SCA Service Component Architecture

SCA服务构件架构

装配模型规范

满江红 redsaga.com

SCA v1.00， 2007.3.15

SCA Service Component Architecture

Assembly Model Specification

SCA Version 1.00, March 15 2007

Technical Contacts:	Michael Beisiegel	IBM Corporation
	Henning Blohm	SAP AG
	Dave Booz	IBM Corporation
	Mike Edwards	IBM Corporation
	Oisin Hurley	IONA Technologies plc.
	Sabin Ielceanu	TIBCO Software Inc.
	Alex Miller	BEA Systems, Inc.
	Anish Karmarkar	Oracle
	Ashok Malhotra	Oracle
	Jim Marino	BEA Systems, Inc.
	Martin Nally	IBM Corporation
	Eric Newcomer	IONA Technologies plc.
	Sanjay Patil	SAP AG
	Greg Pavlik	Oracle
	Martin Raepple	SAP AG
	Michael Rowley	BEA Systems, Inc.
	Ken Tam	BEA Systems, Inc.
	Scott Vorthmann	TIBCO Software Inc.
	Peter Walker	Sun Microsystems Inc.
	Lance Waterman	Sybase, Inc.

Copyright Notice

© Copyright BEA Systems, Inc., Cape Clear Software, International Business Machines Corp, Interface21, IONA Technologies, Oracle, Primeton Technologies, Progress Software, Red Hat, Rogue Wave Software, SAP AG., Siemens AG., Software AG., Sun Microsystems, Inc., Sybase Inc., TIBCO Software Inc., 2005, 2007. All rights reserved.

License

The Service Component Architecture Specification is being provided by the copyright holders under the following license. By using and/or copying this work, you agree that you have read, understood and will comply with the following terms and conditions:

Permission to copy, display and distribute the Service Component Architecture Specification and/or portions thereof, without modification, in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the Service Component Architecture Specification, or portions thereof, that you make:

1. A link or URL to the Service Component Architecture Specification at this location:
   - http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications

2. The full text of the copyright notice as shown in the Service Component Architecture Specification.

BEA, Cape Clear, IBM, Interface21, IONA, Oracle, Primeton, Progress Software, Red Hat, Rogue Wave, SAP, Siemens, Software AG., Sun, Sybase, TIBCO (collectively, the "Authors") agree to grant you a royalty-free license, under reasonable, non-discriminatory terms and conditions to patents that they deem necessary to implement the Service Component Architecture Specification.

THE Service Component Architecture SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, REGARDING THIS SPECIFICATION AND THE IMPLEMENTATION OF ITS CONTENTS, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT OR TITLE.

THE AUTHORS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RELATING TO ANY USE OR DISTRIBUTION OF THE Service Components Architecture SPECIFICATION.

The name and trademarks of the Authors may NOT be used in any manner, including advertising or publicity pertaining to the Service Component Architecture Specification or its contents without specific, written prior permission. Title to copyright in the Service Component Architecture Specification will at all times remain with the Authors.

No other rights are granted by implication, estoppel or otherwise.

Status of this Document

This specification may change before final release and you are cautioned against relying on the content of this specification. The authors are currently soliciting your contributions and suggestions. Licenses are available for the purposes of feedback and (optionally) for implementation.

IBM is a registered trademark of International Business Machines Corporation in the United States, other countries, or both.

BEA is a registered trademark of BEA Systems, Inc.

Cape Clear is a registered trademark of Cape Clear Software

IONA and IONA Technologies are registered trademarks of IONA Technologies plc.
Oracle is a registered trademark of Oracle USA, Inc.
Progress is a registered trademark of Progress Software Corporation

Primeton is a registered trademark of Primeton Technologies, Ltd.
Red Hat is a registered trademark of Red Hat Inc.
Rogue Wave is a registered trademark of Quovadx, Inc

SAP is a registered trademark of SAP AG.

SIEMENS is a registered trademark of SIEMENS AG Software AG is a registered trademark of Software AG
Sun and Sun Microsystems are registered trademarks of Sun Microsystems, Inc.

Sybase is a registered trademark of Sybase, Inc.

TIBCO is a registered trademark of TIBCO Software Inc.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

目录

License 3
Status of this Document 4
目录 5
1. 装配模型 7
1.1 简介 7
1.2 概述 7
1.2.1 描述SCA工件图 8
1.3 构件 11
1.3.1 例子构件 14
1.4 实现 17
1.4.1 构 件 类 型 18
1.5 接口 23
1.5.1 本地和远程接口 24
1.5.2 双向接口 25
1.5.3 会话接口 26
1.5.4 WSDL接口的SCA相关方面 28
1.6 组合构件 29
1.6.1 属性-定义和配置 32
1.6.2 引用 36
1.6.3 服务 40
1.6.4 Wire 42
1.6.5 将组合构件作为构件实现使用 49
1.6.6 通过包含使用组合构件 51
1.6.7 包含多种类型构件实现的组合构件 54
1.6.8 强制类型 54
1.7 绑定 57
1.7.1 包含了并未在服务接口中定义的数据的消息 58
1.7.2 已部署绑定的URI形式 59
1.7.3 SCA Binding 61
1.7.4 Web Service绑定 62
1.7.5 JMS绑定 62
1.8 SCA 定义 62
1.9 扩展模型 63
1.9.1 定义一个接口类型 64
1.9.2 定义一个实现类型 65
1.9.3 定义一个绑定类型 66
1.10 打包以及部署 68
1.10.1 域 68
1.10.2 Contributions 68
1.10.3 已安装的contribution 72
1.10.4 对contribution的操作 73
1.10.5 对已存在（非SCA）的工件解析机制的使用 75
1.10.6 域级组合构件 75
2. 附录 1 76
2.1 XML Schemas 76
2.1.1 sca.xsd 76

2.1.2 sca-core.xsd 77
2.1.3 sca-binding-sca.xsd 83
2.1.4 sca-interface-java.xsd 83
2.1.5 sca-interface-wsdl.xsd 84
2.1.6 sca-implementation-java.xsd 84
2.1.7 sca-implementation-composite.xsd 85
2.1.8 sca-definitions.xsd 86
2.1.9 sca-binding-webservice.xsd 86
2.1.10 sca-binding-jms.xsd 86
2.1.11 sca-policy.xsd 86
2.2 SCA概念 86
2.2.1 绑定 87
2.2.2 构件 87
2.2.3 服务 87
2.2.4 引用 88
2.2.5 实现 88
2.2.6 接口 88
2.2.7 组合构件 89
2.2.8 组合构件包含 89
2.2.9 属性 89
2.2.10 域 90
2.2.11 连线 90
3. 附录 2：参考文献 90

1. 装配模型

1.1 简介

该文档描述SCA装配模型，其覆盖如下内容：

- 服务的装配模型，既适合紧耦合也适合松散耦合。
- 应用基础设施功能于服务与服务交互模型，包括安全性和事务性。

本文档首先简短概述 SCA 装配模型。
然后描述了SCA的核心元素，SCA component和SCA composite
文档的最后部分定义SCA装配模型是如何扩展的。

1.2 概述

SCA为构建基于SOA的应用和解决方案提供了编程模型。它基于这样的理念：将业务功能作为一系列的服务而提供，并由这一系列的服务组装起来的解决方案来满足特定业务需求。这些组合的应用既包括为应用而新创建的特定服务，也包括源自已已存在系统和应用的业务逻辑，这些业务逻辑作为组合构件的一部分被复用。SCA既为服务的组合也为服务构件的创建提供了模型，包括对SCA组组合构件中对已存在应用功能的复用。

SCA就是一个致力于为服务构件以及连接各服务构件的访问方式而包容各种广泛的技术模型。对于构件，不仅仅只是不同的编程语言，还包括那些编程语言普遍使用的框架和环境。对于访问方式，SCA组合允许使用广泛采用的各种通讯和服务访问技术。其中包括如Web service、Messaging 系统以及远程过程调用(RPC)技术等。

SCA装配模型由一系列工件组成。这些工件定义了SCA域的配置信息，域中的组合构件组合构件组合构件包含了服务构件集、连接以及描述它们是如何被连接在一起的相关工件。

SCA的基础工件就是component，它是SCA的构成单元。构件(component)由一个实现的可配置（implementation）实例所组成。在该构件中，实现是提供业务功能的程序代码片段。该业务功能作为服务(service)而提供，为其他构件所使用。实现也许依赖于由其他构件所提供的服务，这些依赖被称作"引用"(reference)。实现可以有一个可设置的属性(properties)，该属性是可以影响业务功能操作的数据值。构件通过提供属性值和连线(wire)到由其他构件提供服务的引用来配置实现。

SCA允许各种广泛采用的实现技术，比如传统的编程语言，如Java,C++,BPEL，也包括脚本语言，如PHP，Javascript，还包括声明性语言，如XQuery和SQL。

