



# Service Component Architecture Assembly Model Specification Version 1.1

**Committee Draft 01 Revision 4 + Issue 89**

**11th December 2008**

**Specification URIs:**

**This Version:**

<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev4.html>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev4.doc>  
<http://docs.oasis-open.org/opencsa/sca-assembly/sca-assembly-1.1-spec-cd01-rev4.pdf>  
(Authoritative)

**Previous Version:**

**Latest Version:**

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

**Latest Approved Version:**

**Technical Committee:**

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

**Chair(s):**

Martin Chapman, Oracle  
Mike Edwards, IBM

**Editor(s):**

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

**Related work:**

This specification replaces or supercedes:

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

This specification is related to:

- Service Component Architecture Policy Framework Specification Version 1.1

**Declared XML Namespace(s):**

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

**Abstract:**

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

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

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

This document describes the SCA Assembly Model, which covers

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

**Status:**

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

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

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

---

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

---

## Notices

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

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

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

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

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

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

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

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

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

---

## Table of Contents

Committee Draft 01 Revision 4 + Issue 89 .....	1
The non-normative errata page for this specification is located at <a href="http://www.oasis-open.org/committees/sca-assembly/">http://www.oasis-open.org/committees/sca-assembly/</a> .....	3
Notices .....	4
Table of Contents .....	5
1 Introduction .....	8
1.1 Terminology .....	8
1.2 Normative References .....	8
1.3 Naming Conventions .....	9
2 Overview .....	10
2.1 Diagram used to Represent SCA Artifacts .....	11
3 Quick Tour by Sample .....	13
4 Implementation and ComponentType .....	14
4.1 Component Type .....	14
4.1.1 Service .....	15
4.1.2 Reference .....	16
4.1.3 Property .....	18
4.1.4 Implementation .....	19
4.2 Example ComponentType .....	20
4.3 Example Implementation .....	20
5 Component .....	23
5.1 Implementation .....	24
5.2 Service .....	25
5.3 Reference .....	27
5.3.1 Specifying the Target Service(s) for a Reference .....	29
5.4 Property .....	30
5.5 Example Component .....	33
6 Composite .....	37
6.1 Service .....	39
6.1.1 Service Examples .....	40
6.2 Reference .....	42
6.2.1 Example Reference .....	44
6.3 Property .....	46
6.3.1 Property Examples .....	47
6.4 Wire .....	50
6.4.1 Wire Examples .....	52
6.4.2 Autowire .....	54
6.4.3 Autowire Examples .....	55
6.5 Using Composites as Component Implementations .....	58
6.5.1 Example of Composite used as a Component Implementation .....	60
6.6 Using Composites through Inclusion .....	61
6.6.1 Included Composite Examples .....	62
6.7 Composites which Include Component Implementations of Multiple Types .....	65

7	ConstrainingType .....	66
7.1	Example constrainingType .....	67
8	Interface.....	69
8.1	Local and Remotable Interfaces .....	70
8.2	Bidirectional Interfaces .....	71
8.3	Conversational Interfaces .....	72
8.4	SCA-Specific Aspects for WSDL Interfaces .....	74
8.5	WSDL Interface Type .....	75
8.5.1	Example of interface.wsdl .....	76
9	Binding.....	77
9.1	Messages containing Data not defined in the Service Interface .....	79
9.2	Form of the URI of a Deployed Binding .....	79
9.2.1	Constructing Hierarchical URIs .....	79
9.2.2	Non-hierarchical URIs .....	80
9.2.3	Determining the URI scheme of a deployed binding.....	80
9.3	SCA Binding .....	81
9.3.1	Example SCA Binding .....	81
9.4	Web Service Binding .....	82
9.5	JMS Binding.....	82
10	SCA Definitions .....	83
11	Extension Model .....	84
11.1	Defining an Interface Type.....	84
11.2	Defining an Implementation Type.....	86
11.3	Defining a Binding Type.....	87
12	Packaging and Deployment .....	90
12.1	Domains.....	90
12.2	Contributions.....	90
12.2.1	SCA Artifact Resolution.....	91
12.2.2	SCA Contribution Metadata Document .....	92
12.2.3	Contribution Packaging using ZIP .....	94
12.3	Installed Contribution .....	94
12.3.1	Installed Artifact URIs .....	94
12.4	Operations for Contributions.....	95
12.4.1	install Contribution & update Contribution .....	95
12.4.2	add Deployment Composite & update Deployment Composite.....	95
12.4.3	remove Contribution .....	95
12.5	Use of Existing (non-SCA) Mechanisms for Resolving Artifacts .....	95
12.6	Domain-Level Composite .....	96
12.6.1	add To Domain-Level Composite.....	96
12.6.2	remove From Domain-Level Composite .....	97
12.6.3	get Domain-Level Composite .....	97
12.6.4	get QName Definition .....	97
13	Conformance .....	98
A.	Pseudo Schema .....	99
A.1	ComponentType.....	99

A.2 Composite .....	100
B. XML Schemas .....	102
B.1 sca.xsd .....	102
B.2 sca-core.xsd .....	102
B.3 sca-binding-sca.xsd .....	111
B.4 sca-interface-java.xsd .....	111
B.5 sca-interface-wsdl.xsd .....	112
B.6 sca-implementation-java.xsd .....	113
B.7 sca-implementation-composite.xsd .....	114
B.8 sca-definitions.xsd .....	114
B.9 sca-binding-webservice.xsd .....	115
B.10 sca-binding-jms.xsd .....	115
B.11 sca-policy.xsd .....	115
B.12 sca-contribution.xsd .....	115
C. SCA Concepts .....	117
C.1 Binding .....	117
C.2 Component .....	117
C.3 Service .....	117
C.3.1 Remotable Service .....	117
C.3.2 Local Service .....	118
C.4 Reference .....	118
C.5 Implementation .....	118
C.6 Interface .....	118
C.7 Composite .....	119
C.8 Composite inclusion .....	119
C.9 Property .....	119
C.10 Domain .....	119
C.11 Wire .....	119
D. Conformance Items .....	120
E. Acknowledgements .....	128
F. Non-Normative Text .....	129
G. 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 should be used for any non-XML serializations.

## 1.1 Terminology

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

## 1.2 Normative References

- [RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, <http://www.ietf.org/rfc/rfc2119.txt>, IETF RFC 2119, March 1997.
- [1] SCA Java Component Implementation Specification  
SCA Java Common Annotations and APIs Specification  
[http://www.osoa.org/download/attachments/35/SCA\\_JavaComponentImplementation\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JavaComponentImplementation_V100.pdf)  
[http://www.osoa.org/download/attachments/35/SCA\\_JavaAnnotationsAndAPIs\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JavaAnnotationsAndAPIs_V100.pdf)
- [2] SDO Specification  
<http://www.osoa.org/download/attachments/36/Java-SDO-Spec-v2.1.0-FINAL.pdf>
- [3] SCA Example Code document  
[http://www.osoa.org/download/attachments/28/SCA\\_BuildingYourFirstApplication\\_V09.pdf](http://www.osoa.org/download/attachments/28/SCA_BuildingYourFirstApplication_V09.pdf)
- [4] JAX-WS Specification  
<http://jcp.org/en/jsr/detail?id=101>
- [5] WS-I Basic Profile  
<http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile>
- [6] WS-I Basic Security Profile  
<http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity>



[7] Business Process Execution Language (BPEL)  
[http://www.oasis-open.org/committees/documents.php?wg\\_abbrev=wsbpel](http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel)

[8] WSDL Specification  
WSDL 1.1: <http://www.w3.org/TR/wsdl>  
WSDL 2.0: <http://www.w3.org/TR/wsdl20/>

[9] SCA Web Services Binding Specification  
[http://www.osoa.org/download/attachments/35/SCA\\_WebServiceBindings\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_WebServiceBindings_V100.pdf)

[10] SCA Policy Framework Specification  
[http://www.osoa.org/download/attachments/35/SCA\\_Policy\\_Framework\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_Policy_Framework_V100.pdf)

[11] SCA JMS Binding Specification  
[http://www.osoa.org/download/attachments/35/SCA\\_JMSBinding\\_V100.pdf](http://www.osoa.org/download/attachments/35/SCA_JMSBinding_V100.pdf)

[12] ZIP Format Definition  
<http://www.pkware.com/documents/casestudies/APPNOTE.TXT>

[13] Infoset Specification  
<http://www.w3.org/TR/xml-infoset/>  
[\[WSDL11 Identifiers\] WSDL 1.1 Element Identifiers](http://www.w3.org/TR/wsdl11elementidentifiers/)  
<http://www.w3.org/TR/wsdl11elementidentifiers/>

Formatted: English (U.S.)

## 1.3 Naming Conventions

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

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

---

## 2 Overview

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.

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

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

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

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

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

## 2.1 Diagram used to Represent SCA Artifacts

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

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

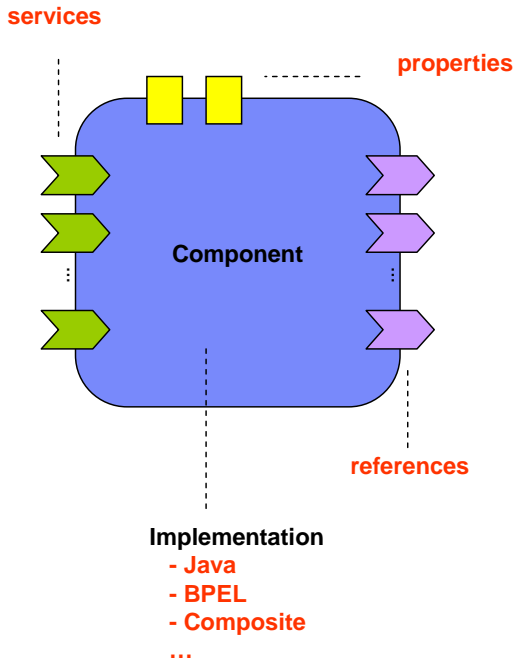


Figure 1: SCA Component Diagram

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

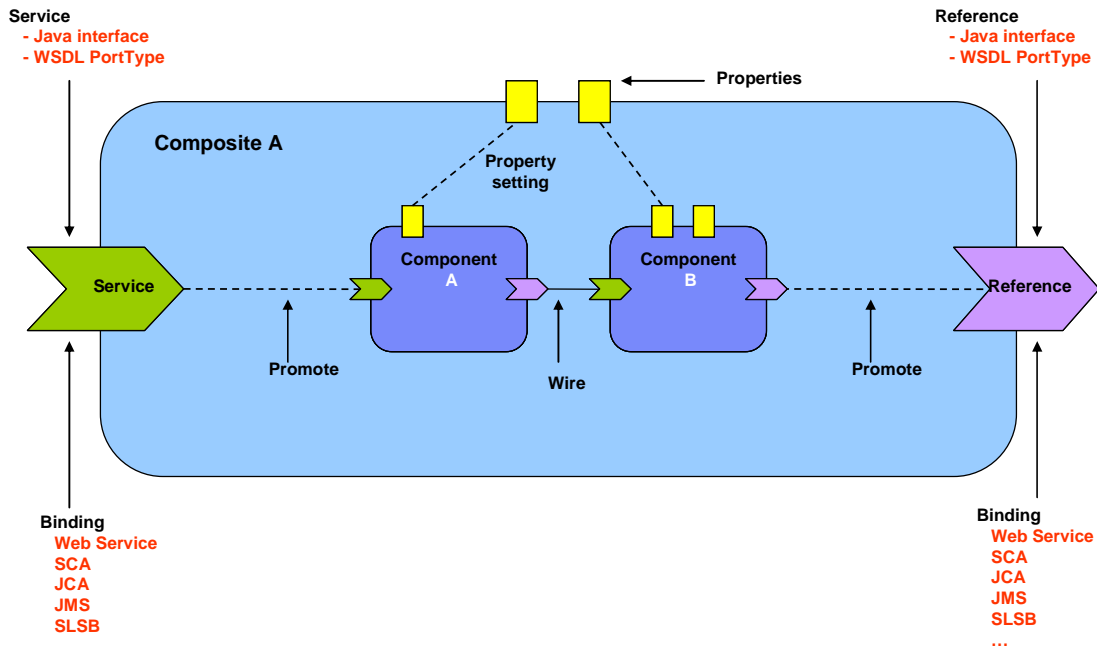


Figure 2: SCA Composite Diagram

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

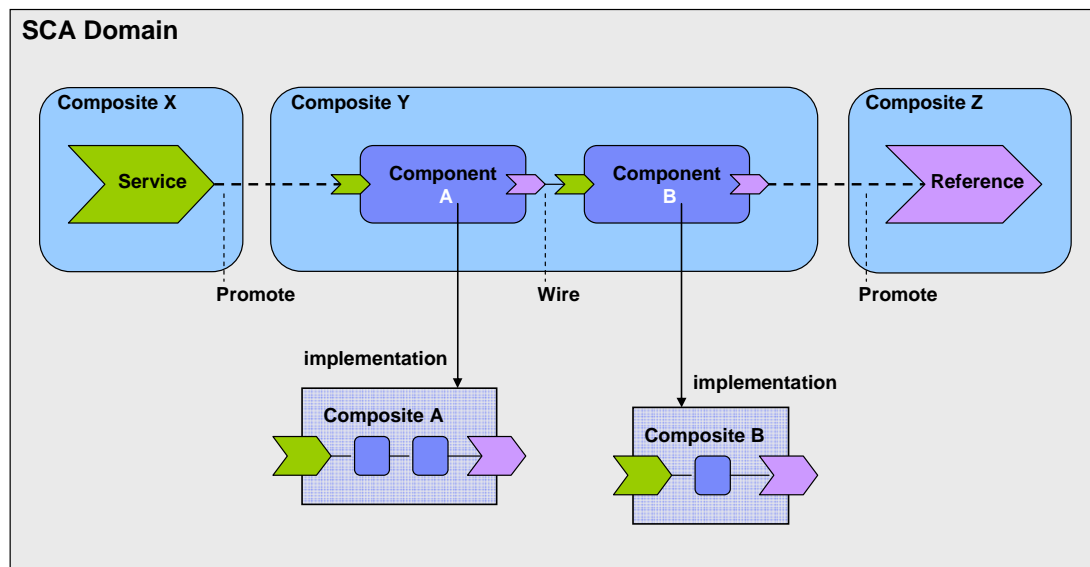


Figure 3: SCA Domain Diagram

149 **3 Quick Tour by Sample**

150 To be completed.

151

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

153

154

---

## 4 Implementation and ComponentType

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

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

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

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

So, for example:

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

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

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

### 4.1 Component Type

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

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

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

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

The following snippet shows the `componentType` schema.

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  constrainingType="QName"? >

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

</componentType>
```

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

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

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

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

#### 4.1.1 Service

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type service schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
>

  <service name="xs:NCName"
    requires="list of xs:QName"? policySets="list of xs:QName"?>*
    <interface ... />
    <binding ... />*
    <callback?>
```

```

251         <binding ... />+
252     </callback>
253 </service>
254
255     <reference ... />*
256     <property ... />*
257     <implementation ... />?
258
259 </componentType>
260

```

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

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

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

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

#### 4.1.2 Reference

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

```

291 <?xml version="1.0" encoding="ASCII"?>
292 <!-- Component type reference schema snippet -->
293 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
294 >
295
296     <service ... />*
297
298     <reference name="xs:NCName"
299         target="list of xs:anyURI"? autowire="xs:boolean"?
300         multiplicity="0..1 or 1..1 or 0..n or 1..n"?
301         wiredByImpl="xs:boolean"?

```



```

302         requires="list of xs:QName"? policySets="list of xs:QName"?>*
303     <interface ... />
304     <binding ... />*
305     <callback?
306         <binding ... />+
307     </callback>
308 </reference>
309
310 <property ... />*
311 <implementation ... />?
312
313 </componentType>
314

```

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

- **name : NCName (1..1)** - the name of the reference. The @name attribute of a <reference/> child element of a <componentType/> MUST be unique amongst the reference elements of that <componentType/>. [ASM40004]
- **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to target services. The multiplicity can have the following values
  - 0..1 – zero or one wire can have the reference as a source
  - 1..1 – one wire can have the reference as a source
  - 0..n - zero or more wires can have the reference as a source
  - 1..n – one or more wires can have the reference as a source

If @multiplicity is not specified, the default value is "1..1".
- **target : anyURI (0..n)** - a list of one or more of target service URI's, depending on multiplicity setting. Each value wires the reference to a component service that resolves the reference. For more details on wiring see [the section on Wires](#).
- **autowire : boolean (0..1)** - whether the reference should be autowired, as described in [the Autowire section](#). Default is false.
- **wiredByImpl : boolean (0..1)** - a boolean value, "false" by default. If set to "false", the reference is wired to the target(s) configured on the reference. If set to "true" it indicates that the target of the reference is set at runtime by the implementation code (eg by the code obtaining an endpoint reference by some means and setting this as the target of the reference through the use of programming interfaces defined by the relevant Client and Implementation specification). If @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be ignored by the runtime. [ASM40006] It is recommended that any references with @wiredByImpl = "true" are left unwired.
- **requires : QName (0..n)** - a list of policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
- **policySets : QName (0..n)** - a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.

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

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

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

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

### 4.1.3 Property

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type property schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
>

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

  <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
    many="xs:boolean"? mustSupply="xs:boolean"?
    requires="list of xs:QName"?
    policySets="list of xs:QName"?>*
    default-property-value?
  </property>

  <implementation ... />?
</componentType>
```

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

- **name : NCName (1..1)** - the name of the property. The @name attribute of a <property/> child element of a <componentType/> MUST be unique amongst the property elements of that <componentType/>. [\[ASM40005\]](#)
- one of **(1..1)**:
  - **type : QName** - the type of the property defined as the qualified name of an XML schema type. The value of the property @type attribute MUST be the QName of an XML schema type. [\[ASM40007\]](#)
  - **element : QName** - the type of the property defined as the qualified name of an XML schema global element – the type is the type of the global element. The value of the property @element attribute MUST be the QName of an XSD global element. [\[ASM40008\]](#)

- **many : boolean (0..1)** - (optional) whether the property is single-valued (false) or multi-valued (true). In the case of a multi-valued property, it is presented to the implementation as a collection of property values.
- **mustSupply : boolean (0..1)** - whether the property value must be supplied by the component that uses the implementation – when mustSupply="true" the component must supply a value since the implementation has no default value for the property. A default-property-value should only be supplied when mustSupply="false" (the default setting for the mustSupply attribute), since the implication of a default value is that it is used only when a value is not supplied by the using component.
- **file : anyURI (0..1)** - a dereferencable URI to a file containing a value for the `property`.

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

The `componentType` property element can contain an SCA default value for the property declared by the implementation. However, the implementation can have a property which has an implementation defined default value, where the default value is not represented in the `componentType`. An example of such a default value is where the default value is computed at runtime by some code contained in the implementation. If a using component needs to control the value of a property used by an implementation, the component sets the value explicitly. 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. [ASM40009]

Comment [mbgl3]: Issue 38

## 4.1.4 Implementation

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type implementation schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ...
>

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

  <implementation requires="list of xs:QName"?
    policySets="list of xs:QName"? />?

</componentType>
```

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

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

## 4.2 Example ComponentType

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

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

  <service name="MyValueService">
    <interface.java interface="services.myvalue.MyValueService"/>
  </service>

  <reference name="customerService">
    <interface.java interface="services.customer.CustomerService"/>
  </reference>
  <reference name="stockQuoteService">
    <interface.java
      interface="services.stockquote.StockQuoteService"/>
  </reference>

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

</componentType>
```

## 4.3 Example Implementation

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

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

```
package services.account;

@Remotable
public interface AccountService {

    AccountReport getAccountReport(String customerID);
}
```

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

```
package services.account;

import java.util.List;

import commonj.sdo.DataFactory;
```

```

502     import org.osoa.sca.annotations.Property;
503     import org.osoa.sca.annotations.Reference;
504
505     import services.accountdata.AccountDataService;
506     import services.accountdata.CheckingAccount;
507     import services.accountdata.SavingsAccount;
508     import services.accountdata.StockAccount;
509     import services.stockquote.StockQuoteService;
510
511     public class AccountServiceImpl implements AccountService {
512
513         @Property
514         private String currency = "USD";
515
516         @Reference
517         private AccountDataService accountDataService;
518         @Reference
519         private StockQuoteService stockQuoteService;
520
521         public AccountReport getAccountReport(String customerID) {
522
523             DataFactory dataFactory = DataFactory.INSTANCE;
524             AccountReport accountReport = (AccountReport)dataFactory.create(AccountReport.class);
525             List accountSummaries = accountReport.getAccountSummaries();
526
527             CheckingAccount checkingAccount = accountDataService.getCheckingAccount(customerID);
528             AccountSummary checkingAccountSummary =
529 (AccountSummary)dataFactory.create(AccountSummary.class);
530             checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
531             checkingAccountSummary.setAccountType("checking");
532             checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance()));
533             accountSummaries.add(checkingAccountSummary);
534
535             SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
536             AccountSummary savingsAccountSummary =
537 (AccountSummary)dataFactory.create(AccountSummary.class);
538             savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
539             savingsAccountSummary.setAccountType("savings");
540             savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()));
541             accountSummaries.add(savingsAccountSummary);
542
543             StockAccount stockAccount = accountDataService.getStockAccount(customerID);
544             AccountSummary stockAccountSummary =
545 (AccountSummary)dataFactory.create(AccountSummary.class);
546             stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
547             stockAccountSummary.setAccountType("stock");
548             float balance=
549 (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
550             stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
551             accountSummaries.add(stockAccountSummary);
552
553             return accountReport;

```

