



# Service Component Architecture Assembly Model Specification Version 1.1

**Committee Draft 03 / Public Review Draft 01**

**10 March 2009**

**Specification URIs:**

**This Version:**

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.html>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.doc>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd03.pdf> (Authoritative)

**Previous 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)

**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, Active Endpoints

**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/200903>

**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, 2009. All Rights Reserved.

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

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

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

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

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

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

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

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

---

# Table of Contents

1	Introduction.....	7
1.1	Terminology.....	7
1.2	Normative References.....	7
1.3	Naming Conventions.....	8
2	Overview.....	10
2.1	Diagram used to Represent SCA Artifacts.....	11
3	Implementation and ComponentType.....	13
3.1	Component Type.....	13
3.1.1	Service.....	14
3.1.2	Reference.....	15
3.1.3	Property.....	17
3.1.4	Implementation.....	18
3.2	Example ComponentType.....	19
3.3	Example Implementation.....	19
4	Component.....	22
4.1	Implementation.....	23
4.2	Service.....	24
4.3	Reference.....	25
4.3.1	Specifying the Target Service(s) for a Reference.....	28
4.4	Property.....	29
4.5	Example Component.....	32
5	Composite.....	35
5.1	Service.....	37
5.1.1	Service Examples.....	38
5.2	Reference.....	39
5.2.1	Example Reference.....	41
5.3	Property.....	43
5.3.1	Property Examples.....	44
5.4	Wire.....	46
5.4.1	Wire Examples.....	48
5.4.2	Autowire.....	49
5.4.3	Autowire Examples.....	50
5.5	Using Composites as Component Implementations.....	53
5.5.1	Example of Composite used as a Component Implementation.....	54
5.6	Using Composites through Inclusion.....	54
5.6.1	Included Composite Examples.....	55
5.7	Composites which Contain Component Implementations of Multiple Types.....	58
5.8	Structural URI of Components.....	58
6	ConstrainingType.....	60
6.1	Example constrainingType.....	61
7	Interface.....	63
7.1	Local and Remotable Interfaces.....	64
7.2	Bidirectional Interfaces.....	64

7.3	Long-running Request-Response Operations .....	66
7.3.1	Background .....	66
7.3.2	Definition of "long-running" .....	66
7.3.3	The asyncInvocation Intent .....	66
7.3.4	Requirements on Bindings .....	66
7.3.5	Implementation Type Support .....	66
7.4	SCA-Specific Aspects for WSDL Interfaces .....	67
7.5	WSDL Interface Type .....	67
7.5.1	Example of interface.wsdl .....	68
8	Binding.....	69
8.1	Messages containing Data not defined in the Service Interface .....	71
8.2	WireFormat .....	71
8.3	OperationSelector .....	71
8.4	Form of the URI of a Deployed Binding.....	72
8.4.1	Non-hierarchical URIs .....	72
8.4.2	Determining the URI scheme of a deployed binding.....	72
8.5	SCA Binding .....	73
8.5.1	Example SCA Binding .....	73
8.6	Web Service Binding .....	74
8.7	JMS Binding.....	74
9	SCA Definitions .....	75
10	Extension Model.....	76
10.1	Defining an Interface Type.....	76
10.2	Defining an Implementation Type.....	77
10.3	Defining a Binding Type.....	79
10.4	Defining an Import Type .....	80
10.5	Defining an Export Type .....	82
11	Packaging and Deployment .....	84
11.1	Domains.....	84
11.2	Contributions.....	84
11.2.1	SCA Artifact Resolution.....	85
11.2.2	SCA Contribution Metadata Document .....	87
11.2.3	Contribution Packaging using ZIP.....	89
11.3	Installed Contribution .....	89
11.3.1	Installed Artifact URIs.....	90
11.4	Operations for Contributions.....	90
11.4.1	install Contribution & update Contribution.....	90
11.4.2	add Deployment Composite & update Deployment Composite.....	90
11.4.3	remove Contribution .....	91
11.5	Use of Existing (non-SCA) Mechanisms for Resolving Artifacts .....	91
11.6	Domain-Level Composite .....	91
11.6.1	add To Domain-Level Composite.....	92
11.6.2	remove From Domain-Level Composite .....	92
11.6.3	get Domain-Level Composite .....	92
11.6.4	get QName Definition .....	92

11.7	Dynamic Behaviour of Wires in the SCA Domain.....	92
11.8	Dynamic Behaviour of Component Property Values .....	93
12	SCA Runtime Considerations.....	94
12.1	Error Handling.....	94
12.1.1	Errors which can be Detected at Deployment Time.....	94
12.1.2	Errors which are Detected at Runtime .....	94
13	Conformance .....	95
13.1	SCA Documents .....	95
13.2	SCA Runtime .....	96
A.	XML Schemas .....	97
A.1	sca.xsd .....	97
A.2	sca-core.xsd .....	97
A.3	sca-binding-sca.xsd.....	106
A.4	sca-interface-java.xsd .....	107
A.5	sca-interface-wsdl.xsd.....	107
A.6	sca-implementation-java.xsd.....	107
A.7	sca-implementation-composite.xsd.....	107
A.8	sca-binding-webservice.xsd .....	108
A.9	sca-binding-jms.xsd.....	108
A.10	sca-policy.xsd .....	108
A.11	sca-contribution.xsd .....	108
A.12	sca-definitions.xsd.....	110
B.	SCA Concepts.....	111
B.1	Binding.....	111
B.2	Component.....	111
B.3	Service.....	111
B.3.1	Remotable Service.....	111
B.3.2	Local Service.....	112
B.4	Reference .....	112
B.5	Implementation .....	112
B.6	Interface.....	112
B.7	Composite .....	113
B.8	Composite inclusion .....	113
B.9	Property .....	113
B.10	Domain .....	113
B.11	Wire .....	113
C.	Conformance Items .....	115
D.	Acknowledgements .....	127
E.	Non-Normative Text .....	129
F.	Revision History.....	130

---

# 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 it is suggested that this is 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.oasis-open.org/apps/org/workgroup/sca-j/download.php/31447/sca-javaci-1.1-spec-wd03.pdf>

[SCA-Common-Java] SCA Java Common Annotations and APIs Specification

<http://www.oasis-open.org/apps/org/workgroup/sca-j/download.php/31427/sca-javacaa-1.1-spec-cd02.pdf>

[SCA BPEL] SCA BPEL Client and Implementation Specification

<http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec-cd-01.pdf>

[SDO] SDO Specification

<http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf>

[3] SCA Example Code document

[http://www.osoa.org/download/attachments/28/SCA\\_BuildingYourFirstApplication\\_V09.pdf](http://www.osoa.org/download/attachments/28/SCA_BuildingYourFirstApplication_V09.pdf)

[4] JAX-WS Specification

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

[5] WS-I Basic Profile

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/wsdl>  
50 WSDL 2.0: <http://www.w3.org/TR/wsdl20/>  
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/wsdl11elementidentifiers/>  
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 e.g. <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  
107 and 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.

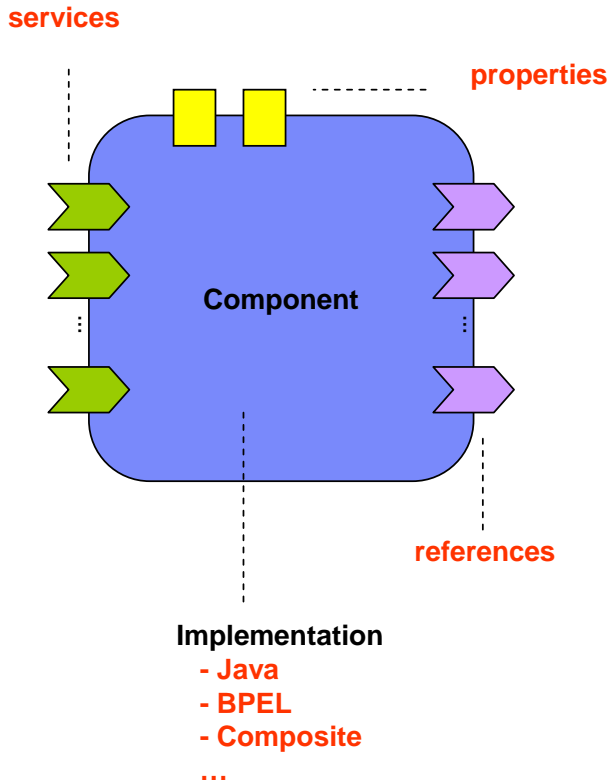
129 Composites are deployed within an **SCA Domain**. An SCA Domain typically represents a set of services  
130 providing an area of business functionality that is controlled by a single organization. As an example, for  
131 the accounts department in a business, the SCA Domain might cover all financial 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 might 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 and do not represent any formal graphical  
145 notation for SCA.

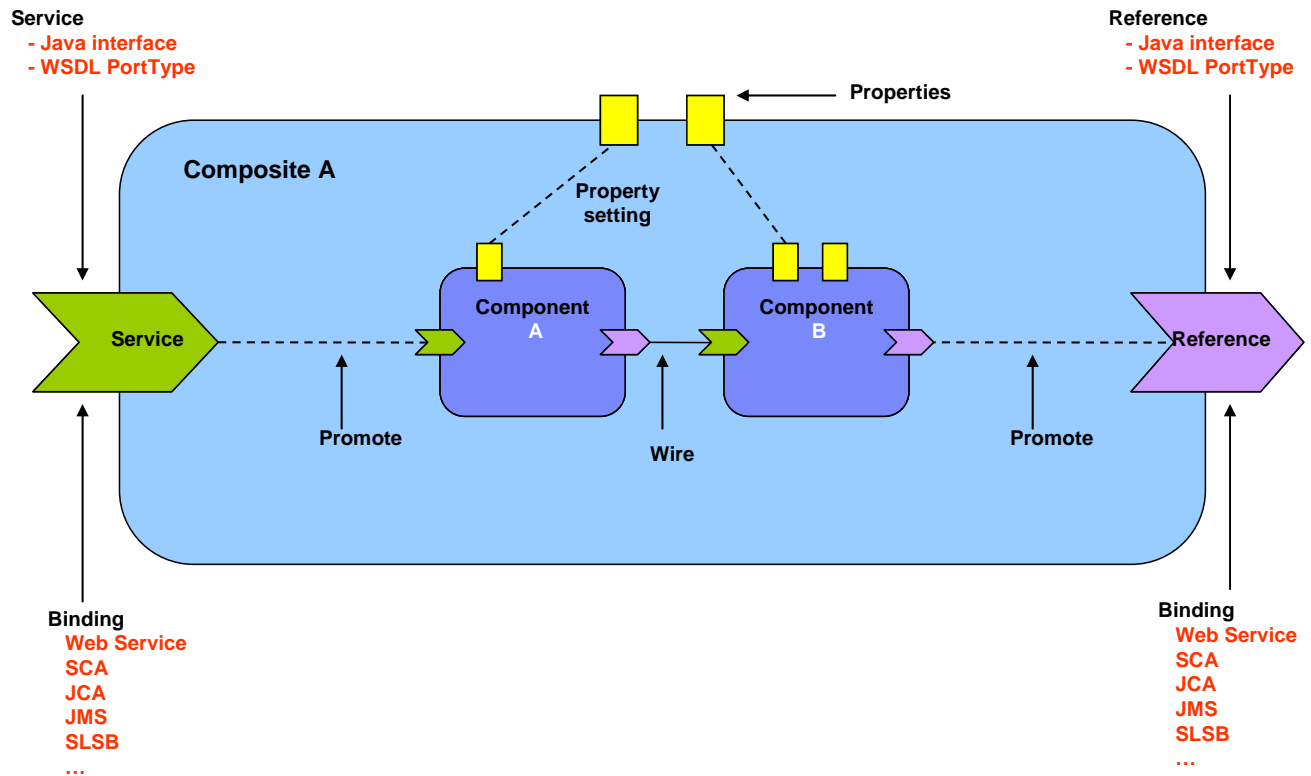
146 The following picture illustrates some of the features of an SCA component:



147  
148 *Figure 1: SCA Component Diagram*

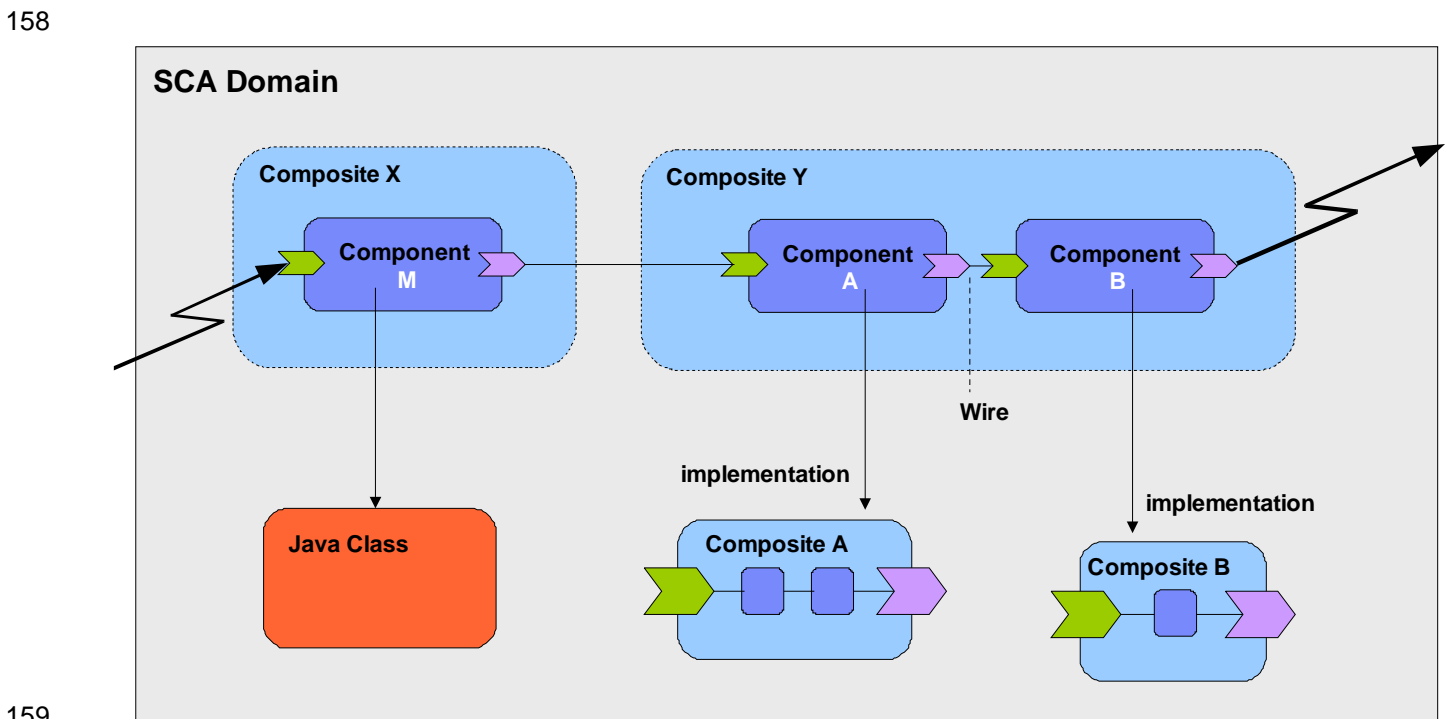
149  
150 The following picture illustrates some of the features of a composite assembled using a set of  
151 components:

152



153  
154 Figure 2: SCA Composite Diagram

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



159  
160 Figure 3: SCA Domain Diagram

161

---

## 3 Implementation and ComponentType

162

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

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

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

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

176 So, for example:

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

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

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

191

### 3.1 Component Type

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

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

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

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

213 **The extension of a componentType side file name MUST be .componentType.** [ASM40001] The  
214 name and location of a componentType side file, if allowed, is defined by the implementation type  
215 specification.

216 If a component type side file is not allowed for a particular implementation type, the effective  
217 component type and introspected component type are one and the same for that implementation  
218 type.

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

221 The following snippet shows the componentType pseudo-schema:

```
222 <?xml version="1.0" encoding="ASCII"?>  
223 <!-- Component type schema snippet -->  
224 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
225     constrainingType="xs:QName"? >  
226  
227     <service ... />*  
228     <reference ... />*  
229     <property ... />*  
230     <implementation ... />?  
231  
232 </componentType>  
233
```

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

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

240

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

- 242 • **service : Service (0..n)** – see [component type service section](#).
- 243 • **reference : Reference (0..n)** – see [component type reference section](#).
- 244 • **property : Property (0..n)** – see [component type property section](#).
- 245 • **implementation : Implementation (0..1)** – see [component type implementation](#)  
246 [section](#).

247

### 248 3.1.1 Service

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

253

```
254 <?xml version="1.0" encoding="ASCII"?>  
255 <!-- Component type service schema snippet -->
```

```

256 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ...
257 >
258
259     <service name="xs:NCName"
260         requires="list of xs:QName"? policySets="list of xs:QName"?>*
261         <interface ... />
262         <binding ... />*
263         <callback?
264             <binding ... />+
265         </callback>
266     </service>
267
268     <reference ... />*
269     <property ... />*
270     <implementation ... />?
271
272 </componentType>
273

```

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

- 275 • **name : NCName (1..1)** - the name of the service. The @name attribute of a <service/>  
276 child element of a <componentType/> MUST be unique amongst the service elements of  
277 that <componentType/>. [ASM40003]
- 278 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
279 [10] for a description of this attribute.
- 280 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
281 [10] for a description of this attribute.

282

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

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

296

### 297 3.1.2 Reference

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

```

303 <?xml version="1.0" encoding="ASCII"?>
304 <!-- Component type reference schema snippet -->
305 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ...
306 >

```

```

307
308     <service ... />*
309
310     <reference name="xs:NCName"
311         autowire="xs:boolean"?
312         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
313         wiredByImpl="xs:boolean"?
314         requires="list of xs:QName"? policySets="list of xs:QName"?>*
315         <interface ... />
316         <binding ... />*
317         <callback?
318             <binding ... />+
319         </callback>
320     </reference>
321
322     <property ... />*
323     <implementation ... />?
324
325 </componentType>
326

```

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

- 328 • **name : NCName (1..1)** - the name of the reference. The @name attribute of a  
329 <reference/> child element of a <componentType/> MUST be unique amongst the  
330 reference elements of that <componentType/>. [ASM40004]
- 331 • **multiplicity : 0..1/1..1/0..n/1..n (0..1)** - defines the number of wires that can connect  
332 the reference to target services. The multiplicity can have the following values
  - 333 ○ 0..1 – zero or one wire can have the reference as a source
  - 334 ○ 1..1 – one wire can have the reference as a source
  - 335 ○ 0..n - zero or more wires can have the reference as a source
  - 336 ○ 1..n – one or more wires can have the reference as a source
337 If @multiplicity is not specified, the default value is "1..1".
- 338 • **autowire : boolean (0..1)** - whether the reference is autowired, as described in the  
339 [Autowire section](#). Default is false.
- 340 • **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the  
341 reference is wired to the target(s) configured on the reference. If set to "true" it indicates  
342 that the target of the reference is set at runtime by the implementation code (e.g. by the  
343 code obtaining an endpoint reference by some means and setting this as the target of the  
344 reference through the use of programming interfaces defined by the relevant Client and  
345 Implementation specification). If @wiredByImpl is set to "true", then any reference  
346 targets configured for this reference MUST be ignored by the runtime. [ASM40006]
- 347 • **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
348 [10] for a description of this attribute.
- 349 • **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
350 [10] for a description of this attribute.

351

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

- 353 • **interface : Interface (1..1)** - A reference has *one interface*, which describes the  
354 operations used by the reference. The interface is described by an *interface element*  
355 which is a child element of the reference element. For details on the interface element see  
356 [the Interface section](#).



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

359 When used with a reference element, a binding element specifies an endpoint which is the  
360 target of that binding. A reference cannot mix the use of endpoints specified via binding  
361 elements with target endpoints specified via the @target attribute. If the @target  
362 attribute is set, the reference cannot also have binding subelements. If binding elements  
363 with endpoints are specified, each endpoint uses the binding type of the binding element  
364 in which it is defined.

365 • **callback (0..1) / binding : Binding (1..n)** - a **callback** element is used if the interface  
366 has a callback defined and the callback element has one or more **binding** elements as  
367 subelements. The **callback** and its binding subelements are specified if there is a need to  
368 have binding details used to handle callbacks. If the callback element is not present, the  
369 behaviour is runtime implementation dependent. For details on callbacks, see [the](#)  
370 [Bidirectional Interfaces section](#).

371 For a full description of the setting of target service(s) for a reference, see the section "[Specifying](#)  
372 [the Target Service\(s\) for a Reference](#)".

### 373 3.1.3 Property

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

```
378 <?xml version="1.0" encoding="ASCII"?>  
379 <!-- Component type property schema snippet -->  
380 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ...  
381 >  
382  
383 <service ... /*>  
384 <reference ... /*>  
385  
386 <property name="xs:NCName" (type="xs:QName" | element="xs:QName")  
387     many="xs:boolean"? mustSupply="xs:boolean"?  
388     requires="list of xs:QName"?  
389     policySets="list of xs:QName"?>*<br>390     default-property-value?  
391 </property>  
392  
393 <implementation ... /*?>  
394  
395 </componentType>  
396
```

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

- 398 ▪ **name : NCName (1..1)** - the name of the property. The @name attribute of a  
399 <property/> child element of a <componentType/> MUST be unique amongst the  
400 property elements of that <componentType/>. [ASM40005]
- 401 ▪ one of (1..1):
  - 402 ○ **type : QName** - the type of the property defined as the qualified name of an XML  
403 schema type. The value of the property @type attribute MUST be the QName of  
404 an XML schema type. [ASM40007]
  - 405 ○ **element : QName** - the type of the property defined as the qualified name of an  
406 XML schema global element – the type is the type of the global element. The value  
407 of the property @element attribute MUST be the QName of an XSD global  
408 element. [ASM40008]

409 A single property element MUST NOT contain both a @type attribute and an @element  
410 attribute. [ASM40010]

- 411 ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued  
412 (true). In the case of a multi-valued property, it is presented to the implementation as a  
413 collection of property values. If many is not specified, it takes a default value of false.
- 414 ▪ **mustSupply : boolean (0..1)** - whether the property value needs to be supplied by the  
415 component that uses the implementation. Default value is "false". When the  
416 componentType has @mustSupply="true" for a property element, a component using the  
417 implementation MUST supply a value for the property since the implementation has no  
418 default value for the property. [ASM40011] If the implementation has a default-property-  
419 value then @mustSupply="false" is appropriate, since the implication of a default value is  
420 that it is used when a value is not supplied by the using component.
- 421 ▪ **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the property.
- 422 ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
423 [10] for a description of this attribute.
- 424 ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
425 [10] for a description of this attribute.

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

428 The value for a property is supplied to the implementation of a component at the time that the  
429 implementation is started. The implementation can use the supplied value in any way that it  
430 chooses. In particular, the implementation can alter the internal value of the property at any time.  
431 However, if the implementation queries the SCA system for the value of the property, the value as  
432 defined in the SCA composite is the value returned.

433 The componentType property element can contain an SCA default value for the property declared  
434 by the implementation. However, the implementation can have a property which has an  
435 implementation defined default value, where the default value is not represented in the  
436 componentType. An example of such a default value is where the default value is computed at  
437 runtime by some code contained in the implementation. If a using component needs to control the  
438 value of a property used by an implementation, the component sets the value explicitly. The SCA  
439 runtime MUST ensure that any implementation default property value is replaced by a value for  
440 that property explicitly set by a component using that implementation. [ASM40009]

441

### 442 3.1.4 Implementation

443 **Implementation** represents characteristics inherent to the implementation itself, in particular  
444 intents and policies. See the [Policy Framework specification](#) [10] for a description of intents and  
445 policies. The following snippet shows the component type pseudo-schema with the pseudo-schema  
446 for a implementation child element:

447

```
448 <?xml version="1.0" encoding="ASCII"?>
449 <!-- Component type implementation schema snippet -->
450 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ...
451 >
452
453     <service ... /*>
454     <reference ... /*>
455     <property ... /*>
456
457     <implementation requires="list of xs:QName"?
458                 policySets="list of xs:QName"?/>?
459
```

```
460 </componentType>
461
```

462 The **implementation** element has the following **attributes**:

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

467

## 468 3.2 Example ComponentType

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

```
472 <?xml version="1.0" encoding="ASCII"?>
473 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903">
474
475     <service name="MyValueService">
476         <interface.java interface="services.myvalue.MyValueService"/>
477     </service>
478
479     <reference name="customerService">
480         <interface.java interface="services.customer.CustomerService"/>
481     </reference>
482     <reference name="stockQuoteService">
483         <interface.java
484             interface="services.stockquote.StockQuoteService"/>
485     </reference>
486
487     <property name="currency" type="xsd:string">USD</property>
488
489 </componentType>
490
```

## 491 3.3 Example Implementation

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

494 **AccountServiceImpl** implements the **AccountService** interface, which is defined via a Java  
495 interface:

```
496 package services.account;
497
498 @Remotable
499 public interface AccountService {
500
501     AccountReport getAccountReport(String customerID);
502 }
503
```

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

```
508 package services.account;
509
510 import java.util.List;
511
```

```

512     import commonj.sdo.DataFactory;
513
514     import org.oasisopen.sca.annotation.Property;
515     import org.oasisopen.sca.annotation.Reference;
516     import org.oasisopen.sca.annotation.Service;
517
518     import services.accountdata.AccountDataService;
519     import services.accountdata.CheckingAccount;
520     import services.accountdata.SavingsAccount;
521     import services.accountdata.StockAccount;
522     import services.stockquote.StockQuoteService;
523
524     @Service(AccountService.class)
525     public class AccountServiceImpl implements AccountService {
526
527         @Property
528         private String currency = "USD";
529
530         @Reference
531         private AccountDataService accountDataService;
532         @Reference
533         private StockQuoteService stockQuoteService;
534
535         public AccountReport getAccountReport(String customerID) {
536
537             DataFactory dataFactory = DataFactory.INSTANCE;
538             AccountReport accountReport = (AccountReport)dataFactory.create(AccountReport.class);
539             List accountSummaries = accountReport.getAccountSummaries();
540
541             CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
542             AccountSummary checkingAccountSummary =
543                 (AccountSummary)dataFactory.create(AccountSummary.class);
544             checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
545             checkingAccountSummary.setAccountType("checking");
546             checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
547             accountSummaries.add(checkingAccountSummary);
548
549             SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
550             AccountSummary savingsAccountSummary =
551                 (AccountSummary)dataFactory.create(AccountSummary.class);
552             savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
553             savingsAccountSummary.setAccountType("savings");
554             savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
555             accountSummaries.add(savingsAccountSummary);
556
557             StockAccount stockAccount = accountDataService.getStockAccount(customerID);
558             AccountSummary stockAccountSummary =
559                 (AccountSummary)dataFactory.create(AccountSummary.class);
560             stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
561             stockAccountSummary.setAccountType("stock");
562             float balance=
563                 (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
564             stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
565             accountSummaries.add(stockAccountSummary);
566
567             return accountReport;
568         }
569
570         private float fromUSDollarToCurrency(float value){
571
572             if (currency.equals("USD")) return value; else
573             if (currency.equals("EURO")) return value * 0.8f; else
574             return 0.0f;
575         }
576     }
577

```

578 The following is the SCA componentType definition for the AccountServiceImpl, derived by  
579 introspection of the code above:

```

580 <?xml version="1.0" encoding="ASCII"?>
581 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"

```

```
582         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
583
584     <service name="AccountService">
585         <interface.java interface="services.account.AccountService"/>
586     </service>
587     <reference name="accountDataService">
588         <interface.java
589             interface="services.accountdata.AccountDataService"/>
590     </reference>
591     <reference name="stockQuoteService">
592         <interface.java
593             interface="services.stockquote.StockQuoteService"/>
594     </reference>
595
596     <property name="currency" type="xsd:string"/>
597
598 </componentType>
599
```

600 Note that the componentType property element for "currency" has no default value declared,  
601 despite the code containing an initializer for the property field setting it to "USD". This is because  
602 the initializer cannot be introspected at runtime and the value cannot be extracted.

603 For full details about Java implementations, see the [Java Component Implementation Specification](#)  
604 [SCA-Java]. Other implementation types have their own specification documents.

---

## 4 Component

605

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

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

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

```
615 <?xml version="1.0" encoding="UTF-8"?>
616 <!-- Component schema snippet -->
617 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
618     ...
619     <component name="xs:NCName" autowire="xs:boolean"?
620         requires="list of xs:QName"? policySets="list of xs:QName"?
621         constrainingType="xs:QName"?>*
622         <implementation ... />?
623         <service ... />*
624         <reference ... />*
625         <property ... />*
626     </component>
627     ...
628 </composite>
```

629

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

- 631 • **name** : **NCName (1..1)** – the name of the component. The @name attribute of a  
632 <component/> child element of a <composite/> MUST be unique amongst the component  
633 elements of that <composite/> [ASM50001]
- 634 • **autowire** : **boolean (0..1)** – whether contained component references are autowired, as  
635 described in the [Autowire section](#). Default is false.
- 636 • **requires** : **QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
637 [10] for a description of this attribute.
- 638 • **policySets** : **QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
639 [10] for a description of this attribute.
- 640 • **constrainingType** : **QName (0..1)** – the name of a constrainingType. When specified,  
641 the set of services, references and properties of the component, plus related intents, is  
642 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
643 for more details.

644

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

- 646 • **implementation** : **ComponentImplementation (0..1)** – see [component](#)  
647 [implementation section](#).
- 648 • **service** : **ComponentService (0..n)** – see [component service section](#).
- 649 • **reference** : **ComponentReference (0..n)** – see [component reference section](#).
- 650 • **property** : **ComponentProperty (0..n)** – see [component property section](#).

651

## 652 4.1 Implementation

653 A component element has **zero or one implementation element** as its child, which points to the  
654 implementation used by the component. A component with no implementation element is not  
655 runnable, but components of this kind can be useful during a "top-down" development process as  
656 a means of defining the necessary characteristics of the implementation before the  
657 implementation is written.

```
658 <?xml version="1.0" encoding="UTF-8"?>
659 <!-- Component Implementation schema snippet -->
660 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
661   ...
662   <component ... >*
663     <implementation ... />?
664     <service ... />*
665     <reference ... />*
666     <property ... />*
667   </component>
668   ...
669 </composite>
670
```

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

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

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

```
681
682 <implementation.java class="services.myvalue.MyValueServiceImpl"/>
683
684 <implementation.bpel process="ans:MoneyTransferProcess"/>
685
686 <implementation.composite name="bns:MyValueComposite"/>
687
```

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

689 At runtime, an **implementation instance** is a specific runtime instantiation of the  
690 implementation – its runtime form depends on the implementation technology used. The  
691 implementation instance derives its business logic from the implementation on which it is based,  
692 but the values for its properties and references are derived from the component which configures  
693 the implementation.