SCA把在装配应用中的内容和联接称为组合构件（composite）。组合构件能包含构件，服务，引用，属性声明以及描述这些元素间连接的连线(wire)。组合构件可以分组并连接以不同技术实现的构件，其允许为每个业务任务使用相应的技术。反过来，组合构件能作为完整的构件实现来使用：提供服务，依赖引用以及可设置的属性值。这样的组合构件实现能用于其他组合构件中的构件，支持业务解决方案的分层构建。在该解决方案中，高层服务内部是由一系列的低层服务实现的。组合构件的内容能作为元素组来使用。该元素组通过包含于高层的组合构件中发挥作用。

组合构件被部署于***SCA Domain***中。典型地，SCA域描述了一系列的服务，这些服务提供了由某个单一组织控制的业务功能区域。举例来说，对于商务中的会计部门，SCA域也许会涵盖所有与财务相关的功能，也可能包含一系列处理特定会计范围的组合构件，一个处理客户帐户，另一个处理帐户支付。为了方便构建和配置SCA域，组合构件被用于分组和配置相关的工件。

SCA为它的工件定义了XML文件格式。这些XML文件定义了SCA工件的可移植描述。SCA运行时可以拥有由这些XML文件所表示工件的其他描述。特别地，采用某种编程语言构件实现可以有属性或用于指定SCA装配模型中某些元素的注解(annotation)。这些XML为SCA 域的配置定义了一个静态的格式。SCA运行时也允许动态地修改域的配置。

1.2.1描述 SCA 工件图

该文档引入图来描述各种SCA工件，可视化在组合件中工件之间的关系。这些图用来配合并说明SCA工件的案例。

下图说明了SCA构件的一些特征：

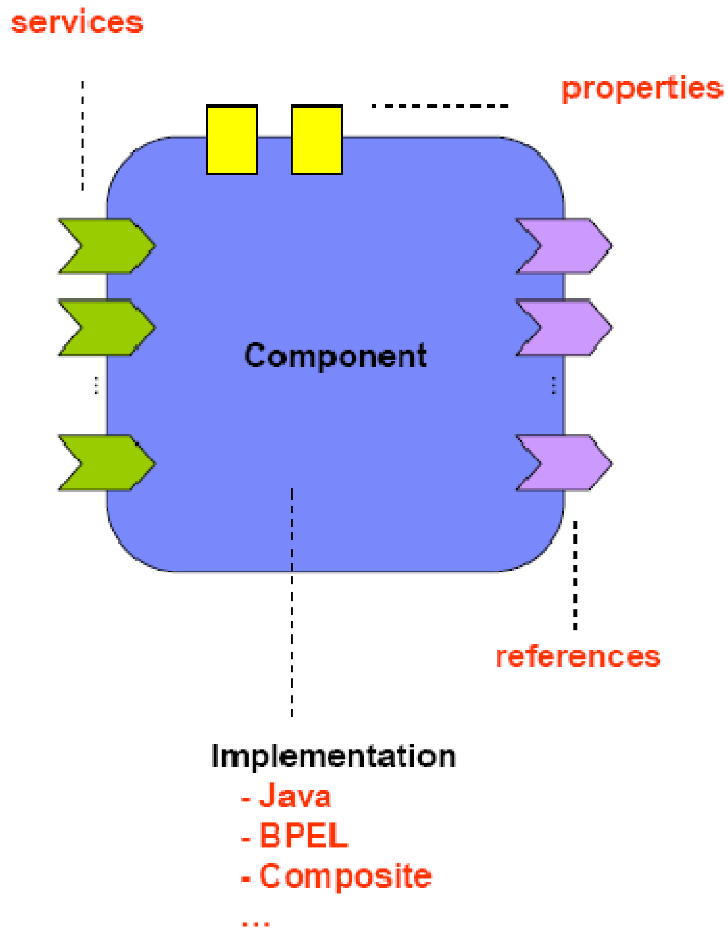

Figure 1: SCA Component Diagram

下图阐述了使用一系列构件组装的组合构件的一些特征：

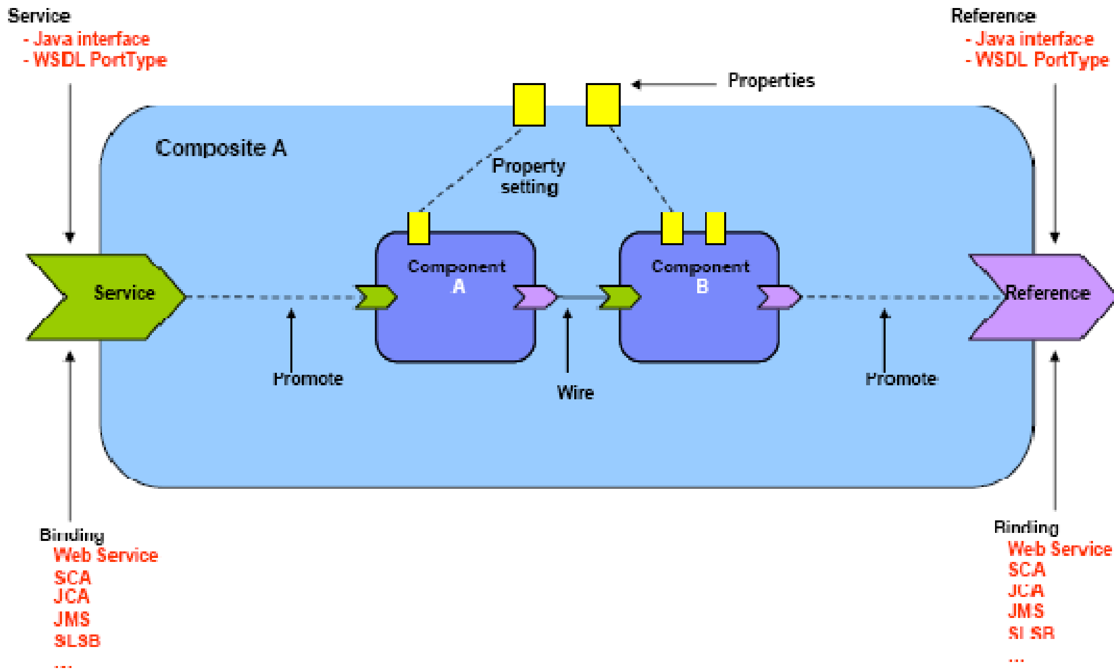

Figure 2: SCA Composite Diagram

下图说明了由一系列高层组合构件装配而成的 SCA 域，其中某些高层组合构件是由低层组合构件依次实现的。

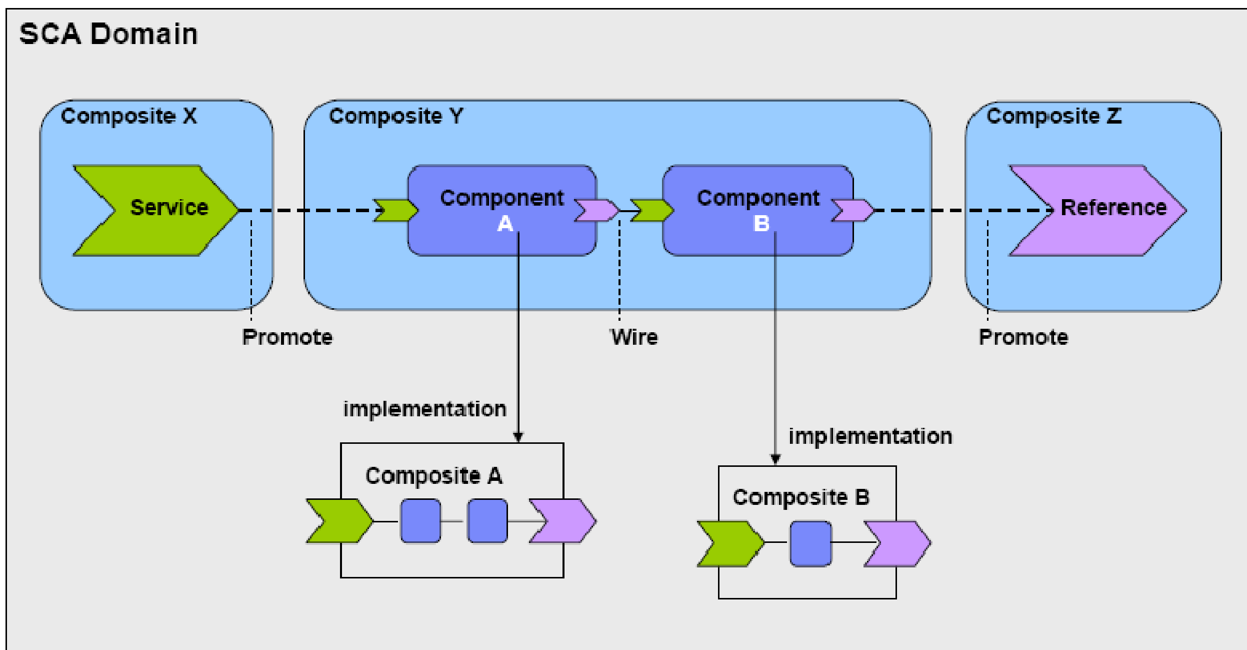

Figure 3: SCA Domain Diagram

1.3 构件

Component是SCA组合件中业务功能的基础元素。其通过SCA组合构件联合成完整的业务解决方案。

Component是实现的可配置实例。构件提供和消费服务。多个构件可以使用和配置为同一个实现，而每个构件又可以对实现有不同的配置。

构件在***xxx.composite***文件中作为组合构件的子元素来声明。***component***元素代表构件，它是组合构件(composite)元素的子元素。在composite中可以有零个或多个component元素。以下片段展示了带有component子元素schema的composite schema。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Component schema snippet -->
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="xs:anyURI"
               name="xs:NCName" local="xs:boolean"?
               autowire="xs:boolean"? constrainingType="QName"?
               requires="list of xs:QName"? policySets="list of xs:QName"?>

   ...

