



---

# Service Component Architecture Assembly Model Specification Version 1.1

**Committee Draft 02**

**14<sup>th</sup> January 2009**

**Specification URIs:**

**This Version:**

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

**Previous Version:**

**Latest Version:**

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

**Latest Approved Version:**

**Technical Committee:**

[OASIS Service Component Architecture / Assembly \(SCA-Assembly\) TC](#)

**Chair(s):**

Martin Chapman, Oracle  
Mike Edwards, IBM

**Editor(s):**

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

**Related work:**

This specification replaces or supercedes:

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

This specification is related to:

- Service Component Architecture Policy Framework Specification Version 1.1

**Declared XML Namespace(s):**

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

**Abstract:**

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

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

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

This document describes the SCA Assembly Model, which covers

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

**Status:**

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

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

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

---

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

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

---

## Notices

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

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

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

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

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

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

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

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

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

---

## Table of Contents

Committee Draft 02 .....	1
The non-normative errata page for this specification is located at <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 .....	11
2.1 Diagram used to Represent SCA Artifacts .....	12
3 Quick Tour by Sample .....	14
4 Implementation and ComponentType .....	15
4.1 Component Type .....	15
4.1.1 Service .....	16
4.1.2 Reference .....	17
4.1.3 Property .....	19
4.1.4 Implementation .....	20
4.2 Example ComponentType .....	21
4.3 Example Implementation .....	22
5 Component .....	25
5.1 Implementation .....	26
5.2 Service .....	27
5.3 Reference .....	29
5.3.1 Specifying the Target Service(s) for a Reference .....	31
5.4 Property .....	32
5.5 Example Component .....	35
6 Composite .....	39
6.1 Service .....	41
6.1.1 Service Examples .....	42
6.2 Reference .....	44
6.2.1 Example Reference .....	46
6.3 Property .....	48
6.3.1 Property Examples .....	49
6.4 Wire .....	51
6.4.1 Wire Examples .....	53
6.4.2 Autowire .....	55
6.4.3 Autowire Examples .....	56
6.5 Using Composites as Component Implementations .....	59
6.5.1 Example of Composite used as a Component Implementation .....	60
6.6 Using Composites through Inclusion .....	60
6.6.1 Included Composite Examples .....	61
6.7 Composites which Include Component Implementations of Multiple Types .....	64

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

12.4.3 remove Contribution .....	101	
12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts .....	101	
12.6 Domain-Level Composite .....	101	
12.6.1 add To Domain-Level Composite.....	102	
12.6.2 remove From Domain-Level Composite .....	102	
12.6.3 get Domain-Level Composite .....	102	
12.6.4 get QName Definition .....	102	
12.7 Dynamic Behaviour of Wires in the SCA Domain .....	103	
12.8 Dynamic Behaviour of Component Property Values .....	103	
13 Conformance .....	105	
A. XML Schemas .....	<del>107</del>	<b>Deleted: 106</b>
A.1 sca.xsd .....	<del>107</del>	<b>Deleted: 106</b>
A.2 sca-core.xsd .....	<del>107</del>	<b>Deleted: 106</b>
A.3 sca-binding-sca.xsd.....	<del>115</del>	<b>Deleted: 114</b>
A.4 sca-interface-java.xsd .....	<del>116</del>	<b>Deleted: 115</b>
A.5 sca-interface-wsdl.xsd .....	<del>116</del>	<b>Deleted: 115</b>
A.6 sca-implementation-java.xsd .....	<del>117</del>	<b>Deleted: 116</b>
A.7 sca-implementation-composite.xsd .....	<del>118</del>	<b>Deleted: 117</b>
A.8 sca-definitions.xsd .....	<del>118</del>	<b>Deleted: 117</b>
A.9 sca-binding-webservice.xsd .....	<del>119</del>	<b>Deleted: 118</b>
A.10 sca-binding-jms.xsd.....	<del>119</del>	<b>Deleted: 118</b>
A.11 sca-policy.xsd .....	<del>119</del>	<b>Deleted: 118</b>
A.12 sca-contribution.xsd .....	<del>119</del>	<b>Deleted: 118</b>
B. SCA Concepts .....	<del>122</del>	<b>Deleted: 121</b>
B.1 Binding.....	<del>122</del>	<b>Deleted: 121</b>
B.2 Component.....	<del>122</del>	<b>Deleted: 121</b>
B.3 Service.....	<del>122</del>	<b>Deleted: 121</b>
B.3.1 Remotable Service.....	<del>122</del>	<b>Deleted: 121</b>
B.3.2 Local Service.....	<del>123</del>	<b>Deleted: 121</b>
B.4 Reference .....	<del>123</del>	<b>Deleted: 122</b>
B.5 Implementation .....	<del>123</del>	<b>Deleted: 122</b>
B.6 Interface.....	<del>123</del>	<b>Deleted: 122</b>
B.7 Composite .....	<del>124</del>	<b>Deleted: 122</b>
B.8 Composite inclusion .....	<del>124</del>	<b>Deleted: 123</b>
B.9 Property .....	<del>124</del>	<b>Deleted: 123</b>
B.10 Domain .....	<del>124</del>	<b>Deleted: 123</b>
B.11 Wire .....	<del>124</del>	<b>Deleted: 123</b>
C. Conformance Items .....	<del>126</del>	<b>Deleted: 123</b>
D. Acknowledgements .....	<del>134</del>	<b>Deleted: 125</b>
E. Non-Normative Text .....	<del>135</del>	<b>Deleted: 133</b>
F. Revision History.....	<del>136</del>	<b>Deleted: 134</b>
		<b>Deleted: 135</b>

---

# 1 Introduction

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

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

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

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

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

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

## 1.1 Terminology

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

## 1.2 Normative References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.

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

[SCA-Common-Java] SCA Java Common Annotations and APIs Specification  
[http://www.osoa.org/download/attachments/35/SCA\\_JavaAnnotationsAndAPIs\\_V100.pdf](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>

- 39 [5] WS-I Basic Profile  
40 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>  
41
- 42 [6] WS-I Basic Security Profile  
43 <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>  
44
- 45 [7] Business Process Execution Language (BPEL)  
46 [http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel)  
47
- 48 [8] WSDL Specification  
49 WSDL 1.1: <http://www.w3.org/TR/wsd>  
50 WSDL 2.0: <http://www.w3.org/TR/wsd20/>  
51
- 52 [9] SCA Web Services Binding Specification  
53 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-wsbinding-1.1-spec-cd01.pdf>  
54
- 55 [10] SCA Policy Framework Specification  
56 <http://docs.oasis-open.org/opencsa/sca-policy/sca-policy-1.1-spec-cd-01.pdf>  
57
- 58 [11] SCA JMS Binding Specification  
59 <http://docs.oasis-open.org/opencsa/sca-bindings/sca-jmsbinding-1.1-spec-cd01.pdf>  
60
- 61 [SCA-CPP-Client] SCA C++ Client and Implementation Specification  
62 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-cppcni-1.1-spec-cd-01.pdf>  
63
- 64 [SCA-C-Client] SCA C Client and Implementation Specification  
65 <http://docs.oasis-open.org/opencsa/sca-c-cpp/sca-ccni-1.1-spec-cd-01.pdf>  
66
- 67 [12] ZIP Format Definition  
68 <http://www.pkware.com/documents/casestudies/APPNOTE.TXT>  
69
- 70 [13] Infoset Specification  
71 <http://www.w3.org/TR/xml-infoset/>  
72
- 73 [WSDL11\_Identifiers] WSDL 1.1 Element Identifiers  
74 <http://www.w3.org/TR/wsd11elementidentifiers/>  
75

## 76 1.3 Naming Conventions

77

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

79 as follows:

80

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

---

## 91 2 Overview

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

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

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

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

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

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

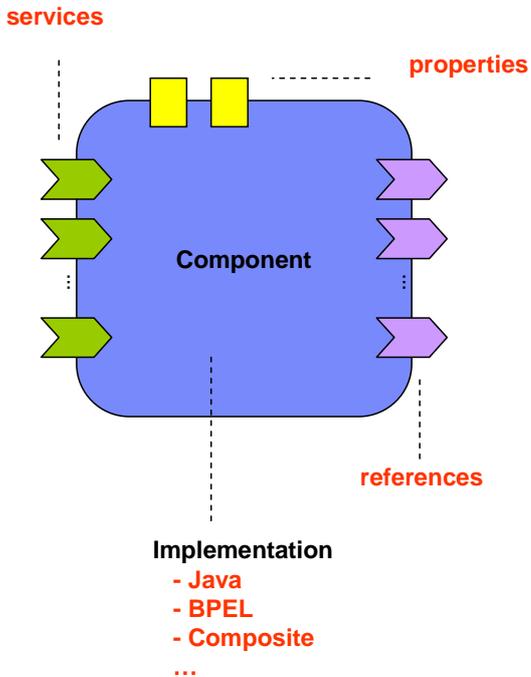
129 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services  
130 providing an area of business functionality that is controlled by a single organization. As an example, for  
131 the accounts department in a business, the SCA Domain might cover all financial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

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

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

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

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



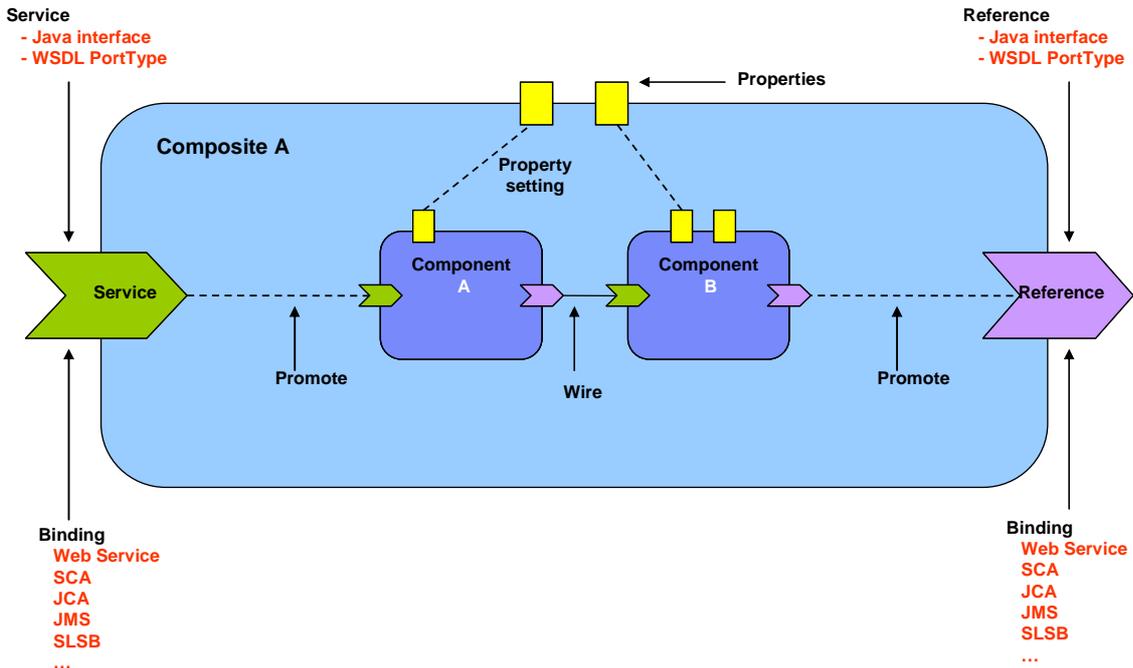
146

147 *Figure 1: SCA Component Diagram*

148

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

151

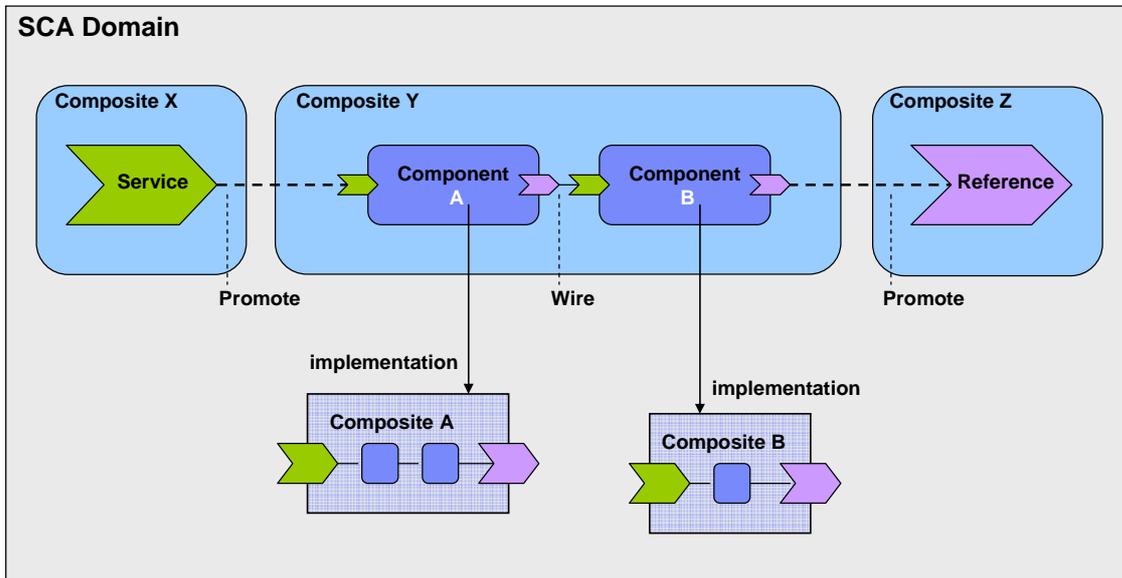


152

153 *Figure 2: SCA Composite Diagram*

154

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



157

158 *Figure 3: SCA Domain Diagram*

---

159 **3 Quick Tour by Sample**

160 To be completed.

161

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

163

164

---

## 165 4 Implementation and ComponentType

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

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

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

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

179 So, for example:

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

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

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

194

### 195 4.1 Component Type

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

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

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

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

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

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

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

224 The following snippet shows the componentType pseudo-schema:

225

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

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

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

244

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

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

251

## 252 4.1.1 Service

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

257

```

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

```

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

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

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

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

## 4.1.2 Reference

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

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

312

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

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

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

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

364

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

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

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

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

389

### 390 4.1.3 Property

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

395

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

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

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

- 416 ▪ **name : NCName (1..1)** - the name of the property. The @name attribute of a  
417 <property/> child element of a <componentType/> MUST be unique amongst the  
418 property elements of that <componentType/>. [ASM40005]
- 419 ▪ one of **(1..1)**:
  - 420 ○ **type : QName** - the type of the property defined as the qualified name of an XML  
421 schema type. The value of the property @type attribute MUST be the QName of  
422 an XML schema type. [ASM40007]
  - 423 ○ **element : QName** - the type of the property defined as the qualified name of an  
424 XML schema global element – the type is the type of the global element. The value  
425 of the property @element attribute MUST be the QName of an XSD global  
426 element. [ASM40008]
- 427 ▪ **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-  
428 valued (true). In the case of a multi-valued property, it is presented to the implementation  
429 as a collection of property values. If many is not specified, it takes a default value of false.
- 430 ▪ **mustSupply : boolean (0..1)** - whether the property value must be supplied by the  
431 component that uses the implementation – when mustSupply="true" the component must  
432 supply a value since the implementation has no default value for the property. A default-  
433 property-value should only be supplied when mustSupply="false" (the default setting for  
434 the mustSupply attribute), since the implication of a default value is that it is used only  
435 when a value is not supplied by the using component. If mustSupply is not specified, it  
436 takes a default value of false.
- 437 ▪ **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property.
- 438 ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
439 [10] for a description of this attribute.
- 440 ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
441 [10] for a description of this attribute.

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

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

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

457

#### 458 4.1.4 Implementation

459 **Implementation** represents characteristics inherent to the implementation itself, in particular  
460 intents and policies. See the [Policy Framework specification](#) [10] for a description of intents and

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

463

```
464 <?xml version="1.0" encoding="ASCII"?>
465 <!-- Component type implementation schema snippet -->
466 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
467 >
468
469     <service ... /*>
470     <reference ... /*>
471     <property ... /*>
472
473     <implementation requires="list of xs:QName"?
474         policySets="list of xs:QName"?/>?
475
476 </componentType>
477
```

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

- 479 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
480 [\[10\]](#) for a description of this attribute.
- 481 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
482 [\[10\]](#) for a description of this attribute.

483

## 484 4.2 Example ComponentType

485

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

489

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

### 509 4.3 Example Implementation

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

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

```
514  
515 package services.account;  
516  
517 @Remotable  
518 public interface AccountService {  
519  
520     AccountReport getAccountReport(String customerID);  
521 }  
522
```

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

```
526  
527 package services.account;  
528  
529 import java.util.List;  
530  
531 import commonj.sdo.DataFactory;  
532  
533 import org.osoa.sca.annotations.Property;  
534 import org.osoa.sca.annotations.Reference;  
535  
536 import services.accountdata.AccountDataService;  
537 import services.accountdata.CheckingAccount;  
538 import services.accountdata.SavingsAccount;  
539 import services.accountdata.StockAccount;  
540 import services.stockquote.StockQuoteService;  
541  
542 public class AccountServiceImpl implements AccountService {  
543  
544     @Property  
545     private String currency = "USD";  
546  
547     @Reference  
548     private AccountDataService accountDataService;  
549     @Reference  
550     private StockQuoteService stockQuoteService;  
551  
552     public AccountReport getAccountReport(String customerID) {  
553  
554         DataFactory dataFactory = DataFactory.INSTANCE;  
555         AccountReport accountReport = (AccountReport) dataFactory.create(AccountReport.class);  
556         List accountSummaries = accountReport.getAccountSummaries();  
557
```

```

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

```

594  
595 The following is the equivalent SCA componentType definition for the AccountServiceImpl, derived  
596 by reflection against the code above:

```

597
598 <?xml version="1.0" encoding="ASCII"?>
599 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
600     xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
601
602     <service name="AccountService">
603         <interface.java interface="services.account.AccountService"/>
604     </service>
605     <reference name="accountDataService">
606         <interface.java
607 interface="services.accountdata.AccountDataService"/>

```

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

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

## 5 Component

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

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

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

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

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

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

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

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

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

## 669 5.1 Implementation

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

```
675
676 <?xml version="1.0" encoding="UTF-8"?>
677 <!-- Component Implementation schema snippet -->
678 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
679   ...
680   <component ... >*
681     <implementation ... />?
682     <service ... />*
683     <reference ... />*
684     <property ... />*
685   </component>
686   ...
687 </composite>
688
```

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

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

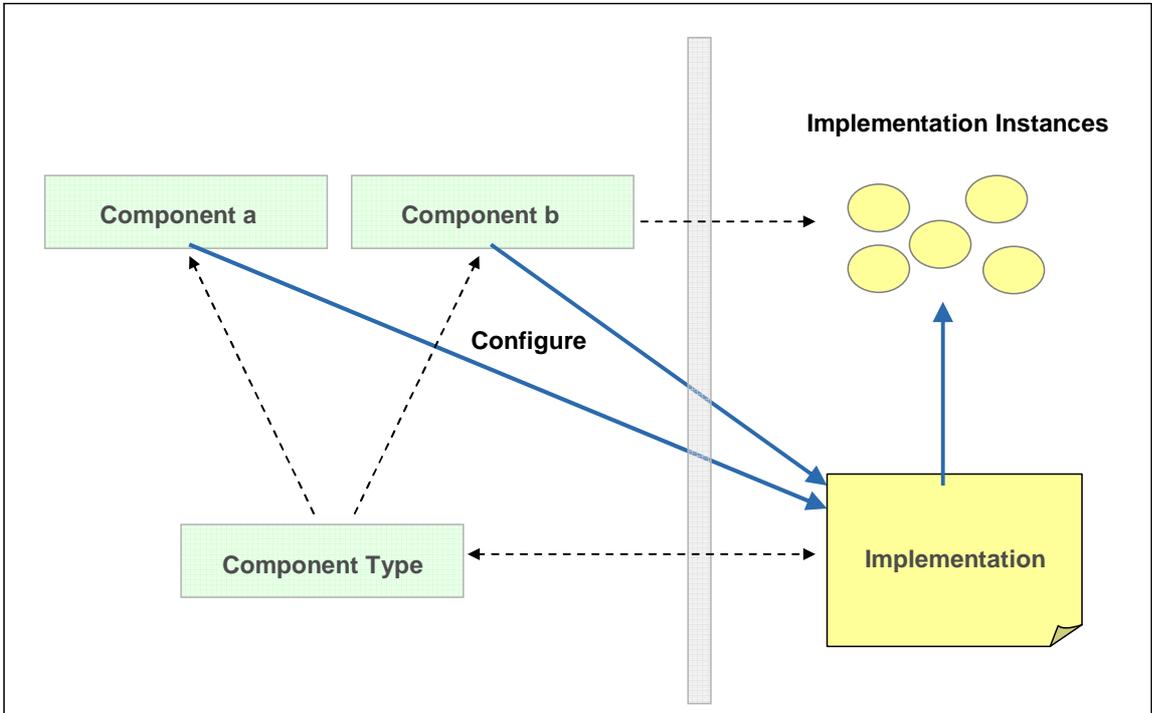
697 The following snippets show implementation elements for the Java and BPEL implementation types  
 698 and for the use of a composite as an implementation:

```
699
700 <implementation.java class="services.myvalue.MyValueServiceImpl"/>
701
702 <implementation.bpel process="ans:MoneyTransferProcess"/>
703
704 <implementation.composite name="bns:MyValueComposite"/>
705
706
```

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

708

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



714  
 715 *Figure 4: Relationship of Component and Implementation*

716

## 717 5.2 Service

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

722

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

```

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

```

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

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

758

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

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

787

## 788 5.3 Reference

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

793

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

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

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

- 838 ○ 0..1 – zero or one wire can have the reference as a source
- 839 ○ 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.

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

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

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

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

860  
 861 The **component reference** element has the following **child elements**:

- 862 • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes  
 863 the operations required by the reference. The interface is described by an **interface**  
 864 **element** which is a child element of the reference element. If no interface is specified,  
 865 then the interface specified for the reference in the componentType of the implementation  
 866 is in effect. If an interface is declared for a component reference it MUST provide a  
 867 compatible superset of the interface declared for the equivalent reference in the  
 868 componentType of the implementation, i.e. provide the same operations and a superset of  
 869 the operations defined by the implementation for the reference. [ASM50011] For details  
 870 on the interface element see [the Interface section](#).
  - 871 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are  
 872 used to describe characteristics of individual operations within the interface. For a detailed  
 873 description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
  - 874 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
 875 children. If no binding elements are specified for the reference, then the bindings specified  
 876 for the equivalent reference in the componentType of the implementation MUST be used,  
 877 but if the componentType also has no bindings specified, then <binding.sca/> MUST be  
 878 used as the binding. If binding elements are specified for the reference, then those  
 879 bindings MUST be used and they override any bindings specified for the equivalent  
 880 reference in the componentType of the implementation. [ASM50012] Details of the binding  
 881 element are described in the [Bindings section](#). The binding, combined with any PolicySets  
 882 in effect for the binding, needs to satisfy the set of policy intents for the reference, as  
 883 described in [the Policy Framework specification](#) [10].
- 884 A reference identifies zero or more target services that satisfy the reference. This can be  
 885 done in a number of ways, which are fully described in section "5.3.1 Specifying the  
 886 Target Service(s) for a Reference"
- 887 • **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional  
 888 **callback** element used if the interface has a callback defined, which has one or more  
 889 **binding** elements as children. The **callback** and its binding child elements are specified if  
 890 there is a need to have binding details used to handle callbacks. If the callback element is  
 891 present and contains one or more binding child elements, then those bindings MUST be

**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.

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

### 894 5.3.1 Specifying the Target Service(s) for a Reference

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

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

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

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

#### 931 5.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

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

- 938           • A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.  
939           [ASM50021]

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

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

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

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

**Deleted:** 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.

**Deleted:** 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.

**Deleted:** 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.

## 962 5.4 Property

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

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

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

974           For example,

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

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

980           For example,

- 981           • property defined using a XML Schema simple type and which contains a single  
982           value

```
983           <property name="pi">  
984                <value>3.14159265</value>  
985           </property>
```

- 986           • property defined using a XML Schema simple type and which contains multiple  
987           values

988                   <property name="currency">

989                    <value>EURO</value>

990                    <value>USDollar</value>

991                   </property>

992                   • property defined using a XML Schema complex type and which contains a single

993                    value

994                    <property name="complexFoo">

995                     <value attr="bar">

996                      <foo:a>TheValue</foo:a>

997                      <foo:b>InterestingURI</foo:b>

998                     </value>

999                    </property>

1000                   • property defined using a XML Schema complex type and which contains multiple

1001                    values

1002                    <property name="complexBar">

1003                     <value anotherAttr="foo">

1004                      <bar:a>AValue</bar:a>

1005                      <bar:b>InterestingURI</bar:b>

1006                     </value>

1007                     <value attr="zing">

1008                      <bar:a>BValue</bar:a>

1009                      <bar:b>BoringURI</bar:b>

1010                     </value>

1011                    </property>

1012                   • As a value, supplied as the content of the property element.

1013                    If a component property value is declared using a child element of the <property/>

1014                    element, the type of the property MUST be an XML Schema global element and the

1015                    declared child element MUST be an instance of that global element. [\[ASM50029\]](#)

1016                    For example,

1017                    • property defined using a XML Schema global element declaration and which

1018                    contains a single value

1019                    <property name="foo">

1020                     <foo:SomeGED ...>...</foo:SomeGED>

1021                    </property>

1022                    • property defined using a XML Schema global element declaration and which

1023                    contains multiple values

1024                    <property name="bar">

1025                     <bar:SomeOtherGED ...>...</bar:SomeOtherGED>

1026                     <bar:SomeOtherGED ...>...</bar:SomeOtherGED>

1027                    </property>

1028                    • By referencing a Property value of the composite which contains the component. The

1029                    reference is made using the **source** attribute of the property element.

1030

1031                    The form of the value of the source attribute follows the form of an XPath expression.

1032 This form allows a specific property of the composite to be addressed by name. Where the  
1033 composite property is of a complex type, the XPath expression can be extended to refer to  
1034 a sub-part of the complex property value.  
1035

1036 So, for example, `source="$currency"` is used to reference a property of the composite  
1037 called "currency", while `source="$currency/a"` references the sub-part "a" of the  
1038 complex composite property with the name "currency".

- 1039 • By specifying a dereferencable URI to a file containing the property value through the **file**  
1040 attribute. The contents of the referenced file are used as the value of the property.

1041

1042 If more than one property value specification is present, the source attribute takes precedence, then  
1043 the file attribute.

1044 For a property defined using a XML Schema simple type and for which a single value is desired, can  
1045 be set either using the @value attribute or the <value> child element. The two forms in such a case  
1046 are equivalent.

1047 When a property has multiple values set, they MUST all be contained within the same property  
1048 element. A <component/> element MUST NOT contain two <property/> subelements with the same  
1049 value of the @name attribute. [ASM50030]

1050 Optionally, the type of the property can be specified in **one** of two ways:

- 1051 • by the qualified name of a type defined in an XML schema, using the **type** attribute
- 1052 • by the qualified name of a global element in an XML schema, using the **element** attribute

1053 The property type specified must be compatible with the type of the property declared in the  
1054 component type of the implementation. If no type is declared in the component property, the type of  
1055 the property declared by the implementation is used.

