOfQNames"/>`

2782

2783 `<simpleType name="listOfQNames">`  
2784 `<list itemType="QName"/>`  
2785 `</simpleType>`

2786 | The @requires attribute can be applied to WSDL Port Type elements (WSDL 1.1). The attribute  
2787 | contains one or more intent names, as defined by the Policy Framework specification [10]. Any  
2788 | service or reference that uses an interface marked with required intents MUST implicitly add those  
2789 | intents to its own @requires list. [ASM80008]

2790 | To specify that a WSDL interface is conversational, the following attribute setting is used on either  
2791 | the WSDL Port Type or WSDL Interface:

2792 | `requires="conversational"`

2793 | SCA defines an **endsConversation** attribute that is used to mark specific operations within a  
2794 | WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces  
2795 | which are also marked conversational. The endsConversation attribute is a global attribute in the  
2796 | SCA namespace, with the following definition:

2797 | `<attribute name="endsConversation" type="boolean" default="false"/>`  
2798 |

2799 | The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2800 | the portType and the **endsConversation** attribute on one of the operations:

2801 | ...  
2802 | `<portType name="LoanService" sca:requires="conversational">`  
2803 |  `<operation name="apply">`  
2804 |  `<input message="tns:ApplicationInput"/>`  
2805 |  `<output message="tns:ApplicationOutput"/>`  
2806 |  `</operation>`  
2807 |  `<operation name="cancel" sca:endsConversation="true">`  
2808 |  `</operation>`  
2809 | ...  
2810 | `</portType>`  
2811 | ...

2812 | The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2813 | the portType and the **endsConversation** attribute on one of the operations:

2814 | ...  
2815 | `<portType name="LoanService" sca:requires="conversational">`  
2816 |  `<operation name="apply">`  
2817 |  `<input message="tns:ApplicationInput"/>`  
2818 |  `<output message="tns:ApplicationOutput"/>`  
2819 |  `</operation>`  
2820 |  `<operation name="cancel" sca:endsConversation="true">`  
2821 |  `</operation>`  
2822 | ...  
2823 | `</portType>`  
2824 | ...

2825 | SCA defines an attribute which is used to indicate that a given WSDL Port Type element (WSDL  
2826 | 1.1) has an associated callback interface. This is the @callback attribute, which applies to a WSDL  
2827 | <portType/> element.

2828 | The @callback attribute is defined as a global attribute in the SCA namespace, as follows:

2830 | `<attribute name="callback" type="QName"/>`

2831 | The value of the @callback attribute is the QName of a Port Type. The port type declared by the  
2832 |

Comment [ME61]: Issue 77

Deleted: ) and to WSDL  
Interface elements (WSDL  
2.0

Deleted: )

2833 @callback attribute is the callback interface to use for the portType which is annotated by the  
2834 @callback attribute.

2835 Here is an example of a portType element with a callback attribute:

```
2837 <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
2838   <operation name="apply">  
2839     <input message="tns:ApplicationInput" />  
2840     <output message="tns:ApplicationOutput" />  
2841   </operation>  
2842   ...  
2843 </portType>
```

Comment [ME62]: Issue 89

## 2845 8.6 WSDL Interface Type

Formatted: Bullets and Numbering

2846 The WSDL interface type is used to declare interfaces for services and for references, where the interface  
2847 is defined in terms of a WSDL document. This WSDL document MUST conform to the WSDL 1.1  
2848 specification. [ASM80009] An interface is defined in terms of a WSDL 1.1 Port Type.

2849 A WSDL interface is declared by an *interface.wSDL* element. The following shows the pseudo-schema  
2850 for the interface.wSDL element:

```
2852 <!-- WSDL Interface schema snippet -->  
2853 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?>
```

2854 The interface.wSDL element has the following *attributes*:

- 2855 • **interface (1..1)** - the URI of a WSDL Port Type
- 2856 • **callbackInterface(0..1)** - an optional callback interface, which is the URI of a WSDL Port  
2857 Type

2858 The form of the URI for WSDL port types follows the syntax described in the WSDL 1.1 Element  
2859 Identifiers specification [WSDL11\_Identifiers]

Formatted: Bullets and Numbering

### 2860 8.6.1 Example of interface.wSDL

```
2861 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2862   wsdl.porttype(StockQuote)"  
2863 callbackInterface="http://www.stockquote.org/StockQuoteService#  
2864   wsdl.porttype(StockQuoteCallback)"/>
```

2866 This declares an interface in terms of the WSDL port type "StockQuote" with a callback interface defined  
2867 by the "StockQuoteCallback" port type.

Comment [ME63]: Issue 89

2868

2869

## 9 Binding

2870  
2871  
2872  
2873

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

2874  
2875  
2876  
2877  
2878

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

2879  
2880  
2881

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

2882

2883  
2884

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Bindings schema snippet -->
```

2885  
2886

```
<composite ... >
  ...
```

2887

```
  <service ... >*
```

2888

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

2889

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

2890

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

2891

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

2892

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

2893

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

2894

```
          <wireFormat/>?
```

2895

```
          <operationSelector/>?
```

2896

```
        </binding>
```

2897

```
      </callback>?
```

2898

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

2899

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

2900

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

2901

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

2902

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

2903

```
            <wireFormat/>?
```

2904

```
            <operationSelector/>?
```

2905

```
          </binding>
```

2906

```
        </callback>
```

2907

```
      </service>
```

2908

```
    ...
```

2909

```
  <reference ... >*
```

2910

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

2911

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

2912

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

2913

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

2914

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

2915

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

2916

```
          <wireFormat/>?
```

2917

```
          <operationSelector/>?
```

2918

```
        </binding>
```

2919

```
      </callback>?
```

2920

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

2921

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

Comment [ME64]: Didn't we get rid of this requirement?

Deleted: 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"?

Deleted: name="xs:NCName" promote="xs:anyURI"
...requires="list of xs:QName"
policySets="list of xs:QName"?

Deleted: .

Deleted: .

Deleted: .

Deleted: /

Comment [ME65]: Issue 30

Deleted: .

Deleted: .

Deleted: .

Deleted: /

Comment [ME66]: Issue 30

Deleted:
name="xs:NCName" target="list of xs:anyURI"
...promote="list of xs:anyURI"
wiredByImpl="xs:boolean"
...multiplicity="0..1 or 1..1 or 0..n or 1..n"
...requires="list of xs:QName" [4]

Deleted: .

Deleted: .

Deleted: .

Deleted: /

Comment [ME67]: Issue 30

Deleted: .

Deleted: .

Deleted: .

```

2922     ,policySets="list of xs:QName"?>+
2923     <operation name="xs:NCName" requires="list of xs:QName"?
2924     policySets="list of xs:QName"?/>*
2925     <wireFormat/?>
2926     <operationSelector/?>
2927     </binding>
2928 </callback>
2929 </reference>
2930     ...
2931 </composite>

```

Deleted: ..

Deleted: /

Comment [ME68]: Issue 30

Deleted:

Comment [ME69]: Issue 79

2933 The element name of the binding element is architected; it is in itself a qualified name. The first  
 2934 qualifier is always named "binding", and the second qualifier names the respective binding-type  
 2935 (e.g. binding.composite, binding.ws, binding.ejb, binding.eis).

2936  
 2937 A binding element has the following attributes:

- 2938 • **uri (0..1)** - has the following semantic.
  - 2939 ○ The uri attribute can be omitted.
  - 2940 ○ For a binding of a **reference** the URI attribute defines the target URI of the  
 2941 reference. This MUST be either the componentName/serviceName for a wire to an  
 2942 endpoint within the SCA domain, or the accessible address of some service  
 2943 endpoint either inside or outside the SCA domain (where the addressing scheme is  
 2944 defined by the type of the binding). [ASM90001]
  - 2945 ○ The circumstances under which the uri attribute can be used are defined in  
 2946 section "5.3.1 Specifying the Target Service(s) for a Reference."
  - 2947 ○ For a binding of a **service** the URI attribute defines the URI relative to the  
 2948 component, which contributes the service to the SCA domain. The default value for  
 2949 the URI is the value of the name attribute of the binding.
- 2950 • **name (0..1)** - a name for the binding instance (an NCName). The name attribute allows  
 2951 distinction between multiple binding elements on a single service or reference. The  
 2952 default value of the name attribute is the service or reference name. When a service or  
 2953 reference has multiple bindings, only one binding can have the default name value; all  
 2954 others must have a name value specified that is unique within the service or reference.  
 2955 [ASM90002] The name also permits the binding instance to be referenced from elsewhere  
 2956 - particularly useful for some types of binding, which can be declared in a definitions  
 2957 document as a template and referenced from other binding instances, simplifying the  
 2958 definition of more complex binding instances (see the JMS Binding specification [11] for  
 2959 examples of this referencing).
- 2960 • **requires (0..1)** - a list of policy intents. See the Policy Framework specification [10] for a  
 2961 description of this attribute.
- 2962 • **policySets (0..1)** - a list of policy sets. See the Policy Framework specification [10] for a  
 2963 description of this attribute.

Comment [ME70]: This contradicts material below - is this the Issue 57 problem?

Deleted: For a binding of a **reference** the URI attribute defines the target URI of the reference. This MUST be either the componentName/serviceName for a wire to an endpoint within the SCA domain, or the accessible address of some service endpoint either inside or outside the SCA domain (where the addressing scheme is defined by the type of the binding).

Deleted: optional

Deleted: optional

2964 A binding element has the following child elements:

- 2965 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are  
 2966 used to describe characteristics of individual operations within the interface. For a detailed  
 2967 decription of the operation element, see the Policy Framework specification [SCA Policy].
- 2968 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See the  
 2969 wireFormat section for details.
- 2970 • **operationSelector(0..1)** - an operationSelector element that is used to match a  
 2971 particular message to a particular operation in the interface. See the operationSelector  
 2972 section for details

Comment [ME71]: Issue 30

Comment [ME72]: Issue 79

2973 When multiple bindings exist for an service, it means that the service is available by any of the  
2974 specified bindings. The technique that the SCA runtime uses to choose among available bindings  
2975 is left to the implementation and it may include additional (nonstandard) configuration. Whatever  
2976 technique is used needs to be documented by the runtime.

2977 Services and References can always have their bindings overridden at the SCA domain level,  
2978 unless restricted by Intents applied to them.

2979 If a reference has any bindings they MUST be resolved which means that each binding MUST  
2980 include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference  
2981 MUST NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds  
2982 of bindings that are acceptable for use with a reference, the user specifies either policy intents or  
2983 policy sets.

2984 Users can also specifically wire, not just to a component service, but to a specific binding offered  
2985 by that target service. To do so, a wire target MAY be specified with a syntax of  
2986 "componentName/serviceName/bindingName". [ASM90004]  
2987

2988

2989 The following sections describe the SCA and Web service binding type in detail.

2990

## 2991 9.1 Messages containing Data not defined in the Service Interface

2992 It is possible for a message to include information that is not defined in the interface used to  
2993 define the service, for instance information may be contained in SOAP headers or as MIME  
2994 attachments.

2995 Implementation types can make this information available to component implementations in their  
2996 execution context. The specifications for these implementation types describe how this  
2997 information is accessed and in what form it is presented.

2998

## 2999 9.2 **WireFormat**

Comment [ME73]: Issue 79

3000 A wireFormat is the form that a data structure takes when it is transmitted using some  
3001 communication binding. Another way to describe this is "the form that the data takes on the wire".  
3002 A wireFormat can be specific to a given communication method, or it may be general, applying to  
3003 many different communication methods. An example of a general wireFormat is XML text format.

3004 Where a particular SCA binding can accommodate transmitting data in more than one format, the  
3005 configuration of the binding MAY include a definition of the wireFormat to use. This is done using  
3006 an optional <sca:wireFormat/> subelement of the <binding/> element.

3007 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats  
3008 to be the default wireFormat which applies if no <wireFormat/> subelement is present.

3009 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a  
3010 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 3011 • <sca:wireFormat.xml/>
- 3012 • A wireFormat that transmits the data as an XML text datastructure
- 3013 • <sca:wireFormat.jms/>
- 3014 • The "default JMS wireFormat" as described in the JMS Binding specification

3015

3016 Specific wireFormats can have elements that include either attributes or subelements or both.

3017 For details about specific wireFormats, see the related SCA Binding specifications.

3018

3019 **9.3 OperationSelector**

3020 An operationSelector is necessary for some types of transport binding where messages are  
3021 transmitted across the transport without any explicit relationship between the message and the  
3022 interface operation to which it relates. SOAP is an example of a protocol where the messages do  
3023 contain explicit information that relates each message to the operation it targets. However, other  
3024 transport bindings have messages where this relationship is not expressed in the message or in  
3025 any related headers (pure JMS messages, for example). In cases where the messages arrive at a  
3026 service without any explicit information that maps them to specific operations, it is necessary for  
3027 the metadata attached to the service binding to contain the required mapping information. The  
3028 information is held in an operationSelector element which is a child element of the binding  
3029 element.

3030 The base sca:operationSelector element is abstract and it has no attributes and no child elements.  
3031 For a particular operationSelector, an extension subtype is defined, using substitution groups, for  
3032 example:

- 3033 • <sca:operationSelector.XPath/>
- 3034 • An operation selector that uses XPath to filter out specific messages and target them to  
3035 particular named operations.

3036 Specific operationSelectors can have elements that include either attributes or subelements or  
3037 both.

3038 For details about specific operationSelectors, see the related SCA Binding specifications.

3041 **9.4 Form of the URI of a Deployed Binding**

3042 SCA Bindings specifications can choose to use the structural URI defined in the section  
3043 "Structural URI of Components" above to derive a binding specific URI according to some Binding-  
3044 related scheme. The relevant binding specification describes this.

3045 Alternatively, <binding/> elements have an optional @URI attribute, which is termed a  
3046 bindingURI.

3047 If the bindingURI is specified on a given <binding/> element, the binding can optionally use it to  
3048 derive an endpoint URI relevant to the binding. The derivation is binding specific and is described  
3049 by the relevant binding specification.

3050 For binding.sca, which is described in the SCA Assembly specification, this is as follows:

- 3051 • If the binding uri attribute is specified on a reference, it identifies the target service in the  
3052 SCA domain by specifying the service's structural URI.
- 3053 • If the binding uri attribute is specified on a service, it is ignored.

3055 **9.4.1 Non-hierarchical URIs**

3056 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) may optionally make  
3057 use of the "uri" attribute, which is the complete representation of the URI for that service  
3058 binding. Where the binding does not use the "uri" attribute, the binding needs to offer a different  
3059 mechanism for specifying the service address.

3060 **9.4.2 Determining the URI scheme of a deployed binding**

3061 One of the things that needs to be determined when building the effective URI of a deployed  
3062 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is  
3063 binding type specific.

Comment [ME74]: Issue 79

Comment [ME75]: Issue 16

**Deleted: <#>Constructing Hierarchical URIs¶**

Bindings that use hierarchical URI schemes construct the effective URI with a combination of the following pieces:¶  
Base System URI for a scheme / Component URI / Service Binding URI¶

¶ Each of these components deserves addition definition:¶

**Base Domain URI for a scheme.** An SCA domain should define a base URI for each hierarchical URI scheme on which it intends to provide services. ¶  
For example: the HTTP and HTTPS schemes would each have their own base URI defined for the domain. An example of a scheme that is not hierarchical, and therefore will have no base URI is the "jms:" scheme.¶

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

**Service Binding URI.** The Service Binding URI is the relative URI specified in the "uri" attribute of a binding element of the service. The default value of the attribute is value of the binding's name attribute treated as a relative URI. If multiple bindings for a single service use the ... [5]

**Formatted:** Bullets and Numbering

**Deleted:** must

**Formatted:** Bullets and Numbering

3064 If the binding type supports a single protocol then there is only one URI scheme associated with it.  
3065 In this case, that URI scheme is used.

3066 If the binding type supports multiple protocols, the binding type implementation determines the  
3067 URI scheme by introspecting the binding configuration, which may include the policy sets  
3068 associated with the binding.

3069 A good example of a binding type that supports multiple protocols is `binding.ws`, which can be  
3070 configured by referencing either an "abstract" WSDL element (i.e. `portType` or `interface`) or a  
3071 "concrete" WSDL element (i.e. `binding`, `port` or `endpoint`). When the binding references a `PortType`  
3072 or `Interface`, the protocol and therefore the URI scheme is derived from the intents/policy sets  
3073 attached to the binding. When the binding references a "concrete" WSDL element, there are two  
3074 cases:

3075 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most  
3076 common case. In this case, the URI scheme is given by the protocol/transport specified in the  
3077 WSDL binding element.

3078 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,  
3079 when HTTP is specified in the `@transport` attribute of the SOAP binding element, both "http"  
3080 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined  
3081 by looking at the policy sets attached to the binding.

3082 It's worth noting that an intent supported by a binding type may completely change the behavior  
3083 of the binding. For example, when the intent "confidentiality/transport" is required by an HTTP  
3084 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to  
3085 "https".

3086

## 3087 9.5 SCA Binding

3088 The SCA binding element is defined by the following schema.

3089

3090 `<binding.sca />`

3091

3092 The SCA binding can be used for service interactions between references and services contained  
3093 within the SCA domain. The way in which this binding type is implemented is not defined by the  
3094 SCA specification and it can be implemented in different ways by different SCA runtimes. The only  
3095 requirement is that the required qualities of service must be implemented for the SCA binding  
3096 type. The SCA binding type is **not** intended to be an interoperable binding type. For  
3097 interoperability, an interoperable binding type such as the Web service binding should be used.

3098 A service definition with no binding element specified uses the SCA binding.

3099 `<binding.sca/>` would only have to be specified in override cases, or when you specify a  
3100 set of bindings on a service definition and the SCA binding should be one of them.

3101 If a reference does not have a binding, then the binding used can be any of the bindings  
3102 specified by the service provider, as long as the intents required by the reference and  
3103 the service are all respected.

3104 If the interface of the service or reference is local, then the local variant of the SCA  
3105 binding will be used. If the interface of the service or reference is remotable, then either  
3106 the local or remote variant of the SCA binding will be used depending on whether source  
3107 and target are co-located or not.

3108 If a reference specifies an URI via its `uri` attribute, then this provides the default wire to a service  
3109 provided by another domain level component. The value of the URI has to be as follows:

3110 

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

3111

## 3112 9.5.1 Example SCA Binding

3113 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
3114 containing the service element for the MyValueService and a reference element for the  
3115 StockQuoteService. Both the service and the reference use an SCA binding. The target for the  
3116 reference is left undefined in this binding and would have to be supplied by the composite in which  
3117 this composite is used.

3118