   <component name="xs:NCName" requires="list of xs:QName"?
           autowire="xs:boolean"?
               requires="list of xs:QName"? policySets="list of xs:QName"?
               constrainingType="xs:QName"?>*
         <implementation/>?
         <service name="xs:NCName" 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"?/>*
         </service>
         <reference name="xs:NCName" multiplicity="0..1 or 1..1 or 0..n or 1..n"?
             autowire="xs:boolean"?
               target="list of xs:anyURI"? policySets="list of xs:QName"?
               wiredByImpl="xs:boolean"? requires="list of xs:QName"?>*
               <interface/>?
               <binding uri="xs:anyURI"? requires="list of xs:QName"?
                     policySets="list of xs:QName"?/>*
         </reference>
         <property name="xs:NCName" (type="xs:QName" | element="xs:QName")?
             mustSupply="xs:boolean"?
               many="xs:boolean"? source="xs:string"? file="xs:anyURI"?>*
               property-value?
         </property>
   </component>

   ...

</composite>
```

Component元素有如下属性：

- ***name(必须)*** –构件的名字。在同一组合构件里的所有构件中名字必须唯一。
- autowire ***(可选)*** **–**指示包含的构件引用是否自动连线，默认是false，autowire在"自动连线"节介绍。
- requires ***(可选)*** **–**策略意图的列表。查看策略框架规范对该属性的描述。
- policySet(***可选)*** **–**策略集列表。查看策略框架规范对该属性的描述。
- ***constrainingType(可选)*** – 强制类型的名字。当指定时，构件的服务、引用和属性的集合，加之相关的意图将被限定为强制类型定义的集合。更多细节请查看"强制类型"节。

component元素有零个或一个***implementation***子元素，该子元素指定构件所使用的实现。没有implementaion子元素的component是不能运行的,但这种构件也许在自顶向下的开发过程中，在实现被开发出来之前作为一种定义实现的特定需求的手段还是有用的。

Component元素有零个或多个service子元素，用于配置构件的服务。可配置的服务是由实现定义的。

service元素有如下属性：

- ***name(必须)*** –服务名字。必须匹配由实现定义的服务名。
- ***requires(可选)*** --策略意图的列表。查看策略框架规范[11]对该属性的描述。
  注意：服务的有效的策略意图集包含由该requires属性中任意显式声明的意图，和由实现为服务所指定的任意意图。
- ***policySets (可选) --***策略集列表。查看策略框架规范[11]对该属性的描述。

service有零个或一个接口。该接口描述了服务提供的操作。接口是由作为service元素的子元素interface元素来描述的。如果没有指定接口，那么实现为服务指定的接口有效。如果指定了接口，就必须提供一个与实现提供的接口相兼容的子集。比如，为服务提供一个由实现定义的操作的子集。Interface元素的细节信息，请看"接口"节。

service元素有零个或多个***binding***子元素。如没有指定binding，那么实现为服务指定的binding有效。如果指定了，那些绑定会覆盖实现所指定的绑定。binding元素的细节信息，请查看"绑定"节。Binding与任意有效的PolicySet结合，必须满足服务上的策略意图集。如Policy Framework specification [10]中所述。

component元素有一个或多个reference子元素，用于配置构件的引用。可以配置的引用是由实现定义的。

reference元素有如下属性：

- ***name (必须)*** **–**引用名。必须匹配实现中定义的引用名称。
- ***autowire(可选)*** –指示引用是否自动连线，默认是false，autowire在"自动连线"节会讲到。
- ***requires (可选)*** –策略意图的列表。查看策略框架规范[10]对该属性的描述。
  注意：引用的有效的策略意图集由该requires属性中任意显式定义的意图，与任何由实现为引用所指定的意图组合而成。

- ***policySet(可选)*** –策略集列表。查看策略框架规范[10]对该属性的描述。

- ***multiplicity(可选)*** –定义了能连接到目标服务的引用的连线数目。覆盖实现上的引用指定的多重性。其值只能遵循或更进一步的限制：比如0..n则0..1或1..n则1..1。multiplicity可以是如下取值：
  - 1..1 –作为源，只能有一个引用的连线
  - 0..1 –作为源，能有0个或1个引用的连线
  - 1..n –作为源，能有1个或多个引用的连线
  - 0..n –作为源，能有0个或多个引用的连线

- ***target (可选)*** – 一个或多个目标服务URI的列表，依赖于multiplicty的设置。每个值都将引用连线到解析该引用的构件服务。具体细节查看“连线”节。覆盖实现中为引用所指定的任意目标。

- ***wiredByImpl（可选）*** – 一个boolean值。默认"false"，指示实现动态地连线该引用。如果设置为“true”，表示这个引用的目标由实现代码在运行时设置（比如，由代码用某种方式来得到端点引用，并通过使用相关的客户程序和实现规范中定义的编程接口作为引用的目标来设置）。如果设置为"true"，那么组合构件中引用不应该被静态地连线，而是不要连线。

引用有零个或一个接口，描述引用必须的操作。接口用作为reference元素的子元素interface元素来描述。如果没有指定接口，那么实现为引用指定的接口有效。如果指定了接口，实现就必须提供一个兼容接口的超集。比如，提供一个实现为引用定义的操作超集。Interface元素的细节信息，请看“接口”节。

reference有0个或多个binding子元素。如没有指定binding，那么实现为引用指定的binding有效。如果指定了，那些绑定会覆盖所有由实现指定的绑定。binding元素的细节信息，请查看“绑定”节。binding与对于binding来说有效的任意PolicySet结合，必须满足引用的策略意图集，如Policy Framework specification [10]所述。

注意：binding元素可以指定端点，该端点是绑定的目标。引用必须不能混用通过binding元素指定的端点和通过target属性指定的目标端点。如果设置了target属性，那么binding元素只能列出一个或多个绑定类型，该绑定类型用于由target属性所标识的连线。这种情况下，所有被标识的绑定类型可用于每条连线上。如果在binding元素中指定了端点，那么每个端点必须使用binding元素中定义的绑定类型。除此之外，这种情况下，每个binding元素必须指定端点。

component元素有零个或多个***property***子元素。该子元素用于配置实现的属性数据值。每个property元素都为命名的属性提供了一个传递给实现的值。可配置的属性及其类型在实现中定义。实现可以声明某个属性为multi-valued（多值），在这种情况下，给定的属性会存在多个属性值。

属性值可以按以下三种方式之一来指定：

- 其值由属性元素的内容
- 通过引用包含构件的组合构件的属性值。该引用通过property元素的***source***属性指定。

Source属性值可以是某个XPath表达式的形式。该形式允许组合构件特定的属性可以通过name来定位。在属性复杂的地方，能扩展XPath表达式来指定复杂值的子部分。

所以，比如，source="$currency"用于引用称为“currency”的组合构件属性，而source="$currency/a"用于引用称为currency的复杂组合构件属性的"a"子部分。

- 通过使用***file***属性指定一个指向含有属性值的文件的浏览器可打开的URI。引用文件的内容作为属性值使用。

（译者注：原文为：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.）

如果存在多于一个的属性值，source属性优先，然后是file属性。

可选地，属性的类型可以按以下两种方式之一来指定：

- 通过在XML schema中定义的类型qualified name，使用**type**属性
- 通过在XML schema中的全局元素的qualified name，使用**element**属性

指定的属性类型必须兼容实现中声明的属性类型。如果没有指定类型，则使用实现中声明的属性类型。

property元素有以下属性：

- ***name（必须）***– property的名字，必须匹配实现中定义的属性名。
- ***type（可选）***–属性的类型定义为XML Schema 类型的qualified name。
- ***element（可选）*** –属性的类型定义为XML Schema 全局元素的qualified name，type即是全局元素的类型

（译者注：原文为：***element (optional)*** – 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（可选）***– 一个XPath表达式，指向组合构件所包含的属性。该构件的属性值源自包含该构件的组合构件的属性值。
- ***file (可选)*** –一个指向包含了属性值的文件的浏览器可打开的URI
- ***many (optional)*** –false指定属性为单值，true为多值。可以覆盖实现中为属性所指定的many。其值只能是等效于或更进一步的限定。比如，如果实现指定many为true，那么构件可以配置为false。在多值属性的情况下，则在实现中以属性值的集合形式存在。

（译者注：原文为：***many (optional)*** – (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.）

1.3.1 例子构件

下图展示了在装配图中用于描述构件的***component***符号。

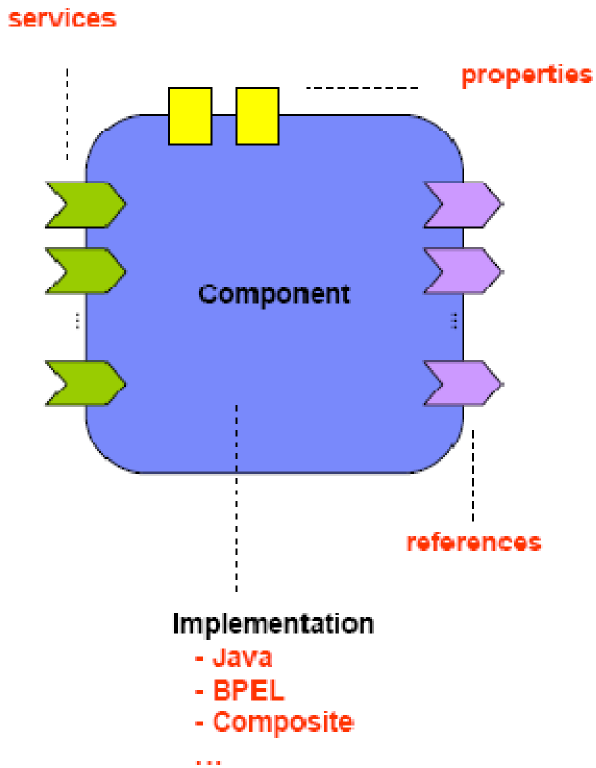

Figure 4: Component symbol

下图展示包含MyValueServiceComponent构件的MyValueComposite组合构件的装配图。

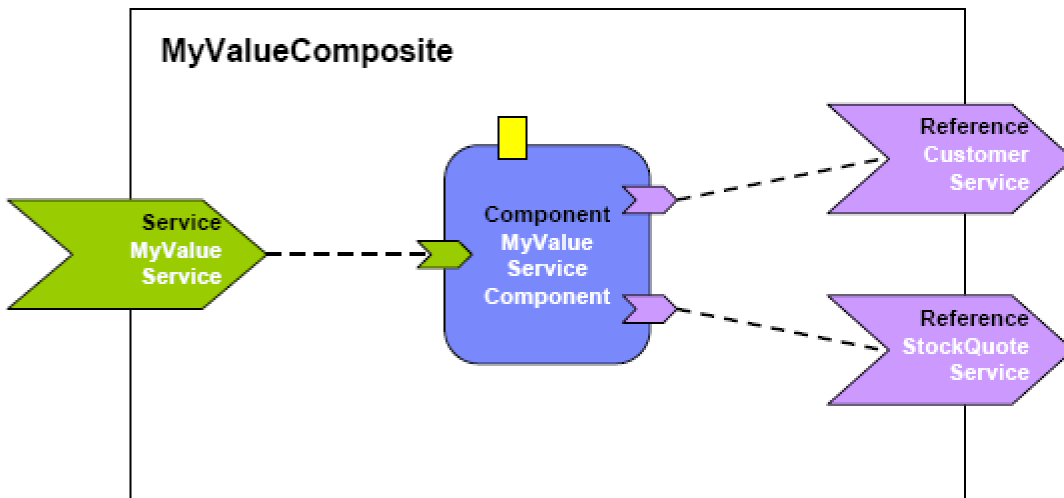

Figure 5: Assembly diagram for MyValueComposite

如下片段为包含MyValueServiceComponent的component元素的MyValueComposite展示了MyValueComposite.composite文件内容。为名为currency的属性设置一个值，并提升（promote）

customerService和stockQuoteService引用。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_1 example -->
<composite      xmlns="http://www.osoa.org/xmlns/sca/1.0"
                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>
```