```

554     }
555
556     private float fromUSDollarToCurrency(float value){
557
558         if (currency.equals("USD")) return value; else
559         if (currency.equals("EURO")) return value * 0.8f; else
560         return 0.0f;
561     }
562 }

```

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

```

567 <?xml version="1.0" encoding="ASCII"?>
568 <componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
569               xmlns:xsd="http://www.w3.org/2001/XMLSchema">
570
571     <service name="AccountService">
572         <interface.java interface="services.account.AccountService"/>
573     </service>
574     <reference name="accountDataService">
575         <interface.java
576 interface="services.accountdata.AccountDataService"/>
577     </reference>
578     <reference name="stockQuoteService">
579         <interface.java
580 interface="services.stockquote.StockQuoteService"/>
581     </reference>
582
583     <property name="currency" type="xsd:string">USD</property>
584
585 </componentType>

```

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

## 5 Component

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

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

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

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

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

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

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

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

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

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

## 5.1 Implementation

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component Implementation schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
  ...
  <component ... >*
    <implementation ... />?
    <service ... />*
    <reference ... />*
    <property ... />*
  </component>
  ...
</composite>
```

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

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

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

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

<implementation.bpel process="ans:MoneyTransferProcess"/>

<implementation.composite name="bns:MyValueComposite"/>
```

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



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

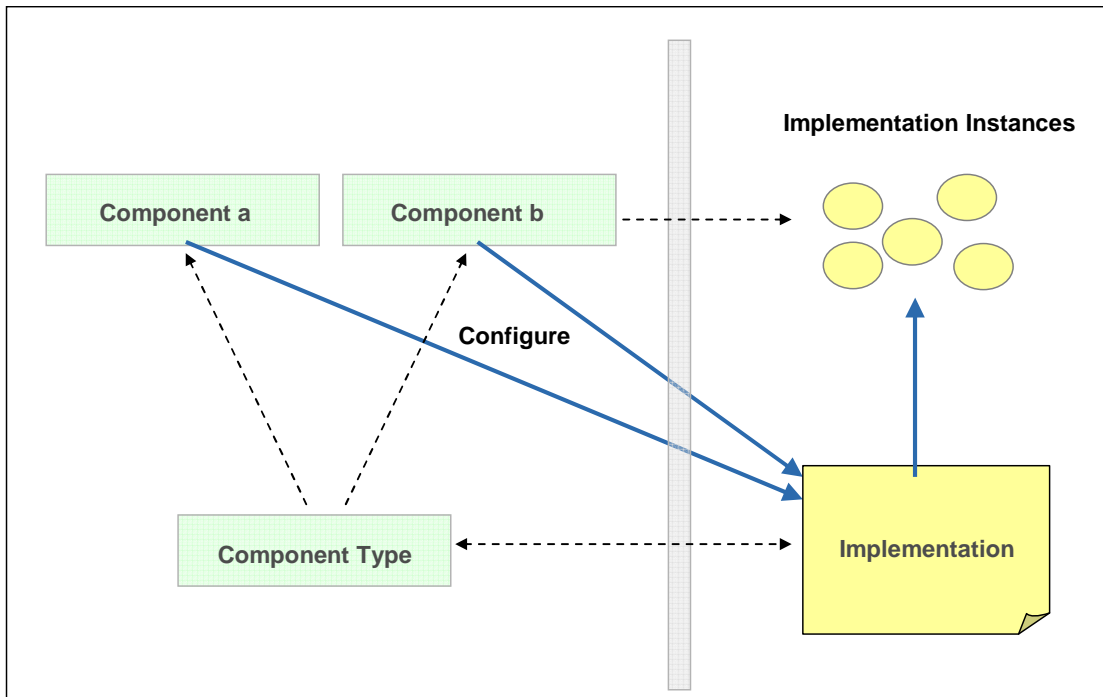


Figure 4: Relationship of Component and Implementation

## 5.2 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:

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

```

702         <callback>?
703             <binding ... />+
704         </callback>
705     </service>
706     <reference ... />*
707     <property ... />*
708 </component>
709 ...
710 </composite>
711

```

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

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

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

- **interface : Interface (0..1)** - A service has **zero or one interface**, which describes the operations provided by the service. The interface is described by an **interface element** which is a child element of the service element. If no interface is specified, then the interface specified for the service in the componentType of the implementation is in effect. 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 [ASM50004] For details on the interface element see [the Interface section](#).
- **binding : Binding (0..n)** - A service element has **zero or more binding elements** as children. 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. [ASM50005] Details of the binding element are described in [the Bindings section](#). The binding, combined with any PolicySets in effect for the binding, needs to satisfy the set of policy intents for the service, as described in [the Policy Framework specification \[10\]](#).
- **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is present and contains one or more binding child elements, then those bindings MUST be used for the callback. [ASM50006] If the callback element is not present, the behaviour is runtime implementation dependent.

## 5.3 Reference

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component Reference schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
  ...
  <component ... >*
    <implementation ... />?
    <service ... />*
    <reference name="xs:NCName"
      target="list of xs:anyURI"? autowire="xs:boolean"?
      multiplicity="0..1 or 1..1 or 0..n or 1..n"?
      wiredByImpl="xs:boolean"? requires="list of xs:QName"?
      policySets="list of xs:QName"?>*
    <interface ... />?
    <binding uri="xs:anyURI"? requires="list of xs:QName"?
      policySets="list of xs:QName"?/>*
    <callback?
      <binding ... />+
    </callback>
  </reference>
  <property ... />*
</component>
  ...
</composite>
```

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

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

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

- **multiplicity : 0..1|1..1|0..n|1..n (0..1)** - defines the number of wires that can connect the reference to target services. Overrides the multiplicity specified for this reference in the componentType of the implementation. The multiplicity can have the following values

- 0..1 – zero or one wire can have the reference as a source
- 1..1 – one wire can have the reference as a source
- 0..n - zero or more wires can have the reference as a source
- 1..n – one or more wires can have the reference as a source

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. [ASM50009]

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used. [ASM50013]
- 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. [ASM50014]
- 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. [ASM50026]
- 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. [ASM50015]
- It is possible that a particular binding type MAY require that the address of a target service uses more than a simple URI. In such cases, the @uri attribute MUST NOT be used to identify the target service - instead, binding specific attributes and/or child elements must be used. [ASM50016]

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

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

- A reference with multiplicity 0..1 or 0..n MAY have no target service defined. [ASM50018]
- A reference with multiplicity 0..1 or 1..1 MUST NOT have more than one target service defined. [ASM50019]
- A reference with multiplicity 1..1 or 1..n MUST have at least one target service defined. [ASM50020]
- A reference with multiplicity 0..n or 1..n MAY have one or more target services defined. [ASM50021]

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

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

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

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. [ASM50025]

## 5.4 Property

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

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

- As a value, supplied in the **value** attribute of the property element.  
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. [ASM50027]

For example,

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

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

For example,

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

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

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

```
<property name="currency">  
  <value>EURO</value>  
  <value>USDollar</value>  
</property>
```

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

```
<property name="complexFoo">
  <value attr="bar">
    <foo:a>TheValue</foo:a>
    <foo:b>InterestingURI</foo:b>
  </value>
</property>
```

- property defined using a XML Schema complex type and which contains multiple values

```
<property name="complexBar">
  <value anotherAttr="foo">
    <bar:a>AValue</bar:a>
    <bar:b>InterestingURI</bar:b>
  </value>
  <value attr="zing">
    <bar:a>BValue</bar:a>
    <bar:b>BoringURI</bar:b>
  </value>
</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. [\[ASM50029\]](#)

For example,

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

```
<property name="foo">
  <foo:SomeGED ...></foo:SomeGED>
</property>
```

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

```
<property name="bar">
  <bar:SomeOtherGED ...></bar:SomeOtherGED>
  <bar:SomeOtherGED ...></bar:SomeOtherGED>
</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.

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.

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.

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

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.

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]

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

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component Property schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
  ...
  <component ... >*
    <implementation ... />?
    <service ... />*
    <reference ... />*
    <property name="xs:NCName"
      (type="xs:QName" | element="xs:QName")?
      mustSupply="xs:boolean"? many="xs:boolean"?
      source="xs:string"? file="xs:anyURI"?
      value="xs:string"?>*
      [<value>+ | xs:any+ ]?
    </property>
  </component>
</composite>
```

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

- **name : NCName (1..1)** – the name of the property. The name attribute of a component property MUST match the name of a property element in the component type of the component implementation. [ASM50031]



- 1035
- zero or one of **(0..1)**:
    - **type : QName** – the type of the property defined as the qualified name of an XML schema type
    - **element : QName** – the type of the property defined as the qualified name of an XML schema global element – the type is the type of the global element
  - **source : string (0..1)** – an XPath expression pointing to a property of the containing composite from which the value of this component property is obtained.
  - **file : anyURI (0..1)** – a dereferencable URI to a file containing a value for the property
  - **many : boolean (0..1)** – (optional) whether the property is single-valued (false) or multi-valued (true). Overrides the many specified for this property on the implementation. The value can only be equal or further restrict, i.e. if the implementation specifies many true, then the component can say false. In the case of a multi-valued property, it is presented to the implementation as a Collection of property [Values](#).
  - **value : string (0..1)** – the value of the property if the property is defined using a simple type.

1040 The **component property** element has the following **child element**:

1041 **value :any (0..n)** - A property has **zero or more**, value elements that specify the value(s) of a  
1042 property that is defined using a XML Schema type. If a property is single-valued, the <value/>  
1043 subelement MUST NOT occur more than once. [\[ASM50032\]](#) A property <value/> subelement MUST  
1044 NOT be used when the @value attribute is used to specify the value for that property. [\[ASM50033\]](#)

## 1055 5.5 Example Component

1056

1057 The following figure shows the **component symbol** that is used to represent a component in an  
1058 assembly diagram.

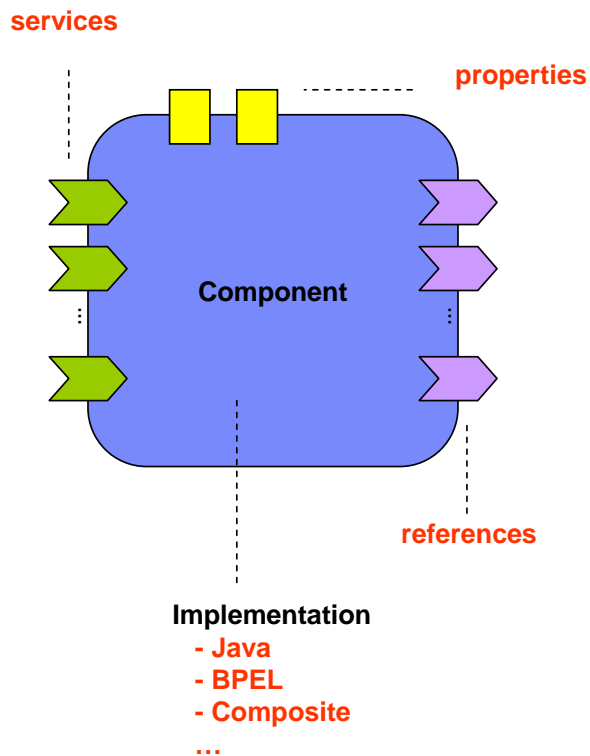


Figure 5: Component symbol

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

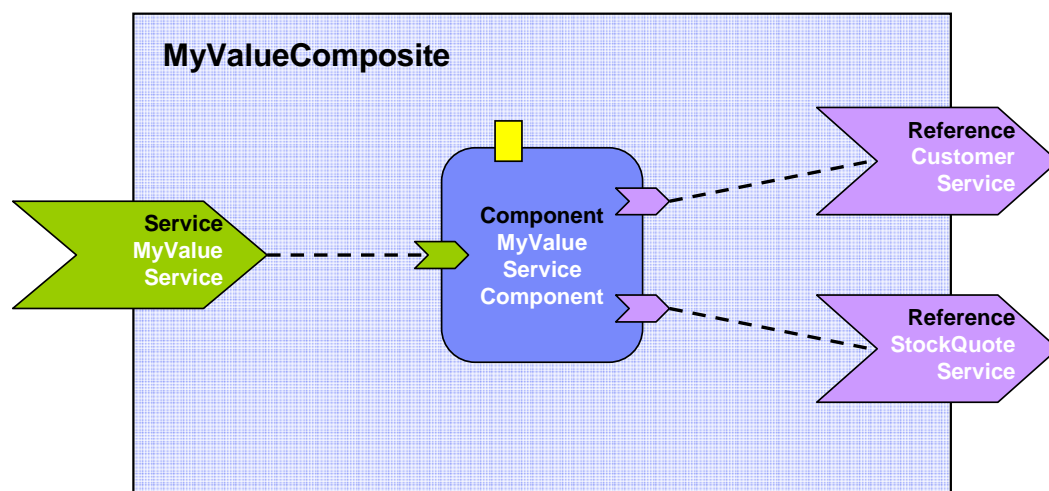


Figure 6: Assembly diagram for MyValueComposite

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_1 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
            targetNamespace="http://foo.com"
            name="MyValueComposite" >

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

    <component name="MyValueServiceComponent">
        <implementation.java
class="services.myvalue.MyValueServiceImpl"/>
        <property name="currency">EURO</property>
        <reference name="customerService"/>
        <reference name="stockQuoteService"/>
    </component>

    <reference name="CustomerService"
        promote="MyValueServiceComponent/customerService"/>

    <reference name="StockQuoteService"
        promote="MyValueServiceComponent/stockQuoteService"/>

</composite>
```

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.

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_2 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
            targetNamespace="http://foo.com"
            name="MyValueComposite" >

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

```

1112     <component name="MyValueServiceComponent">
1113         <implementation.java
1114 class="services.myvalue.MyValueServiceImpl"/>
1115         <property name="currency">EURO</property>
1116         <property name="currency">Yen</property>
1117         <property name="currency">USDollar</property>
1118         <reference name="customerService"
1119             target="InternalCustomer/customerService"/>
1120         <reference name="StockQuoteService"/>
1121     </component>
1122
1123     ...
1124
1125     <reference name="CustomerService"
1126         promote="MyValueServiceComponent/customerService"/>
1127
1128     <reference name="StockQuoteService"
1129         promote="MyValueServiceComponent/StockQuoteService"/>
1130
1131 </composite>

```

1132 ....this assumes that the composite has another component called InternalCustomer (not shown)  
1133 which has a service to which the customerService reference of the MyValueServiceComponent is  
1134 wired as well as being promoted externally through the composite reference CustomerService.  
1135

---

## 6 Composite

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

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

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

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

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

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

    <include ... />*

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

    <component ... />*

    <wire ... />*

</composite>
```

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

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

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

- **service : CompositeService (0..n)** – see composite service section.
- **reference : CompositeReference (0..n)** – see composite reference section.
- **property : CompositeProperty (0..n)** – see composite property section.
- **component : Component (0..n)** – see component section.
- **wire : Wire (0..n)** – see composite wire section.
- **include : Include (0..n)** – see composite include section

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

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

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

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

## 6.1 Service

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

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Composite Service schema snippet -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >
  ...
  <service name="xs:NCName" promote="xs:anyURI"
    requires="list of xs:QName"? policySets="list of xs:QName"?>*
    <interface ... />?
    <binding ... />*
    <callback?
      <binding ... />+
    </callback>
  </service>
  ...
</composite>
```

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

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

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

- **interface : Interface (0..1)** - If a composite service **interface** is specified it must be the same or a compatible subset of the interface provided by the promoted component service, i.e. provide a subset of the operations defined by the component service, [ASM60005] The interface is described by **zero or one interface element** which is a child element of the service element. For details on the interface element see the [Interface section](#).
- **binding : Binding (0..n)** - If bindings are specified they **override** the bindings defined for the promoted component service from the composite service perspective. The bindings defined on the component service are still in effect for local wires within the composite that target the component service. A service element has zero or more **binding elements** as children. Details of the binding element are described in the [Bindings section](#). For more details on wiring see the [Wiring section](#).
- **callback (0..1) / binding : Binding (1..n)** - A service element has an optional **callback** element used if the interface has a callback defined, which has one or more **binding** elements as children. The **callback** and its binding child elements are specified if there is a need to have binding details used to handle callbacks. If the callback element is not present, the behaviour is runtime implementation dependent.

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

### 6.1.1 Service Examples

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

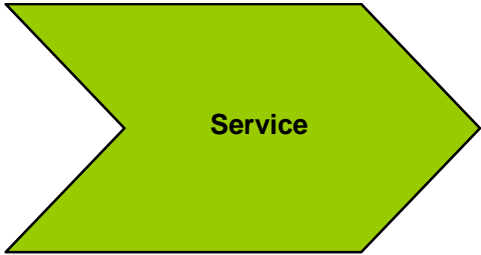


Figure 7: Service symbol

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



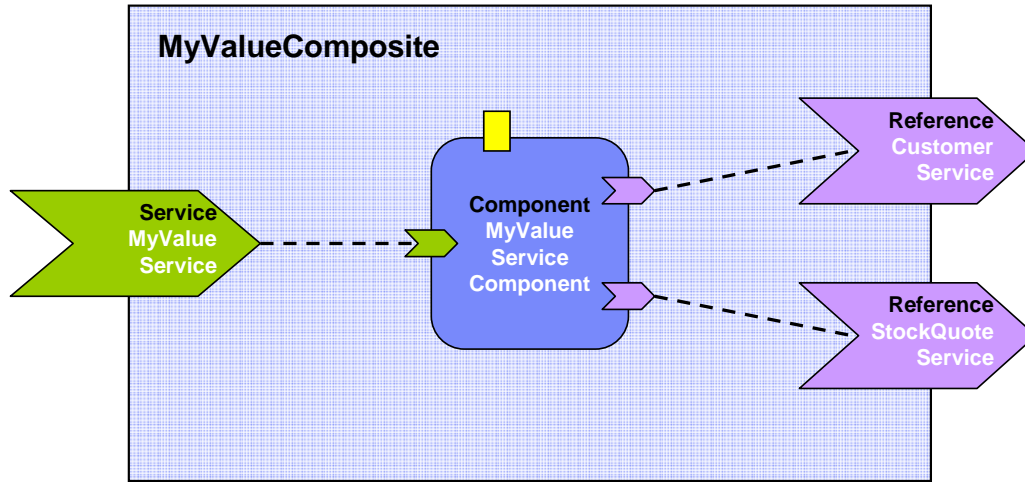


Figure 8: MyValueComposite showing Service

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_4 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  targetNamespace="http://foo.com"
  name="MyValueComposite" >

  ...