```
3119 <?xml version="1.0" encoding="ASCII"?>
3120 <!-- Binding SCA example -->
3121 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3122           targetNamespace="http://foo.com"
3123           name="MyValueComposite" >
3124
3125     <service name="MyValueService" promote="MyValueComponent">
3126       <interface.java interface="services.myvalue.MyValueService"/>
3127       <binding.sca/>
3128       ...
3129     </service>
3130
3131     ...
3132
3133     <reference name="StockQuoteService"
3134              promote="MyValueComponent/StockQuoteReference">
3135       <interface.java interface="services.stockquote.StockQuoteService"/>
3136       <binding.sca/>
3137     </reference>
3138
3139 </composite>
3140
```

## 3141 9.6 Web Service Binding

3142 SCA defines a Web services binding. This is described in [a separate specification document \[9\]](#).

3143

## 3144 9.7 JMS Binding

3145 SCA defines a JMS binding. This is described in [a separate specification document \[11\]](#).

3146  
3147  
3148  
3149  
3150  
3151  
3152  
3153  
3154  
3155  
3156  
3157  
3158  
3159  
3160  
3161  
3162  
3163  
3164  
3165  
3166  
3167  
3168  
3169  
3170  
3171  
3172  
3173  
3174  
3175  
3176  
3177  
3178  
3179  
3180

## 10 SCA Definitions

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

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

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

  <sca:intent/>*

  <sca:policySet/>*

  <sca:binding/>*

  <sca:bindingType/>*

  <sca:implementationType/>*

</definitions>
```

The definitions element has the following attribute:

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

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

**Comment [ME76]:** Issue 44

**Deleted:** All of these artifacts within an SCA Domain are defined in a global, SCA Domain-wide file named definitions.xml.

## 3181 11 Extension Model

3182

3183 The assembly model can be extended with support for new interface types, implementation types  
3184 and binding types. The extension model is based on XML schema substitution groups. There are  
3185 three XML Schema substitution group heads defined in the SCA namespace: **interface**,  
3186 **implementation** and **binding**, for interface types, implementation types and binding types,  
3187 respectively.

3188 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1],  
3189 [9], [11]) use these XML Schema substitution groups to define some basic types of interfaces,  
3190 implementations and bindings, but other types can be defined as required, where support for  
3191 these extra ones is available from the runtime. The interface type elements, implementation type  
3192 elements, and binding type elements defined by the SCA specifications are all part of the SCA  
3193 namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200712"), as indicated in their  
3194 respective schemas. New interface types, implementation types and binding types that are defined  
3195 using this extensibility model, which are not part of these SCA specifications are defined in  
3196 namespaces other than the SCA namespace.

3197 The "." notation is used in naming elements defined by the SCA specifications ( e.g.  
3198 <implementation.java ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel  
3199 extensibility approach but as a naming convention that improves usability of the SCA assembly  
3200 language.

3201

3202 **Note:** How to contribute SCA model extensions and their runtime function to an SCA runtime will  
3203 be defined by a future version of the specification.

3204

### 3205 11.1 Defining an Interface Type

3206 The following snippet shows the base definition for the **interface** element and **Interface** type  
3207 contained in **sca-core.xsd**; see appendix for complete schema.

3208

```
3209 <?xml version="1.0" encoding="UTF-8"?>  
3210 <!-- (c) Copyright SCA Collaboration 2006 -->  
3211 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3212         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3213         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3214         elementFormDefault="qualified">
```

3215

3216 ...

3217

```
3218     <element name="interface" type="sca:Interface" abstract="true"/>
```

```
3219     <complexType name="Interface"/>
```

```
3220     <complexType name="Interface" abstract="true">
```

```
3221       <attribute name="requires" type="sca:listOfQNames" use="optional"/>
```

```
3222       <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
```

```
3223     </complexType>
```

3224

Comment [mbgl77]: Issue  
39

3225  
3226  
3227  
3228

...

</schema>

3229 In the following snippet is an example of how the base definition is extended to support Java  
3230 interfaces. The snippet shows the definition of the **interface.java** element and the  
3231 **JavaInterface** type contained in **sca-interface-java.xsd**.

3232  
3233  
3234  
3235  
3236  
3237  
3238  
3239  
3240  
3241  
3242  
3243  
3244  
3245  
3246  
3247  
3248

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

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

3249 In the following snippet is an example of how the base definition can be extended by other  
3250 specifications to support a new interface not defined in the SCA specifications. The snippet shows  
3251 the definition of the **my-interface-extension** element and the **my-interface-extension-type**  
3252 type.

3253  
3254  
3255  
3256  
3257  
3258  
3259  
3260  
3261  
3262  
3263  
3264  
3265  
3266  
3267  
3268

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

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

3269 </schema>  
3270

## 3271 11.2 Defining an Implementation Type

3272 The following snippet shows the base definition for the **implementation** element and  
3273 **Implementation** type contained in *sca-core.xsd*; see appendix for complete schema.

```
3274  
3275 <?xml version="1.0" encoding="UTF-8"?>  
3276 <!-- (c) Copyright SCA Collaboration 2006 -->  
3277 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3278         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3279         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3280         elementFormDefault="qualified">  
3281  
3282     ...  
3283  
3284     <element name="implementation" type="sca:Implementation"  
3285     abstract="true"/>  
3286     <complexType name="Implementation"/>  
3287  
3288     ...  
3289 </schema>
```

3291  
3292 In the following snippet we show how the base definition is extended to support Java  
3293 implementation. The snippet shows the definition of the **implementation.java** element and the  
3294 **JavaImplementation** type contained in *sca-implementation-java.xsd*.

```
3295  
3296 <?xml version="1.0" encoding="UTF-8"?>  
3297 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3298         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3299         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3300  
3301     <element name="implementation.java" type="sca:JavaImplementation"  
3302             substitutionGroup="sca:implementation"/>  
3303     <complexType name="JavaImplementation">  
3304         <complexContent>  
3305             <extension base="sca:Implementation">  
3306                 <attribute name="class" type="NCName"  
3307                     use="required"/>  
3308             </extension>  
3309         </complexContent>  
3310     </complexType>  
3311 </schema>
```

3312 In the following snippet is an example of how the base definition can be extended by other  
3313 specifications to support a new implementation type not defined in the SCA specifications. The  
3314 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**  
3315 **type** type.

```
3316 <?xml version="1.0" encoding="UTF-8"?>  
3317 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3318     targetNamespace="http://www.example.org/myextension"  
3319     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3320     xmlns:tns="http://www.example.org/myextension">  
3321  
3322     <element name="my-impl-extension" type="tns:my-impl-extension-type"  
3323         substitutionGroup="sca:implementation"/>  
3324     <complexType name="my-impl-extension-type">  
3325         <complexContent>  
3326             <extension base="sca:Implementation">  
3327                 ...  
3328             </extension>  
3329         </complexContent>  
3330     </complexType>  
3331 </schema>  
3332
```

3333 In addition to the definition for the new implementation instance element, there needs to be an  
3334 associated implementationType element which provides metadata about the new implementation  
3335 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3336 <implementationType type="xs:QName"  
3337     alwaysProvides="list of intent xs:QName"  
3338     mayProvide="list of intent xs:QName"/>  
3339
```

3340 The implementation type has the following attributes:

- 3341 • **type (1..1)** – the type of the implementation to which this implementationType element  
3342 applies. This is intended to be the QName of the implementation element for the  
3343 implementation type, such as "sca:implementation.java"
  - 3344 • **alwaysProvides (0..1)** – a set of intents which the implementation type always  
3345 provides. See [the Policy Framework specification \[10\]](#) for details.
  - 3346 • **mayProvide (0..1)** – a set of intents which the implementation type may provide. See  
3347 [the Policy Framework specification \[10\]](#) for details.
- 3348

## 3349 11.3 Defining a Binding Type

3350 The following snippet shows the base definition for the **binding** element and **Binding** type  
3351 contained in **sca-core.xsd**; see appendix for complete schema.

```
3352  
3353 <?xml version="1.0" encoding="UTF-8"?>  
3354 <!-- binding type schema snippet -->  
3355 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

```

3356 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3357         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3358         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3359         elementFormDefault="qualified">
3360
3361     ...
3362
3363     <element name="binding" type="sca:Binding" abstract="true"/>
3364     <complexType name="Binding">
3365         <attribute name="uri" type="anyURI" use="optional"/>
3366         <attribute name="name" type="NCName" use="optional"/>
3367         <attribute name="requires" type="sca:listOfQNames"
3368             use="optional"/>
3369         <attribute name="policySets" type="sca:listOfQNames"
3370             use="optional"/>
3371     </complexType>
3372
3373     ...
3374
3375 </schema>

```

3376 In the following snippet is an example of how the base definition is extended to support Web  
3377 service binding. The snippet shows the definition of the **binding.ws** element and the  
3378 **WebServiceBinding** type contained in **sca-binding-webservice.xsd**.

```

3379
3380 <?xml version="1.0" encoding="UTF-8"?>
3381 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3382         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3383         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3384
3385     <element name="binding.ws" type="sca:WebServiceBinding"
3386         substitutionGroup="sca:binding"/>
3387     <complexType name="WebServiceBinding">
3388         <complexContent>
3389             <extension base="sca:Binding">
3390                 <attribute name="port" type="anyURI" use="required"/>
3391             </extension>
3392         </complexContent>
3393     </complexType>
3394 </schema>

```

3395 In the following snippet is an example of how the base definition can be extended by other  
3396 specifications to support a new binding not defined in the SCA specifications. The snippet shows  
3397 the definition of the **my-binding-extension** element and the **my-binding-extension-type** type.

```

3398 <?xml version="1.0" encoding="UTF-8"?>
3399 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

3400         targetNamespace="http://www.example.org/myextension"
3401         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3402         xmlns:tns="http://www.example.org/myextension">
3403
3404         <element name="my-binding-extension"
3405             type="tns:my-binding-extension-type"
3406             substitutionGroup="sca:binding"/>
3407         <complexType name="my-binding-extension-type">
3408             <complexContent>
3409                 <extension base="sca:Binding">
3410                     ...
3411                 </extension>
3412             </complexContent>
3413         </complexType>
3414     </schema>
3415

```

3416 In addition to the definition for the new binding instance element, there needs to be an associated  
3417 bindingType element which provides metadata about the new binding type. The pseudo schema  
3418 for the bindingType element is shown in the following snippet:

```

3419 <bindingType type="xs:QName"
3420             alwaysProvides="list of intent QNames"?
3421             mayProvide = "list of intent QNames"?/>
3422

```

3423 The binding type has the following attributes:

- 3424 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3425 intended to be the QName of the binding element for the binding type, such as  
3426 "sca:binding.ws"
- 3427 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3428 [the Policy Framework specification \[10\]](#) for details.
- 3429 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)  
3430 [Policy Framework specification \[10\]](#) for details.

## 3431 11.4 Defining an Import Type

3432 The following snippet shows the base definition for the *import* element and *Import* type contained in *sca-*  
3433 *core.xsd*; see appendix for complete schema.

```

3434
3435 <?xml version="1.0" encoding="UTF-8"?>
3436 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3437 IPR and other policies apply. -->
3438 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3439         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3440         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3441         elementFormDefault="qualified">
3442     ...
3443
3444     <!-- Import -->
3445     <element name="importBase" type="sca:Import" abstract="true" />
3446

```

```

3447 <complexType name="Import" abstract="true">
3448   <complexContent>
3449     <extension base="sca:CommonExtensionBase">
3450       <sequence>
3451         <any namespace="##other" processContents="lax" minOccurs="0"
3452           maxOccurs="unbounded"/>
3453       </sequence>
3454     </extension>
3455   </complexContent>
3456 </complexType>
3457
3458 <element name="import" type="sca:ImportType"
3459   substitutionGroup="sca:importBase"/>
3460 <complexType name="ImportType">
3461   <complexContent>
3462     <extension base="sca:Import">
3463       <attribute name="namespace" type="string" use="required"/>
3464       <attribute name="location" type="anyURI" use="required"/>
3465     </extension>
3466   </complexContent>
3467 </complexType>
3468
3469 ...
3470
3471 </schema>
3472

```

3473 In the following snippet we show how the base import definition is extended to support Java imports. In  
3474 the import element, the namespace is expected to be an XML namespace, an `import.java` element uses a  
3475 Java package name instead. The snippet shows the definition of the ***import.java*** element and the  
3476 ***JavaImportType*** type contained in ***sca-import-java.xsd***.

```

3477
3478 <?xml version="1.0" encoding="UTF-8"?>
3479 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3480   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3481   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3482
3483   <element name="import.java" type="sca:JavaImportType"
3484     substitutionGroup="sca:importBase"/>
3485   <complexType name="JavaImportType">
3486     <complexContent>
3487       <extension base="sca:Import">
3488         <attribute name="package" type="xs:String" use="required"/>
3489         <attribute name="location" type="xs:AnyURI" use="optional"/>
3490       </extension>
3491     </complexContent>
3492   </complexType>
3493 </schema>
3494

```

3495 In the following snippet we show an example of how the base definition can be extended by other  
3496 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3497 definition of the ***my-import-extension*** element and the ***my-import-extension-type*** type.

```

3498
3499 <?xml version="1.0" encoding="UTF-8"?>
3500 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3501   targetNamespace="http://www.example.org/myextension"

```

```

3502     xmlns:sca=" http://docs.oasis-open.org/ns/opencsa/sca/200712"
3503     xmlns:tns="http://www.example.org/myextension">
3504
3505     <element name="my-import-extension"
3506         type="tns:my-import-extension-type"
3507         substitutionGroup="sca:importBase"/>
3508     <complexType name="my-import-extension-type">
3509         <complexContent>
3510             <extension base="sca:Import">
3511                 ...
3512             </extension>
3513         </complexContent>
3514     </complexType>
3515 </schema>

```

3517 For a complete example using this extension point, see the definition of *import.java* in the SCA Java  
3518 Common Annotations and APIs Specification [SCA-Java].