注意，显式地声明MyValueServiceComponent引用只是为了清晰的目的。引用由MyValueServiceImpl实现定义，没有必要在component上重新声明，除非想连线它们或覆盖它们的某些方面。

如果既要currency属性为多值又要MyValueServiceComponent的customerService引用声明为多值（即：该属性的many=true并且引用的multiplicity=0..n或1..n），如下片段给出了一个composite文件样式的例子。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_2 example -->
<composite      xmlns="http://www.osoa.org/xmlns/sca/1.0"
                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>
         <property name="currency">Yen</property>
         <property name="currency">USDollar</property>
         <reference name="customerService"
               target="InternalCustomer/customerService"/>
         <reference name="StockQuoteService"/>
   </component>

   ...

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

   <reference name="StockQuoteService"
```

```
        promote="MyValueServiceComponent/StockQuoteService"/>
</composite>
```

这里假设组合构件有另一个称为"InternalCustomer"（这里没有展示）的构件。该构件有一个服务。MyValueSerivceComponent的customerService引用连线到该服务，同时在外部，该customerService引用通过组合构件的CustomerService引用被提升。

1.4 实现

构件实现是提供服务或引用其他地方所提供的服务的业务功能的具体实现。另外，某个实现可以有一些可设置的属性值。

SCA允许你从广泛的实现类型中选择任何一种技术，比如Java、BPEL或C++。这每种类型都代表了特定的实现技术。该技术不仅仅简单定义实现语言，如Java，而且也可以定义使用某个特定的框架或运行时环境。例如包括使用Spring framework或Java EE EJB技术的Java实现。

例如，在一个composite文件中的component的声明中，元素implementation.java和implementation.bpel分别指定Java和BPEL实现类型。Implementation.composite指定将SCA composite作为实现来使用。Implementation.spring和implementation.ejb分别用于指定使用Spring framework和Java EE EJB技术编写的Java构件。

如下片段为Java和BPEL实现类型以及将composite作为实现的方式时的implementation元素。

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

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

service,reference和property是实现的可配置的部分。SCA将它们一起作为component type。Service,reference和property的特性在“构件”节描述。依赖于实现类型，实现可以声明其服务,引用和属性，并能为所有的那些服务,引用以及属性特性设置值。

举例：

- 对于服务，实现可以定义接口,绑定,URI地址,意图和策略集，包括绑定的细节信息。
- 对于引用，实现可以定义接口,绑定,目标URI地址，意图和策略集，包括绑定的细节信息。
- 对于属性，实现可以定义它的类型以及默认值。
- 实现自身可以定义意图和策略集。

服务,引用和属性的大多数特征都可以通过使用和配置实现的component来覆盖，或者component决定并不覆

盖那些特征。某些特征不能覆盖，比如意图。而其他的特征，如接口，只能以特定可控的方式来覆盖（细节信息，查看“component”节）

一个实现声明服务,引用和属性的方式依赖于实现的类型。比如，某些语言，如Java，提供了注解用于在代码中嵌入声明信息。

在运行时期，实现的实例是实现的特定运行时的实例-其运行时的形式依赖于所用的实现技术。实现的实例从其基于的实现派生出业务逻辑，但其属性值和引用则来源于配置该实现的构件。

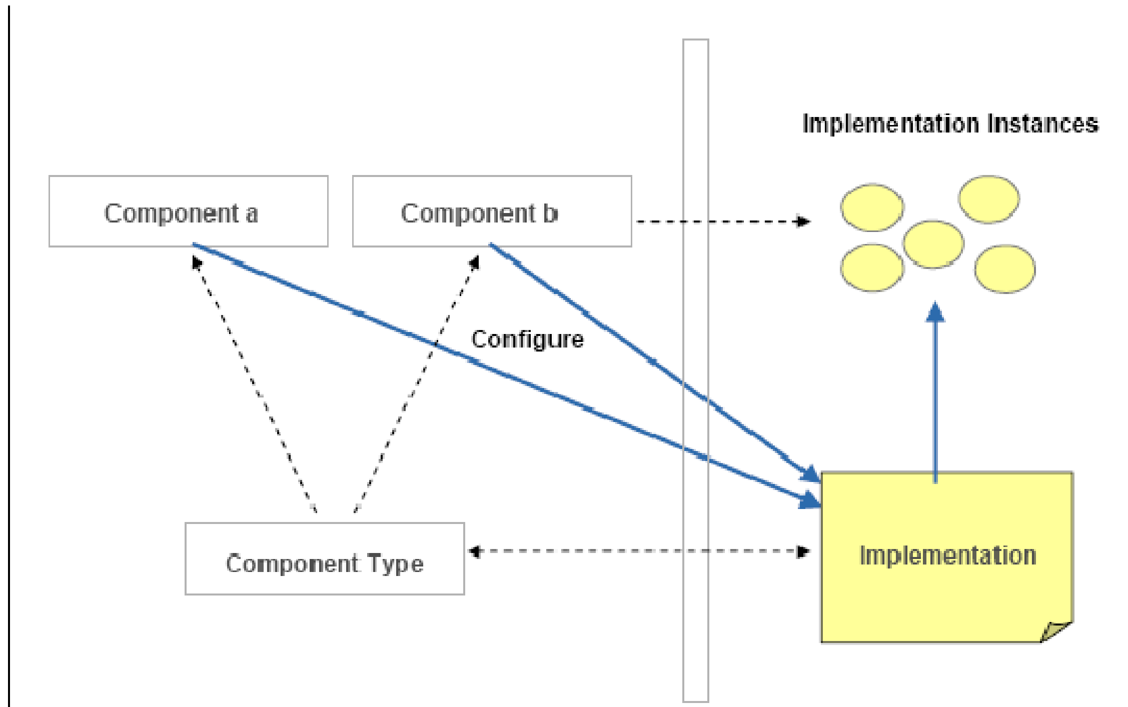

Figure 6: Relationship of Component and Implementation

1.4.1 构件类型

Component Type表示实现的可配置的方面。一个component type由提供的服务、被连线的对其他服务的引用以及可设置的属性组成。使用了该实现的构件来配置可设置的属性以及可设置的对服务的引用。

Component type通过两步来确定。第二步补充在第一步中发现的信息。第一步是内省实现（如果可能的话），包括实现的注解的内省（如果可用的话）。第二步解决那些通过实现的内省不可能的情况，或者无法提供完整信息，需要查找一个SCA component type文件的情况。在component type文件中找到的Component type信息必须与从实现内省来的等效信息兼容。component type文件可以列出部分信息，其他的信息则来源于实现。

在理想的情况下，component type信息是通过内省实现确定的，比如代码注解。component type文件提供了通过内省实现而无法得到信息的情况下的后备方式，即对实现类型提供构件类型信息的机制。

component type文件与实现文件有相同的名字，只是扩展名不同，是".***componentType***"。component type在文件中是通过***componentType***元素来定义的。component type文件的***位置***依赖于构件的实现类型：它在各自实现类型“客户程序与实现模型规范(client and implementation model specification)”中描述。

componentType元素能包含Service元素，Reference元素和Property元素。

以下片段展现了componentType schema。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Component type schema snippet -->
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
   constrainingType="QName"? >

   <service name="xs:NCName" 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"?/>*
   </service>

   <reference name="xs:NCName" target="list of xs:anyURI"?
         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"?/>*
   </reference>

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

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

</componentType>
```