  <service name="MyValueService" promote="MyValueServiceComponent">
    <interface.java interface="services.myvalue.MyValueService"/>
    <binding.ws port="http://www.myvalue.org/MyValueService#
      wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
  </service>

  <component name="MyValueServiceComponent">
    <implementation.java
class="services.myvalue.MyValueServiceImpl"/>
    <property name="currency">EURO</property>
    <service name="MyValueService"/>
    <reference name="customerService"/>
    <reference name="StockQuoteService"/>
  </component>
```

1333  
1334       ...  
1335  
1336       </composite>  
1337

## 1338 6.2 Reference

1339       The **references of a composite** are defined by **promoting** references defined by components  
1340       contained in the composite. Each promoted reference indicates that the component reference  
1341       needs to be resolved by services outside the composite. A component reference is promoted using  
1342       a composite **reference element**.

1343       A composite reference is represented by a **reference element** which is a child of a composite  
1344       element. There can be **zero or more** reference elements in a composite. The following snippet  
1345       shows the composite schema with the schema for a **reference** element.

1346  
1347       <?xml version="1.0" encoding="ASCII"?>  
1348       <!-- Composite Reference schema snippet -->  
1349       <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1350       ...  
1351       <reference name="xs:NCName" target="list of xs:anyURI"?  
1352               promote="list of xs:anyURI" wiredByImpl="xs:boolean"?  
1353               multiplicity="0..1 or 1..1 or 0..n or 1..n"?  
1354               requires="list of xs:QName"? policySets="list of xs:QName"?>\*  
1355           <interface ... />?  
1356           <binding ... />\*  
1357           <callback>?  
1358               <binding ... />+  
1359           </callback>  
1360       </reference>  
1361       ...  
1362       </composite>  
1363  
1364

1365       The **composite reference** element has the following **attributes**:

- 1366       • **name : NCName (1..1)** – the name of the reference. The name of a composite  
1367        <reference/> element MUST be unique across all the composite references in the  
1368        composite. [ASM60006] The name of the composite reference can be different then the  
1369        name of the promoted component reference.
- 1370       • **promote : anyURI (1..n)** – identifies one or more promoted component references. The  
1371        value is a list of values of the form <component-name>/<reference-name> separated by  
1372        spaces. The specification of the reference name is optional if the component has only one  
1373        reference. Each of the URIs declared by a composite reference's @promote attribute MUST  
1374        identify a component reference within the composite. [ASM60007]

1375       The same component reference can be promoted more than once, using different  
1376       composite references, but only if the multiplicity defined on the component reference is  
1377       0..n or 1..n. The multiplicity on the composite reference can restrict accordingly.

1378       Where a composite reference promotes two or more component references:

- 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. [ASM60008]
  - the intents declared on a composite reference and on the component references which it promotes MUST NOT be mutually exclusive. [ASM60009] The intents which apply to the composite reference in this case are the union of the required intents specified for each of the promoted component references plus any intents declared on the composite reference itself. If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST raise an error. [ASM60010]
  - requires : QName (0..n)** – a list of required policy intents. See the [Policy Framework specification \[10\]](#) for a description of this attribute. Specified **required intents** add to or further qualify the required intents defined for the promoted component reference.
  - policySets : QName (0..n)** – a list of policy sets. See the [Policy Framework specification \[10\]](#) for a description of this attribute.
  - multiplicity : 0..1|1..1|0..n|1..n (1..1)** - Defines the number of wires that can connect the reference to target services. The multiplicity can have the following values
    - 0..1 – zero or one wire can have the reference as a source
    - 1..1 – one wire can have the reference as a source
    - 0..n - zero or more wires can have the reference as a source
    - 1..n – one or more wires can have the reference as a source

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. [ASM60011]
  - target : anyURI (0..n)** – a list of one or more of target service URI's, depending on multiplicity setting. Each value wires the reference to a service in a composite that uses the composite containing the reference as an implementation for one of its components. For more details on wiring see [the section on Wires](#).
  - wiredByImpl : boolean (0..1)** – a boolean value, "false" by default, which indicates that the implementation wires this reference dynamically. If set to "true" it indicates that the target of the reference is set at runtime by the implementation code (eg by the code obtaining an endpoint reference by some means and setting this as the target of the reference through the use of programming interfaces defined by the relevant Client and Implementation specification). If "true" is set, then the reference should not be wired statically within a using composite, but left [unwired](#).
- The **composite reference** element has the following **child elements**, whatever is not specified is defaulted from the promoted component reference(s).
- interface : Interface (0..1) - zero or one interface element** which declares an interface for the composite reference. If a composite reference has an **interface** specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s), i.e. provide a superset of the operations in the interface defined by the component for the reference. [ASM60012] 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

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

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

the interface(s) declared by the promoted component reference(s).

[ASM60013] For details on the interface element see [the Interface section](#).

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

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

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

## 6.2.1 Example Reference

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

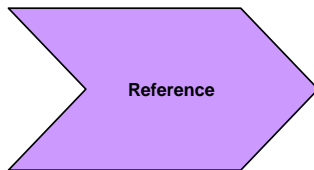


Figure 9: Reference symbol

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

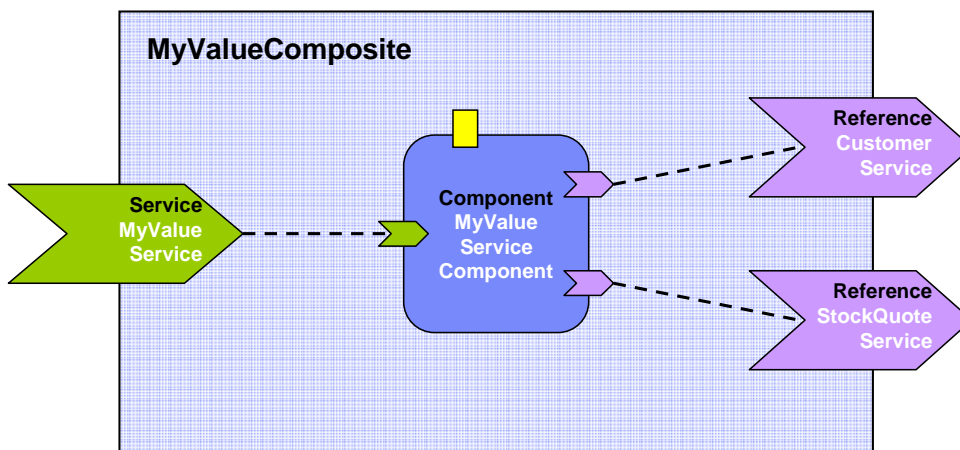


Figure 10: MyValueComposite showing References

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.

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_3 example -->
<composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  targetNamespace="http://foo.com"
  name="MyValueComposite" >

  ...

  <component name="MyValueServiceComponent">
    <implementation.java
class="services.myvalue.MyValueServiceImpl"/>
    <property name="currency">EURO</property>
    <reference name="customerService"/>
    <reference name="StockQuoteService"/>
  </component>

  <reference name="CustomerService"
    promote="MyValueServiceComponent/customerService">
    <interface.java interface="services.customer.CustomerService"/>
    <!-- The following forces the binding to be binding.sca whatever
is -->
    <!-- specified by the component reference or by the underlying
-->
    <!-- implementation
-->
    <binding.sca/>
  </reference>

  <reference name="StockQuoteService"
    promote="MyValueServiceComponent/StockQuoteService">
    <interface.java
interface="services.stockquote.StockQuoteService"/>
    <binding.ws port="http://www.stockquote.org/StockQuoteService#
wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
```

1506       </reference>  
1507  
1508       ...  
1509  
1510       </composite>  
1511

## 1512 6.3 Property

1513 **Properties** allow for the configuration of an implementation with externally set data values. A  
1514 composite can declare zero or more properties. Each property has a type, which may be either  
1515 simple or complex. An implementation can also define a default value for a property. Properties  
1516 can be configured with values in the components that use the implementation.

1517 The declaration of a property in a composite follows the form described in the following schema  
1518 snippet:

1519  
1520 <?xml version="1.0" encoding="ASCII"?>  
1521 <!-- Composite Property schema snippet -->  
1522 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712" ... >  
1523     ...  
1524     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")  
1525         many="xs:boolean"? mustSupply="xs:boolean"?>\*<br>1526         default-property-value?  
1527     </property>  
1528     ...  
1529 </composite>  
1530

1531 The **composite property** element has the following **attributes**:

- 1532     ▪ **name : NCName (1..1)** - the name of the property. [The name attribute of a composite](#)  
1533       [property MUST be unique amongst the properties of the same composite.](#) [\[ASM60014\]](#)
- 1534     ▪ one of **(1..1)**:
- 1535       ◦ **type : QName** – the type of the property - the qualified name of an XML schema  
1536         type
  - 1537       ◦ **element : QName** – the type of the property defined as the qualified name of an  
1538         XML schema global element – the type is the type of the global element
- 1539     ▪ **many : boolean (0..1)** - whether the property is single-valued (false) or multi-valued  
1540       (true). The default is **false**. In the case of a multi-valued property, it is presented to the  
1541       implementation as a collection of property values.
- 1542     ▪ **mustSupply : boolean (0..1)** – whether the property value has to be supplied by the  
1543       component that uses the composite – when mustSupply="true" the component has to  
1544       supply a value since the composite has no default value for the property. A default-  
1545       property-value is only worth declaring when mustSupply="false" (the default setting for  
1546       the mustSupply attribute), since the implication of a default value is that it is used only  
1547       when a value is not supplied by the using component.

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

1548  
1549 The property element may contain an optional **default-property-value**, which provides default  
1550 value for the property. The default value must match the type declared for the property:

- 1551           o a string, if **type** is a simple type (matching the **type** declared)
- 1552           o a complex type value matching the type declared by **type**
- 1553           o an element matching the element named by **element**
- 1554           o multiple values are permitted if many="true" is specified
- 1555

1556 Implementation types other than **composite** can declare properties in an implementation-  
1557 dependent form (eg annotations within a Java class), or through a property declaration of exactly  
1558 the form described above in a componentType file.

1559 Property values can be configured when an implementation is used by a component. The form of  
1560 the property configuration is shown in [the section on Components](#).

### 1561 6.3.1 Property Examples

1562  
1563 For the following example of Property declaration and value setting, the following complex type is  
1564 used as an example:

```
1565 <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
1566             targetNamespace="http://foo.com/"
1567             xmlns:tns="http://foo.com/">
1568   <!-- ComplexProperty schema -->
1569   <xsd:element name="fooElement" type="MyComplexType"/>
1570   <xsd:complexType name="MyComplexType">
1571     <xsd:sequence>
1572       <xsd:element name="a" type="xsd:string"/>
1573       <xsd:element name="b" type="anyURI"/>
1574     </xsd:sequence>
1575     <attribute name="attr" type="xsd:string" use="optional"/>
1576   </xsd:complexType>
1577 </xsd:schema>
```

1579 The following composite demonstrates the declaration of a property of a complex type, with a  
1580 default value, plus it demonstrates the setting of a property value of a complex type within a  
1581 component:

```
1582 <?xml version="1.0" encoding="ASCII"?>
1583
1584 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1585            xmlns:foo="http://foo.com"
1586            targetNamespace="http://foo.com"
1587            name="AccountServices">
1588   <!-- AccountServices Example1 -->
1589
1590   ...
1591
1592   <property name="complexFoo" type="foo:MyComplexType">
1593     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1594       <foo:a>AValue</foo:a>
```

```

1595         <foo:b>InterestingURI</foo:b>
1596     </MyComplexPropertyValue>
1597 </property>
1598
1599 <component name="AccountServiceComponent">
1600     <implementation.java class="foo.AccountServiceImpl"/>
1601     <property name="complexBar" source="$complexFoo"/>
1602     <reference name="accountDataService"
1603         target="AccountDataServiceComponent"/>
1604     <reference name="stockQuoteService" target="StockQuoteService"/>
1605 </component>
1606
1607 ...
1608
1609 </composite>

```

1610 In the declaration of the property named **complexFoo** in the composite **AccountServices**, the  
1611 property is defined to be of type **foo:MyComplexType**. The namespace **foo** is declared in the  
1612 composite and it references the example XSD, where MyComplexType is defined. The declaration  
1613 of complexFoo contains a default value. This is declared as the content of the property element.  
1614 In this example, the default value consists of the element **MyComplexPropertyValue** of type  
1615 foo:MyComplexType and its two child elements <foo:a> and <foo:b>, following the definition of  
1616 MyComplexType.

1617 In the component **AccountServiceComponent**, the component sets the value of the property  
1618 **complexBar**, declared by the implementation configured by the component. In this case, the  
1619 type of complexBar is foo:MyComplexType. The example shows that the value of the complexBar  
1620 property is set from the value of the complexFoo property – the **source** attribute of the property  
1621 element for complexBar declares that the value of the property is set from the value of a property  
1622 of the containing composite. The value of the source attribute is **\$complexFoo**, where  
1623 complexFoo is the name of a property of the composite. This value implies that the whole of the  
1624 value of the source property is used to set the value of the component property.

1625 The following example illustrates the setting of the value of a property of a simple type (a string)  
1626 from **part** of the value of a property of the containing composite which has a complex type:

```

1627 <?xml version="1.0" encoding="ASCII"?>
1628
1629 <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1630                xmlns:foo="http://foo.com"
1631                targetNamespace="http://foo.com"
1632                name="AccountServices">
1633 <!-- AccountServices Example2 -->
1634
1635 ...
1636
1637 <property name="complexFoo" type="foo:MyComplexType">
1638     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1639         <foo:a>AValue</foo:a>
1640         <foo:b>InterestingURI</foo:b>
1641     </MyComplexPropertyValue>

```



```

1642     </property>
1643
1644     <component name="AccountServiceComponent">
1645         <implementation.java class="foo.AccountServiceImpl"/>
1646         <property name="currency" source="$complexFoo/a"/>
1647         <reference name="accountDataService"
1648             target="AccountDataServiceComponent"/>
1649         <reference name="stockQuoteService" target="StockQuoteService"/>
1650     </component>
1651
1652     ...
1653
1654 </composite>

```

1655 In this example, the component **AccountServiceComponent** sets the value of a property called  
1656 **currency**, which is of type string. The value is set from a property of the composite  
1657 **AccountServices** using the source attribute set to **\$complexFoo/a**. This is an XPath expression  
1658 that selects the property name **complexFoo** and then selects the value of the **a** subelement of  
1659 complexFoo. The "a" subelement is a string, matching the type of the currency property.

1660 Further examples of declaring properties and setting property values in a component follow:

1661 Declaration of a property with a simple type and a default value:

```

1662 <property name="SimpleTypeProperty" type="xsd:string">
1663     MyValue
1664 </property>

```

1665
1666 Declaration of a property with a complex type and a default value:

```

1667 <property name="complexFoo" type="foo:MyComplexType">
1668     <MyComplexPropertyValue xsi:type="foo:MyComplexType">
1669         <foo:a>AValue</foo:a>
1670         <foo:b>InterestingURI</foo:b>
1671     </MyComplexPropertyValue>
1672 </property>

```

1673
1674 Declaration of a property with an element type:

```

1675 <property name="elementFoo" element="foo:fooElement">
1676     <foo:fooElement>
1677         <foo:a>AValue</foo:a>
1678         <foo:b>InterestingURI</foo:b>
1679     </foo:fooElement>
1680 </property>

```

1681
1682 Property value for a simple type:

```

1683 <property name="SimpleTypeProperty">
1684     MyValue
1685 </property>

```

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

```
<property name="complexFoo">
  <MyComplexPropertyValue xsi:type="foo:MyComplexType" attr="bar">
    <foo:a>AValue</foo:a>
    <foo:b>InterestingURI</foo:b>
  </MyComplexPropertyValue>
</property>
```

Property value for an element type:

```
<property name="elementFoo">
  <foo:fooElement attr="bar">
    <foo:a>AValue</foo:a>
    <foo:b>InterestingURI</foo:b>
  </foo:fooElement>
</property>
```

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

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

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

```
<property name="complexFoo">
  <MyComplexPropertyValue1 xsi:type="foo:MyComplexType" attr="bar">
    <foo:a>AValue</foo:a>
    <foo:b>InterestingURI</foo:b>
  </MyComplexPropertyValue1>
  <MyComplexPropertyValue2 xsi:type="foo:MyComplexType" attr="zing">
    <foo:a>BValue</foo:a>
    <foo:b>BoringURI</foo:b>
  </MyComplexPropertyValue2>
</property>
```

## 6.4 Wire

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

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

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

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

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

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

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

    <wire source="xs:anyURI" target="xs:anyURI" />*
</composite>
```

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

- **<component-name>/<service-name>**
  - where the target is a service of a component. The specification of the service name is optional if the target component only has one service with a compatible interface

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

- **source (1..1)** – names the source component reference. Valid URI schemes are:
  - **<component-name>/<reference-name>**
    - where the source is a component reference. The specification of the reference name is optional if the source component only has one reference
- **target (1..1)** – names the target component service. Valid URI schemes are
  - **<component-name>/<service-name>**
    - where the target is a service of a component. The specification of the service name is optional if the target component only has one service with a compatible interface

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

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

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

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

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

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

#### 6.4.1 Wire Examples

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

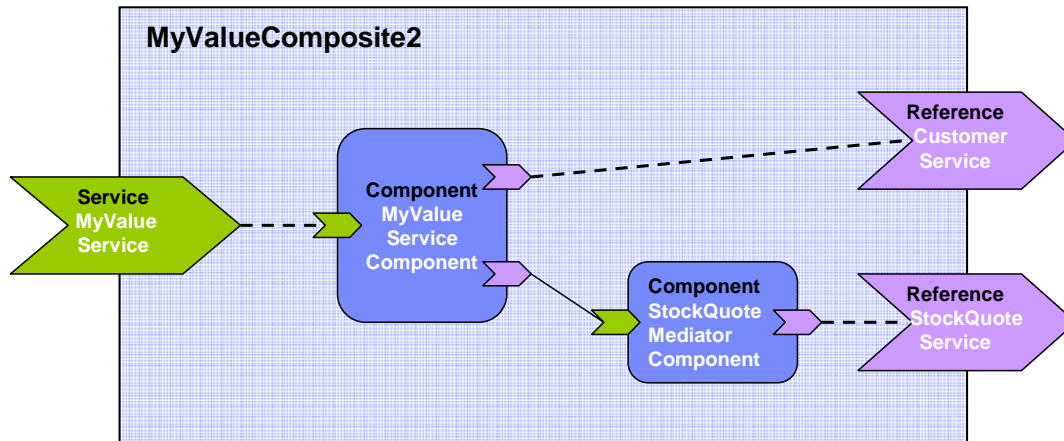


Figure 11: MyValueComposite2 showing Wires

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

```

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

  <service name="MyValueService" promote="MyValueServiceComponent">
    <interface.java interface="services.myvalue.MyValueService"/>
    <binding.ws port="http://www.myvalue.org/MyValueService#
      wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
  </service>

  <component name="MyValueServiceComponent">
    <implementation.java
      class="services.myvalue.MyValueServiceImpl"/>
    <property name="currency">EURO</property>
    <service name="MyValueService"/>
    <reference name="customerService"/>
    <reference name="stockQuoteService"/>
  </component>

```

```

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

```

## 6.4.2 Autowire

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

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

- reference
- component
- composite

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

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

1890 turned off for specific components and specific references through setting autowire="false" on the  
1891 components and references concerned.

1892 For each component reference for which autowire is enabled, the the SCA runtime MUST search  
1893 within the composite for target services which are compatible with the reference. [ASM60022]  
1894 "Compatible" here means:

- 1895
- 1896 • the target service interface MUST be a compatible superset of the reference interface  
when using autowire to wire a reference (as defined in the section on Wires). [ASM60023]
  - 1897 • the intents, and policies applied to the service MUST be compatible with those on the  
1898 reference when using autowire to wire a reference – so that wiring the reference to the  
1899 service will not cause an error due to policy mismatch [ASM60024] (see the Policy  
1900 Framework specification [10] for details)

1901 If the search finds **1 or more** valid target service for a particular reference, the action taken  
1902 depends on the multiplicity of the reference:

- 1903
- 1904 • for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the  
1905 reference to one of the set of valid target services chosen from the set in a runtime-  
dependent fashion [ASM60025]
  - 1906 • for an autowire reference with multiplicity 0..n or 1..n, the reference MUST be wired to all  
1907 of the set of valid target services [ASM60026]

1908 If the search finds **no** valid target services for a particular reference, the action taken depends on  
1909 the multiplicity of the reference:

- 1910
- 1911 • for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid  
1912 target service, there is no problem – no services are wired and the SCA runtime MUST  
NOT raise an error [ASM60027]
  - 1913 • for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid  
1914 target services an error MUST be raised by the SCA runtime since the reference is  
1915 intended to be wired [ASM60028]
- 1916

### 1917 6.4.3 Autowire Examples

1918 This example demonstrates two versions of the same composite – the first version is done using  
1919 explicit wires, with no autowiring used, the second version is done using autowire. In both cases  
1920 the end result is the same – the same wires connect the references to the services.

1921 First, here is a diagram for the composite:

**Formatted:** Default Paragraph  
Font

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

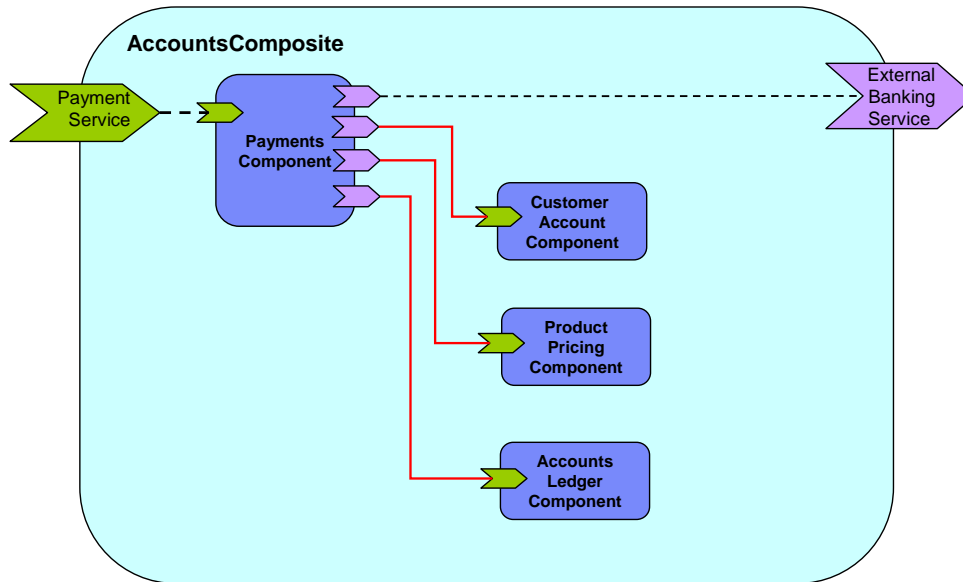


Figure 12: Example Composite for Autowire

First, the composite using explicit wires:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Autowire Example - No autowire -->
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  xmlns:foo="http://foo.com"
  targetNamespace="http://foo.com"
  name="AccountComposite">

  <service name="PaymentService" promote="PaymentsComponent"/>

  <component name="PaymentsComponent">
    <implementation.java class="com.foo.accounts.Payments"/>
    <service name="PaymentService"/>
    <reference name="CustomerAccountService"
      target="CustomerAccountComponent"/>
    <reference name="ProductPricingService"
      target="ProductPricingComponent"/>
    <reference name="AccountsLedgerService"
      target="AccountsLedgerComponent"/>
    <reference name="ExternalBankingService"/>
  </component>

  <component name="CustomerAccountComponent">