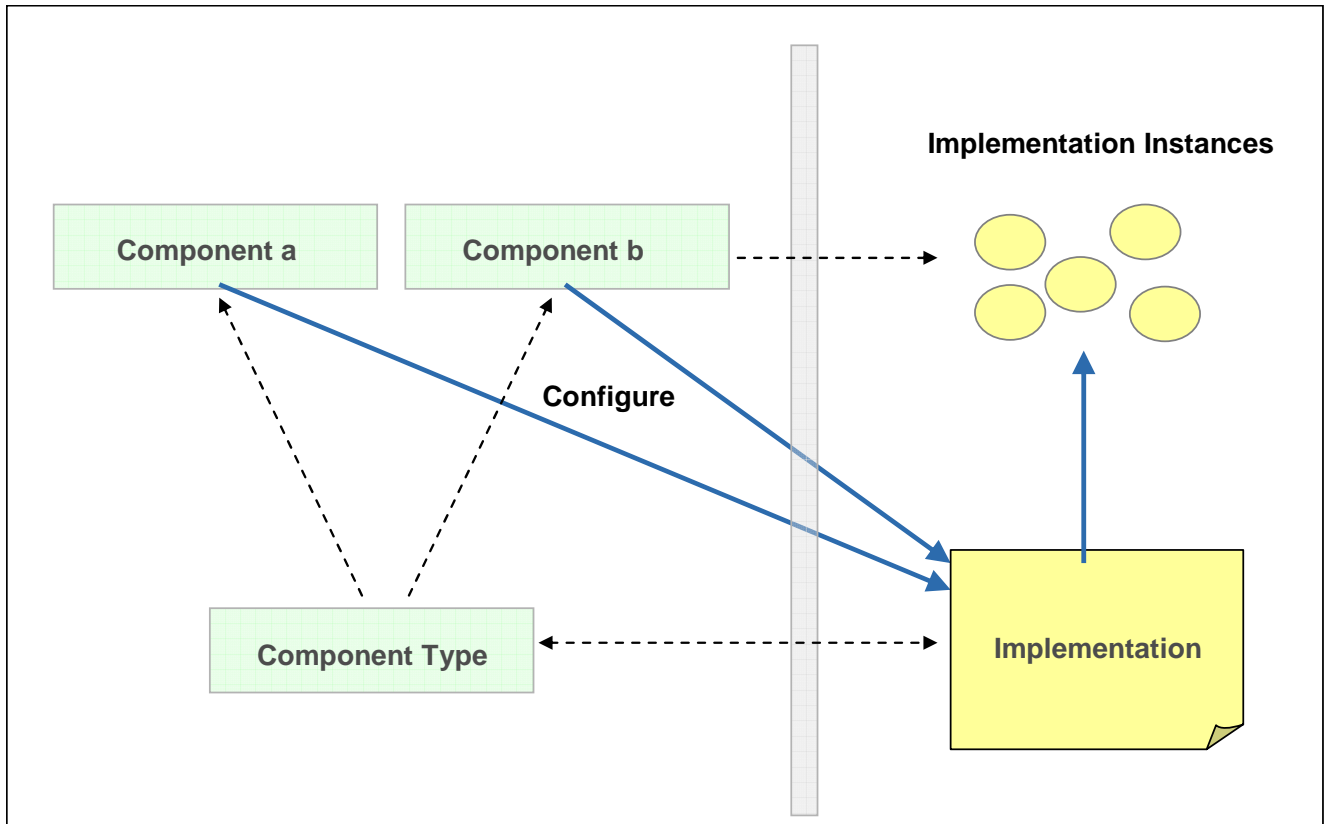


Figure 4: Relationship of Component and Implementation

## 4.2 Service

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

```

694 <?xml version="1.0" encoding="UTF-8"?>
695 <!-- Component Service schema snippet -->
696 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
697   ...
698   <component ... >*
699     <implementation ... />?
700     <service name="xs:NCName" requires="list of xs:QName"?
701       policySets="list of xs:QName"?>*
702       <interface ... />?
703       <binding ... />*
704       <callback?
705         <binding ... />+
706       </callback>
707     </service>
708     <reference ... />*
709     <property ... />*
710   </component>
711   ...
712 </composite>

```



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

- 723 • **name : NCName (1..1)** - the name of the service. The @name attribute of a service  
724 element of a <component/> MUST be unique amongst the service elements of that  
725 <component/> [ASM50002] The @name attribute of a service element of a  
726 <component/> MUST match the @name attribute of a service element of the  
727 componentType of the <implementation/> child element of the component. [ASM50003]
- 728 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
729 [10] for a description of this attribute.  
730 Note: The effective set of policy intents for the service consists of any intents explicitly  
731 stated in this @requires attribute, combined with any intents specified for the service by  
732 the implementation.
- 733 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
734 [10] for a description of this attribute.

735

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

- 737 • **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the  
738 operations provided by the service. The interface is described by an **interface element**  
739 which is a child element of the service element. If no interface is specified, then the  
740 interface specified for the service in the componentType of the implementation is in effect.  
741 If a <service/> element has an interface subelement specified, the interface MUST provide  
742 a compatible subset of the interface declared on the componentType of the  
743 implementation [ASM50004] For details on the interface element see the [Interface section](#).
- 744 • **binding : Binding (0..n)** - A service element has **zero or more binding elements** as  
745 children. If no binding elements are specified for the service, then the bindings specified  
746 for the equivalent service in the componentType of the implementation MUST be used, but  
747 if the componentType also has no bindings specified, then <binding.sca/> MUST be used  
748 as the binding. If binding elements are specified for the service, then those bindings MUST  
749 be used and they override any bindings specified for the equivalent service in the  
750 componentType of the implementation. [ASM50005] Details of the binding element are  
751 described in the [Bindings section](#). The binding, combined with any PolicySets in effect for  
752 the binding, needs to satisfy the set of policy intents for the service, as described in the  
753 [Policy Framework specification](#) [10].
- 754 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface  
755 has a callback defined and the callback element has one or more **binding** elements as  
756 subelements. The **callback** and its binding subelements are specified if there is a need to  
757 have binding details used to handle callbacks. If the callback element is present and  
758 contains one or more binding child elements, then those bindings MUST be used for the  
759 callback. [ASM50006] If the callback element is not present, the behaviour is runtime  
760 implementation dependent.

761

## 762 4.3 Reference

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

```
767 <?xml version="1.0" encoding="UTF-8"?>  
768 <!-- Component Reference schema snippet -->  
769 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >  
770 ...  
771 <component ... >*  
772 <implementation ... />?  
773 <service ... />*
```

```

774     <reference name="xs:NCName"
775         target="list of xs:anyURI"? autowire="xs:boolean"?
776         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
777         nonOverridable="xs:boolean"
778         wiredByImpl="xs:boolean"? requires="list of xs:QName"?
779         policySets="list of xs:QName"?>*
780     <interface ... />?
781     <binding uri="xs:anyURI"? requires="list of xs:QName"?
782         policySets="list of xs:QName"?/>*
783     <callback?
784         <binding ... />+
785     </callback>
786 </reference>
787 <property ... />*
788 </component>
789 ...
790 </composite>
791

```

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

- 793 • **name : NCName (1..1)** – the name of the reference. The @name attribute of a service  
794 element of a <component/> MUST be unique amongst the service elements of that  
795 <component/> [ASM50007] The @name attribute of a reference element of a  
796 <component/> MUST match the @name attribute of a reference element of the  
797 componentType of the <implementation/> child element of the component. [ASM50008]
  - 798 • **autowire : boolean (0..1)** – whether the reference is autowired, as described in the  
799 [Autowire section](#). Default is false.
  - 800 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
801 [10] for a description of this attribute.  
802 Note: The effective set of policy intents for the reference consists of any intents explicitly  
803 stated in this @requires attribute, combined with any intents specified for the reference by  
804 the implementation.
  - 805 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
806 [10] for a description of this attribute.
  - 807 • **multiplicity : 0..1/1..1/0..n/1..n (0..1)** - defines the number of wires that can connect  
808 the reference to target services. Overrides the multiplicity specified for this reference in  
809 the componentType of the implementation. The multiplicity can have the following values
    - 810 ○ 0..1 – zero or one wire can have the reference as a source
    - 811 ○ 1..1 – one wire can have the reference as a source
    - 812 ○ 0..n - zero or more wires can have the reference as a source
    - 813 ○ 1..n – one or more wires can have the reference as a source
- 814 The value of multiplicity for a component reference MUST only be equal or further restrict  
815 any value for the multiplicity of the reference with the same name in the componentType  
816 of the implementation, where further restriction means 0..n to 0..1 or 1..n to 1..1.  
817 [ASM50009]
- 818 If not present, the value of multiplicity is equal to the multiplicity specified for this  
819 reference in the componentType of the implementation - if not present in the  
820 componentType, the value defaults to 1..1.
- 821 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
822 multiplicity setting. Each value wires the reference to a component service that resolves  
823 the reference. For more details on wiring see [the section on Wires](#). Overrides any target  
824 specified for this reference on the implementation.

- 825 • **wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that  
826 the implementation wires this reference dynamically. If set to "true" it indicates that the  
827 target of the reference is set at runtime by the implementation code (e.g. by the code  
828 obtaining an endpoint reference by some means and setting this as the target of the  
829 reference through the use of programming interfaces defined by the relevant Client and  
830 Implementation specification). If @wiredByImpl="true" is set for a reference, then the  
831 reference MUST NOT be wired statically within a composite, but left unwired. [ASM50010]
- 832 • **nonOverridable : boolean (0..1)** - a boolean value, "false" by default, which indicates  
833 whether this component reference can have its targets overridden by a composite  
834 reference which promotes the component reference.  
835 If @nonOverridable=false, the target(s) of the promoting composite reference replace all  
836 the targets explicitly declared on the component reference for any value of @multiplicity  
837 on the component reference. If the component reference has @nonOverridable=false  
838 and @multiplicity 1..1 and the reference has a target, then any composite reference which  
839 promotes the component reference has @multiplicity 0..1 by default and MAY have an  
840 explicit @multiplicity of either 0..1 or 1..1.  
841 If @nonOverridable=true, and the component reference has @multiplicity 0..1 or 1..1  
842 and the component reference also declares a target, promotion implies that the promoting  
843 composite reference has @wiredByImpl=true and the composite reference cannot supply  
844 a target, but can influence the policy attached to the component reference.  
845 If @nonOverridable=true, and the component reference @multiplicity is 0..n or 1..n,  
846 promotion targeting is additive.

847

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

- 849 • **interface : Interface (0..1)** - A reference has **zero or one interface**, which describes  
850 the operations of the reference. The interface is described by an **interface element** which  
851 is a child element of the reference element. If no interface is specified, then the interface  
852 specified for the reference in the componentType of the implementation is in effect. If an  
853 interface is declared for a component reference, the interface MUST provide a compatible  
854 superset of the interface declared for the equivalent reference in the componentType of  
855 the implementation, i.e. provide the same operations or a superset of the operations  
856 defined by the implementation for the reference. [ASM50011] For details on the interface  
857 element see the [Interface section](#).
- 858 • **binding : Binding (0..n)** - A reference element has **zero or more binding elements** as  
859 children. If no binding elements are specified for the reference, then the bindings specified  
860 for the equivalent reference in the componentType of the implementation MUST be used.  
861 If binding elements are specified for the reference, then those bindings MUST be used and  
862 they override any bindings specified for the equivalent reference in the componentType of  
863 the implementation. [ASM50012] It is valid for there to be no binding elements on the  
864 component reference and none on the reference in the componentType - the binding used  
865 for such a reference is determined by the target service. See the [section on the bindings  
866 of component services](#) for a description of how the binding(s) applying to a service are  
867 determined.  
868 Details of the binding element are described in the [Bindings section](#). The binding,  
869 combined with any PolicySets in effect for the binding, needs to satisfy the set of policy  
870 intents for the reference, as described in the [Policy Framework specification \[10\]](#).  
871 A reference identifies zero or more target services that satisfy the reference. This can be  
872 done in a number of ways, which are fully described in section "[Specifying the Target  
873 Service\(s\) for a Reference](#)"
- 874 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element used if the interface  
875 has a callback defined and the callback element has one or more **binding** elements as  
876 subelements. The **callback** and its binding subelements are specified if there is a need to  
877 have binding details used to handle callbacks. If the callback element is present and  
878 contains one or more binding child elements, then those bindings MUST be used for the  
879 callback. [ASM50006] If the callback element is not present, the behaviour is runtime  
880 implementation dependent.

### 881 4.3.1 Specifying the Target Service(s) for a Reference

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

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

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

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

#### 919 4.3.1.1 Multiplicity and the Valid Number of Target Services for a Reference

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

- 921 • A reference with multiplicity 0..1 or 0..n MAY have no target service defined. [ASM50018]
- 922 • A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service  
923 defined. [ASM50019]
- 924 • A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined.  
925 [ASM50020]
- 926 • A reference with multiplicity 0..n or 1..n MAY have one or more target services defined.  
927 [ASM50021]

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

931 For example, where a composite is used as a component implementation, wires and target  
932 services cannot be added to the composite after deployment. As a result, for components which  
933 are part of the composite, both missing wires and wires with a non-existent target can be detected  
934 at deployment time through a scan of the contents of the composite.

935 A contrasting example is a component deployed to the SCA Domain. At the Domain level, the  
936 target of a wire, or even the wire itself, can form part of a separate deployed contribution and as a  
937 result these can be deployed after the original component is deployed. For the cases where it is  
938 valid for the reference to have no target service specified, the component implementation  
939 language specification needs to define the programming model for interacting with an untargetted  
940 reference.

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

## 946 4.4 Property

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

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

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

958 For example,

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

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

963 For example,

- 964 • property defined using a XML Schema simple type and which contains a single  
965 value

```
966 <property name="pi">  
967   <value>3.14159265</value>  
968 </property>
```

- 969 • property defined using a XML Schema simple type and which contains multiple  
970 values

```
971 <property name="currency">  
972   <value>EURO</value>  
973   <value>USDollar</value>  
974 </property>
```

- 975 • property defined using a XML Schema complex type and which contains a single  
976 value

```
977 <property name="complexFoo">  
978   <value attr="bar">
```

```

979         <foo:a>TheValue</foo:a>
980         <foo:b>InterestingURI</foo:b>
981     </value>
982 </property>
983 • property defined using a XML Schema complex type and which contains multiple
984 values

```

```

985     <property name="complexBar">
986         <value anotherAttr="foo">
987             <bar:a>AValue</bar:a>
988             <bar:b>InterestingURI</bar:b>
989         </value>
990         <value attr="zing">
991             <bar:a>BValue</bar:a>
992             <bar:b>BoringURI</bar:b>
993         </value>
994     </property>

```

- As a value, supplied as the content of the property element.  
 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. [ASM50029]

999 For example,

- property defined using a XML Schema global element declartion and which contains a single value

```

1002     <property name="foo">
1003         <foo:SomeGED ...>...</foo:SomeGED>
1004     </property>

```

- property defined using a XML Schema global element declaration and which contains multiple values

```

1007     <property name="bar">
1008         <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1009         <bar:SomeOtherGED ...>...</bar:SomeOtherGED>
1010     </property>

```

- By referencing a Property value of the composite which contains the component. The reference is made using the **@source** attribute of the property element.

1014 The form of the value of the @source attribute follows the form of an XPath expression. This form allows a specific property of the composite to be addressed by name. Where the composite property is of a complex type, the XPath expression can be extended to refer to a sub-part of the complex property value.

1019 So, for example, source="\$currency" is used to reference a property of the composite called "currency", while source="\$currency/a" references the sub-part "a" of the complex composite property with the name "currency".

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

1024

1025 If more than one property value specification is present, the @source attribute takes precedence, then the @file attribute.

1027 For a property defined using a XML Schema simple type and for which a single value is desired, can be set either using the @value attribute or the <value> child element. The two forms in such a case are equivalent.

1030 When a property has multiple values set, they MUST all be contained within the same property element. A <component/> element MUST NOT contain two <property/> subelements with the same value of the @name attribute. [ASM50030]

1033 The type of the property can be specified in **one** of two ways:

- 1034 • by the qualified name of a type defined in an XML schema, using the **@type** attribute
- 1035 • by the qualified name of a global element in an XML schema, using the **@element** attribute

1036 The property type specified for the property element of a component MUST be compatible with the  
 1037 type of the property with the same @name declared in the component type of the implementation  
 1038 used by the component. If no type is declared in the component property element, the type of the  
 1039 property declared in the componentType of the implementation MUST be used. [ASM50036]

1040 The following snippet shows the component schema with the schema for a property child element:

```
1041 <?xml version="1.0" encoding="UTF-8"?>
1042 <!-- Component Property schema snippet -->
1043 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
1044   ...
1045   <component ... >*
1046     <implementation ... />?
1047     <service ... />*
1048     <reference ... />*
1049     <property name="xs:NCName"
1050       (type="xs:QName" | element="xs:QName")?
1051       many="xs:boolean"?
1052       source="xs:string"? file="xs:anyURI"?
1053       requires="list of xs:QName"?
1054       policySets="list of xs:QName"?
1055       value="xs:string"?>*
1056       [<value>+ | xs:any+ ]?
1057     </property>
1058   </component>
1059   ...
1060 </composite>
```

1061  
 1062 The **component property** element has the following **attributes**:

- 1063 ▪ **name : NCName (1..1)** – the name of the property. The @name attribute of a property  
 1064 element of a <component/> MUST be unique amongst the property elements of that  
 1065 <component/>. [ASM50031] The @name attribute of a property element of a  
 1066 <component/> MUST match the @name attribute of a property element of the  
 1067 componentType of the <implementation/> child element of the component. [ASM50037]
- 1068 ▪ zero or one of **(0..1)**:
  - 1069 ○ **type : QName** – the type of the property defined as the qualified name of an XML  
 1070 schema type
  - 1071 ○ **element : QName** – the type of the property defined as the qualified name of an  
 1072 XML schema global element – the type is the type of the global element
- 1073 ▪ A single property element MUST NOT contain both a @type attribute and an @element  
 1074 attribute. [ASM50035]
- 1075 ▪ **source : string (0..1)** – an XPath expression pointing to a property of the containing  
 1076 composite from which the value of this component property is obtained.
- 1077 ▪ **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
- 1078 ▪ **many : boolean (0..1)** – whether the property is single-valued (false) or multi-valued  
 1079 (true). Overrides the many specified for this property in the componentType of the  
 1080 implementation. The value can only be equal or further restrict, i.e. if the implementation  
 1081 specifies many true, then the component can say false. In the case of a multi-valued  
 1082 property, it is presented to the implementation as a Collection of property values. If many  
 1083 is not specified, it takes the value defined by the component type of the implementation  
 1084 used by the component.

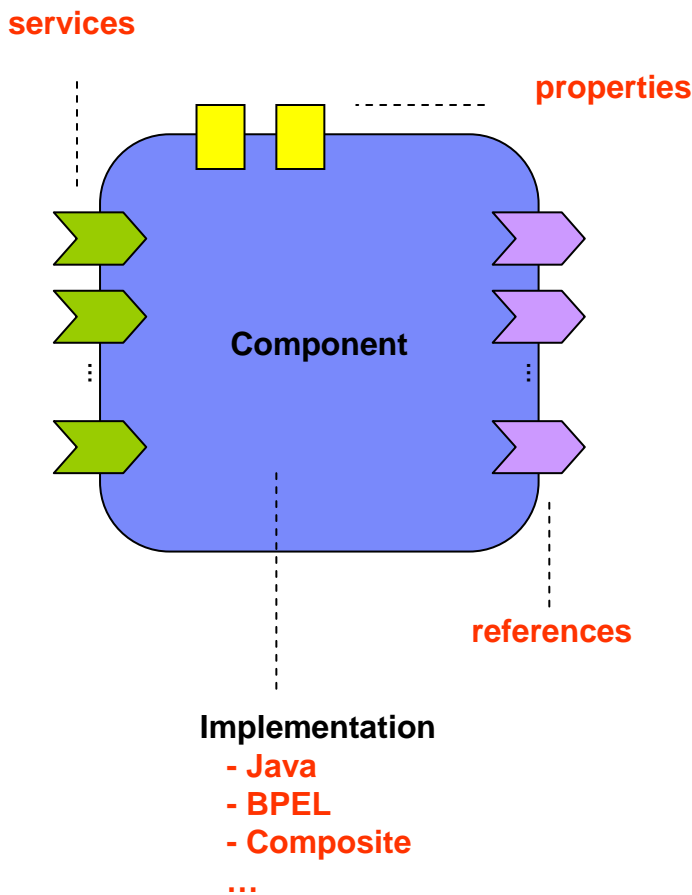
- 1085       ▪ **value : string (0..1)** - the value of the property if the property is defined using a simple  
1086       type.
- 1087       ▪ **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification](#)  
1088       [\[10\]](#) for a description of this attribute.
- 1089       ▪ **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification](#)  
1090       [\[10\]](#) for a description of this attribute.

1091 The **component property** element has the following **child element**:

1092 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1093 property that is defined using a XML Schema type. **If a property is single-valued, the <value/>**  
1094 **subelement MUST NOT occur more than once. [ASM50032]** A property <value/> subelement **MUST**  
1095 **NOT be used when the @value attribute is used to specify the value for that property. [ASM50033]**

## 1096 4.5 Example Component

1097 The following figure shows the **component symbol** that is used to represent a component in an  
1098 assembly diagram.

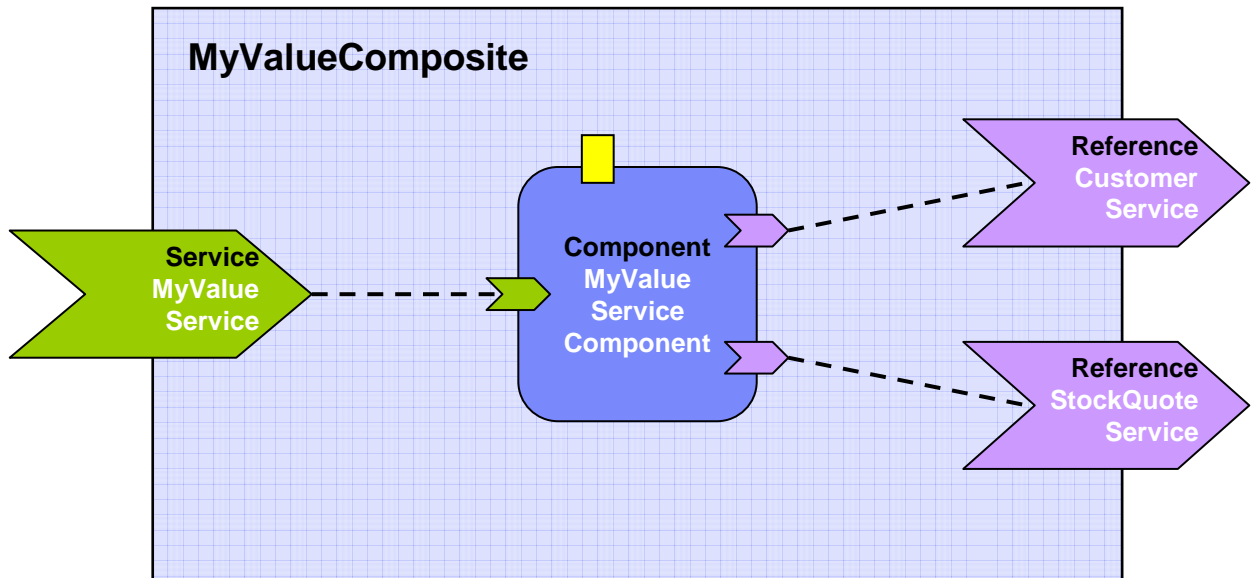


1099  
1100 *Figure 5: Component symbol*

1101 The following figure shows the assembly diagram for the MyValueComposite containing the  
1102 MyValueServiceComponent.

1103





1104  
1105

1106 *Figure 6: Assembly diagram for MyValueComposite*

1107 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1108 containing the component element for the MyValueServiceComponent. A value is set for the  
1109 property named currency, and the customerService and stockQuoteService references are  
1110 promoted:

```

1111 <?xml version="1.0" encoding="ASCII"?>
1112 <!-- MyValueComposite_1 example -->
1113 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1114           targetNamespace="http://foo.com"
1115           name="MyValueComposite" >
1116
1117     <service name="MyValueService" promote="MyValueServiceComponent"/>
1118
1119     <component name="MyValueServiceComponent">
1120       <implementation.java
1121         class="services.myvalue.MyValueServiceImpl"/>
1122       <property name="currency">EURO</property>
1123       <reference name="customerService"/>
1124       <reference name="stockQuoteService"/>
1125     </component>
1126
1127     <reference name="CustomerService"
1128       promote="MyValueServiceComponent/customerService"/>
1129
1130     <reference name="StockQuoteService"
1131       promote="MyValueServiceComponent/stockQuoteService"/>
1132
1133 </composite>

```

1134  
1135  
1136  
1137  
1138

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

1139 The following snippet gives an example of the layout of a composite file if both the currency  
1140 property and the customerService reference of the MyValueServiceComponent are declared to be  
1141 multi-valued (many=true for the property and multiplicity=0..n or 1..n for the reference):

```
1142 <?xml version="1.0" encoding="ASCII"?>
1143 <!-- MyValueComposite_2 example -->
1144 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1145           targetNamespace="http://foo.com"
1146           name="MyValueComposite" >
1147
1148     <service name="MyValueService" promote="MyValueServiceComponent"/>
1149
1150     <component name="MyValueServiceComponent">
1151       <implementation.java
1152         class="services.myvalue.MyValueServiceImpl"/>
1153       <property name="currency">
1154         <value>EURO</value>
1155         <value>Yen</value>
1156         <value>USDollar</value>
1157       </property>
1158       <reference name="customerService"
1159         target="InternalCustomer/customerService"/>
1160       <reference name="stockQuoteService"/>
1161     </component>
1162
1163     ...
1164
1165     <reference name="CustomerService"
1166       promote="MyValueServiceComponent/customerService"/>
1167
1168     <reference name="StockQuoteService"
1169       promote="MyValueServiceComponent/stockQuoteService"/>
1170
1171 </composite>
```

1172  
1173 ....this assumes that the composite has another component called InternalCustomer (not shown)  
1174 which has a service to which the customerService reference of the MyValueServiceComponent is  
1175 wired as well as being promoted externally through the composite reference CustomerService.

---

## 5 Composite

1176

1177 An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of  
1178 composition within an SCA Domain. An **SCA composite** contains a set of components, services,  
1179 references and the wires that interconnect them, plus a set of properties which can be used to  
1180 configure components.

1181 Composites can be used as **component implementations** in higher-level composites – in other  
1182 words the higher-level composites can have components that are implemented by composites.  
1183 For more detail on the use of composites as component implementations see the section [Using](#)  
1184 [Composites as Component Implementations](#).

1185 The content of a composite can be used within another composite through **inclusion**. When a  
1186 composite is included by another composite, all of its contents are made available for use within  
1187 the including composite – the contents are fully visible and can be referenced by other elements  
1188 within the including composite. For more detail on the inclusion of one composite into another see  
1189 the section [Using Composites through Inclusion](#).

1190 A composite can be used as a unit of deployment. When used in this way, composites contribute  
1191 components and wires to an SCA Domain. A composite can be deployed to the SCA Domain either  
1192 by inclusion, or a composite can be deployed to the Domain as an implementation. For more  
1193 detail on the deployment of composites, see the section dealing with the [SCA Domain](#).

1194

1195 A composite is defined in an **xxx.composite** file. A composite is represented by a **composite**  
1196 element. The following snippet shows the schema for the composite element.

```
1197 <?xml version="1.0" encoding="ASCII"?>  
1198 <!-- Composite schema snippet -->  
1199 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
1200     targetNamespace="xs:anyURI"  
1201     name="xs:NCName" local="xs:boolean"?  
1202     autowire="xs:boolean"? constrainingType="xs:QName"?  
1203     requires="list of xs:QName"? policySets="list of xs:QName"?>  
1204  
1205     <include ... />*  
1206  
1207     <service ... />*  
1208     <reference ... />*  
1209     <property ... />*  
1210  
1211     <component ... />*  
1212  
1213     <wire ... />*  
1214  
1215 </composite>
```

1216

1217 The **composite** element has the following **attributes**:

- 1218 • **name : NCName (1..1)** – the name of the composite. The form of a composite name is  
1219 an XML QName, in the namespace identified by the @targetNamespace attribute. **A**  
1220 **composite @name attribute value MUST be unique within the namespace of the**  
1221 **composite.** [ASM60001]
- 1222 • **targetNamespace : anyURI (0..1)** – an identifier for a target namespace into which the  
1223 composite is declared
- 1224 • **local : boolean (0..1)** – whether all the components within the composite all run in the  
1225 same operating system process. **@local="true" for a composite means that all the**

- 1226 components within the composite MUST run in the same operating system process.  
 1227 [ASM60002] local="false", which is the default, means that different components within  
 1228 the composite can run in different operating system processes and they can even run on  
 1229 different nodes on a network.
- 1230 • **autowire : boolean (0..1)** – whether contained component references are autowired, as  
 1231 described in [the Autowire section](#). Default is false.
  - 1232 • **constrainingType : QName (0..1)** – the name of a constrainingType. When specified,  
 1233 the set of services, references and properties of the composite, plus related intents, is  
 1234 constrained to the set defined by the constrainingType. See [the ConstrainingType Section](#)  
 1235 for more details.
  - 1236 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework](#)  
 1237 [specification \[10\]](#) for a description of this attribute.
  - 1238 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
 1239 [\[10\]](#) for a description of this attribute.

1240

1241 The **composite** element has the following **child elements**:

- 1242 • **service : CompositeService (0..n)** – see composite service section.
- 1243 • **reference : CompositeReference (0..n)** – see composite reference section.
- 1244 • **property : CompositeProperty (0..n)** – see composite property section.
- 1245 • **component : Component (0..n)** – see component section.
- 1246 • **wire : Wire (0..n)** – see composite wire section.
- 1247 • **include : Include (0..n)** – see composite include section

1248

1249 Components contain configured implementations which hold the business logic of the composite.  
 1250 The components offer services and use references to other services. **Composite services** define  
 1251 the public services provided by the composite, which can be accessed from outside the composite.  
 1252 **Composite references** represent dependencies which the composite has on services provided  
 1253 elsewhere, outside the composite. Wires describe the connections between component services  
 1254 and component references within the composite. Included composites contribute the elements  
 1255 they contain to the using composite.

1256 Composite services involve the **promotion** of one service of one of the components within the  
 1257 composite, which means that the composite service is actually provided by one of the components  
 1258 within the composite. Composite references involve the **promotion** of one or more references of  
 1259 one or more components. Multiple component references can be promoted to the same composite  
 1260 reference, as long as all the component references are compatible with one another. Where  
 1261 multiple component references are promoted to the same composite reference, then they all share  
 1262 the same configuration, including the same target service(s).

1263 Composite services and composite references can use the configuration of their promoted services  
 1264 and references respectively (such as Bindings and Policy Sets). Alternatively composite services  
 1265 and composite references can override some or all of the configuration of the promoted services  
 1266 and references, through the configuration of bindings and other aspects of the composite service  
 1267 or reference.

1268 Component services and component references can be promoted to composite services and  
 1269 references and also be wired internally within the composite at the same time. For a reference,  
 1270 this only makes sense if the reference supports a multiplicity greater than 1.

1271

## 1272 5.1 Service

1273 The **services of a composite** are defined by promoting services defined by components  
1274 contained in the composite. A component service is promoted by means of a composite **service**  
1275 **element**.

1276 A composite service is represented by a **service element** which is a child of the composite  
1277 element. There can be **zero or more** service elements in a composite. The following snippet  
1278 shows the pseudo-schema for a service child element:

```
1279 <?xml version="1.0" encoding="ASCII"?>
1280 <!-- Composite Service schema snippet -->
1281 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
1282   ...
1283   <service name="xs:NCName" promote="xs:anyURI"
1284     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1285     <interface ... />?
1286     <binding ... />*
1287     <callback?
1288       <binding ... />+
1289     </callback>
1290   </service>
1291   ...
1292 </composite>
1293
```

1294 The **composite service** element has the following **attributes**:

- 1295 • **name : NCName (1..1)** – the name of the service. The name of a composite `<service/>`  
1296 **element MUST be unique across all the composite services in the composite.** [ASM60003]  
1297 The name of the composite service can be different from the name of the promoted  
1298 component service.
- 1299 • **promote : anyURI (1..1)** – identifies the promoted service, the value is of the form  
1300 `<component-name>/<service-name>`. The service name can be omitted if the target  
1301 component only has one service. The same component service can be promoted by more  
1302 than one composite service. A composite `<service/>` element's `@promote` attribute **MUST**  
1303 **identify one of the component services within that composite.** [ASM60004] `<include/>`  
1304 **processing MUST take place before the processing of the `@promote` attribute of a**  
1305 **composite service is performed.** [ASM60038]
- 1306 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
1307 [\[10\]](#) for a description of this attribute. Specified intents add to or further qualify the  
1308 required intents defined by the promoted component service.
- 1309 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
1310 [\[10\]](#) for a description of this attribute.

1311

1312 The **composite service** element has the following **child elements**, whatever is not specified is  
1313 defaulted from the promoted component service.

- 1314 • **interface : Interface (0..1)** - an interface which describes the operations provided by the  
1315 composite service. If a composite service **interface** is specified it **MUST be the same or a**  
1316 **compatible subset of the interface provided by the promoted component service, i.e.**  
1317 **provide a subset of the operations defined by the component service.** [ASM60005] The  
1318 interface is described by **zero or one interface element** which is a child element of the  
1319 service element. For details on the interface element see [the Interface section](#).
- 1320 • **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined  
1321 for the promoted component service from the composite service perspective. The bindings  
1322 defined on the component service are still in effect for local wires within the composite  
1323 that target the component service. A service element has zero or more **binding elements**

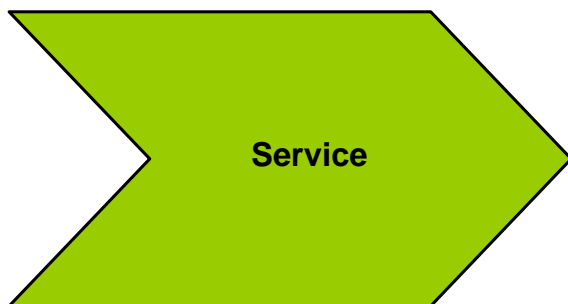
1324 as children. Details of the binding element are described in the [Bindings section](#). For more  
1325 details on wiring see the [Wiring section](#).

- 1326 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface  
1327 has a callback defined and the callback has one or more **binding** elements as  
1328 subelements. The **callback** and its binding subelements are specified if there is a need to  
1329 have binding details used to handle callbacks. If the callback element is not present, the  
1330 behaviour is runtime implementation dependent.