1056

1057 The following snippet shows the component schema with the schema for a property child element:

1058

```
1059 <?xml version="1.0" encoding="UTF-8"?>
1060 <!-- Component Property schema snippet -->
1061 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1062   ...
1063   <component ... >*
1064     <implementation ... />?
1065     <service ... />*
1066     <reference ... />*
1067     <property name="xs:NCName"
1068       (type="xs:QName" | element="xs:QName"?
1069       mustSupply="xs:boolean"? many="xs:boolean"?
1070       source="xs:string"? file="xs:anyURI"?
1071       requires="list of xs:QName"?
1072       policySets="list of xs:QName"?
1073       value="xs:string"?>*
1074       [<value>+ | xs:any+ ]?
1075     </property>
1076   </component>
1077   ...
```

1078 </composite>

1079  
1080

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

- 1081     ▪ **name : NCName (1..1)** – the name of the property. The name attribute of a component  
1082     property MUST match the name of a property element in the component type of the  
1083     component implementation. [ASM50031]
- 1084     ▪ zero or one of **(0..1)**:
- 1085         ○ **type : QName** – the type of the property defined as the qualified name of an XML  
1086         schema type
- 1087         ○ **element : QName** – the type of the property defined as the qualified name of an  
1088         XML schema global element – the type is the type of the global element
- 1089     ▪ **source : string (0..1)** – an XPath expression pointing to a property of the containing  
1090     composite from which the value of this component property is obtained.
- 1091     ▪ **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
- 1092     ▪ **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or  
1093     multi-valued (true). Overrides the many specified for this property on the implementation.  
1094     The value can only be equal or further restrict, i.e. if the implementation specifies many  
1095     true, then the component can say false. In the case of a multi-valued property, it is  
1096     presented to the implementation as a Collection of property values. If many is not  
1097     specified, it takes the value defined by the component type of the implementation used by  
1098     the component.
- 1099     ▪ **value : string (0..1)** - the value of the property if the property is defined using a simple  
1100     type.
- 1101     ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
1102     [\[10\]](#) for a description of this attribute.
- 1103     ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
1104     [\[10\]](#) for a description of this attribute.

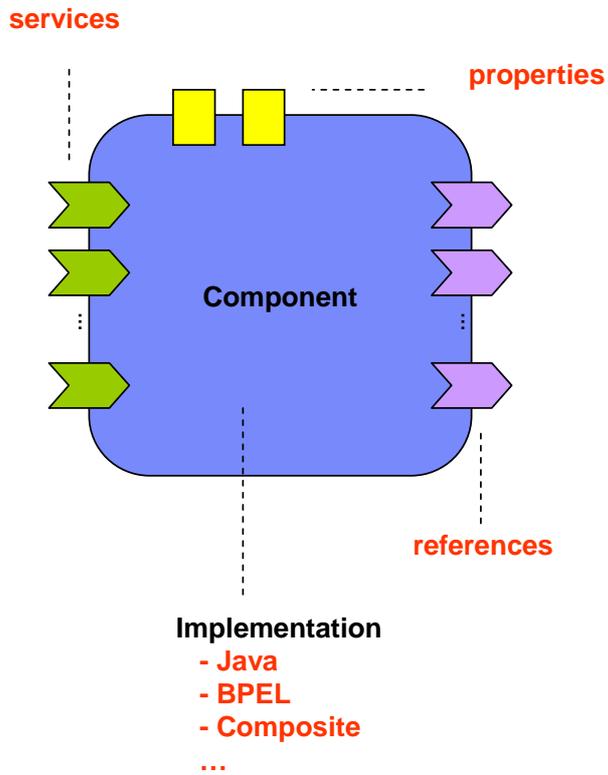
1105 The **component property** element has the following **child element**:

1106 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1107 property that is defined using a XML Schema type. If a property is single-valued, the <value/>  
1108 subelement MUST NOT occur more than once. [ASM50032] A property <value/> subelement MUST  
1109 NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]

## 1110 5.5 Example Component

1111

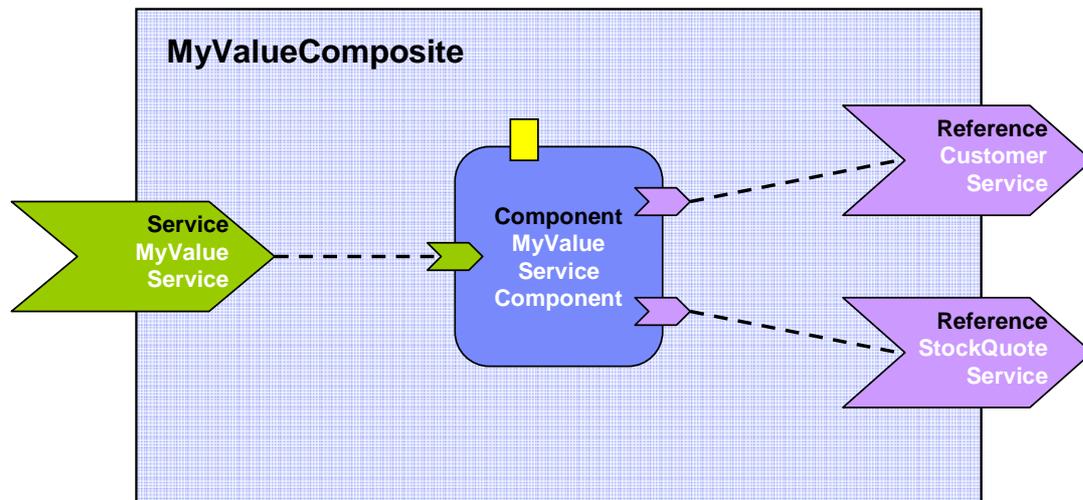
1112 The following figure shows the **component symbol** that is used to represent a component in an  
1113 assembly diagram.



1114  
1115  
1116  
1117  
1118

Figure 5: Component symbol

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



1119  
1120  
1121

Figure 6: Assembly diagram for MyValueComposite

1122

1123 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1124 containing the component element for the MyValueServiceComponent. A value is set for the  
1125 property named currency, and the customerService and stockQuoteService references are  
1126 promoted:

1127

```
1128 <?xml version="1.0" encoding="ASCII"?>
```

```
1129 <!-- MyValueComposite_1 example -->
```

```
1130 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
```

```
1131 targetNamespace="http://foo.com"
```

```
1132 name="MyValueComposite" >
```

1133

```
1134 <service name="MyValueService" promote="MyValueServiceComponent" />
```

1135

```
1136 <component name="MyValueServiceComponent">
```

```
1137 <implementation.java
```

```
1138 class="services.myvalue.MyValueServiceImpl" />
```

```
1139 <property name="currency">EURO</property>
```

```
1140 <reference name="customerService" />
```

```
1141 <reference name="stockQuoteService" />
```

```
1142 </component>
```

1143

```
1144 <reference name="CustomerService"
```

```
1145 promote="MyValueServiceComponent/customerService" />
```

1146

```
1147 <reference name="StockQuoteService"
```

```
1148 promote="MyValueServiceComponent/stockQuoteService" />
```

1149

```
1150 </composite>
```

1151

1152 Note that the references of MyValueServiceComponent are explicitly declared only for purposes of  
1153 clarity – the references are defined by the MyValueServiceImpl implementation and there is no  
1154 need to redeclare them on the component unless the intention is to wire them or to override some  
1155 aspect of them.

1156 The following snippet gives an example of the layout of a composite file if both the currency  
1157 property and the customerService reference of the MyValueServiceComponent are declared to be  
1158 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1159 <?xml version="1.0" encoding="ASCII"?>
```

```
1160 <!-- MyValueComposite_2 example -->
```

```
1161 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
```

```
1162 targetNamespace="http://foo.com"
```

```
1163 name="MyValueComposite" >
```

1164

```
1165 <service name="MyValueService" promote="MyValueServiceComponent" />
```

1166

```
1167     <component name="MyValueServiceComponent" >
1168         <implementation.java
1169 class="services.myvalue.MyValueServiceImpl" />
1170         <property name="currency">EURO</property>
1171         <property name="currency">Yen</property>
1172         <property name="currency">USDollar</property>
1173         <reference name="customerService"
1174             target="InternalCustomer/customerService" />
1175         <reference name="StockQuoteService" />
1176     </component>
1177
1178     ...
1179
1180     <reference name="CustomerService"
1181         promote="MyValueServiceComponent/customerService" />
1182
1183     <reference name="StockQuoteService"
1184         promote="MyValueServiceComponent/StockQuoteService" />
1185
1186 </composite>
1187
1188 ....this assumes that the composite has another component called InternalCustomer (not shown)
1189 which has a service to which the customerService reference of the MyValueServiceComponent is
1190 wired as well as being promoted externally through the composite reference CustomerService.
```

## 1191 6 Composite

1192 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of  
1193 composition within an SCA Domain. An **SCA composite** contains a set of components, services,  
1194 references and the wires that interconnect them, plus a set of properties which can be used to  
1195 configure components.

1196 Composites can be used as **component implementations** in higher-level composites – in other  
1197 words the higher-level composites can have components that are implemented by composites.  
1198 For more detail on the use of composites as component implementations see the section [Using  
1199 Composites as Component Implementations](#).

1200 The content of a composite can be used within another composite through **inclusion**. When a  
1201 composite is included by another composite, all of its contents are made available for use within  
1202 the including composite – the contents are fully visible and can be referenced by other elements  
1203 within the including composite. For more detail on the inclusion of one composite into another see  
1204 the section [Using Composites through Inclusion](#).

1205 A composite can be used as a unit of deployment. When used in this way, composites contribute  
1206 elements to an SCA domain. A composite can be deployed to the SCA domain either by inclusion,  
1207 or a composite can be deployed to the domain as an implementation. For more detail on the  
1208 deployment of composites, see the section dealing with the [SCA Domain](#).

1209

1210 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite**  
1211 element. The following snippet shows the schema for the composite element.

1212

```
1213 <?xml version="1.0" encoding="ASCII"?>  
1214 <!-- Composite schema snippet -->  
1215 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
1216     targetNamespace="xs:anyURI"  
1217     name="xs:NCName" local="xs:boolean"?  
1218     autowire="xs:boolean"? constrainingType="QName"?  
1219     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1220  
1221     <include ... />*  
1222  
1223     <service ... />*  
1224     <reference ... />*  
1225     <property ... />*  
1226  
1227     <component ... />*  
1228  
1229     <wire ... />*  
1230  
1231 </composite>
```

1232

1233 The **composite** element has the following **attributes**:

- 1234 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is  
1235 an XML QName, in the namespace identified by the targetNamespace attribute. A  
1236 composite name must be unique within the namespace of the composite. [\[ASM60001\]](#)
- 1237 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the  
1238 composite is declared
- 1239 • **local : boolean (0..1)** – whether all the components within the composite all run in the  
1240 same operating system process. @local="true" for a composite means that all the

- 1241 components within the composite MUST run in the same operating system process.  
 1242 [ASM60002] local="false", which is the default, means that different components within  
 1243 the composite can run in different operating system processes and they can even run on  
 1244 different nodes on a network.
- 1245 • **autowire : boolean (0..1)** – whether contained component references should be  
 1246 autowired, as described in [the Autowire section](#). Default is false.
  - 1247 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
 1248 the set of services, references and properties of the composite, plus related intents, is  
 1249 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
 1250 for more details.
  - 1251 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)  
 1252 [specification \[10\]](#) for a description of this attribute.
  - 1253 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
 1254 [\[10\]](#) for a description of this attribute.

1255

1256 The **composite** element has the following **child elements**:

- 1257 • **service : CompositeService (0..n)** – see composite service section.
- 1258 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1259 • **property : CompositeProperty (0..n)** – see composite property section.
- 1260 • **component : Component (0..n)** – see component section.
- 1261 • **wire : Wire (0..n)** – see composite wire section.
- 1262 • **include : Include (0..n)** – see composite include section

1263

1264 Components contain configured implementations which hold the business logic of the composite.  
 1265 The components offer services and require references to other services. **Composite services**  
 1266 define the public services provided by the composite, which can be accessed from outside the  
 1267 composite. **Composite references** represent dependencies which the composite has on services  
 1268 provided elsewhere, outside the composite. Wires describe the connections between component  
 1269 services and component references within the composite. Included composites contribute the  
 1270 elements they contain to the using composite.

1271 Composite services involve the **promotion** of one service of one of the components within the  
 1272 composite, which means that the composite service is actually provided by one of the components  
 1273 within the composite. Composite references involve the **promotion** of one or more references of  
 1274 one or more components. Multiple component references can be promoted to the same composite  
 1275 reference, as long as all the component references are compatible with one another. Where  
 1276 multiple component references are promoted to the same composite reference, then they all share  
 1277 the same configuration, including the same target service(s).

1278 Composite services and composite references can use the configuration of their promoted services  
 1279 and references respectively (such as Bindings and Policy Sets). Alternatively composite services  
 1280 and composite references can override some or all of the configuration of the promoted services  
 1281 and references, through the configuration of bindings and other aspects of the composite service  
 1282 or reference.

1283 Component services and component references can be promoted to composite services and  
 1284 references and also be wired internally within the composite at the same time. For a reference,  
 1285 this only makes sense if the reference supports a multiplicity greater than 1.

1286

## 1287 6.1 Service

1288 The **services of a composite** are defined by promoting services defined by components  
1289 contained in the composite. A component service is promoted by means of a composite **service**  
1290 **element**.

1291 A composite service is represented by a **service element** which is a child of the composite  
1292 element. There can be **zero or more** service elements in a composite. The following snippet  
1293 shows the pseudo-schema for a service child element:

```
1294  
1295 <?xml version="1.0" encoding="ASCII"?>  
1296 <!-- Composite Service schema snippet -->  
1297 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1298   ...  
1299   <service name="xs:NCName" promote="xs:anyURI"  
1300     requires="list of xs:QName"? policySets="list of xs:QName"?>*<br>  
1301     <interface ... />?<br>  
1302     <operation name="xs:NCName" requires="list of xs:QName"?<br>  
1303       policySets="list of xs:QName"?/>*<br>  
1304     <binding ... />*<br>  
1305     <callback?>?<br>  
1306       <binding ... />+<br>  
1307     </callback><br>  
1308   </service><br>  
1309   ...<br>  
1310 </composite><br>  
1311
```

1312 The **composite service** element has the following **attributes**:

- 1313 • **name : NCName (1..1)** – the name of the service. The name of a composite <service/>  
1314 element MUST be unique across all the composite services in the composite. [ASM60003]  
1315 The name of the composite service can be different from the name of the promoted  
1316 component service.
- 1317 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form  
1318 <component-name>/<service-name>. The service name is optional if the target  
1319 component only has one service. The same component service can be promoted by more  
1320 than one composite service. A composite <service/> element's promote attribute MUST  
1321 identify one of the component services within that composite. [ASM60004]
- 1322 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1323 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1324 further qualify the required intents defined by the promoted component service.
- 1325 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1326 [\[10\]](#) for a description of this attribute.

1327  
1328 The **composite service** element has the following **child elements**, whatever is not specified is  
1329 defaulted from the promoted component service.

- 1330 • **interface : Interface (0..1)** - If a composite service **interface** is specified it must be the  
1331 same or a compatible subset of the interface provided by the promoted component  
1332 service, i.e. provide a subset of the operations defined by the component service.  
1333 [ASM60005] The interface is described by **zero or one interface element** which is a  
1334 child element of the service element. For details on the interface element see [the Interface](#)  
1335 [section](#).

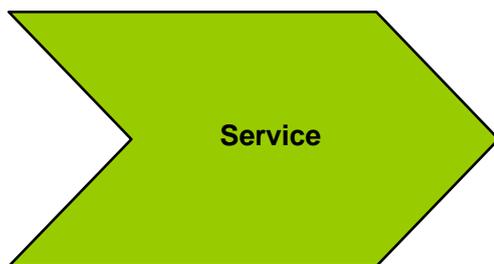
**Deleted:** If a composite service **interface** is specified it must be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service.

- 1336
- 1337
- 1338
- **operation: Operation (0..n)** - Zero or more operation elements. These elements are used to describe characteristics of individual operations within the interface. For a detailed description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
  - **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the promoted component service from the composite service perspective. The bindings defined on the component service are still in effect for local wires within the composite that target the component service. A service element has zero or more **binding elements** as children. Details of the binding element are described in the [Bindings section](#). For more details on wiring see [the Wiring section](#).
  - **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
- 1345
- 1346
- 1347
- 1348
- 1349
- 1350

### 1351 6.1.1 Service Examples

1352

1353 The following figure shows the service symbol that used to represent a service in an assembly  
1354 diagram:

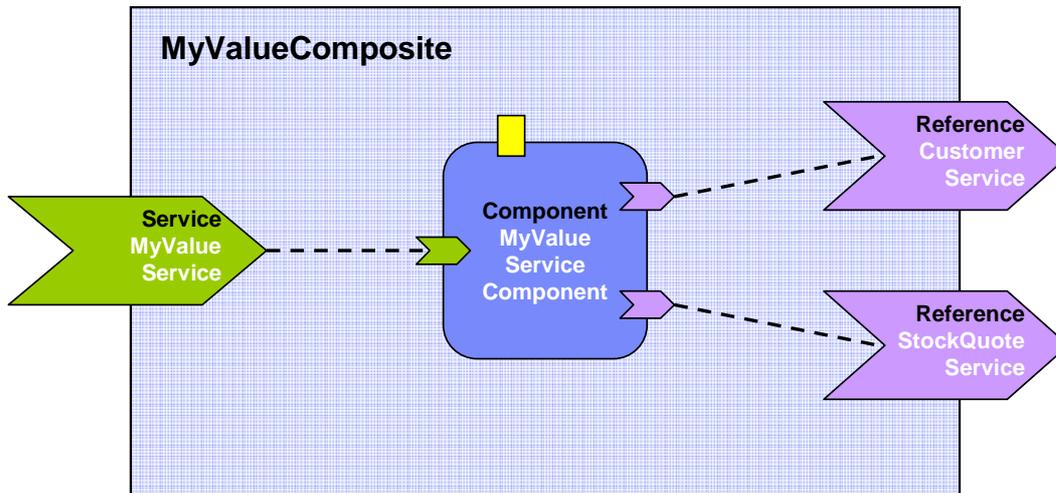


1355

1356 *Figure 7: Service symbol*

1357

1358 The following figure shows the assembly diagram for the MyValueComposite containing the service  
1359 MyValueService.



1360  
1361 *Figure 8: MyValueComposite showing Service*

1362  
1363 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1364 containing the service element for the MyValueService, which is a promote of the service offered  
1365 by the MyValueServiceComponent. The name of the promoted service is omitted since  
1366 MyValueServiceComponent offers only one service. The composite service MyValueService is  
1367 bound using a Web service binding.

1368  
1369 `<?xml version="1.0" encoding="ASCII"?>`  
1370 `<!-- MyValueComposite_4 example -->`  
1371 `<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"`  
1372 `targetNamespace="http://foo.com"`  
1373 `name="MyValueComposite" >`  
1374  
1375 `...`  
1376  
1377 `<service name="MyValueService" promote="MyValueServiceComponent">`  
1378 `<interface.java interface="services.myvalue.MyValueService"/>`  
1379 `<binding.ws port="http://www.myvalue.org/MyValueService#`  
1380 `wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>`  
1381 `</service>`  
1382  
1383 `<component name="MyValueServiceComponent">`  
1384 `<implementation.java`  
1385 `class="services.myvalue.MyValueServiceImpl"/>`  
1386 `<property name="currency">EURO</property>`  
1387 `<service name="MyValueService"/>`  
1388 `<reference name="customerService"/>`  
1389 `<reference name="StockQuoteService"/>`  
1390 `</component>`

1391  
1392       ...  
1393  
1394       </composite>

1395

## 1396 6.2 Reference

1397       The **references of a composite** are defined by **promoting** references defined by components  
1398       contained in the composite. Each promoted reference indicates that the component reference  
1399       needs to be resolved by services outside the composite. A component reference is promoted using  
1400       a composite **reference element**.

1401       A composite reference is represented by a **reference element** which is a child of a composite  
1402       element. There can be **zero or more** reference elements in a composite. The following snippet  
1403       shows the composite schema with the schema for a **reference** element.

1404

```
1405 <?xml version="1.0" encoding="ASCII"?>
1406 <!-- Composite Reference schema snippet -->
1407 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1408   ...
1409   <reference name="xs:NCName" target="list of xs:anyURI"?
1410     promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
1411     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
1412     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1413     <interface ... />?
1414     <operation name="xs:NCName" requires="list of xs:QName"?
1415       policySets="list of xs:QName"?/>*
1416     <binding ... />*
1417     <callback?
1418       <binding ... />+
1419     </callback>
1420   </reference>
1421   ...
1422 </composite>
```

1423

1424

1425       The **composite reference** element has the following **attributes**:

- 1426       • **name : NCName (1..1)** – the name of the reference. The name of a composite  
1427        <reference/> element MUST be unique across all the composite references in the  
1428        composite. [ASM60006] The name of the composite reference can be different then the  
1429        name of the promoted component reference.
- 1430       • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
1431        value is a list of values of the form <component-name>/<reference-name> separated by  
1432        spaces. The specification of the reference name is optional if the component has only one  
1433        reference. Each of the URIs declared by a composite reference's @promote attribute MUST  
1434        identify a component reference within the composite. [ASM60007]

1435       The same component reference can be promoted more than once, using different  
1436       composite references, but only if the multiplicity defined on the component reference is  
1437       0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1438       Where a composite reference promotes two or more component references:

- 1439       • the interfaces of the component references promoted by a composite reference  
1440        MUST be the same, or if the composite reference itself declares an interface then  
1441        all the component reference interfaces must be compatible with the composite

1442 reference interface. Compatible means that the component reference interface is  
1443 the same or is a strict subset of the composite reference interface. [ASM60008]

1444 • the intents declared on a composite reference and on the component references  
1445 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents  
1446 which apply to the composite reference in this case are the union of the required  
1447 intents specified for each of the promoted component references plus any intents  
1448 declared on the composite reference itself. If any intents in the set which apply to  
1449 a composite reference are mutually exclusive then the SCA runtime MUST report  
1450 an error. [ASM60010]

1451 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
1452 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
1453 further qualify the required intents defined for the promoted component reference.

1454 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1455 [\[10\]](#) for a description of this attribute.

1456 • **multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can  
1457 connect the reference to target services. The multiplicity can have the following values

- 1458 ○ 0..1 – zero or one wire can have the reference as a source
- 1459 ○ 1..1 – one wire can have the reference as a source
- 1460 ○ 0..n - zero or more wires can have the reference as a source
- 1461 ○ 1..n – one or more wires can have the reference as a source

1462 The value specified for the **multiplicity** attribute of a composite reference MUST be  
1463 compatible with the multiplicity specified on each of the promoted component references,  
1464 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used  
1465 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be  
1466 used where the promoted component reference has multiplicity 0..n or 1..n and  
1467 multiplicity 1..n can be used where the promoted component reference has multiplicity  
1468 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to  
1469 promote a component reference of multiplicity 0..1 or 1..1 respectively. [ASM60011]

1470 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
1471 multiplicity setting. Each value wires the reference to a service in a composite that uses  
1472 the composite containing the reference as an implementation for one of its components. For  
1473 more details on wiring see [the section on Wires](#).

1474 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
1475 the implementation wires this reference dynamically. If set to "true" it indicates that the  
1476 target of the reference is set at runtime by the implementation code (eg by the code  
1477 obtaining an endpoint reference by some means and setting this as the target of the  
1478 reference through the use of programming interfaces defined by the relevant Client and  
1479 Implementation specification). If "true" is set, then the reference should not be wired  
1480 statically within a using composite, but left unwired.

1481

1482 The **composite reference** element has the following **child elements**, whatever is not specified is  
1483 defaulted from the promoted component reference(s).

1484 • **interface : Interface (0..1) - zero or one interface element** which declares an  
1485 interface for the composite reference. If a composite reference has an **interface** specified,  
1486 it MUST provide an interface which is the same or which is a compatible superset of the  
1487 interface(s) declared by the promoted component reference(s), i.e. provide a superset of  
1488 the operations in the interface defined by the component for the reference. [ASM60012] If  
1489 no interface is declared on a composite reference, the interface from one of its promoted  
1490 component references is used, which MUST be the same as or a compatible superset of  
1491 the interface(s) declared by the promoted component reference(s).  
1492 [ASM60013] For details on the interface element see [the Interface section](#).

**Deleted:** If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST raise an error.

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

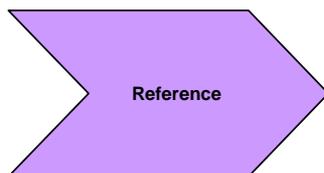
**Deleted:** If a composite reference has an **interface** specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s), i.e. provide a superset of the operations in the interface defined by the component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

- 1493
- 1494
- 1495
- **operation: Operation (0..n)** - Zero or more operation elements. These elements are used to describe characteristics of individual operations within the interface. For a detailed description of the operation element, see [the Policy Framework specification](#) [SCA Policy].
  - **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as children. If one or more **bindings** are specified they **override** any and all of the bindings defined for the promoted component reference from the composite reference perspective. The bindings defined on the component reference are still in effect for local wires within the composite that have the component reference as their source. Details of the binding element are described in the [Bindings section](#). For more details on wiring see [the section on Wires](#).
- 1496
- 1497
- 1498
- 1499
- 1500
- 1501
- 1502
- 1503
- 1504
- 1505
- A reference identifies zero or more target services which satisfy the reference. This can be done in a number of ways, which are fully described in section "5.3.1 Specifying the Target Service(s) for a Reference".
- **callback (0..1) / binding : Binding (1..n)** - A **reference** element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.
- 1506
- 1507
- 1508
- 1509
- 1510
- 1511

## 1512 6.2.1 Example Reference

1513

1514 The following figure shows the reference symbol that is used to represent a reference in an  
1515 assembly diagram.



1516

1517 *Figure 9: Reference symbol*

1518

1519 The following figure shows the assembly diagram for the MyValueComposite containing the  
1520 reference CustomerService and the reference StockQuoteService.

1521

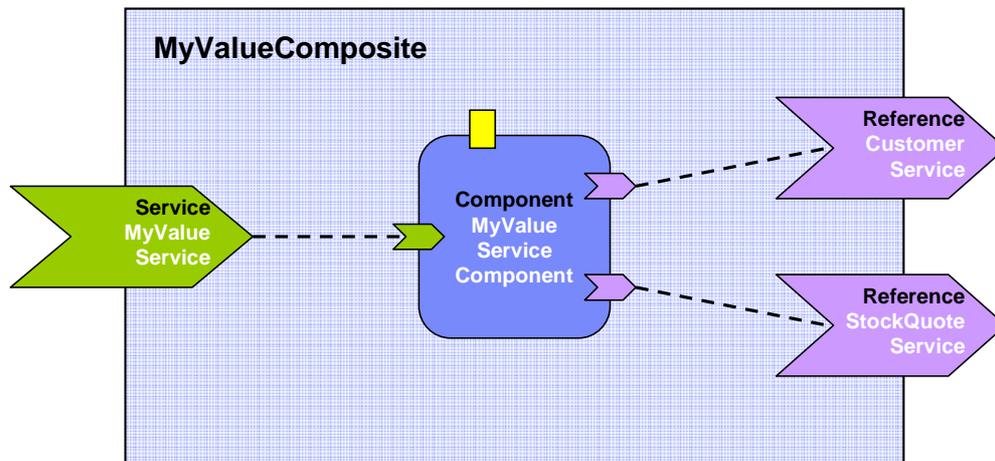


Figure 10: MyValueComposite showing References

1522

1523

1524

1525 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
 1526 containing the reference elements for the CustomerService and the StockQuoteService. The  
 1527 reference CustomerService is bound using the SCA binding. The reference StockQuoteService is  
 1528 bound using the Web service binding. The endpoint addresses of the bindings can be specified, for  
 1529 example using the binding *uri* attribute (for details see the [Bindings](#) section), or overridden in an  
 1530 enclosing composite. Although in this case the reference StockQuoteService is bound to a Web  
 1531 service, its interface is defined by a Java interface, which was created from the WSDL portType of  
 1532 the target web service.

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

1549

1550

1551

1552

1553

1554