```



```

1948         <implementation.java class="com.foo.accounts.CustomerAccount"/>
1949     </component>
1950
1951     <component name="ProductPricingComponent">
1952         <implementation.java class="com.foo.accounts.ProductPricing"/>
1953     </component>
1954
1955     <component name="AccountsLedgerComponent">
1956         <implementation.composite name="foo:AccountsLedgerComposite"/>
1957     </component>
1958
1959     <reference name="ExternalBankingService"
1960         promote="PaymentsComponent/ExternalBankingService"/>
1961
1962 </composite>
1963

```

Secondly, the composite using autowire:

```

1965 <?xml version="1.0" encoding="UTF-8"?>
1966 <!-- Autowire Example - With autowire -->
1967 <composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
1968     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
1969     xmlns:foo="http://foo.com"
1970     targetNamespace="http://foo.com"
1971     name="AccountComposite">
1972
1973     <service name="PaymentService" promote="PaymentsComponent">
1974         <interface.java class="com.foo.PaymentServiceInterface"/>
1975     </service>
1976
1977     <component name="PaymentsComponent" autowire="true">
1978         <implementation.java class="com.foo.accounts.Payments"/>
1979         <service name="PaymentService"/>
1980         <reference name="CustomerAccountService"/>
1981         <reference name="ProductPricingService"/>
1982         <reference name="AccountsLedgerService"/>
1983         <reference name="ExternalBankingService"/>
1984     </component>
1985
1986     <component name="CustomerAccountComponent">
1987         <implementation.java class="com.foo.accounts.CustomerAccount"/>
1988     </component>
1989
1990     <component name="ProductPricingComponent">

```

```

1991         <implementation.java class="com.foo.accounts.ProductPricing"/>
1992     </component>
1993
1994     <component name="AccountsLedgerComponent">
1995         <implementation.composite name="foo:AccountsLedgerComposite"/>
1996     </component>
1997
1998     <reference name="ExternalBankingService"
1999         promote="PaymentsComponent/ExternalBankingService"/>
2000
2001 </composite>

```

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

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

## 6.5 Using Composites as Component Implementations

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

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

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

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

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

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

```

2034 <?xml version="1.0" encoding="ASCII"?>
2035 <!-- Composite Implementation schema snippet -->
2036 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2037     targetNamespace="xs:anyURI"

```

```

2038         name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2039         constrainingType="QName"?
2040         requires="list of xs:QName"? policySets="list of
2041 xs:QName"?>
2042
2043     ...
2044
2045     <component name="xs:NCName" autowire="xs:boolean"?
2046         requires="list of xs:QName"? policySets="list of xs:QName"?>*
2047     <implementation.composite name="xs:QName"/>?
2048     <service name="xs:NCName" requires="list of xs:QName"?
2049         policySets="list of xs:QName"?>*
2050     <interface ... />?
2051     <binding uri="xs:anyURI" name="xs:QName"?
2052         requires="list of xs:QName"
2053         policySets="list of xs:QName"?/>*
2054     <callback>?
2055         <binding uri="xs:anyURI"? name="xs:QName"?
2056             requires="list of xs:QName"?
2057             policySets="list of xs:QName"?/>+
2058     </callback>
2059 </service>
2060 <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
2061     source="xs:string"? file="xs:anyURI"?>*
2062     property-value
2063 </property>
2064 <reference name="xs:NCName" target="list of xs:anyURI"?
2065     autowire="xs:boolean"? wiredByImpl="xs:boolean"?
2066     requires="list of xs:QName"? policySets="list of xs:QName"?
2067     multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
2068 <interface ... />?
2069 <binding uri="xs:anyURI"? name="xs:QName"?
2070     requires="list of xs:QName" policySets="list of
2071 xs:QName"?/>*
2072 <callback>?
2073     <binding uri="xs:anyURI"? name="xs:QName"?
2074         requires="list of xs:QName"?
2075         policySets="list of xs:QName"?/>+
2076 </callback>
2077 </reference>
2078 </component>
2079
2080     ...

```

</composite>

The implementation.composite element has the following attribute:

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- CompositeComponent example -->
<composite xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xsd:schemaLocation="http://docs.oasis-open.org/ns/opencsa/sca/200712
file:/C:/Strategy/SCA/v09_osoaschemas/schemas/sca.xsd"
  xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  targetNamespace="http://foo.com"
  xmlns:foo="http://foo.com"
  name="AccountComposite">

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

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

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

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

```

2125         <reference name="AccountDataService" target="AccountDataService"/>
2126         <reference name="StockQuoteService"/>
2127
2128         <property name="currency" source="$currency"/>
2129     </component>
2130
2131     <component name="AccountDataService">
2132         <implementation.composite name="foo:AccountDataServiceComposite"/>
2133
2134         <property name="currency" source="$currency"/>
2135     </component>
2136
2137 </composite>
2138

```

## 2139 6.6 Using Composites through Inclusion

2140 In order to assist team development, composites may be developed in the form of multiple  
 2141 physical artifacts that are merged into a single logical unit.

2142 A composite is defined in an **xxx.composite** file and the composite may receive additional  
 2143 content through the **inclusion of other composite** files.

2144 The semantics of included composites are that the content of the included composite is inlined into  
 2145 the using composite **xxx.composite** file through **include** elements in the using composite. The  
 2146 effect is one of **textual inclusion** – that is, the text content of the included composite is placed  
 2147 into the using composite in place of the include statement. The included composite element itself  
 2148 is discarded in this process – only its contents are included.

2149 The composite file used for inclusion can have any contents, but always contains a single  
 2150 **composite** element. The composite element can contain any of the elements which are valid as  
 2151 child elements of a composite element, namely components, services, references, wires and  
 2152 includes. There is no need for the content of an included composite to be complete, so that  
 2153 artifacts defined within the using composite or in another associated included composite file may  
 2154 be referenced. For example, it is permissible to have two components in one composite file while a  
 2155 wire specifying one component as the source and the other as the target can be defined in a  
 2156 second included composite file.

2157 The SCA runtime MUST raise an error if the composite resulting from the inclusion of one  
 2158 composite into another is invalid. [ASM60031] For example, it is an error if there are duplicated  
 2159 elements in the using composite (eg. two services with the same uri contributed by different  
 2160 included composites), or if there are wires with non-existent source or target.

2161 The following snippet shows the partial schema for the include element.

```

2162
2163 <?xml version="1.0" encoding="UTF-8"?>
2164 <!-- Include snippet -->
2165 <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2166                targetNamespace="xs:anyURI"
2167                name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
2168                constrainingType="QName"?
2169                requires="list of xs:QName"? policySets="list of
2170 xs:QName"?>

```

```

2171
2172     ...
2173
2174     <include name="xs:QName" />*
2175
2176     ...
2177
2178 </composite>
2179

```

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

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

### 6.6.1 Included Composite Examples

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

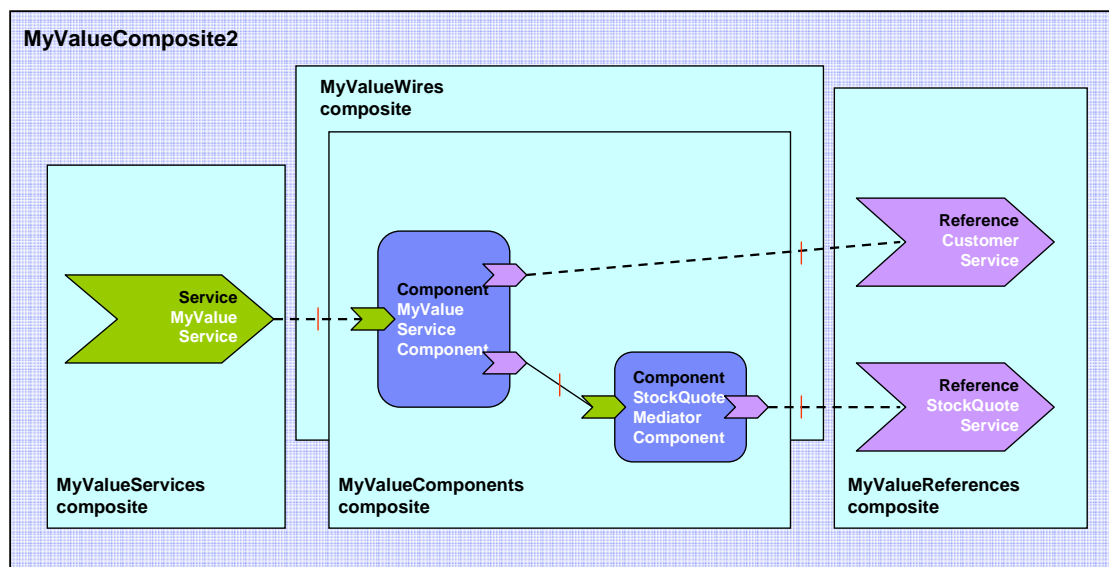


Figure 13 MyValueComposite2 built from 4 included composites

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

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

  <include name="foo:MyValueServices"/>
  <include name="foo:MyValueComponents"/>
  <include name="foo:MyValueReferences"/>
  <include name="foo:MyValueWires"/>

</composite>
```

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

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

  <service name="MyValueService" promote="MyValueServiceComponent">
    <interface.java interface="services.myvalue.MyValueService"/>
    <binding.ws port="http://www.myvalue.org/MyValueService#
      wsdl.endpoint(MyValueService/MyValueServiceSOAP)"/>
  </service>

</composite>
```

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

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

  <component name="MyValueServiceComponent">
```

```

2243         <implementation.java
2244 class="services.myvalue.MyValueServiceImpl"/>
2245         <property name="currency">EURO</property>
2246     </component>
2247
2248     <component name="StockQuoteMediatorComponent">
2249         <implementation.java class="services.myvalue.SQMediatorImpl"/>
2250         <property name="currency">EURO</property>
2251     </component>
2252
2253 </composite>
2254

```

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

```

2255
2256
2257 <?xml version="1.0" encoding="ASCII"?>
2258 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2259     targetNamespace="http://foo.com"
2260     xmlns:foo="http://foo.com"
2261     name="MyValueReferences" >
2262
2263     <reference name="CustomerService"
2264         promote="MyValueServiceComponent/CustomerService">
2265         <interface.java interface="services.customer.CustomerService"/>
2266         <binding.sca/>
2267     </reference>
2268
2269     <reference name="StockQuoteService"
2270 promote="StockQuoteMediatorComponent">
2271         <interface.java
2272 interface="services.stockquote.StockQuoteService"/>
2273         <binding.ws port="http://www.stockquote.org/StockQuoteService#
2274             wsdl.endpoint(StockQuoteService/StockQuoteServiceSOAP)"/>
2275     </reference>
2276
2277 </composite>

```

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

```

2278
2279
2280 <?xml version="1.0" encoding="ASCII"?>
2281 <composite xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2282     targetNamespace="http://foo.com"
2283     xmlns:foo="http://foo.com"
2284     name="MyValueWires" >
2285
2286     <wire source="MyValueServiceComponent/stockQuoteService"

```



```
2287         target="StockQuoteMediatorComponent" />
2288
2289     </composite>
```

## 2290 **6.7 Composites which Include Component Implementations of**

### 2291 **Multiple Types**

2292

2293 A Composite containing multiple components can have multiple component implementation types.

2294 For example, a Composite may include one component with a Java POJO as its implementation

2295 and another component with a BPEL process as its implementation.

2296

---

## 7 ConstrainingType

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

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

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

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

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

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

    <service name="xs:NCName" requires="list of xs:QName"?>*
        <interface ... />?
    </service>

    <reference name="xs:NCName"
        multiplicity="0..1 or 1..1 or 0..n or 1..n"?
        requires="list of xs:QName"?>*
        <interface ... />?
    </reference>

    <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
        many="xs:boolean"? mustSupply="xs:boolean"?>*
```

2342               default-property-value?  
2343       </property>  
2344  
2345   </constrainingType>  
2346

2347   The constrainingType element has the following **attributes**:

- 2348       • **name (1..1)** – the name of the constrainingType. The form of a constrainingType name is  
2349       an XML QName, in the namespace identified by the targetNamespace attribute. The name  
2350       attribute of the constraining type MUST be unique in the SCA domain. [ASM70003]
- 2351       • **targetNamespace (0..1)** – an identifier for a target namespace into which the  
2352       constrainingType is declared
- 2353       • **requires (0..1)** – a list of policy intents. See the Policy Framework specification [10] for  
2354       a description of this attribute.

2355   ConstrainingType contains **zero or more properties, services, references**.

2356

2357   When an implementation is constrained by a constrainingType its component type MUST contain  
2358   all the services, references and properties specified in the constrainingType. [ASM70004] The  
2359   constraining type's references and services will have interfaces specified and can have intents  
2360   specified. An implementation MAY contain additional services, additional optional references  
2361   (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the  
2362   constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or  
2363   1..n) or additional non-optional properties (a property with mustSupply=true). [ASM70005]

2364   When a component is constrained by a constrainingType via the "constrainingType" attribute, the  
2365   entire componentType associated with the component and its implementation is not visible to the  
2366   containing composite. The containing composite can only see a projection of the componentType  
2367   associated with the component and implementation as scoped by the constrainingType of the  
2368   component. Additional services, references and properties provided by the implementation which  
2369   are not declared in the constrainingType associated with a component MUST NOT be configured in  
2370   any way by the containing composite. [ASM70006] This requirement ensures that the  
2371   constrainingType contract cannot be violated by the composite.

2372   The constrainingType can include required intents on any element. Those intents are applied to  
2373   any component that uses that constrainingType. In other words, if requires="reliability" exists on  
2374   a constrainingType, or its child service or reference elements, then a constrained component or its  
2375   implementation must include requires="reliability" on the component or implementation or on its  
2376   corresponding service or reference. A component or implementation can use a qualified form of  
2377   an intent specified in unqualified form in the constrainingType, but if the constrainingType uses  
2378   the qualified form of an intent, then the component or implementation MUST also use the qualified  
2379   form, otherwise there is an error. [ASM70007]

2380   A constrainingType can be applied to an implementation. In this case, the implementation's  
2381   componentType has a constrainingType attribute set to the QName of the constrainingType.

2382

## 2383   7.1 Example constrainingType

2384

2385   The following snippet shows the contents of the component called "MyValueServiceComponent"  
2386   which is constrained by the constrainingType myns:CT. The componentType associated with the  
2387   implementation is also shown.

2388

```
2389   <component name="MyValueServiceComponent" constrainingType="myns:CT">  
2390    <implementation.java class="services.myvalue.MyValueServiceImpl"/>
```

```

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

```

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

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

## 8 Interface

**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.

SCA currently supports the following interface type systems:

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

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](#).

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

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

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

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

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

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

The interface.wSDL element has the following attributes:

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

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

```
<interface.wSDL interface="http://www.stockquote.org/StockQuoteService#  
wSDL.interface(StockQuote)" />
```

2459 For WSDL 1.1, the interface attribute points to a portType in the WSDL. For WSDL 2.0, the  
2460 interface attribute points to an interface in the WSDL. For the WSDL 1.1 portType and WSDL 2.0  
2461 interface type systems, arguments and return of the service operations are described using XML  
2462 schema.

2463 For information about Java interfaces, including details of SCA-specific annotations, see the SCA  
2464 Java Common Annotations and APIs specification [1].

## 2465 8.1 Local and Remotable Interfaces

2466 A remotable service is one which may be called by a client which is running in an operating system  
2467 process different from that of the service itself (this also applies to clients running on different  
2468 machines from the service). Whether a service of a component implementation is remotable is  
2469 defined by the interface of the service. In the case of Java this is defined by adding the  
2470 **@Remotable** annotation to the Java interface (see [Client and Implementation Model Specification](#)  
2471 [for Java](#)). WSDL defined interfaces are always remotable.

2472

2473 The style of remotable interfaces is typically **coarse grained** and intended for **loosely coupled**  
2474 interactions. [Remotable service Interfaces MUST NOT make use of \*\*method or operation\*\*](#)  
2475 [overloading](#). [ASM80002] This restriction on operation overloading for remotable services aligns  
2476 with the WSDL 2.0 specification, which disallows operation overloading, and also with the WS-I  
2477 Basic Profile 1.1 (section 4.5.3 - R2304) which has a constraint which disallows operation  
2478 overloading when using WSDL 1.1.

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

2479

2480 Independent of whether the remotable service is called remotely from outside the process where  
2481 the service runs or from another component running in the same process, the data exchange  
2482 semantics are **by-value**.

2483 Implementations of remotable services can modify input messages (parameters) during or after  
2484 an invocation and can modify return messages (results) after the invocation. If a remotable  
2485 service is called locally or remotely, the SCA container MUST ensure sure that no modification of  
2486 input messages by the service or post-invocation modifications to return messages are seen by  
2487 the caller. [ASM80003]

2488 Here is a snippet which shows an example of a remotable java interface:

```
2489 package services.hello;  
2490  
2491 @Remotable  
2492 public interface HelloService {  
2493  
2494     String hello(String message);  
2495 }  
2496
```

2497

2498 It is possible for the implementation of a remotable service to indicate that it can be called using  
2499 by-reference data exchange semantics when it is called from a component in the same process.  
2500 This can be used to improve performance for service invocations between components that run in  
2501 the same process. This can be done using the @AllowsPassByReference annotation (see the [Java](#)  
2502 [Client and Implementation Specification](#)).

2503

2504 A service typed by a local interface can only be called by clients that are running in the same  
2505 process as the component that implements the local service. Local services cannot be published  
2506 via remotable services of a containing composite. In the case of Java a local service is defined by a  
2507 Java interface definition without a **@Remotable** annotation.

2508  
2509  
2510  
2511  
2512

The style of local interfaces is typically ***fine grained*** and intended for ***tightly coupled*** interactions. Local service interfaces can make use of ***method or operation overloading***.  
The data exchange semantic for calls to services typed by local interfaces is ***by-reference***.

2513 **8.2 Bidirectional Interfaces**

2514 The relationship of a business service to another business service is often peer-to-peer, requiring  
2515 a two-way dependency at the service level. In other words, a business service represents both a  
2516 consumer of a service provided by a partner business service and a provider of a service to the  
2517 partner business service. This is especially the case when the interactions are based on  
2518 asynchronous messaging rather than on remote procedure calls. The notion of ***bidirectional***  
2519 ***interfaces*** is used in SCA to directly model peer-to-peer bidirectional business service  
2520 relationships.

2521 An interface element for a particular interface type system needs to allow the specification of an  
2522 optional callback interface. If a callback interface is specified, SCA refers to the interface as a  
2523 whole as a bidirectional interface.

2524 The following snippet shows the interface element defined using Java interfaces with an optional  
2525 callbackInterface attribute.

2526

2527 `<interface.java interface="services.invoicing.ComputePrice"`  
2528 `callbackInterface="services.invoicing.InvoiceCallback"/>`

2529

2530 If a service is defined using a bidirectional interface element then its implementation implements  
2531 the interface, and its implementation uses the callback interface to converse with the client that  
2532 called the service interface.