1331

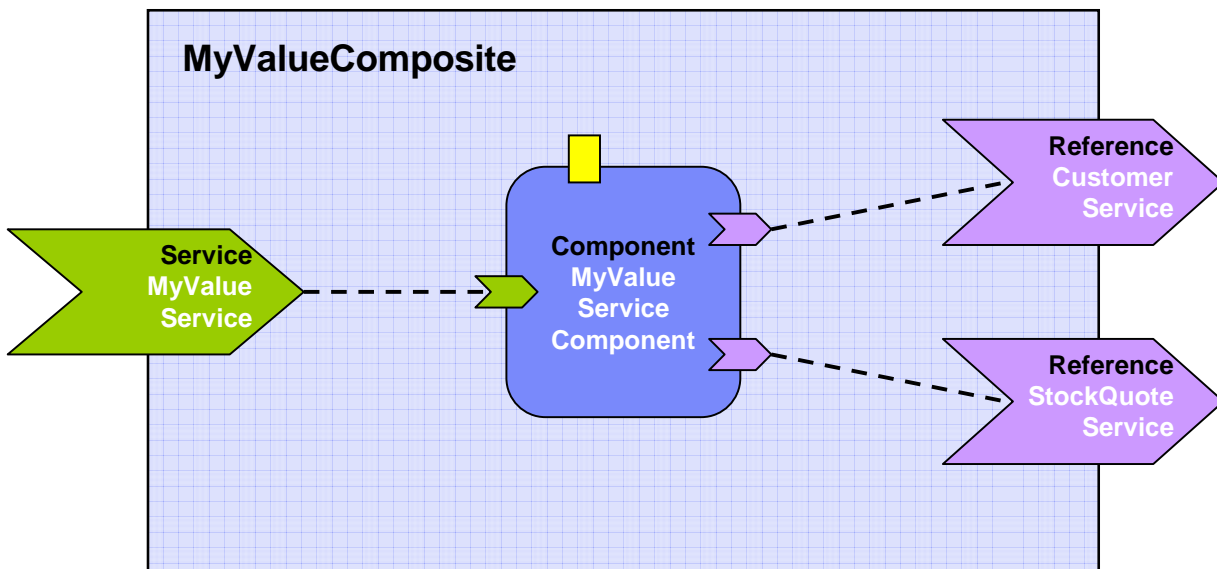
### 1332 5.1.1 Service Examples

1333 The following figure shows the service symbol that used to represent a service in an assembly  
1334 diagram:



1335  
1336 *Figure 7: Service symbol*

1337  
1338 The following figure shows the assembly diagram for the MyValueComposite containing the service  
1339 MyValueService.



1340  
1341 *Figure 8: MyValueComposite showing Service*

1342  
1343 The following snippet shows the MyValueComposite.composite file for the MyValueComposite  
1344 containing the service element for the MyValueService, which is a promote of the service offered  
1345 by the MyValueServiceComponent. The name of the promoted service is omitted since  
1346 MyValueServiceComponent offers only one service. The composite service MyValueService is  
1347 bound using a Web service binding.

```

1348 <?xml version="1.0" encoding="ASCII"?>
1349 <!-- MyValueComposite_4 example -->
1350 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1351           targetNamespace="http://foo.com"
1352           name="MyValueComposite" >
1353
1354     ...
1355
1356     <service name="MyValueService" promote="MyValueServiceComponent">
1357       <interface.java interface="services.myvalue.MyValueService"/>
1358       <binding.ws port="http://www.myvalue.org/MyValueService#
1359         wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1360     </service>
1361
1362     <component name="MyValueServiceComponent">
1363       <implementation.java
1364         class="services.myvalue.MyValueServiceImpl"/>
1365       <property name="currency">EURO</property>
1366       <service name="MyValueService"/>
1367       <reference name="customerService"/>
1368       <reference name="stockQuoteService"/>
1369     </component>
1370
1371     ...
1372
1373 </composite>
1374

```

## 5.2 Reference

1375 The *references of a composite* are defined by *promoting* references defined by components  
 1376 contained in the composite. Each promoted reference indicates that the component reference  
 1377 needs to be resolved by services outside the composite. A component reference is promoted using  
 1378 a composite *reference element*.  
 1379

1380 A composite reference is represented by a *reference element* which is a child of a composite  
 1381 element. There can be *zero or more reference* elements in a composite. The following snippet  
 1382 shows the composite schema with the schema for a *reference* element.

```

1383 <?xml version="1.0" encoding="ASCII"?>
1384 <!-- Composite Reference schema snippet -->
1385 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
1386   ...
1387   <reference name="xs:NCName" target="list of xs:anyURI"?
1388     promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
1389     multiplicity="0..1 or 1..1 or 0..n or 1..n"?
1390     requires="list of xs:QName"? policySets="list of xs:QName"?>*
1391     <interface ... />?
1392     <binding ... />*
1393     <callback?
1394       <binding ... />+
1395     </callback>
1396   </reference>
1397   ...
1398 </composite>
1399

```

1400 The *composite reference* element has the following *attributes*:

- 1401 • **name : NCName (1..1)** – the name of the reference. The name of a composite  
 1402 <reference/> element MUST be unique across all the composite references in the  
 1403 composite. [ASM60006] The name of the composite reference can be different than the  
 1404 name of the promoted component reference.
- 1405 • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
 1406 value is a list of values of the form <component-name>/<reference-name> separated by  
 1407 spaces. The reference name can be omitted if the component has only one reference.  
 1408 Each of the URIs declared by a composite reference's @promote attribute MUST identify a  
 1409 component reference within the composite. [ASM60007] <include/> processing MUST  
 1410 take place before the processing of the @promote attribute of a composite reference is  
 1411 performed. [ASM60037]
- 1412 The same component reference can be promoted more than once, using different  
 1413 composite references, but only if the multiplicity defined on the component reference is  
 1414 0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.
- 1415 Where a composite reference promotes two or more component references:
- 1416 • the interfaces of the component references promoted by a composite reference  
 1417 MUST be the same, or if the composite reference itself declares an interface then  
 1418 all the component reference interfaces MUST be compatible with the composite  
 1419 reference interface. Compatible means that the component reference interface is  
 1420 the same or is a strict subset of the composite reference interface. [ASM60008]
  - 1421 • the intents declared on a composite reference and on the component references  
 1422 which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents  
 1423 which apply to the composite reference in this case are the union of the intents  
 1424 specified for each of the promoted component references plus any intents declared  
 1425 on the composite reference itself. If any intents in the set which apply to a  
 1426 composite reference are mutually exclusive then the SCA runtime MUST raise an  
 1427 error. [ASM60010]
- 1428 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
 1429 [10] for a description of this attribute. Specified intents add to or further qualify the  
 1430 intents defined for the promoted component reference.
- 1431 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
 1432 [10] for a description of this attribute.
- 1433 • **multiplicity : (0..1)** - Defines the number of wires that can connect the reference to  
 1434 target services. When present, the multiplicity can have one of the following values
- 1435 ○ 0..1 – zero or one wire can have the reference as a source
  - 1436 ○ 1..1 – one wire can have the reference as a source
  - 1437 ○ 0..n - zero or more wires can have the reference as a source
  - 1438 ○ 1..n – one or more wires can have the reference as a source
- 1439 The default value for the @multiplicity attribute is 1..1.
- 1440 The value specified for the @multiplicity attribute of a composite reference MUST be  
 1441 compatible with the multiplicity specified on each of the promoted component references,  
 1442 i.e. the multiplicity has to be equal or further restrict. So multiplicity 0..1 can be used  
 1443 where the promoted component reference has multiplicity 0..n, multiplicity 1..1 can be  
 1444 used where the promoted component reference has multiplicity 0..n or 1..n and  
 1445 multiplicity 1..n can be used where the promoted component reference has multiplicity  
 1446 0..n., However, a composite reference of multiplicity 0..n or 1..n cannot be used to  
 1447 promote a component reference of multiplicity 0..1 or 1..1 respectively. [ASM60011]
- 1448
- 1449 • **target : anyURI (0..n)** – a list of one or more of target service URI's, depending on  
 1450 multiplicity setting. Each value wires the reference to a service in a composite that uses  
 1451 the composite containing the reference as an implementation for one of its components. For  
 1452 more details on wiring see [the section on Wires](#).



- 1453 • **wiredByImpl : boolean (0..1)** – a boolean value. If set to "true" it indicates that the  
 1454 target of the reference is set at runtime by the implementation code (for example by the  
 1455 code obtaining an endpoint reference by some means and setting this as the target of the  
 1456 reference through the use of programming interfaces defined by the relevant Client and  
 1457 Implementation specification). If "true" is set, then the reference is not intended to be  
 1458 wired statically within a using composite, but left unwired.  
 1459 All the component references promoted by a single composite reference MUST have the  
 1460 same value for @wiredByImpl. [ASM60035] If the @wiredByImpl attribute is not specified  
 1461 on the composite reference, the default value is "true" if all of the promoted component  
 1462 references have a wiredByImpl value of "true", and the default value is "false" if all the  
 1463 promoted component references have a wiredByImpl value of "false". If the @wiredByImpl  
 1464 attribute is specified, its value MUST be "true" if all of the promoted component references  
 1465 have a wiredByImpl value of "true", and its value MUST be "false" if all the promoted  
 1466 component references have a wiredByImpl value of "false". [ASM60036]

1467  
 1468 The **composite reference** element has the following **child elements**, whatever is not specified is  
 1469 defaulted from the promoted component reference(s).

- 1470 • **interface : Interface (0..1) - zero or one interface element** which declares an  
 1471 interface for the composite reference. If a composite reference has an **interface** specified,  
 1472 it MUST provide an interface which is the same or which is a compatible superset of the  
 1473 interface(s) declared by the promoted component reference(s), i.e. provide a superset of  
 1474 the operations in the interface defined by the component for the reference. [ASM60012] If  
 1475 no interface is declared on a composite reference, the interface from one of its promoted  
 1476 component references is used, which MUST be the same as or a compatible superset of  
 1477 the interface(s) declared by the promoted component reference(s).  
 1478 [ASM60013] For details on the interface element see [the Interface section](#).

- 1479 • **binding : Binding (0..n)** - A reference element has zero or more **binding elements** as  
 1480 children. If one or more **bindings** are specified they **override** any and all of the bindings  
 1481 defined for the promoted component reference from the composite reference perspective.  
 1482 The bindings defined on the component reference are still in effect for local wires within  
 1483 the composite that have the component reference as their source. Details of the binding  
 1484 element are described in the [Bindings section](#). For more details on wiring see [the section](#)  
 1485 [on Wires](#).

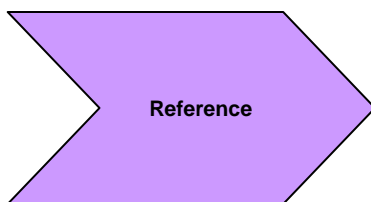
1486 A reference identifies zero or more target services which satisfy the reference. This can be  
 1487 done in a number of ways, which are fully described in section "[Specifying the Target](#)  
 1488 [Service\(s\) for a Reference](#)".

- 1489 • **callback (0..1) / binding : Binding (1..n)** - A **callback** element is used if the interface  
 1490 has a callback defined and the callback element has one or more **binding** elements as  
 1491 subelements. The **callback** and its binding subelements are specified if there is a need to  
 1492 have binding details used to handle callbacks. If the callback element is not present, the  
 1493 behaviour is runtime implementation dependent.

1494

## 1495 5.2.1 Example Reference

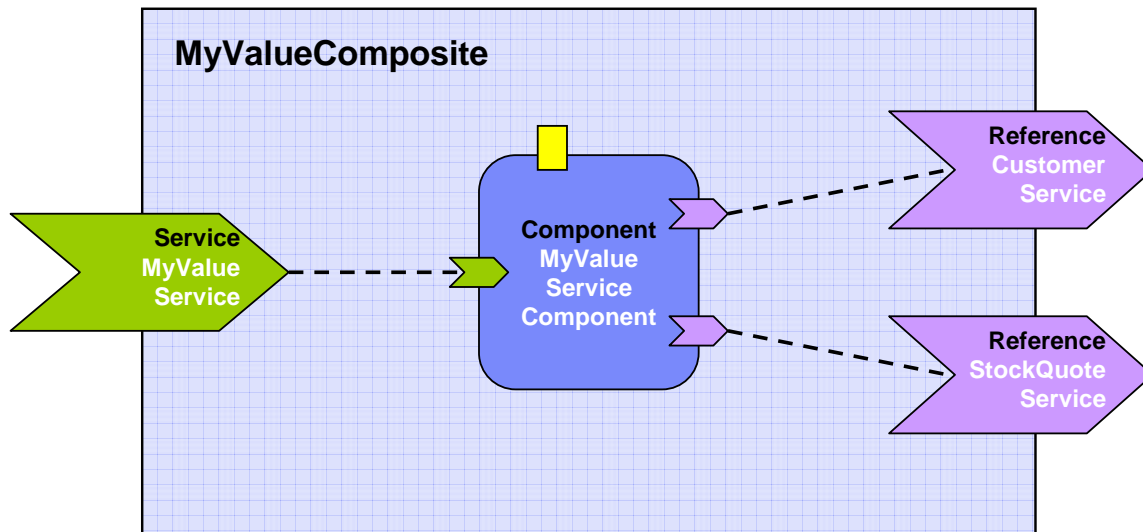
1496 The following figure shows the reference symbol that is used to represent a reference in an  
 1497 assembly diagram.



1498  
 1499 *Figure 9: Reference symbol*

1500  
1501  
1502  
1503

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



1504  
1505  
1506

Figure 10: MyValueComposite showing References

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

```
1515 <?xml version="1.0" encoding="ASCII"?>
1516 <!-- MyValueComposite_3 example -->
1517 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1518           targetNamespace="http://foo.com"
1519           name="MyValueComposite" >
1520
1521     ...
1522
1523     <component name="MyValueServiceComponent">
1524       <implementation.java
1525         class="services.myvalue.MyValueServiceImpl"/>
1526       <property name="currency">EURO</property>
1527       <reference name="customerService"/>
1528       <reference name="stockQuoteService"/>
1529     </component>
1530
1531     <reference name="CustomerService"
1532       promote="MyValueServiceComponent/customerService">
1533       <interface.java interface="services.customer.CustomerService"/>
1534       <!-- The following forces the binding to be binding.sca -->
1535       <!-- whatever is specified by the component reference or -->
1536       <!-- by the underlying implementation -->
1537       <binding.sca/>
```

```

1538     </reference>
1539
1540     <reference name="StockQuoteService"
1541             promote="MyValueServiceComponent/stockQuoteService">
1542         <interface.java
1543             interface="services.stockquote.StockQuoteService"/>
1544         <binding.ws port="http://www.stockquote.org/StockQuoteService#
1545             wsdl.endpoint(StockQuoteService/StockQuoteServicesSOAP)"/>
1546     </reference>
1547
1548     ...
1549
1550 </composite>
1551

```

## 1552 5.3 Property

1553 **Properties** allow for the configuration of an implementation with externally set data values. A  
 1554 composite can declare zero or more properties. Each property has a type, which is either simple  
 1555 or complex. An implementation can also define a default value for a property. Properties can be  
 1556 configured with values in the components that use the implementation.

1557 The declaration of a property in a composite follows the form described in the following schema  
 1558 snippet:

```

1559 <?xml version="1.0" encoding="ASCII"?>
1560 <!-- Composite Property schema snippet -->
1561 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903" ... >
1562     ...
1563     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
1564             requires="list of xs:QName"?
1565             policySets="list of xs:QName"?
1566             many="xs:boolean"? mustSupply="xs:boolean"?>*
1567         default-property-value?
1568     </property>
1569     ...
1570 </composite>
1571

```

1572 The **composite property** element has the following **attributes**:

- 1573 ▪ **name : NCName (1..1)** - the name of the property. The @name attribute of a composite  
 1574 property MUST be unique amongst the properties of the same composite. [ASM60014]
- 1575 ▪ one of (1..1):
  - 1576 ○ **type : QName** – the type of the property - the qualified name of an XML schema  
 1577 type
  - 1578 ○ **element : QName** – the type of the property defined as the qualified name of an  
 1579 XML schema global element – the type is the type of the global element
- 1580 ▪ **A single property element MUST NOT contain both a @type attribute and an @element  
 1581 attribute.** [ASM60040]
- 1582 ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued  
 1583 (true). The default is **false**. In the case of a multi-valued property, it is presented to the  
 1584 implementation as a collection of property values.
- 1585 ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the  
 1586 component that uses the composite – when mustSupply="true" the component has to  
 1587 supply a value since the composite has no default value for the property. A default-  
 1588 property-value is only worth declaring when mustSupply="false" (the default setting for

1589 the @mustSupply attribute), since the implication of a default value is that it is used only  
1590 when a value is not supplied by the using component.

1591 ▪ **requires** : *QName (0..n)* - a list of policy intents. See the [Policy Framework specification](#)  
1592 [\[10\]](#) for a description of this attribute.

1593 ▪ **policySets** : *QName (0..n)* - a list of policy sets. See the [Policy Framework specification](#)  
1594 [\[10\]](#) for a description of this attribute.

1595

1596 The property element can contain a **default-property-value**, which provides default value for the  
1597 property. The form of the default property value is as described [in the section on Component](#)  
1598 [Property](#).

1599

1600 Implementation types other than **composite** can declare properties in an implementation-  
1601 dependent form (e.g. annotations within a Java class), or through a property declaration of exactly  
1602 the form described above in a componentType file.

1603 Property values can be configured when an implementation is used by a component. The form of  
1604 the property configuration is shown in [the section on Components](#).

### 1605 5.3.1 Property Examples

1606 For the following example of Property declaration and value setting, the following complex type is  
1607 used as an example:

```
1608 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"  
1609             targetNamespace="http://foo.com/"  
1610             xmlns:tns="http://foo.com/">  
1611   <!-- ComplexProperty schema -->  
1612   <xsd:element name="fooElement" type="MyComplexType"/>  
1613   <xsd:complexType name="MyComplexType">  
1614     <xsd:sequence>  
1615       <xsd:element name="a" type="xsd:string"/>  
1616       <xsd:element name="b" type="anyURI"/>  
1617     </xsd:sequence>  
1618     <attribute name="attr" type="xsd:string" use="optional"/>  
1619   </xsd:complexType>  
1620 </xsd:schema>
```

1621

1622 The following composite demonstrates the declaration of a property of a complex type, with a  
1623 default value, plus it demonstrates the setting of a property value of a complex type within a  
1624 component:

```
1625 <?xml version="1.0" encoding="ASCII"?>  
1626 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
1627           xmlns:foo="http://foo.com"  
1628           targetNamespace="http://foo.com"  
1629           name="AccountServices">  
1630 <!-- AccountServices Example1 -->  
1631  
1632   ...  
1633  
1634   <property name="complexFoo" type="foo:MyComplexType">  
1635     <value>  
1636       <foo:a>AValue</foo:a>  
1637       <foo:b>InterestingURI</foo:b>  
1638     </value>  
1639   </property>
```

```

1640
1641     <component name="AccountServiceComponent">
1642         <implementation.java class="foo.AccountServiceImpl"/>
1643         <property name="complexBar" source="$complexFoo"/>
1644         <reference name="accountDataService"
1645             target="AccountDataServiceComponent"/>
1646         <reference name="stockQuoteService" target="StockQuoteService"/>
1647     </component>
1648
1649     ...
1650
1651 </composite>
1652

```

1653 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the  
1654 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the  
1655 composite and it references the example XSD, where **MyComplexType** is defined. The declaration  
1656 of **complexFoo** contains a default value. This is declared as the content of the property element.  
1657 In this example, the default value consists of the element **value** which is of type  
1658 **foo:MyComplexType** and it has two child elements **<foo:a>** and **<foo:b>**, following the definition  
1659 of **MyComplexType**.

1660 In the component **AccountServiceComponent**, the component sets the value of the property  
1661 **complexBar**, declared by the implementation configured by the component. In this case, the  
1662 type of **complexBar** is **foo:MyComplexType**. The example shows that the value of the **complexBar**  
1663 property is set from the value of the **complexFoo** property – the **@source** attribute of the property  
1664 element for **complexBar** declares that the value of the property is set from the value of a property  
1665 of the containing composite. The value of the **@source** attribute is **\$complexFoo**, where  
1666 **complexFoo** is the name of a property of the composite. This value implies that the whole of the  
1667 value of the source property is used to set the value of the component property.

1668 The following example illustrates the setting of the value of a property of a simple type (a string)  
1669 from **part** of the value of a property of the containing composite which has a complex type:

```

1670 <?xml version="1.0" encoding="ASCII"?>
1671 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1672     xmlns:foo="http://foo.com"
1673     targetNamespace="http://foo.com"
1674     name="AccountServices">
1675 <!-- AccountServices Example2 -->
1676
1677     ...
1678
1679     <property name="complexFoo" type="foo:MyComplexType">
1680         <value>
1681             <foo:a>AValue</foo:a>
1682             <foo:b>InterestingURI</foo:b>
1683         </value>
1684     </property>
1685
1686     <component name="AccountServiceComponent">
1687         <implementation.java class="foo.AccountServiceImpl"/>
1688         <property name="currency" source="$complexFoo/a"/>
1689         <reference name="accountDataService"
1690             target="AccountDataServiceComponent"/>
1691         <reference name="stockQuoteService" target="StockQuoteService"/>
1692     </component>
1693
1694     ...
1695
1696 </composite>

```

1697

1698 In this example, the component **AccountServiceComponent** sets the value of a property called  
1699 **currency**, which is of type string. The value is set from a property of the composite  
1700 **AccountServices** using the @source attribute set to **\$complexFoo/a**. This is an XPath  
1701 expression that selects the property name **complexFoo** and then selects the value of the **a**  
1702 subelement of the value of complexFoo. The "a" subelement is a string, matching the type of the  
1703 currency property.

1704 Further examples of declaring properties and setting property values in a component follow:

1705 Declaration of a property with a simple type and a default value:

```
1706 <property name="SimpleTypeProperty" type="xsd:string">  
1707 MyValue  
1708 </property>
```

1709

1710 Declaration of a property with a complex type and a default value:

```
1711 <property name="complexFoo" type="foo:MyComplexType">  
1712 <value>  
1713 <foo:a>AValue</foo:a>  
1714 <foo:b>InterestingURI</foo:b>  
1715 </value>  
1716 </property>
```

1717

1718 Declaration of a property with a global element type:

```
1719 <property name="elementFoo" element="foo:fooElement">  
1720 <foo:fooElement>  
1721 <foo:a>AValue</foo:a>  
1722 <foo:b>InterestingURI</foo:b>  
1723 </foo:fooElement>  
1724 </property>
```

1725

## 1726 5.4 Wire

1727 **SCA wires** within a composite connect **source component references** to **target component**  
1728 **services**.

1729 One way of defining a wire is by **configuring a reference of a component using its @target**  
1730 **attribute**. The reference element is configured with the wire-target-URI of the service(s) that  
1731 resolve the reference. Multiple target services are valid when the reference has a multiplicity of  
1732 0..n or 1..n.

1733 An alternative way of defining a Wire is by means of a **wire element** which is a child of the  
1734 composite element. There can be **zero or more** wire elements in a composite. This alternative  
1735 method for defining wires is useful in circumstances where separation of the wiring from the  
1736 elements the wires connect helps simplify development or operational activities. An example is  
1737 where the components used to build a Domain are relatively static but where new or changed  
1738 applications are created regularly from those components, through the creation of new assemblies  
1739 with different wiring. Deploying the wiring separately from the components allows the wiring to  
1740 be created or modified with minimum effort.

1741 Note that a Wire specified via a wire element is equivalent to a wire specified via the @target  
1742 attribute of a reference. The rule which forbids mixing of wires specified with the @target  
1743 attribute with the specification of endpoints in binding subelements of the reference also applies to  
1744 wires specified via separate wire elements.

1745 The following snippet shows the composite schema with the schema for the reference elements of  
1746 components and composite services and the wire child element:

```

1747
1748 <?xml version="1.0" encoding="ASCII"?>
1749 <!-- Wires schema snippet -->
1750 <composite ...>
1751     ...
1752     <wire source="xs:anyURI" target="xs:anyURI" replace="xs:boolean"?/>*
1753     ...
1754 </composite>
1755

```

1756 The **reference element of a component** has a list of one or more of the following **wire-target-**  
1757 **URI** values for the target, with multiple values separated by a space:

- 1758 • <component-name>/<service-name>
  - 1759 ○ where the target is a service of a component. The service name can be omitted if
  - 1760 the target component only has one service with a compatible interface

1761  
1762 The **wire element** has the following attributes:

- 1763 • **source (1..1)** – names the source component reference. Valid URI schemes are:
  - 1764 ○ <component-name>/<reference-name>
    - 1765 ▪ where the source is a component reference. The reference name can be
    - 1766 omitted if the source component only has one reference
- 1767 • **target (1..1)** – names the target component service. Valid URI schemes are
  - 1768 ○ <component-name>/<service-name>
    - 1769 ▪ where the target is a service of a component. The service name can be
    - 1770 omitted if the target component only has one service with a compatible
    - 1771 interface
- 1772 • **replace (0..1)** - a boolean value, with the default of "false". When a wire element has
- 1773 @replace="false", the wire is added to the set of wires which apply to the reference
- 1774 identified by the @source attribute. When a wire element has @replace="true", the wire
- 1775 is added to the set of wires which apply to the reference identified by the @source
- 1776 attribute - but any wires for that reference specified by means of the @target attribute of
- 1777 the reference are removed from the set of wires which apply to the reference.

1778  
1779 In other words, if any <wire/> element with @replace="true" is used for a particular  
1780 reference, the value of the @target attribute on the reference is ignored - and this permits  
1781 existing wires on the reference to be overridden by separate configuration, where the  
1782 reference is on a component at the Domain level.

1783 <include/> processing **MUST** take place before the @source and @target attributes of a wire are  
1784 resolved. [ASM60039]

1785 For a composite used as a component implementation, wires can only link sources and targets  
1786 that are contained in the same composite (irrespective of which file or files are used to describe  
1787 the composite). Wiring to entities outside the composite is done through services and references  
1788 of the composite with wiring defined by the next higher composite.

1789 A wire can only connect a source to a target if the target implements an interface that is  
1790 compatible with the interface declared by the source. The source and the target are compatible if  
1791 the target interface is a **compatible superset** of the source interface, defined as follows:

- 1792 1. the source interface and the target interface of a wire **MUST** either both be remotable or  
1793 else both be local [ASM60015]
- 1794 2. the operations on the target interface of a wire **MUST** be the same as or be a superset of  
1795 the operations in the interface specified on the source [ASM60016]

- 1796 3. compatibility between the source interface and the target interface for a wire for the  
1797 individual operations is defined as compatibility of the signature, that is operation name,  
1798 input types, and output types MUST be the same. [ASM60017]
- 1799 4. the order of the input and output types for operations in the source interface and the  
1800 target interface of a wire also MUST be the same. [ASM60018]
- 1801 5. the set of Faults and Exceptions expected by each operation in the source interface MUST  
1802 be the same or be a superset of those specified by the target interface. [ASM60019]

1803 If either the source interface of a wire or the target interface of a wire declares a callback interface  
1804 then both the source interface and the target interface MUST declare a callback interface and the  
1805 callback interface declared on the target MUST be a compatible superset of the callback interface  
1806 declared on the source. [ASM60020]

1807 A Wire can connect between different interface languages (e.g. Java interfaces and WSDL  
1808 portTypes) in either direction, as long as the operations defined by the two interface types are  
1809 equivalent. They are equivalent if the operation(s), parameter(s), return value(s) and  
1810 faults/exceptions map to each other.

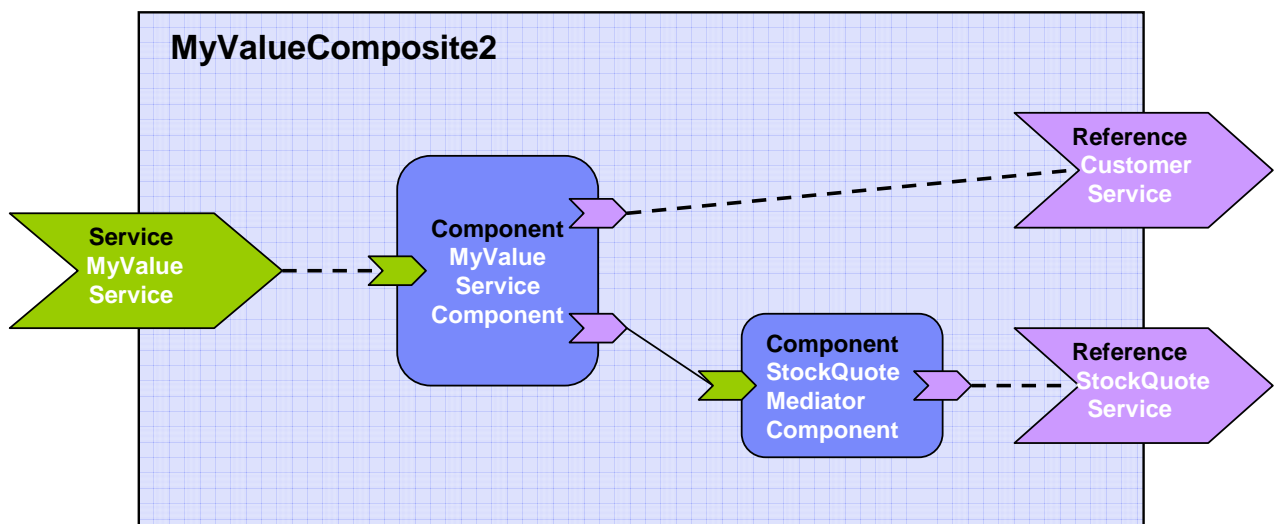
1811 Service clients cannot (portably) ask questions at runtime about additional interfaces that are  
1812 provided by the implementation of the service (e.g. the result of "instance of" in Java is non  
1813 portable). It is valid for an SCA implementation to have proxies for all wires, so that, for example,  
1814 a reference object passed to an implementation might only have the business interface of the  
1815 reference and might not be an instance of the (Java) class which is used to implement the target  
1816 service, even where the interface is local and the target service is running in the same process.

1817 **Note:** It is permitted to deploy a composite that has references that are not wired. For the case of  
1818 an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA  
1819 runtime SHOULD issue a warning. [ASM60021]

1820

## 1821 5.4.1 Wire Examples

1822 The following figure shows the assembly diagram for the MyValueComposite2 containing wires  
1823 between service, components and references.



1824  
1825 *Figure 11: MyValueComposite2 showing Wires*

1826  
1827 The following snippet shows the MyValueComposite2.composite file for the MyValueComposite2  
1828 containing the configured component and service references. The service MyValueService is wired  
1829 to the MyValueServiceComponent, using an explicit <wire/> element. The  
1830 MyValueServiceComponent's customerService reference is wired to the composite's



1831 CustomerService reference. The MyValueServiceComponent's stockQuoteService reference is  
1832 wired to the StockQuoteMediatorComponent, which in turn has its reference wired to the  
1833 StockQuoteService reference of the composite.

```
1834 <?xml version="1.0" encoding="ASCII"?>
1835 <!-- MyValueComposite Wires examples -->
1836 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1837           targetNamespace="http://foo.com"
1838           name="MyValueComposite2" >
1839
1840     <service name="MyValueService" promote="MyValueServiceComponent">
1841       <interface.java interface="services.myvalue.MyValueService"/>
1842       <binding.ws port="http://www.myvalue.org/MyValueService#
1843                 wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
1844     </service>
1845
1846     <component name="MyValueServiceComponent">
1847       <implementation.java
1848         class="services.myvalue.MyValueServiceImpl"/>
1849       <property name="currency">EURO</property>
1850       <service name="MyValueService"/>
1851       <reference name="customerService"/>
1852       <reference name="stockQuoteService"/>
1853     </component>
1854
1855     <wire source="MyValueServiceComponent/stockQuoteService"
1856           target="StockQuoteMediatorComponent"/>
1857
1858     <component name="StockQuoteMediatorComponent">
1859       <implementation.java class="services.myvalue.SQMediatorImpl"/>
1860       <property name="currency">EURO</property>
1861       <reference name="stockQuoteService"/>
1862     </component>
1863
1864     <reference name="CustomerService"
1865               promote="MyValueServiceComponent/customerService">
1866       <interface.java interface="services.customer.CustomerService"/>
1867       <binding.sca/>
1868     </reference>
1869
1870     <reference name="StockQuoteService"
1871               promote="StockQuoteMediatorComponent">
1872       <interface.java
1873         interface="services.stockquote.StockQuoteService"/>
1874       <binding.ws port="http://www.stockquote.org/StockQuoteService#
1875                 wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
1876     </reference>
1877
1878 </composite>
1879
```

## 1880 5.4.2 Autowire

1881 SCA provides a feature named **Autowire**, which can help to simplify the assembly of composites.  
1882 Autowire enables component references to be automatically wired to component services which  
1883 will satisfy those references, without the need to create explicit wires between the references and  
1884 the services. When the autowire feature is used, a component reference which is not promoted  
1885 and which is not explicitly wired to a service within a composite is automatically wired to a target

1886 service within the same composite. Autowire works by searching within the composite for a  
1887 service interface which matches the interface of the references.