**Comment [BA78]:** Similar pointers should be used in the remainder of section 10 instead of the full Java extension definitions. Or alternatively use `interface.wSDL` and `implementation.composite`

## 3519 11.5 Defining an Export Type

3520 The following snippet shows the base definition for the *export* element and *ExportType* type contained in  
3521 *sca-core.xsd*; see appendix for complete schema.

```

3522
3523 <?xml version="1.0" encoding="UTF-8"?>
3524 <!-- Copyright (C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3525 IPR and other policies apply. -->
3526 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3527     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3528     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3529     elementFormDefault="qualified">
3530
3531     ...
3532     <!-- Export -->
3533     <element name="exportBase" type="sca:Export" abstract="true" />
3534     <complexType name="Export" abstract="true">
3535         <complexContent>
3536             <extension base="sca:CommonExtensionBase">
3537                 <sequence>
3538                     <any namespace="##other" processContents="lax" minOccurs="0"
3539                         maxOccurs="unbounded" />
3540                 </sequence>
3541             </extension>
3542         </complexContent>
3543     </complexType>
3544
3545     <element name="export" type="sca:ExportType"
3546         substitutionGroup="sca:exportBase" />
3547     <complexType name="ExportType">
3548         <complexContent>
3549             <extension base="sca:Export">
3550                 <attribute name="namespace" type="string" use="required" />
3551             </extension>
3552         </complexContent>
3553     </complexType>
3554     ...
3555 </schema>

```

3556

3557 The following snippet shows how the base definition is extended to support Java exports. In a base  
3558 *export* element, the *@namespace* attribute specifies XML namespace being exported. An *export.java*  
3559 element uses a *@package* attribute to specify the Java package to be exported. The snippet shows the  
3560 definition of the *export.java* element and the *JavaExport* type contained in *sca-export-java.xsd*.

3561

```
3562 <?xml version="1.0" encoding="UTF-8"?>
3563 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3564         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3565         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3566
3567     <element name="export.java" type="sca:JavaExportType"
3568             substitutionGroup="sca:exportBase"/>
3569     <complexType name="JavaExportType">
3570         <complexContent>
3571             <extension base="sca:Export">
3572                 <attribute name="package" type="xs:String" use="required"/>
3573             </extension>
3574         </complexContent>
3575     </complexType>
3576 </schema>
```

3577

3578 In the following snippet we show an example of how the base definition can be extended by other  
3579 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3580 definition of the *my-export-extension* element and the *my-export-extension-type* type.

3581

```
3582 <?xml version="1.0" encoding="UTF-8"?>
3583 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3584         targetNamespace="http://www.example.org/myextension"
3585         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200712"
3586         xmlns:tns="http://www.example.org/myextension">
3587
3588     <element name="my-export-extension"
3589             type="tns:my-export-extension-type"
3590             substitutionGroup="sca:exportBase"/>
3591     <complexType name="my-export-extension-type">
3592         <complexContent>
3593             <extension base="sca:Export">
3594                 ...
3595             </extension>
3596         </complexContent>
3597     </complexType>
3598 </schema>
```

3599

3600 For a complete example using this extension point, see the definition of *export.java* in the SCA Java  
3601 Common Annotations and APIs Specification [SCA-Java].

3602

---

## 3603 12 Packaging and Deployment

### 3604 12.1 Domains

3605 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3606 of interconnected runtime nodes.

3607 A single SCA domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3608 wires can only be used to connect components within a single SCA domain. Connections to  
3609 services outside the domain must use binding specific mechanisms for addressing services (such  
3610 as WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used  
3611 in the context of a single domain. In general, external clients of a service that is developed and  
3612 deployed using SCA should not be able to tell that SCA was used to implement the service – it is  
3613 an implementation detail.

3614 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3615 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3616 functionality controlled by a single organization. For example, an SCA Domain may be the whole  
3617 of a business, or it may be a department within a business.

3618 As an example, for the accounts department in a business, the SCA Domain might cover all  
3619 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3620 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3621 An SCA domain has the following:

- 3622 • A virtual domain-level composite whose components are deployed and running
- 3623 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3624 necessary to execute components
- 3625 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3626 level composite.

3627 The information associated with an SCA domain can be stored in many ways, including but not  
3628 limited to a specific filesystem structure or a repository.

### 3629 12.2 Contributions

3630 An SCA domain might require a large number of different artifacts in order to work. These  
3631 artifacts include artifacts defined by SCA and other artifacts such as object code files and interface  
3632 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3633 different SCA definition documents are: composite, componentType, constrainingType and  
3634 definitions. XML artifacts that are not defined by SCA but which may be needed by an SCA  
3635 domain include XML Schema documents, WSDL documents, and BPEL documents. SCA  
3636 constructs, like other XML-defined constructs, use XML qualified names for their identity (i.e.  
3637 namespace + local name).

3638 Non-XML artifacts are also required within an SCA domain. The most obvious examples of such  
3639 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3640 implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3641 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3642 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3643 packaging formats, but requires that the ZIP format be supported. When using the ZIP format for  
3644 deploying a contribution, this specification does not specify whether that format is retained after  
3645 deployment. For example, a Java EE based SCA runtime may convert the ZIP package to an EAR  
3646 package. SCA expects certain characteristics of any packaging:

- 3647 • For any contribution packaging it MUST be possible to present the artifacts of the  
3648 packaging to SCA as a hierarchy of resources based off of a single root [ASM12001]

- 3649 • Within any contribution packaging A directory resource SHOULD exist at the root of the  
3650 hierarchy named META-INF [ASM12002]
- 3651 • Within any contribution packaging a document SHOULD exist directly under the META-INF  
3652 directory named sca-contribution.xml which lists the SCA Composites within the  
3653 contribution that are runnable. [ASM12003]  
3654  
3655 The same document also optionally lists namespaces of constructs that are defined within  
3656 the contribution and which may be used by other contributions  
3657 Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the  
3658 namespaces of constructs that are needed by the contribution and which are be found  
3659 elsewhere, for example in other contributions. [ASM12004] These optional elements may  
3660 not be physically present in the packaging, but may be generated based on the definitions  
3661 and references that are present, or they may not exist at all if there are no unresolved  
3662 references.  
3663  
3664 See the section "SCA Contribution Metadata Document" for details of the format of this  
3665 file.

3666 To illustrate that a variety of packaging formats can be used with SCA, the following are examples  
3667 of formats that might be used to package SCA artifacts and metadata (as well as other artifacts)  
3668 as a contribution:

- 3669 • A filesystem directory
- 3670 • An OSGi bundle
- 3671 • A compressed directory (zip, gzip, etc)
- 3672 • A JAR file (or its variants – WAR, EAR, etc)

3673 Contributions do not contain other contributions. If the packaging format is a JAR file that  
3674 contains other JAR files (or any similar nesting of other technologies), the internal files are not  
3675 treated as separate SCA contributions. It is up to the implementation to determine whether the  
3676 internal JAR file should be represented as a single artifact in the contribution hierarchy or whether  
3677 all of the contents should be represented as separate artifacts.

3678 A goal of SCA's approach to deployment is that the contents of a contribution should not need to  
3679 be modified in order to install and use the contents of the contribution in a domain.

3680

## 3681 12.2.1 SCA Artifact Resolution

3682 Contributions can be self-contained, in that all of the artifacts necessary to run the contents of the  
3683 contribution are found within the contribution itself. However, it can also be the case that the  
3684 contents of the contribution make one or many references to artifacts that are not contained  
3685 within the contribution. These references can be to SCA artifacts **such as composites** or they can  
3686 be to other artifacts such as WSDL files, XSD files or to code artifacts such as Java class files and  
3687 BPEL **process files**. Note: This form of artifact resolution does not apply to imports of composite  
3688 files, as described in Section 6.6.

Deleted: may

Deleted: scripts.

3689 A contribution can use some artifact-related or packaging-related means to resolve artifact  
3690 references. Examples of such mechanisms include:

- 3691 • wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema  
3692 artifacts respectively
- 3693 • OSGi bundle mechanisms for resolving Java class and related resource dependencies

3694 Where present, these mechanisms MUST be used by the SCA runtime to resolve artifact  
3695 dependencies. Where present, artifact-related or packaging-related mechanisms MUST be used to  
3696 resolve artifact dependencies. [ASM12005] The SCA runtime MUST raise an error if an artifact  
3697 cannot be resolved using these mechanisms, if present. [ASM12021]

Deleted: Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact dependencies.

3698

3699 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanism is  
3700 used either where no other mechanisms are available, for example in cases where the  
3701 mechanisms used by the various contributions in the same SCA Domain are different. An example  
3702 of the latter case is where an OSGi Bundle is used for one contribution but where a second  
3703 contribution used by the first one is not implemented using OSGi - eg the second contribution  
3704 relates to a mainframe COBOL service whose interfaces are declared using a WSDL which must be  
3705 accessed by the first contribution.

Deleted: are

Deleted: or

Deleted: is

3706 The SCA artifact resolution is likely to be most useful for SCA domains containing heterogeneous  
3707 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to  
3708 work across different kinds of contribution.

3709 SCA artifact resolution works on the principle that a contribution which needs to use artifacts  
3710 defined elsewhere expresses these dependencies using **import** statements in metadata belonging  
3711 to the contribution. A contribution controls which artifacts it makes available to other  
3712 contributions through **export** statements in metadata attached to the contribution. SCA artifact  
3713 resolution is a general mechanism that can be extended for the handling of specific types of  
3714 artifact. The general mechanism that is described in the following paragraphs is mainly intended  
3715 for the handling of XML artifacts. Other types of artifacts, for example Java classes, use an  
3716 extended version of artifact resolution that is specialized to their nature (eg. instead of  
3717 "namespaces", Java uses "packages"). Descriptions of these more specialized forms of artifact  
3718 resolution are contained in the SCA specifications that deal with those artifact types.

3719 Import and export statements for XML artifacts work at the level of namespaces - so that an  
3720 import statement declares that artifacts from a specified namespace are found in other  
3721 contributions, while an export statement makes all the artifacts from a specified namespace  
3722 available to other contributions.

3723 An import declaration can simply specify the namespace to import. In this case, the locations  
3724 which are searched for artifacts in that namespace are the contribution(s) in the Domain which  
3725 have export declarations for the same namespace, if any. Alternatively an import declaration can  
3726 specify a location from which artifacts for the namespace are obtained, in which case, that specific  
3727 location is searched. There can be multiple import declarations for a given namespace. Where  
3728 multiple import declarations are made for the same namespace, all the locations specified MUST  
3729 be searched in lexical order. [ASM12022]

Formatted: Font color: Red

3730 For an XML namespace, artifacts can be declared in multiple locations - for example a given  
3731 namespace can have a WSDL declared in one contribution and have an XSD defining XML data  
3732 types in a second contribution.

3733 If the same artifact is declared in multiple locations, this is not an error. The first location as  
3734 defined by lexical order is chosen. If no locations are specified no order exists and the one chosen  
3735 is implementation dependent.

3736 When a contribution contains a reference to an artifact from a namespace that is declared in an import  
3737 statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the  
3738 SCA runtime MUST resolve artifacts in the following order:

3739 1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT  
3740 be searched recursively in order to locate artifacts (ie only a one-level search is performed).

3741 2. from the contents of the contribution itself. [ASM12023]

Formatted: Indent: First line:  
0.63 cm

Formatted: Bullets and  
Numbering

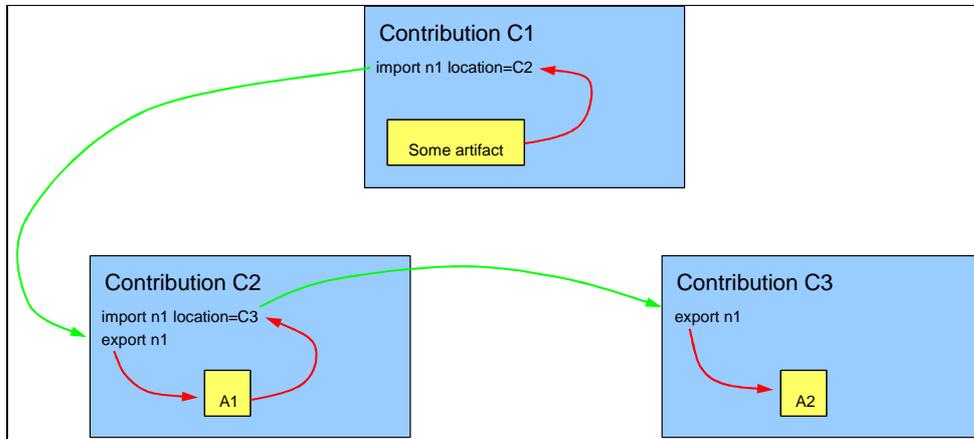
Formatted: Font color: Red

3742 When a contribution uses an artifact contained in another contribution through SCA artifact  
3743 resolution, if that artifact itself has dependencies on other artifacts, the SCA runtime MUST resolve  
3744 these dependencies in the context of the contribution containing the artifact, not in the context of  
3745 the original contribution. [ASM12024]

3746 For example:

- 3747
- 3748 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the  
"n1" namespace from a second contribution "C2".
  - 3749 • in contribution "C2" the artifact "A1" in the "n1" namespace references an artifact "A2"  
3750 also in the "n1" namespace, which is resolved through an import of the "n1" namespace  
3751 in "C2" which specifies the location "C3".

3752



Formatted: Keep with next

3753

3754

Figure 14: Example of SCA Artifact Resolution between Contributions

Formatted: Caption

3755

3756

3757

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

3758

3759

3760

3761

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

3762

3763

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

Formatted: Font color: Red

3764

3765

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

Formatted: Font color: Red

3766

3767

### 12.2.2 SCA Contribution Metadata Document

3768

3769

3770

3771

3772

3773

3774

3775

3776

3777

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

The format of the document is:

3778

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

3779

```
<!-- sca-contribution pseudo-schema -->
```

3780

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

3781

3782

```
  <deployable composite="xs:QName"/>*
```

3783

```
  <import namespace="xs:String" location="xs:AnyURI"?/>*
```

3784

```
  <export namespace="xs:String"/>*
```

3785

3786 </contribution>

3787

3788 **deployable element:** Identifies a composite which is a composite within the contribution that is a  
3789 composite intended for potential inclusion into the virtual domain-level composite. Other  
3790 composites in the contribution are not intended for inclusion but only for use by other composites.  
3791 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)  
3792 [Composite](#) capability and the add To Domain Level Composite capability.

3793 Attributes of the deployable element:

- 3794 • **composite (1..1)** – The QName of a composite within the contribution.

3795

3796 **Export element:** A declaration that artifacts belonging to a particular namespace are exported  
3797 and are available for use within other contributions. An export declaration in a contribution  
3798 specifies a namespace, all of whose definitions are considered to be exported. By default,  
3799 definitions are not exported.

3800 The SCA artifact export is useful for SCA domains containing heterogeneous mixtures of  
3801 contribution packagings and technologies, where artifact-related or packaging-related mechanisms  
3802 are unlikely to work across different kinds of contribution.

3803 Attributes of the export element:

- 3804 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3805 should be the namespace URI for the exported definitions. For XML technologies that  
3806 define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port  
3807 types are a different symbol space from WSDL bindings), all definitions from all symbol  
3808 spaces are exported.