2533

2534 If a reference is defined using a bidirectional interface element, the client component  
2535 implementation using the reference calls the referenced service using the interface. The client  
2536 MUST provide an implementation of the callback interface. [ASM80004]

2537 Callbacks can be used for both remotable and local services. Either both interfaces of a  
2538 bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT  
2539 mix local and remote services. [ASM80005]

2540 Note that an interface document such as a WSDL file or a Java interface can contain annotations  
2541 that declare a callback interface for a particular interface. Whenever an interface document  
2542 declaring a callback interface is used in the declaration of an <interface/> element in SCA, it  
2543 MUST be treated as being bidirectional with the declared callback interface. [ASM80010] In such  
2544 cases, there is no requirement for the <interface/> element to declare the callback interface  
2545 explicitly.

Formatted: Font color: Red

2546 If an <interface/> element references an interface document which declares a callback interface  
2547 and also itself contains a declaration of a callback interface, the two callback interfaces MUST be  
2548 compatible. [ASM80011]

Formatted: Font color: Red

2549 Where a component uses an implementation and the component configuration explicitly declares  
2550 an interface for a service or a reference, if the matching service or reference declaration in the  
2551 component type declares an interface which has a callback interface, then the component interface  
2552 declaration MUST also declare a compatible interface with a compatible callback interface.  
2553 [ASM80012] If the service or reference declaration in the component type declares an interface  
2554 without a callback interface, then the component configuration for the corresponding service or  
2555 reference MUST NOT declare an interface with a callback interface. [ASM80013]

Formatted: Font color: Red

Formatted: Font color: Red

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. [ASM80014] 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. [ASM80015]

Formatted: Font color: Red

Formatted: Font color: Red

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

### 8.3 Conversational Interfaces

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

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

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

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

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

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

- Application Developer: Declares that a service interface is conversational (leaving the details of the protocol up to the binding). Uses lifecycle semantics, APIs or other programmatic mechanisms (as defined by the implementation-type being used) to manage conversational state.
- Application Assembler: chooses a binding that can support conversations
- Binding Provider: implements a protocol that can pass conversational information with each operation request/response.
- Implementation-Type Provider: defines APIs and/or other programmatic mechanisms for application developers to access conversational information. Optionally implements



instance lifecycle semantics that automatically manage implementation state based on the binding's conversational information.

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

Comment [mbgl12]: Issue 35

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

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

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

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

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

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

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

- A message is received for a particular conversation, after the conversation has ended

- 2655           - The conversation identification is invalid (not unique, out of range, etc.)
- 2656           - The conversation identification is not present in the input message of the operation that
- 2657           ends the conversation
- 2658           - The client or the service attempts to send a message in a conversation, after the
- 2659           conversation has ended

2660       This fault is named within the SCA namespace standard prefix "sca", which corresponds to URI

2661       <http://docs.oasis-open.org/ns/opencsa/sca/200712>.

2662       The lifecycle of resources and the association between unique identifiers and conversations are

2663       determined by the service's implementation type and may not be directly affected by the

2664       "endConversation" annotation. For example, a WS-BPEL process can outlive most of the

2665       conversations that it is involved in.

2666       Although conversational interfaces do not require that any identifying information be passed as

2667       part of the body of messages, there is conceptually an identity associated with the conversation.

2668       Individual implementations types can have an API to access the ID associated with the

2669       conversation, although no assumptions can be made about the structure of that identifier.

2670       Implementation types can also have a means to set the conversation ID by either the client or the

2671       service provider, although the operation may only be supported by some binding/policy

2672       combinations.

2673       Implementation-type specifications are encouraged to define and provide conversational instance

2674       lifecycle management for components that implement conversational interfaces. However,

2675       implementations could also manage the conversational state manually.

2676

## 2677   8.4 SCA-Specific Aspects for WSDL Interfaces

2678       There are a number of aspects that SCA applies to interfaces in general, such as marking them

2679       **conversational**. These aspects apply to the interfaces themselves, rather than their use in a

2680       specific place within SCA. There is thus a need to provide appropriate ways of marking the

2681       interface definitions themselves, which go beyond the basic facilities provided by the interface

2682       definition language.

2683       For WSDL interfaces, there is an extension mechanism that permits additional information to be

2684       included within the WSDL document. SCA takes advantage of this extension mechanism. In order

2685       to use the SCA extension mechanism, the SCA namespace ([http://docs.oasis-](http://docs.oasis-open.org/ns/opencsa/sca/200712)

2686       [open.org/ns/opencsa/sca/200712](http://docs.oasis-open.org/ns/opencsa/sca/200712)) needs to be declared within the WSDL document.

2687       First, SCA defines a global attribute in the SCA namespace which provides a mechanism to attach

2688       policy intents - **@requires**. The definition of this attribute is as follows:

2689       

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

2690

2691       

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

2692       

```
  <list itemType="QName"/>
```

2693       

```
</simpleType>
```

Deleted: ¶

2694       The @requires attribute can be applied to WSDL Port Type elements (WSDL 1.1) and to WSDL

2695       Interface elements (WSDL 2.0). The attribute contains one or more intent names, as defined by

2696       the [Policy Framework specification \[10\]](#). Any service or reference that uses an interface marked

2697       with required intents MUST implicitly add those intents to its own @requires list. **[ASM80008]**

2698       To specify that a WSDL interface is conversational, the following attribute setting is used on either

2699       the WSDL Port Type or WSDL Interface:

2700       

```
requires="conversational"
```

2701       SCA defines an **endsConversation** attribute that is used to mark specific operations within a

2702       WSDL interface declaration as ending a conversation. This only has meaning for WSDL interfaces

2703       which are also marked conversational. The endsConversation attribute is a global attribute in the

2704       SCA namespace, with the following definition:

```
2705 <attribute name="endsConversation" type="boolean" default="false"/>
2706
2707 The following snippet is an example of a WSDL Port Type annotated with the requires attribute on
2708 the portType and the endsConversation attribute on one of the operations:
2709 ...
2710 <portType name="LoanService" sca:requires="conversational">
2711   <operation name="apply">
2712     <input message="tns:ApplicationInput"/>
2713     <output message="tns:ApplicationOutput"/>
2714   </operation>
2715   <operation name="cancel" sca:endsConversation="true">
2716   </operation>
2717   ...
2718 </portType>
2719 ...
```

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

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

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

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

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

```
<portType name="LoanService" sca:callback="foo:LoanServiceCallback">
  <operation name="apply">
    <input message="tns:ApplicationInput"/>
    <output message="tns:ApplicationOutput"/>
  </operation>
</portType>
```

### 8.5 WSDL Interface Type

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

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

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

The interface.wsdl element has the following **attributes**:

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

Formatted: Font: 10 pt, Font color: Custom Color(63,127,127))

Formatted: Font: 10 pt, Font color: Black

Formatted: Font: 10 pt, Font color: Custom Color(127,0,127))

Formatted: Font: 10 pt, Font color: Custom Color(42,0,255))

Formatted: Indent: Left: 0.63 cm, First line: 0.63 cm, Space Before: 4 pt, After: 0

Formatted: Font: 10 pt, Font color: Black

Formatted: Font: 10 pt, Bold, Font color: Custom Color(127,0,85))

Formatted ... [1]

Formatted ... [2]

Formatted ... [3]

Formatted ... [4]

Formatted ... [5]

Formatted ... [6]

Formatted ... [7]

Formatted ... [8]

Formatted ... [9]

Formatted ... [10]

Formatted ... [11]

Formatted ... [12]

Formatted ... [13]

Formatted ... [14]

Formatted ... [15]

Formatted ... [16]

Formatted ... [17]

Formatted ... [18]

Formatted ... [19]

Formatted ... [20]

Formatted ... [21]

Formatted ... [22]

Formatted ... [23]

Formatted: Heading 2,H2

Formatted ... [24]

Deleted: ¶

Formatted ... [25]

Formatted ... [26]

2752		<ul style="list-style-type: none"><li>• <b><i>callbackInterface(0..1)</i></b> - an optional callback interface, which is the URI of a WSDL Port Type</li></ul>	
2753			
2754		<a href="#">The form of the URI for WSDL port types follows the syntax described in the WSDL 1.1 Element Identifiers specification [WSDL11 Identifiers]</a>	Formatted: Normal
2755			Formatted: Heading 3,H3
2756		<b>8.5.1 Example of interface.wsdl</b>	
2757		<pre>&lt;interface:wsdl interface="http://www.stockquote.org/StockQuoteService# wsdl.porttype(StockQuote)" callbackInterface="http://www.stockquote.org/StockQuoteService# wsdl.porttype(StockQuoteCallback)"/&gt;</pre>	
2758			
2759			
2760			
2761			
2762		This declares an interface in terms of the WSDL port type "StockQuote" with a callback interface defined by the "StockQuoteCallback" port type.	Formatted: Space Before: 4 pt, After: 4 pt
2763			
2764			

---

## 9 Binding

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.

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

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.

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

  <service name="xs:NCName" promote="xs:anyURI"
    requires="list of xs:QName"? policySets="list of xs:QName"?>*
    <interface ... />?
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"? policySets="list of
xs:QName"?/>*
    <callback?
      <binding uri="xs:anyURI"? name="xs:NCName"?
        requires="list of xs:QName"?
        policySets="list of xs:QName"?/>+
    </callback>
  </service>
  ...

  <reference name="xs:NCName" target="list of xs:anyURI"?
    promote="list of xs:anyURI"? wiredByImpl="xs:boolean"?
    multiplicity="0..1 or 1..1 or 0..n or 1..n"?
  </reference>
```

```

requires="list of xs:QName"? policySets="list of xs:QName"?>*
<interface ... />?
<binding uri="xs:anyURI"? name="xs:NCName"?
    requires="list of xs:QName"? policySets="list of
xs:QName"?/>*
<callback>?
    <binding uri="xs:anyURI"? name="xs:NCName"?
        requires="list of xs:QName"?
        policySets="list of xs:QName"?/>+
</callback>
</reference>
...
</composite>

```

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

A binding element has the following attributes:

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

**Comment [ME14]:** This contradicts material below - is this the Issue 57 problem?

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

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

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

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. [ASM90003] To specify constraints on the kinds of bindings that are acceptable for use with a reference, the user specifies either policy intents or policy sets.

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

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

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

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

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

## 9.2 Form of the URI of a Deployed Binding

### 9.2.1 Constructing Hierarchical URIs

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

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

Each of these components deserves addition definition:

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

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

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

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

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

Base Domain URI for a scheme / Service Binding URI

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

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

http://acme.com/stocksComponent/getQuote

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

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

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

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

## 9.2.2 Non-hierarchical URIs

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

## 9.2.3 Determining the URI scheme of a deployed binding

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

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

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

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



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

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

## 9.3 SCA Binding

The SCA binding element is defined by the following schema.

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

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

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

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

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

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

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

### 9.3.1 Example SCA Binding

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

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Binding SCA example -->
```

```

2993 <composite      xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
2994               targetNamespace="http://foo.com"
2995               name="MyValueComposite" >
2996
2997     <service name="MyValueService" promote="MyValueComponent">
2998       <interface.java interface="services.myvalue.MyValueService"/>
2999       <binding.sca/>
3000       ...
3001     </service>
3002
3003     ...
3004
3005     <reference name="StockQuoteService"
3006     promote="MyValueComponent/StockQuoteReference">
3007       <interface.java
3008       interface="services.stockquote.StockQuoteService"/>
3009       <binding.sca/>
3010     </reference>
3011
3012 </composite>
3013

```

## 3014 9.4 Web Service Binding

3015 SCA defines a Web services binding. This is described in [a separate specification document \[9\]](#).

3016

## 3017 9.5 JMS Binding

3018 SCA defines a JMS binding. This is described in [a separate specification document \[11\]](#).

---

## 10 SCA Definitions

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

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

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

    <sca:intent/>*

    <sca:policySet/>*

    <sca:binding/>*

    <sca:bindingType/>*

    <sca:implementationType/>*

</definitions>
```

The definitions element has the following attribute:

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

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

---

## 11 Extension Model

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

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

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

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

### 11.1 Defining an Interface Type

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

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

  <element name="interface" type="sca:Interface" abstract="true"/>
  <complexType name="Interface"/>
  <complexType name="Interface" abstract="true">
    <attribute name="requires" type="sca:listOfQNames" use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
  </complexType>
```

3095  
3096  
3097  
3098

...

</schema>

3099 In the following snippet is an example of how the base definition is extended to support Java  
3100 interfaces. The snippet shows the definition of the **interface.java** element and the  
3101 **JavaInterface** type contained in **sca-interface-java.xsd**.

3102  
3103  
3104  
3105  
3106  
3107  
3108  
3109  
3110  
3111  
3112  
3113  
3114  
3115  
3116  
3117  
3118

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

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

3119 In the following snippet is an example of how the base definition can be extended by other  
3120 specifications to support a new interface not defined in the SCA specifications. The snippet shows  
3121 the definition of the **my-interface-extension** element and the **my-interface-extension-type**  
3122 type.

3123  
3124  
3125  
3126  
3127  
3128  
3129  
3130  
3131  
3132  
3133  
3134  
3135  
3136  
3137  
3138

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

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

3139       </schema>  
3140

## 3141   11.2 Defining an Implementation Type

3142   The following snippet shows the base definition for the ***implementation*** element and  
3143   ***Implementation*** type contained in ***sca-core.xsd***; see appendix for complete schema.  
3144

```
3145 <?xml version="1.0" encoding="UTF-8"?>  
3146 <!-- (c) Copyright SCA Collaboration 2006 -->  
3147 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3148       targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3149       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3150       elementFormDefault="qualified">  
3151  
3152     ...  
3153  
3154     <element name="implementation" type="sca:Implementation"  
3155     abstract="true"/>  
3156     <complexType name="Implementation"/>  
3157  
3158     ...  
3159  
3160 </schema>
```

3161  
3162   In the following snippet we show how the base definition is extended to support Java  
3163   implementation. The snippet shows the definition of the ***implementation.java*** element and the  
3164   ***JavaImplementation*** type contained in ***sca-implementation-java.xsd***.  
3165

```
3166 <?xml version="1.0" encoding="UTF-8"?>  
3167 <schema xmlns="http://www.w3.org/2001/XMLSchema"  
3168       targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"  
3169       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712">  
3170  
3171     <element name="implementation.java" type="sca:JavaImplementation"  
3172       substitutionGroup="sca:implementation"/>  
3173     <complexType name="JavaImplementation">  
3174       <complexContent>  
3175         <extension base="sca:Implementation">  
3176           <attribute name="class" type="NCName"  
3177           use="required"/>  
3178         </extension>  
3179       </complexContent>  
3180     </complexType>  
3181 </schema>
```

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

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

    <element name="my-impl-extension" type="tns:my-impl-extension-type"
            substitutionGroup="sca:implementation"/>
    <complexType name="my-impl-extension-type">
        <complexContent>
            <extension base="sca:Implementation">
                ...
            </extension>
        </complexContent>
    </complexType>
</schema>
```

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

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

The implementation type has the following attributes:

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

### 11.3 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.

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

```

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        elementFormDefault="qualified">
...
    <element name="binding" type="sca:Binding" abstract="true"/>
    <complexType name="Binding">
        <attribute name="uri" type="anyURI" use="optional"/>
        <attribute name="name" type="NCName" use="optional"/>
        <attribute name="requires" type="sca:listOfQNames"
            use="optional"/>
        <attribute name="policySets" type="sca:listOfQNames"
            use="optional"/>
    </complexType>
...
</schema>

```

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**.

```

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

    <element name="binding.ws" type="sca:WebServiceBinding"
        substitutionGroup="sca:binding"/>
    <complexType name="WebServiceBinding">
        <complexContent>
            <extension base="sca:Binding">
                <attribute name="port" type="anyURI" use="required"/>
            </extension>
        </complexContent>
    </complexType>
</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.

```

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

```



```

3270         targetNamespace="http://www.example.org/myextension"
3271         xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3272         xmlns:tns="http://www.example.org/myextension">
3273
3274     <element name="my-binding-extension"
3275         type="tns:my-binding-extension-type"
3276         substitutionGroup="sca:binding"/>
3277     <complexType name="my-binding-extension-type">
3278         <complexContent>
3279             <extension base="sca:Binding">
3280                 ...
3281             </extension>
3282         </complexContent>
3283     </complexType>
3284 </schema>
3285

```

3286 In addition to the definition for the new binding instance element, there needs to be an associated  
3287 bindingType element which provides metadata about the new binding type. The pseudo schema  
3288 for the bindingType element is shown in the following snippet:

```

3289 <bindingType type="xs:QName"
3290     alwaysProvides="list of intent QNames"?
3291     mayProvide = "list of intent QNames"?/>
3292

```

3293 The binding type has the following attributes:

- 3294 • **type (1..1)** – the type of the binding to which this bindingType element applies. This is  
3295 intended to be the QName of the binding element for the binding type, such as  
3296 "sca:binding.ws"
- 3297 • **alwaysProvides (0..1)** – a set of intents which the binding type always provides. See  
3298 [the Policy Framework specification \[10\]](#) for details.
- 3299 • **mayProvide (0..1)** – a set of intents which the binding type may provide. See [the](#)  
3300 [Policy Framework specification \[10\]](#) for details.

---

## 3301 12 Packaging and Deployment

### 3302 12.1 Domains

3303 An **SCA Domain** represents a complete runtime configuration, potentially distributed over a series  
3304 of interconnected runtime nodes.

3305 A single SCA domain defines the boundary of visibility for all SCA mechanisms. For example, SCA  
3306 wires can only be used to connect components within a single SCA domain. Connections to  
3307 services outside the domain must use binding specific mechanisms for addressing services (such  
3308 as WSDL endpoint URIs). Also, SCA mechanisms such as intents and policySets can only be used  
3309 in the context of a single domain. In general, external clients of a service that is developed and  
3310 deployed using SCA should not be able to tell that SCA was used to implement the service – it is  
3311 an implementation detail.

3312 The size and configuration of an SCA Domain is not constrained by the SCA Assembly specification  
3313 and is expected to be highly variable. An SCA Domain typically represents an area of business  
3314 functionality controlled by a single organization. For example, an SCA Domain may be the whole  
3315 of a business, or it may be a department within a business.

3316 As an example, for the accounts department in a business, the SCA Domain might cover all  
3317 finance-related functions, and it might contain a series of composites dealing with specific areas of  
3318 accounting, with one for Customer accounts and another dealing with Accounts Payable.

3319 An SCA domain has the following:

- 3320 • A virtual domain-level composite whose components are deployed and running
- 3321 • A set of *installed contributions* that contain implementations, interfaces and other artifacts  
3322 necessary to execute components
- 3323 • A set of logical services for manipulating the set of contributions and the virtual domain-  
3324 level composite.

3325 The information associated with an SCA domain can be stored in many ways, including but not  
3326 limited to a specific filesystem structure or a repository.

### 3327 12.2 Contributions

3328 An SCA domain might require a large number of different artifacts in order to work. These  
3329 artifacts include artifacts defined by SCA and other artifacts such as object code files and interface  
3330 definition files. The SCA-defined artifact types are all XML documents. The root elements of the  
3331 different SCA definition documents are: *composite*, *componentType*, *constrainingType* and  
3332 *definitions*. XML artifacts that are not defined by SCA but which may be needed by an SCA  
3333 domain include XML Schema documents, WSDL documents, and BPEL documents. SCA  
3334 constructs, like other XML-defined constructs, use XML qualified names for their identity (i.e.  
3335 namespace + local name).

3336 Non-XML artifacts are also required within an SCA domain. The most obvious examples of such  
3337 non-XML artifacts are Java, C++ and other programming language files necessary for component  
3338 implementations. Since SCA is extensible, other XML and non-XML artifacts may also be required.

3339 SCA defines an interoperable packaging format for contributions (ZIP), as specified below. This  
3340 format is not the only packaging format that an SCA runtime can use. SCA allows many different  
3341 packaging formats, but requires that the ZIP format be supported. When using the ZIP format for  
3342 deploying a contribution, this specification does not specify whether that format is retained after  
3343 deployment. For example, a Java EE based SCA runtime may convert the ZIP package to an EAR  
3344 package. SCA expects certain characteristics of any packaging:

- 3345 • For any contribution packaging it MUST be possible to present the artifacts of the  
3346 packaging to SCA as a hierarchy of resources based off of a single root [ASM12001]

3347       • Within any contribution packaging A directory resource SHOULD exist at the root of the  
3348       hierarchy named META-INF [\[ASM12002\]](#)

3349       • Within any contribution packaging a document SHOULD exist directly under the META-INF  
3350       directory named sca-contribution.xml which lists the SCA Composites within the  
3351       contribution that are runnable. [\[ASM12003\]](#)

3352       The same document also optionally lists namespaces of constructs that are defined within  
3353       the contribution and which may be used by other contributions  
3354       Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the  
3355       namespaces of constructs that are needed by the contribution and which are be found  
3356       elsewhere, for example in other contributions. [\[ASM12004\]](#) These optional elements may  
3357       not be physically present in the packaging, but may be generated based on the definitions  
3358       and references that are present, or they may not exist at all if there are no unresolved  
3359       references.