1888 The autowire feature is not used by default. Autowire is enabled by the setting of an @autowire  
1889 attribute to "true". Autowire is disabled by setting of the @autowire attribute to "false" The  
1890 @autowire attribute can be applied to any of the following elements within a composite:

- 1891 • reference
- 1892 • component
- 1893 • composite

1894 Where an element does not have an explicit setting for the @autowire attribute, it inherits the  
1895 setting from its parent element. Thus a reference element inherits the setting from its containing  
1896 component. A component element inherits the setting from its containing composite. Where  
1897 there is no setting on any level, autowire="false" is the default.

1898 As an example, if a composite element has autowire="true" set, this means that autowiring is  
1899 enabled for all component references within that composite. In this example, autowiring can be  
1900 turned off for specific components and specific references through setting autowire="false" on the  
1901 components and references concerned.

1902 For each component reference for which autowire is enabled, the SCA runtime MUST search within  
1903 the composite for target services which are compatible with the reference. [ASM60022]

1904 "Compatible" here means:

- 1905 • the target service interface MUST be a compatible superset of the reference interface  
1906 when using autowire to wire a reference (as defined in the section on Wires) [ASM60023]
- 1907 • the intents, and policies applied to the service MUST be compatible with those on the  
1908 reference when using autowire to wire a reference – so that wiring the reference to the  
1909 service will not cause an error due to policy mismatch [ASM60024] (see the Policy  
1910 Framework specification [10] for details)

1911 If the search finds **1 or more** valid target service for a particular reference, the action taken  
1912 depends on the multiplicity of the reference:

- 1913 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
1914 reference to one of the set of valid target services chosen from the set in a runtime-  
1915 dependent fashion [ASM60025]
- 1916 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
1917 of the set of valid target services [ASM60026]

1918 If the search finds **no** valid target services for a particular reference, the action taken depends on  
1919 the multiplicity of the reference:

- 1920 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
1921 target service, there is no problem – no services are wired and the SCA runtime MUST  
1922 NOT raise an error [ASM60027]
- 1923 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
1924 target services an error MUST be raised by the SCA runtime since the reference is  
1925 intended to be wired [ASM60028]

1926

### 1927 5.4.3 Autowire Examples

1928 This example demonstrates two versions of the same composite – the first version is done using  
1929 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
1930 the end result is the same – the same wires connect the references to the services.

1931 First, here is a diagram for the composite:

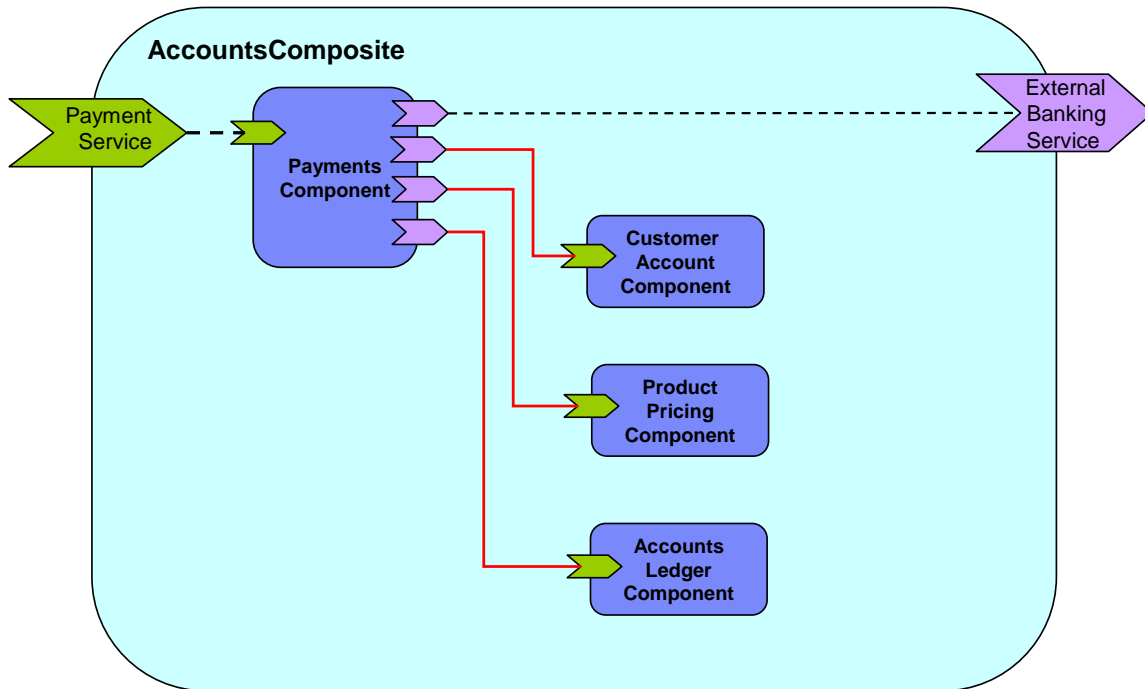


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

1935 <?xml version="1.0" encoding="UTF-8"?>
1936 <!-- Autowire Example - No autowire -->
1937 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1938           xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1939           xmlns:foo="http://foo.com"
1940           targetNamespace="http://foo.com"
1941           name="AccountComposite">
1942
1943     <service name="PaymentService" promote="PaymentsComponent" />
1944
1945     <component name="PaymentsComponent">
1946       <implementation.java class="com.foo.accounts.Payments" />
1947       <service name="PaymentService" />
1948       <reference name="CustomerAccountService"
1949                 target="CustomerAccountComponent" />
1950       <reference name="ProductPricingService"
1951                 target="ProductPricingComponent" />
1952       <reference name="AccountsLedgerService"
1953                 target="AccountsLedgerComponent" />
1954       <reference name="ExternalBankingService" />
1955     </component>
1956
1957     <component name="CustomerAccountComponent">
1958       <implementation.java class="com.foo.accounts.CustomerAccount" />
1959     </component>
1960
1961     <component name="ProductPricingComponent">
1962       <implementation.java class="com.foo.accounts.ProductPricing" />
1963     </component>
1964
1965     <component name="AccountsLedgerComponent">

```

```

1966     <implementation.composite name="foo:AccountsLedgerComposite"/>
1967 </component>
1968
1969     <reference name="ExternalBankingService"
1970         promote="PaymentsComponent/ExternalBankingService"/>
1971
1972 </composite>
1973

```

1974 Secondly, the composite using autowire:

```

1975 <?xml version="1.0" encoding="UTF-8"?>
1976 <!-- Autowire Example - With autowire -->
1977 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1978     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
1979     xmlns:foo="http://foo.com"
1980     targetNamespace="http://foo.com"
1981     name="AccountComposite">
1982
1983     <service name="PaymentService" promote="PaymentsComponent">
1984         <interface.java class="com.foo.PaymentServiceInterface"/>
1985     </service>
1986
1987     <component name="PaymentsComponent" autowire="true">
1988         <implementation.java class="com.foo.accounts.Payments"/>
1989         <service name="PaymentService"/>
1990         <reference name="CustomerAccountService"/>
1991         <reference name="ProductPricingService"/>
1992         <reference name="AccountsLedgerService"/>
1993         <reference name="ExternalBankingService"/>
1994     </component>
1995
1996     <component name="CustomerAccountComponent">
1997         <implementation.java class="com.foo.accounts.CustomerAccount"/>
1998     </component>
1999
2000     <component name="ProductPricingComponent">
2001         <implementation.java class="com.foo.accounts.ProductPricing"/>
2002     </component>
2003
2004     <component name="AccountsLedgerComponent">
2005         <implementation.composite name="foo:AccountsLedgerComposite"/>
2006     </component>
2007
2008     <reference name="ExternalBankingService"
2009         promote="PaymentsComponent/ExternalBankingService"/>
2010
2011 </composite>

```

2012 In this second case, autowire is set on for the PaymentsComponent and there are no explicit wires  
2013 for any of its references – the wires are created automatically through autowire.

2014 **Note:** In the second example, it would be possible to omit all of the service and reference  
2015 elements from the PaymentsComponent. They are left in for clarity, but if they are omitted, the  
2016 component service and references still exist, since they are provided by the implementation used  
2017 by the component.

2018

2019 **5.5 Using Composites as Component Implementations**

2020 Composites can be used as **component implementations** in higher-level composites – in other  
2021 words the higher-level composites can have components which are implemented by composites.

2022 When a composite is used as a component implementation, it defines a boundary of visibility.  
2023 Components within the composite cannot be referenced directly by the using component. The  
2024 using component can only connect wires to the services and references of the used composite and  
2025 set values for any properties of the composite. The internal construction of the composite is  
2026 invisible to the using component. The boundary of visibility, sometimes called encapsulation, can  
2027 be enforced when assembling components and composites, but such encapsulation structures  
2028 might not be enforceable in a particular implementation language.

2029 A composite used as a component implementation also needs to honor a completeness contract.  
2030 The services, references and properties of the composite form a contract (represented by the  
2031 component type of the composite) which is relied upon by the using component. The concept of  
2032 completeness of the composite implies that, once all <include/> element processing is performed  
2033 on the composite:

- 2034 1. For a composite used as a component implementation, each composite service offered by  
2035 the composite MUST promote a component service of a component that is within the  
2036 composite. [ASM60032]
- 2037 2. For a composite used as a component implementation, every component reference of  
2038 components within the composite with a multiplicity of 1..1 or 1..n MUST be wired or  
2039 promoted. [ASM60033] (according to the various rules for specifying target services for a  
2040 component reference described in the section " [Specifying the Target Service\(s\) for a  
2041 Reference](#)").
- 2042 3. For a composite used as a component implementation, all properties of components within  
2043 the composite, where the underlying component implementation specifies  
2044 "mustSupply=true" for the property, MUST either specify a value for the property or  
2045 source the value from a composite property. [ASM60034]

2046 The component type of a composite is defined by the set of composite service elements,  
2047 composite reference elements and composite property elements that are the children of the  
2048 composite element.

2049 Composites are used as component implementations through the use of the  
2050 **implementation.composite** element as a child element of the component. The schema snippet  
2051 for the implementation.composite element is:

```
2052 <!-- implementation.composite pseudo-schema -->  
2053 <implementation.composite name="xs:QName" requires="list of xs:QName"?  
2054 policySets="list of xs:QName"?>
```

2055  
2056 The implementation.composite element has the following attributes:

- 2057 • **name (1..1)** – the name of the composite used as an implementation. The @name  
2058 attribute of an <implementation.composite/> element MUST contain the QName of a  
2059 composite in the SCA Domain. [ASM60030]
- 2060 • **requires : QName (0..n)** – a list of policy intents. See the [Policy Framework specification](#)  
2061 [\[10\]](#) for a description of this attribute. Specified intents add to or further qualify the  
2062 required intents defined for the promoted component reference.
- 2063 • **policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification](#)  
2064 [\[10\]](#) for a description of this attribute.

2065

2066

## 5.5.1 Example of Composite used as a Component Implementation

2067  
2068  
2069

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

2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- CompositeComponent example -->
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200903
  file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
  targetNamespace="http://foo.com"
  xmlns:foo="http://foo.com"
  name="AccountComposite">

  <service name="AccountService" promote="AccountServiceComponent">
    <interface.java interface="services.account.AccountService"/>
    <binding.ws port="AccountService#
      wsdl.endpoint(AccountService/AccountServiceSOAP)"/>
  </service>

  <reference name="stockQuoteService"
    promote="AccountServiceComponent/StockQuoteService">
    <interface.java
      interface="services.stockquote.StockQuoteService"/>
    <binding.ws
      port="http://www.quickstockquote.com/StockQuoteService#
      wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
  </reference>

  <property name="currency" type="xsd:string">EURO</property>

  <component name="AccountServiceComponent">
    <implementation.composite name="foo:AccountServiceCompositel"/>

    <reference name="AccountDataService" target="AccountDataService"/>
    <reference name="StockQuoteService"/>

    <property name="currency" source="$currency"/>
  </component>

  <component name="AccountDataService">
    <implementation.composite name="foo:AccountDataServiceComposite"/>

    <property name="currency" source="$currency"/>
  </component>

</composite>
```

2114

## 5.6 Using Composites through Inclusion

2115  
2116

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

2117  
2118  
2119

A composite can include another composite by using the **include** element. This provides a recursive inclusion capability. The semantics of included composites are that the element content children of the included composite are inlined, with certain modification, into the using composite.

2120 This is done recursively till the resulting composite does not contain an **include** element. The  
2121 outer included composite element itself is discarded in this process – only its contents are included  
2122 as described below:

- 2123 1. All the element content children of the included composite are inlined in the including  
2124 composite.
- 2125 2. The attributes **@targetNamespace**, **@name**, **@constrainingType**, and **@local** of the  
2126 included composites are discarded.
- 2127 3. All the namespace declaration on the included composite element are added to the inlined  
2128 element content children unless the namespace binding is overridden by the element  
2129 content children.
- 2130 4. The attribute **@autowire**, if specified on the included composite, is included on all inlined  
2131 component element children unless the component child already specifies that attribute.
- 2132 5. The attribute values of **@requires** and **@policySet**, if specified on the included  
2133 composite, are merged with corresponding attribute on the inlined component, service and  
2134 reference children elements. Merge in this context means a set union.
- 2135 6. Extension attributes ,if present on the included composite, follow the rules defined for that  
2136 extension. Authors of attribute extensions on the composite element define the rules  
2137 applying to those attributes for inclusion.

2138 If the included composite has the value *true* for the attribute **@local** then the including composite  
2139 MUST have the same value for the **@local** attribute, else it is an error. [ASM60041]

2140 The composite file used for inclusion can have any contents The composite element can contain  
2141 any of the elements which are valid as child elements of a composite element, namely  
2142 components, services, references, wires and includes. There is no need for the content of an  
2143 included composite to be complete, so that artifacts defined within the using composite or in  
2144 another associated included composite file can be referenced. For example, it is permissible to  
2145 have two components in one composite file while a wire specifying one component as the source  
2146 and the other as the target can be defined in a second included composite file.

2147 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one  
2148 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated  
2149 elements in the using composite (e.g. two services with the same uri contributed by different  
2150 included composites). It is not considered an error if the (using) composite resulting from the  
2151 inclusion is incomplete (eg. wires with non-existent source or target). Such incomplete resulting  
2152 composites are permitted to allow recursive composition.

2153 The following snippet shows the pseudo-schema for the include element.

```
2154 <?xml version="1.0" encoding="UTF-8"?>  
2155 <!-- Include snippet -->  
2156 <composite ...>  
2157   ...  
2158   <include name="xs:QName" /> *  
2159   ...  
2160 </composite>  
2161
```

2162 The include element has the following **attribute**:

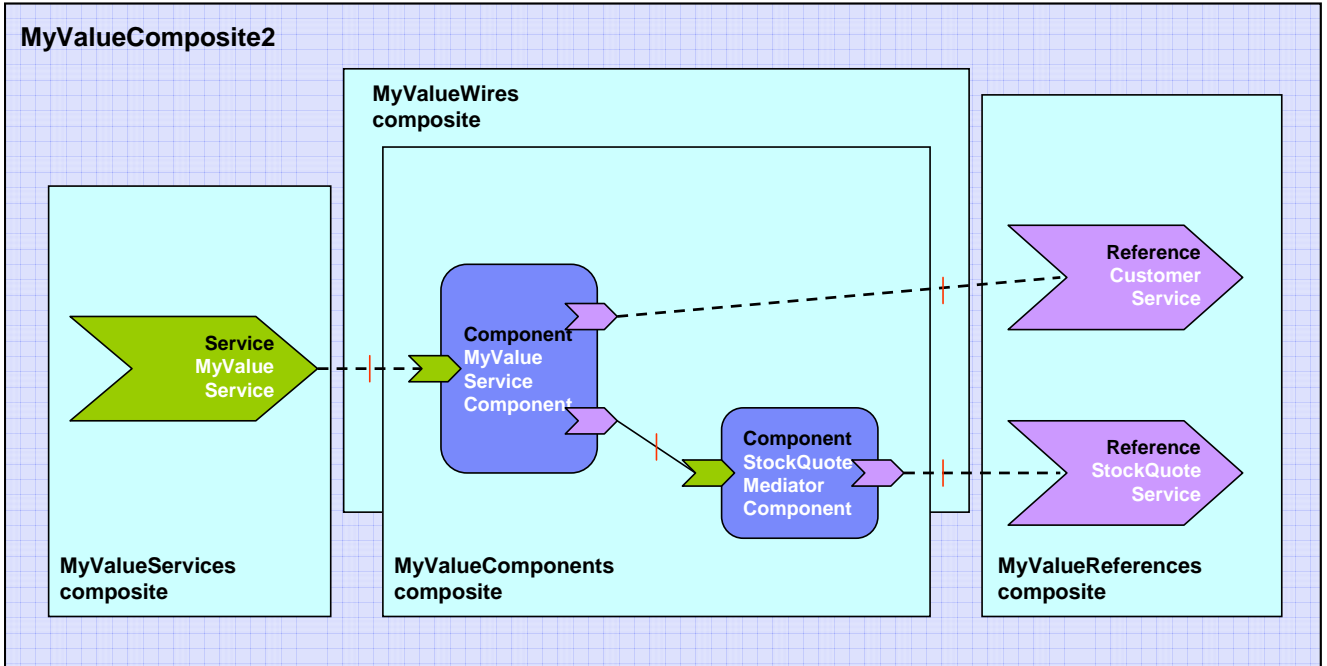
- 2163 • **name: QName (1..1)** – the name of the composite that is included. The **@name** attribute  
2164 of an include element MUST be the QName of a composite in the SCA Domain.  
2165 [ASM60042]

2166

## 2167 5.6.1 Included Composite Examples

2168 The following figure shows the assembly diagram for the MyValueComposite2 containing four  
2169 included composites. The **MyValueServices composite** contains the MyValueService service. The

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



2179  
 2180  
 2181 *Figure 13 MyValueComposite2 built from 4 included composites*

2182  
 2183 The following snippet shows the contents of the MyValueComposite2.composite file for the  
 2184 MyValueComposite2 built using included composites. In this sample it only provides the name of  
 2185 the composite. The composite file itself could be used in a scenario using included composites to  
 2186 define components, services, references and wires.

```
2187 <?xml version="1.0" encoding="ASCII"?>
2188 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2189           targetNamespace="http://foo.com"
2190           xmlns:foo="http://foo.com"
2191           name="MyValueComposite2" >
2192
2193     <include name="foo:MyValueServices" />
2194     <include name="foo:MyValueComponents" />
2195     <include name="foo:MyValueReferences" />
2196     <include name="foo:MyValueWires" />
2197
2198 </composite>
```

2199  
 2200 The following snippet shows the content of the MyValueServices.composite file.

```
2201 <?xml version="1.0" encoding="ASCII"?>
2202 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
```



```

2203         targetNamespace="http://foo.com"
2204         xmlns:foo="http://foo.com"
2205         name="MyValueServices" >
2206
2207         <service name="MyValueService" promote="MyValueServiceComponent">
2208             <interface.java interface="services.myvalue.MyValueService"/>
2209             <binding.ws port="http://www.myvalue.org/MyValueService#"
2210                 wsdl.endpoint(MyValueService/MyValueServicesSOAP)"/>
2211         </service>
2212
2213 </composite>
2214

```

2215 The following snippet shows the content of the MyValueComponents.composite file.

```

2216 <?xml version="1.0" encoding="ASCII"?>
2217 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2218     targetNamespace="http://foo.com"
2219     xmlns:foo="http://foo.com"
2220     name="MyValueComponents" >
2221
2222     <component name="MyValueServiceComponent">
2223         <implementation.java
2224             class="services.myvalue.MyValueServiceImpl"/>
2225         <property name="currency">EURO</property>
2226     </component>
2227
2228     <component name="StockQuoteMediatorComponent">
2229         <implementation.java class="services.myvalue.SQMediatorImpl"/>
2230         <property name="currency">EURO</property>
2231     </component>
2232
2233 </composite>
2234

```

2235 The following snippet shows the content of the MyValueReferences.composite file.

```

2236 <?xml version="1.0" encoding="ASCII"?>
2237 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2238     targetNamespace="http://foo.com"
2239     xmlns:foo="http://foo.com"
2240     name="MyValueReferences" >
2241
2242     <reference name="CustomerService"
2243         promote="MyValueServiceComponent/CustomerService">
2244         <interface.java interface="services.customer.CustomerService"/>
2245         <binding.sca/>
2246     </reference>
2247
2248     <reference name="StockQuoteService"
2249         promote="StockQuoteMediatorComponent">
2250         <interface.java
2251             interface="services.stockquote.StockQuoteService"/>
2252         <binding.ws port="http://www.stockquote.org/StockQuoteService#"
2253             wsdl.endpoint(StockQuoteService/StockQuoteServicesSOAP)"/>
2254     </reference>
2255
2256 </composite>
2257

```

2258 The following snippet shows the content of the MyValueWires.composite file.

```
2259 <?xml version="1.0" encoding="ASCII"?>
2260 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2261           targetNamespace="http://foo.com"
2262           xmlns:foo="http://foo.com"
2263           name="MyValueWires" >
2264
2265     <wire source="MyValueServiceComponent/stockQuoteService"
2266           target="StockQuoteMediatorComponent" />
2267
2268 </composite>
```

## 2269 5.7 Composites which Contain Component Implementations of 2270 Multiple Types

2271 A Composite containing multiple components can have multiple component implementation types.  
2272 For example, a Composite can contain one component with a Java POJO as its implementation and  
2273 another component with a BPEL process as its implementation.

## 2274 5.8 Structural URI of Components

2275 The **structural URI** is a relative URI that describes each use of a given component in the Domain,  
2276 relative to the URI of the Domain itself. It is never specified explicitly, but it calculated from the  
2277 configuration of the components configured into the Domain.

2278 A component in a composite can be used more than once in the Domain, if its containing  
2279 composite is used as the implementation of more than one higher-level component. The structural  
2280 URI is used to separately identify each use of a component - for example, the structural URI can  
2281 be used to attach different policies to each separate use of a component.

2282 For components directly deployed into the Domain, the structural URI is simply the name of the  
2283 component.

2284 Where components are nested within a composite which is used as the implementation of a higher  
2285 level component, the structural URI consists of the name of the nested component prepended with  
2286 each of the names of the components upto and including the Domain level component.

2287 For example, consider a component named Component1 at the Domain level, where its  
2288 implementation is Composite1 which in turn contains a component named Component2, which is  
2289 implemented by Composite2 which contains a component named Component3. The three  
2290 components in this example have the following structural URIs:

- 2291 1. Component1: Component1
- 2292 2. Component2: Component1/Component2
- 2293 3. Component3: Component1/Component2/Component3

2294 The structural URI can also be extended to refer to specific parts of a component, such as a  
2295 service or a reference, by appending an appropriate fragment identifier to the component's  
2296 structural URI, as follows:

- 2297 • Service:  
2298 #service(servicename)  
2299
- 2300 • Reference:  
2301 #reference(referencename)  
2302
- 2303 • Service binding:  
2304 #service-binding(servicename/bindingname)  
2305

2306           • Reference binding:  
2307            #reference-binding(referencename/bindingname)

2308           So, for example, the structural URI of the service named "testservice" of component  
2309           "Component1" is Component1#service(testservice).

2310

2311

## 6 ConstrainingType

2312  
2313  
2314  
2315  
2316

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 necessary form of component implementations, before any of the implementations are developed.

2317  
2318  
2319  
2320  
2321  
2322  
2323

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, **constrainingType** does not contain bindings, policySets, property values or default wiring information. The **constrainingType** is applied to a component through a **@constrainingType** attribute on the component.

2324  
2325  
2326  
2327  
2328  
2329  
2330

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 provided 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**.

2331  
2332  
2333

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

2334  
2335

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

2336  
2337  
2338  
2339  
2340  
2341  
2342  
2343  
2344  
2345  
2346  
2347  
2348  
2349  
2350  
2351  
2352  
2353  
2354  
2355  
2356  
2357

```
<?xml version="1.0" encoding="ASCII"?>
<!-- ConstrainingType schema snippet -->
<constrainingType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
    targetNamespace="xs:anyURI"?
    name="xs:NCName">
    <service name="xs:NCName">*
        <interface ... />?
    </service>
    <reference name="xs:NCName"
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?>*
        <interface ... />?
    </reference>
    <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
        many="xs:boolean"? mustSupply="xs:boolean"?/>*
</constrainingType>
```

2358

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

2359  
2360  
2361  
2362

- **name (1..1)** – the name of the **constrainingType**. The form of a **constrainingType** name is an XML QName, in the namespace identified by the **@targetNamespace** attribute. **The @name attribute of the constraining type MUST be unique in the SCA Domain. [ASM70003]**

- 2363
- **targetNamespace (0..1)** – an identifier for a target namespace into which the
- 2364

2365 ConstrainingType contains **zero or more properties, services, references**.

2366 When an implementation is constrained by a constrainingType its component type MUST contain

2367 all the services, references and properties specified in the constrainingType. [ASM70004] The

2368 constraining type's references and services will have interfaces specified and can have intents

2369 specified. An implementation MAY contain additional services, additional references with

2370 @multiplicity=0..1 or @multiplicity=0..n and additional properties with @mustSupply=false

2371 beyond those declared in the constraining type, but MUST NOT contain additional references with

2372 @multiplicity=1..1 or @multiplicity=1..n or additional properties with @mustSupply=true

2373 [ASM70005]

2374 When a component is constrained by a constrainingType via the @constrainingType attribute, the

2375 entire componentType associated with the component and its implementation is not visible to the

2376 containing composite. The containing composite can only see a projection of the componentType

2377 associated with the component and implementation as scoped by the constrainingType of the

2378 component. Additional services, references and properties provided by the implementation which

2379 are not declared in the constrainingType associated with a component MUST NOT be configured in

2380 any way by the containing composite. [ASM70006] This requirement ensures that the

2381 constrainingType contract cannot be violated by the composite.

2382 A constrainingType can be applied to an implementation. In this case, the implementation's

2383 componentType has a @constrainingType attribute set to the QName of the constrainingType.

2384

## 2385 6.1 Example constrainingType

2386 The following snippet shows the contents of the component called "MyValueServiceComponent"

2387 which is constrained by the constrainingType myns:CT. The componentType associated with the

2388 implementation is also shown.

2389

```
2390 <component name="MyValueServiceComponent" constrainingType="myns:CT">
2391   <implementation.java class="services.myvalue.MyValueServiceImpl"/>
2392   <property name="currency">EURO</property>
2393   <reference name="customerService" target="CustomerService">
2394     <binding.ws ...>
2395   <reference name="stockQuoteService"
2396     target="StockQuoteMediatorComponent"/>
2397 </component>
2398
2399 <constrainingType name="CT"
2400   targetNamespace="http://myns.com">
2401   <service name="MyValueService">
2402     <interface.java interface="services.myvalue.MyValueService"/>
2403   </service>
2404   <reference name="customerService">
2405     <interface.java interface="services.customer.CustomerService"/>
2406   </reference>
2407   <reference name="stockQuoteService">
2408     <interface.java interface="services.stockquote.StockQuoteService"/>
2409   </reference>
2410   <property name="currency" type="xsd:string"/>
2411 </constrainingType>
2412
```

2413 The component MyValueServiceComponent is constrained by the constrainingType CT which

2414 means that it needs to provide:

- 2415 • service ***MyValueService*** with the interface `services.myvalue.MyValueService`
- 2416 • reference ***customerService*** with the interface `services.stockquote.StockQuoteService`
- 2417 • reference ***stockQuoteService*** with the interface `services.stockquote.StockQuoteService`
- 2418 • property ***currency*** of type `xsd:string`.

2419

## 7 Interface

2420

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

2421

2422

2423

2424

2425

2426

SCA currently supports the following interface type systems:

2427

- Java interfaces

2428

- WSDL 1.1 portTypes ([Web Services Definition Language \[8\]](#))

2429

- C++ classes

2430

- Collections of 'C' functions

2431

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

2432

2433

2434

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

2435

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

2436

2437

2438

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

2439

- **remotable : boolean (0..1)** – indicates whether an interface is remotable or not (see **Error! Reference source not found.**). A value of “true” means the interface is remotable, and a value of “false” means it is not. The @remotable attribute has no default value. This attribute is used as an alternative to interface type specific mechanisms such as the @Remotable annotation on a Java interface. The remotable nature of an interface in the absence of this attribute is interface type specific. The rules governing how this attribute relates to interface type specific mechanisms are defined by each interface type. When specified on an interface definition which includes a callback, this attribute also applies to the callback interface (see **Error! Reference source not found.**).

2440

2441

2442

2443

2444

2445

2446

2447

2448

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

2449

2450

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

2451

2452

2453

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

2454

2455

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

2456

2457

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

2458

2459

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

2460

## 2461 7.1 Local and Remotable Interfaces

2462 A remotable service is one which can be called by a client which is running in an operating system  
2463 process different from that of the service itself (this also applies to clients running on different  
2464 machines from the service). Whether a service of a component implementation is remotable is  
2465 defined by the interface of the service. WSDL defined interfaces are always remotable. See the  
2466 relevant specifications for details of interfaces defined using other languages.

2467 The style of remotable interfaces is typically *coarse grained* and intended for *loosely coupled*  
2468 interactions. **Remotable service Interfaces MUST NOT make use of *method or operation***  
2469 ***overloading***. [ASM80002] This restriction on operation overloading for remotable services aligns  
2470 with the WSDL 2.0 specification, which disallows operation overloading, and also with the WS-I  
2471 Basic Profile 1.1 (section 4.5.3 - R2304) which has a constraint which disallows operation  
2472 overloading when using WSDL 1.1.

2473 Independent of whether the remotable service is called remotely from outside the process where  
2474 the service runs or from another component running in the same process, the data exchange  
2475 semantics are *by-value*.

2476 Implementations of remotable services can modify input messages (parameters) during or after  
2477 an invocation and can modify return messages (results) after the invocation. **If a remotable**  
2478 **service is called locally or remotely, the SCA container MUST ensure sure that no modification of**  
2479 **input messages by the service or post-invocation modifications to return messages are seen by**  
2480 **the caller.** [ASM80003]

2481 Here is a snippet which shows an example of a remotable java interface:

```
2482 package services.hello;  
2483  
2484 @Remotable  
2485 public interface HelloService {  
2486  
2487     String hello(String message);  
2488 }
```

2489

2490 It is possible for the implementation of a remotable service to indicate that it can be called using  
2491 by-reference data exchange semantics when it is called from a component in the same process.  
2492 This can be used to improve performance for service invocations between components that run in  
2493 the same process. This can be done using the `@AllowsPassByReference` annotation (see the [Java](#)  
2494 [Client and Implementation Specification](#)).

2495 A service typed by a local interface can only be called by clients that are running in the same  
2496 process as the component that implements the local service. Local services cannot be published  
2497 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2498 Java interface definition without a *@Remotable* annotation.

2499 The style of local interfaces is typically *fine grained* and intended for *tightly coupled*  
2500 interactions. Local service interfaces can make use of *method or operation overloading*.

2501 The data exchange semantic for calls to services typed by local interfaces is *by-reference*.

2502

## 2503 7.2 Bidirectional Interfaces

2504 The relationship of a business service to another business service is often peer-to-peer, requiring  
2505 a two-way dependency at the service level. In other words, a business service represents both a  
2506 consumer of a service provided by a partner business service and a provider of a service to the  
2507 partner business service. This is especially the case when the interactions are based on  
2508 asynchronous messaging rather than on remote procedure calls. The notion of *bidirectional*  
2509 *interfaces* is used in SCA to directly model peer-to-peer bidirectional business service  
2510 relationships.



2511 An interface element for a particular interface type system needs to allow the specification of a  
2512 callback interface. If a callback interface is specified, SCA refers to the interface as a whole as a  
2513 bidirectional interface.

2514 The following snippet shows the interface element defined using Java interfaces with a  
2515 @callbackInterface attribute.

```
2516 <interface.java interface="services.invoicing.ComputePrice"  
2517         callbackInterface="services.invoicing.InvoiceCallback"/>
```

2518 If a service is defined using a bidirectional interface element then its implementation implements  
2519 the interface, and its implementation uses the callback interface to converse with the client that  
2520 called the service interface.

2521 If a reference is defined using a bidirectional interface element, the client component  
2522 implementation using the reference calls the referenced service using the interface. The client  
2523 MUST provide an implementation of the callback interface. [ASM80004]

2524 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2525 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2526 mix local and remote services. [ASM80005]

2527 Note that an interface document such as a WSDL file or a Java interface can contain annotations  
2528 that declare a callback interface for a particular interface (see [the section on WSDL Interface type](#)  
2529 and the Java Common Annotations and APIs specification [SCA-Common-Java]). Whenever an  
2530 interface document declaring a callback interface is used in the declaration of an <interface/>  
2531 element in SCA, it MUST be treated as being bidirectional with the declared callback interface.  
2532 [ASM80010] In such cases, there is no requirement for the <interface/> element to declare the  
2533 callback interface explicitly.