ComponentType只有一个属性：

- ***constrainingType（可选）***– constrainingType的名字。当指定的时候，实现的服务,引用和属性的集合，再加上相关的意图，都会受限于costrainingType所定义的集合。查看ConstrainingType节可以查看更多的细节信息。

Service描述了实现的可寻址的接口。服务通过componentType元素的service子元素来描述。在componentType中可以有0个或多个service元素。具体细节，请查看“服务”节。

Reference描述了实现需要其他组件所提供的服务。reference通过componentType元素的reference子元素来描述。component type定义中可以有0个或多个reference元素。具体细节，请查看reference节。

Property允许通过外部设置值来配置实现。每个属性都作为property元素定义。componentType元素可以有0个或多个property子元素。具体细节，请查看property节。

Implementation描述了实现自身固有的特性，特别是意图和策略。意图和策略的描述信息，请查看策略框架规范Policy Framework specification[10]。

1.4.1.1 componentType案例

如下片段为MyValueServiceImpl实现展示了componentType文件的内容。componentType文件展示了MyValueServiceImpl实现的服务,引用和属性。在这个案例中，使用Java来定义接口：

```
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0">

   <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>
```

1.4.1.2 实现案例

以下是一个用java编码的案例实现。具体细节查看SCA案例代码文档SCA Example Code document [3]。

AccoutServiceImpl实现了***AccountService***接口，该接口是通过Java接口定义的：

```
package services.account;

@Remotable
public interface AccountService{

   public AccountReport getAccountReport(String customerID);
}
```

以下是AccountServiceImpl类的完整列表，展现了其实现的服务，加上其服务引用和可设置的属性。注意标注代码SCA方面的Java注解的使用，包括@Property和@Reference标记：

```
package services.account;

import java.util.List;

import commonj.sdo.DataFactory;

import org.osoa.sca.annotations.Property;


import org.osoa.sca.annotations.Reference;

import services.accountdata.AccountDataService;
import services.accountdata.CheckingAccount;
import services.accountdata.SavingsAccount;
import services.accountdata.StockAccount;
import services.stockquote.StockQuoteService;

public class AccountServiceImpl implements AccountService {

   @Property(译者注：此annotation不能应用于private的属性上)
   private String currency = "USD";

   @Reference(译者注：此annotation不能应用于private的属性上)
   private AccountDataService accountDataService;
   @Reference(译者注：此annotation不能应用于private的属性上)
   private StockQuoteService stockQuoteService;

   public AccountReport getAccountReport(String customerID) {

    DataFactory dataFactory = DataFactory.INSTANCE;
    AccountReport accountReport =
    (AccountReport)dataFactory.create(AccountReport.class);
    List accountSummaries = accountReport.getAccountSummaries();

    CheckingAccount  checkingAccount  =  accountDataService.getCheckingAccount(customerID);
    AccountSummary checkingAccountSummary =
    (AccountSummary)dataFactory.create(AccountSummary.class);
    checkingAccountSummary.setAccountNumber(checkingAccount.getAccountNumber());
    checkingAccountSummary.setAccountType("checking");
    checkingAccountSummary.setBalance(fromUSDollarToCurrency(checkingAccount.getBalance
    ()));
    accountSummaries.add(checkingAccountSummary);
```

```
        SavingsAccount savingsAccount = accountDataService.getSavingsAccount(customerID);
        AccountSummary savingsAccountSummary =
        (AccountSummary)dataFactory.create(AccountSummary.class);
        savingsAccountSummary.setAccountNumber(savingsAccount.getAccountNumber());
        savingsAccountSummary.setAccountType("savings");
        savingsAccountSummary.setBalance(fromUSDollarToCurrency(savingsAccount.getBalance()
        ));
        accountSummaries.add(savingsAccountSummary);

        StockAccount stockAccount = accountDataService.getStockAccount(customerID);
        AccountSummary stockAccountSummary =
        (AccountSummary)dataFactory.create(AccountSummary.class);
        stockAccountSummary.setAccountNumber(stockAccount.getAccountNumber());
        stockAccountSummary.setAccountType("stock");
        float balance=
        (stockQuoteService.getQuote(stockAccount.getSymbol()))*stockAccount.getQuantity();
        stockAccountSummary.setBalance(fromUSDollarToCurrency(balance));
        accountSummaries.add(stockAccountSummary);

        return accountReport;
       }

       private float fromUSDollarToCurrency(float value){

        if (currency.equals("USD")) return value; else
        if (currency.equals("EURO")) return value * 0.8f; else
        return 0.0f;
       }
    }
```

对于AccountServiceImpl，以下是等效的SCA componentType定义，通过对以上代码反射出来的：

```xml
<?xml version="1.0" encoding="ASCII"?>
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <service name="AccountService">
         <interface.java interface="services.account.AccountService"/>
   </service>
   <reference name="accountDataService">
         <interface.java interface="services.accountdata.AccountDataService"/>
   </reference>

      <reference name="stockQuoteService">
            <interface.java interface="services.stockquote.StockQuoteService"/>
      </reference>
```

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

</componentType>
```

对于关于Java实现的完整细节，请查看“Java客户程序与实现规范” Java Client and Implementation Specification和SCA例子代码文档。其他的实现类型有各自的规范文档。

1.5 接口

Interface定义了一个或多个业务功能。这些业务功能是通过服务提供或通过引用来使用的。为了其他组件的使用，一个服务提供了严格的单一接口的业务功能。每个接口都定义了一个或多个服务***操作***，每个operation都有0个或1个***请求（输入）消息***和0个或1个***响应（输出）消息***。请求和响应消息可以是简单类型，如string 值，也可以是复杂类型。

SCA当前支持如下的接口类型系统：

- Java interfaces
- WSDL 1.1 portTypes
- WSDL 2.0 interfaces

(WSDL: Web Services Definition Language [8])

SCA在接口类型(interface type)方面也是可以扩展的。可以通过SCA的扩展机制支持其它的接口类型系统，这些在“扩展模型”节会讲到。

以下片段展示了Java interface元素的schema。

```
<interface.java interface="NCName" … />
```

Interface.java元素有如下属性：

- ***interface*** –Java interface的全限定名(qualified name)

以下样例为Java interface元素展示了使用例子。

```
<interface.java interface="services.stockquote.StockQuoteService"/>
```

这里，java interface定义在./services/stockquote/StockQuoteService.class 的java类文件中。该文件的根目录是由Interface所在的contribution定义的。

对于Java接口类型系统，service方法的参数和返回值都使用Java类或简单Java类型来描述。SDO（Service Data Objects [2]）是Java类的首选形式，因为它们与XML技术的集成。

需要更多的关于Java接口的信息，包括SCA规范的注解细节，请查看《Java客户程序与实现规范》the Java Client and Implementation specification [1]。

以下片段给出了WSDL portType（WSDL 1.1） 或WSDL interface（WSDL2.0）元素