3360       See the section "SCA Contribution Metadata Document" for details of the format of this  
3361       file.

3364       To illustrate that a variety of packaging formats can be used with SCA, the following are examples  
3365       of formats that might be used to package SCA artifacts and metadata (as well as other artifacts)  
3366       as a contribution:

- 3367       • A filesystem directory
- 3368       • An OSGi bundle
- 3369       • A compressed directory (zip, gzip, etc)
- 3370       • A JAR file (or its variants – WAR, EAR, etc)

3371       Contributions do not contain other contributions. If the packaging format is a JAR file that  
3372       contains other JAR files (or any similar nesting of other technologies), the internal files are not  
3373       treated as separate SCA contributions. It is up to the implementation to determine whether the  
3374       internal JAR file should be represented as a single artifact in the contribution hierarchy or whether  
3375       all of the contents should be represented as separate artifacts.

3376       A goal of SCA's approach to deployment is that the contents of a contribution should not need to  
3377       be modified in order to install and use the contents of the contribution in a domain.

3378

### 3379   12.2.1 SCA Artifact Resolution

3380       Contributions may be self-contained, in that all of the artifacts necessary to run the contents of  
3381       the contribution are found within the contribution itself. However, it can also be the case that the  
3382       contents of the contribution make one or many references to artifacts that are not contained  
3383       within the contribution. These references can be to SCA artifacts or they can be to other artifacts  
3384       such as WSDL files, XSD files or to code artifacts such as Java class files and BPEL scripts.

3385       A contribution can use some artifact-related or packaging-related means to resolve artifact  
3386       references. Examples of such mechanisms include:

- 3387       • wsdlLocation and schemaLocation attributes in references to WSDL and XSD schema  
3388       artifacts respectively
- 3389       • OSGi bundle mechanisms for resolving Java class and related resource dependencies

3390       Where present, artifact-related or packaging-related mechanisms MUST be used to resolve artifact  
3391       dependencies. [\[ASM12005\]](#)

3392       SCA also provides an artifact resolution mechanism. The SCA artifact resolution mechanisms are  
3393       used either where no other mechanisms are available, or in cases where the mechanisms used by  
3394       the various contributions in the same SCA Domain are different. An example of the latter case is  
3395       where an OSGi Bundle is used for one contribution but where a second contribution used by the  
3396       first one is not implemented using OSGi - eg the second contribution is a mainframe COBOL

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

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

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

## 12.2.2 SCA Contribution Metadata Document

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

The format of the document is:

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

    <deployable composite="xs:QName"/>*
    <import namespace="xs:String" location="xs:AnyURI"?/>*
    <export namespace="xs:String"/>*

</contribution>
```

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

Attributes of the deployable element:

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

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

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

Attributes of the export element:

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

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

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

Attributes of the import element:

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

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

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

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

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

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

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

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

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

3498 dependent contributions should be specified only when deploying or updating contributions as  
3499 specified in the section 'Operations for Contributions' below.

### 3500 12.2.3 Contribution Packaging using ZIP

3501 SCA allows many different packaging formats that SCA runtimes can support, but SCA requires  
3502 that all runtimes MUST support the ZIP packaging format for contributions. [ASM12006] This  
3503 format allows that metadata specified by the section 'SCA Contribution Metadata Document' be  
3504 present. Specifically, it can contain a top-level "META-INF" directory and a "META-INF/sca-  
3505 contribution.xml" file and there can also be an optional "META-INF/sca-contribution-  
3506 generated.xml" file in the package. SCA defined artifacts as well as non-SCA defined artifacts such  
3507 as object files, WSDL definition, Java classes can be present anywhere in the ZIP archive,

3508 A up to date definition of the ZIP file format is published by PKWARE in [an Application Note on the](#)  
3509 [.ZIP file format \[12\]](#).

3510

## 3511 12.3 Installed Contribution

3512 As noted in the section above, the contents of a contribution do not need to be modified in order  
3513 to install and use it within a domain. An *installed contribution* is a contribution with all of the  
3514 associated information necessary in order to execute *deployable composites* within the  
3515 contribution.

3516 An installed contribution is made up of the following things:

- 3517 • Contribution Packaging – the contribution that will be used as the starting point for  
3518 resolving all references
- 3519 • Contribution base URI
- 3520 • Dependent contributions: a set of snapshots of other contributions that are used to resolve  
3521 the import statements from the root composite and from other dependent contributions
  - 3522 ○ Dependent contributions might or might not be shared with other installed  
3523 contributions.
  - 3524 ○ When the snapshot of any contribution is taken is implementation defined, ranging  
3525 from the time the contribution is installed to the time of execution
- 3526 • Deployment-time composites.  
3527 These are composites that are added into an installed contribution after it has been  
3528 deployed. This makes it possible to provide final configuration and access to  
3529 implementations within a contribution without having to modify the contribution. These  
3530 are optional, as composites that already exist within the contribution can also be used for  
3531 deployment.

3532

3533 Installed contributions provide a context in which to resolve qualified names (e.g. QNames in XML,  
3534 fully qualified class names in Java).

3535 If multiple dependent contributions have exported definitions with conflicting qualified names, the  
3536 algorithm used to determine the qualified name to use is implementation dependent.

3537 Implementations of SCA MAY also generate an error if there are conflicting names exported from  
3538 multiple contributions. [ASM12007]

3539

### 3540 12.3.1 Installed Artifact URIs

3541 When a contribution is installed, all artifacts within the contribution are assigned URIs, which are  
3542 constructed by starting with the base URI of the contribution and adding the relative URI of each  
3543 artifact (recalling that SCA requires that any packaging format be able to offer up its artifacts in a  
3544 single hierarchy).

3545

## 3546 12.4 Operations for Contributions

3547 SCA Domains provide the following conceptual functionality associated with contributions  
3548 (meaning the function might not be represented as addressable services and also meaning that  
3549 equivalent functionality might be provided in other ways). The functionality is optional meaning  
3550 that some SCA runtimes MAY choose not to provide the contribution functions functionality in any  
3551 way. [ASM12008]

### 3552 12.4.1 install Contribution & update Contribution

3553 Creates or updates an installed contribution with a supplied root contribution, and installed at a  
3554 supplied base URI. A supplied dependent contribution list (<export/> elements) specifies the  
3555 contributions that should be used to resolve the dependencies of the root contribution and other  
3556 dependent contributions. These override any dependent contributions explicitly listed via the  
3557 location attribute in the import statements of the contribution.

3558 SCA follows the simplifying assumption that the use of a contribution for resolving anything also  
3559 means that all other exported artifacts can be used from that contribution. Because of this, the  
3560 dependent contribution list is just a list of installed contribution URIs. There is no need to specify  
3561 what is being used from each one.

3562 Each dependent contribution is also an installed contribution, with its own dependent  
3563 contributions. By default these dependent contributions of the dependent contributions (which we  
3564 will call *indirect dependent contributions*) are included as dependent contributions of the installed  
3565 contribution. However, if a contribution in the dependent contribution list exports any conflicting  
3566 definitions with an indirect dependent contribution, then the indirect dependent contribution is not  
3567 included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions).  
3568 Also, if there is ever a conflict between two indirect dependent contributions, then the conflict  
3569 MUST be resolved by an explicit entry in the dependent contribution list. [ASM12009]

3570 Note that in many cases, the dependent contribution list can be generated. In particular, if the  
3571 creator of a domain is careful to avoid creating duplicate definitions for the same qualified name,  
3572 then it is easy for this list to be generated by tooling.

### 3573 12.4.2 add Deployment Composite & update Deployment Composite

3574 Adds or updates a deployment composite using a supplied composite ("composite by value" – a  
3575 data structure, not an existing resource in the domain) to the contribution identified by a supplied  
3576 contribution URI. The added or updated deployment composite is given a relative URI that  
3577 matches the @name attribute of the composite, with a ".composite" suffix. Since all composites  
3578 must run within the context of a installed contribution (any component implementations or other  
3579 definitions are resolved within that contribution), this functionality makes it possible for the  
3580 deployer to create a composite with final configuration and wiring decisions and add it to an  
3581 installed contribution without having to modify the contents of the root contribution.

3582 Also, in some use cases, a contribution might include only implementation code (e.g. PHP scripts).  
3583 It is then possible for those to be given component names by a (possibly generated) composite  
3584 that is added into the installed contribution, without having to modify the packaging.

### 3585 12.4.3 remove Contribution

3586 Removes the deployed contribution identified by a supplied contribution URI.

3587

## 3588 12.5 Use of Existing (non-SCA) Mechanisms for Resolving Artifacts

3589

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

Examples of these mechanisms include:

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

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

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

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

## 12.6 Domain-Level Composite

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

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

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

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

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

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

[ASM12013]

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

### 12.6.1 add To Domain-Level Composite

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

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

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



3637 supplied composite. All of the composite's components become *top-level* components and the  
3638 services become externally visible services (eg. they would be present in a WSDL description of  
3639 the domain).

## 3640 12.6.2 remove From Domain-Level Composite

3641 Removes from the Domain Level composite the elements corresponding to the composite  
3642 identified by a supplied composite URI. This means that the removal of the components, wires,  
3643 services and references originally added to the domain level composite by the identified  
3644 composite.

## 3645 12.6.3 get Domain-Level Composite

3646 Returns a <composite> definition that has an <include> line for each composite that had been  
3647 added to the domain level composite. It is important to note that, in dereferencing the included  
3648 composites, any referenced artifacts must be resolved in terms of that installed composite.

## 3649 12.6.4 get QName Definition

3650 In order to make sense of the domain-level composite (as returned by get Domain-Level  
3651 Composite), it must be possible to get the definitions for named artifacts in the included  
3652 composites. This functionality takes the supplied URI of an installed contribution (which provides  
3653 the context), a supplied qualified name of a definition to look up, and a supplied symbol space (as  
3654 a QName, eg wsdl:PortType). The result is a single definition, in whatever form is appropriate for  
3655 that definition type.

3656 Note that this, like all the other domain-level operations, is a conceptual operation. Its capabilities  
3657 should exist in some form, but not necessarily as a service operation with exactly this signature.

3658

---

## 13 Conformance

3659

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

3660

3661

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

3662

[ASM10001]

---

## A. Pseudo Schema

### A.1 ComponentType

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type schema snippet -->
<componentType xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
  constrainingType="QName"? >

  <service name="xs:NCName" requires="list of xs:QName"?
    policySets="list of xs:QName"?>*
    <interface ... />
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?/>*
    <callback?
      <binding ... />+
    </callback>
  </service>

  <reference name="xs:NCName"
    target="list of xs:anyURI"? autowire="xs:boolean"?
    multiplicity="0..1 or 1..1 or 0..n or 1..n"?
    wiredByImpl="xs:boolean"? requires="list of xs:QName"?
    policySets="list of xs:QName"?>*
    <interface ... />
    <binding uri="xs:anyURI"? name="xs:NCName"?
      requires="list of xs:QName"?
      policySets="list of xs:QName"?/>*
    <callback?
      <binding ... />+
    </callback>
  </reference>

  <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
    many="xs:boolean"? mustSupply="xs:boolean"?
    policySets="list of xs:QName"?>*
    default-property-value?
  </property>

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

3703  
3704     </componentType>  
3705

## 3706   A.2 Composite

```
3707     <?xml version="1.0" encoding="ASCII"?>
3708     <!-- Composite schema snippet -->
3709     <composite     xmlns="http://docs.oasis-open.org/ns/opencsa/sca/200712"
3710                    targetNamespace="xs:anyURI"
3711                    name="xs:NCName" local="xs:boolean"?
3712                    autowire="xs:boolean"? constrainingType="QName"?
3713                    requires="list of xs:QName"? policySets="list of
3714     xs:QName"?>
3715
3716         <include name="xs:QName"/>*
3717
3718         <service name="xs:NCName" promote="xs:anyURI"
3719                 requires="list of xs:QName"? policySets="list of xs:QName"?>*
3720             <interface ... />?
3721             <binding uri="xs:anyURI"? name="xs:NCName"?
3722                 requires="list of xs:QName"? policySets="list of
3723     xs:QName"? />*
3724             <callback>?
3725                 <binding uri="xs:anyURI"? name="xs:NCName"?
3726                     requires="list of xs:QName"?
3727                     policySets="list of xs:QName"? />+
3728             </callback>
3729         </service>
3730
3731         <reference name="xs:NCName" target="list of xs:anyURI"?
3732                 promote="list of xs:anyURI" wiredByImpl="xs:boolean"?
3733                 multiplicity="0..1 or 1..1 or 0..n or 1..n"?
3734                 requires="list of xs:QName"? policySets="list of xs:QName"?>*
3735             <interface ... />?
3736             <binding uri="xs:anyURI"? name="xs:NCName"?
3737                 requires="list of xs:QName"? policySets="list of
3738     xs:QName"? />*
3739             <callback>?
3740                 <binding uri="xs:anyURI"? name="xs:NCName"?
3741                     requires="list of xs:QName"?
3742                     policySets="list of xs:QName"? />+
3743             </callback>
3744         </reference>
3745
```

```

3746 <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3747     many="xs:boolean"? mustSupply="xs:boolean"?>*
3748     default-property-value?
3749 </property>
3750
3751 <component name="xs:NCName" autowire="xs:boolean"?
3752     requires="list of xs:QName"? policySets="list of xs:QName"?>*
3753     <implementation ... />?
3754     <service name="xs:NCName" requires="list of xs:QName"?
3755         policySets="list of xs:QName"?>*
3756         <interface ... />?
3757         <binding uri="xs:anyURI"? name="xs:NCName"?
3758             requires="list of xs:QName"?
3759             policySets="list of xs:QName"?/>*
3760         <callback>?
3761             <binding uri="xs:anyURI"? name="xs:NCName"?
3762                 requires="list of xs:QName"?
3763                 policySets="list of xs:QName"?/>+
3764         </callback>
3765     </service>
3766     <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
3767         source="xs:string"? file="xs:anyURI"? value="xs:string"?>*
3768         [<value>+ | xs:any+]?
3769     </property>
3770     <reference name="xs:NCName" target="list of xs:anyURI"?
3771         autowire="xs:boolean"? wiredByImpl="xs:boolean"?
3772         requires="list of xs:QName"? policySets="list of xs:QName"?
3773         multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
3774     <interface ... />?
3775     <binding uri="xs:anyURI"? name="xs:NCName"?
3776         requires="list of xs:QName"?
3777         policySets="list of xs:QName"?/>*
3778     <callback>?
3779         <binding uri="xs:anyURI"? name="xs:NCName"?
3780             requires="list of xs:QName"?
3781             policySets="list of xs:QName"?/>+
3782     </callback>
3783 </reference>
3784 </component>
3785
3786 <wire source="xs:anyURI" target="xs:anyURI" />*
3787
3788 </composite>

```

---

## B. XML Schemas

### B.1 sca.xsd

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

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

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

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

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

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

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

</schema>
```

### B.2 sca-core.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- (c) Copyright SCA Collaboration 2006, 2007 -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
        elementFormDefault="qualified">

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

    <complexType name="ComponentType">
```

```

3828     <sequence>
3829         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
3830     <choice minOccurs="0" maxOccurs="unbounded">
3831         <element name="service" type="sca:ComponentService" />
3832         <element name="reference" type="sca:ComponentReference" />
3833         <element name="property" type="sca:Property" />
3834     </choice>
3835     <any namespace="##other" processContents="lax" minOccurs="0"
3836         maxOccurs="unbounded" />
3837 </sequence>
3838 <attribute name="constrainingType" type="QName" use="optional"/>
3839 <anyAttribute namespace="##other" processContents="lax"/>
3840 </complexType>
3841
3842 <element name="composite" type="sca:Composite"/>
3843 <complexType name="Composite">
3844     <sequence>
3845         <element name="include" type="anyURI" minOccurs="0"
3846             maxOccurs="unbounded" />
3847         <choice minOccurs="0" maxOccurs="unbounded">
3848             <element name="service" type="sca:Service"/>
3849             <element name="property" type="sca:Property"/>
3850             <element name="component" type="sca:Component"/>
3851             <element name="reference" type="sca:Reference"/>
3852             <element name="wire" type="sca:Wire"/>
3853         </choice>
3854         <any namespace="##other" processContents="lax" minOccurs="0"
3855             maxOccurs="unbounded" />
3856     </sequence>
3857     <attribute name="name" type="NCName" use="required"/>
3858     <attribute name="targetNamespace" type="anyURI" use="required"/>
3859     <attribute name="local" type="boolean" use="optional"
3860         default="false"/>
3861     <attribute name="autowire" type="boolean" use="optional"
3862         default="false"/>
3863     <attribute name="constrainingType" type="QName" use="optional"/>
3864     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3865     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3866     <anyAttribute namespace="##other" processContents="lax"/>
3867 </complexType>
3868
3869 <complexType name="Service">
3870     <sequence>

```

```

3871         <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3872         <element name="operation" type="sca:Operation" minOccurs="0"
3873             maxOccurs="unbounded" />
3874         <element ref="sca:binding" minOccurs="0" maxOccurs="unbounded" />
3875         <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3876         <any namespace="##other" processContents="lax" minOccurs="0"
3877             maxOccurs="unbounded" />
3878     </sequence>
3879     <attribute name="name" type="NCName" use="required" />
3880     <attribute name="promote" type="anyURI" use="required" />
3881     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3882     <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3883     <anyAttribute namespace="##other" processContents="lax" />
3884 </complexType>
3885
3886 <element name="interface" type="sca:Interface" abstract="true" />
3887 <complexType name="Interface" abstract="true">
3888     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3889     <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3890 </complexType>
3891
3892 <complexType name="Reference">
3893     <sequence>
3894         <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
3895         <element name="operation" type="sca:Operation" minOccurs="0"
3896             maxOccurs="unbounded" />
3897         <element ref="sca:binding" minOccurs="0" maxOccurs="unbounded" />
3898         <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
3899         <any namespace="##other" processContents="lax" minOccurs="0"
3900             maxOccurs="unbounded" />
3901     </sequence>
3902     <attribute name="name" type="NCName" use="required" />
3903     <attribute name="target" type="sca:listOfAnyURIs" use="optional" />
3904     <attribute name="wiredByImpl" type="boolean" use="optional"
3905         default="false" />
3906     <attribute name="multiplicity" type="sca:Multiplicity"
3907         use="optional" default="1..1" />
3908     <attribute name="promote" type="sca:listOfAnyURIs" use="required" />
3909     <attribute name="requires" type="sca:listOfQNames" use="optional" />
3910     <attribute name="policySets" type="sca:listOfQNames" use="optional" />
3911     <anyAttribute namespace="##other" processContents="lax" />
3912 </complexType>
3913

```



```

3914 <complexType name="SCAPropertyBase" mixed="true">
3915   <!-- mixed="true" to handle simple type -->
3916   <sequence>
3917     <choice minOccurs="0">
3918       <element name="value" minOccurs="1" maxOccurs="unbounded"
3919         type="anyType"/>
3920       <any namespace="##any" processContents="lax" minOccurs="1"
3921         maxOccurs="unbounded" />
3922       <!-- NOT an extension point; This xsd:any exists
3923         to accept the element-based or complex type
3924         property i.e. no element-based extension point
3925         under "sca:property" -->
3926     </choice>
3927   </sequence>
3928 </complexType>
3929
3930 <!-- complex type for sca:property declaration -->
3931 <complexType name="Property" mixed="true">
3932   <complexContent>
3933     <extension base="sca:SCAPropertyBase">
3934       <!-- extension defines the place to hold default value -->
3935       <attribute name="name" type="NCName" use="required"/>
3936       <attribute name="value" type="xs:string" use="optional"/>
3937       <attribute name="type" type="QName" use="optional"/>
3938       <attribute name="element" type="QName" use="optional"/>
3939       <attribute name="many" type="boolean" default="false"
3940         use="optional"/>
3941       <attribute name="mustSupply" type="boolean" default="false"
3942         use="optional"/>
3943       <anyAttribute namespace="##other" processContents="lax"/>
3944       <!-- an extension point ; attribute-based only -->
3945     </extension>
3946   </complexContent>
3947 </complexType>
3948
3949 <complexType name="PropertyValue" mixed="true">
3950   <complexContent>
3951     <extension base="sca:SCAPropertyBase">
3952       <attribute name="name" type="NCName" use="required"/>
3953       <attribute name="value" type="xs:string" use="optional"/>
3954       <attribute name="type" type="QName" use="optional"/>
3955       <attribute name="element" type="QName" use="optional"/>
3956       <attribute name="many" type="boolean" default="false"

```