2534 If an <interface/> element references an interface document which declares a callback interface  
2535 and also itself contains a declaration of a callback interface, the two callback interfaces MUST be  
2536 compatible. [ASM80011]

2537 Where a component uses an implementation and the component configuration explicitly declares  
2538 an interface for a service or a reference, if the matching service or reference declaration in the  
2539 component type declares an interface which has a callback interface, then the component interface  
2540 declaration MUST also declare a compatible interface with a compatible callback interface.  
2541 [ASM80012] If the service or reference declaration in the component type declares an interface  
2542 without a callback interface, then the component configuration for the corresponding service or  
2543 reference MUST NOT declare an interface with a callback interface. [ASM80013]

2544 Where a composite declares an interface for a composite service or a composite reference, if the  
2545 promoted service or promoted reference has an interface which has a callback interface, then the  
2546 interface declaration for the composite service or the composite reference MUST also declare a  
2547 compatible interface with a compatible callback interface. [ASM80014] If the promoted service or  
2548 promoted reference has an interface without a callback interface, then the interface declaration for  
2549 the composite service or composite reference MUST NOT declare a callback interface.  
2550 [ASM80015]

2551 See Section 6.4 Wires for a definition of "compatible interfaces".

2552 In a bidirectional interface, the service interface can have more than one operation defined, and  
2553 the callback interface can also have more than one operation defined. SCA runtimes MUST allow  
2554 an invocation of any operation on the service interface to be followed by zero, one or many  
2555 invocations of any of the operations on the callback interface. [ASM80009] These callback  
2556 operations can be invoked either before or after the operation on the service interface has  
2557 returned a response message, if there is one.

2558 For a given invocation of a service operation, which operations are invoked on the callback  
2559 interface, when these are invoked, the number of operations invoked, and their sequence are not  
2560 described by SCA. It is possible that this metadata about the bidirectional interface can be  
2561 supplied through mechanisms outside SCA. For example, it might be provided as a written  
2562 description attached to the callback interface.

## 2563 7.3 Long-running Request-Response Operations

### 2564 7.3.1 Background

2565 A service offering one or more operations which map to a WSDL request-response pattern might  
2566 be implemented in a long-running, potentially interruptible, way. Consider a BPEL process with  
2567 receive and reply activities referencing the WSDL request-response operation. Between the two  
2568 activities, the business process logic could be a long-running sequence of steps, including activities  
2569 causing the process to be interrupted. Typical examples are steps where the process waits for  
2570 another message to arrive or a specified time interval to expire, or the process performs  
2571 asynchronous interactions such as service invocations bound to asynchronous protocols or user  
2572 interactions. This is a common situation in business processes, and it causes the implementation  
2573 of the WSDL request-response operation to run for a very long time, e.g., several months (!). In  
2574 this case, it is not meaningful for any caller to remain in a synchronous wait for the response while  
2575 blocking system resources or holding database locks.

2576 Note that it is possible to model long-running interactions as a pair of two independent operations  
2577 as described in the section on bidirectional interfaces. However, it is a common practice (and in  
2578 fact much more convenient) to model a request-response operation and let the infrastructure deal  
2579 with the asynchronous message delivery and correlation aspects instead of putting this burden on  
2580 the application developer.

### 2581 7.3.2 Definition of "long-running"

2582 A request-response operation is considered long-running if the implementation does not guarantee  
2583 the delivery of the response within any specified time interval. Clients invoking such request-  
2584 response operations are strongly discouraged from making assumptions about when the response  
2585 can be expected.

### 2586 7.3.3 The asyncInvocation Intent

2587 This specification permits a long-running request-response operation or a complete interface  
2588 containing such operations to be marked using a policy intent with the name *asyncInvocation*. It  
2589 is also possible for a service to set the *asyncInvocation*. intent when using an interface which is  
2590 not marked with the *asyncInvocation*. intent. This can be useful when reusing an existing interface  
2591 definition that does not contain SCA information.

### 2592 7.3.4 Requirements on Bindings

2593 In order to support a service operation which is marked with the *asyncInvocation* intent, it is  
2594 necessary for the binding (and its associated policies) to support separate handling of the request  
2595 message and the response message. Bindings which only support a synchronous style of message  
2596 handling, such as a conventional HTTP binding, cannot be used to support long-running  
2597 operations.

2598 The requirements on a binding to support the *asyncInvocation* intent are the same as those to  
2599 support services with bidirectional interfaces - namely that the binding needs to be able to treat  
2600 the transmission of the request message separately from the transmission of the response  
2601 message, with an arbitrarily large time interval between the two transmissions.

2602 An example of a binding/policy combination that supports long-running request-response  
2603 operations is a Web service binding used in conjunction with the WS-Addressing  
2604 "wsam:NonAnonymousResponses" assertion.

### 2605 7.3.5 Implementation Type Support

2606 SCA implementation types can provide special asynchronous client-side and asynchronous server-  
2607 side mappings to assist in the development of services and clients for long-running request-  
2608 response operations.

## 2609 7.4 SCA-Specific Aspects for WSDL Interfaces

2610 There are a number of aspects that SCA applies to interfaces in general, such as marking them as  
2611 having a callback interface. These aspects apply to the interfaces themselves, rather than their  
2612 use in a specific place within SCA. There is thus a need to provide appropriate ways of marking  
2613 the interface definitions themselves, which go beyond the basic facilities provided by the interface  
2614 definition language.

2615 For WSDL interfaces, there is an extension mechanism that permits additional information to be  
2616 included within the WSDL document. SCA takes advantage of this extension mechanism. In order  
2617 to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-](http://docs.oasis-open.org/ns/opencsa/sca/200903)  
2618 [open.org/ns/opencsa/sca/200903](http://docs.oasis-open.org/ns/opencsa/sca/200903)) needs to be declared within the WSDL document.

2619 First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach  
2620 policy intents - **@requires**. The definition of this attribute is as follows:

```
2621 <attribute name="requires" type="sca:listOfQNames"/>
```

2622

```
2623 <simpleType name="listOfQNames">  
2624 <list itemType="QName"/>  
2625 </simpleType>
```

2626 The @requires attribute can be applied to WSDL Port Type elements (WSDL 1.1). The attribute  
2627 contains one or more intent names, as defined by [the Policy Framework specification \[10\]](#). **Any**  
2628 **service or reference that uses an interface marked with intents MUST implicitly add those intents**  
2629 **to its own @requires list.** [ASM80008]

2630 SCA defines an attribute which is used to indicate that a given WSDL Port Type element (WSDL  
2631 1.1) has an associated callback interface. This is the @callback attribute, which applies to a WSDL  
2632 <portType/> element.

2633

2634 The @callback attribute is defined as a global attribute in the SCA namespace, as follows:

```
2635 <attribute name="callback" type="QName"/>
```

2636

2637 The value of the @callback attribute is the QName of a Port Type. The port type declared by the  
2638 @callback attribute is the callback interface to use for the portType which is annotated by the  
2639 @callback attribute.

2640

2641 Here is an example of a portType element with a @callback attribute:

2642

```
2643 <portType name="LoanService" sca:callback="foo:LoanServiceCallback">  
2644 <operation name="apply">  
2645 <input message="tns:ApplicationInput"/>  
2646 <output message="tns:ApplicationOutput"/>  
2647 </operation>  
2648 ...  
2649 </portType>
```

## 2650 7.5 WSDL Interface Type

2651 The WSDL interface type is used to declare interfaces for services and for references, where the interface  
2652 is defined in terms of a WSDL document. An interface is defined in terms of a WSDL 1.1 Port Type with  
2653 the arguments and return of the service operations described using XML schema.

2654

2655 A WSDL interface is declared by an **interface.wSDL** element. The following shows the pseudo-schema  
2656 for the interface.wSDL element:

```
2657 <!-- WSDL Interface schema snippet -->
2658 <interface.wSDL interface="xs:anyURI" callbackInterface="xs:anyURI"?
2659     remotable="xs:boolean"? >
```

2660 The interface.wSDL element has the following **attributes**:

- 2661 • **interface : uri (1..1)** - the URI of a WSDL Port Type  
2662 The interface.wSDL @interface attribute MUST reference a portType of a WSDL 1.1  
2663 document. [ASM80001]
- 2664 • **callbackInterface : uri (0..1)** - a callback interface, which is the URI of a WSDL Port Type  
2665 The interface.wSDL @callbackInterface attribute, if present, MUST reference a portType of a  
2666 WSDL 1.1 document. [ASM80016]
- 2667 • **remotable : boolean (0..1)** – indicates whether the interface is remotable or not. @remotable  
2668 has a default value of true. WSDL interfaces are always remotable and therefore an  
2669 <interface.wSDL/> element MUST NOT contain remotable="false". [ASM80017]

2670

2671 The form of the URI for WSDL port types follows the syntax described in the WSDL 1.1 Element  
2672 Identifiers specification [WSDL11\_Identifiers]

## 2673 7.5.1 Example of interface.wSDL

```
2674 <interface.wSDL interface="http://www.stockquote.org/StockQuoteService#
2675     wsdl.porttype(StockQuote)"
2676     callbackInterface="http://www.stockquote.org/StockQuoteService#
2677     wsdl.porttype(StockQuoteCallback)"/>
2678
```

2679 This declares an interface in terms of the WSDL port type "StockQuote" with a callback interface defined  
2680 by the "StockQuoteCallback" port type.

2681

2682

## 8 Binding

2683  
2684  
2685  
2686

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

2687  
2688  
2689  
2690

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

2691  
2692  
2693

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

2694  
2695  
2696  
2697  
2698  
2699  
2700  
2701  
2702  
2703  
2704  
2705  
2706  
2707  
2708  
2709  
2710  
2711  
2712  
2713  
2714  
2715  
2716  
2717  
2718  
2719  
2720  
2721  
2722  
2723  
2724  
2725  
2726  
2727  
2728  
2729  
2730  
2731  
2732  
2733  
2734

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Bindings schema snippet -->
<composite ... >
  ...
  <service ... >*
    <interface ... />?
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?>*
      <wireFormat/>?
      <operationSelector/>?
    </binding>
    <callback?
      <binding uri="xs:anyURI"? name="xs:NCName"?
        requires="list of xs:QName"?
        policySets="list of xs:QName"?>+
        <wireFormat/>?
        <operationSelector/>?
      </binding>
    </callback>
  </service>
  ...
  <reference ... >*
    <interface ... />?
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?>*
      <wireFormat/>?
      <operationSelector/>?
    </binding>
    <callback?
      <binding uri="xs:anyURI"? name="xs:NCName"?
        requires="list of xs:QName"?
        policySets="list of xs:QName"?>+
        <wireFormat/>?
        <operationSelector/>?
      </binding>
    </callback>
  </reference>
  ...
</composite>
```

2735

2736 The element name of the binding element is architected; it is in itself a qualified name. The first  
2737 qualifier is always named "binding", and the second qualifier names the respective binding-type  
2738 (e.g. binding.sca, binding.ws, binding.ejb, binding.eis).

2739

2740 A binding element has the following attributes:

- 2741
- **uri (0..1)** - has the following semantic.
    - The @uri attribute can be omitted.
    - For a binding of a **reference** the @uri attribute defines the target URI of the reference. This MUST be either the componentName/serviceName for a wire to an endpoint within the SCA Domain, or the accessible address of some service endpoint either inside or outside the SCA Domain (where the addressing scheme is defined by the type of the binding). [ASM90001]
    - The circumstances under which the @uri attribute can be used are defined in section "Specifying the Target Service(s) for a Reference."
    - For a binding of a **service** the @uri attribute defines the bindingURI. If present, the bindingURI can be used by the binding as described in the section "Form of the URI of a Deployed Binding".
  - **name (0..1)** – a name for the binding instance (an NCName). The @name attribute allows distinction between multiple binding elements on a single service or reference. The default value of the @name attribute is the service or reference name. When a service or reference has multiple bindings, only one binding can have the default @name value; all others MUST have a @name value specified that is unique within the service or reference. [ASM90002] The @name also permits the binding instance to be referenced from elsewhere – particularly useful for some types of binding, which can be declared in a definitions document as a template and referenced from other binding instances, simplifying the definition of more complex binding instances (see the JMS Binding specification [11] for examples of this referencing).
  - **requires (0..1)** - a list of policy intents. See the Policy Framework specification [10] for a description of this attribute.
  - **policySets (0..1)** – a list of policy sets. See the Policy Framework specification [10] for a description of this attribute.

2767 A binding element has the following child elements:

- 2768
- **wireFormat (0..1)** - a wireFormat to apply to the data flowing using the binding. See the wireFormat section for details.
  - **operationSelector(0..1)** - an operationSelector element that is used to match a particular message to a particular operation in the interface. See the operationSelector section for details

2773 When multiple bindings exist for a service, it means that the service is available through any of  
2774 the specified bindings. The technique that the SCA runtime uses to choose among available  
2775 bindings is left to the implementation and it might include additional (nonstandard) configuration.  
2776 Whatever technique is used needs to be documented by the runtime.

2777 Services and References can always have their bindings overridden at the SCA Domain level,  
2778 unless restricted by Intents applied to them.

2779 If a reference has any bindings, they MUST be resolved, which means that each binding MUST  
2780 include a value for the @uri attribute or MUST otherwise specify an endpoint. The reference MUST  
2781 NOT be wired using other SCA mechanisms. [ASM90003] To specify constraints on the kinds of  
2782 bindings that are acceptable for use with a reference, the user specifies either policy intents or  
2783 policy sets.

2784  
2785 Users can also specifically wire, not just to a component service, but to a specific binding offered

2786 by that target service. To do so, a wire target MAY be specified with a syntax of  
2787 "componentName/serviceName/bindingName". [ASM90004]

2788

2789 The following sections describe the SCA and Web service binding type in detail.

## 2790 8.1 Messages containing Data not defined in the Service Interface

2791 It is possible for a message to include information that is not defined in the interface used to  
2792 define the service, for instance information can be contained in SOAP headers or as MIME  
2793 attachments.

2794 Implementation types can make this information available to component implementations in their  
2795 execution context. The specifications for these implementation types describe how this  
2796 information is accessed and in what form it is presented.

## 2797 8.2 WireFormat

2798 A wireFormat is the form that a data structure takes when it is transmitted using some  
2799 communication binding. Another way to describe this is "the form that the data takes on the wire".  
2800 A wireFormat can be specific to a given communication method, or it can be general, applying to  
2801 many different communication methods. An example of a general wireFormat is XML text format.

2802 Where a particular SCA binding can accommodate transmitting data in more than one format, the  
2803 configuration of the binding can include a definition of the wireFormat to use. This is done using an  
2804 <sca:wireFormat/> subelement of the <binding/> element.

2805 Where a binding supports more than one wireFormat, the binding defines one of the wireFormats  
2806 to be the default wireFormat which applies if no <wireFormat/> subelement is present.

2807 The base sca:wireFormat element is abstract and it has no attributes and no child elements. For a  
2808 particular wireFormat, an extension subtype is defined, using substitution groups, for example:

- 2809 • <sca:wireFormat.xml/>  
2810 A wireFormat that transmits the data as an XML text datastructure
- 2811 • <sca:wireFormat.jms/>  
2812 The "default JMS wireFormat" as described in the JMS Binding specification

2813

2814 Specific wireFormats can have elements that include either attributes or subelements or both.

2815 For details about specific wireFormats, see the related SCA Binding specifications.

## 2816 8.3 OperationSelector

2817 An operationSelector is necessary for some types of transport binding where messages are  
2818 transmitted across the transport without any explicit relationship between the message and the  
2819 interface operation to which it relates. SOAP is an example of a protocol where the messages do  
2820 contain explicit information that relates each message to the operation it targets. However, other  
2821 transport bindings have messages where this relationship is not expressed in the message or in  
2822 any related headers (pure JMS messages, for example). In cases where the messages arrive at a  
2823 service without any explicit information that maps them to specific operations, it is necessary for  
2824 the metadata attached to the service binding to contain the mapping information. The information  
2825 is held in an operationSelector element which is a child element of the binding element.

2826 The base sca:operationSelector element is abstract and it has no attributes and no child elements.  
2827 For a particular operationSelector, an extension subtype is defined, using substitution groups, for  
2828 example:

- 2829 • <sca:operationSelector.XPath/>  
2830 An operation selector that uses XPath to filter out specific messages and target them to  
2831 particular named operations.

2832  
2833 Specific operationSelectors can have elements that include either attributes or subelements or  
2834 both.  
2835 For details about specific operationSelectors, see the related SCA Binding specifications.

## 2836 8.4 Form of the URI of a Deployed Binding

2837 SCA Bindings specifications can choose to use the **structural URI** defined in the section  
2838 "[Structural URI of Components](#)" above to derive a binding specific URI according to some Binding-  
2839 related scheme. The relevant binding specification describes this.

2840 Alternatively, <binding/> elements have a @uri attribute, which is termed a bindingURI.

2841 If the bindingURI is specified on a given <binding/> element, the binding can use it to derive an  
2842 endpoint URI relevant to the binding. The derivation is binding specific and is described by the  
2843 relevant binding specification.

2844 For binding.sca, which is described in the SCA Assembly specification, this is as follows:

- 2845 • If the binding @uri attribute is specified on a reference, it identifies the target service in  
2846 the SCA Domain by specifying the service's structural URI.
- 2847 • If the binding @uri attribute is specified on a service, it is ignored.

### 2848 8.4.1 Non-hierarchical URIs

2849 Bindings that use non-hierarchical URI schemes (such as jms: or mailto:) can make use of the  
2850 @uri attribute, which is the complete representation of the URI for that service binding. Where  
2851 the binding does not use the @uri attribute, the binding needs to offer a different mechanism for  
2852 specifying the service address.

### 2853 8.4.2 Determining the URI scheme of a deployed binding

2854 One of the things that needs to be determined when building the effective URI of a deployed  
2855 binding (i.e. endpoint) is the URI scheme. The process of determining the endpoint URI scheme is  
2856 binding type specific.

2857 If the binding type supports a single protocol then there is only one URI scheme associated with it.  
2858 In this case, that URI scheme is used.

2859 If the binding type supports multiple protocols, the binding type implementation determines the  
2860 URI scheme by introspecting the binding configuration, which can include the policy sets  
2861 associated with the binding.

2862 A good example of a binding type that supports multiple protocols is binding.ws, which can be  
2863 configured by referencing either an "abstract" WSDL element (i.e. portType or interface) or a  
2864 "concrete" WSDL element (i.e. binding, port or endpoint). When the binding references a PortType  
2865 or Interface, the protocol and therefore the URI scheme is derived from the intents/policy sets  
2866 attached to the binding. When the binding references a "concrete" WSDL element, there are two  
2867 cases:

- 2868 1) The referenced WSDL binding element uniquely identifies a URI scheme. This is the most  
2869 common case. In this case, the URI scheme is given by the protocol/transport specified in the  
2870 WSDL binding element.
- 2871 2) The referenced WSDL binding element doesn't uniquely identify a URI scheme. For example,  
2872 when HTTP is specified in the @transport attribute of the SOAP binding element, both "http"  
2873 and "https" could be used as valid URI schemes. In this case, the URI scheme is determined  
2874 by looking at the policy sets attached to the binding.

2875 It is worth noting that an intent supported by a binding type can completely change the behavior  
2876 of the binding. For example, when the intent "confidentiality/transport" is attached to an HTTP  
2877 binding, SSL is turned on. This basically changes the URI scheme of the binding from "http" to  
2878 "https".



2879

## 2880 8.5 SCA Binding

2881 The SCA binding element is defined by the following schema.

2882 `<binding.sca />`

2883 The SCA binding can be used for service interactions between references and services contained  
2884 within the SCA Domain. The way in which this binding type is implemented is not defined by the  
2885 SCA specification and it can be implemented in different ways by different SCA runtimes. The only  
2886 requirement is that any specified qualities of service are implemented for the SCA binding type.  
2887 The SCA binding type is **not** intended to be an interoperable binding type. For interoperability, an  
2888 interoperable binding type such as the Web service binding is used.

2889 A service definition with no binding element specified uses the SCA binding.  
2890 `<binding.sca/>` would only have to be specified in override cases, or when you specify a  
2891 set of bindings on a service definition and the SCA binding needs to be one of them.

2892 If a reference does not have a binding, then the binding used can be any of the bindings  
2893 specified by the service provider, as long as the intents attached to the reference and  
2894 the service are all honoured.

2895 If the interface of the service or reference is local, then the local variant of the SCA  
2896 binding will be used. If the interface of the service or reference is remotable, then either  
2897 the local or remote variant of the SCA binding will be used depending on whether source  
2898 and target are co-located or not.

2899 If a reference specifies a URI via its `@uri` attribute, then this provides the default wire to a service  
2900 provided by another Domain level component. The value of the URI has to be as follows:

- 2901
- `<domain-component-name>/<service-name>`

### 2902 8.5.1 Example SCA Binding

2903 The following snippet shows the `MyValueComposite.composite` file for the `MyValueComposite`  
2904 containing the service element for the `MyValueService` and a reference element for the  
2905 `StockQuoteService`. Both the service and the reference use an SCA binding. The target for the  
2906 reference is left undefined in this binding and would have to be supplied by the composite in which  
2907 this composite is used.

```
2908 <?xml version="1.0" encoding="ASCII"?>
2909 <!-- Binding SCA example -->
2910 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2911           targetNamespace="http://foo.com"
2912           name="MyValueComposite" >
2913
2914     <service name="MyValueService" promote="MyValueComponent">
2915       <interface.java interface="services.myvalue.MyValueService"/>
2916       <binding.sca/>
2917       ...
2918     </service>
2919
2920     ...
2921
2922     <reference name="StockQuoteService"
2923               promote="MyValueComponent/StockQuoteReference">
2924       <interface.java interface="services.stockquote.StockQuoteService"/>
2925       <binding.sca/>
2926     </reference>
2927
2928 </composite>
```

2929 **8.6 Web Service Binding**

2930 SCA defines a Web services binding. This is described in [a separate specification document \[9\]](#).

2931 **8.7 JMS Binding**

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

2933

## 9 SCA Definitions

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

2937 All of these artifacts within an SCA Domain are defined in SCA contributions in files called META-  
2938 INF/definitions.xml (relative to the contribution base URI). **An SCA runtime MUST make available**  
2939 **to the Domain all the artifacts contained within the definitions.xml files in the Domain.**  
2940 **[ASM10002] An SCA runtime MUST reject a definitions.xml file that does not conform to the sca-**  
2941 **definitions.xsd schema. [ASM10003]**

2942 Although the definitions are specified within a single SCA contribution, the definitions are visible  
2943 throughout the Domain. Because of this, **all of the QNames for the definitions contained in**  
2944 **definitions.xml files MUST be unique within the Domain.** **[ASM10001]** The definitions.xml file  
2945 contains a definitions element that conforms to the following pseudo-schema snippet:

```
2946 <?xml version="1.0" encoding="ASCII"?>  
2947 <!-- Composite schema snippet -->  
2948 <definitions xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200903"  
2949             targetNamespace="xs:anyURI">  
2950  
2951     <sca:intent/*>  
2952  
2953     <sca:policySet/*>  
2954  
2955     <sca:binding/*>  
2956  
2957     <sca:bindingType/*>  
2958  
2959     <sca:implementationType/*>  
2960  
2961 </definitions>
```

2962 The definitions element has the following attribute:

- 2963 • **targetNamespace (1..1)** – the namespace into which the child elements of this  
2964 definitions element are placed (used for artifact resolution)

2965 The definitions element contains child elements – intent, policySet, binding, bindingType and  
2966 implementationType. These elements are described elsewhere in this specification or in [the SCA](#)  
2967 [Policy Framework specification \[10\]](#). The use of the elements declared within a definitions  
2968 element is described in the [SCA Policy Framework specification \[10\]](#) and in [the JMS Binding](#)  
2969 [specification \[11\]](#).

2970

## 10 Extension Model

2971 The assembly model can be extended with support for new interface types, implementation types  
2972 and binding types. The extension model is based on XML schema substitution groups. There are  
2973 three XML Schema substitution group heads defined in the SCA namespace: **interface**,  
2974 **implementation** and **binding**, for interface types, implementation types and binding types,  
2975 respectively.

2976 The SCA Client and Implementation specifications and the SCA Bindings specifications (see [1],  
2977 [9], [11]) use these XML Schema substitution groups to define some basic types of interfaces,  
2978 implementations and bindings, but additional types can be defined as needed, where support for  
2979 these extra ones is available from the runtime. The interface type elements, implementation type  
2980 elements, and binding type elements defined by the SCA specifications are all part of the SCA  
2981 namespace ("http://docs.oasis-open.org/ns/opencsa/sca/200903"), as indicated in their  
2982 respective schemas. New interface types, implementation types and binding types that are defined  
2983 using this extensibility model, which are not part of these SCA specifications are defined in  
2984 namespaces other than the SCA namespace.

2985 The "." notation is used in naming elements defined by the SCA specifications ( e.g.  
2986 <implementation.java ... />, <interface.wsdl ... />, <binding.ws ... />), not as a parallel  
2987 extensibility approach but as a naming convention that improves usability of the SCA assembly  
2988 language.

2989

2990 **Note:** How to contribute SCA model extensions and their runtime function to an SCA runtime will  
2991 be defined by a future version of the specification.

### 10.1 Defining an Interface Type

2993 The following snippet shows the base definition for the **interface** element and **Interface** type  
2994 contained in **sca-core.xsd**; see appendix for complete schema.

```
2995 <?xml version="1.0" encoding="UTF-8"?>
2996 <!-- (c) Copyright SCA Collaboration 2006 -->
2997 <schema xmlns="http://www.w3.org/2001/XMLSchema"
2998         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
2999         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3000         elementFormDefault="qualified">
3001
3002     ...
3003
3004     <element name="interface" type="sca:Interface" abstract="true"/>
3005     <complexType name="Interface"/>
3006     <complexType name="Interface" abstract="true">
3007         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3008         <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3009     </complexType>
3010
3011     ...
3012
3013 </schema>
```

3014 In the following snippet is an example of how the base definition is extended to support Java  
3015 interfaces. The snippet shows the definition of the **interface.java** element and the  
3016 **JavaInterface** type contained in **sca-interface-java.xsd**.

```
3017 <?xml version="1.0" encoding="UTF-8"?>
3018 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3019         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
```

```

3020         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3021
3022     <element name="interface.java" type="sca:JavaInterface"
3023         substitutionGroup="sca:interface"/>
3024     <complexType name="JavaInterface">
3025         <complexContent>
3026             <extension base="sca:Interface">
3027                 <attribute name="interface" type="NCName"
3028                     use="required"/>
3029             </extension>
3030         </complexContent>
3031     </complexType>
3032 </schema>

```

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

```

3037 <?xml version="1.0" encoding="UTF-8"?>
3038 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3039     targetNamespace="http://www.example.org/myextension"
3040     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3041     xmlns:tns="http://www.example.org/myextension">
3042
3043     <element name="my-interface-extension"
3044         type="tns:my-interface-extension-type"
3045         substitutionGroup="sca:interface"/>
3046     <complexType name="my-interface-extension-type">
3047         <complexContent>
3048             <extension base="sca:Interface">
3049                 ...
3050             </extension>
3051         </complexContent>
3052     </complexType>
3053 </schema>

```

## 3054 10.2 Defining an Implementation Type

3055 The following snippet shows the base definition for the *implementation* element and  
3056 *Implementation* type contained in *sca-core.xsd*; see appendix for complete schema.

```

3057 <?xml version="1.0" encoding="UTF-8"?>
3058 <!-- (c) Copyright SCA Collaboration 2006 -->
3059 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3060     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3061     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3062     elementFormDefault="qualified">
3063
3064     ...
3065
3066     <element name="implementation" type="sca:Implementation"
3067         abstract="true"/>
3068     <complexType name="Implementation"/>
3069
3070     ...
3071
3072 </schema>
3073
3074

```

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

```
3078 <?xml version="1.0" encoding="UTF-8"?>
3079 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3080         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3081         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3082
3083     <element name="implementation.java" type="sca:JavaImplementation"
3084             substitutionGroup="sca:implementation"/>
3085     <complexType name="JavaImplementation">
3086         <complexContent>
3087             <extension base="sca:Implementation">
3088                 <attribute name="class" type="NCName"
3089                         use="required"/>
3090             </extension>
3091         </complexContent>
3092     </complexType>
3093 </schema>
```

3094 In the following snippet is an example of how the base definition can be extended by other  
3095 specifications to support a new implementation type not defined in the SCA specifications. The  
3096 snippet shows the definition of the **my-impl-extension** element and the **my-impl-extension-**  
3097 **type** type.

```
3098 <?xml version="1.0" encoding="UTF-8"?>
3099 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3100         targetNamespace="http://www.example.org/myextension"
3101         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3102         xmlns:tns="http://www.example.org/myextension">
3103
3104     <element name="my-impl-extension" type="tns:my-impl-extension-type"
3105             substitutionGroup="sca:implementation"/>
3106     <complexType name="my-impl-extension-type">
3107         <complexContent>
3108             <extension base="sca:Implementation">
3109                 ...
3110             </extension>
3111         </complexContent>
3112     </complexType>
3113 </schema>
```

3115 In addition to the definition for the new implementation instance element, there needs to be an  
3116 associated implementationType element which provides metadata about the new implementation  
3117 type. The pseudo schema for the implementationType element is shown in the following snippet:

```
3118 <implementationType type="xs:QName"
3119                   alwaysProvides="list of intent xs:QName"
3120                   mayProvide="list of intent xs:QName"/>
```

3122 The implementation type has the following attributes:

- 3123 • **type (1..1)** – the type of the implementation to which this implementationType element  
3124 applies. This is intended to be the QName of the implementation element for the  
3125 implementation type, such as "sca:implementation.java"
- 3126 • **alwaysProvides (0..1)** – a set of intents which the implementation type always  
3127 provides. See [the Policy Framework specification \[10\]](#) for details.

- **mayProvide (0..1)** – a set of intents which the implementation type provides only when the intent is attached to the implementation element. See [the Policy Framework specification \[10\]](#) for details.

### 10.3 Defining a Binding Type

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

```

3134 <?xml version="1.0" encoding="UTF-8"?>
3135 <!-- binding type schema snippet -->
3136 <!-- (c) Copyright SCA Collaboration 2006, 2009 -->
3137 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3138         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3139         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3140         elementFormDefault="qualified">
3141
3142     ...
3143
3144     <element name="binding" type="sca:Binding" abstract="true"/>
3145     <complexType name="Binding">
3146         <attribute name="uri" type="anyURI" use="optional"/>
3147         <attribute name="name" type="NCName" use="optional"/>
3148         <attribute name="requires" type="sca:listOfQNames"
3149             use="optional"/>
3150         <attribute name="policySets" type="sca:listOfQNames"
3151             use="optional"/>
3152     </complexType>
3153
3154     ...
3155 </schema>
3156
3157

```

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

```

3161 <?xml version="1.0" encoding="UTF-8"?>
3162 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3163         targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3164         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3165
3166     <element name="binding.ws" type="sca:WebServiceBinding"
3167         substitutionGroup="sca:binding"/>
3168     <complexType name="WebServiceBinding">
3169         <complexContent>
3170             <extension base="sca:Binding">
3171                 <attribute name="port" type="anyURI" use="required"/>
3172             </extension>
3173         </complexContent>
3174     </complexType>
3175 </schema>

```

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

```

3179 <?xml version="1.0" encoding="UTF-8"?>
3180 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3181         targetNamespace="http://www.example.org/myextension"

```

```

3182     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3183     xmlns:tns="http://www.example.org/myextension">
3184
3185     <element name="my-binding-extension"
3186         type="tns:my-binding-extension-type"
3187         substitutionGroup="sca:binding"/>
3188     <complexType name="my-binding-extension-type">
3189         <complexContent>
3190             <extension base="sca:Binding">
3191                 ...
3192             </extension>
3193         </complexContent>
3194     </complexType>
3195 </schema>

```

3196 In addition to the definition for the new binding instance element, there needs to be an associated  
3197 bindingType element which provides metadata about the new binding type. The pseudo schema  
3198 for the bindingType element is shown in the following snippet:

```

3199 <bindingType type="xs:QName"
3200     alwaysProvides="list of intent QNames"?
3201     mayProvide = "list of intent QNames"?/>
3202

```

3203 The binding type has the following attributes:

- 3204 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3205 intended to be the QName of the binding element for the binding type, such as  
3206 "sca:binding.ws"
- 3207 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3208 the [Policy Framework specification \[10\]](#) for details.
- 3209 • **mayProvide (0..1)** – a set of intents which the binding type provides only when the  
3210 intent is attached to the binding element. See the [Policy Framework specification \[10\]](#) for  
3211 details.

## 3212 10.4 Defining an Import Type

3213 The following snippet shows the base definition for the **import** element and **Import** type contained in **sca-**  
3214 **core.xsd**; see appendix for complete schema.