3809 Technologies that use naming schemes other than QNames must use a different export  
3810 element from the same substitution group as the the SCA <export> element. The  
3811 element used identifies the technology, and can use any value for the namespace that is  
3812 appropriate for that technology. For example, <export.java> can be used can be used to  
3813 export java definitions, in which case the namespace is a fully qualified package name.  
3814

3815

3816 **Import element:** Import declarations specify namespaces of definitions that are needed by the  
3817 definitions and implementations within the contribution, but which are not present in the  
3818 contribution. It is expected that in most cases import declarations will be generated based on  
3819 introspection of the contents of the contribution. In this case, the import declarations would be  
3820 found in the META-INF/ sca-contribution-generated.xml document.

3821 Attributes of the import element:

- 3822 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3823 is the namespace URI for the imported definitions. For XML technologies that define  
3824 multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are  
3825 a different symbol space from WSDL bindings), all definitions from all symbol spaces are  
3826 imported.

3827 Technologies that use naming schemes other than QNames must use a different import  
3828 element from the same substitution group as the the SCA <import> element. The  
3829 element used identifies the technology, and can use any value for the namespace that is  
3830 appropriate for that technology. For example, <import.java> can be used can be used to  
3831 import java definitions, in which case the namespace is a fully qualified package name.  
3832

- 3833 • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific  
3834 requirements for the form of this URI, nor the means by which it is resolved. It can point  
3835 to another contribution (through its URI) or it can point to some location entirely outside  
3836 the SCA Domain.

3837

3838 It is expected that SCA runtimes can define implementation specific ways of resolving location  
3839 information for artifact resolution between contributions. These mechanisms will however usually  
3840 be limited to sets of contributions of one runtime technology and one hosting environment.

3841 In order to accommodate imports of artifacts between contributions of disparate runtime  
3842 technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location  
3843 specification.

3844 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are  
3845 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact  
3846 location specifications.

3847 The order in which the import statements are specified can play a role in this mechanism. Since  
3848 definitions of one namespace can be distributed across several artifacts, multiple import  
3849 declarations can be made for one namespace.  
3850

3851 The location value is only a default, and dependent contributions listed in the call to  
3852 installContribution can override the value if there is a conflict. However, the specific mechanism  
3853 for resolving conflicts between contributions that define conflicting definitions is implementation  
3854 specific.

3855  
3856 If the value of the location attribute is an SCA contribution URI, then the contribution packaging  
3857 can become dependent on the deployment environment. In order to avoid such a dependency,  
3858 dependent contributions should be specified only when deploying or updating contributions as  
3859 specified in the section 'Operations for Contributions' below.

### 3860 12.2.3 Contribution Packaging using ZIP

3861 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires  
3862 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This  
3863 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3864 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3865 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-  
3866 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such  
3867 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3868 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)  
3869 [.ZIP file format \[12\]](#).

3870

## 3871 12.3 Installed Contribution

3872 As noted in the section above, the contents of a contribution do not need to be modified in order  
3873 to install and use it within a domain. An *installed contribution* is a contribution with all of the  
3874 associated information necessary in order to execute *deployable composites* within the  
3875 contribution.

3876 An installed contribution is made up of the following things:

- 3877 • Contribution Packaging – the contribution that will be used as the starting point for  
3878 resolving all references
- 3879 • Contribution base URI
- 3880 • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3881 the import statements from the root composite and from other dependent contributions
  - 3882 ○ Dependent contributions might or might not be shared with other installed  
3883 contributions.
  - 3884 ○ When the snapshot of any contribution is taken is implementation defined, ranging  
3885 from the time the contribution is installed to the time of execution

- 3886
- Deployment-time composites.
- 3887 These are composites that are added into an installed contribution after it has been  
3888 deployed. This makes it possible to provide final configuration and access to  
3889 implementations within a contribution without having to modify the contribution. These  
3890 are optional, as composites that already exist within the contribution can also be used for  
3891 deployment.

3892

3893 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3894 fully qualified class names in Java).

3895 If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3896 algorithm used to determine the qualified name to use is implementation dependent.  
3897 Implementations of SCA MAY also generate an error if there are conflicting names exported from  
3898 multiple contributions. [ASM12007]

3899

### 3900 12.3.1 Installed Artifact URIs

3901 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3902 constructed by starting with the base URI of the contribution and adding the relative URI of each  
3903 artifact (recalling that SCA requires that any packaging format be able to offer up its artifacts in a  
3904 single hierarchy).

3905

## 3906 12.4 Operations for Contributions

3907 SCA Domains provide the following conceptual functionality associated with contributions  
3908 (meaning the function might not be represented as addressable services and also meaning that  
3909 equivalent functionality might be provided in other ways). The functionality is optional meaning  
3910 that some SCA runtimes MAY choose not to provide the contribution functions functionality in any  
3911 way. [ASM12008]

### 3912 12.4.1 install Contribution & update Contribution

3913 Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3914 supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3915 contributions that should be used to resolve the dependencies of the root contribution and other  
3916 dependent contributions. These override any dependent contributions explicitly listed via the  
3917 location attribute in the import statements of the contribution.

3918 SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3919 means that all other exported artifacts can be used from that contribution. Because of this, the  
3920 dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3921 what is being used from each one.

3922 Each dependent contribution is also an installed contribution, with its own dependent  
3923 contributions. By default these dependent contributions of the dependent contributions (which we  
3924 will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3925 contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3926 definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3927 included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3928 Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3929 MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3930 Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3931 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,  
3932 then it is easy for this list to be generated by tooling.

## 3933 12.4.2 add Deployment Composite & update Deployment Composite

3934 Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3935 data structure, not an existing resource in the domain) to the contribution identified by a supplied  
3936 contribution URI. The added or updated deployment composite is given a relative URI that  
3937 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3938 must run within the context of a installed contribution (any component implementations or other  
3939 definitions are resolved within that contribution), this functionality makes it possible for the  
3940 deployer to create a composite with final configuration and wiring decisions and add it to an  
3941 installed contribution without having to modify the contents of the root contribution.

3942 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3943 It is then possible for those to be given component names by a (possibly generated) composite  
3944 that is added into the installed contribution, without having to modify the packaging.

## 3945 12.4.3 remove Contribution

3946 Removes the deployed contribution identified by a supplied contribution URI.

3947

## 3948 12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3949

3950 For certain types of artifact, there are existing and commonly used mechanisms for referencing a  
3951 specific concrete location where the artifact can be resolved.

3952 Examples of these mechanisms include:

- 3953 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the  
3954 place holding the WSDL itself.
- 3955 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a  
3956 URI where the XSD is found.

3957 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does  
3958 not have to be dereferenced.

3959 SCA permits the use of these mechanisms. Where present, non-SCA artifact resolution  
3960 mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.  
3961 [ASM12010] However, use of these mechanisms is discouraged because tying assemblies to  
3962 addresses in this way makes the assemblies less flexible and prone to errors when changes are  
3963 made to the overall SCA Domain.

3964 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to  
3965 find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the  
3966 SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an  
3967 alternative. [ASM12011]

3968

## 3969 12.6 Domain-Level Composite

3970 The domain-level composite is a virtual composite, in that it is not defined by a composite  
3971 definition document. Rather, it is built up and modified through operations on the domain.  
3972 However, in other respects it is very much like a composite, since it contains components, wires,  
3973 services and references.

3974

3975 The value of @autowire for the logical domain composite MUST be autowire="false". [ASM12012]

3976

3977 For components at the Domain level, with References for which @autowire="true" applies, the  
3978 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:  
3979 1) The SCA runtime MAY disallow deployment of any components with autowire References. In  
3980 this case, the SCA runtime MUST generate an exception at the point where the component is  
3981 deployed.  
3982 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component  
3983 is deployed and not update those targets when later deployment actions occur.  
3984 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later  
3985 deployment actions occur resulting in updated reference targets which match the new Domain  
3986 configuration. How the new configuration of the reference takes place is described by the relevant  
3987 client and implementation specifications.  
3988 [ASM12013]

3989 The abstract domain-level functionality for modifying the domain-level composite is as follows,  
3990 although a runtime may supply equivalent functionality in a different form:

### 3991 12.6.1 add To Domain-Level Composite

3992 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.  
3993 The supplied composite URI must refer to a composite within a installed contribution. The  
3994 composite's installed contribution determines how the composite's artifacts are resolved (directly  
3995 and indirectly). The supplied composite is added to the domain composite with semantics that  
3996 correspond to the domain-level composite having an <include> statement that references the  
3997 supplied composite. All of the composite's components become *top-level* components and the  
3998 services become externally visible services (eg. they would be present in a WSDL description of  
3999 the domain).

### 4000 12.6.2 remove From Domain-Level Composite

4001 Removes from the Domain Level composite the elements corresponding to the composite  
4002 identified by a supplied composite URI. This means that the removal of the components, wires,  
4003 services and references originally added to the domain level composite by the identified  
4004 composite.

### 4005 12.6.3 get Domain-Level Composite

4006 Returns a <composite> definition that has an <include> line for each composite that had been  
4007 added to the domain level composite. It is important to note that, in dereferencing the included  
4008 composites, any referenced artifacts must be resolved in terms of that installed composite.

### 4009 12.6.4 get QName Definition

4010 In order to make sense of the domain-level composite (as returned by get Domain-Level  
4011 Composite), it must be possible to get the definitions for named artifacts in the included  
4012 composites. This functionality takes the supplied URI of an installed contribution (which provides  
4013 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
4014 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for  
4015 that definition type.

4016 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
4017 should exist in some form, but not necessarily as a service operation with exactly this signature.

## 4018 12.7 Dynamic Behaviour of Wires in the SCA Domain

4019 For components with references which are at the Domain level, there is the potential for dynamic  
4020 behaviour when the wires for a component reference change (this can only apply to component  
4021 references at the Domain level and not to components within composites used as implementations):

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

Deleted: For components at the Domain level, with References for which @autowire="true" applies, the behaviour of the SCA runtime MUST take ONE of the 3 following forms:¶ 1) The SCA runtime MAY disallow deployment of any components with autowire References. In this case, the SCA runtime MUST generate an exception at the point where the component is deployed.¶ 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component is deployed and not update those targets when later deployment actions occur. ¶ 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later deployment actions occur resulting in updated reference targets which match the new Domain configuration. How the new configuration of the reference takes place is described by the relevant client and implementation specifications.

Comment [ME79]: Issue 41

4022 The configuration of the wires for a component reference of a component at the Domain level can change  
4023 by means of deployment actions:

- 4024 1. <wire/> elements can be added, removed or replaced by deployment actions  
4025 2. Components can be updated by deployment actions (ie this may change the component reference  
4026 configuration)  
4027 3. Components which are the targets of reference wires can be updated or removed  
4028 4. Components can be added that are potential targets for references which are marked with  
4029 @autowire=true  
4030

4031 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose  
4032 references are affected by those deployment actions MAY have their references updated by the SCA  
4033 runtime dynamically without the need to stop and start those components. [ASM12014]

4034 Where components are updated by deployment actions (their configuration is changed in some way,  
4035 which may include changing the wires of component references), the new configuration MUST apply to all  
4036 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY  
4037 choose to maintain existing instances with the old configuration of components updated by deployment  
4038 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.  
4039 [ASM12016]

4040 Where a component that is the target of a wire is removed, without the wire being changed, then future  
4041 invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is  
4042 the result of the autowire process, the SCA runtime MUST:

- 4043
  - either cause future invocation of the target component's services to fail with a  
4044 ServiceUnavailable fault
  - or alternatively, if an alternative target component is available that satisfies the autowire  
4045 process, update the reference of the source component [ASM12017]

← - - - Formatted: Bullets and  
Numbering

4047 Where a component that is the target of a wire is updated, future invocations of that reference SHOULD  
4048 use the updated component. [ASM12018] Where an existing domain level component is updated, an  
4049 SCA runtime MAY maintain a copy of a component offering a conversational service until all existing  
4050 conversations complete - alternatively all existing conversations MAY be terminated. [ASM12019]

4051 Where a component is added to the domain that is a potential target for a domain level component  
4052 reference where that reference is marked as @autowire=true, the SCA runtime MUST:

- 4053
  - either update the references for the source component once the new component is running,
  - or alternatively, defer the updating of the references of the source component until the source  
4054 component is stopped and restarted. [ASM12020]

← - - - Formatted: Bullets and  
Numbering

4056

## 4057 12.8 Dynamic Behaviour of Component Property Values

4058 For a domain level component with a Property whose value is obtained from a Domain-level Property  
4059 through the use of the @source attribute, if the domain level property is updated by means of deployment  
4060 actions, the SCA runtime MUST

- 4061
  - either update the property value of the domain level component, once the update of the domain  
4062 property is complete
  - or alternative defer the updating of the component property value until the compoennt is stopped  
4063 and restarted

4065

4066  
4067  
4068  
4069  
4070  
4071

## 13 Conformance

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

An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.  
[ASM13001]

**Deleted:** An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema

**Comment [ME80]:** Issue 72

**Formatted:** Font color: Red

**Deleted:** [ASM10001]

```
Deleted: <#>Pseudo Schema
<#>ComponentType
<?xml version="1.0"
encoding="ASCII"?>
<!-- Component type
schema snippet -->
<componentType
xmlns="http://docs.oasis-
open.org/ns/opencsa/sca
/200712"
.constrainingType="QName"? >
<service
name="xs:NCName"
requires="list of
xs:QName"?
..policySets="list of
xs:QName"?*
..<interface ... />
..<binding
uri="xs:anyURI"?
name="xs:NCName"?
requires="list of
xs:QName"?
..policySets="list of
xs:QName"?/*
..<callback?>
..<binding ... />+
</callback>
</service>
<reference
name="xs:NCName"
target="list of
xs:anyURI"?
autowire="xs:boolean"?
..multiplicity="0..1 or
1..1 or 0..n or 1..n"?
..wiredByImpl="xs:boolean"?
requires="list of
xs:QName"?
..policySets="list of
xs:QName"?*
..<interface ... />
..<binding
uri="xs:anyURI"?
name="xs:NCName"?
requires="list of
xs:QName"?
..policySets="list of
xs:QName"?/*
..<callback?>
..<binding ... />+
</callback>
</reference>
<property
name="xs:NCName"
(type="xs:QName" |
element="xs:QName" ... [6]
```

4072

## A. XML Schemas

### 4073 A.1 sca.xsd

4074

```
4075 <?xml version="1.0" encoding="UTF-8"?>
4076 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4077 IPR and other policies apply. -->
4078 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4079   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4080   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
4081
4082   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4083
4084   <include schemaLocation="sca-interface-java-1.1-schema-200803.xsd"/>
4085   <include schemaLocation="sca-interface-wsdl-1.1-schema-200803.xsd"/>
4086   <include schemaLocation="sca-interface-cpp-1.1-schema-200803.xsd"/>
4087   <include schemaLocation="sca-interface-c-1.1-schema-200803.xsd"/>
4088
4089   <include schemaLocation="sca-implementation-java-1.1-schema-200803.xsd"/>
4090   <include schemaLocation=
4091     "sca-implementation-composite-1.1-schema-200803.xsd"/>
4092   <include schemaLocation="sca-implementation-cpp-1.1-schema-200803.xsd"/>
4093   <include schemaLocation="sca-implementation-c-1.1-schema-200803.xsd"/>
4094   <include schemaLocation="sca-implementation-bpel-1.1-schema-200803.xsd"/>
4095
4096   <include schemaLocation="sca-binding-webservice-1.1-schema-200803.xsd"/>
4097   <include schemaLocation="sca-binding-jms-1.1-schema-200803.xsd"/>
4098   <include schemaLocation="sca-binding-sca-1.1-schema-200803.xsd"/>
4099
4100   <include schemaLocation="sca-definitions-1.1-schema-200803.xsd"/>
4101   <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4102
4103   <include schemaLocation="sca-contribution-1.1-schema-200803.xsd"/>
4104
4105 </schema>
```