```

3957         use="optional"/>
3958     <attribute name="source" type="string" use="optional"/>
3959     <attribute name="file" type="anyURI" use="optional"/>
3960     <anyAttribute namespace="##other" processContents="lax"/>
3961     <!-- an extension point ; attribute-based only -->
3962 </extension>
3963 </complexContent>
3964 </complexType>
3965
3966 <element name="binding" type="sca:Binding" abstract="true"/>
3967 <complexType name="Binding" abstract="true">
3968     <sequence>
3969         <element name="operation" type="sca:Operation" minOccurs="0"
3970             maxOccurs="unbounded" />
3971     </sequence>
3972     <attribute name="uri" type="anyURI" use="optional"/>
3973     <attribute name="name" type="NCName" use="optional"/>
3974     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3975     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3976 </complexType>
3977
3978 <element name="bindingType" type="sca:BindingType"/>
3979 <complexType name="BindingType">
3980     <sequence minOccurs="0" maxOccurs="unbounded">
3981         <any namespace="##other" processContents="lax" />
3982     </sequence>
3983     <attribute name="type" type="QName" use="required"/>
3984     <attribute name="alwaysProvides" type="sca:listOfQNames"
3985         use="optional"/>
3986     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
3987     <anyAttribute namespace="##other" processContents="lax"/>
3988 </complexType>
3989
3990 <element name="callback" type="sca:Callback"/>
3991 <complexType name="Callback">
3992     <choice minOccurs="0" maxOccurs="unbounded">
3993         <element ref="sca:binding"/>
3994         <any namespace="##other" processContents="lax"/>
3995     </choice>
3996     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
3997     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
3998     <anyAttribute namespace="##other" processContents="lax"/>
3999 </complexType>

```

```

4000
4001 <complexType name="Component">
4002     <sequence>
4003         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
4004         <choice minOccurs="0" maxOccurs="unbounded">
4005             <element name="service" type="sca:ComponentService"/>
4006             <element name="reference" type="sca:ComponentReference"/>
4007             <element name="property" type="sca:PropertyValue" />
4008         </choice>
4009         <any namespace="##other" processContents="lax" minOccurs="0"
4010             maxOccurs="unbounded" />
4011     </sequence>
4012     <attribute name="name" type="NCName" use="required"/>
4013     <attribute name="autowire" type="boolean" use="optional" />
4014     <attribute name="constrainingType" type="QName" use="optional"/>
4015     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4016     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4017     <anyAttribute namespace="##other" processContents="lax"/>
4018 </complexType>
4019
4020 <complexType name="ComponentService">
4021     <complexContent>
4022         <restriction base="sca:Service">
4023             <sequence>
4024                 <element ref="sca:interface" minOccurs="0"
4025                     maxOccurs="1"/>
4026                 <element name="operation" type="sca:Operation"
4027                     minOccurs="0" maxOccurs="unbounded" />
4028                 <element ref="sca:binding" minOccurs="0"
4029                     maxOccurs="unbounded" />
4030                 <element ref="sca:callback" minOccurs="0"
4031                     maxOccurs="1"/>
4032                 <any namespace="##other" processContents="lax"
4033                     minOccurs="0" maxOccurs="unbounded" />
4034             </sequence>
4035             <attribute name="name" type="NCName" use="required"/>
4036             <attribute name="requires" type="sca:listOfQNames"
4037                 use="optional"/>
4038             <attribute name="policySets" type="sca:listOfQNames"
4039                 use="optional"/>
4040             <anyAttribute namespace="##other" processContents="lax"/>
4041         </restriction>
4042     </complexContent>

```

```

4043 </complexType>
4044
4045 <complexType name="ComponentReference">
4046   <complexContent>
4047     <restriction base="sca:Reference">
4048       <sequence>
4049         <element ref="sca:interface" minOccurs="0"
4050           maxOccurs="1" />
4051         <element name="operation" type="sca:Operation"
4052           minOccurs="0" maxOccurs="unbounded" />
4053         <element ref="sca:binding" minOccurs="0"
4054           maxOccurs="unbounded" />
4055         <element ref="sca:callback" minOccurs="0"
4056           maxOccurs="1" />
4057         <any namespace="##other" processContents="lax"
4058           minOccurs="0" maxOccurs="unbounded" />
4059       </sequence>
4060       <attribute name="name" type="NCName" use="required" />
4061       <attribute name="autowire" type="boolean" use="optional" />
4062       <attribute name="wiredByImpl" type="boolean" use="optional"
4063         default="false"/>
4064       <attribute name="target" type="sca:listOfAnyURIs"
4065         use="optional"/>
4066       <attribute name="multiplicity" type="sca:Multiplicity"
4067         use="optional" default="1..1" />
4068       <attribute name="requires" type="sca:listOfQNames"
4069         use="optional"/>
4070       <attribute name="policySets" type="sca:listOfQNames"
4071         use="optional"/>
4072       <anyAttribute namespace="##other" processContents="lax" />
4073     </restriction>
4074   </complexContent>
4075 </complexType>
4076
4077 <element name="implementation" type="sca:Implementation"
4078   abstract="true" />
4079 <complexType name="Implementation" abstract="true">
4080   <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4081   <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4082 </complexType>
4083
4084 <element name="implementationType" type="sca:ImplementationType"/>
4085 <complexType name="ImplementationType">

```

```

4086     <sequence minOccurs="0" maxOccurs="unbounded">
4087         <any namespace="##other" processContents="lax" />
4088     </sequence>
4089     <attribute name="type" type="QName" use="required"/>
4090     <attribute name="alwaysProvides" type="sca:listOfQNames"
4091 use="optional"/>
4092     <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
4093     <anyAttribute namespace="##other" processContents="lax"/>
4094 </complexType>
4095
4096 <complexType name="Wire">
4097     <sequence>
4098         <any namespace="##other" processContents="lax" minOccurs="0"
4099 maxOccurs="unbounded" />
4100     </sequence>
4101     <attribute name="source" type="anyURI" use="required"/>
4102     <attribute name="target" type="anyURI" use="required"/>
4103     <anyAttribute namespace="##other" processContents="lax"/>
4104 </complexType>
4105
4106 <element name="include" type="sca:Include"/>
4107 <complexType name="Include">
4108     <attribute name="name" type="QName"/>
4109     <anyAttribute namespace="##other" processContents="lax"/>
4110 </complexType>
4111
4112 <complexType name="Operation">
4113     <attribute name="name" type="NCName" use="required"/>
4114     <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4115     <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
4116     <anyAttribute namespace="##other" processContents="lax"/>
4117 </complexType>
4118
4119 <element name="constrainingType" type="sca:ConstrainingType"/>
4120 <complexType name="ConstrainingType">
4121     <sequence>
4122         <choice minOccurs="0" maxOccurs="unbounded">
4123             <element name="service" type="sca:ComponentService"/>
4124             <element name="reference" type="sca:ComponentReference"/>
4125             <element name="property" type="sca:Property" />
4126         </choice>
4127         <any namespace="##other" processContents="lax" minOccurs="0"
4128 maxOccurs="unbounded" />

```

```

4129         </sequence>
4130         <attribute name="name" type="NCName" use="required"/>
4131         <attribute name="targetNamespace" type="anyURI"/>
4132         <attribute name="requires" type="sca:listOfQNames" use="optional"/>
4133         <anyAttribute namespace="##other" processContents="lax"/>
4134     </complexType>
4135
4136
4137     <simpleType name="Multiplicity">
4138         <restriction base="string">
4139             <enumeration value="0..1"/>
4140             <enumeration value="1..1"/>
4141             <enumeration value="0..n"/>
4142             <enumeration value="1..n"/>
4143         </restriction>
4144     </simpleType>
4145
4146     <simpleType name="OverrideOptions">
4147         <restriction base="string">
4148             <enumeration value="no"/>
4149             <enumeration value="may"/>
4150             <enumeration value="must"/>
4151         </restriction>
4152     </simpleType>
4153
4154     <!-- Global attribute definition for @requires to permit use of intents
4155          within WSDL documents -->
4156     <attribute name="requires" type="sca:listOfQNames"/>
4157
4158     <!-- Global attribute definition for @endsConversation to mark operations
4159          as ending a conversation -->
4160     <attribute name="endsConversation" type="boolean" default="false"/>
4161
4162     <!-- Global attribute definition for @callback to mark a WSDL port type
4163          as having a callback interface defined in terms of a second port
4164          type. -->
4165     <attribute name="callback" type="anyUri" use="optional"/>
4166
4167     <simpleType name="listOfQNames">
4168         <list itemType="QName"/>
4169     </simpleType>
4170
4171     <simpleType name="listOfAnyURIs">

```

```

4172         <list itemType="anyURI"/>
4173     </simpleType>
4174
4175 </schema>

```

## 4176 B.3 sca-binding-sca.xsd

```

4177
4178 <?xml version="1.0" encoding="UTF-8"?>
4179 <!-- (c) Copyright SCA Collaboration 2006, 2007 -->
4180 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4181     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4182     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4183     elementFormDefault="qualified">
4184
4185     <include schemaLocation="sca-core.xsd"/>
4186
4187     <element name="binding.sca" type="sca:SCABinding"
4188         substitutionGroup="sca:binding"/>
4189     <complexType name="SCABinding">
4190         <complexContent>
4191             <extension base="sca:Binding">
4192                 <sequence>
4193                     <element name="operation" type="sca:Operation"
4194 minOccurs="0"
4195                             maxOccurs="unbounded" />
4196                 </sequence>
4197                 <attribute name="uri" type="anyURI" use="optional"/>
4198                 <attribute name="name" type="QName" use="optional"/>
4199                 <attribute name="requires" type="sca:listOfQNames"
4200                     use="optional"/>
4201                 <attribute name="policySets" type="sca:listOfQNames"
4202                     use="optional"/>
4203                 <anyAttribute namespace="##other" processContents="lax"/>
4204             </extension>
4205         </complexContent>
4206     </complexType>
4207 </schema>
4208

```

## 4209 B.4 sca-interface-java.xsd

```

4210
4211 <?xml version="1.0" encoding="UTF-8"?>
4212 <!-- (c) Copyright SCA Collaboration 2006 -->

```

```

4213 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4214       targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4215       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4216       elementFormDefault="qualified">
4217
4218   <include schemaLocation="sca-core.xsd"/>
4219
4220   <element name="interface.java" type="sca:JavaInterface"
4221         substitutionGroup="sca:interface"/>
4222   <complexType name="JavaInterface">
4223     <complexContent>
4224       <extension base="sca:Interface">
4225         <sequence>
4226           <any namespace="##other" processContents="lax"
4227 minOccurs="0"           maxOccurs="unbounded"/>
4228         </sequence>
4229         <attribute name="interface" type="NCName" use="required"/>
4230         <attribute name="callbackInterface" type="NCName"
4231 use="optional"/>
4232         <anyAttribute namespace="##other" processContents="lax"/>
4233       </extension>
4234     </complexContent>
4235   </complexType>
4236 </schema>
4237

```

## 4238 B.5 sca-interface-wsdl.xsd

```

4239
4240 <?xml version="1.0" encoding="UTF-8"?>
4241 <!-- (c) Copyright SCA Collaboration 2006 -->
4242 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4243       targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4244       xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4245       elementFormDefault="qualified">
4246
4247   <include schemaLocation="sca-core.xsd"/>
4248
4249   <element name="interface.wsdl" type="sca:WSDLPortType"
4250         substitutionGroup="sca:interface"/>
4251   <complexType name="WSDLPortType">
4252     <complexContent>
4253       <extension base="sca:Interface">
4254         <sequence>
4255           <any namespace="##other" processContents="lax"
4256 minOccurs="0"           maxOccurs="unbounded"/>

```



```

4257         </sequence>
4258         <attribute name="interface" type="anyURI" use="required"/>
4259         <attribute name="callbackInterface" type="anyURI"
4260 use="optional"/>
4261         <anyAttribute namespace="##other" processContents="lax"/>
4262     </extension>
4263 </complexContent>
4264 </complexType>
4265 </schema>
4266

```

## 4267 B.6 sca-implementation-java.xsd

```

4268
4269 <?xml version="1.0" encoding="UTF-8"?>
4270 <!-- (c) Copyright SCA Collaboration 2006 -->
4271 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4272     targetNamespace="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4273     xmlns:sca="http://docs.oasis-open.org/ns/opencsa/sca/200712"
4274     elementFormDefault="qualified">
4275
4276     <include schemaLocation="sca-core.xsd"/>
4277
4278     <element name="implementation.java" type="sca:JavaImplementation"
4279         substitutionGroup="sca:implementation"/>
4280     <complexType name="JavaImplementation">
4281         <complexContent>
4282             <extension base="sca:Implementation">
4283                 <sequence>
4284                     <any namespace="##other" processContents="lax"
4285                         minOccurs="0" maxOccurs="unbounded"/>
4286                 </sequence>
4287                 <attribute name="class" type="NCName" use="required"/>
4288                 <attribute name="requires" type="sca:listOfQNames"
4289 use="optional"/>
4290                 <attribute name="policySets" type="sca:listOfQNames"
4291                     use="optional"/>
4292                 <anyAttribute namespace="##other" processContents="lax"/>
4293             </extension>
4294         </complexContent>
4295     </complexType>
4296 </schema>

```

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

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

  <include schemaLocation="sca-core.xsd"/>
  <element name="implementation.composite" type="sca:SCAImplementation"
    substitutionGroup="sca:implementation"/>
  <complexType name="SCAImplementation">
    <complexContent>
      <extension base="sca:Implementation">
        <sequence>
          <any namespace="##other" processContents="lax"
minOccurs="0"
            maxOccurs="unbounded"/>
        </sequence>
        <attribute name="name" type="QName" use="required"/>
        <attribute name="requires" type="sca:listOfQNames"
use="optional"/>
        <attribute name="policySets" type="sca:listOfQNames"
            use="optional"/>
        <anyAttribute namespace="##other" processContents="lax"/>
      </extension>
    </complexContent>
  </complexType>
</schema>
```

## B.8 sca-definitions.xsd

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

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

```

4339     <element name="definitions">
4340         <complexType>
4341             <choice minOccurs="0" maxOccurs="unbounded">
4342                 <element ref="sca:intent"/>
4343                 <element ref="sca:policySet"/>
4344                 <element ref="sca:binding"/>
4345                 <element ref="sca:bindingType"/>
4346                 <element ref="sca:implementationType"/>
4347                 <any namespace="##other" processContents="lax" minOccurs="0"
4348                     maxOccurs="unbounded" />
4349             </choice>
4350         </complexType>
4351     </element>
4352 </schema>
4353
4354

```

## 4355 B.9 sca-binding-webservice.xsd

4356 Is described in [the SCA Web Services Binding specification \[9\]](#)

## 4357 B.10 sca-binding-jms.xsd

4358 Is described in [the SCA JMS Binding specification \[11\]](#)

## 4359 B.11 sca-policy.xsd

4360 Is described in [the SCA Policy Framework specification \[10\]](#)

4361

## 4362 B.12 sca-contribution.xsd

4363

```

4364 <?xml version="1.0" encoding="UTF-8"?>
4365 <!-- (c) Copyright SCA Collaboration 2007 -->
4366 <schema xmlns="http://www.w3.org/2001/XMLSchema"
4367     targetNamespace="http://www.osoa.org/xmlns/sca/1.0"
4368     xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
4369     elementFormDefault="qualified">
4370
4371     <include schemaLocation="sca-core.xsd"/>
4372
4373
4374     <element name="contribution" type="sca:ContributionType"/>
4375     <complexType name="ContributionType">
4376         <sequence>
4377             <element name="deployable" type="sca:DeployableType"
4378 minOccurs="1" maxOccurs="unbounded"/>
4379             <element name="import" type="sca:ImportType" minOccurs="0"
4380 maxOccurs="unbounded"/>
4381             <element name="export" type="sca:ExportType" minOccurs="0"
4382 maxOccurs="unbounded"/>

```

```

4383         <any namespace="##other" processContents="lax" minOccurs="0"
4384 maxOccurs="unbounded" />
4385     </sequence>
4386     <anyAttribute namespace="##other" processContents="lax" />
4387 </complexType>
4388
4389
4390
4391     <complexType name="DeployableType">
4392     <sequence>
4393         <any namespace="##other" processContents="lax" minOccurs="0"
4394 maxOccurs="unbounded" />
4395     </sequence>
4396     <attribute name="composite" type="QName" use="required" />
4397     <anyAttribute namespace="##other" processContents="lax" />
4398 </complexType>
4399
4400
4401     <complexType name="ImportType">
4402     <sequence>
4403         <any namespace="##other" processContents="lax" minOccurs="0"
4404 maxOccurs="unbounded" />
4405     </sequence>
4406     <attribute name="namespace" type="string" use="required" />
4407     <attribute name="location" type="anyURI" use="required" />
4408     <anyAttribute namespace="##other" processContents="lax" />
4409 </complexType>
4410
4411     <complexType name="ExportType">
4412     <sequence>
4413         <any namespace="##other" processContents="lax" minOccurs="0"
4414 maxOccurs="unbounded" />
4415     </sequence>
4416     <attribute name="namespace" type="string" use="required" />
4417     <anyAttribute namespace="##other" processContents="lax" />
4418 </complexType>
4419 </schema>
4420
4421

```

---

## 4422 C. SCA Concepts

### 4423 C.1 Binding

4424 **Bindings** are used by services and references. References use bindings to describe the access  
4425 mechanism used to call the service to which they are wired. Services use bindings to describe the  
4426 access mechanism(s) that clients should use to call the service.

4427 SCA supports multiple different types of bindings. Examples include **SCA service**, **Web service**,  
4428 **stateless session EJB**, **data base stored procedure**, **EIS service**. SCA provides an extensibility  
4429 mechanism by which an SCA runtime can add support for additional binding types.

4430

### 4431 C.2 Component

4432 **SCA components** are configured instances of **SCA implementations**, which provide and consume  
4433 services. SCA allows many different implementation technologies such as Java, BPEL, C++. SCA defines  
4434 an **extensibility mechanism** that allows you to introduce new implementation types. The current  
4435 specification does not mandate the implementation technologies to be supported by an SCA run-time,  
4436 vendors may choose to support the ones that are important for them. A single SCA implementation may  
4437 be used by multiple Components, each with a different configuration.

4438 The Component has a reference to an implementation of which it is an instance, a set of property values,  
4439 and a set of service reference values. Property values define the values of the properties of the  
4440 component as defined by the component's implementation. Reference values define the services that  
4441 resolve the references of the component as defined by its implementation. These values can either be a  
4442 particular service of a particular component, or a reference of the containing composite.

### 4443 C.3 Service

4444 **SCA services** are used to declare the externally accessible services of an **implementation**. For a  
4445 composite, a service is typically provided by a service of a component within the composite, or by a  
4446 reference defined by the composite. The latter case allows the republication of a service with a new  
4447 address and/or new bindings. The service can be thought of as a point at which messages from external  
4448 clients enter a composite or implementation.

4449 A service represents an addressable set of operations of an implementation that are designed to be  
4450 exposed for use by other implementations or exposed publicly for use elsewhere (eg public Web services  
4451 for use by other organizations). The operations provided by a service are specified by an Interface, as  
4452 are the operations required by the service client (if there is one). An implementation may contain  
4453 multiple services, when it is possible to address the services of the implementation separately.

4454 A service may be provided **as SCA remote services**, **as Web services**, **as stateless session EJB's**, **as**  
4455 **EIS services**, **and so on**. Services use **bindings** to describe the way in which they are published. SCA  
4456 provides an **extensibility mechanism** that makes it possible to introduce new binding types for new  
4457 types of services.

#### 4458 C.3.1 Remotable Service

4459 A Remotable Service is a service that is designed to be published remotely in a loosely-coupled SOA  
4460 architecture. For example, SCA services of SCA implementations can define implementations of industry-  
4461 standard web services. Remotable services use pass-by-value semantics for parameters and returned  
4462 results.

4463 A service is remotable if it is defined by a WSDL port type or if it defined by a Java interface marked with  
4464 the @Remotable annotation.