```

3215 <?xml version="1.0" encoding="UTF-8"?>
3216 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved. OASIS trademark,
3217 IPR and other policies apply. -->
3218 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3219     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3220     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3221     elementFormDefault="qualified">
3222
3223     ...
3224
3225     <!-- Import -->
3226     <element name="importBase" type="sca:Import" abstract="true" />
3227     <complexType name="Import" abstract="true">
3228         <complexContent>
3229             <extension base="sca:CommonExtensionBase">
3230                 <sequence>
3231                     <any namespace="##other" processContents="lax" minOccurs="0"
3232                         maxOccurs="unbounded"/>
3233                 </sequence>
3234             </extension>

```



```

3235     </complexContent>
3236 </complexType>
3237
3238 <element name="import" type="sca:ImportType"
3239     substitutionGroup="sca:importBase"/>
3240 <complexType name="ImportType">
3241     <complexContent>
3242         <extension base="sca:Import">
3243             <attribute name="namespace" type="string" use="required"/>
3244             <attribute name="location" type="anyURI" use="required"/>
3245         </extension>
3246     </complexContent>
3247 </complexType>
3248
3249 ...
3250
3251 </schema>
3252

```

3253 In the following snippet we show how the base import definition is extended to support Java imports. In  
3254 the import element, the namespace is expected to be an XML namespace, an `import.java` element uses a  
3255 Java package name instead. The snippet shows the definition of the *import.java* element and the  
3256 *JavaImportType* type contained in *sca-import-java.xsd*.

```

3257 <?xml version="1.0" encoding="UTF-8"?>
3258 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3259     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3260     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3261
3262     <element name="import.java" type="sca:JavaImportType"
3263     substitutionGroup="sca:importBase"/>
3264     <complexType name="JavaImportType">
3265         <complexContent>
3266             <extension base="sca:Import">
3267                 <attribute name="package" type="xs:String" use="required"/>
3268                 <attribute name="location" type="xs:AnyURI" use="optional"/>
3269             </extension>
3270         </complexContent>
3271     </complexType>
3272 </schema>
3273

```

3274 In the following snippet we show an example of how the base definition can be extended by other  
3275 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3276 definition of the *my-import-extension* element and the *my-import-extension-type* type.

```

3277 <?xml version="1.0" encoding="UTF-8"?>
3278 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3279     targetNamespace="http://www.example.org/myextension"
3280     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3281     xmlns:tns="http://www.example.org/myextension">
3282
3283     <element name="my-import-extension"
3284     type="tns:my-import-extension-type"
3285     substitutionGroup="sca:importBase"/>
3286     <complexType name="my-import-extension-type">
3287         <complexContent>
3288             <extension base="sca:Import">
3289                 ...
3290             </extension>

```

```
3291     </complexContent>
3292   </complexType>
3293 </schema>
```

3294

3295 For a complete example using this extension point, see the definition of *import.java* in the SCA Java  
3296 Common Annotations and APIs Specification [SCA-Java].

## 3297 10.5 Defining an Export Type

3298 The following snippet shows the base definition for the *export* element and *ExportType* type contained in  
3299 *sca-core.xsd*; see appendix for complete schema.

```
3300 <?xml version="1.0" encoding="UTF-8"?>
3301 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved. OASIS trademark,
3302 IPR and other policies apply. -->
3303 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3304   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3305   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3306   elementFormDefault="qualified">
3307   ...
3308   ...
3309   <!-- Export -->
3310   <element name="exportBase" type="sca:Export" abstract="true" />
3311   <complexType name="Export" abstract="true">
3312     <complexContent>
3313       <extension base="sca:CommonExtensionBase">
3314         <sequence>
3315           <any namespace="##other" processContents="lax" minOccurs="0"
3316             maxOccurs="unbounded" />
3317         </sequence>
3318       </extension>
3319     </complexContent>
3320   </complexType>
3321   ...
3322   <element name="export" type="sca:ExportType"
3323     substitutionGroup="sca:exportBase"/>
3324   <complexType name="ExportType">
3325     <complexContent>
3326       <extension base="sca:Export">
3327         <attribute name="namespace" type="string" use="required"/>
3328       </extension>
3329     </complexContent>
3330   </complexType>
3331   ...
3332 </schema>
```

3333

3334 The following snippet shows how the base definition is extended to support Java exports. In a base  
3335 *export* element, the *@namespace* attribute specifies XML namespace being exported. An *export.java*  
3336 element uses a *@package* attribute to specify the Java package to be exported. The snippet shows the  
3337 definition of the *export.java* element and the *JavaExport* type contained in *sca-export-java.xsd*.

```
3338 <?xml version="1.0" encoding="UTF-8"?>
3339 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3340   targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3341   xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3342   ...
3343   <element name="export.java" type="sca:JavaExportType"
3344     substitutionGroup="sca:exportBase"/>
```

```

3345     <complexType name="JavaExportType">
3346         <complexContent>
3347             <extension base="sca:Export">
3348                 <attribute name="package" type="xs:String" use="required"/>
3349             </extension>
3350         </complexContent>
3351     </complexType>
3352 </schema>

```

3353

3354 In the following snippet we show an example of how the base definition can be extended by other  
3355 specifications to support a new interface not defined in the SCA specifications. The snippet shows the  
3356 definition of the **my-export-extension** element and the **my-export-extension-type** type.

```

3357 <?xml version="1.0" encoding="UTF-8"?>
3358 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3359         targetNamespace="http://www.example.org/myextension"
3360         xmlns:sca="http:// docs.oasis-open.org/ns/opencsa/sca/200903"
3361         xmlns:tns="http://www.example.org/myextension">
3362
3363     <element name="my-export-extension"
3364             type="tns:my-export-extension-type"
3365             substitutionGroup="sca:exportBase"/>
3366     <complexType name="my-export-extension-type">
3367         <complexContent>
3368             <extension base="sca:Export">
3369                 ...
3370             </extension>
3371         </complexContent>
3372     </complexType>
3373 </schema>

```

3374

3375 For a complete example using this extension point, see the definition of **export.java** in the SCA Java  
3376 Common Annotations and APIs Specification [SCA-Java].

---

## 3377 11 Packaging and Deployment

3378 This section describes the SCA Domain and the packaging and deployment of artifacts contributed to the  
3379 Domain.

### 3380 11.1 Domains

3381 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3382 of interconnected runtime nodes.

3383 A single SCA Domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3384 wires can only be used to connect components within a single SCA Domain. Connections to  
3385 services outside the Domain use binding specific mechanisms for addressing services (such as  
3386 WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used in  
3387 the context of a single Domain. In general, external clients of a service that is developed and  
3388 deployed using SCA are not able to tell that SCA is used to implement the service – it is an  
3389 implementation detail.

3390 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3391 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3392 functionality controlled by a single organization. For example, an SCA Domain might be the whole  
3393 of a business, or it might be a department within a business.

3394 As an example, for the accounts department in a business, the SCA Domain might cover all  
3395 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3396 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3397 An SCA Domain has the following:

- 3398 • A virtual domain-level composite whose components are deployed and running
- 3399 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3400 necessary to execute components
- 3401 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3402 level composite.

3403 The information associated with an SCA Domain can be stored in many ways, including but not  
3404 limited to a specific filesystem structure or a repository.

### 3405 11.2 Contributions

3406 An SCA Domain might need a large number of different artifacts in order to work. These artifacts  
3407 include artifacts defined by SCA and other artifacts such as object code files and interface  
3408 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3409 different SCA definition documents are: `composite`, `componentType`, `constrainingType` and  
3410 definitions. XML artifacts that are not defined by SCA but which are needed by an SCA Domain  
3411 include XML Schema documents, WSDL documents, and BPEL documents. SCA constructs, like  
3412 other XML-defined constructs, use XML qualified names for their identity (i.e. namespace + local  
3413 name).

3414 Non-XML artifacts are also needed within an SCA Domain. The most obvious examples of such  
3415 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3416 implementations. Since SCA is extensible, other XML and non-XML artifacts might also be needed.

3417 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3418 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3419 packaging formats, but it is necessary for an SCA runtime to support the ZIP contribution format.  
3420 When using the ZIP format for deploying a contribution, this specification does not specify whether  
3421 that format is retained after deployment. For example, a Java EE based SCA runtime could convert  
3422 the ZIP package to an EAR package. SCA expects certain characteristics of any packaging:

- 3423
- 3424
- For any contribution packaging it MUST be possible to present the artifacts of the packaging to SCA as a hierarchy of resources based off of a single root [ASM12001]
  - Within any contribution packaging A directory resource SHOULD exist at the root of the hierarchy named META-INF [ASM12002]
  - Within any contribution packaging a document SHOULD exist directly under the META-INF directory named sca-contribution.xml which lists the SCA Composites within the contribution that are runnable. [ASM12003]

3427

3428

3429

3430

3431

3432

3433

3434

3435

3436

3437

3438

3439

3440

The same document can also list namespaces of constructs that are defined within the contribution and which are available for use by other contributions, through export elements.

**Error! Reference source not found.**

These additional elements might not be physically present in the packaging, but might be generated based on the definitions and references that are present, or they might not exist at all if there are no unresolved references.

See the section "[SCA Contribution Metadata Document](#)" for details of the format of this file.

3441

3442

3443

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

- 3444
- 3445
- 3446
- 3447
- A filesystem directory
  - An OSGi bundle
  - A compressed directory (zip, gzip, etc)
  - A JAR file (or its variants – WAR, EAR, etc)

3448

3449

3450

3451

3452

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

3453

3454

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

## 3455 11.2.1 SCA Artifact Resolution

3456

3457

3458

3459

3460

3461

3462

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

3463

3464

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

- 3465
- 3466
- 3467
- @wsdlLocation and @schemaLocation attributes in references to WSDL and XSD schema artifacts respectively
  - OSGi bundle mechanisms for resolving Java class and related resource dependencies

3468

3469

3470

3471

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

3472 SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanism is can  
3473 be used where no other mechanisms are available, for example in cases where the mechanisms  
3474 used by the various contributions in the same SCA Domain are different. An example of this is  
3475 where an OSGi Bundle is used for one contribution but where a second contribution used by the  
3476 first one is not implemented using OSGi - e.g. the second contribution relates to a mainframe  
3477 COBOL service whose interfaces are declared using a WSDL which is accessed by the first  
3478 contribution.

3479 The SCA artifact resolution is likely to be most useful for SCA Domains containing heterogeneous  
3480 mixtures of contribution, where artifact-related or packaging-related mechanisms are unlikely to  
3481 work across different kinds of contribution.

3482 SCA artifact resolution works on the principle that a contribution which needs to use artifacts  
3483 defined elsewhere expresses these dependencies using *import* statements in metadata belonging  
3484 to the contribution. A contribution controls which artifacts it makes available to other  
3485 contributions through *export* statements in metadata attached to the contribution. SCA artifact  
3486 resolution is a general mechanism that can be extended for the handling of specific types of  
3487 artifact. The general mechanism that is described in the following paragraphs is mainly intended  
3488 for the handling of XML artifacts. Other types of artifacts, for example Java classes, use an  
3489 extended version of artifact resolution that is specialized to their nature (eg. instead of  
3490 "namespaces", Java uses "packages"). Descriptions of these more specialized forms of artifact  
3491 resolution are contained in the SCA specifications that deal with those artifact types.

3492 Import and export statements for XML artifacts work at the level of namespaces - so that an  
3493 import statement declares that artifacts from a specified namespace are found in other  
3494 contributions, while an export statement makes all the artifacts from a specified namespace  
3495 available to other contributions.

3496 An import declaration can simply specify the namespace to import. In this case, the locations  
3497 which are searched for artifacts in that namespace are the contribution(s) in the Domain which  
3498 have export declarations for the same namespace, if any. Alternatively an import declaration can  
3499 specify a location from which artifacts for the namespace are obtained, in which case, that specific  
3500 location is searched. There can be multiple import declarations for a given namespace. Where  
3501 multiple import declarations are made for the same namespace, all the locations specified MUST  
3502 be searched in lexical order. [ASM12022]

3503 For an XML namespace, artifacts can be declared in multiple locations - for example a given  
3504 namespace can have a WSDL declared in one contribution and have an XSD defining XML data  
3505 types in a second contribution.

3506 If the same artifact is declared in multiple locations, this is not an error. The first location as  
3507 defined by lexical order is chosen. If no locations are specified no order exists and the one chosen  
3508 is implementation dependent.

3509 When a contribution contains a reference to an artifact from a namespace that is declared in an import  
3510 statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the  
3511 SCA runtime MUST resolve artifacts in the following order:

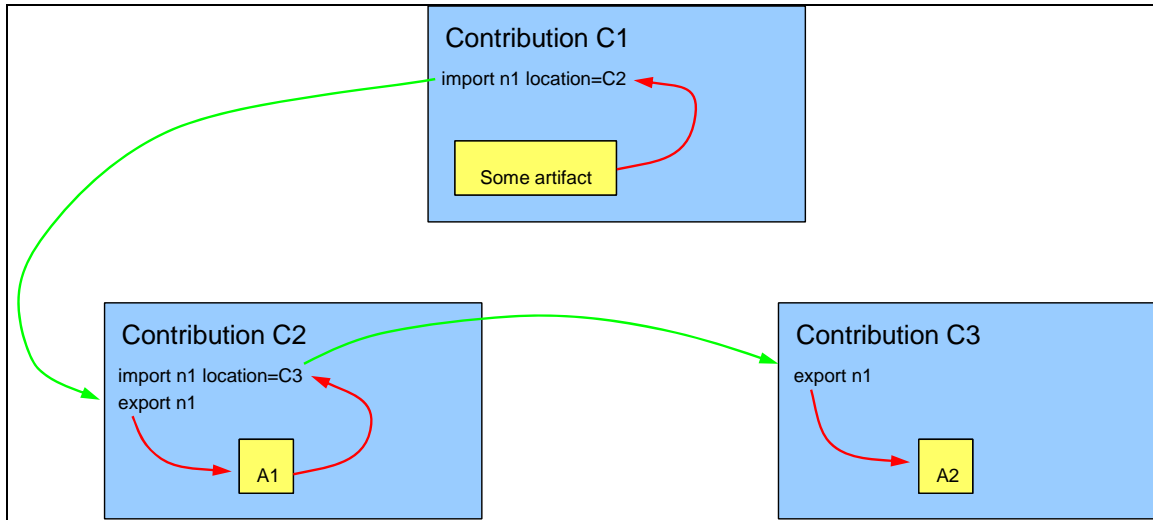
- 3512 1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT  
3513 be searched recursively in order to locate artifacts (i.e. only a one-level search is performed).
- 3514 2. from the contents of the contribution itself. [ASM12023]

3515 When a contribution uses an artifact contained in another contribution through SCA artifact  
3516 resolution, if that artifact itself has dependencies on other artifacts, the SCA runtime MUST resolve  
3517 these dependencies in the context of the contribution containing the artifact, not in the context of  
3518 the original contribution. [ASM12031]

3519 For example:

- 3520 • a first contribution "C1" references an artifact "A1" in the namespace "n1" and imports the  
3521 "n1" namespace from a second contribution "C2".
- 3522 • in contribution "C2" the artifact "A1" in the "n1" namespace references an artifact "A2"  
3523 also in the "n1" namespace", which is resolved through an import of the "n1" namespace  
3524 in "C2" which specifies the location "C3".

3525



3526

3527 *Figure 14: Example of SCA Artifact Resolution between Contributions*

3528 The "A2" artifact is contained within the third contribution "C3" from which it is resolved by the  
3529 contribution "C2". The "C3" contribution is never used to resolve artifacts directly for the "C1"  
3530 contribution, since "C3" is not declared as an import location for "C1".

3531 For example, if for a contribution "C1", an import is used to resolve a composite "X1" contained in  
3532 contribution "C2", and composite "X1" contains references to other artifacts such as WSDL files or  
3533 XSDs, those references in "X1" are resolved in the context of contribution "C2" and not in the  
3534 context of contribution "C1".

3535 The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through  
3536 resolving an import statement. [ASM12024]

3537 The SCA runtime MUST raise an error if an artifact cannot be resolved by using artifact-related or  
3538 packaging-related artifact resolution mechanisms, if present, by searching locations identified by  
3539 the import statements of the contribution, if present, and by searching the contents of the  
3540 contribution. [ASM12025]

### 3541 11.2.2 SCA Contribution Metadata Document

3542 The contribution can contain a document that declares runnable composites, exported definitions  
3543 and imported definitions. The document is found at the path of META-INF/sca-contribution.xml  
3544 relative to the root of the contribution. Frequently some SCA metadata needs to be specified by  
3545 hand while other metadata is generated by tools (such as the <import> elements described  
3546 below). To accommodate this, it is also possible to have an identically structured document at  
3547 META-INF/sca-contribution-generated.xml. If this document exists (or is generated on an as-  
3548 needed basis), it will be merged into the contents of sca-contribution.xml, with the entries in sca-  
3549 contribution.xml taking priority if there are any conflicting declarations.

3550 An SCA runtime MUST make the <import/> and <export/> elements found in the META-INF/sca-  
3551 contribution.xml and META-INF/sca-contribution-generated.xml files available for the SCA artifact  
3552 resolution process. [ASM12026] An SCA runtime MUST reject files that do not conform to the  
3553 schema declared in sca-contribution.xsd. [ASM12027] An SCA runtime MUST merge the contents  
3554 of sca-contribution-generated.xml into the contents of sca-contribution.xml, with the entries in  
3555 sca-contribution.xml taking priority if there are any conflicting declarations. [ASM12028]

3556 The format of the document is:

```
3557
3558 <?xml version="1.0" encoding="ASCII"?>
3559 <!-- sca-contribution pseudo-schema -->
3560 <contribution xmlns=http://docs.oasis-open.org/ns/opencsa/sca/200903>
3561
```

```
3562     <deployable composite="xs:QName"/>*
3563     <import namespace="xs:String" location="xs:AnyURI"?/>*
3564     <export namespace="xs:String"/>*
3565
3566 </contribution>
3567
```

3568 **deployable element:** Identifies a composite which is a composite within the contribution that is a  
3569 composite intended for potential inclusion into the virtual domain-level composite. Other  
3570 composites in the contribution are not intended for inclusion but only for use by other composites.  
3571 New composites can be created for a contribution after it is installed, by using the [add Deployment](#)  
3572 [Composite](#) capability and the add To Domain Level Composite capability. An SCA runtime MAY  
3573 deploy the composites in <deployable/> elements found in the META-INF/sca-contribution.xml  
3574 and META-INF/sca-contribution-generated.xml files. [ASM12029]

3575 Attributes of the deployable element:

- 3576 • **composite (1..1)** – The QName of a composite within the contribution.

3577

3578 **Export element:** A declaration that artifacts belonging to a particular namespace are exported  
3579 and are available for use within other contributions. An export declaration in a contribution  
3580 specifies a namespace, all of whose definitions are considered to be exported. By default,  
3581 definitions are not exported.

3582 The SCA artifact export is useful for SCA Domains containing heterogeneous mixtures of  
3583 contribution packagings and technologies, where artifact-related or packaging-related mechanisms  
3584 are unlikely to work across different kinds of contribution.

3585 Attributes of the export element:

- 3586 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the  
3587 @namespace attribute of the export element SHOULD be the namespace URI for the  
3588 exported definitions. [ASM12030] For XML technologies that define multiple *symbol spaces*  
3589 that can be used within one namespace (e.g. WSDL port types are a different symbol  
3590 space from WSDL bindings), all definitions from all symbol spaces are exported.

3591 Technologies that use naming schemes other than QNames use a different export element  
3592 from the same substitution group as the the SCA <export> element. The element used  
3593 identifies the technology, and can use any value for the namespace that is appropriate for  
3594 that technology. For example, <export.java> can be used to export java definitions, in  
3595 which case the namespace is a fully qualified package name.

3597

3598 **Import element:** Import declarations specify namespaces of definitions that are needed by the  
3599 definitions and implementations within the contribution, but which are not present in the  
3600 contribution. It is expected that in most cases import declarations will be generated based on  
3601 introspection of the contents of the contribution. In this case, the import declarations would be  
3602 found in the META-INF/ sca-contribution-generated.xml document.

3603 Attributes of the import element:

- 3604 • **namespace (1..1)** – For XML definitions, which are identified by QNames, the namespace  
3605 is the namespace URI for the imported definitions. For XML technologies that define  
3606 multiple *symbol spaces* that can be used within one namespace (e.g. WSDL port types are  
3607 a different symbol space from WSDL bindings), all definitions from all symbol spaces are  
3608 imported.

3609 Technologies that use naming schemes other than QNames use a different import element  
3610 from the same substitution group as the the SCA <import> element. The element used  
3611 identifies the technology, and can use any value for the namespace that is appropriate for  
3612 that technology. For example, <import.java> can be used to import java definitions, in  
3613 which case the namespace is a fully qualified package name.



3615           • **location (0..1)** – a URI to resolve the definitions for this import. SCA makes no specific  
3616 requirements for the form of this URI, nor the means by which it is resolved. It can point  
3617 to another contribution (through its URI) or it can point to some location entirely outside  
3618 the SCA Domain.  
3619

3620 It is expected that SCA runtimes can define implementation specific ways of resolving location  
3621 information for artifact resolution between contributions. These mechanisms will however usually  
3622 be limited to sets of contributions of one runtime technology and one hosting environment.

3623 In order to accommodate imports of artifacts between contributions of disparate runtime  
3624 technologies, it is strongly suggested that SCA runtimes honor SCA contribution URIs as location  
3625 specification.

3626 SCA runtimes that support contribution URIs for cross-contribution resolution of SCA artifacts are  
3627 expected to do so similarly when used as @schemaLocation and @wsdlLocation and other artifact  
3628 location specifications.

3629 The order in which the import statements are specified can play a role in this mechanism. Since  
3630 definitions of one namespace can be distributed across several artifacts, multiple import  
3631 declarations can be made for one namespace.

3632 The location value is only a default, and dependent contributions listed in the call to  
3633 installContribution can override the value if there is a conflict. However, the specific mechanism  
3634 for resolving conflicts between contributions that define conflicting definitions is implementation  
3635 specific.

3636 If the value of the @location attribute is an SCA contribution URI, then the contribution packaging  
3637 can become dependent on the deployment environment. In order to avoid such a dependency, it  
3638 is recommended that dependent contributions are specified only when deploying or updating  
3639 contributions as specified in the section 'Operations for Contributions' below.

### 3640 **11.2.3 Contribution Packaging using ZIP**

3641 SCA allows many different packaging formats that SCA runtimes can support, but **SCA requires**  
3642 **that all runtimes MUST support the ZIP packaging format for contributions.** [ASM12006] This  
3643 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3644 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3645 contribution.xml" file and there can also be a "META-INF/sca-contribution-generated.xml" file in  
3646 the package. SCA defined artifacts as well as non-SCA defined artifacts such as object files, WSDL  
3647 definition, Java classes can be present anywhere in the ZIP archive,

3648 A definition of the ZIP file format is published by PKWARE in [an Application Note on the .ZIP file](#)  
3649 [format \[12\]](#).

### 3650 **11.3 Installed Contribution**

3651 As noted in the section above, the contents of a contribution do not need to be modified in order  
3652 to install and use it within a Domain. An *installed contribution* is a contribution with all of the  
3653 associated information necessary in order to execute *deployable composites* within the  
3654 contribution.

3655 An installed contribution is made up of the following things:

- 3656           • Contribution Packaging – the contribution that will be used as the starting point for  
3657           resolving all references
- 3658           • Contribution base URI
- 3659           • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3660           the import statements from the root composite and from other dependent contributions
  - 3661               ◦ Dependent contributions might or might not be shared with other installed  
3662               contributions.

3663                   o When the snapshot of any contribution is taken is implementation defined, ranging  
3664                   from the time the contribution is installed to the time of execution

3665                   • Deployment-time composites.  
3666                   These are composites that are added into an installed contribution after it has been  
3667                   deployed. This makes it possible to provide final configuration and access to  
3668                   implementations within a contribution without having to modify the contribution. These do  
3669                   not have to be provided as composites that already exist within the contribution can also  
3670                   be used for deployment.

3671                   Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3672                   fully qualified class names in Java).

3673                   If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3674                   algorithm used to determine the qualified name to use is implementation dependent.

3675                   Implementations of SCA MAY also raise an error if there are conflicting names exported from  
3676                   multiple contributions. [ASM12007]

### 3677   **11.3.1 Installed Artifact URIs**

3678                   When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3679                   constructed by starting with the base URI of the contribution and adding the relative URI of each  
3680                   artifact (recalling that SCA demands that any packaging format be able to offer up its artifacts in a  
3681                   single hierarchy).

## 3682   **11.4 Operations for Contributions**

3683                   SCA Runtimes provide the following conceptual functionality associated with contributions to the  
3684                   Domain (meaning the function might not be represented as addressable services and also  
3685                   meaning that equivalent functionality might be provided in other ways). An SCA runtime MAY  
3686                   provide the contribution operation functions (install Contribution, update Contribution, add  
3687                   Deployment Composite, update Deployment Composite, remove Contribution). [ASM12008]

### 3688   **11.4.1 install Contribution & update Contribution**

3689                   Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3690                   supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3691                   contributions that are used to resolve the dependencies of the root contribution and other  
3692                   dependent contributions. These override any dependent contributions explicitly listed via the  
3693                   @location attribute in the import statements of the contribution.

3694                   SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3695                   means that all other exported artifacts can be used from that contribution. Because of this, the  
3696                   dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3697                   what is being used from each one.

3698                   Each dependent contribution is also an installed contribution, with its own dependent  
3699                   contributions. By default these dependent contributions of the dependent contributions (which we  
3700                   will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3701                   contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3702                   definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3703                   included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3704                   Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3705                   MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3706                   Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3707                   creator of a Domain is careful to avoid creating duplicate definitions for the same qualified name,  
3708                   then it is easy for this list to be generated by tooling.

### 3709   **11.4.2 add Deployment Composite & update Deployment Composite**

3710                   Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3711                   data structure, not an existing resource in the Domain) to the contribution identified by a supplied

3712 contribution URI. The added or updated deployment composite is given a relative URI that  
3713 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3714 run within the context of a installed contribution (any component implementations or other  
3715 definitions are resolved within that contribution), this functionality makes it possible for the  
3716 deployer to create a composite with final configuration and wiring decisions and add it to an  
3717 installed contribution without having to modify the contents of the root contribution.

3718 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3719 It is then possible for those to be given component names by a (possibly generated) composite  
3720 that is added into the installed contribution, without having to modify the packaging.

### 3721 11.4.3 remove Contribution

3722 Removes the deployed contribution identified by a supplied contribution URI.

## 3723 11.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3724 For certain types of artifact, there are existing and commonly used mechanisms for referencing a  
3725 specific concrete location where the artifact can be resolved.

3726 Examples of these mechanisms include:

- 3727 • For WSDL files, the **@wsdlLocation** attribute is a hint that has a URI value pointing to the  
3728 place holding the WSDL itself.
- 3729 • For XSDs, the **@schemaLocation** attribute is a hint which matches the namespace to a  
3730 URI where the XSD is found.

3731 **Note:** In neither of these cases is the runtime obliged to use the location hint and the URI does  
3732 not have to be dereferenced.

3733 SCA permits the use of these mechanisms Where present, non-SCA artifact resolution  
3734 mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.  
3735 [ASM12010] However, use of these mechanisms is discouraged because tying assemblies to  
3736 addresses in this way makes the assemblies less flexible and prone to errors when changes are  
3737 made to the overall SCA Domain.

3738 **Note:** If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to  
3739 find the resource indicated when using the mechanism (e.g. the URI is incorrect or invalid, say)  
3740 the SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms  
3741 as an alternative. [ASM12011]

## 3742 11.6 Domain-Level Composite

3743 The domain-level composite is a virtual composite, in that it is not defined by a composite  
3744 definition document. Rather, it is built up and modified through operations on the Domain.  
3745 However, in other respects it is very much like a composite, since it contains components, wires,  
3746 services and references.

3747 The value of @autowire for the logical Domain composite MUST be autowire="false". [ASM12012]

3748 For components at the Domain level, with References for which @autowire="true" applies, the  
3749 behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms:

3750 1) The SCA runtime MAY disallow deployment of any components with autowire References. In  
3751 this case, the SCA runtime MUST raise an exception at the point where the component is  
3752 deployed.

3753 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component  
3754 is deployed and not update those targets when later deployment actions occur.

3755 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later  
3756 deployment actions occur resulting in updated reference targets which match the new Domain  
3757 configuration. How the new configuration of the reference takes place is described by the relevant  
3758 client and implementation specifications.

3759 [ASM12013]

3760 The abstract domain-level functionality for modifying the domain-level composite is as follows,  
3761 although a runtime can supply equivalent functionality in a different form:

### 3762 **11.6.1 add To Domain-Level Composite**

3763 This functionality adds the composite identified by a supplied URI to the Domain Level Composite.  
3764 The supplied composite URI refers to a composite within an installed contribution. The  
3765 composite's installed contribution determines how the composite's artifacts are resolved (directly  
3766 and indirectly). The supplied composite is added to the domain composite with semantics that  
3767 correspond to the domain-level composite having an <include> statement that references the  
3768 supplied composite. All of the composites components become top-level components and the  
3769 component services become externally visible services (eg. they would be present in a WSDL  
3770 description of the Domain). The meaning of any promoted services and references in the supplied  
3771 composite is not defined; since there is no composite scope outside the domain composite, the  
3772 usual idea of promotion has no utility.

### 3773 **11.6.2 remove From Domain-Level Composite**

3774 Removes from the Domain Level composite the elements corresponding to the composite  
3775 identified by a supplied composite URI. This means that the removal of the components, wires,  
3776 services and references originally added to the domain level composite by the identified  
3777 composite.

### 3778 **11.6.3 get Domain-Level Composite**

3779 Returns a <composite> definition that has an <include> line for each composite that had been  
3780 added to the domain level composite. It is important to note that, in dereferencing the included  
3781 composites, any referenced artifacts are resolved in terms of that installed composite.

### 3782 **11.6.4 get QName Definition**

3783 In order to make sense of the domain-level composite (as returned by get Domain-Level  
3784 Composite), it needs to be possible to get the definitions for named artifacts in the included  
3785 composites. This functionality takes the supplied URI of an installed contribution (which provides  
3786 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
3787 a QName, e.g. wsdl:PortType). The result is a single definition, in whatever form is appropriate  
3788 for that definition type.

3789 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
3790 need to exist in some form, but not necessarily as a service operation with exactly this signature.

## 3791 **11.7 Dynamic Behaviour of Wires in the SCA Domain**

3792 For components with references which are at the Domain level, there is the potential for dynamic  
3793 behaviour when the wires for a component reference change (this can only apply to component  
3794 references at the Domain level and not to components within composites used as implementations):

3795 The configuration of the wires for a component reference of a component at the Domain level can change  
3796 by means of deployment actions:

- 3797 1. <wire/> elements can be added, removed or replaced by deployment actions
- 3798 2. Components can be updated by deployment actions (i.e. this can change the component reference  
3799 configuration)
- 3800 3. Components which are the targets of reference wires can be updated or removed
- 3801 4. Components can be added that are potential targets for references which are marked with  
3802 @autowire=true

3803

3804 Where <wire/> elements are added, removed or replaced by deployment actions, the components whose  
3805 references are affected by those deployment actions MAY have their references updated by the SCA  
3806 runtime dynamically without the need to stop and start those components. [ASM12014]

3807 Where components are updated by deployment actions (their configuration is changed in some way,  
3808 which includes changing the wires of component references), the new configuration MUST apply to all  
3809 new instances of those components once the update is complete. [ASM12015] An SCA runtime MAY  
3810 choose to maintain existing instances with the old configuration of components updated by deployment  
3811 actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.  
3812 [ASM12016]

3813 Where a component that is the target of a wire is removed, without the wire being changed, then future  
3814 invocations of the reference that use that wire SHOULD fail with a ServiceUnavailable fault. If the wire is  
3815 the result of the autowire process, the SCA runtime MUST:

- 3816 • either cause future invocation of the target component's services to fail with a  
3817 ServiceUnavailable fault
- 3818 • or alternatively, if an alternative target component is available that satisfies the autowire  
3819 process, update the reference of the source component [ASM12017]

3820 Where a component that is the target of a wire is updated, future invocations of that reference SHOULD  
3821 use the updated component. [ASM12018]

3822 Where a component is added to the Domain that is a potential target for a domain level component  
3823 reference where that reference is marked as @autowire=true, the SCA runtime MUST:

- 3824 - either update the references for the source component once the new component is running.
- 3825 - or alternatively, defer the updating of the references of the source component until the source  
3826 component is stopped and restarted. [ASM12020]

## 3827 11.8 Dynamic Behaviour of Component Property Values

3828 For a domain level component with a Property whose value is obtained from a Domain-level Property  
3829 through the use of the @source attribute, if the domain level property is updated by means of deployment  
3830 actions, the SCA runtime MUST

- 3831 - either update the property value of the domain level component once the update of the domain  
3832 property is complete
- 3833 - or defer the updating of the component property value until the component is stopped and  
3834 restarted

3835

---

## 3836 12SCA Runtime Considerations

3837 This section describes aspects of an SCA Runtime that are defined by this specification.

### 3838 12.1 Error Handling

3839 The SCA Assembly specification identifies situations where the configuration of the SCA Domain and its  
3840 contents are in error. When one of these situations occurs, the specification requires that the SCA  
3841 Runtime that is interacting with the SCA Domain and the artifacts it contains recognises that there is an  
3842 error, raise the error in a suitable manner and also refuse to run components and services that are in  
3843 error.

3844 The SCA Assembly specification is not prescriptive about the functionality of an SCA Runtime and the  
3845 specification recognizes that there can be a range of design points for an SCA runtime. As a result, the  
3846 SCA Assembly specification describes a range of error handling approaches which can be adopted by an  
3847 SCA runtime.

#### 3848 12.1.1 Errors which can be Detected at Deployment Time

3849 Some error situations can be detected at the point that artifacts are deployed to the Domain. An example  
3850 is a composite document that is invalid in a way that can be detected by static analysis, such as  
3851 containing a component with two services with the same @name attribute.

3852 An SCA runtime SHOULD detect errors at deployment time where those errors can be found through  
3853 static analysis. [ASM14001] The SCA runtime SHOULD prevent deployment of contributions that are in  
3854 error, and raise the error to the process performing the deployment (e.g. write a message to an interactive  
3855 console or write a message to a log file). [ASM14002]

3856 The SCA Assembly specification recognizes that there are reasons why a particular SCA runtime finds it  
3857 desirable to deploy contributions that contain errors (e.g. to assist in the process of development and  
3858 debugging) - and as a result also supports an error handling strategy that is based on detecting problems  
3859 at runtime. However, it is wise to consider reporting problems at an early stage in the deployment  
3860 process.

#### 3861 12.1.2 Errors which are Detected at Runtime

3862 An SCA runtime can detect problems at runtime. These errors can include some which can be found  
3863 from static analysis (e.g. the inability to wire a reference because the target service does not exist in the  
3864 Domain) and others that can only be discovered dynamically (e.g. the inability to invoke some remote  
3865 Web service because the remote endpoint is unavailable).

3866 Where errors can be detected through static analysis, the principle is that components that are known to  
3867 be in error are not run. So, for example, if there is a component with a required reference (multiplicity 1..1  
3868 or 1..n) which is not wired, best practice is that the component is not run. If an attempt is made to invoke  
3869 a service operation of that component, a "ServiceUnavailable" fault is raised to the invoker. It is also  
3870 regarded as best practice that errors of this kind are also raised through appropriate management  
3871 interfaces, for example to the deployer or to the operator of the system.

3872 Where errors are only detected at runtime, when the error is detected an error MUST be raised to the  
3873 component that is attempting the activity concerned with the error. [ASM14003] For example, if a  
3874 component invokes an operation on a reference, but the target service is unavailable, a  
3875 "ServiceUnavailable" fault is raised to the component. When an error that could have been detected  
3876 through static analysis is detected and raised at runtime for a component, the component SHOULD NOT  
3877 be run until the error is fixed. [ASM14004] Such errors can be fixed by redeployment or deployment of  
3878 other components in the domain.

---

## 3879 13 Conformance

3880 The XML schema pointed to by the RDDL document at the namespace URI, defined by this specification,  
3881 are considered to be authoritative and take precedence over the XML schema defined in the appendix of  
3882 this  
3883 document.

3884 An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd, sca-interface-  
3885 wsdl.xsd, sca-implementation-composite.xsd and sca-binding-sca.xsd schema. [ASM13001]

3886 An SCA runtime MUST reject a contribution file that does not conform to the sca-contribution.xsd schema.  
3887 [ASM13002]

3888 An SCA runtime MUST reject a definitions file that does not conform to the sca-definitions.xsd schema.  
3889 [ASM13003]

3890 There are two categories of artifacts that this specification defines conformance for: SCA Documents and  
3891 SCA Runtimes.

### 3892 13.1 SCA Documents

3893 For a document to be a valid SCA Document, it MUST comply with one of the SCA document types  
3894 below:

#### 3895 **SCA Composite Document:**

3896 An SCA Composite Document is a file that MUST have an SCA <composite/> element as its root  
3897 element and MUST conform to the sca-core-1.1.xsd schema and MUST comply with the  
3898 additional constraints on the document contents as defined in Appendix C.

#### 3899 **SCA ComponentType Document:**

3900 An SCA ComponentType Document is a file that MUST have an SCA <componentType/>  
3901 element as its root element and MUST conform to the sca-core-1.1.xsd schema and MUST  
3902 comply with the additional constraints on the document contents as defined in  
3903 Appendix C.

#### 3904 **SCA ConstrainingType Document:**

3905 An SCA ConstrainingType Document is a file that MUST have an SCA <constrainingType/>  
3906 element as its root element and MUST conform to the sca-core-1.1.xsd schema and MUST  
3907 comply with the additional constraints on the document contents as defined in Appendix C.

#### 3908 **SCA Definitions Document:**

3909 An SCA Definitions Document is a file that MUST have an SCA <definitions/> element as its root  
3910 and MUST conform to the sca-definition-1.1.xsd schema and MUST comply with the additional  
3911 constraints on the document contents as defined in Appendix C.

#### 3912 **SCA Contribution Document:**

3913 An SCA Contribution Document is a file that MUST have an SCA <contribution/> element as its  
3914 root element and MUST conform to the sca-contribution-1.1.xsd schema and MUST comply with  
3915 the additional constraints on the document contents as defined in Appendix C.

#### 3916 **SCA Interoperable Packaging Document:**

3917 A ZIP file containing SCA Documents and other related artifacts. The ZIP file SHOULD contain a  
3918 top-level "META-INF" directory, and SHOULD contain a "META-INF/sca-contribution.xml" file, and  
3919 MAY contain a "META-INF/sca-contribution-generated.xml" file.

3920

3921

## 3922 **13.2 SCA Runtime**

3923 An implementation that claims to conform to the requirements of an SCA Runtime defined in this  
3924 specification MUST meet the following conditions:

- 3925 1. The implementation MUST comply with all statements in [Appendix C](#): Conformance Items related to  
3926 an SCA Runtime, notably all MUST statements have to be implemented.
- 3927 2. The implementation MUST conform to the SCA Policy Framework v 1.1 Specification [Policy].
- 3928 3. The implementation MUST support and comply with at least one of the OpenCSA Member Section  
3929 adopted implementation types.
- 3930 4. The implementation MUST support binding.sca and MUST support and conform to the SCA Web  
3931 Service Binding Specification v 1.1.

3932



3933

## A. XML Schemas

3934

### A.1 sca.xsd

3935