### 4106 A.2 sca-core.xsd

4107

```
4108 <?xml version="1.0" encoding="UTF-8"?>
4109 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4110 IPR and other policies apply. -->
4111 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4112   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4113   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4114   elementFormDefault="qualified">
4115
4116   <import namespace="http://www.w3.org/XML/1998/namespace"
4117     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
4118
4119   <!-- Common extension base for SCA definitions -->
4120   <complexType name="CommonExtensionBase">
4121     <sequence>
```

```

4122         <element ref="sca:documentation" minOccurs="0"
4123             maxOccurs="unbounded" />
4124     </sequence>
4125     <anyAttribute namespace="##other" processContents="lax" />
4126 </complexType>
4127
4128 <element name="documentation" type="sca:Documentation" />
4129 <complexType name="Documentation" mixed="true">
4130     <sequence>
4131         <any namespace="##other" processContents="lax" minOccurs="0"
4132             maxOccurs="unbounded" />
4133     </sequence>
4134     <attribute ref="xml:lang" />
4135 </complexType>
4136
4137 <!-- Component Type -->
4138 <element name="componentType" type="sca:ComponentType" />
4139 <complexType name="ComponentType">
4140     <complexContent>
4141         <extension base="sca:CommonExtensionBase">
4142             <sequence>
4143                 <element ref="sca:implementation" minOccurs="0" />
4144                 <choice minOccurs="0" maxOccurs="unbounded">
4145                     <element name="service" type="sca:ComponentService" />
4146                     <element name="reference"
4147                         type="sca:ComponentTypeReference" />
4148                     <element name="property" type="sca:Property" />
4149                 </choice>
4150                 <any namespace="##other" processContents="lax" minOccurs="0"
4151                     maxOccurs="unbounded" />
4152             </sequence>
4153             <attribute name="constrainingType" type="QName" use="optional" />
4154         </extension>
4155     </complexContent>
4156 </complexType>
4157
4158 <!-- Composite -->
4159 <element name="composite" type="sca:Composite" />
4160 <complexType name="Composite">
4161     <complexContent>
4162         <extension base="sca:CommonExtensionBase">
4163             <sequence>
4164                 <element name="include" type="anyURI" minOccurs="0"
4165                     maxOccurs="unbounded" />
4166                 <choice minOccurs="0" maxOccurs="unbounded">
4167                     <element name="service" type="sca:Service" />
4168                     <element name="property" type="sca:Property" />
4169                     <element name="component" type="sca:Component" />
4170                     <element name="reference" type="sca:Reference" />
4171                     <element name="wire" type="sca:Wire" />
4172                 </choice>
4173                 <any namespace="##other" processContents="lax" minOccurs="0"
4174                     maxOccurs="unbounded" />
4175             </sequence>
4176             <attribute name="name" type="NCName" use="required" />
4177             <attribute name="targetNamespace" type="anyURI" use="required" />
4178             <attribute name="local" type="boolean" use="optional"
4179                 default="false" />

```

Comment [ME81]: Issue 5

```

4180     <attribute name="autowire" type="boolean" use="optional"
4181         default="false"/>
4182     <attribute name="constrainingType" type="QName" use="optional"/>
4183     <attribute name="requires" type="sca:listOfQNames"
4184         use="optional"/>
4185     <attribute name="policySets" type="sca:listOfQNames"
4186         use="optional"/>
4187     </extension>
4188 </complexContent>
4189 </complexType>
4190
4191 <!-- Contract base type for Service, Reference -->
4192 <complexType name="Contract" abstract="true">
4193     <complexContent>
4194         <extension base="sca:CommonExtensionBase">
4195             <sequence>
4196                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4197                 <element name="operation" type="sca:Operation" minOccurs="0"
4198                     maxOccurs="unbounded" />
4199                 <element ref="sca:binding" minOccurs="0"
4200                     maxOccurs="unbounded" />
4201                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4202                 <any namespace="##other" processContents="lax" minOccurs="0"
4203                     maxOccurs="unbounded" />
4204             </sequence>
4205             <attribute name="name" type="NCName" use="required" />
4206             <attribute name="requires" type="sca:listOfQNames"
4207                 use="optional"/>
4208             <attribute name="policySets" type="sca:listOfQNames"
4209                 use="optional"/>
4210         </extension>
4211     </complexContent>
4212 </complexType>
4213
4214 <!-- Service -->
4215 <complexType name="Service">
4216     <complexContent>
4217         <extension base="sca:Contract">
4218             <attribute name="promote" type="anyURI" use="required"/>
4219         </extension>
4220     </complexContent>
4221 </complexType>
4222
4223 <!-- Interface -->
4224 <element name="interface" type="sca:Interface" abstract="true"/>
4225 <complexType name="Interface" abstract="true">
4226     <complexContent>
4227         <extension base="sca:CommonExtensionBase"/>
4228     </complexContent>
4229 </complexType>
4230
4231 <!-- Reference -->
4232 <complexType name="Reference">
4233     <complexContent>
4234         <extension base="sca:Contract">
4235             <attribute name="autowire" type="boolean" use="optional"/>
4236             <attribute name="target" type="sca:listOfAnyURIs"
4237                 use="optional"/>

```

**Comment [ME82]:** Issue 12

**Comment [ME83]:** Issue 12

```

Deleted: <!-- Service
-->
<complexType
name="Service">
  <complexContent>
    <extension
base="sca:CommonExtensi
onBase">
      <sequence>
        <element
ref="sca:interface"
minOccurs="0"/>
        <element
name="operation"
type="sca:Operation"
minOccurs="0"
maxOccurs="unbounded"/>
        <element
ref="sca:binding"
minOccurs="0"
maxOccurs="unbounded"/>
        <element
ref="sca:callback"
minOccurs="0"
maxOccurs="1"/>
        <any
namespace="##other"
processContents="lax"
minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
      <attribute
name="name"
type="NCName"
use="required"/>
      <attribute
name="promote"
type="anyURI"
use="required"/>
      <attribute
name="requires"
type="sca:listOfQNames"
use="optional"/>
      <attribute
name="policySets"
type="sca:listOfQNames"
use="optional"/>
    </extension>
  </complexContent>
</complexType>

```

```

4238     <attribute name="wiredByImpl" type="boolean" use="optional"
4239         default="false"/>
4240     <attribute name="multiplicity" type="sca:Multiplicity"
4241         use="optional" default="1..1"/>
4242     <attribute name="promote" type="sca:listOfAnyURIs"
4243         use="required"/>
4244     </extension>
4245 </complexContent>
4246 </complexType>
4247
4248 <!-- Property -->
4249 <complexType name="SCAPropertyBase" mixed="true">
4250     <sequence>
4251         <any namespace="##any" processContents="lax" minOccurs="0"/>
4252         <!-- NOT an extension point; This any exists to accept
4253             the element-based or complex type property
4254             i.e. no element-based extension point under "sca:property" -->
4255     </sequence>
4256     <!-- mixed="true" to handle simple type -->
4257     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4258     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4259 </complexType>
4260
4261 <complexType name="Property" mixed="true">
4262     <complexContent mixed="true">
4263         <extension base="sca:SCAPropertyBase">
4264             <attribute name="name" type="NCName" use="required"/>
4265             <attribute name="type" type="QName" use="optional"/>
4266             <attribute name="element" type="QName" use="optional"/>
4267             <attribute name="many" type="boolean" use="optional"
4268                 default="false"/>
4269             <attribute name="mustSupply" type="boolean" use="optional"
4270                 default="false"/>
4271             <anyAttribute namespace="##any" processContents="lax"/>
4272         </extension>
4273         <!-- extension defines the place to hold default value -->
4274         <!-- an extension point ; attribute-based only -->
4275     </complexContent>
4276 </complexType>
4277
4278 <complexType name="PropertyValue" mixed="true">
4279     <complexContent mixed="true">
4280         <extension base="sca:SCAPropertyBase">
4281             <attribute name="name" type="NCName" use="required"/>
4282             <attribute name="type" type="QName" use="optional"/>
4283             <attribute name="element" type="QName" use="optional"/>
4284             <attribute name="many" type="boolean" use="optional"
4285                 default="false"/>
4286             <attribute name="source" type="string" use="optional"/>
4287             <attribute name="file" type="anyURI" use="optional"/>
4288             <anyAttribute namespace="##any" processContents="lax"/>
4289         </extension>
4290         <!-- an extension point ; attribute-based only -->
4291     </complexContent>
4292 </complexType>
4293
4294 <!-- Binding -->
4295 <element name="binding" type="sca:Binding" abstract="true"/>

```

**Comment [ME84]: Issue 12**

```

Deleted: <!--
Reference -->
<complexType
name="Reference">
  <complexContent>
    <extension
base="sca:CommonExtensionBase">
      <sequence>
        <element
ref="sca:interface"
minOccurs="0"/>
        <element
name="operation"
type="sca:Operation"
minOccurs="0"
maxOccurs="unbounded"/>
        <element
ref="sca:binding"
minOccurs="0"
maxOccurs="unbounded"/>
        <element
ref="sca:callback"
minOccurs="0"/>
        <any
namespace="##other"
processContents="lax"
minOccurs="0"
maxOccurs="unbounded"/>
      </sequence>
      <attribute
name="name"
type="NCName"
use="required"/>
      <attribute
name="autowire"
type="boolean"
use="optional"/>
      <attribute
name="target"
type="sca:listOfAnyURIs"
use="optional"/>
      <attribute
name="wiredByImpl"
type="boolean"
use="optional"
default="false"/>
      <attribute
name="multiplicity"
type="sca:Multiplicity"
use="optional"
default="1..1"/>
      <attribute
name="promote"
type="sca:listOfAnyURIs"
use="required"/>
      <attribute
name="requires"
type="sca:listOfQNames"

```

[7]

**Comment [ME85]: Issue 45**

```

4296 <complexType name="Binding" abstract="true">
4297   <complexContent>
4298     <extension base="sca:CommonExtensionBase">
4299       <sequence>
4300         <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
4301         <element ref="sca:operationSelector"
4302           minOccurs="0" maxOccurs="1" />
4303         <element name="operation" type="sca:Operation" minOccurs="0"
4304           maxOccurs="unbounded" />
4305       </sequence>
4306       <attribute name="uri" type="anyURI" use="optional"/>
4307       <attribute name="name" type="NCName" use="optional"/>
4308       <attribute name="requires" type="sca:listOfQNames"
4309         use="optional"/>
4310       <attribute name="policySets" type="sca:listOfQNames"
4311         use="optional"/>
4312     </extension>
4313   </complexContent>
4314 </complexType>
4315
4316 <!-- Binding Type -->
4317 <element name="bindingType" type="sca:BindingType"/>
4318 <complexType name="BindingType">
4319   <complexContent>
4320     <extension base="sca:CommonExtensionBase">
4321       <sequence>
4322         <any namespace="##other" processContents="lax" minOccurs="0"
4323           maxOccurs="unbounded" />
4324       </sequence>
4325       <attribute name="type" type="QName" use="required"/>
4326       <attribute name="alwaysProvides" type="sca:listOfQNames"
4327         use="optional"/>
4328       <attribute name="mayProvide" type="sca:listOfQNames"
4329         use="optional"/>
4330     </extension>
4331   </complexContent>
4332 </complexType>
4333
4334 <!-- WireFormat Type -->
4335 <element name="wireFormat" type="sca:WireFormatType"/>
4336 <complexType name="WireFormatType" abstract="true">
4337   <sequence>
4338     <any namespace="##other" processContents="lax" minOccurs="0"
4339       maxOccurs="unbounded" />
4340   </sequence>
4341   <anyAttribute namespace="##other" processContents="lax"/>
4342 </complexType>
4343
4344 <!-- OperationSelector Type -->
4345 <element name="operationSelector" type="sca:OperationSelectorType"/>
4346 <complexType name="OperationSelectorType" abstract="true">
4347   <sequence>
4348     <any namespace="##other" processContents="lax" minOccurs="0"
4349       maxOccurs="unbounded" />
4350   </sequence>
4351   <anyAttribute namespace="##other" processContents="lax"/>
4352 </complexType>
4353 <!-- Callback -->

```

Comment [ME86]: Issue 79

Comment [ME87]: ISSUE 79

```

4354 <element name="callback" type="sca:Callback"/>
4355 <complexType name="Callback">
4356   <complexContent>
4357     <extension base="sca:CommonExtensionBase">
4358       <choice minOccurs="0" maxOccurs="unbounded">
4359         <element ref="sca:binding"/>
4360         <any namespace="##other" processContents="lax"/>
4361       </choice>
4362       <attribute name="requires" type="sca:listOfQNames"
4363         use="optional"/>
4364       <attribute name="policySets" type="sca:listOfQNames"
4365         use="optional"/>
4366     </extension>
4367   </complexContent>
4368 </complexType>
4369
4370 <!-- Component -->
4371 <complexType name="Component">
4372   <complexContent>
4373     <extension base="sca:CommonExtensionBase">
4374       <sequence>
4375         <element ref="sca:implementation" minOccurs="0"/>
4376         <choice minOccurs="0" maxOccurs="unbounded">
4377           <element name="service" type="sca:ComponentService"/>
4378           <element name="reference" type="sca:ComponentReference"/>
4379           <element name="property" type="sca:PropertyValue"/>
4380         </choice>
4381         <any namespace="##other" processContents="lax" minOccurs="0"
4382           maxOccurs="unbounded"/>
4383       </sequence>
4384       <attribute name="name" type="NCName" use="required"/>
4385       <attribute name="autowire" type="boolean" use="optional"/>
4386       <attribute name="constrainingType" type="QName" use="optional"/>
4387       <attribute name="requires" type="sca:listOfQNames"
4388         use="optional"/>
4389       <attribute name="policySets" type="sca:listOfQNames"
4390         use="optional"/>
4391     </extension>
4392   </complexContent>
4393 </complexType>
4394
4395 <!-- Component Service -->
4396 <complexType name="ComponentService">
4397   <complexContent>
4398     <extension base="sca:Contract">
4399   </extension>
4400 </complexContent>
4401 </complexType>
4402
4403 <!-- Component Reference -->
4404 <complexType name="ComponentReference">
4405   <complexContent>
4406     <extension base="sca:Contract">
4407       <attribute name="autowire" type="boolean" use="optional"/>
4408       <attribute name="target" type="sca:listOfAnyURIs"
4409         use="optional"/>
4410       <attribute name="wiredByImpl" type="boolean" use="optional"
4411         default="false"/>

```

**Comment [ME88]: Issue 18**

```

Deleted: <!--
Component Service -->
  <complexType
name="ComponentService"
>
    <complexContent>
      <restriction
base="sca:Service">
        <sequence>
          <element
ref="sca:documentation"
minOccurs="0"
maxOccurs="unbounded"/>
          <element
ref="sca:interface"
minOccurs="0"/>
          <element
name="operation"
type="sca:Operation"
minOccurs="0"
maxOccurs="unbounded"/>
          <element
ref="sca:binding"
minOccurs="0"
maxOccurs="unbounded"/>
          <element
ref="sca:callback"
minOccurs="0"/>
          <any
namespace="##other"
processContents="lax"
minOccurs="0"
maxOccurs="unbounded"/>
        </sequence>
      </restriction>
    </complexType>