```

<?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 -->

```

```

1555         <!-- specified by the component reference or by the underlying
1556 -->
1557         <!-- implementation
1558 -->
1559         <binding.sca/>
1560     </reference>
1561
1562     <reference name="StockQuoteService"
1563         promote="MyValueServiceComponent/StockQuoteService">
1564         <interface.java
1565 interface="services.stockquote.StockQuoteService"/>
1566         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1567 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1568     </reference>
1569
1570     ...
1571
1572
1573 </composite>
1574

```

### 6.3 Property

1575 **Properties** allow for the configuration of an implementation with externally set data values. A  
1576 composite can declare zero or more properties. Each property has a type, which may be either  
1577 simple or complex. An implementation can also define a default value for a property. Properties  
1578 can be configured with values in the components that use the implementation.

1580 The declaration of a property in a composite follows the form described in the following schema  
1581 snippet:

```

1582
1583 <?xml version="1.0" encoding="ASCII"?>
1584 <!-- Composite Property schema snippet -->
1585 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
1586     ...
1587     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1588         requires="list of xs:QName"?
1589         policySets="list of xs:QName"?
1590         many="xs:boolean"? mustSupply="xs:boolean"?>*
1591         default-property-value?
1592     </property>
1593     ...
1594 </composite>
1595

```

1596 The **composite property** element has the following **attributes**:

- 1597 ▪ **name : NCName (1..1)** - the name of the property. **The name attribute of a composite**  
1598 **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.

- 1599       ▪ one of **(1..1)**:
  - 1600           ○ **type : QName** – the type of the property - the qualified name of an XML schema type
  - 1601
  - 1602           ○ **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
  - 1603
  - 1604       ▪ **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.
  - 1605
  - 1606
  - 1607       ▪ **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.
  - 1608
  - 1609
  - 1610
  - 1611
  - 1612
  - 1613       ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
  - 1614
  - 1615       ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
  - 1616

1617

1618       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](#).

1621

1622       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.

1625       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](#).

### 1627   6.3.1 Property Examples

1628

1629       For the following example of Property declaration and value setting, the following complex type is used as an example:

```

1631 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1632             targetNamespace="http://foo.com/"
1633             xmlns:tns="http://foo.com/">
1634   <!-- ComplexProperty schema -->
1635   <xsd:element name="fooElement" type="MyComplexType"/>
1636   <xsd:complexType name="MyComplexType">
1637     <xsd:sequence>
1638       <xsd:element name="a" type="xsd:string"/>
1639       <xsd:element name="b" type="anyURI"/>
1640     </xsd:sequence>
1641     <attribute name="attr" type="xsd:string" use="optional"/>
1642   </xsd:complexType>
1643 </xsd:schema>
1644
```

1645       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:

```

1648 <?xml version="1.0" encoding="ASCII"?>
1649 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1650           xmlns:foo="http://foo.com"
1651           targetNamespace="http://foo.com"
1652           name="AccountServices">
1653 <!-- AccountServices Example1 -->
1654
1655     ...
1656
1657     <property name="complexFoo" type="foo:MyComplexType">
1658       <value>
1659         <foo:a>AValue</foo:a>
1660         <foo:b>InterestingURI</foo:b>
1661       </value>
1662     </property>
1663
1664     <component name="AccountServiceComponent">
1665       <implementation.java class="foo.AccountServiceImpl"/>
1666       <property name="complexBar" source="$complexFoo"/>
1667       <reference name="accountDataService"
1668         target="AccountDataServiceComponent"/>
1669       <reference name="stockQuoteService" target="StockQuoteService"/>
1670     </component>
1671
1672     ...
1673
1674 </composite>
1675

```

1676 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the
1677 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the
1678 composite and it references the example XSD, where MyComplexType is defined. The declaration
1679 of complexFoo contains a default value. This is declared as the content of the property element.
1680 In this example, the default value consists of the element **value** which is required to be of type
1681 foo:MyComplexType and its two child elements <foo:a> and <foo:b>, following the definition of
1682 MyComplexType.

1683 In the component **AccountServiceComponent**, the component sets the value of the property
1684 **complexBar**, declared by the implementation configured by the component. In this case, the
1685 type of complexBar is foo:MyComplexType. The example shows that the value of the complexBar
1686 property is set from the value of the complexFoo property – the **source** attribute of the property
1687 element for complexBar declares that the value of the property is set from the value of a property
1688 of the containing composite. The value of the source attribute is **\$complexFoo**, where
1689 complexFoo is the name of a property of the composite. This value implies that the whole of the
1690 value of the source property is used to set the value of the component property.

1691 The following example illustrates the setting of the value of a property of a simple type (a string)
1692 from **part** of the value of a property of the containing composite which has a complex type:

```

1693 <?xml version="1.0" encoding="ASCII"?>
1694 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1695           xmlns:foo="http://foo.com"
1696           targetNamespace="http://foo.com"
1697           name="AccountServices">
1698 <!-- AccountServices Example2 -->
1699
1700     ...
1701
1702     <property name="complexFoo" type="foo:MyComplexType">
1703       <value>
1704         <foo:a>AValue</foo:a>

```

```

1705         <foo:b>InterestingURI</foo:b>
1706     </value>
1707 </property>
1708
1709     <component name="AccountServiceComponent">
1710         <implementation.java class="foo.AccountServiceImpl"/>
1711         <property name="currency" source="$complexFoo/a"/>
1712         <reference name="accountDataService"
1713             target="AccountDataServiceComponent"/>
1714         <reference name="stockQuoteService" target="StockQuoteService"/>
1715     </component>
1716
1717     ...
1718
1719 </composite>
1720

```

1721 In this example, the component **AccountServiceComponent** sets the value of a property called  
1722 **currency**, which is of type string. The value is set from a property of the composite  
1723 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression  
1724 that selects the property name **complexFoo** and then selects the value of the **a** subelement of  
1725 the value of complexFoo. The "a" subelement is a string, matching the type of the currency  
1726 property.

1727 Further examples of declaring properties and setting property values in a component follow:

1728 Declaration of a property with a simple type and a default value:

```

1729 <property name="SimpleTypeProperty" type="xsd:string">
1730 MyValue
1731 </property>

```

1732

1733 Declaration of a property with a complex type and a default value:

```

1734 <property name="complexFoo" type="foo:MyComplexType">
1735     <value>
1736         <foo:a>AValue</foo:a>
1737         <foo:b>InterestingURI</foo:b>
1738     </value>
1739 </property>
1740

```

1741 Declaration of a property with a global element type:

```

1742 <property name="elementFoo" element="foo:fooElement">
1743     <foo:fooElement>
1744         <foo:a>AValue</foo:a>
1745         <foo:b>InterestingURI</foo:b>
1746     </foo:fooElement>
1747 </property>
1748

```

## 1749 6.4 Wire

1750 **SCA wires** within a composite connect **source component references** to **target component**  
1751 **services**.

1752 One way of defining a wire is by **configuring a reference of a component using its target**  
1753 **attribute**. The reference element is configured with the wire-target-URI of the service(s) that  
1754 resolve the reference. Multiple target services are valid when the reference has a multiplicity of  
1755 0..n or 1..n.

1756 An alternative way of defining a Wire is by means of a **wire element** which is a child of the  
1757 composite element. There can be **zero or more** wire elements in a composite. This alternative  
1758 method for defining wires is useful in circumstances where separation of the wiring from the  
1759 elements the wires connect helps simplify development or operational activities. An example is  
1760 where the components used to build a domain are relatively static but where new or changed  
1761 applications are created regularly from those components, through the creation of new assemblies  
1762 with different wiring. Deploying the wiring separately from the components allows the wiring to  
1763 be created or modified with minimum effort.

1764 Note that a Wire specified via a wire element is equivalent to a wire specified via the target  
1765 attribute of a reference. The rule which forbids mixing of wires specified with the target attribute  
1766 with the specification of endpoints in binding subelements of the reference also applies to wires  
1767 specified via separate wire elements.

1768 The following snippet shows the composite schema with the schema for the reference elements of  
1769 components and composite services and the wire child element:

1770

```
1771 <?xml version="1.0" encoding="ASCII"?>  
1772 <!-- Wires schema snippet -->  
1773 <composite ...>  
1774   ...  
1775   <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/> *  
1776   ...  
1777 </composite>  
1778
```

1779 The **reference element of a component** and the **reference element of a service** has a list of  
1780 one or more of the following **wire-target-URI** values for the target, with multiple values  
1781 separated by a space:

- 1782 • `<component-name>/<service-name>`
  - 1783 ○ where the target is a service of a component. The specification of the service  
1784 name is optional if the target component only has one service with a compatible  
1785 interface

1786

1787 The **wire element** has the following attributes:

- 1788 • **source (1..1)** – names the source component reference. Valid URI schemes are:
  - 1789 ○ `<component-name>/<reference-name>`
    - 1790 ▪ where the source is a component reference. The specification of the  
1791 reference name is optional if the source component only has one reference
- 1792 • **target (1..1)** – names the target component service. Valid URI schemes are
  - 1793 ○ `<component-name>/<service-name>`
    - 1794 ▪ where the target is a service of a component. The specification of the  
1795 service name is optional if the target component only has one service with  
1796 a compatible interface
- 1797 • **replace (0..1)** – a boolean value, with the default of "false". When a wire element has  
1798 `@replace="false"`, the wire is added to the set of wires which apply to the reference  
1799 identified by the `@source` attribute. When a wire element has `@replace="true"`, the wire  
1800 is added to the set of wires which apply to the reference identified by the `@source`  
1801 attribute - but any wires for that reference specified by means of the `@target` attribute of  
1802 the reference are removed from the set of wires which apply to the reference.

1803

1804 In other words, if any `<wire/>` element with `@replace="true"` is used for a particular  
1805 reference, the value of the `@target` attribute on the reference is ignored - and this permits

1806 existing wires on the reference to be overridden by separate configuration, if required,  
1807 where the reference is on a component at the Domain level.

1808 For a composite used as a component implementation, wires can only link sources and targets  
1809 that are contained in the same composite (irrespective of which file or files are used to describe  
1810 the composite). Wiring to entities outside the composite is done through services and references  
1811 of the composite with wiring defined by the next higher composite.

1812 A wire may only connect a source to a target if the target implements an interface that is  
1813 compatible with the interface required by the source. The source and the target are compatible if:

- 1814 1. the source interface and the target interface of a wire MUST either both be remotable or  
1815 else both be local [ASM60015]
- 1816 2. the operations on the target interface of a wire MUST be the same as or be a superset of  
1817 the operations in the interface specified on the source [ASM60016]
- 1818 3. compatibility between the source interface and the target interface for a wire for the  
1819 individual operations is defined as compatibility of the signature, that is operation name,  
1820 input types, and output types MUST be the same. [ASM60017]
- 1821 4. the order of the input and output types for operations in the source interface and the  
1822 target interface of a wire also MUST be the same. [ASM60018]
- 1823 5. the set of Faults and Exceptions expected by each operation in the source interface MUST  
1824 be the same or be a superset of those specified by the target interface. [ASM60019]
- 1825 6. other specified attributes of the source interface and the target interface of a wire MUST  
1826 match, including Scope and Callback interface [ASM60020]

1827 A Wire can connect between different interface languages (eg. Java interfaces and WSDL  
1828 portTypes) in either direction, as long as the operations defined by the two interface types are  
1829 equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and  
1830 faults/exceptions map to each other.

1831 Service clients cannot (portably) ask questions at runtime about additional interfaces that are  
1832 provided by the implementation of the service (e.g. the result of "instance of" in Java is non  
1833 portable). It is valid for an SCA implementation to have proxies for all wires, so that, for example,  
1834 a reference object passed to an implementation may only have the business interface of the  
1835 reference and may not be an instance of the (Java) class which is used to implement the target  
1836 service, even where the interface is local and the target service is running in the same process.

1837 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of  
1838 an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA  
1839 runtime SHOULD issue a warning. [ASM60021]

1840

## 1841 6.4.1 Wire Examples

1842

1843 The following figure shows the assembly diagram for the MyValueComposite2 containing wires  
1844 between service, components and references.

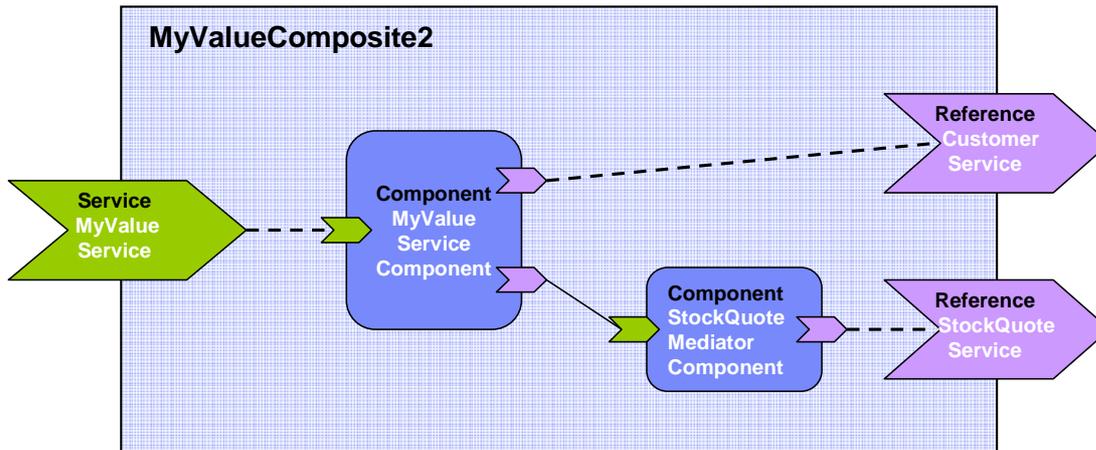


Figure 11: MyValueComposite2 showing Wires

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.

```

<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite Wires examples -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
targetNamespace="http://foo.com"
name="MyValueComposite2" >

    <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>

```

```

1877     <wire source="MyValueServiceComponent/stockQuoteService"
1878           target="StockQuoteMediatorComponent" />
1879
1880     <component name="StockQuoteMediatorComponent">
1881       <implementation.java class="services.myvalue.SQMediatorImpl" />
1882       <property name="currency">EURO</property>
1883       <reference name="stockQuoteService" />
1884     </component>
1885
1886     <reference name="CustomerService"
1887               promote="MyValueServiceComponent/customerService">
1888       <interface.java interface="services.customer.CustomerService" />
1889       <binding.sca/>
1890     </reference>
1891
1892     <reference name="StockQuoteService"
1893               promote="StockQuoteMediatorComponent">
1894       <interface.java
1895         interface="services.stockquote.StockQuoteService" />
1896       <binding.ws port="http://www.stockquote.org/StockQuoteService#
1897         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)" />
1898     </reference>
1899
1900 </composite>
1901

```

## 1902 6.4.2 Autowire

1903 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.  
 1904 Autowire enables component references to be automatically wired to component services which  
 1905 will satisfy those references, without the need to create explicit wires between the references and  
 1906 the services. When the autowire feature is used, a component reference which is not promoted  
 1907 and which is not explicitly wired to a service within a composite is automatically wired to a target  
 1908 service within the same composite. Autowire works by searching within the composite for a  
 1909 service interface which matches the interface of the references.

1910 The autowire feature is not used by default. Autowire is enabled by the setting of an autowire  
 1911 attribute to "true". Autowire is disabled by setting of the autowire attribute to "false". The autowire  
 1912 attribute can be applied to any of the following elements within a composite:

- 1913 • reference
- 1914 • component
- 1915 • composite

1916 Where an element does not have an explicit setting for the autowire attribute, it inherits the  
 1917 setting from its parent element. Thus a reference element inherits the setting from its containing  
 1918 component. A component element inherits the setting from its containing composite. Where  
 1919 there is no setting on any level, autowire="false" is the default.

1920 As an example, if a composite element has autowire="true" set, this means that autowiring is  
 1921 enabled for all component references within that composite. In this example, autowiring can be

1922 turned off for specific components and specific references through setting autowire="false" on the  
1923 components and references concerned.

1924 For each component reference for which autowire is enabled, the the SCA runtime MUST search  
1925 within the composite for target services which are compatible with the reference. [ASM60022]  
1926 "Compatible" here means:

- 1927 • the target service interface MUST be a compatible superset of the reference interface  
1928 when using autowire to wire a reference (as defined in the section on Wires), [ASM60023]
- 1929 • the intents, and policies applied to the service MUST be compatible with those on the  
1930 reference when using autowire to wire a reference – so that wiring the reference to the  
1931 service will not cause an error due to policy mismatch [ASM60024] (see [the Policy](#)  
1932 [Framework specification \[10\]](#) for details)

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

1933 If the search finds **1 or more** valid target service for a particular reference, the action taken  
1934 depends on the multiplicity of the reference:

- 1935 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
1936 reference to one of the set of valid target services chosen from the set in a runtime-  
1937 dependent fashion [ASM60025]
- 1938 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
1939 of the set of valid target services [ASM60026]

1940 If the search finds **no** valid target services for a particular reference, the action taken depends on  
1941 the multiplicity of the reference:

- 1942 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
1943 target service, there is no problem – no services are wired and the SCA runtime MUST  
1944 NOT report an error, [ASM60027]
- 1945 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
1946 target services an error MUST be reported by the SCA runtime since the reference is  
1947 intended to be wired, [ASM60028]

**Deleted:** 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

1948

### 1949 6.4.3 Autowire Examples

1950 This example demonstrates two versions of the same composite – the first version is done using  
1951 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
1952 the end result is the same – the same wires connect the references to the services.

1953 First, here is a diagram for the composite:

**Deleted:** 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

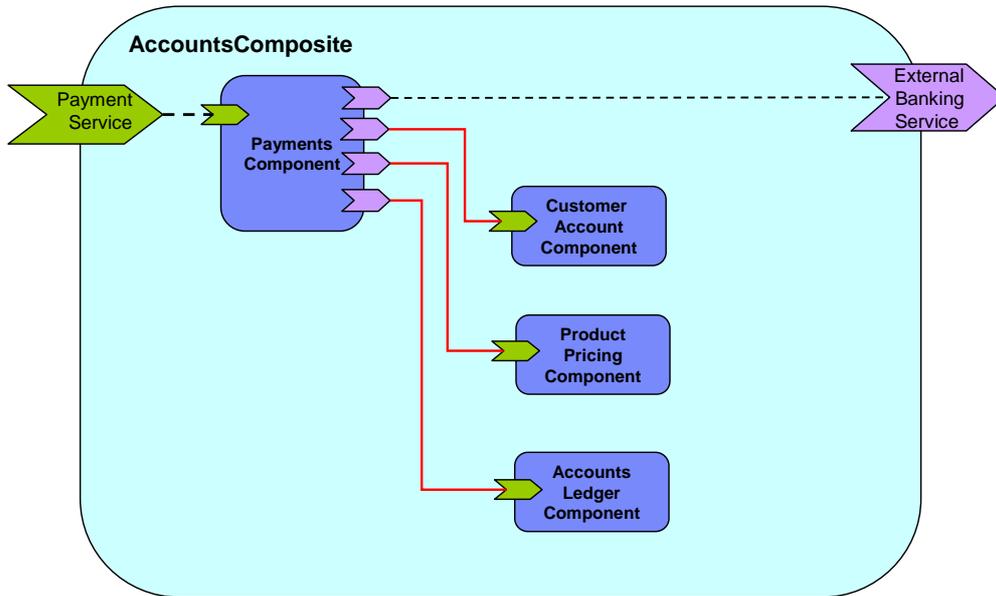


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1954
1955
1956
1957 <?xml version="1.0" encoding="UTF-8"?>
1958 <!-- Autowire Example - No autowire -->
1959 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1960   xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1961   xmlns:foo="http://foo.com"
1962   targetNamespace="http://foo.com"
1963   name="AccountComposite">
1964
1965   <service name="PaymentService" promote="PaymentsComponent" />
1966
1967   <component name="PaymentsComponent">
1968     <implementation.java class="com.foo.accounts.Payments" />
1969     <service name="PaymentService" />
1970     <reference name="CustomerAccountService"
1971       target="CustomerAccountComponent" />
1972     <reference name="ProductPricingService"
1973       target="ProductPricingComponent" />
1974     <reference name="AccountsLedgerService"
1975       target="AccountsLedgerComponent" />
1976     <reference name="ExternalBankingService" />
1977   </component>
1978
1979   <component name="CustomerAccountComponent">
1980     <implementation.java class="com.foo.accounts.CustomerAccount" />
1981   </component>
1982
1983   <component name="ProductPricingComponent">
1984     <implementation.java class="com.foo.accounts.ProductPricing" />
1985   </component>
1986
1987   <component name="AccountsLedgerComponent">

```

```

1988     <implementation.composite name="foo:AccountsLedgerComposite"/>
1989 </component>
1990
1991     <reference name="ExternalBankingService"
1992         promote="PaymentsComponent/ExternalBankingService"/>
1993
1994 </composite>
1995

```

1996 Secondly, the composite using autowire:

```

1997 <?xml version="1.0" encoding="UTF-8"?>
1998 <!-- Autowire Example - With autowire -->
1999 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2000     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2001     xmlns:foo="http://foo.com"
2002     targetNamespace="http://foo.com"
2003     name="AccountComposite">
2004
2005     <service name="PaymentService" promote="PaymentsComponent">
2006         <interface.java class="com.foo.PaymentServiceInterface"/>
2007     </service>
2008
2009     <component name="PaymentsComponent" autowire="true">
2010         <implementation.java class="com.foo.accounts.Payments"/>
2011         <service name="PaymentService"/>
2012         <reference name="CustomerAccountService"/>
2013         <reference name="ProductPricingService"/>
2014         <reference name="AccountsLedgerService"/>
2015         <reference name="ExternalBankingService"/>
2016     </component>
2017
2018     <component name="CustomerAccountComponent">
2019         <implementation.java class="com.foo.accounts.CustomerAccount"/>
2020     </component>
2021
2022     <component name="ProductPricingComponent">
2023         <implementation.java class="com.foo.accounts.ProductPricing"/>
2024     </component>
2025
2026     <component name="AccountsLedgerComponent">
2027         <implementation.composite name="foo:AccountsLedgerComposite"/>
2028     </component>
2029
2030     <reference name="ExternalBankingService"
2031         promote="PaymentsComponent/ExternalBankingService"/>
2032
2033 </composite>

```

2034 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires  
2035 for any of its references – the wires are created automatically through autowire.

2036 **Note:** In the second example, it would be possible to omit all of the service and reference  
2037 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the  
2038 component service and references still exist, since they are provided by the implementation used  
2039 by the component.

2040

## 2041 6.5 Using Composites as Component Implementations

2042 Composites may form **component implementations** in higher-level composites – in other words  
2043 the higher-level composites can have components which are implemented by composites.

2044 When a composite is used as a component implementation, it defines a boundary of visibility.  
2045 Components within the composite cannot be referenced directly by the using component. The  
2046 using component can only connect wires to the services and references of the used composite and  
2047 set values for any properties of the composite. The internal construction of the composite is  
2048 invisible to the using component. The boundary of visibility, sometimes called encapsulation, can  
2049 be enforced when assembling components and composites, but such encapsulation structures  
2050 might not be enforceable in a particular implementation language.

2051 A composite used as a component implementation must also honor a completeness contract. The  
2052 services, references and properties of the composite form a contract (represented by the  
2053 component type of the composite) which is relied upon by the using component. The concept of  
2054 completeness of the composite implies that, once all <include/> element processing is performed  
2055 on the composite:

- 2056 1. For a composite used as a component implementation, each composite service offered by  
2057 the composite MUST promote a component service of a component that is within the  
2058 composite. [ASM60032]
- 2059 2. For a composite used as a component implementation, every component reference of  
2060 components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or  
2061 promoted (according to the various rules for specifying target services for a component  
2062 reference described in section 5.3.1). [ASM60033]
- 2063 3. For a composite used as a component implementation, all properties of components within  
2064 the composite, where the underlying component implementation specifies  
2065 "mustSupply=true" for the property, MUST either specify a value for the property or  
2066 source the value from a composite property. [ASM60034]

2067 The component type of a composite is defined by the set of composite service elements,  
2068 composite reference elements and composite property elements that are the children of the  
2069 composite element.

2070 Composites are used as component implementations through the use of the  
2071 **implementation.composite** element as a child element of the component. The schema snippet  
2072 for the implementation.composite element is:

```
2073  
2074 <!-- implementation.composite pseudo-schema -->  
2075 <implementation.composite name="xs:QName" requires="list of xs:QName"?  
2076 policySets="list of xs:QName"?>
```

2077

2078

2079 The implementation.composite element has the following attributes:

- 2080 • **name (1..1)** – the name of the composite used as an implementation. The @name  
2081 attribute of an <implementation.composite/> element MUST contain the QName of a  
2082 composite in the SCA Domain. [ASM60030]
- 2083 • **requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework](#)  
2084 [specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or  
2085 further qualify the required intents defined for the promoted component reference.
- 2086 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2087 [\[10\]](#) for a description of this attribute.

2088

## 2089 6.5.1 Example of Composite used as a Component Implementation

2090

2091 The following is an example of a composite which contains two components, each of which is  
2092 implemented by a composite:  
2093

```
2094 <?xml version="1.0" encoding="UTF-8"?>
2095 <!-- CompositeComponent example -->
2096 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
2097           xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200712
2098           file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
2099           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2100           targetNamespace="http://foo.com"
2101           xmlns:foo="http://foo.com"
2102           name="AccountComposite">
2103
2104     <service name="AccountService" promote="AccountServiceComponent">
2105       <interface.java interface="services.account.AccountService"/>
2106       <binding.ws port="AccountService#
2107         wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
2108     </service>
2109
2110     <reference name="stockQuoteService"
2111       promote="AccountServiceComponent/StockQuoteService">
2112       <interface.java
2113         interface="services.stockquote.StockQuoteService"/>
2114       <binding.ws
2115         port="http://www.quickstockquote.com/StockQuoteService#
2116         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2117     </reference>
2118
2119     <property name="currency" type="xsd:string">EURO</property>
2120
2121     <component name="AccountServiceComponent">
2122       <implementation.composite name="foo:AccountServiceComposite1"/>
2123
2124       <reference name="AccountDataService" target="AccountDataService"/>
2125       <reference name="StockQuoteService"/>
2126
2127       <property name="currency" source="$currency"/>
2128     </component>
2129
2130     <component name="AccountDataService">
2131       <implementation.composite name="foo:AccountDataServiceComposite"/>
2132
2133       <property name="currency" source="$currency"/>
2134     </component>
2135
2136 </composite>
2137
```

## 2138 6.6 Using Composites through Inclusion

2139 In order to assist team development, composites may be developed in the form of multiple  
2140 physical artifacts that are merged into a single logical unit.

2141 A composite may include another composite by using the **include** element. This provides a  
2142 recursive inclusion capability. The semantics of included composites are that the element content

2143 children of the included composite are inlined, with certain modification, into the using composite.  
2144 This is done recursively till the resulting composite does not contain an **include** element. The  
2145 outer included composite element itself is discarded in this process – only its contents are included  
2146 as described below:

- 2147 1. All the element content children of the included composite are inlined in the including  
2148 composite.
- 2149 2. The attributes **targetNamespace**, **name**, **constrainingType**, and **local** of the included  
2150 composites are discarded.
- 2151 3. All the namespace declaration on the included composite element are added to the inlined  
2152 element content children unless the namespace binding is overridden by the element  
2153 content children.
- 2154 4. The attribute **autowire**, if specified on the included composite, is included on all inlined  
2155 component element children unless the component child already specifies that attribute.
- 2156 5. The attribute values of **requires** and **policySet**, if specified on the included composite,  
2157 are merged with corresponding attribute on the inlined component, service and reference  
2158 children elements. Merge in this context means a set union.
- 2159 6. Extension attributes ,if present on the included composite, must follow the rules defined  
2160 for that extension. Authors of attribute extensions on the composite element must define  
2161 rules for inclusion.