```
3936 <?xml version="1.0" encoding="UTF-8"?>
3937 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
3938     OASIS trademark, IPR and other policies apply. -->
3939 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3940     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3941     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903">
3942
3943     <include schemaLocation="sca-core-1.1-cd03.xsd"/>
3944
3945     <include schemaLocation="sca-interface-java-1.1-cd03.xsd"/>
3946     <include schemaLocation="sca-interface-wsdl-1.1-cd03.xsd"/>
3947     <include schemaLocation="sca-interface-cpp-1.1-cd02.xsd"/>
3948     <include schemaLocation="sca-interface-c-1.1-cd02.xsd"/>
3949
3950     <include schemaLocation="sca-implementation-java-1.1-cd01.xsd"/>
3951     <include schemaLocation="sca-implementation-composite-1.1-cd03.xsd"/>
3952     <include schemaLocation="sca-implementation-cpp-1.1-cd02.xsd"/>
3953     <include schemaLocation="sca-implementation-c-1.1-cd02.xsd"/>
3954     <include schemaLocation="sca-implementation-bpel-1.1-cd02.xsd"/>
3955
3956     <include schemaLocation="sca-binding-ws-1.1-cd02.xsd"/>
3957     <include schemaLocation="sca-binding-jms-1.1-cd02.xsd"/>
3958     <include schemaLocation="sca-binding-jca-1.1-cd02.xsd"/>
3959     <include schemaLocation="sca-binding-sca-1.1-cd03.xsd"/>
3960
3961     <include schemaLocation="sca-definitions-1.1-cd03.xsd"/>
3962     <include schemaLocation="sca-policy-1.1-cd02.xsd"/>
3963
3964     <include schemaLocation="sca-contribution-1.1-cd03.xsd"/>
3965     <include schemaLocation="sca-contribution-cpp-1.1-cd02.xsd"/>
3966     <include schemaLocation="sca-contribution-c-1.1-cd02.xsd"/>
3967
3968 </schema>
3969
```

3970

### A.2 sca-core.xsd

3971

```
3972 <?xml version="1.0" encoding="UTF-8"?>
3973 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
3974     OASIS trademark, IPR and other policies apply. -->
3975 <schema xmlns="http://www.w3.org/2001/XMLSchema"
3976     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3977     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
3978     elementFormDefault="qualified">
3979
3980     <import namespace="http://www.w3.org/XML/1998/namespace"
3981             schemaLocation="http://www.w3.org/2001/xml.xsd"/>
3982
```

```

3983 <!-- Common extension base for SCA definitions -->
3984 <complexType name="CommonExtensionBase">
3985   <sequence>
3986     <element ref="sca:documentation" minOccurs="0"
3987       maxOccurs="unbounded"/>
3988   </sequence>
3989   <anyAttribute namespace="##other" processContents="lax"/>
3990 </complexType>
3991
3992 <element name="documentation" type="sca:Documentation"/>
3993 <complexType name="Documentation" mixed="true">
3994   <sequence>
3995     <any namespace="##other" processContents="lax" minOccurs="0"
3996       maxOccurs="unbounded"/>
3997   </sequence>
3998   <attribute ref="xml:lang"/>
3999 </complexType>
4000
4001 <!-- Component Type -->
4002 <element name="componentType" type="sca:ComponentType"/>
4003 <complexType name="ComponentType">
4004   <complexContent>
4005     <extension base="sca:CommonExtensionBase">
4006       <sequence>
4007         <element ref="sca:implementation" minOccurs="0"/>
4008         <choice minOccurs="0" maxOccurs="unbounded">
4009           <element name="service" type="sca:ComponentService"/>
4010           <element name="reference"
4011             type="sca:ComponentTypeReference"/>
4012           <element name="property" type="sca:Property"/>
4013         </choice>
4014         <any namespace="##other" processContents="lax" minOccurs="0"
4015           maxOccurs="unbounded"/>
4016       </sequence>
4017       <attribute name="constrainingType" type="QName" use="optional"/>
4018     </extension>
4019   </complexContent>
4020 </complexType>
4021
4022 <!-- Composite -->
4023 <element name="composite" type="sca:Composite"/>
4024 <complexType name="Composite">
4025   <complexContent>
4026     <extension base="sca:CommonExtensionBase">
4027       <sequence>
4028         <element name="include" type="anyURI" minOccurs="0"
4029           maxOccurs="unbounded"/>
4030         <choice minOccurs="0" maxOccurs="unbounded">
4031           <element name="service" type="sca:Service"/>
4032           <element name="property" type="sca:Property"/>
4033           <element name="component" type="sca:Component"/>
4034           <element name="reference" type="sca:Reference"/>
4035           <element name="wire" type="sca:Wire"/>
4036         </choice>
4037         <any namespace="##other" processContents="lax" minOccurs="0"
4038           maxOccurs="unbounded"/>
4039       </sequence>
4040       <attribute name="name" type="NCName" use="required"/>

```

```

4041     <attribute name="targetNamespace" type="anyURI" use="required"/>
4042     <attribute name="local" type="boolean" use="optional"
4043         default="false"/>
4044     <attribute name="autowire" type="boolean" use="optional"
4045         default="false"/>
4046     <attribute name="constrainingType" type="QName" use="optional"/>
4047     <attribute name="requires" type="sca:listOfQNames"
4048         use="optional"/>
4049     <attribute name="policySets" type="sca:listOfQNames"
4050         use="optional"/>
4051     </extension>
4052 </complexContent>
4053 </complexType>
4054
4055 <!-- Contract base type for Service, Reference -->
4056 <complexType name="Contract" abstract="true">
4057     <complexContent>
4058         <extension base="sca:CommonExtensionBase">
4059             <sequence>
4060                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4061                 <element ref="sca:binding" minOccurs="0"
4062                     maxOccurs="unbounded" />
4063                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4064                 <any namespace="##other" processContents="lax" minOccurs="0"
4065                     maxOccurs="unbounded" />
4066             </sequence>
4067             <attribute name="name" type="NCName" use="required" />
4068             <attribute name="requires" type="sca:listOfQNames"
4069                 use="optional" />
4070             <attribute name="policySets" type="sca:listOfQNames"
4071                 use="optional"/>
4072         </extension>
4073     </complexContent>
4074 </complexType>
4075
4076 <!-- Service -->
4077 <complexType name="Service">
4078     <complexContent>
4079         <extension base="sca:Contract">
4080             <attribute name="promote" type="anyURI" use="required"/>
4081         </extension>
4082     </complexContent>
4083 </complexType>
4084
4085 <!-- Interface -->
4086 <element name="interface" type="sca:Interface" abstract="true"/>
4087 <complexType name="Interface" abstract="true">
4088     <complexContent>
4089         <extension base="sca:CommonExtensionBase">
4090             <attribute name="remotable" type="boolean" use="optional"/>
4091             <attribute name="requires" type="sca:listOfQNames"
4092                 use="optional"/>
4093             <attribute name="policySets" type="sca:listOfQNames"
4094                 use="optional"/>
4095         </extension>
4096     </complexContent>
4097 </complexType>
4098

```

```

4099 <!-- Reference -->
4100 <complexType name="Reference">
4101   <complexContent>
4102     <extension base="sca:Contract">
4103       <attribute name="autowire" type="boolean" use="optional"/>
4104       <attribute name="target" type="sca:listOfAnyURIs"
4105         use="optional"/>
4106       <attribute name="wiredByImpl" type="boolean" use="optional"
4107         default="false"/>
4108       <attribute name="multiplicity" type="sca:Multiplicity"
4109         use="optional" default="1..1"/>
4110       <attribute name="promote" type="sca:listOfAnyURIs"
4111         use="required"/>
4112     </extension>
4113   </complexContent>
4114 </complexType>
4115
4116 <!-- Property -->
4117 <complexType name="SCAPropertyBase" mixed="true">
4118   <sequence>
4119     <any namespace="##any" processContents="lax" minOccurs="0"/>
4120     <!-- NOT an extension point; This any exists to accept
4121       the element-based or complex type property
4122       i.e. no element-based extension point under "sca:property" -->
4123   </sequence>
4124   <!-- mixed="true" to handle simple type -->
4125   <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4126   <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4127 </complexType>
4128
4129 <complexType name="Property" mixed="true">
4130   <complexContent mixed="true">
4131     <extension base="sca:SCAPropertyBase">
4132       <attribute name="name" type="NCName" use="required"/>
4133       <attribute name="type" type="QName" use="optional"/>
4134       <attribute name="element" type="QName" use="optional"/>
4135       <attribute name="many" type="boolean" use="optional"
4136         default="false"/>
4137       <attribute name="mustSupply" type="boolean" use="optional"
4138         default="false"/>
4139       <anyAttribute namespace="##any" processContents="lax"/>
4140     </extension>
4141     <!-- extension defines the place to hold default value -->
4142     <!-- an extension point ; attribute-based only -->
4143   </complexContent>
4144 </complexType>
4145
4146 <!-- ConstrainingProperty is equivalent to the Property type but removes
4147   the capability to contain a value -->
4148 <complexType name="ConstrainingProperty" mixed="true">
4149   <complexContent mixed="true">
4150     <restriction base="sca:Property">
4151       <attribute name="name" type="NCName" use="required"/>
4152       <attribute name="type" type="QName" use="optional"/>
4153       <attribute name="element" type="QName" use="optional"/>
4154       <attribute name="many" type="boolean" use="optional"
4155         default="false"/>
4156       <attribute name="mustSupply" type="boolean" use="optional"

```

```

4157         default="false"/>
4158     <anyAttribute namespace="##any" processContents="lax"/>
4159 </restriction>
4160 </complexContent>
4161 </complexType>
4162
4163 <complexType name="PropertyValue" mixed="true">
4164     <complexContent mixed="true">
4165         <extension base="sca:SCAPropertyBase">
4166             <attribute name="name" type="NCName" use="required"/>
4167             <attribute name="type" type="QName" use="optional"/>
4168             <attribute name="element" type="QName" use="optional"/>
4169             <attribute name="many" type="boolean" use="optional"
4170                 default="false"/>
4171             <attribute name="source" type="string" use="optional"/>
4172             <attribute name="file" type="anyURI" use="optional"/>
4173             <anyAttribute namespace="##any" processContents="lax"/>
4174         </extension>
4175         <!-- an extension point ; attribute-based only -->
4176     </complexContent>
4177 </complexType>
4178
4179 <!-- Binding -->
4180 <element name="binding" type="sca:Binding" abstract="true"/>
4181 <complexType name="Binding" abstract="true">
4182     <complexContent>
4183         <extension base="sca:CommonExtensionBase">
4184             <sequence>
4185                 <element ref="sca:wireFormat" minOccurs="0" maxOccurs="1" />
4186                 <element ref="sca:operationSelector" minOccurs="0"
4187                     maxOccurs="1" />
4188             </sequence>
4189             <attribute name="uri" type="anyURI" use="optional"/>
4190             <attribute name="name" type="NCName" use="optional"/>
4191             <attribute name="requires" type="sca:listOfQNames"
4192                 use="optional"/>
4193             <attribute name="policySets" type="sca:listOfQNames"
4194                 use="optional"/>
4195         </extension>
4196     </complexContent>
4197 </complexType>
4198
4199 <!-- Binding Type -->
4200 <element name="bindingType" type="sca:BindingType"/>
4201 <complexType name="BindingType">
4202     <complexContent>
4203         <extension base="sca:CommonExtensionBase">
4204             <sequence>
4205                 <any namespace="##other" processContents="lax" minOccurs="0"
4206                     maxOccurs="unbounded"/>
4207             </sequence>
4208             <attribute name="type" type="QName" use="required"/>
4209             <attribute name="alwaysProvides" type="sca:listOfQNames"
4210                 use="optional"/>
4211             <attribute name="mayProvide" type="sca:listOfQNames"
4212                 use="optional"/>
4213         </extension>
4214     </complexContent>

```

```

4215     </complexType>
4216
4217     <!-- WireFormat Type -->
4218     <element name="wireFormat" type="sca:WireFormatType"/>
4219     <complexType name="WireFormatType" abstract="true">
4220         <sequence>
4221             <any namespace="##other" processContents="lax" minOccurs="0"
4222                 maxOccurs="unbounded" />
4223         </sequence>
4224         <anyAttribute namespace="##other" processContents="lax"/>
4225     </complexType>
4226
4227     <!-- OperationSelector Type -->
4228     <element name="operationSelector" type="sca:OperationSelectorType"/>
4229     <complexType name="OperationSelectorType" abstract="true">
4230         <sequence>
4231             <any namespace="##other" processContents="lax" minOccurs="0"
4232                 maxOccurs="unbounded" />
4233         </sequence>
4234         <anyAttribute namespace="##other" processContents="lax"/>
4235     </complexType>
4236
4237     <!-- Callback -->
4238     <element name="callback" type="sca:Callback"/>
4239     <complexType name="Callback">
4240         <complexContent>
4241             <extension base="sca:CommonExtensionBase">
4242                 <choice minOccurs="0" maxOccurs="unbounded">
4243                     <element ref="sca:binding"/>
4244                     <any namespace="##other" processContents="lax"/>
4245                 </choice>
4246                 <attribute name="requires" type="sca:listOfQNames"
4247                     use="optional"/>
4248                 <attribute name="policySets" type="sca:listOfQNames"
4249                     use="optional"/>
4250             </extension>
4251         </complexContent>
4252     </complexType>
4253
4254     <!-- Component -->
4255     <complexType name="Component">
4256         <complexContent>
4257             <extension base="sca:CommonExtensionBase">
4258                 <sequence>
4259                     <element ref="sca:implementation" minOccurs="0"/>
4260                     <choice minOccurs="0" maxOccurs="unbounded">
4261                         <element name="service" type="sca:ComponentService"/>
4262                         <element name="reference" type="sca:ComponentReference"/>
4263                         <element name="property" type="sca:PropertyValue"/>
4264                     </choice>
4265                     <any namespace="##other" processContents="lax" minOccurs="0"
4266                         maxOccurs="unbounded" />
4267                 </sequence>
4268                 <attribute name="name" type="NCName" use="required"/>
4269                 <attribute name="autowire" type="boolean" use="optional"/>
4270                 <attribute name="constrainingType" type="QName" use="optional"/>
4271                 <attribute name="requires" type="sca:listOfQNames"
4272                     use="optional"/>

```

```

4273         <attribute name="policySets" type="sca:listOfQNames"
4274             use="optional"/>
4275     </extension>
4276 </complexContent>
4277 </complexType>
4278
4279 <!-- Component Service -->
4280 <complexType name="ComponentService">
4281     <complexContent>
4282         <extension base="sca:Contract">
4283             </extension>
4284         </complexContent>
4285     </complexType>
4286
4287 <!-- Constraining Service -->
4288 <complexType name="ConstrainingService">
4289     <complexContent>
4290         <restriction base="sca:ComponentService">
4291             <sequence>
4292                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4293                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4294                 <any namespace="##other" processContents="lax" minOccurs="0"
4295                     maxOccurs="unbounded" />
4296             </sequence>
4297             <attribute name="name" type="NCName" use="required" />
4298         </restriction>
4299     </complexContent>
4300 </complexType>
4301
4302
4303 <!-- Component Reference -->
4304 <complexType name="ComponentReference">
4305     <complexContent>
4306         <extension base="sca:Contract">
4307             <attribute name="autowire" type="boolean" use="optional"/>
4308             <attribute name="target" type="sca:listOfAnyURIs"
4309                 use="optional"/>
4310             <attribute name="wiredByImpl" type="boolean" use="optional"
4311                 default="false"/>
4312             <attribute name="multiplicity" type="sca:Multiplicity"
4313                 use="optional" default="1..1"/>
4314             <attribute name="nonOverridable" type="boolean" use="optional"
4315                 default="false"/>
4316         </extension>
4317     </complexContent>
4318 </complexType>
4319
4320 <!-- Constraining Reference -->
4321 <complexType name="ConstrainingReference">
4322     <complexContent>
4323         <restriction base="sca:ComponentReference">
4324             <sequence>
4325                 <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
4326                 <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
4327                 <any namespace="##other" processContents="lax" minOccurs="0"
4328                     maxOccurs="unbounded" />
4329             </sequence>
4330             <attribute name="name" type="NCName" use="required" />

```

```

4331         <attribute name="autowire" type="boolean" use="optional"/>
4332         <attribute name="wiredByImpl" type="boolean" use="optional"
4333             default="false"/>
4334         <attribute name="multiplicity" type="sca:Multiplicity"
4335             use="optional" default="1..1"/>
4336     </restriction>
4337 </complexContent>
4338 </complexType>
4339
4340 <!-- Component Type Reference -->
4341 <complexType name="ComponentTypeReference">
4342     <complexContent>
4343         <restriction base="sca:ComponentReference">
4344             <sequence>
4345                 <element ref="sca:documentation" minOccurs="0"
4346                     maxOccurs="unbounded"/>
4347                 <element ref="sca:interface" minOccurs="0"/>
4348                 <element ref="sca:binding" minOccurs="0"
4349                     maxOccurs="unbounded"/>
4350                 <element ref="sca:callback" minOccurs="0"/>
4351                 <any namespace="##other" processContents="lax" minOccurs="0"
4352                     maxOccurs="unbounded"/>
4353             </sequence>
4354             <attribute name="name" type="NCName" use="required"/>
4355             <attribute name="autowire" type="boolean" use="optional"/>
4356             <attribute name="wiredByImpl" type="boolean" use="optional"
4357                 default="false"/>
4358             <attribute name="multiplicity" type="sca:Multiplicity"
4359                 use="optional" default="1..1"/>
4360             <attribute name="requires" type="sca:listOfQNames"
4361                 use="optional"/>
4362             <attribute name="policySets" type="sca:listOfQNames"
4363                 use="optional"/>
4364             <anyAttribute namespace="##other" processContents="lax"/>
4365         </restriction>
4366     </complexContent>
4367 </complexType>
4368
4369 <!-- Implementation -->
4370 <element name="implementation" type="sca:Implementation" abstract="true"/>
4371 <complexType name="Implementation" abstract="true">
4372     <complexContent>
4373         <extension base="sca:CommonExtensionBase">
4374             <attribute name="requires" type="sca:listOfQNames"
4375                 use="optional"/>
4376             <attribute name="policySets" type="sca:listOfQNames"
4377                 use="optional"/>
4378         </extension>
4379     </complexContent>
4380 </complexType>
4381
4382 <!-- Implementation Type -->
4383 <element name="implementationType" type="sca:ImplementationType"/>
4384 <complexType name="ImplementationType">
4385     <complexContent>
4386         <extension base="sca:CommonExtensionBase">
4387             <sequence>

```



```

4389         <any namespace="##other" processContents="lax" minOccurs="0"
4390             maxOccurs="unbounded" />
4391     </sequence>
4392     <attribute name="type" type="QName" use="required" />
4393     <attribute name="alwaysProvides" type="sca:listOfQNames"
4394         use="optional" />
4395     <attribute name="mayProvide" type="sca:listOfQNames"
4396         use="optional" />
4397 </extension>
4398 </complexContent>
4399 </complexType>
4400
4401 <!-- Wire -->
4402 <complexType name="Wire">
4403     <complexContent>
4404         <extension base="sca:CommonExtensionBase">
4405             <sequence>
4406                 <any namespace="##other" processContents="lax" minOccurs="0"
4407                     maxOccurs="unbounded" />
4408             </sequence>
4409             <attribute name="source" type="anyURI" use="required" />
4410             <attribute name="target" type="anyURI" use="required" />
4411             <attribute name="replace" type="boolean" use="optional"
4412                 default="false" />
4413         </extension>
4414     </complexContent>
4415 </complexType>
4416
4417 <!-- Include -->
4418 <element name="include" type="sca:Include" />
4419 <complexType name="Include">
4420     <complexContent>
4421         <extension base="sca:CommonExtensionBase">
4422             <attribute name="name" type="QName" />
4423         </extension>
4424     </complexContent>
4425 </complexType>
4426
4427 <!-- Constraining Type -->
4428 <element name="constrainingType" type="sca:ConstrainingType" />
4429 <complexType name="ConstrainingType">
4430     <complexContent>
4431         <extension base="sca:CommonExtensionBase">
4432             <sequence>
4433                 <choice minOccurs="0" maxOccurs="unbounded">
4434                     <element name="service" type="sca:ConstrainingService" />
4435                     <element name="reference"
4436                         type="sca:ConstrainingReference" />
4437                     <element name="property" type="sca:ConstrainingProperty" />
4438                 </choice>
4439                 <any namespace="##other" processContents="lax" minOccurs="0"
4440                     maxOccurs="unbounded" />
4441             </sequence>
4442             <attribute name="name" type="NCName" use="required" />
4443             <attribute name="targetNamespace" type="anyURI" />
4444         </extension>
4445     </complexContent>
4446 </complexType>

```

```

4447
4448 <!-- Intents within WSDL documents -->
4449 <attribute name="requires" type="sca:listOfQNames"/>
4450
4451 <!-- Global attribute definition for @callback to mark a WSDL port type
4452 as having a callback interface defined in terms of a second port
4453 type. -->
4454 <attribute name="callback" type="anyURI"/>
4455
4456 <!-- Miscellaneous simple type definitions -->
4457 <simpleType name="Multiplicity">
4458 <restriction base="string">
4459 <enumeration value="0..1"/>
4460 <enumeration value="1..1"/>
4461 <enumeration value="0..n"/>
4462 <enumeration value="1..n"/>
4463 </restriction>
4464 </simpleType>
4465
4466 <simpleType name="OverrideOptions">
4467 <restriction base="string">
4468 <enumeration value="no"/>
4469 <enumeration value="may"/>
4470 <enumeration value="must"/>
4471 </restriction>
4472 </simpleType>
4473
4474 <simpleType name="listOfQNames">
4475 <list itemType="QName"/>
4476 </simpleType>
4477
4478 <simpleType name="listOfAnyURIs">
4479 <list itemType="anyURI"/>
4480 </simpleType>
4481
4482 </schema>
4483

```

### 4484 **A.3 sca-binding-sca.xsd**

```

4485
4486 <?xml version="1.0" encoding="UTF-8"?>
4487 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
4488 OASIS trademark, IPR and other policies apply. -->
4489 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4490 targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4491 xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4492 elementFormDefault="qualified">
4493
4494 <include schemaLocation="sca-core-1.1-cd03.xsd"/>
4495
4496 <!-- SCA Binding -->
4497 <element name="binding.sca" type="sca:SCABinding"
4498 substitutionGroup="sca:binding"/>
4499 <complexType name="SCABinding">
4500 <complexContent>
4501 <extension base="sca:Binding"/>

```

```
4502     </complexContent>
4503   </complexType>
4504
4505 </schema>
4506
```

## 4507 **A.4 sca-interface-java.xsd**

4508 Is described in the [SCA Java Common Annotations and APIs specification](#) [SCA-Common-Java].  
4509

## 4510 **A.5 sca-interface-wsdl.xsd**

```
4511
4512 <?xml version="1.0" encoding="UTF-8"?>
4513 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
4514     OASIS trademark, IPR and other policies apply. -->
4515 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4516     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4517     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4518     elementFormDefault="qualified">
4519
4520   <include schemaLocation="sca-core-1.1-cd03.xsd"/>
4521
4522   <!-- WSDL Interface -->
4523   <element name="interface.wsdl" type="sca:WSDLPortType"
4524     substitutionGroup="sca:interface"/>
4525   <complexType name="WSDLPortType">
4526     <complexContent>
4527       <extension base="sca:Interface">
4528         <sequence>
4529           <any namespace="##other" processContents="lax" minOccurs="0"
4530             maxOccurs="unbounded"/>
4531         </sequence>
4532         <attribute name="interface" type="anyURI" use="required"/>
4533         <attribute name="callbackInterface" type="anyURI"
4534           use="optional"/>
4535         <anyAttribute namespace="##any" processContents="lax"/>
4536       </extension>
4537     </complexContent>
4538   </complexType>
4539 </schema>
4540
4541
```

## 4542 **A.6 sca-implementation-java.xsd**

4543 Is described in the [Java Component Implementation specification](#) [SCA-Java]

## 4544 **A.7 sca-implementation-composite.xsd**

```
4545
4546 <?xml version="1.0" encoding="UTF-8"?>
4547 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
4548     OASIS trademark, IPR and other policies apply. -->
4549 <schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```

4550     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4551     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4552     elementFormDefault="qualified">
4553
4554     <include schemaLocation="sca-core-1.1-cd03.xsd"/>
4555
4556     <!-- Composite Implementation -->
4557     <element name="implementation.composite" type="sca:SCAImplementation"
4558         substitutionGroup="sca:implementation"/>
4559     <complexType name="SCAImplementation">
4560         <complexContent>
4561             <extension base="sca:Implementation">
4562                 <sequence>
4563                     <any namespace="##other" processContents="lax" minOccurs="0"
4564                         maxOccurs="unbounded"/>
4565                 </sequence>
4566                 <attribute name="name" type="QName" use="required"/>
4567             </extension>
4568         </complexContent>
4569     </complexType>
4570
4571 </schema>
4572

```

## 4573 **A.8 sca-binding-webservice.xsd**

4574 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4575 **A.9 sca-binding-jms.xsd**

4576 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4577 **A.10 sca-policy.xsd**

4578 Is described in [the SCA Policy Framework specification \[10\]](#)

4579

## 4580 **A.11 sca-contribution.xsd**

4581

```

4582 <?xml version="1.0" encoding="UTF-8"?>
4583 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
4584     OASIS trademark, IPR and other policies apply. -->
4585 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4586     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4587     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4588     elementFormDefault="qualified">
4589
4590     <include schemaLocation="sca-core-1.1-cd03.xsd"/>
4591
4592     <!-- Contribution -->
4593     <element name="contribution" type="sca:ContributionType"/>
4594     <complexType name="ContributionType">
4595         <complexContent>
4596             <extension base="sca:CommonExtensionBase">
4597                 <sequence>
4598                     <element name="deployable" type="sca:DeployableType"

```