```

```

4412     <attribute name="multiplicity" type="sca:Multiplicity"
4413         use="optional" default="1..1"/>
4414     </extension>
4415 </complexContent>
4416 </complexType>
4417
4418 <!-- Component Type Reference -->
4419 <complexType name="ComponentTypeReference">
4420     <complexContent>
4421         <restriction base="sca:ComponentReference">
4422             <sequence>
4423                 <element ref="sca:documentation" minOccurs="0"
4424                     maxOccurs="unbounded"/>
4425                 <element ref="sca:interface" minOccurs="0"/>
4426                 <element name="operation" type="sca:Operation" minOccurs="0"
4427                     maxOccurs="unbounded"/>
4428                 <element ref="sca:binding" minOccurs="0"
4429                     maxOccurs="unbounded"/>
4430                 <element ref="sca:callback" minOccurs="0"/>
4431                 <any namespace="##other" processContents="lax" minOccurs="0"
4432                     maxOccurs="unbounded"/>
4433             </sequence>
4434             <attribute name="name" type="NCName" use="required"/>
4435             <attribute name="autowire" type="boolean" use="optional"/>
4436             <attribute name="wiredByImpl" type="boolean" use="optional"
4437                 default="false"/>
4438             <attribute name="multiplicity" type="sca:Multiplicity"
4439                 use="optional" default="1..1"/>
4440             <attribute name="requires" type="sca:listOfQNames"
4441                 use="optional"/>
4442             <attribute name="policySets" type="sca:listOfQNames"
4443                 use="optional"/>
4444             <anyAttribute namespace="##other" processContents="lax"/>
4445         </restriction>
4446     </complexContent>
4447 </complexType>
4448
4449 <!-- Implementation -->
4450 <element name="implementation" type="sca:Implementation" abstract="true"/>
4451 <complexType name="Implementation" abstract="true">
4452     <complexContent>
4453         <extension base="sca:CommonExtensionBase">
4454             <attribute name="requires" type="sca:listOfQNames"
4455                 use="optional"/>
4456             <attribute name="policySets" type="sca:listOfQNames"
4457                 use="optional"/>
4458         </extension>
4459     </complexContent>
4460 </complexType>
4461
4462 <!-- Implementation Type -->
4463 <element name="implementationType" type="sca:ImplementationType"/>
4464 <complexType name="ImplementationType">
4465     <complexContent>
4466         <extension base="sca:CommonExtensionBase">
4467             <sequence>
4468                 <any namespace="##other" processContents="lax" minOccurs="0"
4469                     maxOccurs="unbounded"/>

```

**Comment [ME89]:** Issue 18

```

Deleted: <!--
Component Reference -->
<complexType
name="ComponentReferenc
e">
    <complexContent>
        <restriction
base="sca:Reference">
            <sequence>
                <element
ref="sca:documentation"
minOccurs="0"
maxOccurs="unbounded"/>
                <element
ref="sca:interface"
minOccurs="0"/>
                <element
name="operation"
type="sca:Operation"
minOccurs="0"
maxOccurs="unbounded"/>
                <element
ref="sca:binding"
minOccurs="0"
maxOccurs="unbounded"/>
                <element
ref="sca:callback"
minOccurs="0"/>
                <any
namespace="##other"
processContents="lax"
minOccurs="0"
maxOccurs="unbounded"/>
            </sequence>
            <attribute
name="name"
type="NCName"
use="required"/>
            <attribute
name="autowire"
type="boolean"
use="optional"/>
            <attribute
name="target"
type="sca:listOfAnyURIs"
use="optional"/>
            <attribute
name="wiredByImpl"
type="boolean"
use="optional"
default="false"/>
            <attribute
name="multiplicity"
type="sca:Multiplicity"
use="optional"
default="1..1"/>
            <attribute
name="requires"
type="sca:listOfQNames"
use="optional"/>

```

... [8]

**Comment [ME90]:** Issue 5

```

4470         </sequence>
4471         <attribute name="type" type="QName" use="required"/>
4472         <attribute name="alwaysProvides" type="sca:listOfQNames"
4473             use="optional"/>
4474         <attribute name="mayProvide" type="sca:listOfQNames"
4475             use="optional"/>
4476     </extension>
4477 </complexContent>
4478 </complexType>
4479
4480 <!-- Wire -->
4481 <complexType name="Wire">
4482     <complexContent>
4483         <extension base="sca:CommonExtensionBase">
4484             <sequence>
4485                 <any namespace="##other" processContents="lax" minOccurs="0"
4486                     maxOccurs="unbounded" />
4487             </sequence>
4488             <attribute name="source" type="anyURI" use="required"/>
4489             <attribute name="target" type="anyURI" use="required"/>
4490         </extension>
4491     </complexContent>
4492 </complexType>
4493
4494 <!-- Include -->
4495 <element name="include" type="sca:Include"/>
4496 <complexType name="Include">
4497     <complexContent>
4498         <extension base="sca:CommonExtensionBase">
4499             <attribute name="name" type="QName"/>
4500         </extension>
4501     </complexContent>
4502 </complexType>
4503
4504 <!-- Operation -->
4505 <complexType name="Operation">
4506     <complexContent>
4507         <extension base="sca:CommonExtensionBase">
4508             <attribute name="name" type="NCName" use="required"/>
4509             <attribute name="requires" type="sca:listOfQNames"
4510                 use="optional"/>
4511             <attribute name="policySets" type="sca:listOfQNames"
4512                 use="optional"/>
4513         </extension>
4514     </complexContent>
4515 </complexType>
4516
4517 <!-- Constraining Type -->
4518 <element name="constrainingType" type="sca:ConstrainingType"/>
4519 <complexType name="ConstrainingType">
4520     <complexContent>
4521         <extension base="sca:CommonExtensionBase">
4522             <sequence>
4523                 <choice minOccurs="0" maxOccurs="unbounded">
4524                     <element name="service" type="sca:ComponentService"/>
4525                     <element name="reference" type="sca:ComponentReference"/>
4526                     <element name="property" type="sca:Property"/>
4527                 </choice>

```

```

4528         <any namespace="##other" processContents="lax" minOccurs="0"
4529             maxOccurs="unbounded" />
4530     </sequence>
4531     <attribute name="name" type="NCName" use="required"/>
4532     <attribute name="targetNamespace" type="anyURI"/>
4533     <attribute name="requires" type="sca:listOfQNames"
4534         use="optional"/>
4535     </extension>
4536 </complexContent>
4537 </complexType>
4538
4539 <!-- Intents within WSDL documents -->
4540 <attribute name="requires" type="sca:listOfQNames"/>
4541
4542 <!-- Marker for operations ending a conversation -->
4543 <attribute name="endsConversation" type="boolean" default="false"/>
4544
4545 <!-- Global attribute definition for @callback to mark a WSDL port type
4546 as having a callback interface defined in terms of a second port
4547 type. -->
4548 <attribute name="callback" type="anyURI"/>
4549
4550 <!-- Miscellaneous simple type definitions -->
4551 <simpleType name="Multiplicity">
4552     <restriction base="string">
4553         <enumeration value="0..1"/>
4554         <enumeration value="1..1"/>
4555         <enumeration value="0..n"/>
4556         <enumeration value="1..n"/>
4557     </restriction>
4558 </simpleType>
4559
4560 <simpleType name="OverrideOptions">
4561     <restriction base="string">
4562         <enumeration value="no"/>
4563         <enumeration value="may"/>
4564         <enumeration value="must"/>
4565     </restriction>
4566 </simpleType>
4567
4568 <simpleType name="listOfQNames">
4569     <list itemType="QName"/>
4570 </simpleType>
4571
4572 <simpleType name="listOfAnyURIs">
4573     <list itemType="anyURI"/>
4574 </simpleType>
4575
4576 </schema>
4577

```

Comment [ME91]: Issue 89

### 4578 A.3 sca-binding-sca.xsd

```

4579
4580 <?xml version="1.0" encoding="UTF-8"?>
4581 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4582 IPR and other policies apply. -->
4583 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

4584     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4585     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4586     elementFormDefault="qualified">
4587
4588     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4589
4590     <!-- SCA Binding -->
4591     <element name="binding.sca" type="sca:SCABinding"
4592         substitutionGroup="sca:binding"/>
4593     <complexType name="SCABinding">
4594         <complexContent>
4595             <extension base="sca:Binding"/>
4596         </complexContent>
4597     </complexType>
4598
4599 </schema>
4600

```

#### 4601 **A.4 sca-interface-java.xsd**

```

4602
4603 <?xml version="1.0" encoding="UTF-8"?>
4604 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4605 IPR and other policies apply. -->
4606 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4607     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4608     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4609     elementFormDefault="qualified">
4610
4611     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4612
4613     <!-- Java Interface -->
4614     <element name="interface.java" type="sca:JavaInterface"
4615         substitutionGroup="sca:interface"/>
4616     <complexType name="JavaInterface">
4617         <complexContent>
4618             <extension base="sca:Interface">
4619                 <sequence>
4620                     <any namespace="##other" processContents="lax" minOccurs="0"
4621                         maxOccurs="unbounded"/>
4622                 </sequence>
4623                 <attribute name="interface" type="NCName" use="required"/>
4624                 <attribute name="callbackInterface" type="NCName"
4625                     use="optional"/>
4626                 <anyAttribute namespace="##any" processContents="lax"/>
4627             </extension>
4628         </complexContent>
4629     </complexType>
4630
4631 </schema>
4632
4633

```

#### 4634 **A.5 sca-interface-wsdl.xsd**

4635

```

4636 <?xml version="1.0" encoding="UTF-8"?>
4637 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4638 IPR and other policies apply. -->
4639 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4640   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4641   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4642   elementFormDefault="qualified">
4643
4644   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4645
4646   <!-- WSDL Interface -->
4647   <element name="interface.wsdl" type="sca:WSDLPortType"
4648     substitutionGroup="sca:interface"/>
4649   <complexType name="WSDLPortType">
4650     <complexContent>
4651       <extension base="sca:Interface">
4652         <sequence>
4653           <any namespace="##other" processContents="lax" minOccurs="0"
4654             maxOccurs="unbounded"/>
4655         </sequence>
4656         <attribute name="interface" type="anyURI" use="required"/>
4657         <attribute name="callbackInterface" type="anyURI"
4658           use="optional"/>
4659         <anyAttribute namespace="##any" processContents="lax"/>
4660       </extension>
4661     </complexContent>
4662   </complexType>
4663 </schema>
4664
4665
4666

```

## 4667 **A.6 sca-implementation-java.xsd**

```

4668
4669 <?xml version="1.0" encoding="UTF-8"?>
4670 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4671 IPR and other policies apply. -->
4672 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4673   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4674   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4675   elementFormDefault="qualified">
4676
4677   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4678
4679   <!-- Java Implementation -->
4680   <element name="implementation.java" type="sca:JavaImplementation"
4681     substitutionGroup="sca:implementation"/>
4682   <complexType name="JavaImplementation">
4683     <complexContent>
4684       <extension base="sca:Implementation">
4685         <sequence>
4686           <any namespace="##other" processContents="lax" minOccurs="0"
4687             maxOccurs="unbounded"/>
4688         </sequence>
4689         <attribute name="class" type="NCName" use="required"/>
4690         <anyAttribute namespace="##any" processContents="lax"/>

```

```
4691         </extension>
4692     </complexContent>
4693 </complexType>
4694
4695 </schema>
```

## 4696 **A.7 sca-implementation-composite.xsd**

```
4697
4698 <?xml version="1.0" encoding="UTF-8"?>
4699 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4700 IPR and other policies apply. -->
4701 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4702     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4703     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4704     elementFormDefault="qualified">
4705
4706     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4707
4708     <!-- Composite Implementation -->
4709     <element name="implementation.composite" type="sca:SCAImplementation"
4710         substitutionGroup="sca:implementation"/>
4711     <complexType name="SCAImplementation">
4712         <complexContent>
4713             <extension base="sca:Implementation">
4714                 <sequence>
4715                     <any namespace="##other" processContents="lax" minOccurs="0"
4716                         maxOccurs="unbounded"/>
4717                 </sequence>
4718                 <attribute name="name" type="QName" use="required"/>
4719             </extension>
4720         </complexContent>
4721     </complexType>
4722
4723 </schema>
4724
```

**Comment [ME92]:** Removed anyAttribute as it contradicts the extension from CommonExtensionBase

## 4725 **A.8 sca-definitions.xsd**

```
4726
4727 <?xml version="1.0" encoding="UTF-8"?>
4728 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4729 IPR and other policies apply. -->
4730 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4731     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4732     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4733     elementFormDefault="qualified">
4734
4735     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4736     <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4737
4738     <!-- Definitions -->
4739     <element name="definitions" type="sca:tDefinitions"/>
4740     <complexType name="tDefinitions">
4741         <complexContent>
4742             <extension base="sca:CommonExtensionBase">
4743                 <choice minOccurs="0" maxOccurs="unbounded">
```

```

4744         <element ref="sca:intent" />
4745         <element ref="sca:policySet" />
4746         <element ref="sca:binding" />
4747         <element ref="sca:bindingType" />
4748         <element ref="sca:implementationType" />
4749         <any namespace="##other" processContents="lax" minOccurs="0"
4750             maxOccurs="unbounded" />
4751     </choice>
4752 </extension>
4753 </complexContent>
4754 </complexType>
4755
4756 </schema>
4757
4758

```

## 4759 **A.9 sca-binding-webservice.xsd**

4760 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4761 **A.10 sca-binding-jms.xsd**

4762 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4763 **A.11 sca-policy.xsd**

4764 Is described in [the SCA Policy Framework specification \[10\]](#)

4765

## 4766 **A.12 sca-contribution.xsd**

**Comment [mbgl93]:** Issue 28

```

4767
4768 <?xml version="1.0" encoding="UTF-8"?>
4769 <!-- Copyright (C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4770 IPR and other policies apply. -->
4771 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4772     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4773     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4774     elementFormDefault="qualified">
4775
4776     <include schemaLocation="sca-core-1.1-schema-200803.xsd" />
4777
4778     <!-- Contribution -->
4779     <element name="contribution" type="sca:ContributionType" />
4780     <complexType name="ContributionType">
4781         <complexContent>
4782             <extension base="sca:CommonExtensionBase">
4783                 <sequence>
4784                     <element name="deployable" type="sca:DeployableType"
4785                         maxOccurs="unbounded" />
4786                     <element name="import" type="sca:ImportType" minOccurs="0"
4787                         maxOccurs="unbounded" />
4788                     <element name="export" type="sca:ExportType" minOccurs="0"
4789                         maxOccurs="unbounded" />
4790                     <any namespace="##other" processContents="lax" minOccurs="0"
4791                         maxOccurs="unbounded" />
4792                 </sequence>

```

```

4793         </extension>
4794     </complexContent>
4795 </complexType>
4796
4797 <!-- Deployable -->
4798 <complexType name="DeployableType">
4799     <complexContent>
4800         <extension base="sca:CommonExtensionBase">
4801             <sequence>
4802                 <any namespace="##other" processContents="lax" minOccurs="0"
4803                     maxOccurs="unbounded" />
4804             </sequence>
4805             <attribute name="composite" type="QName" use="required" />
4806         </extension>
4807     </complexContent>
4808 </complexType>
4809
4810 <!-- Import -->
4811 <element name="importBase" type="sca:Import" abstract="true" />
4812 <complexType name="Import" abstract="true">
4813     <complexContent>
4814         <extension base="sca:CommonExtensionBase">
4815             <sequence>
4816                 <any namespace="##other" processContents="lax" minOccurs="0"
4817                     maxOccurs="unbounded" />
4818             </sequence>
4819         </extension>
4820     </complexContent>
4821 </complexType>
4822
4823 <element name="import" type="sca:ImportType" />
4824 <complexType name="ImportType">
4825     <complexContent>
4826         <extension base="sca:Import">
4827             <attribute name="namespace" type="string" use="required" />
4828             <attribute name="location" type="anyURI" use="optional" />
4829         </extension>
4830     </complexContent>
4831 </complexType>
4832
4833 <!-- Export -->
4834 <element name="exportBase" type="sca:Export" abstract="true" />
4835 <complexType name="Export" abstract="true">
4836     <complexContent>
4837         <extension base="sca:CommonExtensionBase">
4838             <sequence>
4839                 <any namespace="##other" processContents="lax" minOccurs="0"
4840                     maxOccurs="unbounded" />
4841             </sequence>
4842         </extension>
4843     </complexContent>
4844 </complexType>
4845
4846 <element name="export" type="sca:ExportType" />
4847 <complexType name="ExportType">
4848     <complexContent>
4849         <extension base="sca:Export">
4850             <attribute name="namespace" type="string" use="required" />

```

```
4851         </extension>
4852     </complexContent>
4853 </complexType>
4854 </schema>
4855
4856
4857
```

## 4858 B. SCA Concepts

### 4859 B.1 Binding

4860 **Bindings** are used by services and references. References use bindings to describe the access  
4861 mechanism used to call the service to which they are wired. Services use bindings to describe the  
4862 access mechanism(s) that clients should use to call the service.

4863 SCA supports multiple different types of bindings. Examples include **SCA service, Web service,**  
4864 **stateless session EJB, data base stored procedure, EIS service.** SCA provides an extensibility  
4865 mechanism by which an SCA runtime can add support for additional binding types.