### 4465 C.3.2 Local Service

4466 Local services are services that are designed to be only used “locally” by other implementations that are  
4467 deployed concurrently in a tightly-coupled architecture within the same operating system process.  
4468 Local services may rely on by-reference calling conventions, or may assume a very fine-grained  
4469 interaction style that is incompatible with remote distribution. They may also use technology-specific data-  
4470 types.  
4471 Currently a service is local only if it defined by a Java interface not marked with the @Remotable  
4472 annotation.  
4473

### 4474 C.4 Reference

4475 **SCA references** represent a dependency that an implementation has on a service that is supplied by  
4476 some other implementation, where the service to be used is specified through configuration. In other  
4477 words, a reference is a service that an implementation may call during the execution of its business  
4478 function. References are typed by an interface.  
4479 For composites, composite references can be accessed by components within the composite like any  
4480 service provided by a component within the composite. Composite references can be used as the targets  
4481 of wires from component references when configuring Components.  
4482 A composite reference can be used to access a service such as: an SCA service provided by another  
4483 SCA composite, a Web service, a stateless session EJB, a data base stored procedure or an EIS service,  
4484 and so on. References use **bindings** to describe the access method used to their services. SCA provides  
4485 an **extensibility mechanism** that allows the introduction of new binding types to references.  
4486

### 4487 C.5 Implementation

4488 An implementation is concept that is used to describe a piece of software technology such as a Java  
4489 class, BPEL process, XSLT transform, or C++ class that is used to implement one or more services in a  
4490 service-oriented application. An SCA composite is also an implementation.  
4491 Implementations define points of variability including properties that can be set and settable references to  
4492 other services. The points of variability are configured by a component that uses the implementation. The  
4493 specification refers to the configurable aspects of an implementation as its **componentType**.

### 4494 C.6 Interface

4495 **Interfaces** define one or more business functions. These business functions are provided by Services  
4496 and are used by components through References. Services are defined by the Interface they implement.  
4497 SCA currently supports a number of interface type systems, for example:

- 4498 • Java interfaces
- 4499 • WSDL portTypes
- 4500 • C, C++ header files

4501  
4502 SCA also provides an extensibility mechanism by which an SCA runtime can add support for additional  
4503 interface type systems.  
4504 Interfaces may be **bi-directional**. A bi-directional service has service operations which must be provided  
4505 by each end of a service communication – this could be the case where a particular service requires a  
4506 “callback” interface on the client, which is calls during the process of handing service requests from the  
4507 client.  
4508

## 4509 C.7 Composite

4510 An SCA composite is the basic unit of composition within an SCA Domain. An **SCA Composite** is an  
4511 assembly of Components, Services, References, and the Wires that interconnect them. Composites can  
4512 be used to contribute elements to an **SCA Domain**.

4513 A **composite** has the following characteristics:

- 4514 • It may be used as a component implementation. When used in this way, it defines a boundary for  
4515 Component visibility. Components may not be directly referenced from outside of the composite  
4516 in which they are declared.
- 4517 • It can be used to define a unit of deployment. Composites are used to contribute business logic  
4518 artifacts to an SCA domain.

4519

## 4520 C.8 Composite inclusion

4521 One composite can be used to provide part of the definition of another composite, through the process of  
4522 inclusion. This is intended to make team development of large composites easier. Included composites  
4523 are merged together into the using composite at deployment time to form a single logical composite.

4524 Composites are included into other composites through `<include.../>` elements in the using composite.  
4525 The SCA Domain uses composites in a similar way, through the deployment of composite files to a  
4526 specific location.

4527

## 4528 C.9 Property

4529 **Properties** allow for the configuration of an implementation with externally set data values. The data  
4530 value is provided through a Component, possibly sourced from the property of a containing composite.

4531 Each Property is defined by the implementation. Properties may be defined directly through the  
4532 implementation language or through annotations of implementations, where the implementation language  
4533 permits, or through a componentType file. A Property can be either a simple data type or a complex data  
4534 type. For complex data types, XML schema is the preferred technology for defining the data types.

4535

## 4536 C.10 Domain

4537 An SCA Domain represents a set of Services providing an area of Business functionality that is controlled  
4538 by a single organization. As an example, for the accounts department in a business, the SCA Domain  
4539 might cover all finance-related functions, and it might contain a series of composites dealing with specific  
4540 areas of accounting, with one for Customer accounts, another dealing with Accounts Payable.

4541 A domain specifies the instantiation, configuration and connection of a set of components, provided via  
4542 one or more composite files. The domain, like a composite, also has Services and References. Domains  
4543 also contain Wires which connect together the Components, Services and References.

4544

## 4545 C.11 Wire

4546 **SCA wires** connect **service references** to **services**.

4547 Within a composite, valid wire sources are component references and composite services. Valid wire  
4548 targets are component services and composite references.

4549 When using included composites, the sources and targets of the wires don't have to be declared in the  
4550 same composite as the composite that contains the wire. The sources and targets can be defined by  
4551 other included composites. Targets can also be external to the SCA domain.

4552

4553  
4554  
4555

## D. Conformance Items

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

Conformance ID	Description
[ASM10001]	An SCA runtime MUST reject a composite file that does not conform to the sca-core.xsd schema.
[ASM40002]	If present, the @constrainingType attribute of a <componentType/> element MUST reference a <constrainingType/> element in the Domain through its QName.
[ASM40003]	The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>.
[ASM40004]	The @name attribute of a <reference/> child element of a <componentType/> MUST be unique amongst the reference elements of that <componentType/>.
[ASM40005]	The @name attribute of a <property/> child element of a <componentType/> MUST be unique amongst the property elements of that <componentType/>.
[ASM40006]	If @wiredByImpl is set to "true", then any reference targets configured for this reference MUST be ignored by the runtime.
[ASM40007]	The value of the property @type attribute MUST be the QName of an XML schema type.
[ASM40008]	The value of the property @element attribute MUST be the QName of an XSD global element.
[ASM40009]	The SCA runtime MUST ensure that any implementation default property value is replaced by a value for that property explicitly set by a component using that implementation.

Deleted: [ASM40003]

Deleted: [ASM40004]

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

The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/>



component elements of that <composite/> [ASM50001]

[ASM50002]

The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>

[ASM50003]

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

[ASM50004]

If a <service/> element has an interface subelement specified, the interface MUST provide a compatible subset of the interface declared on the componentType of the implementation

[ASM50005]

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

[ASM50006]

If the callback element is present and contains one or more binding child elements, then those bindings MUST be used for the callback.

[ASM50007]

The @name attribute of a service element of a <component/> MUST be unique amongst the service elements of that <component/>

[ASM50008]

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

[ASM50009]

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

[ASM50010]

If @wiredByImpl="true" is set for a reference, then the reference MUST NOT be wired statically within a composite, but left unwired.

[ASM50011]

If an interface is declared for a component reference it MUST provide a compatible superset of the interface declared for the equivalent reference in the componentType of the implementation, i.e. provide the same operations or a superset of the operations defined by the implementation for the reference.

[ASM50012]

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

[ASM50013]

If @wiredByImpl="true", other methods of specifying the target service MUST NOT be used.

[ASM50014]

If @autowire="true", the autowire procedure MUST only be used if no

Deleted: The @name attribute of a <service/> child element of a <componentType/> MUST be unique amongst the service elements of that <componentType/>. The @name attribute of a <component/> child element of a <composite/> MUST be unique amongst the component elements of that <composite/> [ASM50001]

Deleted: [ASM50004]

Deleted: [ASM50005]

Deleted: [ASM50006]

Deleted: [ASM50011]

Deleted: [ASM50012]

target is identified by any of the other ways listed above. It is not an error if @autowire="true" and a target is also defined through some other means, however in this case the autowire procedure **MUST NOT** be used.

- [ASM50015]** If a binding element has a value specified for a target service using its @uri attribute, the binding element **MUST NOT** identify target services using binding specific attributes or elements.
- [ASM50016]** It is possible that a particular binding type **MAY** require that the address of a target service uses more than a simple URI. In such cases, the @uri attribute **MUST NOT** be used to identify the target service - instead, binding specific attributes and/or child elements must be used.
- [ASM50018]** A reference with multiplicity 0..1 or 0..n **MAY** have no target service defined.
- [ASM50019]** A reference with multiplicity 0..1 or 1..1 **MUST NOT** have more than one target service defined.
- [ASM50020]** A reference with multiplicity 1..1 or 1..n **MUST** have at least one target service defined.
- [ASM50021]** A reference with multiplicity 0..n or 1..n **MAY** have one or more target services defined.
- [ASM50022]** Where it is detected that the rules for the number of target services for a reference have been violated, either at deployment or at execution time, an SCA Runtime **MUST** generate an error no later than when the reference is invoked by the component implementation.
- [ASM50023]** Some reference multiplicity errors can be detected at deployment time. In these cases, an error **SHOULD** be generated by the SCA runtime at deployment time.
- [ASM50024]** Other reference multiplicity errors can only be checked at runtime. In these cases, the SCA runtime **MUST** generate an error no later than when the reference is invoked by the component implementation.
- [ASM50025]** Where a component reference is promoted by a composite reference, the promotion **MUST** be treated from a multiplicity perspective as providing 0 or more target services for the component reference, depending upon the further configuration of the composite reference. These target services are in addition to any target services identified on the component reference itself, subject to the rules relating to multiplicity.
- [ASM50026]** If a reference has a value specified for one or more target services in its @target attribute, there **MUST NOT** be any child <binding/> elements declared for that reference.
- [ASM50027]** If the @value attribute of a component property element is declared, the type of the property **MUST** be an XML Schema simple type and the @value attribute **MUST** contain a single value of that type.
- [ASM50028]** If the value subelement of a component property is specified, the type of the property **MUST** be an XML Schema simple type or an XML schema complex type.
- [ASM50029]** If a component property value is declared using a child element of the

<property/> element, the type of the property MUST be an XML Schema global element and the declared child element MUST be an instance of that global element.

[ASM50030]

A <component/> element MUST NOT contain two <property/> subelements with the same value of the @name attribute.

[ASM50031]

The name attribute of a component property MUST match the name of a property element in the component type of the component implementation.

[ASM50032]

If a property is single-valued, the <value/> subelement MUST NOT occur more than once.

[ASM50033]

A property <value/> subelement MUST NOT be used when the @value attribute is used to specify the value for that property.

[ASM60001]

A composite name must be unique within the namespace of the composite.

[ASM60002]

@local="true" for a composite means that all the components within the composite MUST run in the same operating system process.

[ASM60003]

The name of a composite <service/> element MUST be unique across all the composite services in the composite.

[ASM60004]

A composite <service/> element's promote attribute MUST identify one of the component services within that composite.

[ASM60005]

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

Deleted: [ASM60005]

[ASM60006]

The name of a composite <reference/> element MUST be unique across all the composite references in the composite.

[ASM60007]

Each of the URIs declared by a composite reference's @promote attribute MUST identify a component reference within the composite.

[ASM60008]

the interfaces of the component references promoted by a composite reference MUST be the same, or if the composite reference itself declares an interface then all the component reference interfaces must be compatible with the composite reference interface. Compatible means that the component reference interface is the same or is a strict subset of the composite reference interface.

[ASM60009]

the intents declared on a composite reference and on the component references which it promotes MUST NOT be mutually exclusive.

[ASM60010]

If any intents in the set which apply to a composite reference are mutually exclusive then the SCA runtime MUST 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 *interface* specified, it MUST provide an interface which is the same or which is a compatible superset of the interface(s) declared by the promoted component reference(s), i.e. provide a superset of the operations in the interface defined by the component for the reference.

Deleted: [ASM60012]

[ASM60013]

If no interface is declared on a composite reference, the interface from one of its promoted component references is used, which MUST be the same as or a compatible superset of the interface(s) declared by the promoted component reference(s).

[ASM60014]

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

[ASM60015]

the source interface and the target interface of a wire MUST either both be remotable or else both be local

[ASM60016]

the operations on the target interface of a wire MUST be the same as or be a superset of the operations in the interface specified on the source

[ASM60017]

compatibility between the source interface and the target interface for a wire for the individual operations is defined as compatibility of the signature, that is operation name, input types, and output types MUST be the same.

[ASM60018]

the order of the input and output types for operations in the source interface and the target interface of a wire also MUST be the same.

[ASM60019]

the set of Faults and Exceptions expected by each operation in the source interface MUST be the same or be a superset of those specified by the target interface.

[ASM60020]

other specified attributes of the source interface and the target interface of a wire MUST match, including Scope and Callback interface

[ASM60021]

For the case of an un-wired reference with multiplicity 1..1 or 1..n the deployment process provided by an SCA runtime SHOULD issue a warning.

[ASM60022]

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

Deleted: [ASM60022]

[ASM60023]

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

[ASM60024]

the intents, and policies applied to the service MUST be compatible with those on the reference when using autowire to wire a reference – so that wiring the reference to the service will not cause an error due to policy mismatch

Comment [mbgl24]: Issue 57

[ASM60025]

for an autowire reference with multiplicity 0..1 or 1..1, the SCA runtime MUST wire the reference to one of the set of valid target services chosen from the set in a runtime-dependent fashion

[ASM60026]

for an autowire reference with multiplicity 0..n or 1..n, the reference

MUST be wired to all of the set of valid target services

[ASM60027]

for an autowire reference with multiplicity 0..1 or 0..n, if the SCA runtime finds no valid target service, there is no problem – no services are wired and the SCA runtime MUST NOT raise an error

[ASM60028]

for an autowire reference with multiplicity 1..1 or 1..n, if the SCA runtime finds no valid target services an error MUST be raised by the SCA runtime since the reference is intended to be wired

[ASM60030]

The @name attribute of an <implementation.composite/> element MUST contain the QName of a composite in the SCA Domain.

[ASM60031]

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

[ASM70001]

The constrainingType specifies the services, references and properties that MUST be implemented by the implementation of the component to which the constrainingType is attached.

[ASM70002]

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

[ASM70003]

The name attribute of the constraining type MUST be unique in the SCA domain.

[ASM70004]

When an implementation is constrained by a constrainingType its component type MUST contain all the services, references and properties specified in the constrainingType.

[ASM70005]

An implementation MAY contain additional services, additional optional references (multiplicity 0..1 or 0..n) and additional optional properties beyond those declared in the constraining type, but MUST NOT contain additional non-optional references (multiplicity 1..1 or 1..n) or additional non-optional properties (a property with mustSupply=true).

[ASM70006]

Additional services, references and properties provided by the implementation which are not declared in the constrainingType associated with a component MUST NOT be configured in any way by the containing composite.

Deleted: [ASM70006]

[ASM70007]

A component or implementation can use a qualified form of an intent specified in unqualified form in the constrainingType, but if the constrainingType uses the qualified form of an intent, then the component or implementation MUST also use the qualified form, otherwise there is an error.

Deleted: [ASM70007]

[ASM80001]

The interface.wsdl @interface attribute MUST reference a portType of a WSDL 1.0 document OR an interface element of a WSDL 2.0 document.

[ASM80002]

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

[ASM80003]

If a remotable service is called locally or remotely, the SCA container MUST ensure sure that no modification of input messages by the service or post-invocation modifications to return messages are seen by the caller.

[ASM80004]

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

[ASM80005]

Either both interfaces of a bidirectional service MUST be remotable, or both MUST be local. A bidirectional service MUST NOT mix local and remote services.

[ASM80006]

Where a service or a reference has a conversational interface, the conversational intent MUST be attached either to the interface itself, or to the service or reference using the interface.

[ASM80007]

Once an operation marked with endsConversation has been invoked, any subsequent attempts to call an operation or a callback operation associated with the same conversation MUST generate a sca:ConversationViolation fault.

[ASM80008]

Any service or reference that uses an interface marked with required intents MUST implicitly add those intents to its own @requires list.

[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.

Formatted: Font color: Red

[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.

Formatted: Font color: Red

Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface. [ASM80012]

Where a component uses an implementation and the component configuration explicitly declares an interface for a service or a reference, if the matching service or reference declaration in the component type declares an interface which has a callback interface, then the component interface declaration MUST also declare a compatible interface with a compatible callback interface.

Formatted: Font color: Red

[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.

Formatted: Font color: Red

[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.

Formatted: Font color: Red

Formatted: Font color: Red

[\[ASM80015\]](#) 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.

**[ASM90001]** For a binding of a *reference* the URI attribute defines the target URI of the reference. This MUST be either the componentName/serviceName for a wire to an endpoint within the SCA domain, or the accessible address of some service endpoint either inside or outside the SCA domain (where the addressing scheme is defined by the type of the binding).

**[ASM90002]** When a service or reference has multiple bindings, only one binding can have the default name value; all others must have a name value specified that is unique within the service or reference.

**[ASM90003]** If a reference has any bindings they MUST be resolved which means that each binding MUST include a value for the @URI attribute or MUST otherwise specify an endpoint. The reference MUST NOT be wired using other SCA mechanisms.

Comment [mbgl25]: Issue 57

[\[ASM90004\]](#) a wire target MAY be specified with a syntax of "componentName/serviceName/bindingName".

Deleted: [ASM90004]

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

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

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

**[ASM12004]** Optionally, in the sca-contribution.xml file, additional elements MAY exist that list the namespaces of constructs that are needed by the contribution and which are be found elsewhere, for example in other contributions.

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

[\[ASM12006\]](#) SCA requires that all runtimes MUST support the ZIP packaging format for contributions.

Deleted: [ASM12006]

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

**[ASM12008]** SCA runtimes MAY choose not to provide the contribution functions functionality in any way.

**[ASM12009]** if there is ever a conflict between two indirect dependent contributions, then the conflict MUST be resolved by an explicit entry in the dependent contribution list.

[\[ASM12010\]](#) Where present, non-SCA artifact resolution mechanisms MUST be used by the SCA runtime in precedence to the SCA mechanisms.

Deleted: [ASM12010]

[\[ASM12011\]](#) If one of the non-SCA artifact resolution mechanisms is present, but there is a failure to find the resource indicated when using the

Deleted: [ASM12011]

mechanism (eg the URI is incorrect or invalid, say) the SCA runtime **MUST** raise an error and **MUST NOT** attempt to use SCA resolution mechanisms as an alternative.

[ASM12012]

The value of @autowire for the logical domain composite **MUST** be autowire="false".

Comment [mbgl26]: Issue 42

[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:

Comment [mbgl27]: Issue 40

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



4557  
4558  
4559  
4560  
4561  
4562  
4563

---

## E. Acknowledgements

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

**Participants:**

- [Participant Name, Affiliation | Individual Member]
- [Participant Name, Affiliation | Individual Member]

---

## **F. Non-Normative Text**

4565 **G. Revision History**

4566 [optional; should not be included in OASIS Standards]

4567

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</li> </ul> <p>"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."</p>
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 Combellack's review contained in email: <a href="http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html">http://lists.oasis-open.org/archives/sca-assembly/200804/msg00089.html</a></p>
7 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>

4568

Page 75: [1] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Font: 10 pt, Font color: Custom Color(RGB(42,0,255))

Page 75: [2] Formatted	Mike Edwards	12/11/2008 12:07:00 PM
------------------------	--------------	------------------------

Font: 10 pt, Font color: Black

Page 75: [3] Formatted	Mike Edwards	12/11/2008 12:07:00 PM
------------------------	--------------	------------------------

Font: Courier New, 10 pt, Font color: Black

Page 75: [4] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [5] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [6] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [7] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [8] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [9] Formatted	Mike Edwards	12/11/2008 12:08:00 PM
------------------------	--------------	------------------------

Body Text Char2,Body Text Char Char1,Body Text Char1 Char1 Char1,Body Text Char Char Char1 Char1,Body Text Char1 Char1 Char Char Char1,Body Text Char Char Char1 Char Char Char1,Body Text Char1 Char1 Char Char Char Char Char1,Body Text Char1 Char, Font: Ar

Page 75: [10] Formatted	Mike Edwards	12/11/2008 12:09:00 PM
-------------------------	--------------	------------------------

Font: 10 pt, Font color: Custom Color(RGB(42,0,255))

Page 75: [11] Formatted	Mike Edwards	12/11/2008 12:09:00 PM
-------------------------	--------------	------------------------

Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))

Page 75: [12] Formatted	Mike Edwards	12/11/2008 12:09:00 PM
-------------------------	--------------	------------------------

Font: 10 pt, Font color: Custom Color(RGB(42,0,255))

Page 75: [13] Formatted	Mike Edwards	12/11/2008 12:09:00 PM
-------------------------	--------------	------------------------

Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))

<b>Page 75: [14] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [15] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [16] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [17] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [18] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [19] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [20] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [21] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [22] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [23] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 12:09:00 PM</b>
Font: Courier New, 10 pt, Font color: Custom Color(RGB(42,0,255))		
<b>Page 75: [24] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 1:21:00 PM</b>
Space Before: 4 pt, After: 4 pt		
<b>Page 75: [25] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 1:21:00 PM</b>
Space Before: 4 pt, After: 4 pt		
<b>Page 75: [26] Formatted</b>	<b>Mike Edwards</b>	<b>12/11/2008 1:21:00 PM</b>
Space Before: 4 pt, After: 4 pt, Bulleted + Level: 1 + Aligned at: 1.27 cm + Tab after: 1.9 cm + Indent at: 1.9 cm		