2162 If the included composite has the value *true* for the attribute **local** then the including composite  
2163 must have the same value for the **local** attribute, else it is considered an error.

2164 The composite file used for inclusion can have any contents, but its document root element must  
2165 be **composite**. The composite element may contain any of the elements which are valid as child  
2166 elements of a composite element, namely components, services, references, wires and includes.  
2167 There is no need for the content of an included composite to be complete, so that artifacts defined  
2168 within the using composite or in another associated included composite file may be referenced. For  
2169 example, it is permissible to have two components in one composite file while a wire specifying  
2170 one component as the source and the other as the target can be defined in a second included  
2171 composite file.

2172 **The SCA runtime MUST report an error if the composite resulting from the inclusion of one**  
2173 **composite into another is invalid, [ASM60031]** For example, it is an error if there are duplicated  
2174 elements in the using composite (eg. two services with the same uri contributed by different  
2175 included composites). It is not considered an error if the (using) composite resulting from the  
2176 inclusion is incomplete (eg. wires with non-existent source or target). Such incomplete resulting  
2177 composites are permitted to allow recursive composition.

**Deleted:** The SCA runtime MUST raise an error if the composite resulting from the inclusion of one composite into another is invalid.

2178 The following snippet shows the pseudo-schema for the include element.

2179

```
2180 <?xml version="1.0" encoding="UTF-8"?>  
2181 <!-- Include snippet -->  
2182 <composite ...>  
2183   ...  
2184   <include name="xs:QName" /> *  
2185   ...  
2186 </composite>  
2187
```

2188 The include element has the following **attribute**:

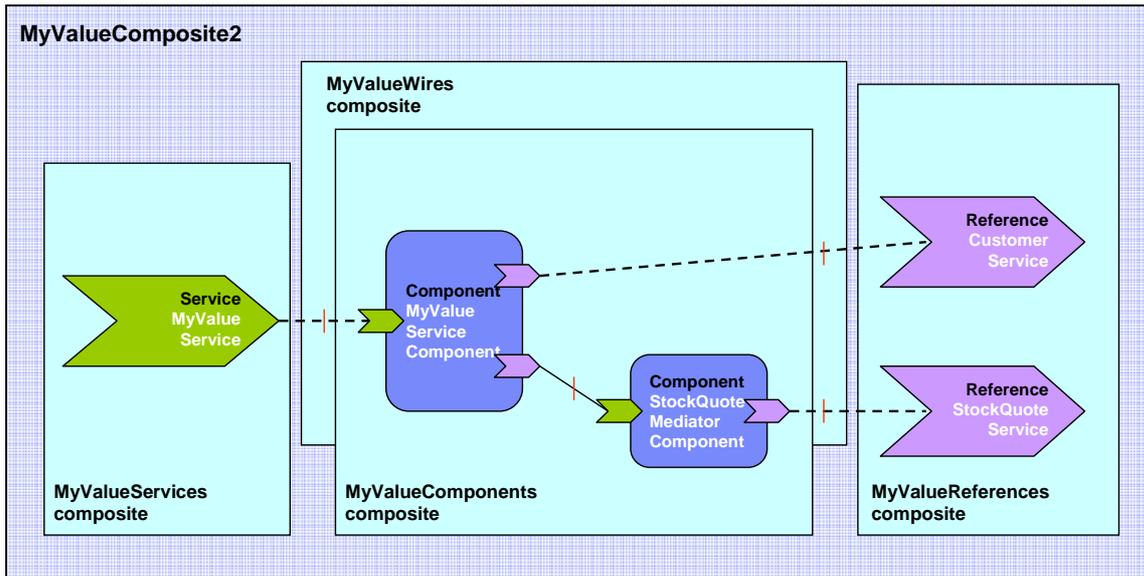
- 2189 • **name (required)** – the name of the composite that is included.

2190

## 2191 6.6.1 Included Composite Examples

2192

2193 The following figure shows the assembly diagram for the MyValueComposite2 containing four  
 2194 included composites. The **MyValueServices composite** contains the MyValueService service. The  
 2195 **MyValueComponents composite** contains the MyValueServiceComponent and the  
 2196 StockQuoteMediatorComponent as well as the wire between them. The **MyValueReferences**  
 2197 **composite** contains the CustomerService and StockQuoteService references. The **MyValueWires**  
 2198 **composite** contains the wires that connect the MyValueService service to the  
 2199 MyValueServiceComponent, that connect the customerService reference of the  
 2200 MyValueServiceComponent to the CustomerService reference, and that connect the  
 2201 stockQuoteService reference of the StockQuoteMediatorComponent to the StockQuoteService  
 2202 reference. Note that this is just one possible way of building the MyValueComposite2 from a set of  
 2203 included composites.



2204  
 2205  
 2206 *Figure 13 MyValueComposite2 built from 4 included composites*  
 2207

2208 The following snippet shows the contents of the MyValueComposite2.composite file for the  
 2209 MyValueComposite2 built using included composites. In this sample it only provides the name of  
 2210 the composite. The composite file itself could be used in a scenario using included composites to  
 2211 define components, services, references and wires.

```
2212
2213 <?xml version="1.0" encoding="ASCII"?>
2214 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2215           targetNamespace="http://foo.com"
2216           xmlns:foo="http://foo.com"
2217           name="MyValueComposite2" >
2218
2219     <include name="foo:MyValueServices" />
2220     <include name="foo:MyValueComponents" />
2221     <include name="foo:MyValueReferences" />
2222     <include name="foo:MyValueWires" />
2223
2224 </composite>
2225
```

2226 The following snippet shows the content of the MyValueServices.composite file.

2227

```
2228 <?xml version="1.0" encoding="ASCII"?>
2229 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2230           targetNamespace="http://foo.com"
2231           xmlns:foo="http://foo.com"
2232           name="MyValueServices" >
2233
2234     <service name="MyValueService" promote="MyValueServiceComponent">
2235       <interface.java interface="services.myvalue.MyValueService"/>
2236       <binding.ws port="http://www.myvalue.org/MyValueService#
2237                 wsdl.endpoint(MyValueService/MyValueServicesSOAP)"/>
2238     </service>
2239
2240 </composite>
2241
```

2242 The following snippet shows the content of the MyValueComponents.composite file.

2243

```
2244 <?xml version="1.0" encoding="ASCII"?>
2245 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2246           targetNamespace="http://foo.com"
2247           xmlns:foo="http://foo.com"
2248           name="MyValueComponents" >
2249
2250   <component name="MyValueServiceComponent">
2251     <implementation.java
2252       class="services.myvalue.MyValueServiceImpl"/>
2253     <property name="currency">EURO</property>
2254   </component>
2255
2256   <component name="StockQuoteMediatorComponent">
2257     <implementation.java class="services.myvalue.SQMediatorImpl"/>
2258     <property name="currency">EURO</property>
2259   </component>
2260
2261 </composite>
2262
```

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

2264

```
2265 <?xml version="1.0" encoding="ASCII"?>
2266 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2267           targetNamespace="http://foo.com"
2268           xmlns:foo="http://foo.com"
2269           name="MyValueReferences" >
2270
2271   <reference name="CustomerService"
2272     promote="MyValueServiceComponent/CustomerService">
2273     <interface.java interface="services.customer.CustomerService"/>
2274     <binding.sca/>
2275   </reference>
2276
2277   <reference name="StockQuoteService"
2278     promote="StockQuoteMediatorComponent">
2279     <interface.java
```

```

2280         interface="services.stockquote.StockQuoteService" />
2281     <binding.ws port="http://www.stockquote.org/StockQuoteService#
2282         wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP) " />
2283     </reference>
2284 </composite>
2285
2286

```

2287 The following snippet shows the content of the MyValueWires.composite file.

```

2288
2289 <?xml version="1.0" encoding="ASCII" ?>
2290 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2291     targetNamespace="http://foo.com"
2292     xmlns:foo="http://foo.com"
2293     name="MyValueWires" >
2294
2295     <wire source="MyValueServiceComponent/stockQuoteService"
2296         target="StockQuoteMediatorComponent" />
2297
2298 </composite>

```

## 2299 6.7 Composites which Include Component Implementations of 2300 Multiple Types

2301 A Composite containing multiple components can have multiple component implementation types.  
2302 For example, a Composite may include one component with a Java POJO as its implementation  
2303 and another component with a BPEL process as its implementation.

## 2304 6.8 Structural URI of Components

2305 The **structural URI** is a relative URI that describes each use of a given component in the Domain,  
2306 relative to the URI of the domain itself. It is never specified explicitly, but it calculated from the  
2307 configuration of the components configured into the Domain.

2308 A component in a composite may be used more than once in the domain, if its containing  
2309 composite is used as the implementation of more than one higher-level component. The structural  
2310 URI may be used to separately identify each use of a component - for example, the structural URI  
2311 may be used to attach different policies to each separate use of a component.

2312 For components directly deployed into the domain, the structural URI is simply the name of the  
2313 component.

2314 Where components are nested within a composite which is used as the implementation of a higher  
2315 level component, the structural URI consists of the name of the nested component prepended with  
2316 each of the names of the components upto and including the domain level component.

2317 For example, consider a component named Component1 at the domain level, where its  
2318 implementation is Composite1 which in turn contains a component named Component2, which is  
2319 implemented by Composite2 which contains a component named Component3. The three  
2320 components in this example have the following structural URIs:

- 2321 1. Component1: Component1
- 2322 2. Component2: Component1/Component2
- 2323 3. Component3: Component1/Component2/Component3

2324 The structural URI can also be extended to refer to specific parts of a component, such as a  
2325 service or a reference, by appending an appropriate fragment identifier to the component's  
2326 structural URI, as follows:

2327  
2328  
2329

- Service:  
#service(servicename)

2330  
2331  
2332

- Reference:  
#reference(referencename)

2333  
2334  
2335

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

2336  
2337

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

2338  
2339  
2340

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

2341

## 7 ConstrainingType

2342  
2343  
2344  
2345  
2346

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

2347  
2348  
2349  
2350  
2351  
2352

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

2353  
2354  
2355  
2356  
2357  
2358  
2359

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

2360  
2361  
2362

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

**Deleted:** 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.

2363  
2364

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

2365

2366

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

2367

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

2368

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

2369

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

2370

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

2371

2372

2373

2374

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

2375

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

2376

```
    </service>
```

2377

2378

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

2379

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

2380

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

2381

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

2382

```
    </reference>
```

2383

2384

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

2385

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

2386                   default-property-value?

2387                   </property>

2388

2389                   </constrainingType>

2390

2391                   The constrainingType element has the following **attributes**:

- 2392                   • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is  
2393                    an XML QName, in the namespace identified by the targetNamespace attribute. The name  
2394                    attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2395                   • **targetNamespace (0..1)** – an identifier for a target namespace into which the  
2396                    constrainingType is declared
- 2397                   • **requires (0..1)** – a list of policy intents. See [the Policy Framework specification \[10\]](#) for  
2398                    a description of this attribute.

2399                   ConstrainingType contains **zero or more properties, services, references**.

2400

2401                   When an implementation is constrained by a constrainingType its component type MUST contain  
2402                   all the services, references and properties specified in the constrainingType. [ASM70004] The  
2403                   constraining type's references and services will have interfaces specified and can have intents  
2404                   specified. An implementation MAY contain additional services, additional optional references  
2405                   (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the  
2406                   constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or  
2407                   1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

2408                   When a component is constrained by a constrainingType via the "constrainingType" attribute, the  
2409                   entire componentType associated with the component and its implementation is not visible to the  
2410                   containing composite. The containing composite can only see a projection of the componentType  
2411                   associated with the component and implementation as scoped by the constrainingType of the  
2412                   component. Additional services, references and properties provided by the implementation which  
2413                   are not declared in the constrainingType associated with a component MUST NOT be configured in  
2414                   any way by the containing composite. [ASM70006] This requirement ensures that the  
2415                   constrainingType contract cannot be violated by the composite.

2416                   The constrainingType can include required intents on any element. Those intents are applied to  
2417                   any component that uses that constrainingType. In other words, if requires="reliability" exists on  
2418                   a constrainingType, or its child service or reference elements, then a constrained component or its  
2419                   implementation must include requires="reliability" on the component or implementation or on its  
2420                   corresponding service or reference. A component or implementation can use a qualified form of  
2421                   an intent specified in unqualified form in the constrainingType, but if the constrainingType uses  
2422                   the qualified form of an intent, then the component or implementation MUST also use the qualified  
2423                   form, otherwise there is an error. [ASM70007]

2424                   A constrainingType can be applied to an implementation. In this case, the implementation's  
2425                   componentType has a constrainingType attribute set to the QName of the constrainingType.

2426

## 2427                   7.1 Example constrainingType

2428

2429                   The following snippet shows the contents of the component called "MyValueServiceComponent"  
2430                   which is constrained by the constrainingType myns:CT. The componentType associated with the  
2431                   implementation is also shown.

2432

2433                   <component name="MyValueServiceComponent" constrainingType="myns:CT">

2434                    <implementation.java class="services.myvalue.MyValueServiceImpl" />

```

2435     <property name="currency">EURO</property>
2436     <reference name="customerService" target="CustomerService">
2437         <binding.ws ...>
2438     <reference name="StockQuoteService"
2439         target="StockQuoteMediatorComponent"/>
2440 </component>
2441
2442 <constrainingType name="CT"
2443     targetNamespace="http://myns.com">
2444     <service name="MyValueService">
2445         <interface.java interface="services.myvalue.MyValueService"/>
2446     </service>
2447     <reference name="customerService">
2448         <interface.java interface="services.customer.CustomerService"/>
2449     </reference>
2450     <reference name="stockQuoteService">
2451         <interface.java interface="services.stockquote.StockQuoteService"/>
2452     </reference>
2453     <property name="currency" type="xsd:string"/>
2454 </constrainingType>

```

2455 The component MyValueServiceComponent is constrained by the constrainingType CT which  
2456 means that it must provide:

- 2457 • service **MyValueService** with the interface services.myvalue.MyValueService
- 2458 • reference **customerService** with the interface services.stockquote.StockQuoteService
- 2459 • reference **stockQuoteService** with the interface services.stockquote.StockQuoteService
- 2460 • property **currency** of type xsd:string.

## 2461 8 Interface

2462 **Interfaces** define one or more business functions. These business functions are provided by  
2463 Services and are used by References. A Service offers the business functionality of exactly one  
2464 interface for use by other components. Each interface defines one or more service **operations**  
2465 and each operation has zero or one **request (input) message** and zero or one **response**  
2466 **(output) message**. The request and response messages can be simple types such as a string  
2467 value or they can be complex types.

2468 SCA currently supports the following interface type systems:

- 2469 • Java interfaces
- 2470 • WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))
- 2471 • C++ classes
- 2472 • Collections of 'C' functions

2473 SCA is also extensible in terms of interface types. Support for other interface type systems can be  
2474 added through the extensibility mechanisms of SCA, as described in [the Extension Model section](#).

2475

2476 The following snippet shows the definition for the **interface** base element.

2477

```
2478 <interface requires="list of xs:QName"? policySets="list of xs:QName"?/>
```

2479

2480 The **interface** base element has the following **attributes**:

- 2481 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
2482 [\[10\]](#) for a description of this attribute
- 2483 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2484 [\[10\]](#) for a description of this attribute.

2485

2486 For information about Java interfaces, including details of SCA-specific annotations, see the SCA  
2487 Java Common Annotations and APIs specification [SCA-Common-Java].

2488 For information about WSDL interfaces, including details of SCA-specific extensions, see SCA-  
2489 Specific Aspects for WSDL Interfaces and WSDL Interface Type.

2490 For information about C++ interfaces, see the SCA C++ Client and Implementation Model  
2491 specification [SCA-CPP-Client].

2492 For information about C interfaces, see the SCA C Client and Implementation Model specification  
2493 [SCA-C-Client].

### 2494 8.1 Local and Remotable Interfaces

2495 A remotable service is one which may be called by a client which is running in an operating system  
2496 process different from that of the service itself (this also applies to clients running on different  
2497 machines from the service). Whether a service of a component implementation is remotable is  
2498 defined by the interface of the service. WSDL defined interfaces are always remotable. See the  
2499 relevant specifications for details of interfaces defined using other languages.

2500

2501 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**  
2502 interactions. Remotable service Interfaces MUST NOT make use of **method or operation**  
2503 **overloading.** [\[ASM80002\] This restriction on operation overloading for remotable services aligns](#)

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

2504 with the WSDL 2.0 specification, which disallows operation overloading, and also with the WS-I  
2505 Basic Profile 1.1 (section 4.5.3 - R2304) which has a constraint which disallows operation  
2506 overloading when using WSDL 1.1.  
2507

2508 Independent of whether the remotable service is called remotely from outside the process where  
2509 the service runs or from another component running in the same process, the data exchange  
2510 semantics are **by-value**.

2511 Implementations of remotable services can modify input messages (parameters) during or after  
2512 an invocation and can modify return messages (results) after the invocation. If a remotable  
2513 service is called locally or remotely, the SCA container MUST ensure sure that no modification of  
2514 input messages by the service or post-invocation modifications to return messages are seen by  
2515 the caller. [ASM80003]

2516 Here is a snippet which shows an example of a remotable java interface:

```
2517  
2518 package services.hello;  
2519  
2520 @Remotable  
2521 public interface HelloService {  
2522  
2523     String hello(String message);  
2524 }  
2525
```

2526 It is possible for the implementation of a remotable service to indicate that it can be called using  
2527 by-reference data exchange semantics when it is called from a component in the same process.  
2528 This can be used to improve performance for service invocations between components that run in  
2529 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java  
2530 Client and Implementation Specification](#)).

2531 A service typed by a local interface can only be called by clients that are running in the same  
2532 process as the component that implements the local service. Local services cannot be published  
2533 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2534 Java interface definition without a **@Remotable** annotation.

2535 The style of local interfaces is typically **fine grained** and intended for **tightly coupled**  
2536 interactions. Local service interfaces can make use of **method or operation overloading**.

2537 The data exchange semantic for calls to services typed by local interfaces is **by-reference**.

2538

## 2539 8.2 Bidirectional Interfaces

2540 The relationship of a business service to another business service is often peer-to-peer, requiring  
2541 a two-way dependency at the service level. In other words, a business service represents both a  
2542 consumer of a service provided by a partner business service and a provider of a service to the  
2543 partner business service. This is especially the case when the interactions are based on  
2544 asynchronous messaging rather than on remote procedure calls. The notion of **bidirectional  
2545 interfaces** is used in SCA to directly model peer-to-peer bidirectional business service  
2546 relationships.

2547 An interface element for a particular interface type system needs to allow the specification of an  
2548 optional callback interface. If a callback interface is specified, SCA refers to the interface as a  
2549 whole as a bidirectional interface.

2550 The following snippet shows the interface element defined using Java interfaces with an optional  
2551 callbackInterface attribute.

2552

```
2553 <interface.java interface="services.invoicing.ComputePrice"  
2554 callbackInterface="services.invoicing.InvoiceCallback"/>
```

2555

2556 If a service is defined using a bidirectional interface element then its implementation implements  
2557 the interface, and its implementation uses the callback interface to converse with the client that  
2558 called the service interface.

2559

2560 If a reference is defined using a bidirectional interface element, the client component  
2561 implementation using the reference calls the referenced service using the interface. The client  
2562 MUST provide an implementation of the callback interface. [ASM80004]

2563 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2564 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2565 mix local and remote services. [ASM80005]

2566 Note that an interface document such as a WSDL file or a Java interface can contain annotations  
2567 that declare a callback interface for a particular interface (see [the section on WSDL Interface type](#)  
2568 and the Java Common Annotations and APIs specification [SCA-Common-Java]). Whenever an  
2569 interface document declaring a callback interface is used in the declaration of an <interface/>  
2570 element in SCA, it MUST be treated as being bidirectional with the declared callback interface.  
2571 [ASM80010] In such cases, there is no requirement for the <interface/> element to declare the  
2572 callback interface explicitly.

2573 If an <interface/> element references an interface document which declares a callback interface  
2574 and also itself contains a declaration of a callback interface, the two callback interfaces MUST be  
2575 compatible. [ASM80011]

2576 Where a component uses an implementation and the component configuration explicitly declares  
2577 an interface for a service or a reference, if the matching service or reference declaration in the  
2578 component type declares an interface which has a callback interface, then the component interface  
2579 declaration MUST also declare a compatible interface with a compatible callback interface.  
2580 [ASM80012] If the service or reference declaration in the component type declares an interface  
2581 without a callback interface, then the component configuration for the corresponding service or  
2582 reference MUST NOT declare an interface with a callback interface. [ASM80013]

2583 Where a composite declares an interface for a composite service or a composite reference, if the  
2584 promoted service or promoted reference has an interface which has a callback interface, then the  
2585 interface declaration for the composite service or the composite reference MUST also declare a  
2586 compatible interface with a compatible callback interface. [ASM80014] If the promoted service or  
2587 promoted reference has an interface without a callback interface, then the interface declaration for  
2588 the composite service or composite reference MUST NOT declare a callback interface.  
2589 [ASM80015]

2590 See Section 6.4 Wires for a definition of "compatible interfaces".

2591 In a bidirectional interface, the service interface can have more than one operation defined, and  
2592 the callback interface can also have more than one operation defined. SCA runtimes MUST allow  
2593 an invocation of any operation on the service interface to be followed by zero, one or many  
2594 invocations of any of the operations on the callback interface. [ASM80009] These callback  
2595 operations can be invoked either before or after the operation on the service interface has  
2596 returned a response message, if there is one.

2597 For a given invocation of a service operation, which operations are invoked on the callback  
2598 interface, when these are invoked, the number of operations invoked, and their sequence are not  
2599 described by SCA. It is possible that this metadata about the bidirectional interface can be  
2600 supplied through mechanisms outside SCA. For example, it might be provided as a written  
2601 description attached to the callback interface.

## 2602 8.3 Conversational Interfaces

2603 Services sometimes cannot easily be defined so that each operation stands alone and is  
2604 completely independent of the other operations of the same service. Instead, there is a sequence  
2605 of operations that must be called in order to achieve some higher level goal. SCA calls this  
2606 sequence of operations a **conversation**. If the service uses a bidirectional interface, the  
2607 conversation may include both operations and callbacks.

2608 Such **conversational services** are typically managed by using conversation identifiers that are  
2609 either (1) part of the application data (message parts or operation parameters) or 2)  
2610 communicated separately from application data (possibly in headers). SCA introduces the concept  
2611 of **conversational interfaces** for describing the interface contract for conversational services of  
2612 the second form above. With this form, it is possible for the runtime to automatically manage the  
2613 conversation, with the help of an appropriate binding specified at deployment. SCA does not  
2614 standardize any aspect of conversational services that are maintained using application data.  
2615 Such services are neither helped nor hindered by SCA's conversational service support.

2616 Conversational services typically involve state data that relates to the conversation that is taking  
2617 place. The creation and management of the state data for a conversation has a significant impact  
2618 on the development of both clients and implementations of conversational services.

2619

2620 Traditionally, application developers who have needed to write conversational services have been  
2621 required to write a lot of plumbing code. They need to:

2622

- 2623 - choose or define a protocol to communicate conversational (correlation) information  
2624 between the client & provider
- 2625 - route conversational messages in the provider to a machine that can handle that  
2626 conversation, while handling concurrent data access issues
- 2627 - write code in the client to use/encode the conversational information
- 2628 - maintain state that is specific to the conversation, sometimes persistently and  
2629 transactionally, both in the implementation and the client.

2630

2631 SCA makes it possible to divide the effort associated with conversational services between a  
2632 number of roles:

- 2633 - Application Developer: Declares that a service interface is conversational (leaving the  
2634 details of the protocol up to the binding). Uses lifecycle semantics, APIs or other  
2635 programmatic mechanisms (as defined by the implementation-type being used) to  
2636 manage conversational state.
- 2637 - Application Assembler: chooses a binding that can support conversations
- 2638 - Binding Provider: implements a protocol that can pass conversational information with  
2639 each operation request/response.
- 2640 - Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for  
2641 application developers to access conversational information. Optionally implements  
2642 instance lifecycle semantics that automatically manage implementation state based on  
2643 the binding's conversational information.

2644

2645 There is a policy intent with the name **conversational** which is used to mark an interface as being  
2646 conversational in nature. Where a service or a reference has a conversational interface, the  
2647 conversational intent MUST be attached either to the interface itself, or to the service or reference  
2648 using the interface. [\[ASM80006\]](#) How to attach the conversational intent to an interface depends  
2649 on the type of the interface. For a WSDL interface, this is described in section 8.4 "[SCA-Specific](#)  
2650 [Aspects for WSDL Interfaces](#)". For a Java interface, it is described in the Java Common  
2651 Annotations and APIs specification. Note that setting the conversational intent on the service or

2652 reference element is useful when reusing an existing interface definition that contains no SCA  
2653 information, since it requires no modification of the interface artifact.

2654 The meaning of the conversational intent is that both the client and the provider of the interface  
2655 can assume that messages (in either direction) will be handled as part of an ongoing conversation  
2656 without depending on identifying information in the body of the message (i.e. in parameters of the  
2657 operations). In effect, the conversation interface specifies a high-level abstract protocol that must  
2658 be satisfied by any actual binding/policy combination used by the service.

2659 Examples of binding/policy combinations that support conversational interfaces are:

- 2660 - Web service binding with a WS-RM policy
- 2661 - Web service binding with a WS-Addressing policy
- 2662 - Web service binding with a WS-Context policy
- 2663 - JMS binding with a conversation policy that uses the JMS correlationID header

2664

2665 Conversations occur between one client and one target service. Consequently, requests originating  
2666 from one client to multiple target conversational services will result in multiple conversations. For  
2667 example, if a client A calls services B and C, both of which implement conversational interfaces,  
2668 two conversations result, one between A and B and another between A and C. Likewise, requests  
2669 flowing through multiple implementation instances will result in multiple conversations. For  
2670 example, a request flowing from A to B and then from B to C will involve two conversations (A and  
2671 B, B and C). In the previous example, if a request was then made from C to A, a third  
2672 conversation would result (and the implementation instance for A would be different from the one  
2673 making the original request).

2674 Invocation of any operation of a conversational interface can start a conversation. The decision on  
2675 whether an operation starts a conversation depends on the component's implementation and its  
2676 implementation type. Implementation types can support components which provide conversational  
2677 services. If an implementation type does provide this support, the specification for that  
2678 implementation type defines a mechanism for determining when a new conversation should be  
2679 used for an operation (for example, in Java, the conversation is new on the first use of an injected  
2680 reference; in BPEL, the conversation is new when the client's partnerLink comes into scope).

2681

2682 One or more operations in a conversational interface can be annotated with an  
2683 **endsConversation** annotation (the mechanism for annotating the interface depends on the  
2684 interface type) which indicates that when the operation is invoked, the conversation is at an end.  
2685 Where an interface is **bidirectional**, operations may also be annotated in this way on operations  
2686 of the callback interface. When a conversation ending operation is called, it indicates to both the  
2687 client and the service provider that the conversation is complete. Once an operation marked with  
2688 endsConversation has been invoked, any subsequent attempts to call an operation or a callback  
2689 operation associated with the same conversation MUST report a sca:ConversationViolation fault,  
2690 [ASM80007]

2691 A sca:ConversationViolation fault is thrown when one of the following errors occur:

- 2692 - A message is received for a particular conversation, after the conversation has ended
- 2693 - The conversation identification is invalid (not unique, out of range, etc.)
- 2694 - The conversation identification is not present in the input message of the operation that  
2695 ends the conversation
- 2696 - The client or the service attempts to send a message in a conversation, after the  
2697 conversation has ended

2698 This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI  
2699 <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2700 The lifecycle of resources and the association between unique identifiers and conversations are  
2701 determined by the service's implementation type and may not be directly affected by the

**Deleted:** 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.

2702 "endConversation" annotation. For example, a **WS-BPEL** process can outlive most of the  
2703 conversations that it is involved in.