4866

### 4867 B.2 Component

4868 **SCA components** are configured instances of **SCA implementations**, which provide and consume  
4869 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines  
4870 an **extensibility mechanism** that allows you to introduce new implementation types. The current  
4871 specification does not mandate the implementation technologies to be supported by an SCA run-time,  
4872 vendors may choose to support the ones that are important for them. A single SCA implementation may  
4873 be used by multiple Components, each with a different configuration.

4874 The Component has a reference to an implementation of which it is an instance, a set of property values,  
4875 and a set of service reference values. Property values define the values of the properties of the  
4876 component as defined by the component's implementation. Reference values define the services that  
4877 resolve the references of the component as defined by its implementation. These values can either be a  
4878 particular service of a particular component, or a reference of the containing composite.

### 4879 B.3 Service

4880 **SCA services** are used to declare the externally accessible services of an **implementation**. For a  
4881 composite, a service is typically provided by a service of a component within the composite, or by a  
4882 reference defined by the composite. The latter case allows the republication of a service with a new  
4883 address and/or new bindings. The service can be thought of as a point at which messages from external  
4884 clients enter a composite or implementation.

4885 A service represents an addressable set of operations of an implementation that are designed to be  
4886 exposed for use by other implementations or exposed publicly for use elsewhere (eg public Web services  
4887 for use by other organizations). The operations provided by a service are specified by an Interface, as  
4888 are the operations required by the service client (if there is one). An implementation may contain  
4889 multiple services, when it is possible to address the services of the implementation separately.

4890 A service may be provided **as SCA remote services, as Web services, as stateless session EJB's, as**  
4891 **EIS services, and so on.** Services use **bindings** to describe the way in which they are published. SCA  
4892 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new  
4893 types of services.

#### 4894 B.3.1 Remotable Service

4895 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA  
4896 architecture. For example, SCA services of SCA implementations can define implementations of industry-  
4897 standard web services. Remotable services use pass-by-value semantics for parameters and returned  
4898 results.

4899 How a Service is identified as remotable is dependant on the Component implementation technology  
4900 used. See the relevant SCA Implementation Specification for more information. As an example, to define  
4901 a Remotable Service, a Component implemented in Java would have a Java Interface with the  
4902 @Remotable annotation.

Comment [ME94]: Issue 63

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

### 4903 B.3.2 Local Service

4904 Local services are services that are designed to be only used “locally” by other implementations that are  
4905 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4906 Local services may rely on by-reference calling conventions, or may assume a very fine-grained  
4907 interaction style that is incompatible with remote distribution. They may also use technology-specific data-  
4908 types.

4909 How a Service is identified as local is dependant on the Component implementation technology used.  
4910 See the relevant SCA Implementation Specification for more information. As an example, to define a  
4911 Local Service, a Component implemented in Java would define a Java Interface that does not have the  
4912 @Remotable annotation.

Comment [ME95]: Issue 63

**Deleted:** Currently a service is local only if it defined by a Java interface not marked with the @Remotable annotation

4913

### 4914 B.4 Reference

4915 **SCA references** represent a dependency that an implementation has on a service that is supplied by  
4916 some other implementation, where the service to be used is specified through configuration. In other  
4917 words, a reference is a service that an implementation may call during the execution of its business  
4918 function. References are typed by an interface.

4919 For composites, composite references can be accessed by components within the composite like any  
4920 service provided by a component within the composite. Composite references can be used as the targets  
4921 of wires from component references when configuring Components.

4922 A composite reference can be used to access a service such as: an SCA service provided by another  
4923 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,  
4924 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4925 an **extensibility mechanism** that allows the introduction of new binding types to references.

4926

### 4927 B.5 Implementation

4928 An implementation is concept that is used to describe a piece of software technology such as a Java  
4929 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4930 service-oriented application. An SCA composite is also an implementation.

4931 Implementations define points of variability including properties that can be set and settable references to  
4932 other services. The points of variability are configured by a component that uses the implementation. The  
4933 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4934 B.6 Interface

4935 **Interfaces** define one or more business functions. These business functions are provided by Services  
4936 and are used by components through References. Services are defined by the Interface they implement.  
4937 SCA currently supports a number of interface type systems, for example:

- 4938 • Java interfaces
- 4939 • WSDL portTypes
- 4940 • C, C++ header files

4941

4942 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4943 interface type systems.

4944 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided  
4945 by each end of a service communication – this could be the case where a particular service requires a  
4946 “callback” interface on the client, which is calls during the process of handing service requests from the  
4947 client.

4948

## 4949 B.7 Composite

4950 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4951 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4952 be used to contribute elements to an **SCA Domain**.

4953 A **composite** has the following characteristics:

- 4954 • It may be used as a component implementation. When used in this way, it defines a boundary for  
4955 Component visibility. Components may not be directly referenced from outside of the composite  
4956 in which they are declared.
- 4957 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
4958 artifacts to an SCA domain.

4959

## 4960 B.8 Composite inclusion

4961 One composite can be used to provide part of the definition of another composite, through the process of  
4962 inclusion. This is intended to make team development of large composites easier. Included composites  
4963 are merged together into the using composite at deployment time to form a single logical composite.

4964 Composites are included into other composites through <include.../> elements in the using composite.  
4965 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
4966 specific location.

4967

## 4968 B.9 Property

4969 **Properties** allow for the configuration of an implementation with externally set data values. The data  
4970 value is provided through a Component, possibly sourced from the property of a containing composite.

4971 Each Property is defined by the implementation. Properties may be defined directly through the  
4972 implementation language or through annotations of implementations, where the implementation language  
4973 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
4974 type. For complex data types, XML schema is the preferred technology for defining the data types.

4975

## 4976 B.10 Domain

4977 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
4978 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
4979 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
4980 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4981 A domain specifies the instantiation, configuration and connection of a set of components, provided via  
4982 one or more composite files. The domain, like a composite, also has Services and References. Domains  
4983 also contain Wires which connect together the Components, Services and References.

4984

## 4985 B.11 Wire

4986 **SCA wires** connect **service references** to **services**.

4987 **Valid wire sources are component references. Valid wire targets are component services.**

4988 When using included composites, the sources and targets of the wires don't have to be declared in the  
4989 same composite as the composite that contains the wire. The sources and targets can be defined by  
4990 other included composites. Targets can also be external to the SCA domain.

**Comment [ME96]:** Issue 75

**Deleted:** Within a composite, valid wire sources are component references and composite services. Valid wire targets are component services and composite references



4992

## C. Conformance Items

4993

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

4994

Conformance ID	Description
<del>[ASM13001]</del>	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.
[ASM40001]	The extension of a componentType side file name MUST be <u>.componentType</u> .
[ASM40002]	If present, the @constrainingType attribute of a <componentType/> element MUST reference a <constrainingType/> element in the Domain through its QName.
<del>[ASM40003]</del>	The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
<del>[ASM40004]</del>	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.
<del>[ASM50001]</del>	The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/>
[ASM50002]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50003]	The @name attribute of a service element of a <component/> MUST match the @name attribute of a service element of the componentType of the <implementation/> child element of the component.
<del>[ASM50004]</del>	If a <service/> element has an interface subelement specified, the interface MUST provide a compatible subset of the interface

Deleted: [ASM10001]

Formatted: Font color: Red

Deleted: [ASM40003]

Deleted: [ASM40004]

Deleted: [ASM50001][ASM50001]

Formatted Table

Deleted: [ASM50004]

	declared on the componentType of the implementation
<del>[ASM50005]</del>	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.
<del>[ASM50006]</del>	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.
<del>[ASM50011]</del>	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.
<del>[ASM50012]</del>	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

Deleted: [ASM50005]

Deleted: [ASM50006]

Deleted: [ASM50011]

Deleted: [ASM50012]

	target services using binding specific attributes or elements.
[ASM50016]	It is possible that a particular binding type MAY require that the address of a target service uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to identify the target service - instead, binding specific attributes and/or child elements must be used.
[ASM50018]	A reference with multiplicity 0..1 or 0..n MAY have no target service defined.
[ASM50019]	A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service defined.
[ASM50020]	A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.
[ASM50021]	A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.
[ASM50022]	Where it is detected that the rules for the number of target services for a reference have been violated, either at deployment or at execution time, an SCA Runtime MUST generate an error no later than when the reference is invoked by the component implementation.
[ASM50023]	Some reference multiplicity errors can be detected at deployment time. In these cases, an error SHOULD be generated by the SCA runtime at deployment time.
[ASM50024]	Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA runtime MUST generate an error no later than when the reference is invoked by the component implementation.
[ASM50025]	Where a component reference is promoted by a composite reference, the promotion MUST be treated from a multiplicity perspective as providing 0 or more target services for the component reference, depending upon the further configuration of the composite reference. These target services are in addition to any target services identified on the component reference itself, subject to the rules relating to multiplicity.
[ASM50026]	If a reference has a value specified for one or more target services in its @target attribute, there MUST NOT be any child <binding/> elements declared for that reference.
[ASM50027]	If the @value attribute of a component property element is declared, the type of the property MUST be an XML Schema simple type and the @value attribute MUST contain a single value of that type.
[ASM50028]	If the value subelement of a component property is specified, the type of the property MUST be an XML Schema simple type or an XML schema complex type.
[ASM50029]	If a component property value is declared using a child element of the <property/> element, the type of the property MUST be an XML Schema global element and the declared child element MUST be an instance of that global element.

[ASM50030]	A <component/> element MUST NOT contain two <property/> subelements with the same value of the @name attribute.
[ASM50031]	The name attribute of a component property MUST match the name of a property element in the component type of the component implementation.
[ASM50032]	If a property is single-valued, the <value/> subelement MUST NOT occur more than once.
[ASM50033]	A property <value/> subelement MUST NOT be used when the @value attribute is used to specify the value for that property.
[ASM50034]	<u>If any &lt;wire/&gt; element with its @replace attribute set to "true" has a particular reference specified in its @source attribute, the value of the @target attribute for that reference MUST be ignored and MUST NOT be used to define target services for that reference.</u>
[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

Comment [ME97]: Issue 41

Deleted: [ASM60005]

	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.
<del>[ASM60012]</del>	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.
<del>[ASM60022]</del>	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, <del>and</del> policies applied to the service MUST be

Deleted: [ASM60012]

Deleted: [ASM60022]

Comment [mbgl98]: Issue 57

	compatible with those on the reference when using autowire to wire a reference – so that wiring the reference to the service will not cause an error due to policy mismatch
[ASM60025]	for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the reference to one of the set of valid target services chosen from the set in a runtime-dependent fashion
[ASM60026]	for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all of the set of valid target services
[ASM60027]	for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT raise an error
[ASM60028]	for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid target services an error MUST be raised by the SCA runtime since the reference is intended to be wired
[ASM60030]	The @name attribute of an <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.
[ASM60031]	The SCA runtime MUST raise an error if the composite resulting from the inclusion of one composite into another is invalid.
[ASM60032]	For a composite used as a component implementation, each composite service offered by the composite MUST promote a component service of a component that is within the composite.
[ASM60033]	For a composite used as a component implementation, every component reference of components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or promoted (according to the various rules for specifying target services for a component reference described in section 5.3.1).
[ASM60034]	For a composite used as a component implementation, all properties of components within the composite, where the underlying component implementation specifies "mustSupply=true" for the property, MUST either specify a value for the property or source the value from a composite property.
[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

**Formatted:** Font color: Red

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

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

**Comment [ME99]:** Issue 51

	(multiplicity 1..1 or 1..n) or additional non-optional properties (a property with mustSupply=true).
<del>[ASM70006]</del>	<del>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.</del>
<del>[ASM70007]</del>	<del>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.</del>
[ASM80001]	The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.0 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 sure that no modification of input messages by the service or post-invocation modifications to return messages are seen by the caller.
[ASM80004]	If a reference is defined using a bidirectional interface element, the client component implementation using the reference calls the referenced service using the interface. The client MUST provide an implementation of the callback interface.
[ASM80005]	Either both interfaces of a bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and remote services.
[ASM80006]	Where a service or a reference has a conversational interface, the conversational intent MUST be attached either to the interface itself, or to the service or reference using the interface.
[ASM80007]	Once an operation marked with endsConversation has been invoked, any subsequent attempts to call an operation or a callback operation associated with the same conversation MUST generate a sca:ConversationViolation fault.
[ASM80008]	Any service or reference that uses an interface marked with required intents MUST implicitly add those intents to its own @requires list.
[ASM80009]	<u>In a bidirectional interface, the service interface can have more than one operation defined, and the callback interface can also have more than one operation defined. SCA runtimes MUST allow an invocation of any operation on the service interface to be followed by zero, one or many invocations of any of the operations on the callback interface.</u>
[ASM80010]	<u>Whenever an interface document declaring a callback interface is used in the declaration of an &lt;interface/&gt; element in SCA, it MUST be treated as being bidirectional with the declared callback interface.</u>
[ASM80011]	<u>If an &lt;interface/&gt; element references an interface document</u>

Deleted: [ASM70006]

Deleted: [ASM70007]

Comment [ME100]: Issue 77

Deleted: OR an interface element of a WSDL 2.0 document

Comment [ME101]: Issue 56

Comment [ME102]: Issue 89

	which declares a callback interface and also itself contains a declaration of a callback interface, the two callback interfaces MUST be compatible.	
[Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface. [ASM80012]	Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface.	Comment [ME103]: Issue 89
[ASM80013]	If the service or reference declaration in the component type declares an interface without a callback interface, then the component configuration for the corresponding service or reference MUST NOT declare an interface with a callback interface.	Comment [ME104]: Issue 89
[ASM80014]	Where a composite declares an interface for a composite service or a composite reference, if the promoted service or promoted reference has an interface which has a callback interface, then the interface declaration for the composite service or the composite reference MUST also declare a compatible interface with a compatible callback interface.	Comment [ME105]: Issue 89
[ASM80015]	If the promoted service or promoted reference has an interface without a callback interface, then the interface declaration for the composite service or composite reference MUST NOT declare a callback interface.	Comment [ME106]: Issue 89
[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).	Comment [ME107]: Issue 89
[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.	Comment [mbgl108]: Issue 57
[ASM90004]	a wire target MAY be specified with a syntax of "componentName/serviceName/bindingName".	
[ASM10001]	all of the QNames for the definitions contained in definitions.xml files MUST be unique within the domain.	Comment [ME109]: Issue 44
[ASM12001]	For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA as a hierarchy of resources	Comment [ME110]: Issue 44

	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]	<del>Where present, these mechanisms MUST be used by the SCA runtime to resolve artifact dependencies.</del>
<del>[ASM12006]</del>	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.
<del>[ASM12010]</del>	Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.
<del>[ASM12011]</del>	If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an alternative.
[ASM12012]	The value of @autowire for the logical domain composite MUST be autowire="false".
[ASM12013]	For components at the Domain level, with References for which @autowire="true" applies, the behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms: 1) The SCA runtime MAY disallow deployment of any components with autowire References. In this case, the SCA runtime MUST generate an exception at the point where the component is deployed. 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component is deployed and not update those targets when later deployment actions occur. 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later deployment actions occur resulting in updated reference targets which match the new Domain configuration. How the new configuration of the reference takes place is described by the relevant client and implementation

**Deleted:** Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact dependencies.

**Deleted:** [ASM12006]

**Deleted:** [ASM12010]

**Deleted:** [ASM12011]

**Comment [mbgl111]:** Issue 42

**Comment [mbgl112]:** Issue 40

	specifications.
[ASM12014]	Where <wire/> elements are added, removed or replaced by deployment actions, the components whose references are affected by those deployment actions MAY have their references updated by the SCA runtime dynamically without the need to stop and start those components.
[ASM12015]	Where components are updated by deployment actions (their configuration is changed in some way, which may include changing the wires of component references), the new configuration MUST apply to all new instances of those components once the update is complete.
[ASM12016]	An SCA runtime MAY choose to maintain existing instances with the old configuration of components updated by deployment actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.
[ASM12017]	<p>Where a component that is the target of a wire is removed, without the wire being changed, then future invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is the result of the autowire process, the SCA runtime MUST:</p> <ul style="list-style-type: none"> <li>• either cause future invocation of the target component's services to fail with a ServiceUnavailable fault</li> <li>• or alternatively, if an alternative target component is available that satisfies the autowire process, update the reference of the source component</li> </ul>
[ASM12018]	Where a component that is the target of a wire is updated, future invocations of that reference SHOULD use the updated component.
[ASM12019]	Where an existing domain level component is updated, an SCA runtime MAY maintain a copy of a component offering a conversational service until all existing conversations complete - alternatively all existing conversations MAY be terminated.
[ASM12020]	<p>Where a component is added to the domain that is a potential target for a domain level component reference where that reference is marked as @autowire=true, the SCA runtime MUST:</p> <ul style="list-style-type: none"> <li>- either update the references for the source component once the new component is running.</li> </ul> <p>or alternatively, defer the updating of the references of the source component until the source component is stopped and restarted.</p>
[ASM12021]	The SCA runtime MUST raise an error if an artifact cannot be resolved using these mechanisms, if present.
[ASM12022]	There can be multiple import declarations for a given namespace. Where multiple import declarations are made for the same namespace, all the locations specified MUST be searched in lexical order.
[ASM12023]	When a contribution contains a reference to an artifact from a namespace that is declared in an import statement of the

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Font color: Red

Formatted: Font color: Red

	<p>contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the SCA runtime MUST resolve artifacts in the following order:</p> <ol style="list-style-type: none"> <li>1. _____ from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to locate artifacts (ie only a one-level search is performed).</li> <li>2. _____ from the contents of the contribution itself.</li> </ol>
[ASM12024]	The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through resolving an import statement.
[ASM12025]	The SCA runtime MUST raise an error if an artifact cannot be resolved by the precedence order above.