```
<interface.wsdl interface="xs:anyURI" … />
```

interface.wsdl元素有如下属性：

- ***interface*** – portType/interface 如下格式的URI地址
  - <WSDL-namespace-URI>#wsdl.interface(<portTypeOrInterface-name>)

如下片段为WSDL portType/interface元素给出了一个使用样例：

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

对于WSDL 1.1，interface属性指向WSDL中的某个portType。对于WSDL 2.0，interface属性指向WSDL里的某个interface。对于WSDL1.1 portType和WSDL2.0接口类型系统，服务操作的参数和返回值都是使用XML schema描述的。

1.5.1 本地和远程接口

一个远程服务是被运行于不同于服务本身的某个操作系统进程的客户程序所调用的（这也可以是运行于与服务不同的机器的客户程序）。构件实现的服务是否是远程的是由服务的接口定义的。如果使用java，则通过在java接口上添加@Remotable注解来定义远程接口（查看《Java客户程序与实现模型规范》）。WSDL定义的接口总是远程的。

远程接口的风格是典型的粗粒度的，松耦合的交互模式。远程服务接口禁止使用方法或操作的重载。

不管远程服务是从进程外被远程调用还是被其他运行于同一个进程的构件所调用，数据交换语义都是传值的方式。

远程服务的实现可以在调用期间或之后修改输入消息（参数），在调用之后修改返回的消息（结果）。如果远程服务被本地或远程地调用，SCA容器负责确保对输入消息的修改和返回消息的调用后修改，对调用者来说，是透明的（译者注：即调用者看到的就象不存在对输入消息的修改和返回消息的调用后修改这回事）。

这里是一个片段，展现了远程java接口的例子：

```
package services.hello;

@Remotable

public interface HelloService {

   String hello(String message);
}
```

当被同一个进程中的构件调用时，远程服务的实现可能使用传引用的数据交换语义。这可以改善运行于同一进程的构件间服务调用的性能。使用@AllowsPassByReference注解（查看《Java客户程序与实现规范》）来完成此功能。

本地接口类型的服务只能被运行于与实现本地服务的构件处于同一个进程的客户程序调用。本地服务不能通过所在composite的远程服务来发布。在Java接口时，本地服务是通过Java interface definition来定义，不需要@Remotable注解。

本地接口的风格是典型的细粒度、用于紧耦合的交互。本地服务接口可以使用方法或操作的重载。
对于本地接口类型服务的调用，其数据交换语义是传引用。

1.5.2 双向接口

一个业务服务与另一个业务服务的关系总是点对点的，在服务级要求双路依赖。换句话说，一个业务服务既是某个业务服务合作者所提供服务的消费者，同时又为服务合作者提供服务。特别是在基于异步消息而非远程过程调用（RPC）的交互情形下。双向接口的理念在SCA中用于直接地建模点对点双向业务服务关系。

对于特定的接口类型系统的interface元素必须要求一个可选的回调接口规范。如果指定了一个回调接口，那么SCA将该接口整体作为一个双向接口。

如下片段展示了使用带有一个可选的callbackInterface属性Java接口定义的interface元素。

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

如果一个服务是使用一个双向接口元素定义的，那么其实现实现该接口，并且通过回调接口来与调用服务接口的客户程序进行交互。

如果使用双向接口元素定义一个引用，那么使用该引用的客户构件实现会通过该接口来调用被引用的服务。客户构件实现必须实现该回调接口。

回调既能用于远程也能用于本地服务。一个双向服务接口必须要么都为远程的，要么都为本地的。一个双向服务不能同时混用本地服务和远程服务。

SCA中提供这样的方式，允许与源服务调用的客户构件不同的构件提供一个回调接口。这些是如何做到的请查看《SCA java客户程序与实现规范》（“将会话服务作为参数传递” 章节）。

1.5.3会话接口

服务有时并不能简单地定义，以便每个操作都自治而完全不依赖于同一服务的其他操作。相反地，为达到更高级别的目标，得有一个操作调用的序列。SCA称此操作序列为会话conversation。如果服务使用了双向接口，那么会话同时包含操作和回调。

这些会话服务一般是通过使用会话标识来管理的。这些会话标识既可以是1)应用数据（消息部分或操作的参数）的一部分，也可以是2)与应用数据隔离的通讯部分（可能在头部信息中）。SCA为了描述上述第二种形式的会话服务接口合约，引入了会话接口conversational interface这个概念。以这种形式，运行时环境可以在部署时所指定的相应绑定机制的帮助下自动管理会话。SCA并不强制通过应用数据来维护会话服务的任何方面的标准。SCA会话服务的支持对通过应用数据维护会话的会话服务不起任何作用。（译者注：因为SCA是使用上述第2种方式管理会话的）

典型地，会话服务关心与正在进行的会话相关的状态数据。会话状态数据的创建和管理对客户程序以及会话服务的实现都有重大的影响。

一般来说，有开发会话服务需求的应用开发者要求书写许多基础代码。他们需要：
- 选择或定义客户与提供者间通讯会话信息的协议
- 在提供者端，将会话消息路由到可以处理会话的机器上，同时处理并发数据访问问题
- 在客户程序中编写代码来使用和编码会话信息
- 在实现以及客户程序中，维护与会话相关的状态，有时是持久化和事务性。

SCA为许多角色之间划分与会话服务相关的职责提供了可能：
- 应用开发者：声明服务接口是会话的（具体协议的细节信息留给绑定）。使用生命周期语义，API或 其他的编码机制（由所使用的实现类型来定义）来管理会话的状态。
- 应用装配者：选择支持会话的绑定
- 绑定提供者：实现一个协议，该协议可以传递与每个操作请求/响应相关的会话信息
-实现类型提供者：为应用开发者定义API以及其他的编程机制，以便可以访问会话信息。可选地，实现实例生命

周期语义。该生命周期语义自动管理基于绑定的会话信息的实现状态。

该规范要求以一个名为"conversational"的策略意图的方式，来标注接口为“会话的”。这种意图标注的形式依赖与接口类型。注意，当使用并未标注conversational意图的接口时，也可能为服务或引用设置coversational意图。在复用某个已存在但并不包含SCA信息的接口定义时，这一点很有用。

conversational意图的意思是：无论是接口的客户还是提供者都假定消息（在每个方向上）都会被作为正在进行的会话的一部分来处理，而不依赖于在消息体中的标识信息（比如在操作的参数中）。实际上，会话接口指定某个高层抽象协议必须满足服务所用的任意实际上的绑定/策略组合。

支持会话接口的绑定/策略组合例子如下：
- 带有WS-RM policy的Web service绑定
- 带有WS-Addressing policy的Web service绑定
- 带有WS-Context policy的Web service 绑定
- 带有使用了JMS correlationID 头部信息的conversation policy的JMS绑定

会话发生于某个客户与某个目标服务之间。因此，来自一个客户的请求到达多个目标会话服务，将导致多个会话。比如，如果客户A调用了服务B和C（这两个服务都实现了会话接口），两个会话结果，一个是A与B之间的，另一个是A与C之间的。同样地，经由多个实现实例的请求会导致多个会话。比如，从A到B的请求，然后又经由B到C，会产生两个会话（A和B，B和C）。在前面的例子中，如果请求再从C到A发起，那么会产生第三个会话（并且A的实现实例不同于发起源请求的那个实例A）。

一个会话接口的任意操作调用都可能会启动一个会话。一个操作是否会启动一个会话依赖于构件的实现及其实现类型。实现类型可能支持会话性服务构件。如果一个实现类型确实提供了该支持，那么就必须提供一个机制，确定什么时候一个新会话会用于一个操作（比如，在java中，在第一次使用某个注入的引用时，生成一个新的会话；在BPEL中，当客户的partnerLink进入作用域时，生成一个新会话）。

会话接口中的一个或多个操作可能会用endsConversation注解注释（注解接口的机制依赖于接口类型）。当一个接口是双向的时候，也可以以这种方式在一个回调接口的操作上进行注解。当一个会话结束操作被调用时，指示客户和服务提供者会话已经完成。任意的尝试调用同一个会话相关的操作或回调的后续行为都会引发一个sca:ConversationViolation错误。

sca:ConversationViolation错误在以下错误之一发生时被抛出：
- 在特定会话结束后接收到一个消息
- 会话标识无效（不唯一、超出范围等等）
- 在结束会话操作的输入消息中不存在会话标识
- 客户程序或服务尝试在会话结束之后发送会话消息

该错误使用SCA命名空间标准的"sca"前缀，该前缀与http://www.osoa.org/xmlns/sca/1.0的URI相关联。

资源的生命周期以及唯一标识符与会话之间的关系都是由服务的实现类型决定的，并且"endConversation"注解有可能不直接影响它们。比如，一个WS-BPEL流程可能比大多数的会话生命周期都要长。

尽管会话接口并不要求任何标识信息做为消息体部分传递，但其实概念上是存在一个与会话关联的标识的。个别的实现类型可能提供一个API来访问与会话相关的ID，尽管并不假定标识的结构。实现类型也可能为客户程序或服务提供者提供一种设置conversation ID的手段，尽管这种操作仅仅被某些绑定/策略组合所支持。

鼓励实现类型规范为实现了会话接口的构件，定义并提供会话性实例的生命周期管理。然而，实现也可以手工管理会话状态。

1.5.4 WSDL 接口的 SCA 相关方面

一般来说，SCA有许多方面应用到接口上，比如标注它们为conversational。这些方面应用于接口本身，胜过它们在SCA中特定位置的使用。提供标注接口定义自身的适当手段是一个附加的需求，该需求超出了接口定义语言提供的基本功能。

对于WSDL 接口，存在一个扩展机制，允许附加信息被包含于WSDL文档之中。SCA利用了该扩展机制。为了使用SCA扩展机制，SCA命名空间http://www.osoa.org/xmlns/sca/1.0 必须在WSDL文档中声明。

首先，SCA在SCA命名空间中声明了一个全局属性，该全局属性提供了附加策略意图-@requires的机制。该属性的定义如下：

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

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

@requires属性可以应用于WSDL portType元素（WSDL1.1）和WSDL Interface元素（WSDL2.0）中。该属性包含一个或多个意图名字，这些意图名字在《策略框架规范》[10]中定义了。任何使用了带有required意图接口的服务或引用，都会隐式地增加这些意图到其自身的@requires列表中。

为了指定WSDL 接口是会话的，在WSDL portType或WSDL Interface上使用如下属性设定：