2704 Although conversational interfaces do not require that any identifying information be passed as  
2705 part of the body of messages, there is conceptually an identity associated with the conversation.  
2706 Individual implementations types can have an API to access the ID associated with the  
2707 conversation, although no assumptions can be made about the structure of that identifier.  
2708 Implementation types can also have a means to set the conversation ID by either the client or the  
2709 service provider, although the operation may only be supported by some binding/policy  
2710 combinations.

2711 Implementation-type specifications are encouraged to define and provide conversational instance  
2712 lifecycle management for components that implement conversational interfaces. However,  
2713 implementations could also manage the conversational state manually.

2714

## 2715 8.4 Long-running Request-Response Operations

### 2716 8.4.1 Background

2717 A service offering one or more operations which map to a WSDL request-response pattern may be  
2718 implemented in a long-running, potentially interruptible, way. Consider a BPEL process with  
2719 receive and reply activities referencing the WSDL request-response operation. Between the two  
2720 activities, the business process logic may be a long-running sequence of steps, including activities  
2721 causing the process to be interrupted. Typical examples are steps where the process waits for  
2722 another message to arrive or a specified time interval to expire, or the process may perform  
2723 asynchronous interactions such as service invocations bound to asynchronous protocols or user  
2724 interactions. This is a common situation in business processes, and it causes the implementation  
2725 of the WSDL request-response operation to run for a very long time, e.g., several months (!). In  
2726 this case, it is not meaningful for any caller to remain in a synchronous wait for the response while  
2727 blocking system resources or holding database locks.

2728 Note that it is possible to model long-running interactions as a pair of two independent operations  
2729 as described in the section on bidirectional interfaces. However, it is a common practice (and in  
2730 fact much more convenient) to model a request-response operation and let the infrastructure deal  
2731 with the asynchronous message delivery and correlation aspects instead of putting this burden on  
2732 the application developer.

2733

### 2734 8.4.2 Definition of "long-running"

2735 A request-response operation is considered long-running if the implementation does not guarantee  
2736 the delivery of the response within any specified time interval. Clients invoking such request-  
2737 response operations are strongly discouraged from making assumptions about when the response  
2738 can be expected.

2739

### 2740 8.4.3 The asyncInvocation Intent

2741 This specification permits a long-running request-response operation or a complete interface  
2742 containing such operations to be marked using a policy intent with the name **asyncInvocation**. It  
2743 is also possible for a service to set the `asyncInvocation` intent when using an interface which is  
2744 not marked with the `asyncInvocation` intent. This can be useful when reusing an existing interface  
2745 definition that does not contain SCA information.

2746

#### 2747 8.4.4 Requirements on Bindings

2748 In order to support a service operation which is marked with the `asyncInvocation` intent, it is  
2749 necessary for the binding (and its associated policies) to support separate handling of the request  
2750 message and the response message. Bindings which only support a synchronous style of message  
2751 handling, such as a conventional HTTP binding, cannot be used to support long-running  
2752 operations.

2753 The requirements on a binding to support the `asyncInvocation` intent are the same as those  
2754 required to support services with bidirectional interfaces - namely that the binding needs to be  
2755 able to treat the transmission of the request message separately from the transmission of the  
2756 response message, with an arbitrarily large time interval between the two transmissions.

2757 An example of a binding/policy combination that supports long-running request-response  
2758 operations is a Web service binding used in conjunction with the WS-Addressing  
2759 "wsam:NonAnonymousResponses" assertion.

2760

#### 2761 8.4.5 Implementation Type Support

2762 SCA implementation types can provide special asynchronous client-side and asynchronous server-  
2763 side mappings to assist in the development of services and clients for long-running request-  
2764 response operations.

### 2765 8.5 SCA-Specific Aspects for WSDL Interfaces

2766 There are a number of aspects that SCA applies to interfaces in general, such as marking them  
2767 **conversational**. These aspects apply to the interfaces themselves, rather than their use in a  
2768 specific place within SCA. There is thus a need to provide appropriate ways of marking the  
2769 interface definitions themselves, which go beyond the basic facilities provided by the interface  
2770 definition language.

2771 For WSDL interfaces, there is an extension mechanism that permits additional information to be  
2772 included within the WSDL document. SCA takes advantage of this extension mechanism. In order  
2773 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-  
2774 open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2775 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach  
2776 policy intents - **@requires**. The definition of this attribute is as follows:

```
2777 <attribute name="requires" type="sca:listOfQNames"/>
```

2778

```
2779 <simpleType name="listOfQNames">
```

```
2780 <list itemType="QName"/>
```

```
2781 </simpleType>
```

2782 The `@requires` attribute can be applied to WSDL Port Type elements (WSDL 1.1). The attribute  
2783 contains one or more intent names, as defined by [the Policy Framework specification \[10\]](#). Any  
2784 service or reference that uses an interface marked with required intents MUST implicitly add those  
2785 intents to its own `@requires` list. **[ASM80008]**

2786 To specify that a WSDL interface is conversational, the following attribute setting is used on either  
2787 the WSDL Port Type or WSDL Interface:

```
2788 requires="conversational"
```

2789 SCA defines an **endsConversation** attribute that is used to mark specific operations within a  
2790 WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces  
2791 which are also marked conversational. The `endsConversation` attribute is a global attribute in the  
2792 SCA namespace, with the following definition:

```
2793 <attribute name="endsConversation" type="boolean" default="false"/>
```

2794

2795 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2796 the portType and the **endsConversation** attribute on one of the operations:

```
2797 ...  
2798 <portType name="LoanService" sca:requires="conversational">  
2799   <operation name="apply">  
2800     <input message="tns:ApplicationInput" />  
2801     <output message="tns:ApplicationOutput" />  
2802   </operation>  
2803   <operation name="cancel" sca:endsConversation="true">  
2804   </operation>  
2805   ...  
2806 </portType>  
2807 ...
```

2808 The following snippet is an example of a WSDL Port Type annotated with the **requires** attribute on  
2809 the portType and the **endsConversation** attribute on one of the operations:

```
2810 ...  
2811 <portType name="LoanService" sca:requires="conversational">  
2812   <operation name="apply">  
2813     <input message="tns:ApplicationInput" />  
2814     <output message="tns:ApplicationOutput" />  
2815   </operation>  
2816   <operation name="cancel" sca:endsConversation="true">  
2817   </operation>  
2818   ...  
2819 </portType>  
2820 ...
```

2821 SCA defines an attribute which is used to indicate that a given WSDL Port Type element (WSDL  
2822 1.1) has an associated callback interface. This is the @callback attribute, which applies to a WSDL  
2823 <portType/> element.

2824 The @callback attribute is defined as a global attribute in the SCA namespace, as follows:

```
2826 <attribute name="callback" type="QName" />
```

2827 The value of the @callback attribute is the QName of a Port Type. The port type declared by the  
2828 @callback attribute is the callback interface to use for the portType which is annotated by the  
2829 @callback attribute.

2830 Here is an example of a portType element with a callback attribute:

```
2833 <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
2834   <operation name="apply">  
2835     <input message="tns:ApplicationInput" />  
2836     <output message="tns:ApplicationOutput" />  
2837   </operation>  
2838   ...  
2839 </portType>
```

## 2841 8.6 WSDL Interface Type

2842 The WSDL interface type is used to declare interfaces for services and for references, where the interface  
2843 is defined in terms of a WSDL document. An interface is defined in terms of a WSDL 1.1 Port Type with  
2844 the arguments and return of the service operations described using XML schema.

2845  
2846 A WSDL interface is declared by an *interface.wSDL* element. The following shows the pseudo-schema  
2847 for the interface.wSDL element:

```
2848 <!-- WSDL Interface schema snippet -->  
2849 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?>
```

2850 The interface.wSDL element has the following *attributes*:

- 2851 • **interface (1..1)** - the URI of a WSDL Port Type

2852 The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.1  
2853 document. [ASM80001]

- 2854 • **callbackInterface(0..1)** - an optional callback interface, which is the URI of a WSDL Port  
2855 Type

2856 The interface.wSDL @callbackInterface attribute, if present, MUST reference a portType of a  
2857 WSDL 1.1 document. [ASM80016]

2858

2859 The form of the URI for WSDL port types follows the syntax described in the WSDL 1.1 Element  
2860 Identifiers specification [WSDL11\_Identifiers]

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

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

### 2861 8.6.1 Example of interface.wSDL

```
2862 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
2863 wSDL.portType(StockQuote)"  
2864 callbackInterface="http://www.stockquote.org/StockQuoteService#  
2865 wSDL.portType(StockQuoteCallback)"/>
```

2867 This declares an interface in terms of the WSDL port type "StockQuote" with a callback interface defined  
2868 by the "StockQuoteCallback" port type.

2869

## 2870 9 Binding

2871 Bindings are used by services and references. References use bindings to describe the access  
2872 mechanism used to call a service (which can be a service provided by another SCA composite).  
2873 Services use bindings to describe the access mechanism that clients (which can be a client from  
2874 another SCA composite) have to use to call the service.

2875 SCA supports the use of multiple different types of bindings. Examples include **SCA service, Web**  
2876 **service, stateless session EJB, data base stored procedure, EIS service**. An SCA runtime  
2877 MUST provide support for SCA service and Web service binding types. SCA provides an  
2878 extensibility mechanism by which an SCA runtime can add support for additional binding types.  
2879 For details on how additional binding types are defined, see the section on the Extension Model.

2880 A binding is defined by a **binding element** which is a child element of a service or of a reference  
2881 element in a composite. The following snippet shows the composite schema with the schema for  
2882 the binding element.

2883

```
2884 <?xml version="1.0" encoding="ASCII"?>
2885 <!-- Bindings schema snippet -->
2886 <composite ... >
2887   ...
2888   <service ... >*
2889     <interface ... />?
2890     <binding uri="xs:anyURI"? name="xs:NCName"?
2891       requires="list of xs:QName"?
2892       policySets="list of xs:QName"?>*
2893     <operation name="xs:NCName" requires="list of xs:QName"?
2894       policySets="list of xs:QName"?/>*
2895     <wireFormat/>?
2896     <operationSelector/>?
2897   </binding>
2898   <callback?>
2899     <binding uri="xs:anyURI"? name="xs:NCName"?
2900       requires="list of xs:QName"?
2901       policySets="list of xs:QName"?>+
2902     <operation name="xs:NCName" requires="list of xs:QName"?
2903       policySets="list of xs:QName"?/>*
2904     <wireFormat/>?
2905     <operationSelector/>?
2906   </binding>
2907 </callback>
2908 </service>
2909   ...
2910 <reference ... >*
2911   <interface ... />?
2912   <binding uri="xs:anyURI"? name="xs:NCName"?
2913     requires="list of xs:QName"?
2914     policySets="list of xs:QName"?>*
2915   <operation name="xs:NCName" requires="list of xs:QName"?
2916     policySets="list of xs:QName"?/>*
2917   <wireFormat/>?
2918   <operationSelector/>?
2919 </binding>
2920 <callback?>
2921   <binding uri="xs:anyURI"? name="xs:NCName"?
2922     requires="list of xs:QName"?
```

```

2923         policySets="list of xs:QName"?>+
2924         <operation name="xs:NCName" requires="list of xs:QName"?
2925             policySets="list of xs:QName"?/*
2926         <wireFormat/*?
2927         <operationSelector/*?
2928     </binding>
2929 </callback>
2930 </reference>
2931     ...
2932 </composite>
2933

```

2934 The element name of the binding element is architected; it is in itself a qualified name. The first  
 2935 qualifier is always named "binding", and the second qualifier names the respective binding-type  
 2936 (e.g. binding.composite, binding.ws, binding.ejb, binding.eis).

2937

2938 A binding element has the following attributes:

- 2939
- 2940 • **uri (0..1)** - has the following semantic.
    - 2941 ○ The uri attribute can be omitted.
    - 2942 ○ For a binding of a **reference** the URI attribute defines the target URI of the  
 2943 reference. This MUST be either the componentName/serviceName for a wire to an  
 2944 endpoint within the SCA domain, or the accessible address of some service  
 2945 endpoint either inside or outside the SCA domain (where the addressing scheme is  
 2946 defined by the type of the binding). [ASM90001]
    - 2947 ○ The circumstances under which the uri attribute can be used are defined in  
 section "5.3.1 Specifying the Target Service(s) for a Reference."
    - 2948 ○ For a binding of a **service** the URI attribute defines the URI relative to the  
 2949 component, which contributes the service to the SCA domain. The default value for  
 2950 the URI is the value of the name attribute of the binding.
  - 2951 • **name (0..1)** – a name for the binding instance (an NCName). The name attribute allows  
 2952 distinction between multiple binding elements on a single service or reference. The  
 2953 default value of the name attribute is the service or reference name. When a service or  
 2954 reference has multiple bindings, only one binding can have the default name value; all  
 2955 others must have a name value specified that is unique within the service or reference.  
 2956 [ASM90002] The name also permits the binding instance to be referenced from elsewhere  
 2957 – particularly useful for some types of binding, which can be declared in a definitions  
 2958 document as a template and referenced from other binding instances, simplifying the  
 2959 definition of more complex binding instances (see [the JMS Binding specification \[11\]](#) for  
 2960 examples of this referencing).
  - 2961 • **requires (0..1)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a  
 2962 description of this attribute.
  - 2963 • **policySets (0..1)** – a list of policy sets. See the [Policy Framework specification \[10\]](#) for a  
 2964 description of this attribute.

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

2965 A binding element has the following child elements:

- 2966
- 2967 • **operation: Operation (0..n)** - Zero or more operation elements. These elements are  
 2968 used to describe characteristics of individual operations within the interface. For a detailed  
 description of the operation element, see [the Policy Framework specification \[SCA Policy\]](#).
  - 2969 • **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See [the](#)  
 2970 [wireFormat section](#) for details.
  - 2971 • **operationSelector(0..1)** - an operationSelector element that is used to match a  
 2972 particular message to a particular operation in the interface. See [the operationSelector](#)  
 2973 [section](#) for details

2974 When multiple bindings exist for an service, it means that the service is available by any of the  
2975 specified bindings. The technique that the SCA runtime uses to choose among available bindings  
2976 is left to the implementation and it may include additional (nonstandard) configuration. Whatever  
2977 technique is used needs to be documented by the runtime.

2978 Services and References can always have their bindings overridden at the SCA domain level,  
2979 unless restricted by Intents applied to them.

2980 If a reference has any bindings they MUST be resolved which means that each binding MUST  
2981 include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference  
2982 MUST NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds  
2983 of bindings that are acceptable for use with a reference, the user specifies either policy intents or  
2984 policy sets.

2985 Users can also specifically wire, not just to a component service, but to a specific binding offered  
2986 by that target service. To do so, a wire target MAY be specified with a syntax of  
2987 "componentName/serviceName/bindingName". [ASM90004]  
2988

2989

2990 The following sections describe the SCA and Web service binding type in detail.

2991

## 2992 9.1 Messages containing Data not defined in the Service Interface

2993 It is possible for a message to include information that is not defined in the interface used to  
2994 define the service, for instance information may be contained in SOAP headers or as MIME  
2995 attachments.

2996 Implementation types can make this information available to component implementations in their  
2997 execution context. The specifications for these implementation types describe how this  
2998 information is accessed and in what form it is presented.

2999

## 3000 9.2 WireFormat

3001 A wireFormat is the form that a data structure takes when it is transmitted using some  
3002 communication binding. Another way to describe this is "the form that the data takes on the wire".  
3003 A wireFormat can be specific to a given communication method, or it may be general, applying to  
3004 many different communication methods. An example of a general wireFormat is XML text format.

3005 Where a particular SCA binding can accommodate transmitting data in more than one format, the  
3006 configuration of the binding MAY include a definition of the wireFormat to use. This is done using  
3007 an optional <sca:wireFormat/> subelement of the <binding/> element.

3008 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats  
3009 to be the default wireFormat which applies if no <wireFormat/> subelement is present.

3010 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a  
3011 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 3012 • <sca:wireFormat.xml/>
- 3013 • A wireFormat that transmits the data as an XML text datastructure
- 3014 • <sca:wireFormat.jms/>
- 3015 • The "default JMS wireFormat" as described in the JMS Binding specification

3016

3017 Specific wireFormats can have elements that include either attributes or subelements or both.

3018 For details about specific wireFormats, see the related SCA Binding specifications.

3019

## 3020 9.3 OperationSelector

3021 An operationSelector is necessary for some types of transport binding where messages are  
3022 transmitted across the transport without any explicit relationship between the message and the  
3023 interface operation to which it relates. SOAP is an example of a protocol where the messages do  
3024 contain explicit information that relates each message to the operation it targets. However, other  
3025 transport bindings have messages where this relationship is not expressed in the message or in  
3026 any related headers (pure JMS messages, for example). In cases where the messages arrive at a  
3027 service without any explicit information that maps them to specific operations, it is necessary for  
3028 the metadata attached to the service binding to contain the required mapping information. The  
3029 information is held in an operationSelector element which is a child element of the binding  
3030 element.

3031 The base sca:operationSelector element is abstract and it has no attributes and no child elements.  
3032 For a particular operationSelector, an extension subtype is defined, using substitution groups, for  
3033 example:

- 3034
- <sca:operationSelector.XPath/>
  - An operation selector that uses XPath to filter out specific messages and target them to particular named operations.
- 3035  
3036

3037

3038 Specific operationSelectors can have elements that include either attributes or subelements or  
3039 both.

3040 For details about specific operationSelectors, see the related SCA Binding specifications.

3041

## 3042 9.4 Form of the URI of a Deployed Binding

3043 SCA Bindings specifications can choose to use the **structural URI** defined in the section  
3044 "[Structural URI of Components](#)" above to derive a binding specific URI according to some Binding-  
3045 related scheme. The relevant binding specification describes this.

3046 Alternatively, <binding/> elements have an optional @URI attribute, which is termed a  
3047 bindingURI.

3048 If the bindingURI is specified on a given <binding/> element, the binding can optionally use it to  
3049 derive an endpoint URI relevant to the binding. The derivation is binding specific and is described  
3050 by the relevant binding specification.

3051 For binding.sca, which is described in the SCA Assembly specification, this is as follows:

- 3052
- If the binding uri attribute is specified on a reference, it identifies the target service in the SCA domain by specifying the service's structural URI.
  - If the binding uri attribute is specified on a service, it is ignored.
- 3053  
3054

3055

### 3056 9.4.1 Non-hierarchical URIs

3057 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) may optionally make  
3058 use of the "uri" attribute, which is the complete representation of the URI for that service  
3059 binding. Where the binding does not use the "uri" attribute, the binding needs to offer a different  
3060 mechanism for specifying the service address.

### 3061 9.4.2 Determining the URI scheme of a deployed binding

3062 One of the things that needs to be determined when building the effective URI of a deployed  
3063 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is  
3064 binding type specific.

3065 If the binding type supports a single protocol then there is only one URI scheme associated with it.  
3066 In this case, that URI scheme is used.

3067 If the binding type supports multiple protocols, the binding type implementation determines the  
3068 URI scheme by introspecting the binding configuration, which may include the policy sets  
3069 associated with the binding.

3070 A good example of a binding type that supports multiple protocols is binding.ws, which can be  
3071 configured by referencing either an "abstract" WSDL element (i.e. portType or interface) or a  
3072 "concrete" WSDL element (i.e. binding, port or endpoint). When the binding references a PortType  
3073 or Interface, the protocol and therefore the URI scheme is derived from the intents/policy sets  
3074 attached to the binding. When the binding references a "concrete" WSDL element, there are two  
3075 cases:

3076 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most  
3077 common case. In this case, the URI scheme is given by the protocol/transport specified in the  
3078 WSDL binding element.

3079 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,  
3080 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"  
3081 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined  
3082 by looking at the policy sets attached to the binding.

3083 It's worth noting that an intent supported by a binding type may completely change the behavior  
3084 of the binding. For example, when the intent "confidentiality/transport" is required by an HTTP  
3085 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to  
3086 "https".

3087

## 3088 9.5 SCA Binding

3089 The SCA binding element is defined by the following schema.

3090

3091 `<binding.sca />`

3092

3093 The SCA binding can be used for service interactions between references and services contained  
3094 within the SCA domain. The way in which this binding type is implemented is not defined by the  
3095 SCA specification and it can be implemented in different ways by different SCA runtimes. The only  
3096 requirement is that the required qualities of service must be implemented for the SCA binding  
3097 type. The SCA binding type is **not** intended to be an interoperable binding type. For  
3098 interoperability, an interoperable binding type such as the Web service binding should be used.

3099 A service definition with no binding element specified uses the SCA binding.

3100 `<binding.sca/>` would only have to be specified in override cases, or when you specify a  
3101 set of bindings on a service definition and the SCA binding should be one of them.

3102 If a reference does not have a binding, then the binding used can be any of the bindings  
3103 specified by the service provider, as long as the intents required by the reference and  
3104 the service are all respected.

3105 If the interface of the service or reference is local, then the local variant of the SCA  
3106 binding will be used. If the interface of the service or reference is remotable, then either  
3107 the local or remote variant of the SCA binding will be used depending on whether source  
3108 and target are co-located or not.

3109 If a reference specifies an URI via its uri attribute, then this provides the default wire to a service  
3110 provided by another domain level component. The value of the URI has to be as follows:

3111 

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

3112

### 3113 9.5.1 Example SCA Binding

3114 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
3115 containing the service element for the MyValueService and a reference element for the  
3116 StockQuoteService. Both the service and the reference use an SCA binding. The target for the  
3117 reference is left undefined in this binding and would have to be supplied by the composite in which  
3118 this composite is used.

3119

```
3120 <?xml version="1.0" encoding="ASCII"?>
3121 <!-- Binding SCA example -->
3122 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3123           targetNamespace="http://foo.com"
3124           name="MyValueComposite" >
3125
3126     <service name="MyValueService" promote="MyValueComponent">
3127       <interface.java interface="services.myvalue.MyValueService"/>
3128       <binding.sca/>
3129       ...
3130     </service>
3131
3132     ...
3133
3134     <reference name="StockQuoteService"
3135               promote="MyValueComponent/StockQuoteReference">
3136       <interface.java interface="services.stockquote.StockQuoteService"/>
3137       <binding.sca/>
3138     </reference>
3139
3140 </composite>
3141
```

### 3142 9.6 Web Service Binding

3143 SCA defines a Web services binding. This is described in a [separate specification document \[9\]](#).

3144

### 3145 9.7 JMS Binding

3146 SCA defines a JMS binding. This is described in a [separate specification document \[11\]](#).

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  
3181

## 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 \[11\]](#).

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

## 3182 11 Extension Model

3183

3184 The assembly model can be extended with support for new interface types, implementation types  
3185 and binding types. The extension model is based on XML schema substitution groups. There are  
3186 three XML Schema substitution group heads defined in the SCA namespace: **interface**,  
3187 **implementation** and **binding**, for interface types, implementation types and binding types,  
3188 respectively.

3189 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1],  
3190 [9], [11]) use these XML Schema substitution groups to define some basic types of interfaces,  
3191 implementations and bindings, but other types can be defined as required, where support for  
3192 these extra ones is available from the runtime. The interface type elements, implementation type  
3193 elements, and binding type elements defined by the SCA specifications are all part of the SCA  
3194 namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200712"), as indicated in their  
3195 respective schemas. New interface types, implementation types and binding types that are defined  
3196 using this extensibility model, which are not part of these SCA specifications are defined in  
3197 namespaces other than the SCA namespace.

3198 The "." notation is used in naming elements defined by the SCA specifications ( e.g.  
3199 <implementation.java ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel  
3200 extensibility approach but as a naming convention that improves usability of the SCA assembly  
3201 language.

3202

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

3205

### 3206 11.1 Defining an Interface Type

3207 The following snippet shows the base definition for the **interface** element and **Interface** type  
3208 contained in **sca-core.xsd**; see appendix for complete schema.

3209

```
3210 <?xml version="1.0" encoding="UTF-8"?>
3211 <!-- (c) Copyright SCA Collaboration 2006 -->
3212 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3213         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3214         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3215         elementFormDefault="qualified">
3216
3217     ...
3218
3219     <element name="interface" type="sca:Interface" abstract="true"/>
3220     <complexType name="Interface"/>
3221     <complexType name="Interface" abstract="true">
3222         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3223         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3224     </complexType>
3225
```

3226  
3227  
3228  
3229

...

</schema>

3230 In the following snippet is an example of how the base definition is extended to support Java  
3231 interfaces. The snippet shows the definition of the *interface.java* element and the  
3232 **JavaInterface** type contained in *sca-interface-java.xsd*.

3233

```
3234 <?xml version="1.0" encoding="UTF-8"?>
3235 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3236         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3237         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3238
3239     <element name="interface.java" type="sca:JavaInterface"
3240             substitutionGroup="sca:interface"/>
3241     <complexType name="JavaInterface">
3242         <complexContent>
3243             <extension base="sca:Interface">
3244                 <attribute name="interface" type="NCName"
3245                         use="required"/>
3246             </extension>
3247         </complexContent>
3248     </complexType>
3249 </schema>
```

3250 In the following snippet is an example of how the base definition can be extended by other  
3251 specifications to support a new interface not defined in the SCA specifications. The snippet shows  
3252 the definition of the *my-interface-extension* element and the *my-interface-extension-type*  
3253 type.

3254

```
3255 <?xml version="1.0" encoding="UTF-8"?>
3256 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3257         targetNamespace="http://www.example.org/myextension"
3258         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3259         xmlns:tns="http://www.example.org/myextension">
3260
3261     <element name="my-interface-extension"
3262             type="tns:my-interface-extension-type"
3263             substitutionGroup="sca:interface"/>
3264     <complexType name="my-interface-extension-type">
3265         <complexContent>
3266             <extension base="sca:Interface">
3267                 ...
3268             </extension>
3269         </complexContent>
3270     </complexType>
```

3270 </schema>  
3271

## 3272 11.2 Defining an Implementation Type

3273 The following snippet shows the base definition for the *implementation* element and  
3274 **Implementation** type contained in *sca-core.xsd*; see appendix for complete schema.

```
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     <element name="implementation" type="sca:Implementation"  
3283     abstract="true"/>  
3284     <complexType name="Implementation"/>  
3285     ...  
3286 </schema>
```

3293 In the following snippet we show how the base definition is extended to support Java  
3294 implementation. The snippet shows the definition of the *implementation.java* element and the  
3295 **JavaImplementation** type contained in *sca-implementation-java.xsd*.

```
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     <element name="implementation.java" type="sca:JavaImplementation"  
3301     substitutionGroup="sca:implementation"/>  
3302     <complexType name="JavaImplementation">  
3303     <complexContent>  
3304     <extension base="sca:Implementation">  
3305     <attribute name="class" type="NCName"  
3306     use="required"/>  
3307     </extension>  
3308     </complexContent>  
3309     </complexType>  
3310 </schema>
```

3313 In the following snippet is an example of how the base definition can be extended by other  
3314 specifications to support a new implementation type not defined in the SCA specifications. The  
3315 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**  
3316 **type** type.

```
3317 <?xml version="1.0" encoding="UTF-8"?>
3318 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3319         targetNamespace="http://www.example.org/myextension"
3320         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3321         xmlns:tns="http://www.example.org/myextension">
3322
3323     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3324             substitutionGroup="sca:implementation"/>
3325     <complexType name="my-impl-extension-type">
3326         <complexContent>
3327             <extension base="sca:Implementation">
3328                 ...
3329             </extension>
3330         </complexContent>
3331     </complexType>
3332 </schema>
3333
```

3334 In addition to the definition for the new implementation instance element, there needs to be an  
3335 associated implementationType element which provides metadata about the new implementation  
3336 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3337 <implementationType type="xs:QName"
3338                   alwaysProvides="list of intent xs:QName"
3339                   mayProvide="list of intent xs:QName"/>
3340
```

3341 The implementation type has the following attributes:

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

3349

## 3350 11.3 Defining a Binding Type

3351 The following snippet shows the base definition for the **binding** element and **Binding** type  
3352 contained in **sca-core.xsd**; see appendix for complete schema.