```

4599         maxOccurs="unbounded" />
4600     <element ref="sca:importBase" minOccurs="0"
4601         maxOccurs="unbounded" />
4602     <element ref="sca:exportBase" minOccurs="0"
4603         maxOccurs="unbounded" />
4604     <any namespace="##other" processContents="lax" minOccurs="0"
4605         maxOccurs="unbounded" />
4606     </sequence>
4607 </extension>
4608 </complexContent>
4609 </complexType>
4610
4611 <!-- Deployable -->
4612 <complexType name="DeployableType">
4613     <complexContent>
4614         <extension base="sca:CommonExtensionBase">
4615             <sequence>
4616                 <any namespace="##other" processContents="lax" minOccurs="0"
4617                     maxOccurs="unbounded" />
4618             </sequence>
4619             <attribute name="composite" type="QName" use="required" />
4620         </extension>
4621     </complexContent>
4622 </complexType>
4623
4624 <!-- Import -->
4625 <element name="importBase" type="sca:Import" abstract="true" />
4626 <complexType name="Import" abstract="true">
4627     <complexContent>
4628         <extension base="sca:CommonExtensionBase">
4629             <sequence>
4630                 <any namespace="##other" processContents="lax" minOccurs="0"
4631                     maxOccurs="unbounded" />
4632             </sequence>
4633         </extension>
4634     </complexContent>
4635 </complexType>
4636
4637 <element name="import" type="sca:ImportType"
4638     substitutionGroup="sca:importBase" />
4639 <complexType name="ImportType">
4640     <complexContent>
4641         <extension base="sca:Import">
4642             <attribute name="namespace" type="string" use="required" />
4643             <attribute name="location" type="anyURI" use="optional" />
4644         </extension>
4645     </complexContent>
4646 </complexType>
4647
4648 <!-- Export -->
4649 <element name="exportBase" type="sca:Export" abstract="true" />
4650 <complexType name="Export" abstract="true">
4651     <complexContent>
4652         <extension base="sca:CommonExtensionBase">
4653             <sequence>
4654                 <any namespace="##other" processContents="lax" minOccurs="0"
4655                     maxOccurs="unbounded" />
4656             </sequence>

```

```

4657         </extension>
4658     </complexContent>
4659 </complexType>
4660
4661 <element name="export" type="sca:ExportType"
4662     substitutionGroup="sca:exportBase"/>
4663 <complexType name="ExportType">
4664     <complexContent>
4665         <extension base="sca:Export">
4666             <attribute name="namespace" type="string" use="required"/>
4667         </extension>
4668     </complexContent>
4669 </complexType>
4670
4671 </schema>
4672

```

## 4673 A.12 sca-definitions.xsd

```

4674
4675 <?xml version="1.0" encoding="UTF-8"?>
4676 <!-- Copyright(C) OASIS(R) 2005,2009. All Rights Reserved.
4677     OASIS trademark, IPR and other policies apply. -->
4678 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4679     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4680     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200903"
4681     elementFormDefault="qualified">
4682
4683     <include schemaLocation="sca-core-1.1-cd03.xsd"/>
4684     <include schemaLocation="sca-policy-1.1-cd02.xsd"/>
4685
4686     <!-- Definitions -->
4687     <element name="definitions" type="sca:tDefinitions"/>
4688     <complexType name="tDefinitions">
4689         <complexContent>
4690             <extension base="sca:CommonExtensionBase">
4691                 <choice minOccurs="0" maxOccurs="unbounded">
4692                     <element ref="sca:intent"/>
4693                     <element ref="sca:policySet"/>
4694                     <element ref="sca:binding"/>
4695                     <element ref="sca:bindingType"/>
4696                     <element ref="sca:implementationType"/>
4697                     <any namespace="##other" processContents="lax"
4698                         minOccurs="0" maxOccurs="unbounded"/>
4699                 </choice>
4700                 <attribute name="targetNamespace" type="anyURI" use="required"/>
4701             </extension>
4702         </complexContent>
4703     </complexType>
4704
4705 </schema>
4706

```

---

4707

## B. SCA Concepts

4708

### B.1 Binding

4709 **Bindings** are used by services and references. References use bindings to describe the access  
4710 mechanism used to call the service to which they are wired. Services use bindings to describe the  
4711 access mechanism(s) that clients use to call the service.

4712 SCA supports multiple different types of bindings. Examples include **SCA service, Web service,**  
4713 **stateless session EJB, database stored procedure, EIS service.** SCA provides an extensibility  
4714 mechanism by which an SCA runtime can add support for additional binding types.

4715

4716

### B.2 Component

4717 **SCA components** are configured instances of **SCA implementations**, which provide and consume  
4718 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines  
4719 an **extensibility mechanism** that allows you to introduce new implementation types. The current  
4720 specification does not mandate the implementation technologies to be supported by an SCA runtime,  
4721 vendors can choose to support the ones that are important for them. A single SCA implementation can be  
4722 used by multiple Components, each with a different configuration.

4723 The Component has a reference to an implementation of which it is an instance, a set of property values,  
4724 and a set of service reference values. Property values define the values of the properties of the  
4725 component as defined by the component's implementation. Reference values define the services that  
4726 resolve the references of the component as defined by its implementation. These values can either be a  
4727 particular service of a particular component, or a reference of the containing composite.

4728

### B.3 Service

4729 **SCA services** are used to declare the externally accessible services of an **implementation**. For a  
4730 composite, a service is typically provided by a service of a component within the composite, or by a  
4731 reference defined by the composite. The latter case allows the republication of a service with a new  
4732 address and/or new bindings. The service can be thought of as a point at which messages from external  
4733 clients enter a composite or implementation.

4734 A service represents an addressable set of operations of an implementation that are designed to be  
4735 exposed for use by other implementations or exposed publicly for use elsewhere (e.g. public Web  
4736 services for use by other organizations). The operations provided by a service are specified by an  
4737 Interface, as are the operations needed by the service client (if there is one). An implementation can  
4738 contain multiple services, when it is possible to address the services of the implementation separately.

4739 A service can be provided **as SCA remote services, as Web services, as stateless session EJB's, as**  
4740 **EIS services, and so on.** Services use **bindings** to describe the way in which they are published. SCA  
4741 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new  
4742 types of services.

4743

#### B.3.1 Remotable Service

4744 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA  
4745 architecture. For example, SCA services of SCA implementations can define implementations of industry-  
4746 standard web services. Remotable services use pass-by-value semantics for parameters and returned  
4747 results.

4748 Interfaces can be identified as remotable through the <interface /> XML, but are typically specified as  
4749 remotable using a component implementation technology specific mechanism, such as Java annotations.  
4750 See the relevant SCA Implementation Specification for more information. As an example, to define a

4751 Remotable Service, a Component implemented in Java would have a Java Interface with the  
4752 @Remotable annotation

### 4753 **B.3.2 Local Service**

4754 Local services are services that are designed to be only used “locally” by other implementations that are  
4755 deployed concurrently in a tightly-coupled architecture within the same operating system process.

4756 Local services can rely on by-reference calling conventions, or can assume a very fine-grained interaction  
4757 style that is incompatible with remote distribution. They can also use technology-specific data-types.

4758 How a Service is identified as local is dependant on the Component implementation technology used.

4759 See the relevant SCA Implementation Specification for more information. As an example, to define a  
4760 Local Service, a Component implemented in Java would define a Java Interface that does not have the  
4761 @Remotable annotation.

4762

### 4763 **B.4 Reference**

4764 **SCA references** represent a dependency that an implementation has on a service that is provided by  
4765 some other implementation, where the service to be used is specified through configuration. In other  
4766 words, a reference is a service that an implementation can call during the execution of its business  
4767 function. References are typed by an interface.

4768 For composites, composite references can be accessed by components within the composite like any  
4769 service provided by a component within the composite. Composite references can be used as the targets  
4770 of wires from component references when configuring Components.

4771 A composite reference can be used to access a service such as: an SCA service provided by another  
4772 SCA composite, a Web service, a stateless session EJB, a database stored procedure or an EIS service,  
4773 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4774 an **extensibility mechanism** that allows the introduction of new binding types to references.

4775

### 4776 **B.5 Implementation**

4777 An implementation is concept that is used to describe a piece of software technology such as a Java  
4778 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4779 service-oriented application. An SCA composite is also an implementation.

4780 Implementations define points of variability including properties that can be set and settable references to  
4781 other services. The points of variability are configured by a component that uses the implementation. The  
4782 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4783 **B.6 Interface**

4784 Interfaces define one or more business functions. These business functions are provided by Services  
4785 and are used by components through References. Services are defined by the Interface they implement.  
4786 SCA currently supports a number of interface type systems, for example:

- 4787 • Java interfaces
- 4788 • WSDL portTypes
- 4789 • C, C++ header files

4790

4791 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4792 interface type systems.

4793 Interfaces can be **bi-directional**. A bi-directional service has service operations which are provided by  
4794 each end of a service communication – this could be the case where a particular service demands a



4795 “callback” interface on the client, which it calls during the process of handing service requests from the  
4796 client.  
4797

## 4798 **B.7 Composite**

4799 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4800 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4801 be used to contribute elements to an **SCA Domain**.

4802 A **composite** has the following characteristics:

- 4803 • It can be used as a component implementation. When used in this way, it defines a boundary for  
4804 Component visibility. Components cannot be directly referenced from outside of the composite in  
4805 which they are declared.
- 4806 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
4807 artifacts to an SCA Domain.

4808

## 4809 **B.8 Composite inclusion**

4810 One composite can be used to provide part of the definition of another composite, through the process of  
4811 inclusion. This is intended to make team development of large composites easier. Included composites  
4812 are merged together into the using composite at deployment time to form a single logical composite.

4813 Composites are included into other composites through `<include.../>` elements in the using composite.  
4814 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
4815 specific location.

4816

## 4817 **B.9 Property**

4818 **Properties** allow for the configuration of an implementation with externally set data values. The data  
4819 value is provided through a Component, possibly sourced from the property of a containing composite.

4820 Each Property is defined by the implementation. Properties can be defined directly through the  
4821 implementation language or through annotations of implementations, where the implementation language  
4822 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
4823 type. For complex data types, XML schema is the preferred technology for defining the data types.

4824

## 4825 **B.10 Domain**

4826 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
4827 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
4828 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
4829 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4830 A Domain specifies the instantiation, configuration and connection of a set of components, provided via  
4831 one or more composite files. A Domain also contains Wires that connect together the Components. A  
4832 Domain does not contain promoted Services or promoted References, since promotion has no meaning  
4833 at the Domain level.

4834

## 4835 **B.11 Wire**

4836 **SCA wires** connect **service references** to **services**.

4837 Valid wire sources are component references. Valid wire targets are component services.

4838 When using included composites, the sources and targets of the wires don't have to be declared in the  
4839 same composite as the composite that contains the wire. The sources and targets can be defined by  
4840 other included composites. Targets can also be external to the SCA Domain.  
4841

4842

## C. Conformance Items

4843

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

4844

Conformance ID	Description
[ASM13001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd, sca-interface-wsdl.xsd, sca-implementation-composite.xsd and sca-binding-sca.xsd schema.
[ASM13002]	An SCA runtime MUST reject a contribution file that does not conform to the sca-contribution.xsd schema.
[ASM13003]	An SCA runtime MUST reject a definitions file that does not conform to the sca-definitions.xsd schema.
[ASM40001]	The extension of a componentType side file name MUST be .componentType.
[ASM40002]	If present, the @constrainingType attribute of a <componentType/> element MUST reference a <constrainingType/> element in the Domain through its QName.
[ASM40003]	The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
[ASM40004]	The @name attribute of a <reference/> child element of a <componentType/> MUST be unique amongst the reference elements of that <componentType/>.
[ASM40005]	The @name attribute of a <property/> child element of a <componentType/> MUST be unique amongst the property elements of that <componentType/>.
[ASM40006]	If @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be ignored by the runtime.
[ASM40007]	The value of the property @type attribute MUST be the QName of an XML schema type.
[ASM40008]	The value of the property @element attribute MUST be the QName of an XSD global element.
[ASM40009]	The SCA runtime MUST ensure that any implementation default property value is replaced by a value for that property explicitly set by a component using that implementation.
[ASM40010]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM40011]	When the componentType has @mustSupply="true" for a property element, a component using the implementation MUST supply a value for the property since the implementation has no default value for the property.
[ASM50001]	The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements

	of that <composite/>
[ASM50002]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50003]	The @name attribute of a service element of a <component/> MUST match the @name attribute of a service element of the componentType of the <implementation/> child element of the component.
[ASM50004]	If a <service/> element has an interface subelement specified, the interface MUST provide a compatible subset of the interface declared on the componentType of the implementation
[ASM50005]	If no binding elements are specified for the service, then the bindings specified for the equivalent service in the componentType of the implementation MUST be used, but if the componentType also has no bindings specified, then <binding.sca/> MUST be used as the binding. If binding elements are specified for the service, then those bindings MUST be used and they override any bindings specified for the equivalent service in the componentType of the implementation.
[ASM50006]	If the callback element is present and contains one or more binding child elements, then those bindings MUST be used for the callback.
[ASM50007]	The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>
[ASM50008]	The @name attribute of a reference element of a <component/> MUST match the @name attribute of a reference element of the componentType of the <implementation/> child element of the component.
[ASM50009]	The value of multiplicity for a component reference MUST only be equal or further restrict any value for the multiplicity of the reference with the same name in the componentType of the implementation, where further restriction means 0..n to 0..1 or 1..n to 1..1.
[ASM50010]	If @wiredByImpl="true" is set for a reference, then the reference MUST NOT be wired statically within a composite, but left unwired.
[ASM50011]	If an interface is declared for a component reference, the interface MUST provide a compatible superset of the interface declared for the equivalent reference in the componentType of the implementation, i.e. provide the same operations or a superset of the operations defined by the implementation for the reference.
[ASM50012]	If no binding elements are specified for the reference, then the bindings specified for the equivalent reference in the componentType of the implementation MUST be used. If binding elements are specified for the reference, then those bindings MUST be used and they override any bindings specified for the equivalent reference in the componentType of the

	implementation.
[ASM50013]	If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.
[ASM50014]	If @autowire="true", the autowire procedure MUST only be used if no target is identified by any of the other ways listed above. It is not an error if @autowire="true" and a target is also defined through some other means, however in this case the autowire procedure MUST NOT be used.
[ASM50015]	If a binding element has a value specified for a target service using its @uri attribute, the binding element MUST NOT identify target services using binding specific attributes or elements.
[ASM50016]	It is possible that a particular binding type MAY require that the address of a target service uses more than a simple URI. In cases where a reference element has a binding subelement of such a type, the @uri attribute of the binding element 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 raise an error no later than when the reference is invoked by the component implementation.
[ASM50025]	Where a component reference is promoted by a composite reference, the promotion MUST be treated from a multiplicity perspective as providing 0 or more target services for the component reference, depending upon the further configuration of the composite reference. These target services are in addition to any target services identified on the component reference itself, subject to the rules relating to multiplicity.
[ASM50026]	If a reference has a value specified for one or more target services in its @target attribute, there MUST NOT be any child <binding/> elements declared for that reference.
[ASM50027]	If the @value attribute of a component property element is declared, the type of the property MUST be an XML Schema simple type and the @value attribute MUST contain a single value of that type.

[ASM50028]	If the value subelement of a component property is specified, the type of the property MUST be an XML Schema simple type or an XML schema complex type.
[ASM50029]	If a component property value is declared using a child element of the <property/> element, the type of the property MUST be an XML Schema global element and the declared child element MUST be an instance of that global element.
[ASM50030]	A <component/> element MUST NOT contain two <property/> subelements with the same value of the @name attribute.
[ASM50031]	The @name attribute of a property element of a <component/> MUST be unique amongst the property elements of that <component/>.
[ASM50032]	If a property is single-valued, the <value/> subelement MUST NOT occur more than once.
[ASM50033]	A property <value/> subelement MUST NOT be used when the @value attribute is used to specify the value for that property.
[ASM50034]	If any <wire/> element with its @replace attribute set to "true" has a particular reference specified in its @source attribute, the value of the @target attribute for that reference MUST be ignored and MUST NOT be used to define target services for that reference.
[ASM50035]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM50036]	The property type specified for the property element of a component MUST be compatible with the type of the property with the same @name declared in the component type of the implementation used by the component. If no type is declared in the component property element, the type of the property declared in the componentType of the implementation MUST be used.
[ASM50037]	The @name attribute of a property element of a <component/> MUST match the @name attribute of a property element of the componentType of the <implementation/> child element of the component.
[ASM60001]	A composite @name attribute value MUST be unique within the namespace of the composite.
[ASM60002]	@local="true" for a composite means that all the components within the composite MUST run in the same operating system process.
[ASM60003]	The name of a composite <service/> element MUST be unique across all the composite services in the composite.
[ASM60004]	A composite <service/> element's @promote attribute MUST identify one of the component services within that composite.
[ASM60005]	If a composite service <b>interface</b> is specified it MUST be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service.

[ASM60006]	The name of a composite <reference/> element MUST be unique across all the composite references in the composite.
[ASM60007]	Each of the URIs declared by a composite reference's @promote attribute MUST identify a component reference within the composite.
[ASM60008]	the interfaces of the component references promoted by a composite reference MUST be the same, or if the composite reference itself declares an interface then all the component reference interfaces MUST be compatible with the composite reference interface. Compatible means that the component reference interface is the same or is a strict subset of the composite reference interface.
[ASM60009]	the intents declared on a composite reference and on the component references which it promotes MUST NOT be mutually exclusive.
[ASM60010]	If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST raise an error.
[ASM60011]	The value specified for the @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 <b>interface</b> specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s), i.e. provide a superset of the operations in the interface defined by the component for the reference.
[ASM60013]	If no interface is declared on a composite reference, the interface from one of its promoted component references is used, which MUST be the same as or a compatible superset of the interface(s) declared by the promoted component reference(s).
[ASM60014]	The @name attribute of a composite property MUST be unique amongst the properties of the same composite.
[ASM60015]	the source interface and the target interface of a wire MUST either both be remotable or else both be local
[ASM60016]	the operations on the target interface of a wire MUST be the same as or be a superset of the operations in the interface specified on the source
[ASM60017]	compatibility between the source interface and the target interface for a wire for the individual operations is defined as compatibility of the signature, that is operation name, input types, and output types MUST be the same.

[ASM60018]	the order of the input and output types for operations in the source interface and the target interface of a wire also MUST be the same.
[ASM60019]	the set of Faults and Exceptions expected by each operation in the source interface MUST be the same or be a superset of those specified by the target interface.
[ASM60020]	If either the source interface of a wire or the target interface of a wire declares a callback interface then both the source interface and the target interface MUST declare a callback interface and the callback interface declared on the target MUST be a compatible superset of the callback interface declared on the source.
[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 SCA runtime MUST search within the composite for target services which are compatible with the reference.
[ASM60023]	the target service interface MUST be a compatible superset of the reference interface when using autowire to wire a reference (as defined in <a href="#">the section on Wires</a> )
[ASM60024]	the intents, and policies applied to the service MUST be compatible with those on the reference when using autowire to wire a reference – so that wiring the reference to the service will not cause an error due to policy mismatch
[ASM60025]	for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the reference to one of the set of valid target services chosen from the set in a runtime-dependent fashion
[ASM60026]	for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all of the set of valid target services
[ASM60027]	for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT raise an error
[ASM60028]	for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid target services an error MUST be raised by the SCA runtime since the reference is intended to be wired
[ASM60030]	The @name attribute of an <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.
[ASM60031]	The SCA runtime MUST raise an error if the composite resulting from the inclusion of one composite into another is invalid.
[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.



[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.
[ASM60035]	All the component references promoted by a single composite reference MUST have the same value for @wiredByImpl.
[ASM60036]	If the @wiredByImpl attribute is not specified on the composite reference, the default value is "true" if all of the promoted component references have a wiredByImpl value of "true", and the default value is "false" if all the promoted component references have a wiredByImpl value of "false". If the @wiredByImpl attribute is specified, its value MUST be "true" if all of the promoted component references have a wiredByImpl value of "true", and its value MUST be "false" if all the promoted component references have a wiredByImpl value of "false".
[ASM60037]	<include/> processing MUST take place before the processing of the @promote attribute of a composite reference is performed.
[ASM60038]	<include/> processing MUST take place before the processing of the @promote attribute of a composite service is performed.
[ASM60039]	<include/> processing MUST take place before the @source and @target attributes of a wire are resolved.
[ASM60040]	A single property element MUST NOT contain both a @type attribute and an @element attribute.
[ASM60041]	If the included composite has the value <i>true</i> for the attribute @local then the including composite MUST have the same value for the @local attribute, else it is an error.
[ASM60042]	The @name attribute of an include element MUST be the QName of a composite in the SCA Domain.
[ASM70001]	The constrainingType specifies the services, references and properties that MUST be provided by the implementation of the component to which the constrainingType is attached.
[ASM70002]	If the configuration of the component or its implementation does not conform to the constrainingType specified on the component element, the SCA runtime MUST raise an error.
[ASM70003]	The @name attribute of the constraining type MUST be unique in the SCA Domain.
[ASM70004]	When an implementation is constrained by a constrainingType its component type MUST contain all the services, references and properties specified in the constrainingType.
[ASM70005]	An implementation MAY contain additional services, additional references with @multiplicity=0..1 or @multiplicity=0..n and additional properties with @mustSupply=false beyond those declared in the constraining type, but MUST NOT contain additional references with @multiplicity=1..1 or @multiplicity=1..n or additional properties 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.
[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 <b>method or operation overloading</b> .
[ASM80003]	If a remotable service is called locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the service or post-invocation modifications to return messages are seen by the caller.
[ASM80004]	If a reference is defined using a bidirectional interface element, the client component implementation using the reference calls the referenced service using the interface. The client MUST provide an implementation of the callback interface.
[ASM80005]	Either both interfaces of a bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and remote services.
[ASM80008]	Any service or reference that uses an interface marked with 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.
[ASM80017]	WSDL interfaces are always remotable and therefore an <interface.wSDL/> element MUST NOT contain remotable="false".
[ASM90001]	For a binding of a <b>reference</b> the @uri attribute defines the target URI of the reference. This MUST be either the componentName/serviceName for a wire to an endpoint within the SCA Domain, or the accessible address of some service endpoint either inside or outside the SCA Domain (where the addressing scheme is defined by the type of the binding).
[ASM90002]	When a service or reference has multiple bindings, only one binding can have the default @name value; all others MUST have a @name value specified that is unique within the service or reference.
[ASM90003]	If a reference has any bindings, they MUST be resolved, which means that each binding MUST include a value for the @uri attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired using other SCA mechanisms.
[ASM90004]	a wire target MAY be specified with a syntax of "componentName/serviceName/bindingName".
[ASM10001]	all of the QNames for the definitions contained in definitions.xml files MUST be unique within the Domain.
[ASM10002]	An SCA runtime MUST make available to the Domain all the artifacts contained within the definitions.xml files in the Domain.
[ASM10003]	An SCA runtime MUST reject a definitions.xml file that does not conform to the sca-definitions.xsd schema.
[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.

[ASM12005]	Where present, artifact-related or packaging-related artifact resolution 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.
[ASM12007]	Implementations of SCA MAY also raise an error if there are conflicting names exported from multiple contributions.
[ASM12008]	An SCA runtime MAY provide the contribution operation functions (install Contribution, update Contribution, add Deployment Composite, update Deployment Composite, remove Contribution).
[ASM12009]	if there is ever a conflict between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in the dependent contribution list.
[ASM12010]	Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.
[ASM12011]	If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the resource indicated when using the mechanism (e.g. the URI is incorrect or invalid, say) the SCA runtime MUST raise an error and MUST NOT attempt to use SCA resolution mechanisms as an alternative.
[ASM12012]	The value of @autowire for the logical Domain composite MUST be autowire="false".
[ASM12013]	For components at the Domain level, with References for which @autowire="true" applies, the behaviour of the SCA runtime for a given Domain MUST take ONE of the 3 following forms: 1) The SCA runtime MAY disallow deployment of any components with autowire References. In this case, the SCA runtime MUST raise an exception at the point where the component is deployed. 2) The SCA runtime MAY evaluate the target(s) for the reference at the time that the component is deployed and not update those targets when later deployment actions occur. 3) The SCA runtime MAY re-evaluate the target(s) for the reference dynamically as later deployment actions occur resulting in updated reference targets which match the new Domain configuration. How the new configuration of the reference takes place is described by the relevant client and implementation specifications.
[ASM12014]	Where <wire/> elements are added, removed or replaced by deployment actions, the components whose references are affected by those deployment actions MAY have their references updated by the SCA runtime dynamically without the need to stop and start those components.
[ASM12015]	Where components are updated by deployment actions (their configuration is changed in some way, which includes changing the wires of component references), the new configuration MUST

	apply to all new instances of those components once the update is complete.
[ASM12016]	An SCA runtime MAY choose to maintain existing instances with the old configuration of components updated by deployment actions, but an SCA runtime MAY choose to stop and discard existing instances of those components.
[ASM12017]	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: <ul style="list-style-type: none"> <li>• either cause future invocation of the target component's services to fail with a ServiceUnavailable fault</li> <li>• or alternatively, if an alternative target component is available that satisfies the autowire process, update the reference of the source component</li> </ul>
[ASM12018]	Where a component that is the target of a wire is updated, future invocations of that reference SHOULD use the updated component.
[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: <ul style="list-style-type: none"> <li>- either update the references for the source component once the new component is running.</li> <li>- or alternatively, defer the updating of the references of the source component until the source component is stopped and restarted.</li> </ul>
[ASM12021]	The SCA runtime MUST raise an error if an artifact cannot be resolved using these mechanisms, if present.
[ASM12022]	There can be multiple import declarations for a given namespace. Where multiple import declarations are made for the same namespace, all the locations specified MUST be searched in lexical order.
[ASM12023]	When a contribution contains a reference to an artifact from a namespace that is declared in an import statement of the contribution, if the SCA artifact resolution mechanism is used to resolve the artifact, the SCA runtime MUST resolve artifacts in the following order: <ol style="list-style-type: none"> <li>1. from the locations identified by the import statement(s) for the namespace. Locations MUST NOT be searched recursively in order to locate artifacts (i.e. only a one-level search is performed).</li> <li>2. from the contents of the contribution itself.</li> </ol>
[ASM12024]	The SCA runtime MUST ignore local definitions of an artifact if the artifact is found through resolving an import statement.
[ASM12025]	The SCA runtime MUST raise an error if an artifact cannot be resolved by using artifact-related or packaging-related artifact resolution mechanisms, if present, by searching locations

	identified by the import statements of the contribution, if present, and by searching the contents of the contribution.
[ASM12026]	An SCA runtime MUST make the <import/> and <export/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-generated.xml files available for the SCA artifact resolution process.
[ASM12027]	An SCA runtime MUST reject files that do not conform to the schema declared in sca-contribution.xsd.
[ASM12028]	An SCA runtime MUST merge the contents of sca-contribution-generated.xml into the contents of sca-contribution.xml, with the entries in sca-contribution.xml taking priority if there are any conflicting declarations.
[ASM12029]	An SCA runtime MAY deploy the composites in <deployable/> elements found in the META-INF/sca-contribution.xml and META-INF/sca-contribution-generated.xml files.
[ASM12030]	For XML definitions, which are identified by QNames, the @namespace attribute of the export element SHOULD be the namespace URI for the exported definitions.
[ASM12031]	When a contribution uses an artifact contained in another contribution through SCA artifact resolution, if that artifact itself has dependencies on other artifacts, the SCA runtime MUST resolve these dependencies in the context of the contribution containing the artifact, not in the context of the original contribution.
[ASM14001]	An SCA runtime SHOULD detect errors at deployment time where those errors can be found through static analysis.
[ASM14002]	The SCA runtime SHOULD prevent deployment of contributions that are in error, and raise the error to the process performing the deployment (e.g. write a message to an interactive console or write a message to a log file).
[ASM14003]	Where errors are only detected at runtime, when the error is detected an error MUST be raised to the component that is attempting the activity concerned with the error.
[ASM14004]	When an error that could have been detected through static analysis is detected and raised at runtime for a component, the component SHOULD NOT be run until the error is fixed.

4846

## D. Acknowledgements

4847 The following individuals have participated in the creation of this specification and are gratefully  
4848 acknowledged:

4849 **Participants:**

4850

Participant Name	Affiliation
Bryan Aupperle	IBM
Ron Barack	SAP AG*
Michael Beisiegel	IBM
Megan Beynon	IBM
Vladislav Bezrukov	SAP AG*
Henning Blohm	SAP AG*
Fraser Bohm	IBM
David Booz	IBM
Fred Carter	AmberPoint
Martin Chapman	Oracle Corporation
Graham Charters	IBM
Shih-Chang Chen	Oracle Corporation
Chris Cheng	Primeton Technologies, Inc.
Vamsavardhana Reddy Chillakuru	IBM
Mark Combella	Avaya, Inc.
Jean-Sebastien Delfino	IBM
David DiFranco	Oracle Corporation
Mike Edwards	IBM
Jeff Estefan	Jet Propulsion Laboratory:*
Raymond Feng	IBM
Billy Feng	Primeton Technologies, Inc.
Paul Fremantle	WSO2*
Robert Freund	Hitachi, Ltd.
Peter Furniss	Iris Financial Solutions Ltd.
Genadi Genov	SAP AG*
Mark Hapner	Sun Microsystems
Zahir HEZOUAT	IBM
Simon Holdsworth	IBM
Sabin Ielceanu	TIBCO Software Inc.
Bo Ji	Primeton Technologies, Inc.
Uday Joshi	Oracle Corporation
Mike Kaiser	IBM
Khanderao Kand	Oracle Corporation
Anish Karmarkar	Oracle Corporation
Nickolaos Kavantzias	Oracle Corporation
Rainer Kerth	SAP AG*
Dieter Koenig	IBM
Meeraj Kunnumpurath	Individual
Jean Baptiste Laviro	Axway Software*

Simon Laws	IBM
Rich Levinson	Oracle Corporation
Mark Little	Red Hat
Ashok Malhotra	Oracle Corporation
Jim Marino	Individual
Carl Mattocks	CheckMi*
Jeff Mischkinsky	Oracle Corporation
Ian Mitchell	IBM
Dale Moberg	Axway Software*
Simon Moser	IBM
Simon Nash	Individual
Peter Niblett	IBM
Duane Nickull	Adobe Systems
Eisaku Nishiyama	Hitachi, Ltd.
Sanjay Patil	SAP AG*
Plamen Pavlov	SAP AG*
Peter Peshev	SAP AG*
Gilbert Pilz	Oracle Corporation
Nilesh Rade	Deloitte Consulting LLP
Martin Raepple	SAP AG*
Luciano Resende	IBM
Michael Rowley	Active Endpoints, Inc.
Vicki Shipkowitz	SAP AG*
Ivana Trickovic	SAP AG*
Clemens Utschig - Utschig	Oracle Corporation
Scott Vorthmann	TIBCO Software Inc.
Feng Wang	Primeton Technologies, Inc.
Tim Watson	Oracle Corporation
Eric Wells	Hitachi, Ltd.
Robin Yang	Primeton Technologies, Inc.
Prasad Yendluri	Software AG, Inc.*

4851



---

## E. Non-Normative Text

4853

## F. Revision History

4854

[optional; should not be included in OASIS Standards]

4855

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
14 CD02-Rev2	2009-01-30	Mike Edwards	Issue 94 applied (removal of conversations)
15 CD02-Rev3	2009-01-30	Mike Edwards	Issue 98 - Section 5.3 Minor editorial cleanup (various locations) Removal of <operation/> element as decided at Jan 2009 F2F - various sections Issue 95 - Section 6.2 Issue 2 - Section 2.1 Issue 37 - Sections 2.1, 6, 12.6.1, B10 Issue 48 - Sections 5.3, A2 Issue 90 - Sections 6.1, 6.2, 6.4 Issue 64 - Sections 7, A2 Issue 100 - Section 6.2 Issue 103 - Sections 10, 12.2.2, A.13 Issue 104 - Sections 4.1.3, 5.4, 6.3 Section 3 (Quick Tour By Sample) removed by decision of Jan 2009 Assembly F2F meeting
16 CD02-Rev4	2009-02-06	Mike Edwards	All changes accepted Major Editorial work to clean out all RFC2119 wording and to ensure that no normative statements have been missed.
16 CD02-Rev6	2009-02-24	Mike Edwards	Issue 107 - sections 4, 5, 11, Appendix C Editorial updates resulting from Review Issue 34 - new section 12 inserted, + minor editorial changes in sections 4, 11 Issue 110 - Section 8.0 Issue 111 - Section 4.4, Appendix C Issue 112 - Section 4.5 Issue 113 - Section 3.3 Issue 108 - Section 13, Appendix C Minor editorial changes to the example in section 3.3
17 CD02-Rev7	2009-03-02	Mike Edwards	Editorial changes resulting from Vamsi's review of CD02 Rev6 Issue 109 - Section 8, Appendix A.2, Appendix B.3.1, Appendix C Added back @requires and @policySets to <interface/> as editorial correction since they were lost by accident in earlier revision Issue 101 - Section 13 Issue 120 - Section
18 CD02-Rev8	2009-03-05	Mike Edwards	XSDs corrected and given new namespace. Namespace updated throughout document.
19 CD03	2009-03-05	Mike Edwards	All Changes Accepted
20 CD03	2009-03-17	Anish Karmarkar	Changed CD03 per TC's CD03/PR01 resolution. Fixed the footer, front page.

4856