```
requires="conversational"
```

SCA定义了一个***endsConversation***属性，该属性用于在WSDL 接口声明中标记特定的操作会结束会话。这仅仅对于那些也标注为conversational的WSDL 接口有意义。这个endsConversataion属性在SCA命名空间中是全局属性，其定义如下：

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

如下片段是在portType上标注requires属性，在一个operation上标注endsConversation属性的WSDL PortType的样例。

```
...
<portType name="LoanService" sca:requires="conversational">
    <operation name="apply">
       <input message="tns:ApplicationInput"/>
       <output message="tns:ApplicationOutput"/>
    </operation>
    <operation name="cancel" sca:endsConversation="true">
    </operation>
    ...
</portType>
...
```

1.6 组合构件

SCA 组合构件被用于在逻辑分组中装配SCA元素。其在SCA 域中是组合的基本单元。SCA 组合构件包含一系列的构件,服务,引用以及内连它们的连线，外加上用于配置构件的一系列属性。

组合构件可以在更高层的组合构件里作为构件实现的形式。换句话说，高层的组合构件可以包含由组合构件实现的构件。关于将组合构件作为构件实现使用的更详细的信息，请查看将组合构件作为构件实现使用这一节。

组合构件的内容可以通过inclusion在其他组合构件中使用。当一个组合构件被另一个组合构件包含时，全部内容可以在其上级组合构件中被有效地使用。其内容完全可见，并能被上级组合构件中的其他元素所引用。关于将一个组合构件包含进另一个组合构件的更详细的信息，请查看通过包含使用组合构件节。

一个组合构件可以作为部署的单元来使用。用这种方式使用的时候，组合构件contribute元素到一个SCA域中。一个组合构件既能通过inclusion的方式部署到SCA域中，也能通过将组合构件作为一个实现部署到域中。更多组合构件部署的细节，请查看处理SCA域节。
（译者注：原文为：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.）

组合构件在xxx.composite文件中定义。组合构件用composite元素来描述。如下片段展示了composite元素的schema。

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

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

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

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

   <component name="xs:NCName" autowire="xs:boolean"?
         requires="list of xs:QName"? policySets="list of xs:QName"?>*
         <implementation/>?
         <service name="xs:NCName" requires="list of xs:QName"?
               policySets="list of xs:QName"?>*
               <interface/>?
               <binding uri="xs:anyURI"? name="xs:QName"?
                     requires="list of xs:QName"?
                     policySets="list of xs:QName"?/>*
               <callback>?
                     <binding uri="xs:anyURI"? name="xs:QName"?
                           requires="list of xs:QName"?
                           policySets="list of xs:QName"?/>+
               </callback>
         </service>
         <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
               source="xs:string"? file="xs:anyURI"?>*
               property-value
         </property>
         <reference name="xs:NCName" target="list of xs:anyURI"?
               autowire="xs:boolean"? wiredByImpl="xs:boolean"?
               requires="list of xs:QName"? policySets="list of xs:QName"?
               multiplicity="0..1 or 1..1 or 0..n or 1..n"?/>*
               <interface/>?
               <binding uri="xs:anyURI"? name="xs:QName"?
                     requires="list of xs:QName"?
                        policySets="list of xs:QName"?/>*
```

```
            <callback>?
            </callback>
            <binding uri="xs:anyURI"? name="xs:QName"?
                  requires="list of xs:QName"?
                  policySets="list of xs:QName"?/>+
        </reference>
    </component>

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

</composite>
```

composite元素有如下属性：

- ***name (必须)*** –组合构件的名字。组合构件名字的形式是一个XML QName，在由targetNamespace属性标识的命名空间中。
- ***targetNamespace (可选)*** –composite声明所在的目标命名空间标识符
- ***local (可选)*** – 组合构件中的所有构件是否必须运行于同一个操作系统进程。Local="true"指示所有的构件运行于同一进程。Local="false"，是默认值，指示组合构件里的不同的构件运行于不同的操作系统进程，这些构件可能运行在某个网络的不同节点上。
- ***autowire (可选)*** – 被包含的构件引用是否自动连线，默认为false,在自动连线节描述。
- ***constrainingType (可选)*** –constrainingType的名字。当指定的时候，组合构件的服务,引用和属性的集合，再加上相关的意图，都会受限于costrainingType所定义的设置。查看ConstrainingType节可以查看更多的细节信息。
- ***requires (可选)*** –策略意图的列表。查看策略框架规范对该属性的描述。
- ***policySets (可选)*** – 策略集列表。查看策略框架规范对该属性的描述。

组合构件包含0个或多个属性，服务，构件，引用，连线以及内含的组合构件。这些工件具体的细节在以下章节会描述。

构件包含配置化的实现，该实现承载其组合构件的业务逻辑。构件提供服务并引用其他的服务。组合构件服务定义了组合构件所提供的公开服务，并能被组合构件的外部访问。组合构件引用描述该组合构件依赖于其外部其他地方所提供的服务。连线描述组合构件中的构件服务与构件引用之间的连接。被包含的组合构件contribute它们所包含的元素给使用它们的组合构件。（译者注：这里的它们指代那些被包含的组合构件）

组合构件服务是对组合构件中某一个构件的某一服务的提升（***promotion***）。这意味着，组合构件服务实际上是由其内部的一个构件所提供的。组合构件引用是对一个或多个构件的一个或多个引用的提升。只要所有的构件引用相互兼容，多个构件引用能被提升为同一个组合构件的引用。在多个构件引用被提升为同一个组合构件的引用的地方，它们都共享相同的配置，包括相同的目标服务。

组合构件的服务和引用分别使用它们被提升的服务与引用的配置（比如绑定和策略集）。可选地，通过对组合构件服务或引用的绑定与其它方面的配置，组合构件的服务与引用可以覆盖被提升的服务和引用的全部或某些配置。

构件的服务和引用可以提升为组合构件的服务和引用，同时也可以在组合构件的内部被连线。对于一个引用，只有该

引用支持大于1的multiplicity才有意义。

1.6.1 属性-定义和配置

属性允许通过外部设置数据值对实现进行配置。一个实现，包括组合构件，可以定义0个或多个属性。每个属性都一个类型，可以是简单类型也可以是复杂类型。实现也可以为属性定义一个默认值。属性可以用构件中的值进行配置。

组合构件中属性的声明遵从如下schema片段所描述的形式：

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

<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               name="xs:QCName" ... >

   ...

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

</composite>
```

属性元素有下列属性：

- ***name（必须）*** - 属性的名字。
- 以下之一（***必须***）
  - ***type*** – 属性的类型 –XML schema类型的限定名
  - ***element*** –属性的类型为某个XML schema全局元素的类型
- ***many (可选)*** – 属性是单值（指定为false）或多值(指定为true).默认值为***false***. 多值属性的情况下，在实现中以属性值的集合出现。
- ***mustSupply (可选) –*** 属性值是否必须由构件提供-当mustSupply="true"时，构件必须提供一个值，因为实现对于该属性没有默认值。当mustSupply="false"时（mustSupply属性的默认设置），只能提供默认的属性值,因为默认值的含义是当使用该属性的构件没有提供值时，才能使用默认值。

property元素可以包含一个可选的***default-property-value,***它为属性提供一个默认值。默认值必须匹配属性声明的类型:

- 字符串，如果***type***是简单类型（必须匹配声明的类型)
- 匹配通过***type***声明的复杂类型
- 匹配由***element***指定的元素类型
- 如果指定了many="true"，则允许多值

对于非composite类型的构件实现，能以与具体实现相关的形式（如java类中的注解）或通过在某个

componentType文件中的属性声明，来声明属性。（componentType在前面描述过）

当构件使用某个实现，可以配置实现的属性值。属性的配置形式在构件节展现。

1.6.1.1 属性例子

对于如下属性声明以及设置值的案例，用到了复杂类型：