```
3353
3354 <?xml version="1.0" encoding="UTF-8"?>
3355 <!-- binding type schema snippet -->
3356 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
```

```

3357 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3358         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3359         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3360         elementFormDefault="qualified">
3361
3362     ...
3363
3364     <element name="binding" type="sca:Binding" abstract="true"/>
3365     <complexType name="Binding">
3366         <attribute name="uri" type="anyURI" use="optional"/>
3367         <attribute name="name" type="NCName" use="optional"/>
3368         <attribute name="requires" type="sca:listOfQNames"
3369             use="optional"/>
3370         <attribute name="policySets" type="sca:listOfQNames"
3371             use="optional"/>
3372     </complexType>
3373
3374     ...
3375
3376 </schema>

```

3377 In the following snippet is an example of how the base definition is extended to support Web  
3378 service binding. The snippet shows the definition of the **binding.ws** element and the  
3379 **WebServiceBinding** type contained in **sca-binding-webservice.xsd**.

```

3380
3381 <?xml version="1.0" encoding="UTF-8"?>
3382 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3383         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3384         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3385
3386     <element name="binding.ws" type="sca:WebServiceBinding"
3387         substitutionGroup="sca:binding"/>
3388     <complexType name="WebServiceBinding">
3389         <complexContent>
3390             <extension base="sca:Binding">
3391                 <attribute name="port" type="anyURI" use="required"/>
3392             </extension>
3393         </complexContent>
3394     </complexType>
3395 </schema>

```

3396 In the following snippet is an example of how the base definition can be extended by other  
3397 specifications to support a new binding not defined in the SCA specifications. The snippet shows  
3398 the definition of the **my-binding-extension** element and the **my-binding-extension-type** type.

```

3399 <?xml version="1.0" encoding="UTF-8"?>
3400 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

3401         targetNamespace="http://www.example.org/myextension"
3402         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3403         xmlns:tns="http://www.example.org/myextension">
3404
3405     <element name="my-binding-extension"
3406         type="tns:my-binding-extension-type"
3407         substitutionGroup="sca:binding" />
3408     <complexType name="my-binding-extension-type">
3409         <complexContent>
3410             <extension base="sca:Binding">
3411                 ...
3412             </extension>
3413         </complexContent>
3414     </complexType>
3415 </schema>
3416

```

3417 In addition to the definition for the new binding instance element, there needs to be an associated  
3418 bindingType element which provides metadata about the new binding type. The pseudo schema  
3419 for the bindingType element is shown in the following snippet:

```

3420 <bindingType type="xs:QName"
3421     alwaysProvides="list of intent QNames"?
3422     mayProvide = "list of intent QNames"?/>
3423

```

3424 The binding type has the following attributes:

- 3425 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3426 intended to be the QName of the binding element for the binding type, such as  
3427 "sca:binding.ws"
- 3428 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3429 [the Policy Framework specification \[10\]](#) for details.
- 3430 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)  
3431 [Policy Framework specification \[10\]](#) for details.

## 3432 11.4 Defining an Import Type

3433 The following snippet shows the base definition for the *import* element and *Import* type contained in *sca-*  
3434 *core.xsd*; see appendix for complete schema.

```

3435
3436 <?xml version="1.0" encoding="UTF-8"?>
3437 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3438 IPR and other policies apply. -->
3439 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3440     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3441     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3442     elementFormDefault="qualified">
3443     ...
3444
3445     <!-- Import -->
3446     <element name="importBase" type="sca:Import" abstract="true" />
3447

```

```

3448     <complexType name="Import" abstract="true">
3449         <complexContent>
3450             <extension base="sca:CommonExtensionBase">
3451                 <sequence>
3452                     <any namespace="##other" processContents="lax" minOccurs="0"
3453                         maxOccurs="unbounded" />
3454                 </sequence>
3455             </extension>
3456         </complexContent>
3457     </complexType>
3458
3459     <element name="import" type="sca:ImportType"
3460         substitutionGroup="sca:importBase" />
3461     <complexType name="ImportType">
3462         <complexContent>
3463             <extension base="sca:Import">
3464                 <attribute name="namespace" type="string" use="required" />
3465                 <attribute name="location" type="anyURI" use="required" />
3466             </extension>
3467         </complexContent>
3468     </complexType>
3469
3470     ...
3471
3472 </schema>
3473

```

3474 In the following snippet we show how the base import definition is extended to support Java imports. In  
3475 the import element, the namespace is expected to be an XML namespace, an import.java element uses a  
3476 Java package name instead. The snippet shows the definition of the **import.java** element and the  
3477 **JavaImportType** type contained in **sca-import-java.xsd**.

```

3478
3479 <?xml version="1.0" encoding="UTF-8"?>
3480 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3481     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3482     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3483
3484     <element name="import.java" type="sca:JavaImportType"
3485         substitutionGroup="sca:importBase" />
3486     <complexType name="JavaImportType">
3487         <complexContent>
3488             <extension base="sca:Import">
3489                 <attribute name="package" type="xs:String" use="required" />
3490                 <attribute name="location" type="xs:AnyURI" use="optional" />
3491             </extension>
3492         </complexContent>
3493     </complexType>
3494 </schema>
3495

```

3496 In the following snippet we show an example of how the base definition can be extended by other  
3497 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3498 definition of the **my-import-extension** element and the **my-import-extension-type** type.

```

3499
3500 <?xml version="1.0" encoding="UTF-8"?>
3501 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3502     targetNamespace="http://www.example.org/myextension"

```

```

3503     xmlns:sca=" http://docs.oasis-open.org/ns/opencsa/sca/200712"
3504     xmlns:tns="http://www.example.org/myextension">
3505
3506     <element name="my-import-extension"
3507         type="tns:my-import-extension-type"
3508         substitutionGroup="sca:importBase"/>
3509     <complexType name="my-import-extension-type">
3510         <complexContent>
3511             <extension base="sca:Import">
3512                 ...
3513             </extension>
3514         </complexContent>
3515     </complexType>
3516 </schema>

```

3518 For a complete example using this extension point, see the definition of *import.java* in the SCA Java  
3519 Common Annotations and APIs Specification [SCA-Java].

## 3520 11.5 Defining an Export Type

3521 The following snippet shows the base definition for the *export* element and *ExportType* type contained in  
3522 *sca-core.xsd*; see appendix for complete schema.

```

3523
3524 <?xml version="1.0" encoding="UTF-8"?>
3525 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
3526 IPR and other policies apply. -->
3527 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3528     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3529     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3530     elementFormDefault="qualified">
3531
3532     ...
3533     <!-- Export -->
3534     <element name="exportBase" type="sca:Export" abstract="true" />
3535     <complexType name="Export" abstract="true">
3536         <complexContent>
3537             <extension base="sca:CommonExtensionBase">
3538                 <sequence>
3539                     <any namespace="##other" processContents="lax" minOccurs="0"
3540                         maxOccurs="unbounded"/>
3541                 </sequence>
3542             </extension>
3543         </complexContent>
3544     </complexType>
3545
3546     <element name="export" type="sca:ExportType"
3547         substitutionGroup="sca:exportBase"/>
3548     <complexType name="ExportType">
3549         <complexContent>
3550             <extension base="sca:Export">
3551                 <attribute name="namespace" type="string" use="required"/>
3552             </extension>
3553         </complexContent>
3554     </complexType>
3555     ...
3556 </schema>

```

3557

3558 The following snippet shows how the base definition is extended to support Java exports. In a base  
3559 *export* element, the *@namespace* attribute specifies XML namespace being exported. An *export.java*  
3560 element uses a *@package* attribute to specify the Java package to be exported. The snippet shows the  
3561 definition of the *export.java* element and the *JavaExport* type contained in *sca-export-java.xsd*.

3562

```
3563 <?xml version="1.0" encoding="UTF-8"?>
3564 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3565         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3566         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
3567
3568     <element name="export.java" type="sca:JavaExportType"
3569           substitutionGroup="sca:exportBase" />
3570     <complexType name="JavaExportType">
3571       <complexContent>
3572         <extension base="sca:Export">
3573           <attribute name="package" type="xs:String" use="required" />
3574         </extension>
3575       </complexContent>
3576     </complexType>
3577 </schema>
```

3578

3579 In the following snippet we show an example of how the base definition can be extended by other  
3580 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3581 definition of the *my-export-extension* element and the *my-export-extension-type* type.

3582

```
3583 <?xml version="1.0" encoding="UTF-8"?>
3584 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3585         targetNamespace="http://www.example.org/myextension"
3586         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200712"
3587         xmlns:tns="http://www.example.org/myextension">
3588
3589     <element name="my-export-extension"
3590           type="tns:my-export-extension-type"
3591           substitutionGroup="sca:exportBase" />
3592     <complexType name="my-export-extension-type">
3593       <complexContent>
3594         <extension base="sca:Export">
3595           ...
3596         </extension>
3597       </complexContent>
3598     </complexType>
3599 </schema>
```

3600

3601 For a complete example using this extension point, see the definition of *export.java* in the SCA Java  
3602 Common Annotations and APIs Specification [SCA-Java].

3603

---

## 3604 12 Packaging and Deployment

### 3605 12.1 Domains

3606 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3607 of interconnected runtime nodes.

3608 A single SCA domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3609 wires can only be used to connect components within a single SCA domain. Connections to  
3610 services outside the domain must use binding specific mechanisms for addressing services (such  
3611 as WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used  
3612 in the context of a single domain. In general, external clients of a service that is developed and  
3613 deployed using SCA should not be able to tell that SCA was used to implement the service – it is  
3614 an implementation detail.

3615 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3616 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3617 functionality controlled by a single organization. For example, an SCA Domain may be the whole  
3618 of a business, or it may be a department within a business.

3619 As an example, for the accounts department in a business, the SCA Domain might cover all  
3620 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3621 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3622 An SCA domain has the following:

- 3623 • A virtual domain-level composite whose components are deployed and running
- 3624 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3625 necessary to execute components
- 3626 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3627 level composite.

3628 The information associated with an SCA domain can be stored in many ways, including but not  
3629 limited to a specific filesystem structure or a repository.

### 3630 12.2 Contributions

3631 An SCA domain might require a large number of different artifacts in order to work. These  
3632 artifacts include artifacts defined by SCA and other artifacts such as object code files and interface  
3633 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3634 different SCA definition documents are: composite, componentType, constrainingType and  
3635 definitions. XML artifacts that are not defined by SCA but which may be needed by an SCA  
3636 domain include XML Schema documents, WSDL documents, and BPEL documents. SCA  
3637 constructs, like other XML-defined constructs, use XML qualified names for their identity (i.e.  
3638 namespace + local name).

3639 Non-XML artifacts are also required within an SCA domain. The most obvious examples of such  
3640 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3641 implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3642 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3643 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3644 packaging formats, but requires that the ZIP format be supported. When using the ZIP format for  
3645 deploying a contribution, this specification does not specify whether that format is retained after  
3646 deployment. For example, a Java EE based SCA runtime may convert the ZIP package to an EAR  
3647 package. SCA expects certain characteristics of any packaging:

- 3648 • For any contribution packaging it MUST be possible to present the artifacts of the  
3649 packaging to SCA as a hierarchy of resources based off of a single root [\[ASM12001\]](#)

3650 • Within any contribution packaging A directory resource SHOULD exist at the root of the  
3651 hierarchy named META-INF [ASM12002]

3652 • Within any contribution packaging a document SHOULD exist directly under the META-INF  
3653 directory named sca-contribution.xml which lists the SCA Composites within the  
3654 contribution that are runnable. [ASM12003]

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 reported 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

Deleted: generated

3664 See the section "SCA Contribution Metadata Document" for details of the format of this  
3665 file.  
3666

3667 To illustrate that a variety of packaging formats can be used with SCA, the following are examples  
3668 of formats that might be used to package SCA artifacts and metadata (as well as other artifacts)  
3669 as a contribution:

- 3670 • A filesystem directory
- 3671 • An OSGi bundle
- 3672 • A compressed directory (zip, gzip, etc)
- 3673 • A JAR file (or its variants – WAR, EAR, etc)

3674 Contributions do not contain other contributions. If the packaging format is a JAR file that  
3675 contains other JAR files (or any similar nesting of other technologies), the internal files are not  
3676 treated as separate SCA contributions. It is up to the implementation to determine whether the  
3677 internal JAR file should be represented as a single artifact in the contribution hierarchy or whether  
3678 all of the contents should be represented as separate artifacts.

3679 A goal of SCA's approach to deployment is that the contents of a contribution should not need to  
3680 be modified in order to install and use the contents of the contribution in a domain.

3681

## 3682 12.2.1 SCA Artifact Resolution

3683 Contributions can be self-contained, in that all of the artifacts necessary to run the contents of the  
3684 contribution are found within the contribution itself. However, it can also be the case that the  
3685 contents of the contribution make one or many references to artifacts that are not contained  
3686 within the contribution. These references can be to SCA artifacts such as composites or they can  
3687 be to other artifacts such as WSDL files, XSD files or to code artifacts such as Java class files and  
3688 BPEL process files. Note: This form of artifact resolution does not apply to imports of composite  
3689 files, as described in Section 6.6.

3690 A contribution can use some artifact-related or packaging-related means to resolve artifact  
3691 references. Examples of such mechanisms include:

- 3692 • wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema  
3693 artifacts respectively
- 3694 • OSGi bundle mechanisms for resolving Java class and related resource dependencies

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

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

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.

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]

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]

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]

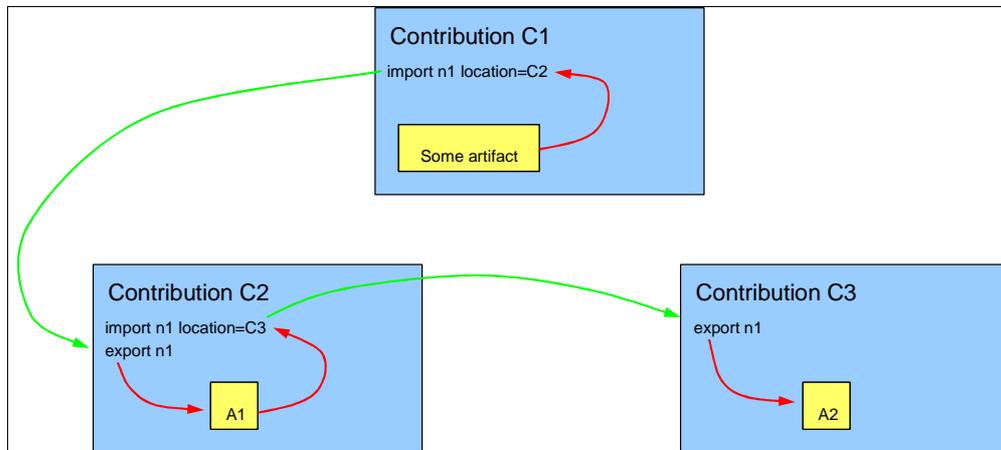
3746 For example:

- 3747 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the  
3748 "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".

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

**Deleted:** When a contribution contains a reference to an artifact from a namespace that is declared in an import statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the SCA runtime MUST resolve artifacts in the following order:  
1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to locate artifacts (ie only a one-level search is performed).  
2. from the contents of the contribution itself.

3752



3753

3754 The "A2" artifact is contained within the third contribution "C3" from which it is resolved by the  
3755 contribution "C2". The "C3" contribution is never used to resolve artifacts directly for the "C1"  
3756 contribution, since "C3" is not declared as an import location for "C1".

3757 For example, if for a contribution "C1", an import is used to resolve a composite "X1" contained in  
3758 contribution "C2", and composite "X1" contains references to other artifacts such as WSDL files or  
3759 XSDs, those references in "X1" are resolved in the context of contribution "C2" and not in the  
3760 context of contribution "C1".

3761 The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through  
3762 resolving an import statement. [ASM12024]

3763 The SCA runtime MUST report an error if an artifact cannot be resolved by the precedence order  
3764 above. [ASM12025]

3765

**Deleted:** The SCA runtime MUST raise an error if an artifact cannot be resolved by the precedence order above.

## 3766 12.2.2 SCA Contribution Metadata Document

3767 The contribution optionally contains a document that declares runnable composites, exported  
3768 definitions and imported definitions. The document is found at the path of META-INF/sca-  
3769 contribution.xml relative to the root of the contribution. Frequently some SCA metadata needs to  
3770 be specified by hand while other metadata is generated by tools (such as the <import> elements  
3771 described below). To accommodate this, it is also possible to have an identically structured  
3772 document at META-INF/sca-contribution-generated.xml. If this document exists (or is generated  
3773 on an as-needed basis), it will be merged into the contents of sca-contribution.xml, with the  
3774 entries in sca-contribution.xml taking priority if there are any conflicting declarations.  
3775

3776 The format of the document is:

```
3777 <?xml version="1.0" encoding="ASCII"?>
3778 <!-- sca-contribution pseudo-schema -->
3779 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200712>
```

3780

Figure 14: Example of SCA Artifact Resolution between Contributions

```
3781     <deployable composite="xs:QName" />*
3782     <import namespace="xs:String" location="xs:AnyURI"? />*
3783     <export namespace="xs:String" />*
3784
3785 </contribution>
3786
```

3787 **deployable element:** Identifies a composite which is a composite within the contribution that is a  
3788 composite intended for potential inclusion into the virtual domain-level composite. Other  
3789 composites in the contribution are not intended for inclusion but only for use by other composites.  
3790 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)  
3791 [Composite](#) capability and the add To Domain Level Composite capability.

3792 Attributes of the deployable element:

- 3793 • **composite (1..1)** – The QName of a composite within the contribution.

3794

3795 **Export element:** A declaration that artifacts belonging to a particular namespace are exported  
3796 and are available for use within other contributions. An export declaration in a contribution  
3797 specifies a namespace, all of whose definitions are considered to be exported. By default,  
3798 definitions are not exported.

3799 The SCA artifact export is useful for SCA domains containing heterogeneous mixtures of  
3800 contribution packagings and technologies, where artifact-related or packaging-related mechanisms  
3801 are unlikely to work across different kinds of contribution.

3802 Attributes of the export element:

- 3803 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3804 should be the namespace URI for the exported definitions. For XML technologies that  
3805 define multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port  
3806 types are a different symbol space from WSDL bindings), all definitions from all symbol  
3807 spaces are exported.

3808 Technologies that use naming schemes other than QNames must use a different export  
3809 element from the same substitution group as the the SCA <export> element. The  
3810 element used identifies the technology, and can use any value for the namespace that is  
3811 appropriate for that technology. For example, <export.java> can be used can be used to  
3812 export java definitions, in which case the namespace is a fully qualified package name.  
3813

3814

3815 **Import element:** Import declarations specify namespaces of definitions that are needed by the  
3816 definitions and implementations within the contribution, but which are not present in the  
3817 contribution. It is expected that in most cases import declarations will be generated based on  
3818 introspection of the contents of the contribution. In this case, the import declarations would be  
3819 found in the META-INF/ sca-contribution-generated.xml document.

3820 Attributes of the import element:

- 3821 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3822 is the namespace URI for the imported definitions. For XML technologies that define  
3823 multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are  
3824 a different symbol space from WSDL bindings), all definitions from all symbol spaces are  
3825 imported.

3826 Technologies that use naming schemes other than QNames must use a different import  
3827 element from the same substitution group as the the SCA <import> element. The  
3828 element used identifies the technology, and can use any value for the namespace that is  
3829 appropriate for that technology. For example, <import.java> can be used can be used to  
3830 import java definitions, in which case the namespace is a fully qualified package name.  
3831

3832 • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific  
3833 requirements for the form of this URI, nor the means by which it is resolved. It can point  
3834 to another contribution (through its URI) or it can point to some location entirely outside  
3835 the SCA Domain.  
3836

3837 It is expected that SCA runtimes can define implementation specific ways of resolving location  
3838 information for artifact resolution between contributions. These mechanisms will however usually  
3839 be limited to sets of contributions of one runtime technology and one hosting environment.

3840 In order to accommodate imports of artifacts between contributions of disparate runtime  
3841 technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location  
3842 specification.

3843 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are  
3844 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact  
3845 location specifications.

3846 The order in which the import statements are specified can play a role in this mechanism. Since  
3847 definitions of one namespace can be distributed across several artifacts, multiple import  
3848 declarations can be made for one namespace.  
3849

3850 The location value is only a default, and dependent contributions listed in the call to  
3851 installContribution can override the value if there is a conflict. However, the specific mechanism  
3852 for resolving conflicts between contributions that define conflicting definitions is implementation  
3853 specific.

3854 If the value of the location attribute is an SCA contribution URI, then the contribution packaging  
3855 can become dependent on the deployment environment. In order to avoid such a dependency,  
3856 dependent contributions should be specified only when deploying or updating contributions as  
3857 specified in the section 'Operations for Contributions' below.  
3858

### 3859 12.2.3 Contribution Packaging using ZIP

3860 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires  
3861 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This  
3862 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3863 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3864 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-  
3865 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such  
3866 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3867 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)  
3868 [.ZIP file format \[12\]](#).

3869

## 3870 12.3 Installed Contribution

3871 As noted in the section above, the contents of a contribution do not need to be modified in order  
3872 to install and use it within a domain. An *installed contribution* is a contribution with all of the  
3873 associated information necessary in order to execute *deployable composites* within the  
3874 contribution.

3875 An installed contribution is made up of the following things:

- 3876 • Contribution Packaging – the contribution that will be used as the starting point for  
3877 resolving all references
- 3878 • Contribution base URI
- 3879 • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3880 the import statements from the root composite and from other dependent contributions

- 3881                   o Dependent contributions might or might not be shared with other installed  
3882                   contributions.
- 3883                   o When the snapshot of any contribution is taken is implementation defined, ranging  
3884                   from the time the contribution is installed to the time of execution
- 3885                   • Deployment-time composites.  
3886                   These are composites that are added into an installed contribution after it has been  
3887                   deployed. This makes it possible to provide final configuration and access to  
3888                   implementations within a contribution without having to modify the contribution. These  
3889                   are optional, as composites that already exist within the contribution can also be used for  
3890                   deployment.

3891

3892                   Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3893                   fully qualified class names in Java).

3894                   If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3895                   algorithm used to determine the qualified name to use is implementation dependent.  
3896                   [Implementations of SCA MAY also report an error if there are conflicting names exported from](#)  
3897                   [multiple contributions.](#) [ASM12007]

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

3898

### 3899   12.3.1 Installed Artifact URIs

3900                   When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3901                   constructed by starting with the base URI of the contribution and adding the relative URI of each  
3902                   artifact (recalling that SCA requires that any packaging format be able to offer up its artifacts in a  
3903                   single hierarchy).

3904

## 3905   12.4 Operations for Contributions

3906                   SCA Domains provide the following conceptual functionality associated with contributions  
3907                   (meaning the function might not be represented as addressable services and also meaning that  
3908                   equivalent functionality might be provided in other ways). The functionality is optional meaning  
3909                   that some SCA runtimes MAY choose not to provide the contribution functions functionality in any  
3910                   way. [ASM12008]

### 3911   12.4.1 install Contribution & update Contribution

3912                   Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3913                   supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3914                   contributions that should be used to resolve the dependencies of the root contribution and other  
3915                   dependent contributions. These override any dependent contributions explicitly listed via the  
3916                   location attribute in the import statements of the contribution.

3917                   SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3918                   means that all other exported artifacts can be used from that contribution. Because of this, the  
3919                   dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3920                   what is being used from each one.

3921                   Each dependent contribution is also an installed contribution, with its own dependent  
3922                   contributions. By default these dependent contributions of the dependent contributions (which we  
3923                   will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3924                   contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3925                   definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3926                   included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3927                   Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3928                   MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3929 Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3930 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,  
3931 then it is easy for this list to be generated by tooling.

## 3932 12.4.2 add Deployment Composite & update Deployment Composite

3933 Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3934 data structure, not an existing resource in the domain) to the contribution identified by a supplied  
3935 contribution URI. The added or updated deployment composite is given a relative URI that  
3936 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3937 must run within the context of a installed contribution (any component implementations or other  
3938 definitions are resolved within that contribution), this functionality makes it possible for the  
3939 deployer to create a composite with final configuration and wiring decisions and add it to an  
3940 installed contribution without having to modify the contents of the root contribution.

3941 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3942 It is then possible for those to be given component names by a (possibly generated) composite  
3943 that is added into the installed contribution, without having to modify the packaging.

## 3944 12.4.3 remove Contribution

3945 Removes the deployed contribution identified by a supplied contribution URI.  
3946

## 3947 12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3948  
3949 For certain types of artifact, there are existing and commonly used mechanisms for referencing a  
3950 specific concrete location where the artifact can be resolved.

3951 Examples of these mechanisms include:

- 3952 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the  
3953 place holding the WSDL itself.
- 3954 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a  
3955 URI where the XSD is found.

3956 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does  
3957 not have to be dereferenced.

3958 SCA permits the use of these mechanisms. Where present, non-SCA artifact resolution  
3959 mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.  
3960 [ASM12010] However, use of these mechanisms is discouraged because tying assemblies to  
3961 addresses in this way makes the assemblies less flexible and prone to errors when changes are  
3962 made to the overall SCA Domain.

3963 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to  
3964 find the resource indicated when using the mechanism (eg the URI is incorrect or invalid, say) the  
3965 SCA runtime MUST report an error and MUST NOT attempt to use SCA resolution mechanisms as  
3966 an alternative. [ASM12011]

**Deleted:** 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.

## 3968 12.6 Domain-Level Composite

3969 The domain-level composite is a virtual composite, in that it is not defined by a composite  
3970 definition document. Rather, it is built up and modified through operations on the domain.  
3971 However, in other respects it is very much like a composite, since it contains components, wires,  
3972 services and references.

3973

3974 The value of @autowire for the logical domain composite MUST be autowire="false". [ASM12012]

3975

3976 For components at the Domain level, with References for which @autowire="true" applies, the  
3977 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

3978 1) The SCA runtime MAY disallow deployment of any components with autowire References. In  
3979 this case, the SCA runtime MUST report an exception at the point where the component is  
3980 deployed.

3981 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component  
3982 is deployed and not update those targets when later deployment actions occur.

3983 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later  
3984 deployment actions occur resulting in updated reference targets which match the new Domain  
3985 configuration. How the new configuration of the reference takes place is described by the relevant  
3986 client and implementation specifications.

3987 [ASM12013]

3988 The abstract domain-level functionality for modifying the domain-level composite is as follows,  
3989 although a runtime may supply equivalent functionality in a different form:

### 3990 12.6.1 add To Domain-Level Composite

3991 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.  
3992 The supplied composite URI must refer to a composite within a installed contribution. The  
3993 composite's installed contribution determines how the composite's artifacts are resolved (directly  
3994 and indirectly). The supplied composite is added to the domain composite with semantics that  
3995 correspond to the domain-level composite having an <include> statement that references the  
3996 supplied composite. All of the composite's components become *top-level* components and the  
3997 services become externally visible services (eg. they would be present in a WSDL description of  
3998 the domain).

### 3999 12.6.2 remove From Domain-Level Composite

4000 Removes from the Domain Level composite the elements corresponding to the composite  
4001 identified by a supplied composite URI. This means that the removal of the components, wires,  
4002 services and references originally added to the domain level composite by the identified  
4003 composite.

### 4004 12.6.3 get Domain-Level Composite

4005 Returns a <composite> definition that has an <include> line for each composite that had been  
4006 added to the domain level composite. It is important to note that, in dereferencing the included  
4007 composites, any referenced artifacts must be resolved in terms of that installed composite.

### 4008 12.6.4 get QName Definition

4009 In order to make sense of the domain-level composite (as returned by get Domain-Level  
4010 Composite), it must be possible to get the definitions for named artifacts in the included  
4011 composites. This functionality takes the supplied URI of an installed contribution (which provides  
4012 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
4013 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for  
4014 that definition type.

4015 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
4016 should exist in some form, but not necessarily as a service operation with exactly this signature.

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

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

## 4017 12.7 Dynamic Behaviour of Wires in the SCA Domain

4018 For components with references which are at the Domain level, there is the potential for dynamic  
4019 behaviour when the wires for a component reference change (this can only apply to component  
4020 references at the Domain level and not to components within composites used as implementations):

4021 The configuration of the wires for a component reference of a component at the Domain level can change  
4022 by means of deployment actions:

- 4023 1. <wire/> elements can be added, removed or replaced by deployment actions
- 4024 2. Components can be updated by deployment actions (ie this may change the component reference  
4025 configuration)
- 4026 3. Components which are the targets of reference wires can be updated or removed
- 4027 4. Components can be added that are potential targets for references which are marked with  
4028 @autowire=true

4029  
4030 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose  
4031 references are affected by those deployment actions MAY have their references updated by the SCA  
4032 runtime dynamically without the need to stop and start those components. [ASM12014]

4033 Where components are updated by deployment actions (their configuration is changed in some way,  
4034 which may include changing the wires of component references), the new configuration MUST apply to all  
4035 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY  
4036 choose to maintain existing instances with the old configuration of components updated by deployment  
4037 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.  
4038 [ASM12016]

4039 Where a component that is the target of a wire is removed, without the wire being changed, then future  
4040 invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is  
4041 the result of the autowire process, the SCA runtime MUST:

- 4042 • either cause future invocation of the target component's services to fail with a  
4043 ServiceUnavailable fault
- 4044 • or alternatively, if an alternative target component is available that satisfies the autowire  
4045 process, update the reference of the source component. [ASM12017]

4046 Where a component that is the target of a wire is updated, future invocations of that reference SHOULD  
4047 use the updated component. [ASM12018] Where an existing domain level component is updated, an  
4048 SCA runtime MAY maintain a copy of a component offering a conversational service until all existing  
4049 conversations complete - alternatively all existing conversations MAY be terminated. [ASM12019]

4050 Where a component is added to the domain that is a potential target for a domain level component  
4051 reference where that reference is marked as @autowire=true, the SCA runtime MUST:

- 4052 - either update the references for the source component once the new component is running.
- 4053 - or alternatively, defer the updating of the references of the source component until the source  
4054 component is stopped and restarted. [ASM12020]

## 4056 12.8 Dynamic Behaviour of Component Property Values

4057 For a domain level component with a Property whose value is obtained from a Domain-level Property  
4058 through the use of the @source attribute, if the domain level property is updated by means of deployment  
4059 actions, the SCA runtime MUST

- 4060 - either update the property value of the domain level component. once the update of the domain  
4061 property is complete
- 4062 - or alternative defer the updating of the component property value until the compoennt is stopped  
4063 and restarted

**Formatted:** Bullets and  
Numbering

**Deleted:** Where a component  
that is the target of a wire is  
removed, without the wire  
being changed, then future  
invocations of the reference  
that use that wire SHOULD fail  
with a ServiceUnavailable  
fault. If the wire is the result of  
the autowire process, the SCA  
runtime MUST:¶  
<#>either cause future  
invocation of the target  
component's services to fail  
with a ServiceUnavailable fault¶  
or alternatively, if an alternative  
target component is available  
that satisfies the autowire  
process, update the reference  
of the source component

**Formatted:** Bullets and  
Numbering

**Deleted:** Where a component  
is added to the domain that is  
a potential target for a domain  
level component reference  
where that reference is marked  
as @autowire=true, the SCA  
runtime MUST:¶  
<#>either update the  
references for the source  
component once the new  
component is running. ¶  
or alternatively, defer the  
updating of the references of  
the source component until the  
source component is stopped  
and restarted.



4065  
4066  
4067  
4068  
4069  
4070  
4071  
4072  
4073  
4074  
4075  
4076  
4077  
4078  
4079  
4080  
4081  
4082  
4083  
4084  
4085  
4086  
4087  
4088  
4089  
4090  
4091  
4092  
4093  
4094  
4095  
4096  
4097  
4098  
4099  
4100  
4101  
4102  
4103  
4104  
4105  
4106  
4107  
4108

## **13 SCA Runtime Considerations**

This section describes aspects of an SCA Runtime that are defined by this specification.

### **13.1 Error Handling**

The SCA Assembly specification identifies situations where the configuration of the SCA Domain and its contents are in error. When one of these situations occurs, the specification requires that the SCA Runtime that is interacting with the SCA Domain and the artifacts it contains should recognise that there is an error, report the error in a suitable manner and also refuse to run components and services that are in error.

The SCA Assembly specification is not prescriptive about the functionality of an SCA Runtime and the specification recognizes that there can be a range of design points for an SCA runtime. As a result, the SCA Assembly specification describes a range of error handling approaches which can be adopted by an SCA runtime.

#### **13.1.1 Errors which can be Detected at Deployment Time**

Some error situations can be detected at the point that artifacts are deployed to the Domain. An example is a composite document that is invalid in a way that can be detected by static analysis, such as containing a component with two services with the same @name attribute.

It is recommended that an SCA runtime SHOULD detect errors at deployment time where those errors can be found through static analysis. The SCA runtime either SHOULD prevent deployment of contributions that are in error, and report the error to the process performing the deployment OR SHOULD report the error to the process that is performing the deployment (eg write a message to an interactive console or write a message to a log file).

The SCA Assembly specification recognizes that there are reasons why a particular SCA runtime finds it desirable to deploy contributions that contain errors (eg to assist in the process of development and debugging) - and as a result also supports an error handling strategy that is based on detecting problems at runtime. However, it is wise to consider reporting problems at an early stage in the deployment process.

#### **13.1.2 Errors which are Detected at Runtime**

An SCA runtime can detect problems at runtime. These errors can include some which can be found from static analysis (eg the inability to wire a reference because the target service does not exist in the Domain) and others that can only be discovered dynamically (eg the inability to invoke some remote Web service because the remote endpoint is unavailable).

Where errors can be detected through static analysis, the principle is that components that are known to be in error SHOULD NOT run. So, for example, if there is a component with a required reference (multiplicity 1..1 or 1..n) which is not wired, the component SHOULD NOT be run. If an attempt is made to invoke a service operation of that component, a "ServiceUnavailable" fault SHOULD be reported to the invoker. Errors of this kind SHOULD also be reported through appropriate management interfaces, for example to the deployer or the operator of the system.

Where errors can only be detected at runtime, the components SHOULD all be run, but when the error is detected, a fault MUST be reported to the component that is attempting some activity, eg if a component invokes an operation on a reference, but the target service is unavailable, a "ServiceUnavailable" fault is reported to the component.

Formatted: Bullets and Numbering

Formatted: Normal

Formatted: Heading 2,H2

Formatted: Bullets and Numbering

Formatted: Normal

Formatted: Heading 3,H3

Formatted: Bullets and Numbering

Formatted: Normal

Formatted: Heading 3,H3

Formatted: Bullets and Numbering

Formatted: Normal

## 4109 **14** Conformance

4110 The XML schema available at the namespace URI, defined by this specification, is considered to be  
4111 authoritative and takes precedence over the XML Schema defined in the appendix of this document.

4112 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema..  
4113 [\[ASM13001\]](#)

4114

---

## 4115 A. XML Schemas

### 4116 A.1 sca.xsd