Formatted: Font color: Red

Formatted: Font color: Red

Formatted: Bullets and Numbering

4996

## **D. Acknowledgements**

4997  
4998

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

4999

### **Participants:**

5000

[Participant Name, Affiliation | Individual Member]

5001

[Participant Name, Affiliation | Individual Member]

5002

## E. Non-Normative Text

Formatted: Bullets and Numbering

5004

## F. Revision History

5005

[optional; should not be included in OASIS Standards]

5006

Revision	Date	Editor	Changes Made
1	2007-09-24	Anish Karmarkar	Applied the OASIS template + related changes to the Submission
2	2008-01-04	Michael Beisiegel	<p>composite section</p> <ul style="list-style-type: none"> <li>- changed order of subsections from property, reference, service to service, reference, property</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> <li>- added section in appendix to contain complete pseudo schema of composite</li> </ul> <p>- moved component section after implementation section</p> <ul style="list-style-type: none"> <li>- made the ConstrainingType section a top level section</li> <li>- moved interface section to after constraining type section</li> </ul> <p>component section</p> <ul style="list-style-type: none"> <li>- added subheadings for Implementation, Service, Reference, Property</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> </ul> <p>implementation section</p> <ul style="list-style-type: none"> <li>- changed title to "Implementation and ComponentType"</li> <li>- moved implementation instance related stuff from implementation section to component implementation section</li> <li>- added subheadings for Service, Reference, Property, Implementation</li> <li>- progressive disclosure of pseudo schemas, each section only shows what is described</li> <li>- attributes description now starts with name : type (cardinality)</li> <li>- child element description as list, each item starting with name : type (cardinality)</li> <li>- attribute and element description still needs to be completed, all implementation statements</li> </ul>

			<p>on services, references, and properties should go here</p> <ul style="list-style-type: none"> <li>- added complete pseudo schema of componentType in appendix</li> <li>- added "Quick Tour by Sample" section, no content yet</li> <li>- added comment to introduction section that the following text needs to be added <ul style="list-style-type: none"> <li>"This specification is defined in terms of infoset and not XML 1.0, even though the spec uses XML 1.0/1.1 terminology. A mapping from XML to infoset (... link to infoset specification ...) is trivial and should be used for non-XML serializations."</li> </ul> </li> </ul>
3	2008-02-15	Anish Karmarkar Michael Beisiegel	<p>Incorporated resolutions from 2008 Jan f2f.</p> <ul style="list-style-type: none"> <li>- issue 9</li> <li>- issue 19</li> <li>- issue 21</li> <li>- issue 4</li> <li>- issue 1A</li> <li>- issue 27</li> <li>- in Implementation and ComponentType section added attribute and element description for service, reference, and property</li> <li>- removed comments that helped understand the initial restructuring for WD02</li> <li>- added changes for issue 43</li> <li>- added changes for issue 45, except the changes for policySet and requires attribute on property elements</li> <li>- used the NS <a href="http://docs.oasis-open.org/ns/opencsa/sca/200712">http://docs.oasis-open.org/ns/opencsa/sca/200712</a></li> <li>- updated copyright stmt</li> <li>- added wordings to make PDF normative and xml schema at the NS uri authoritative</li> </ul>
4	2008-04-22	Mike Edwards	<p>Editorial tweaks for CD01 publication:</p> <ul style="list-style-type: none"> <li>- updated URL for spec documents</li> <li>- removed comments from published CD01 version</li> <li>- removed blank pages from body of spec</li> </ul>
5	2008-06-30	Anish Karmarkar Michael Beisiegel	<p>Incorporated resolutions of issues: 3, 6, 14 (only as it applies to the component property element), 23, 25, 28, 25, 38, 39, 40, 42, 45 (except for adding @requires and @policySets to property elements), 57, 67, 68, 69</p>
6	2008-09-23	Mike Edwards	<p>Editorial fixes in response to Mark Combella's review contained in email: <a href="http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html">http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html</a></p>
7 CD01 - Rev3	2008-11-18	Mike Edwards	<ul style="list-style-type: none"> <li>• Specification marked for conformance statements. New Appendix (D) added</li> </ul>

			containing a table of all conformance statements. Mass of related minor editorial changes to remove the use of RFC2119 words where not appropriate.
8 CD01 - Rev4	2008-12-11	Mike Edwards	<ul style="list-style-type: none"> <li>- Fix problems of misplaced statements in Appendix D</li> <li>- Fixed problems in the application of Issue 57 - section 5.3.1 &amp; Appendix D as defined in email: <a href="http://lists.oasis-open.org/archives/sca-assembly/200811/msg00045.html">http://lists.oasis-open.org/archives/sca-assembly/200811/msg00045.html</a></li> <li>- Added Conventions section, 1.3, as required by resolution of Issue 96.</li> <li>- Issue 32 applied - section B2</li> <li>- Editorial addition to section 8.1 relating to no operation overloading for remotable interfaces, as agreed at TC meeting of 16/09/2008.</li> </ul>
9 CD01 - Rev5	2008-12-22	Mike Edwards	<ul style="list-style-type: none"> <li>- Schemas in Appendix B updated with resolutions of Issues 32 and 60</li> <li>- Schema for contributions - Appendix B12 - updated with resolutions of Issues 53 and 74.</li> <li>- Issues 53 and 74 incorporated - Sections 11.4, 11.5</li> </ul>
10 CD01-Rev6	2008-12-23	Mike Edwards	<ul style="list-style-type: none"> <li>- Issues 5, 71, 92</li> <li>- Issue 14 - remaining updates applied to ComponentType (section 4.1.3) and to Composite Property (section 6.3)</li> </ul>
<a href="#">11 CD01-Rev7</a>	<a href="#">2008-12-23</a>	<a href="#">Mike Edwards</a>	<p><a href="#">All changes accepted before revision from Rev6 started - due to changes being applied to previously changed sections in the Schemas</a></p> <ul style="list-style-type: none"> <li><a href="#">Issues 12 &amp; 18 - Section B2</a></li> <li><a href="#">Issue 63 - Section C3</a></li> <li><a href="#">Issue 75 - Section C12</a></li> <li><a href="#">Issue 65 - Section 7.0</a></li> <li><a href="#">Issue 77 - Section 8 + Appendix D</a></li> <li><a href="#">Issue 69 - Sections 5.1, 8</a></li> <li><a href="#">Issue 45 - Sections 4.1.3, 5.4, 6.3, B2.</a></li> <li><a href="#">Issue 56 - Section 8.2, Appendix D</a></li> <li><a href="#">Issue 41 - Sections 5.3.1, 6.4, 12.7, 12.8, Appendix D</a></li> </ul>
<a href="#">12 CD01-Rev8</a>	<a href="#">2008-12-30</a>	<a href="#">Mike Edwards</a>	<ul style="list-style-type: none"> <li><a href="#">Issue 72 - Removed Appendix A</a></li> <li><a href="#">Issue 79 - Sections 9.0, 9.2, 9.3, Appendix A.2</a></li> <li><a href="#">Issue 62 - Sections 4.1.3, 5.4</a></li> <li><a href="#">Issue 26 - Section 6.5</a></li> <li><a href="#">Issue 51 - Section 6.5</a></li> <li><a href="#">Issue 36 - Section 4.1</a></li> <li><a href="#">Issue 44 - Section 10, Appendix C</a></li> <li><a href="#">Issue 89 - Section 8.2, 8.5, Appendix A, Appendix C</a></li> <li><a href="#">Issue 16 - Section 6.8, 9.4</a></li> <li><a href="#">Issue 8 - Section 11.2.1</a></li> <li><a href="#">Issue 17 - Section 6.6</a></li> <li><a href="#">Issue 30 - Sections 4.1.1, 4.1.2, 5.2, 5.3, 6.1, 6.2, 9</a></li> <li><a href="#">Issue 33 - insert new Section 8.4</a></li> </ul>

5007

**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.

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

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

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

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

```
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"?
```

```
name="xs:NCName" target="list of xs:anyURI"?
```

```
promote="list of xs:anyURI"? wiredByImpl="xs:boolean"?
multiplicity="0..1 or 1..1 or 0..n or 1..n"?
requires="list of xs:QName"? policySets="list of xs:QName"?
```

## Constructing Hierarchical URIs

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

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

Each of these components deserves addition definition:

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

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

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

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

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

Base Domain URI for a scheme / Service Binding URI

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

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

http://acme.com/stocksComponent/getQuote

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

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

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

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

---

# Pseudo Schema

## ComponentType

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  constrainingType="QName"? >

  <service name="xs:NCName" requires="list of xs:QName"?
    policySets="list of xs:QName"?>*
    <interface ... />
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?/>*
    <callback?
      <binding ... />+
    </callback>
  </service>

  <reference name="xs:NCName"
    target="list of xs:anyURI"? autowire="xs:boolean"?
    multiplicity="0..1 or 1..1 or 0..n or 1..n"?
    wiredByImpl="xs:boolean"? requires="list of xs:QName"?
    policySets="list of xs:QName"?>*
    <interface ... />
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?/>*
    <callback?
      <binding ... />+
    </callback>
  </reference>

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

```

    <implementation requires="list of xs:QName"?
        policySets="list of xs:QName"?/>?

</componentType>

```

## Composite

```

<?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 name="xs:QName"/>*

    <service name="xs:NCName" promote="xs:anyURI"
        requires="list of xs:QName"? policySets="list of
xs:QName"?>*
        <interface ... />?
        <binding uri="xs:anyURI"? name="xs:NCName"?
            requires="list of xs:QName"? policySets="list of
xs:QName"?/>*
        <callback?
            <binding uri="xs:anyURI"? name="xs:NCName"?
                requires="list of xs:QName"?
                policySets="list of xs:QName"?/>+
        </callback>
    </service>

    <reference name="xs:NCName" target="list of xs:anyURI"?
        promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?
        requires="list of xs:QName"? policySets="list of
xs:QName"?>*
        <interface ... />?
        <binding uri="xs:anyURI"? name="xs:NCName"?
            requires="list of xs:QName"? policySets="list of
xs:QName"?/>*
        <callback?
            <binding uri="xs:anyURI"? name="xs:NCName"?

```

```

        requires="list of xs:QName"?
        policySets="list of xs:QName"?/>>+
    </callback>
</reference>

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

<component name="xs:NCName" autowire="xs:boolean"?
    requires="list of xs:QName"? policySets="list of
xs:QName"?>*
    <implementation ... />?
    <service name="xs:NCName" requires="list of xs:QName"?
        policySets="list of xs:QName"?>*
        <interface ... />?
        <binding uri="xs:anyURI"? name="xs:NCName"?
            requires="list of xs:QName"?
            policySets="list of xs:QName"?/>>*
        <callback>?
            <binding uri="xs:anyURI"? name="xs:NCName"?
                requires="list of xs:QName"?
                policySets="list of xs:QName"?/>>+
        </callback>
    </service>
    <property name="xs:NCName" (type="xs:QName" |
element="xs:QName")
        source="xs:string"? file="xs:anyURI"?
value="xs:string"?>*
        [<value>+ | xs:any+]?
    </property>
    <reference name="xs:NCName" target="list of xs:anyURI"?
        autowire="xs:boolean"? wiredByImpl="xs:boolean"?
        requires="list of xs:QName"? policySets="list of
xs:QName"?
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>>*
    <interface ... />?
    <binding uri="xs:anyURI"? name="xs:NCName"?
        requires="list of xs:QName"?
        policySets="list of xs:QName"?/>>*
    <callback>?
        <binding uri="xs:anyURI"? name="xs:NCName"?

```

```

        requires="list of xs:QName"?
        policySets="list of xs:QName"?/>+
    </callback>
</reference>
</component>

<wire source="xs:anyURI" target="xs:anyURI" />*

</composite>

```

**Page 107: [7] Deleted**

**Mike Edwards**

**12/23/2008 1:32:00 PM**

```

<!-- Reference -->
<complexType name="Reference">
  <complexContent>
    <extension base="sca:CommonExtensionBase">
      <sequence>
        <element ref="sca:interface" minOccurs="0"/>
        <element name="operation" type="sca:Operation"
minOccurs="0"
          maxOccurs="unbounded"/>
        <element ref="sca:binding" minOccurs="0"
minOccurs="0"
          maxOccurs="unbounded"/>
        <element ref="sca:callback" minOccurs="0"/>
        <any namespace="##other" processContents="lax"
minOccurs="0"
          maxOccurs="unbounded"/>
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="autowire" type="boolean" use="optional"/>
      <attribute name="target" type="sca:listOfAnyURIs"
        use="optional"/>
      <attribute name="wiredByImpl" type="boolean" use="optional"
        default="false"/>
      <attribute name="multiplicity" type="sca:Multiplicity"
        use="optional" default="1.1"/>
      <attribute name="promote" type="sca:listOfAnyURIs"
        use="required"/>
      <attribute name="requires" type="sca:listOfQNames"
        use="optional"/>
      <attribute name="policySets" type="sca:listOfQNames"
        use="optional"/>
    </extension>
  </complexContent>
</complexType>

```

**Page 110: [8] Deleted**

**Mike Edwards**

**12/23/2008 1:33:00 PM**

```

<!-- Component Reference -->
<complexType name="ComponentReference">
  <complexContent>
    <restriction base="sca:Reference">
      <sequence>
        <element ref="sca:documentation" minOccurs="0"
maxOccurs="unbounded"/>

```

```

        <element ref="sca:interface" minOccurs="0"/>
        <element name="operation" type="sca:Operation"
minOccurs="0"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <element ref="sca:binding" minOccurs="0"
            minOccurs="0"
            maxOccurs="unbounded"/>
        <element ref="sca:callback" minOccurs="0"/>
        <any namespace="##other" processContents="lax"
minOccurs="0"
            minOccurs="0"
            maxOccurs="unbounded"/>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="autowire" type="boolean" use="optional"/>
    <attribute name="target" type="sca:listOfAnyURIs"
        use="optional"/>
    <attribute name="wiredByImpl" type="boolean" use="optional"
        default="false"/>
    <attribute name="multiplicity" type="sca:Multiplicity"
        use="optional" default="1..1"/>
    <attribute name="requires" type="sca:listOfQNames"
        use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames"
        use="optional"/>
    <anyAttribute namespace="##other" processContents="lax"/>
</restriction>
</complexContent>
</complexType>

```