```
4117
4118 <?xml version="1.0" encoding="UTF-8"?>
4119 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4120 IPR and other policies apply. -->
4121 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4122   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4123   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">
4124
4125   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4126
4127   <include schemaLocation="sca-interface-java-1.1-schema-200803.xsd"/>
4128   <include schemaLocation="sca-interface-wsdl-1.1-schema-200803.xsd"/>
4129   <include schemaLocation="sca-interface-cpp-1.1-schema-200803.xsd"/>
4130   <include schemaLocation="sca-interface-c-1.1-schema-200803.xsd"/>
4131
4132   <include schemaLocation="sca-implementation-java-1.1-schema-200803.xsd"/>
4133   <include schemaLocation=
4134     "sca-implementation-composite-1.1-schema-200803.xsd"/>
4135   <include schemaLocation="sca-implementation-cpp-1.1-schema-200803.xsd"/>
4136   <include schemaLocation="sca-implementation-c-1.1-schema-200803.xsd"/>
4137   <include schemaLocation="sca-implementation-bpel-1.1-schema-200803.xsd"/>
4138
4139   <include schemaLocation="sca-binding-webservice-1.1-schema-200803.xsd"/>
4140   <include schemaLocation="sca-binding-jms-1.1-schema-200803.xsd"/>
4141   <include schemaLocation="sca-binding-sca-1.1-schema-200803.xsd"/>
4142
4143   <include schemaLocation="sca-definitions-1.1-schema-200803.xsd"/>
4144   <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4145
4146   <include schemaLocation="sca-contribution-1.1-schema-200803.xsd"/>
4147
4148 </schema>
```

### 4149 A.2 sca-core.xsd

```
4150
4151 <?xml version="1.0" encoding="UTF-8"?>
4152 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4153 IPR and other policies apply. -->
4154 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4155   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4156   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4157   elementFormDefault="qualified">
4158
4159   <import namespace="http://www.w3.org/XML/1998/namespace"
4160     schemaLocation="http://www.w3.org/2001/xml.xsd"/>
4161
4162   <!-- Common extension base for SCA definitions -->
4163   <complexType name="CommonExtensionBase">
4164     <sequence>
```

```

4165         <element ref="sca:documentation" minOccurs="0"
4166             maxOccurs="unbounded" />
4167     </sequence>
4168     <anyAttribute namespace="##other" processContents="lax" />
4169 </complexType>
4170
4171 <element name="documentation" type="sca:Documentation" />
4172 <complexType name="Documentation" mixed="true">
4173     <sequence>
4174         <any namespace="##other" processContents="lax" minOccurs="0"
4175             maxOccurs="unbounded" />
4176     </sequence>
4177     <attribute ref="xml:lang" />
4178 </complexType>
4179
4180 <!-- Component Type -->
4181 <element name="componentType" type="sca:ComponentType" />
4182 <complexType name="ComponentType">
4183     <complexContent>
4184         <extension base="sca:CommonExtensionBase">
4185             <sequence>
4186                 <element ref="sca:implementation" minOccurs="0" />
4187                 <choice minOccurs="0" maxOccurs="unbounded">
4188                     <element name="service" type="sca:ComponentService" />
4189                     <element name="reference"
4190                         type="sca:ComponentTypeReference" />
4191                     <element name="property" type="sca:Property" />
4192                 </choice>
4193                 <any namespace="##other" processContents="lax" minOccurs="0"
4194                     maxOccurs="unbounded" />
4195             </sequence>
4196             <attribute name="constrainingType" type="QName" use="optional" />
4197         </extension>
4198     </complexContent>
4199 </complexType>
4200
4201 <!-- Composite -->
4202 <element name="composite" type="sca:Composite" />
4203 <complexType name="Composite">
4204     <complexContent>
4205         <extension base="sca:CommonExtensionBase">
4206             <sequence>
4207                 <element name="include" type="anyURI" minOccurs="0"
4208                     maxOccurs="unbounded" />
4209                 <choice minOccurs="0" maxOccurs="unbounded">
4210                     <element name="service" type="sca:Service" />
4211                     <element name="property" type="sca:Property" />
4212                     <element name="component" type="sca:Component" />
4213                     <element name="reference" type="sca:Reference" />
4214                     <element name="wire" type="sca:Wire" />
4215                 </choice>
4216                 <any namespace="##other" processContents="lax" minOccurs="0"
4217                     maxOccurs="unbounded" />
4218             </sequence>
4219             <attribute name="name" type="NCName" use="required" />
4220             <attribute name="targetNamespace" type="anyURI" use="required" />
4221             <attribute name="local" type="boolean" use="optional"
4222                 default="false" />

```

```

4223         <attribute name="autowire" type="boolean" use="optional"
4224             default="false"/>
4225         <attribute name="constrainingType" type="QName" use="optional"/>
4226         <attribute name="requires" type="sca:listOfQNames"
4227             use="optional"/>
4228         <attribute name="policySets" type="sca:listOfQNames"
4229             use="optional"/>
4230     </extension>
4231 </complexContent>
4232 </complexType>
4233
4234 <!-- Contract base type for Service, Reference -->
4235 <complexType name="Contract" abstract="true">
4236     <complexContent>
4237         <extension base="sca:CommonExtensionBase">
4238             <sequence>
4239                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4240                 <element name="operation" type="sca:Operation" minOccurs="0"
4241                     maxOccurs="unbounded" />
4242                 <element ref="sca:binding" minOccurs="0"
4243                     maxOccurs="unbounded"/>
4244                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4245                 <any namespace="##other" processContents="lax" minOccurs="0"
4246                     maxOccurs="unbounded" />
4247             </sequence>
4248             <attribute name="name" type="NCName" use="required" />
4249             <attribute name="requires" type="sca:listOfQNames"
4250                 use="optional"/>
4251             <attribute name="policySets" type="sca:listOfQNames"
4252                 use="optional"/>
4253         </extension>
4254     </complexContent>
4255 </complexType>
4256
4257 <!-- Service -->
4258 <complexType name="Service">
4259     <complexContent>
4260         <extension base="sca:Contract">
4261             <attribute name="promote" type="anyURI" use="required"/>
4262         </extension>
4263     </complexContent>
4264 </complexType>
4265
4266 <!-- Interface -->
4267 <element name="interface" type="sca:Interface" abstract="true"/>
4268 <complexType name="Interface" abstract="true">
4269     <complexContent>
4270         <extension base="sca:CommonExtensionBase"/>
4271     </complexContent>
4272 </complexType>
4273
4274 <!-- Reference -->
4275 <complexType name="Reference">
4276     <complexContent>
4277         <extension base="sca:Contract">
4278             <attribute name="autowire" type="boolean" use="optional"/>
4279             <attribute name="target" type="sca:listOfAnyURIs"
4280                 use="optional"/>

```

```

4281         <attribute name="wiredByImpl" type="boolean" use="optional"
4282             default="false"/>
4283         <attribute name="multiplicity" type="sca:Multiplicity"
4284             use="optional" default="1..1"/>
4285         <attribute name="promote" type="sca:listOfAnyURIs"
4286             use="required"/>
4287     </extension>
4288 </complexContent>
4289 </complexType>
4290
4291 <!-- Property -->
4292 <complexType name="SCAPropertyBase" mixed="true">
4293     <sequence>
4294         <any namespace="##any" processContents="lax" minOccurs="0"/>
4295         <!-- NOT an extension point; This any exists to accept
4296             the element-based or complex type property
4297             i.e. no element-based extension point under "sca:property" -->
4298     </sequence>
4299     <!-- mixed="true" to handle simple type -->
4300     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4301     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4302 </complexType>
4303
4304 <complexType name="Property" mixed="true">
4305     <complexContent mixed="true">
4306         <extension base="sca:SCAPropertyBase">
4307             <attribute name="name" type="NCName" use="required"/>
4308             <attribute name="type" type="QName" use="optional"/>
4309             <attribute name="element" type="QName" use="optional"/>
4310             <attribute name="many" type="boolean" use="optional"
4311                 default="false"/>
4312             <attribute name="mustSupply" type="boolean" use="optional"
4313                 default="false"/>
4314             <anyAttribute namespace="##any" processContents="lax"/>
4315         </extension>
4316         <!-- extension defines the place to hold default value -->
4317         <!-- an extension point ; attribute-based only -->
4318     </complexContent>
4319 </complexType>
4320
4321 <complexType name="PropertyValue" mixed="true">
4322     <complexContent mixed="true">
4323         <extension base="sca:SCAPropertyBase">
4324             <attribute name="name" type="NCName" use="required"/>
4325             <attribute name="type" type="QName" use="optional"/>
4326             <attribute name="element" type="QName" use="optional"/>
4327             <attribute name="many" type="boolean" use="optional"
4328                 default="false"/>
4329             <attribute name="source" type="string" use="optional"/>
4330             <attribute name="file" type="anyURI" use="optional"/>
4331             <anyAttribute namespace="##any" processContents="lax"/>
4332         </extension>
4333         <!-- an extension point ; attribute-based only -->
4334     </complexContent>
4335 </complexType>
4336
4337 <!-- Binding -->
4338 <element name="binding" type="sca:Binding" abstract="true"/>

```

```

4339 <complexType name="Binding" abstract="true">
4340 <complexContent>
4341 <extension base="sca:CommonExtensionBase">
4342 <sequence>
4343 <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
4344 <element ref="sca:operationSelector"
4345 <minOccurs="0" maxOccurs="1" />
4346 <element name="operation" type="sca:Operation" minOccurs="0"
4347 <maxOccurs="unbounded" />
4348 </sequence>
4349 <attribute name="uri" type="anyURI" use="optional" />
4350 <attribute name="name" type="NCName" use="optional" />
4351 <attribute name="requires" type="sca:listOfQNames"
4352 <use="optional" />
4353 <attribute name="policySets" type="sca:listOfQNames"
4354 <use="optional" />
4355 </extension>
4356 </complexContent>
4357 </complexType>
4358
4359 <!-- Binding Type -->
4360 <element name="bindingType" type="sca:BindingType" />
4361 <complexType name="BindingType">
4362 <complexContent>
4363 <extension base="sca:CommonExtensionBase">
4364 <sequence>
4365 <any namespace="##other" processContents="lax" minOccurs="0"
4366 <maxOccurs="unbounded" />
4367 </sequence>
4368 <attribute name="type" type="QName" use="required" />
4369 <attribute name="alwaysProvides" type="sca:listOfQNames"
4370 <use="optional" />
4371 <attribute name="mayProvide" type="sca:listOfQNames"
4372 <use="optional" />
4373 </extension>
4374 </complexContent>
4375 </complexType>
4376
4377 <!-- WireFormat Type -->
4378 <element name="wireFormat" type="sca:WireFormatType" />
4379 <complexType name="WireFormatType" abstract="true">
4380 <sequence>
4381 <any namespace="##other" processContents="lax" minOccurs="0"
4382 <maxOccurs="unbounded" />
4383 </sequence>
4384 <anyAttribute namespace="##other" processContents="lax" />
4385 </complexType>
4386
4387 <!-- OperationSelector Type -->
4388 <element name="operationSelector" type="sca:OperationSelectorType" />
4389 <complexType name="OperationSelectorType" abstract="true">
4390 <sequence>
4391 <any namespace="##other" processContents="lax" minOccurs="0"
4392 <maxOccurs="unbounded" />
4393 </sequence>
4394 <anyAttribute namespace="##other" processContents="lax" />
4395 </complexType>
4396 <!-- Callback -->

```

```

4397 <element name="callback" type="sca:Callback"/>
4398 <complexType name="Callback">
4399   <complexContent>
4400     <extension base="sca:CommonExtensionBase">
4401       <choice minOccurs="0" maxOccurs="unbounded">
4402         <element ref="sca:binding"/>
4403         <any namespace="##other" processContents="lax"/>
4404       </choice>
4405       <attribute name="requires" type="sca:listOfQNames"
4406         use="optional"/>
4407       <attribute name="policySets" type="sca:listOfQNames"
4408         use="optional"/>
4409     </extension>
4410   </complexContent>
4411 </complexType>
4412
4413 <!-- Component -->
4414 <complexType name="Component">
4415   <complexContent>
4416     <extension base="sca:CommonExtensionBase">
4417       <sequence>
4418         <element ref="sca:implementation" minOccurs="0"/>
4419         <choice minOccurs="0" maxOccurs="unbounded">
4420           <element name="service" type="sca:ComponentService"/>
4421           <element name="reference" type="sca:ComponentReference"/>
4422           <element name="property" type="sca:PropertyValue"/>
4423         </choice>
4424         <any namespace="##other" processContents="lax" minOccurs="0"
4425           maxOccurs="unbounded"/>
4426       </sequence>
4427       <attribute name="name" type="NCName" use="required"/>
4428       <attribute name="autowire" type="boolean" use="optional"/>
4429       <attribute name="constrainingType" type="QName" use="optional"/>
4430       <attribute name="requires" type="sca:listOfQNames"
4431         use="optional"/>
4432       <attribute name="policySets" type="sca:listOfQNames"
4433         use="optional"/>
4434     </extension>
4435   </complexContent>
4436 </complexType>
4437
4438 <!-- Component Service -->
4439 <complexType name="ComponentService">
4440   <complexContent>
4441     <extension base="sca:Contract">
4442     </extension>
4443   </complexContent>
4444 </complexType>
4445
4446 <!-- Component Reference -->
4447 <complexType name="ComponentReference">
4448   <complexContent>
4449     <extension base="sca:Contract">
4450       <attribute name="autowire" type="boolean" use="optional"/>
4451       <attribute name="target" type="sca:listOfAnyURIs"
4452         use="optional"/>
4453       <attribute name="wiredByImpl" type="boolean" use="optional"
4454         default="false"/>

```

```

4455         <attribute name="multiplicity" type="sca:Multiplicity"
4456             use="optional" default="1..1"/>
4457     </extension>
4458 </complexContent>
4459 </complexType>
4460
4461 <!-- Component Type Reference -->
4462 <complexType name="ComponentTypeReference">
4463     <complexContent>
4464         <restriction base="sca:ComponentReference">
4465             <sequence>
4466                 <element ref="sca:documentation" minOccurs="0"
4467                     maxOccurs="unbounded"/>
4468                 <element ref="sca:interface" minOccurs="0"/>
4469                 <element name="operation" type="sca:Operation" minOccurs="0"
4470                     maxOccurs="unbounded"/>
4471                 <element ref="sca:binding" minOccurs="0"
4472                     maxOccurs="unbounded"/>
4473                 <element ref="sca:callback" minOccurs="0"/>
4474                 <any namespace="##other" processContents="lax" minOccurs="0"
4475                     maxOccurs="unbounded"/>
4476             </sequence>
4477             <attribute name="name" type="NCName" use="required"/>
4478             <attribute name="autowire" type="boolean" use="optional"/>
4479             <attribute name="wiredByImpl" type="boolean" use="optional"
4480                 default="false"/>
4481             <attribute name="multiplicity" type="sca:Multiplicity"
4482                 use="optional" default="1..1"/>
4483             <attribute name="requires" type="sca:listOfQNames"
4484                 use="optional"/>
4485             <attribute name="policySets" type="sca:listOfQNames"
4486                 use="optional"/>
4487             <anyAttribute namespace="##other" processContents="lax"/>
4488         </restriction>
4489     </complexContent>
4490 </complexType>
4491
4492 <!-- Implementation -->
4493 <element name="implementation" type="sca:Implementation" abstract="true"/>
4494 <complexType name="Implementation" abstract="true">
4495     <complexContent>
4496         <extension base="sca:CommonExtensionBase">
4497             <attribute name="requires" type="sca:listOfQNames"
4498                 use="optional"/>
4499             <attribute name="policySets" type="sca:listOfQNames"
4500                 use="optional"/>
4501         </extension>
4502     </complexContent>
4503 </complexType>
4504
4505 <!-- Implementation Type -->
4506 <element name="implementationType" type="sca:ImplementationType"/>
4507 <complexType name="ImplementationType">
4508     <complexContent>
4509         <extension base="sca:CommonExtensionBase">
4510             <sequence>
4511                 <any namespace="##other" processContents="lax" minOccurs="0"
4512                     maxOccurs="unbounded"/>

```

```

4513         </sequence>
4514         <attribute name="type" type="QName" use="required"/>
4515         <attribute name="alwaysProvides" type="sca:listOfQNames"
4516             use="optional"/>
4517         <attribute name="mayProvide" type="sca:listOfQNames"
4518             use="optional"/>
4519     </extension>
4520 </complexContent>
4521 </complexType>
4522
4523 <!-- Wire -->
4524 <complexType name="Wire">
4525     <complexContent>
4526         <extension base="sca:CommonExtensionBase">
4527             <sequence>
4528                 <any namespace="##other" processContents="lax" minOccurs="0"
4529                     maxOccurs="unbounded"/>
4530             </sequence>
4531             <attribute name="source" type="anyURI" use="required"/>
4532             <attribute name="target" type="anyURI" use="required"/>
4533         </extension>
4534     </complexContent>
4535 </complexType>
4536
4537 <!-- Include -->
4538 <element name="include" type="sca:Include"/>
4539 <complexType name="Include">
4540     <complexContent>
4541         <extension base="sca:CommonExtensionBase">
4542             <attribute name="name" type="QName"/>
4543         </extension>
4544     </complexContent>
4545 </complexType>
4546
4547 <!-- Operation -->
4548 <complexType name="Operation">
4549     <complexContent>
4550         <extension base="sca:CommonExtensionBase">
4551             <attribute name="name" type="NCName" use="required"/>
4552             <attribute name="requires" type="sca:listOfQNames"
4553                 use="optional"/>
4554             <attribute name="policySets" type="sca:listOfQNames"
4555                 use="optional"/>
4556         </extension>
4557     </complexContent>
4558 </complexType>
4559
4560 <!-- Constraining Type -->
4561 <element name="constrainingType" type="sca:ConstrainingType"/>
4562 <complexType name="ConstrainingType">
4563     <complexContent>
4564         <extension base="sca:CommonExtensionBase">
4565             <sequence>
4566                 <choice minOccurs="0" maxOccurs="unbounded">
4567                     <element name="service" type="sca:ComponentService"/>
4568                     <element name="reference" type="sca:ComponentReference"/>
4569                     <element name="property" type="sca:Property"/>
4570                 </choice>

```

```

4571         <any namespace="##other" processContents="lax" minOccurs="0"
4572             maxOccurs="unbounded" />
4573     </sequence>
4574     <attribute name="name" type="NCName" use="required" />
4575     <attribute name="targetNamespace" type="anyURI" />
4576     <attribute name="requires" type="sca:listOfQNames"
4577         use="optional" />
4578 </extension>
4579 </complexContent>
4580 </complexType>
4581
4582 <!-- Intents within WSDL documents -->
4583 <attribute name="requires" type="sca:listOfQNames" />
4584
4585 <!-- Marker for operations ending a conversation -->
4586 <attribute name="endsConversation" type="boolean" default="false" />
4587
4588 <!-- Global attribute definition for @callback to mark a WSDL port type
4589     as having a callback interface defined in terms of a second port
4590     type. -->
4591 <attribute name="callback" type="anyURI" />
4592
4593 <!-- Miscellaneous simple type definitions -->
4594 <simpleType name="Multiplicity">
4595     <restriction base="string">
4596         <enumeration value="0..1" />
4597         <enumeration value="1..1" />
4598         <enumeration value="0..n" />
4599         <enumeration value="1..n" />
4600     </restriction>
4601 </simpleType>
4602
4603 <simpleType name="OverrideOptions">
4604     <restriction base="string">
4605         <enumeration value="no" />
4606         <enumeration value="may" />
4607         <enumeration value="must" />
4608     </restriction>
4609 </simpleType>
4610
4611 <simpleType name="listOfQNames">
4612     <list itemType="QName" />
4613 </simpleType>
4614
4615 <simpleType name="listOfAnyURIs">
4616     <list itemType="anyURI" />
4617 </simpleType>
4618
4619 </schema>
4620

```

### 4621 **A.3 sca-binding-sca.xsd**

```

4622
4623 <?xml version="1.0" encoding="UTF-8"?>
4624 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4625 IPR and other policies apply. -->
4626 <schema xmlns="http://www.w3.org/2001/XMLSchema"

```

```

4627     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4628     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4629     elementFormDefault="qualified">
4630
4631     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4632
4633     <!-- SCA Binding -->
4634     <element name="binding.sca" type="sca:SCABinding"
4635         substitutionGroup="sca:binding"/>
4636     <complexType name="SCABinding">
4637         <complexContent>
4638             <extension base="sca:Binding"/>
4639         </complexContent>
4640     </complexType>
4641 </schema>
4642
4643

```

#### 4644 **A.4 sca-interface-java.xsd**

```

4645
4646 <?xml version="1.0" encoding="UTF-8"?>
4647 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4648 IPR and other policies apply. -->
4649 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4650     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4651     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4652     elementFormDefault="qualified">
4653
4654     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4655
4656     <!-- Java Interface -->
4657     <element name="interface.java" type="sca:JavaInterface"
4658         substitutionGroup="sca:interface"/>
4659     <complexType name="JavaInterface">
4660         <complexContent>
4661             <extension base="sca:Interface">
4662                 <sequence>
4663                     <any namespace="##other" processContents="lax" minOccurs="0"
4664                         maxOccurs="unbounded"/>
4665                 </sequence>
4666                 <attribute name="interface" type="NCName" use="required"/>
4667                 <attribute name="callbackInterface" type="NCName"
4668                     use="optional"/>
4669                 <anyAttribute namespace="##any" processContents="lax"/>
4670             </extension>
4671         </complexContent>
4672     </complexType>
4673 </schema>
4674
4675
4676

```

#### 4677 **A.5 sca-interface-wsdl.xsd**

4678

```

4679 <?xml version="1.0" encoding="UTF-8"?>
4680 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4681 IPR and other policies apply. -->
4682 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4683   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4684   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4685   elementFormDefault="qualified">
4686
4687   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4688
4689   <!-- WSDL Interface -->
4690   <element name="interface.wsdl" type="sca:WSDLPortType"
4691     substitutionGroup="sca:interface"/>
4692   <complexType name="WSDLPortType">
4693     <complexContent>
4694       <extension base="sca:Interface">
4695         <sequence>
4696           <any namespace="##other" processContents="lax" minOccurs="0"
4697             maxOccurs="unbounded"/>
4698         </sequence>
4699         <attribute name="interface" type="anyURI" use="required"/>
4700         <attribute name="callbackInterface" type="anyURI"
4701           use="optional"/>
4702         <anyAttribute namespace="##any" processContents="lax"/>
4703       </extension>
4704     </complexContent>
4705   </complexType>
4706 </schema>
4707
4708
4709

```

## 4710 A.6 sca-implementation-java.xsd

```

4711
4712 <?xml version="1.0" encoding="UTF-8"?>
4713 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4714 IPR and other policies apply. -->
4715 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4716   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4717   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4718   elementFormDefault="qualified">
4719
4720   <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4721
4722   <!-- Java Implementation -->
4723   <element name="implementation.java" type="sca:JavaImplementation"
4724     substitutionGroup="sca:implementation"/>
4725   <complexType name="JavaImplementation">
4726     <complexContent>
4727       <extension base="sca:Implementation">
4728         <sequence>
4729           <any namespace="##other" processContents="lax" minOccurs="0"
4730             maxOccurs="unbounded"/>
4731         </sequence>
4732         <attribute name="class" type="NCName" use="required"/>
4733         <anyAttribute namespace="##any" processContents="lax"/>

```

```
4734         </extension>
4735     </complexContent>
4736 </complexType>
4737
4738 </schema>
```

## 4739 **A.7 sca-implementation-composite.xsd**

```
4740
4741 <?xml version="1.0" encoding="UTF-8"?>
4742 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4743 IPR and other policies apply. -->
4744 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4745 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4746 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4747 elementFormDefault="qualified">
4748
4749     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4750
4751     <!-- Composite Implementation -->
4752     <element name="implementation.composite" type="sca:SCAImplementation"
4753 substitutionGroup="sca:implementation"/>
4754     <complexType name="SCAImplementation">
4755         <complexContent>
4756             <extension base="sca:Implementation">
4757                 <sequence>
4758                     <any namespace="##other" processContents="lax" minOccurs="0"
4759 maxOccurs="unbounded"/>
4760                 </sequence>
4761                 <attribute name="name" type="QName" use="required"/>
4762             </extension>
4763         </complexContent>
4764     </complexType>
4765
4766 </schema>
4767
```

## 4768 **A.8 sca-definitions.xsd**

```
4769
4770 <?xml version="1.0" encoding="UTF-8"?>
4771 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4772 IPR and other policies apply. -->
4773 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4774 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4775 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4776 elementFormDefault="qualified">
4777
4778     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4779     <include schemaLocation="sca-policy-1.1-schema-200803.xsd"/>
4780
4781     <!-- Definitions -->
4782     <element name="definitions" type="sca:tDefinitions"/>
4783     <complexType name="tDefinitions">
4784         <complexContent>
4785             <extension base="sca:CommonExtensionBase">
4786                 <choice minOccurs="0" maxOccurs="unbounded">
```

```

4787         <element ref="sca:intent"/>
4788     <element ref="sca:policySet"/>
4789     <element ref="sca:binding"/>
4790     <element ref="sca:bindingType"/>
4791     <element ref="sca:implementationType"/>
4792     <any namespace="##other" processContents="lax" minOccurs="0"
4793         maxOccurs="unbounded"/>
4794     </choice>
4795 </extension>
4796 </complexContent>
4797 </complexType>
4798
4799 </schema>
4800
4801

```

## 4802 **A.9 sca-binding-webservice.xsd**

4803 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4804 **A.10 sca-binding-jms.xsd**

4805 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4806 **A.11 sca-policy.xsd**

4807 Is described in [the SCA Policy Framework specification \[10\]](#)

4808

## 4809 **A.12 sca-contribution.xsd**

4810

```

4811 <?xml version="1.0" encoding="UTF-8"?>
4812 <!-- Copyright(C) OASIS(R) 2005,2008. All Rights Reserved. OASIS trademark,
4813 IPR and other policies apply. -->
4814 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4815     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4816     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4817     elementFormDefault="qualified">
4818
4819     <include schemaLocation="sca-core-1.1-schema-200803.xsd"/>
4820
4821     <!-- Contribution -->
4822     <element name="contribution" type="sca:ContributionType"/>
4823     <complexType name="ContributionType">
4824         <complexContent>
4825             <extension base="sca:CommonExtensionBase">
4826                 <sequence>
4827                     <element name="deployable" type="sca:DeployableType"
4828                         maxOccurs="unbounded"/>
4829                     <element name="import" type="sca:ImportType" minOccurs="0"
4830                         maxOccurs="unbounded"/>
4831                     <element name="export" type="sca:ExportType" minOccurs="0"
4832                         maxOccurs="unbounded"/>
4833                     <any namespace="##other" processContents="lax" minOccurs="0"
4834                         maxOccurs="unbounded"/>
4835                 </sequence>

```

```

4836         </extension>
4837     </complexContent>
4838 </complexType>
4839
4840 <!-- Deployable -->
4841 <complexType name="DeployableType">
4842     <complexContent>
4843         <extension base="sca:CommonExtensionBase">
4844             <sequence>
4845                 <any namespace="##other" processContents="lax" minOccurs="0"
4846                     maxOccurs="unbounded" />
4847             </sequence>
4848             <attribute name="composite" type="QName" use="required" />
4849         </extension>
4850     </complexContent>
4851 </complexType>
4852
4853 <!-- Import -->
4854 <element name="importBase" type="sca:Import" abstract="true" />
4855 <complexType name="Import" abstract="true">
4856     <complexContent>
4857         <extension base="sca:CommonExtensionBase">
4858             <sequence>
4859                 <any namespace="##other" processContents="lax" minOccurs="0"
4860                     maxOccurs="unbounded" />
4861             </sequence>
4862         </extension>
4863     </complexContent>
4864 </complexType>
4865
4866 <element name="import" type="sca:ImportType" />
4867 <complexType name="ImportType">
4868     <complexContent>
4869         <extension base="sca:Import">
4870             <attribute name="namespace" type="string" use="required" />
4871             <attribute name="location" type="anyURI" use="optional" />
4872         </extension>
4873     </complexContent>
4874 </complexType>
4875
4876 <!-- Export -->
4877 <element name="exportBase" type="sca:Export" abstract="true" />
4878 <complexType name="Export" abstract="true">
4879     <complexContent>
4880         <extension base="sca:CommonExtensionBase">
4881             <sequence>
4882                 <any namespace="##other" processContents="lax" minOccurs="0"
4883                     maxOccurs="unbounded" />
4884             </sequence>
4885         </extension>
4886     </complexContent>
4887 </complexType>
4888
4889 <element name="export" type="sca:ExportType" />
4890 <complexType name="ExportType">
4891     <complexContent>
4892         <extension base="sca:Export">
4893             <attribute name="namespace" type="string" use="required" />

```

```
4894         </extension>
4895     </complexContent>
4896 </complexType>
4897
4898 </schema>
4899
4900
```

---

## 4901 B. SCA Concepts

### 4902 B.1 Binding

4903 **Bindings** are used by services and references. References use bindings to describe the access  
4904 mechanism used to call the service to which they are wired. Services use bindings to describe the  
4905 access mechanism(s) that clients should use to call the service.

4906 SCA supports multiple different types of bindings. Examples include **SCA service, Web service,**  
4907 **stateless session EJB, data base stored procedure, EIS service.** SCA provides an extensibility  
4908 mechanism by which an SCA runtime can add support for additional binding types.

4909

### 4910 B.2 Component

4911 **SCA components** are configured instances of **SCA implementations**, which provide and consume  
4912 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines  
4913 an **extensibility mechanism** that allows you to introduce new implementation types. The current  
4914 specification does not mandate the implementation technologies to be supported by an SCA run-time,  
4915 vendors may choose to support the ones that are important for them. A single SCA implementation may  
4916 be used by multiple Components, each with a different configuration.

4917 The Component has a reference to an implementation of which it is an instance, a set of property values,  
4918 and a set of service reference values. Property values define the values of the properties of the  
4919 component as defined by the component's implementation. Reference values define the services that  
4920 resolve the references of the component as defined by its implementation. These values can either be a  
4921 particular service of a particular component, or a reference of the containing composite.

### 4922 B.3 Service

4923 **SCA services** are used to declare the externally accessible services of an **implementation**. For a  
4924 composite, a service is typically provided by a service of a component within the composite, or by a  
4925 reference defined by the composite. The latter case allows the republication of a service with a new  
4926 address and/or new bindings. The service can be thought of as a point at which messages from external  
4927 clients enter a composite or implementation.

4928 A service represents an addressable set of operations of an implementation that are designed to be  
4929 exposed for use by other implementations or exposed publicly for use elsewhere (eg public Web services  
4930 for use by other organizations). The operations provided by a service are specified by an Interface, as  
4931 are the operations required by the service client (if there is one). An implementation may contain  
4932 multiple services, when it is possible to address the services of the implementation separately.

4933 A service may be provided **as SCA remote services, as Web services, as stateless session EJB's, as**  
4934 **EIS services, and so on.** Services use **bindings** to describe the way in which they are published. SCA  
4935 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new  
4936 types of services.

#### 4937 B.3.1 Remotable Service

4938 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA  
4939 architecture. For example, SCA services of SCA implementations can define implementations of industry-  
4940 standard web services. Remotable services use pass-by-value semantics for parameters and returned  
4941 results.

4942 How a Service is identified as remotable is dependant on the Component implementation technology  
4943 used. See the relevant SCA Implementation Specification for more information. As an example, to define  
4944 a Remotable Service, a Component implemented in Java would have a Java Interface with the  
4945 @Remotable annotation

### 4946 **B.3.2 Local Service**

4947 Local services are services that are designed to be only used “locally” by other implementations that are  
4948 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4949 Local services may rely on by-reference calling conventions, or may assume a very fine-grained  
4950 interaction style that is incompatible with remote distribution. They may also use technology-specific data-  
4951 types.

4952 How a Service is identified as local is dependant on the Component implementation technology used.  
4953 See the relevant SCA Implementation Specification for more information. As an example, to define a  
4954 Local Service, a Component implemented in Java would define a Java Interface that does not have the  
4955 @Remotable annotation.

4956

### 4957 **B.4 Reference**

4958 **SCA references** represent a dependency that an implementation has on a service that is supplied by  
4959 some other implementation, where the service to be used is specified through configuration. In other  
4960 words, a reference is a service that an implementation may call during the execution of its business  
4961 function. References are typed by an interface.

4962 For composites, composite references can be accessed by components within the composite like any  
4963 service provided by a component within the composite. Composite references can be used as the targets  
4964 of wires from component references when configuring Components.

4965 A composite reference can be used to access a service such as: an SCA service provided by another  
4966 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,  
4967 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4968 an **extensibility mechanism** that allows the introduction of new binding types to references.

4969

### 4970 **B.5 Implementation**

4971 An implementation is concept that is used to describe a piece of software technology such as a Java  
4972 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4973 service-oriented application. An SCA composite is also an implementation.

4974 Implementations define points of variability including properties that can be set and settable references to  
4975 other services. The points of variability are configured by a component that uses the implementation. The  
4976 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4977 **B.6 Interface**

4978 **Interfaces** define one or more business functions. These business functions are provided by Services  
4979 and are used by components through References. Services are defined by the Interface they implement.  
4980 SCA currently supports a number of interface type systems, for example:

- 4981 • Java interfaces
- 4982 • WSDL portTypes
- 4983 • C, C++ header files

4984

4985 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4986 interface type systems.

4987 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided  
4988 by each end of a service communication – this could be the case where a particular service requires a  
4989 “callback” interface on the client, which is calls during the process of handing service requests from the  
4990 client.

4991

## 4992 **B.7 Composite**

4993 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4994 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4995 be used to contribute elements to an **SCA Domain**.

4996 A **composite** has the following characteristics:

4997 • It may be used as a component implementation. When used in this way, it defines a boundary for  
4998 Component visibility. Components may not be directly referenced from outside of the composite  
4999 in which they are declared.

5000 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
5001 artifacts to an SCA domain.

5002

## 5003 **B.8 Composite inclusion**

5004 One composite can be used to provide part of the definition of another composite, through the process of  
5005 inclusion. This is intended to make team development of large composites easier. Included composites  
5006 are merged together into the using composite at deployment time to form a single logical composite.

5007 Composites are included into other composites through <include.../> elements in the using composite.

5008 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
5009 specific location.

5010

## 5011 **B.9 Property**

5012 **Properties** allow for the configuration of an implementation with externally set data values. The data  
5013 value is provided through a Component, possibly sourced from the property of a containing composite.

5014 Each Property is defined by the implementation. Properties may be defined directly through the  
5015 implementation language or through annotations of implementations, where the implementation language  
5016 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
5017 type. For complex data types, XML schema is the preferred technology for defining the data types.

5018

## 5019 **B.10 Domain**

5020 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
5021 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
5022 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
5023 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

5024 A domain specifies the instantiation, configuration and connection of a set of components, provided via  
5025 one or more composite files. The domain, like a composite, also has Services and References. Domains  
5026 also contain Wires which connect together the Components, Services and References.

5027

## 5028 **B.11 Wire**

5029 **SCA wires** connect **service references** to **services**.

5030 Valid wire sources are component references. Valid wire targets are component services.

5031 When using included composites, the sources and targets of the wires don't have to be declared in the  
5032 same composite as the composite that contains the wire. The sources and targets can be defined by  
5033 other included composites. Targets can also be external to the SCA domain.



5035

## C. Conformance Items

5036 This section contains a list of conformance items for the SCA Assembly specification.

5037

Conformance ID	Description
[ASM13001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.
[ASM40001]	The extension of a componentType side file name MUST be .componentType.
[ASM40002]	If present, the @constrainingType attribute of a <componentType/> element MUST reference a <constrainingType/> element in the Domain through its QName.
<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.
[ASM50001]	The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/>
[ASM50002]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50003]	The @name attribute of a service element of a <component/> MUST match the @name attribute of a service element of the componentType of the <implementation/> child element of the component.
<del>[ASM50004]</del>	If a <service/> element has an interface subelement specified, the interface MUST provide a compatible subset of the interface declared on the componentType of the implementation
<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

Deleted: [ASM40003]

Deleted: [ASM40004]

Deleted: [ASM50004]

Deleted: [ASM50005]

Deleted: [ASM50006]

<implementation/> child element of the component.

[ASM50009]

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

[ASM50010]

If @wiredByImpl="true" is set for a reference, then the reference MUST NOT be wired statically within a composite, but left unwired.

[ASM50011]

If an interface is declared for a component reference it MUST provide a compatible superset of the interface declared for the equivalent reference in the componentType of the implementation, i.e. provide the same operations or a superset of the operations defined by the implementation for the reference.

Deleted: [ASM50011]

[ASM50012]

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

Deleted: [ASM50012]

[ASM50013]

If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.

[ASM50014]

If @autowire="true", the autowire procedure MUST only be used if no target is identified by any of the other ways listed above. It is not an error if @autowire="true" and a target is also defined through some other means, however in this case the autowire procedure MUST NOT be used.

[ASM50015]

If a binding element has a value specified for a target service using its @uri attribute, the binding element MUST NOT identify target services using binding specific attributes or elements.

[ASM50016]

It is possible that a particular binding type MAY require that the address of a target service uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to identify the target service - instead, binding specific attributes and/or child elements must be used.

[ASM50018]

A reference with multiplicity 0..1 or 0..n MAY have no target service defined.

[ASM50019]

A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service defined.

[ASM50020]

A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.

[ASM50021]

A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.

[ASM50022]

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

Deleted: generate

[ASM50023]

Some reference multiplicity errors can be detected at deployment time. In these cases, an error SHOULD be reported by the SCA runtime at deployment time.

Deleted: generated

[ASM50024]

Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA runtime MUST report an error no later than when the reference is invoked by the component implementation.

Deleted: generate

[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] If any <wire/> element with its @replace attribute set to "true" has a particular reference specified in its @source attribute, the value of the @target attribute for that reference MUST be ignored and MUST NOT be used to define target services for that reference.
- [ASM60001] A composite name must be unique within the namespace of the composite.
- [ASM60002] @local="true" for a composite means that all the components within the composite MUST run in the same operating system process.
- [ASM60003] The name of a composite <service/> element MUST be unique across all the composite services in the composite.
- [ASM60004] A composite <service/> element's promote attribute MUST identify one of the component services within that composite.
- [ASM60005] If a composite service **interface** is specified it must be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service. Deleted: [ASM60005]
- [ASM60006] The name of a composite <reference/> element MUST be unique across all the composite references in the composite.
- [ASM60007] Each of the URIs declared by a composite reference's @promote attribute MUST identify a component reference within the composite.
- [ASM60008] the interfaces of the component references promoted by a composite reference MUST be the same, or if the composite reference itself declares an interface then all the component reference interfaces must be compatible with the composite reference interface. Compatible means that the component reference interface is the same or is a strict subset of the composite reference interface.
- [ASM60009] the intents declared on a composite reference and on the component references which it promotes MUST NOT be mutually exclusive.
- [ASM60010] If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST report an error. Deleted: raise
- [ASM60011] The value specified for the **multiplicity** attribute of a composite reference MUST be compatible with the multiplicity specified on each of the promoted component references, i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be used where the promoted component reference has multiplicity 0..n or 1..n and multiplicity 1..n can be used where the promoted component reference has multiplicity 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to promote a component reference of multiplicity 0..1 or 1..1 respectively.
- [ASM60012] If a composite reference has an **interface** specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared Deleted: [ASM60012]

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

[ASM60013]

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

[ASM60014]

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

[ASM60015]

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

[ASM60016]

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

[ASM60017]

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

[ASM60018]

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

[ASM60019]

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

[ASM60020]

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

[ASM60021]

For the case of an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA runtime SHOULD issue a warning.

[ASM60022]

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

Deleted: [ASM60022]

[ASM60023]

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

[ASM60024]

the intents, and policies applied to the service MUST be compatible with those on the reference when using autowire to wire a reference – so that wiring the reference to the service will not cause an error due to policy mismatch

[ASM60025]

for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the reference to one of the set of valid target services chosen from the set in a runtime-dependent fashion

[ASM60026]

for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all of the set of valid target services

[ASM60027]

for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT **report** an error

Deleted: raise

[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 **reported** by the SCA runtime since the reference is intended to be wired

Deleted: raise

[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 **report** an error if the composite resulting from the inclusion of one composite into another is invalid.

Deleted: raise

[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

Deleted: [ASM60033]

(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.

Deleted: [ASM60034]

[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 **report** an error.

Deleted: raise

[ASM70003]

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

[ASM70004]

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

[ASM70005]

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

[ASM70006]

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

Deleted: [ASM70006]

[ASM70007]

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

Deleted: [ASM70007]

[ASM80001]

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

[ASM80002]

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

[ASM80003]

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

[ASM80004]

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

[ASM80005]

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

[ASM80006]

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

[ASM80007]

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

Deleted: generate

[ASM80008]

Any service or reference that uses an interface marked with required intents MUST implicitly add those intents to its own @requires list.

[ASM80009]

In a bidirectional interface, the service interface can have more than one operation defined, and the callback interface can also have more than one operation defined. SCA runtimes MUST allow an invocation of any operation on the service interface to be followed by zero, one or many invocations of any of the operations on the callback interface.

[ASM80010]

Whenever an interface document declaring a callback interface is used in the

declaration of an <interface/> element in SCA, it MUST be treated as being bidirectional with the declared callback interface.

[ASM80011]

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

[ASM80012]

Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface.

[ASM80013]

If the service or reference declaration in the component type declares an interface without a callback interface, then the component configuration for the corresponding service or reference MUST NOT declare an interface with a callback interface.

[ASM80014]

Where a composite declares an interface for a composite service or a composite reference, if the promoted service or promoted reference has an interface which has a callback interface, then the interface declaration for the composite service or the composite reference MUST also declare a compatible interface with a compatible callback interface.

[ASM80015]

If the promoted service or promoted reference has an interface without a callback interface, then the interface declaration for the composite service or composite reference MUST NOT declare a callback interface.

[ASM80016]

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

Deleted: [ASM80016]

[ASM90001]

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

[ASM90002]

When a service or reference has multiple bindings, only one binding can have the default name value; all others must have a name value specified that is unique within the service or reference.

[ASM90003]

If a reference has any bindings they MUST be resolved which means that each binding MUST include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired using other SCA mechanisms.

[ASM90004]

a wire target MAY be specified with a syntax of "componentName/serviceName/bindingName".

Deleted: [ASM90004]

[ASM10001]

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

[ASM12001]

For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA as a hierarchy of resources based off of a single root

[ASM12002]

Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy named META-INF

[ASM12003]

Within any contribution packaging a document SHOULD exist directly under the META-INF directory named sca-contribution.xml which lists the SCA Composites within the contribution that are runnable.

[ASM12004]

Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the namespaces of constructs that are needed by the contribution and which are be found elsewhere, for example in other contributions.

[ASM12005]

Where present, these mechanisms MUST be used by the SCA runtime to resolve artifact dependencies.

[ASM12006]

SCA requires that all runtimes MUST support the ZIP packaging format for contributions.

Deleted: [ASM12006]

[ASM12007]

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

Deleted: generate

[ASM12008]

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

[ASM12009]

if there is ever a conflict between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in the dependent contribution list.

[ASM12010]

Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.

Deleted: [ASM12010]

[ASM12011]

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

Deleted: [ASM12011]

Deleted: raise

[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 **report** an exception at the point where the component is deployed.

Deleted: generate

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

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

[ASM12014]

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

[ASM12015]

Where components are updated by deployment actions (their configuration is changed in some way, which may include changing the wires of component references), the new configuration MUST apply to all new instances of those components once the update is complete.

Deleted: [ASM12015]

[ASM12016]

An SCA runtime MAY choose to maintain existing instances with the old configuration of components updated by deployment actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.

Deleted: [ASM12016]

[ASM12017]

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

Deleted: [ASM12017]

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

[ASM12018]

Where a component that is the target of a wire is updated, future invocations of that reference SHOULD use the updated component.

Deleted: [ASM12018]

[ASM12019]

Where an existing domain level component is updated, an SCA runtime MAY maintain a copy of a component offering a conversational service until all existing conversations complete - alternatively all existing conversations MAY be terminated.

Deleted: [ASM12019]

[ASM12020]

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

Deleted: [ASM12020]

- either update the references for the source component once the new component is running.

or alternatively, defer the updating of the references of the source component until

the source component is stopped and restarted.

[ASM12021]

The SCA runtime MUST **report** an error if an artifact cannot be resolved using these mechanisms, if present.

Deleted: raise

[ASM12022]

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

[ASM12023]

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

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

2. from the contents of the contribution itself.

[ASM12024]

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

[ASM12025]

The SCA runtime MUST **report** an error if an artifact cannot be resolved by the precedence order above.

Deleted: raise

5039

---

## D. Acknowledgements

5040 The following individuals have participated in the creation of this specification and are gratefully  
5041 acknowledged:

5042 **Participants:**

5043 [Participant Name, Affiliation | Individual Member]

5044 [Participant Name, Affiliation | Individual Member]

5045

---

## **E. Non-Normative Text**

5047

## F. Revision History

5048 [optional; should not be included in OASIS Standards]

5049

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>
11 CD01-Rev7	2008-12-23	Mike Edwards	<p>All changes accepted before revision from Rev6 started - due to changes being applied to previously changed sections in the Schemas</p> <ul style="list-style-type: none"> <li>Issues 12 &amp; 18 - Section B2</li> <li>Issue 63 - Section C3</li> <li>Issue 75 - Section C12</li> <li>Issue 65 - Section 7.0</li> <li>Issue 77 - Section 8 + Appendix D</li> <li>Issue 69 - Sections 5.1, 8</li> <li>Issue 45 - Sections 4.1.3, 5.4, 6.3, B2.</li> <li>Issue 56 - Section 8.2, Appendix D</li> <li>Issue 41 - Sections 5.3.1, 6.4, 12.7, 12.8, Appendix D</li> </ul>
12 CD01-Rev8	2008-12-30	Mike Edwards	<ul style="list-style-type: none"> <li>Issue 72 - Removed Appendix A</li> <li>Issue 79 - Sections 9.0, 9.2, 9.3, Appendix A.2</li> <li>Issue 62 - Sections 4.1.3, 5.4</li> <li>Issue 26 - Section 6.5</li> <li>Issue 51 - Section 6.5</li> <li>Issue 36 - Section 4.1</li> <li>Issue 44 - Section 10, Appendix C</li> <li>Issue 89 - Section 8.2, 8.5, Appendix A, Appendix C</li> <li>Issue 16 - Section 6.8, 9.4</li> <li>Issue 8 - Section 11.2.1</li> <li>Issue 17 - Section 6.6</li> <li>Issue 30 - Sections 4.1.1, 4.1.2, 5.2, 5.3, 6.1, 6.2, 9</li> <li>Issue 33 - insert new Section 8.4</li> </ul>
12 CD01-Rev8a	2009-01-13	Bryan Aupperle Mike Edwards	Issue 99 - Section 8

13 CD02	2009-01-14	Mike Edwards	All changes accepted All comments removed
---------	------------	--------------	--

5050