```
<xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="http://foo.com/"
               xmlns:tns="http://foo.com/">
   <!-- ComplexProperty schema -->
   <xsd:element name="fooElement" type="MyComplexType"/>
   <xsd:complexType name="MyComplexType">
         <xsd:sequence>
               <xsd:element name="a" type="xsd:string"/>
               <xsd:element name="b" type="anyURI"/>
         </xsd:sequence>
         <attribute name="attr" type="xsd:string" use="optional"/>
   </xsd:complexType>
</xsd:schema>
```

下述组合构件展示了带有默认值，复杂类型属性的声明，同时也展示了构件中复杂类型属性值的设置：

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

<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               xmlns:foo="http://foo.com"
               targetNamespace="http://foo.com"
               name="AccountServices">
<!-- AccountServices Example1 -->
   ...
   <property name="complexFoo" type="foo:MyComplexType">
         <MyComplexPropertyValue xsi:type="foo:MyComplexType">
               <foo:a>AValue</foo:a>
               <foo:b>InterestingURI</foo:b>
         </MyComplexPropertyValue>
   </property>

   <component name="AccountServiceComponent">
         <implementation.java class="foo.AccountServiceImpl"/>
         <property name="complexBar" source="$complexFoo"/>
         <reference name="accountDataService"
               target="AccountDataServiceComponent"/>
         <reference name="stockQuoteService" target="StockQuoteService"/>
   </component>

   ...

</composite>
```

AccountServices组合构件中，名为***complexFoo***的属性声明，属性被定义为***foo:MyComplexType***类型。组合构件中定义了命名空间foo，该命名空间引用了example XSD。MyComplexType定义在该命名空间中。complexFoo的声明包含一个默认值。该默认值作为property元素的内容被定义。在这个例子中，默认值由类型为foo:MyComplexType的***MyComplexPropertyValue***元素和它的两个子元素<foo:a>、<foo:b>，再加

上MyComplexType的定义组成。

AccountServiceComponent构件中，设置属性***complexBar***的值。该属性由实现声明，构件来配置。这个例子中，complexBar的类型是foo:MyComplexType类型。案例展示了complexBar属性值是通过complexFoo来设定的– complexBar属性元素的***source***属性声明了其属性值来自于组合构件的某个属性值。Source属性值为***$complexFoo***,这里complexFoo是该组合构件的属性名。这个source属性值表明整个来源属性值将用于设置构件的属性值。

如下案例展示了简单类型属性值的设置。其源自于组合构件的复杂类型属性值中的属性域(***part***)。

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

<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
               xmlns:foo="http://foo.com"
               targetNamespace="http://foo.com"
               name="AccountServices">
<!-- AccountServices Example2 -->

   ...

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

   <component name="AccountServiceComponent">
         <implementation.java class="foo.AccountServiceImpl"/>
         <property name="currency" source="$complexFoo/a"/>
         <reference name="accountDataService"
               target="AccountDataServiceComponent"/>
         <reference name="stockQuoteService" target="StockQuoteService"/>
   </component>

   ...

</composite>
```

在这个案例中，***AccountServiceComponent***构件设置了名为***currency***属性的值，其类型为string。通过将source属性设置为***$complexFoo/a***，***currency***属性值可以从***AccountServices***组合构件的属性获取。***$complexFoo/a***是一个XPath表达式，该表达式先选到complexFoo的属性名，然后再选到complexFoo的***a***子元素。"a"子元素是一个子符串，匹配currency属性的类型。

声明构件中属性并设置属性值的更多案例如下：

简单类型并带有默认值的属性声明：

```
<property name="SimpleTypeProperty" type="xsd:string">
MyValue
```

```
</property>
```

复杂类型并带有默认值的属性声明：

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

用element声明类型的属性声明：

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

属性值为简单类型：

```
<property name="SimpleTypeProperty">
MyValue
</property>
```

复杂类型属性值，同时展示复杂类型属性值的设置

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

对于元素类型，属性值：

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

复杂类型并支持多值的属性声明：

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

提供多值的属性值的设置

```
<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>
```

1.6.2 引用

组合构件的引用是通过提升其内部构件所定义的引用来定义的。每个被提升的引用都指示其内部构件引用必须解析为组合构件外部的服务。通过使用组合构件的reference元素，来提升其内部构件的引用。

组合构件的引用，用composite元素的***reference***子元素来表示。在组合构件中有0个或多个reference元素。如下片段展示了带有reference元素schema的composite schema。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Reference schema snippet -->
<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="xs:anyURI"
               name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
               constrainingType="QName"?
               requires="list of xs:QName"? policySets="list of xs:QName"?>

   ...

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

   ...

</composite>
```

reference元素有如下属性：

- ***name (必须)*** **–** 引用的名字. 在组合构件中，所有的引用名必须唯一. 组合构件引用名可以与被提升的构件引用名不同。
- ***promote (必须)*** **–** 标识一个或多个被提升的构件引用。该值是用空格分隔的，形式为<component-name>/<reference-name>的值的列表。如果构件只有一个引用，那么对引用名的指定是可选的。
- ***requires (可选)*** –策略意图的列表。查看策略框架规范对该属性的描述。
- ***policySets (可选)*** –策略集列表。查看策略框架规范对该属性的描述。
- ***multiplicity (可选)*** -定义了能连接到目标服务的引用的连线数目。multiplicity可以是如下取值：
  - 1..1 –作为源，只能有一个引用的连线
  - 0..1 –作为源，能有0个或1个引用的连线
  - 1..n –作为源，能有1个或多个引用的连线
  - 0..n –作为源，能有0个或多个引用的连线
- ***target (可选)*** – 一个或多个目标服务URI的列表，依赖于multiplicty的设置。每个值都将引用连线到组合构件中一个服务。该组合构件将包含引用的组合构件作为某个构件的实现来使用。具体细节查看"连线"节。覆盖实现中为引用所指定的任意目标。
- ***wiredByImpl（可选）*** – 一个boolean值。默认"false"，指示实现动态地连线该引用。如果设置为"true"，表示这个引用的目标由实现代码在运行时设置（比如，由代码用某种方式来维护端点引用，并通过使用相关的客户程序和实现规范中定义的编程接口作为引用的目标来设置）。如果设置为"true"，那么组合构件中引用不应该被静态地连线，而是不要连线。

组合构件引用可选地指定一个接口（***interface***），multiplicity，必须的意图（***required intents***）和绑定。只要是没指定的方面，都默认为被提升构件的引用配置。

如果指定了接口，则必须提供一个相同的或与被提升构件引用定义的接口兼容的超集（superset）。比如，为引用提供一个由构件定义的操作的超集。接口用一个或多个interface元素描述，interface元素为reference元素的子元素。Interface元素的细节请查看接口节。

multiplicity属性指定的值必须与构件引用上指定的multiplicity兼容。比如，必须等效或更进一步的限制。所以multiplicity 为0..1或1..1的组合构件引用可以分别用于被提升的构件引用multiplicity为0..n和1..n的地方。然而multiplicity为0..n 或1..n的组合构件引用不能分别用于提升multiplicity为0..1或1..1的构件引用。

指定的***required intents***添加到或更进一步限定，被提升的构件引用上定义的必须的意图（***required intents***）

如果指定了一个或多个绑定（***bindings***），从组合构件引用的视角，它们会覆盖被提升的构件引用所指定的任意和所有的绑定定义。对于其组合构件中的局部连线而言，构件引用上的绑定定义仍然有效。reference元素可以0个或多个binding子元素。Binding元素的细节在绑定节描述。更多关于连线的细节，请查看连线节。

注意：binding元素可以指定绑定的一个目标端点(endpoint)。一个引用必须不能将通过binding元素指定的端点

和通过target属性指定的目标端点混用。如果设置了target属性，那么binding元素只能列出一个或多个绑定类型。绑定类型用于由target属性标识的连线。这种情况下，每个连线上被标识的所有绑定类型都是有效的。如果是在binding元素中指定端点，那么每个端点必须使用binding元素中定义的绑定类型。另外，这种情况下，需要为每个binding元素指定一个端点。

如果接口有一个定义的回调，***reference***元素有一个可选的***callback***元素,该元素有一个或多个binding子元素。如果有必要指定用于处理回调的绑定细节，可以指定***callback***及其binding子元素。如果不存在callback元素，其行为依赖于运行时的实现。

使用不同的组合构件引用，同一个构件引用也许会被提升多次，但只有在构件引用上的multipicity定义为0..n或1..n时才行。组合构件引用上的multipicity可以相应地作限制。

两个或多个构件引用可以被同一个组合构件引用提升，但只能当如下情况都满足才行：

- 构件引用的接口是相同，或如果组合构件引用自身声明了一个接口，那么所有的构件引用必须有与组合构件引用的接口相兼容的接口
- 构件引用的multiplicity兼容。比如，一个可以是另一个的限定形式，也就意味着组合构件引用是隐式或显式的限定形式
- 构件引用上声明的意图必须兼容 – 这种情况下，应用于组合构件引用的意图是每个被提升的构件引用所指定的必须意图（required intents）的并集。如果任何意图冲突(比如对于特定的意图存在互不兼容的限定符)，则存在错误。

1.6.2.1引用例子

下图展示了在装配图中用于描述引用的引用图标。

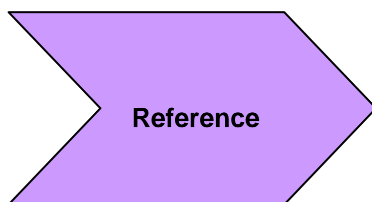

Figure 7: Reference symbol

下图为包含CustomerService引用和StockQuoteService引用的MyValueComposite展示了装配图。

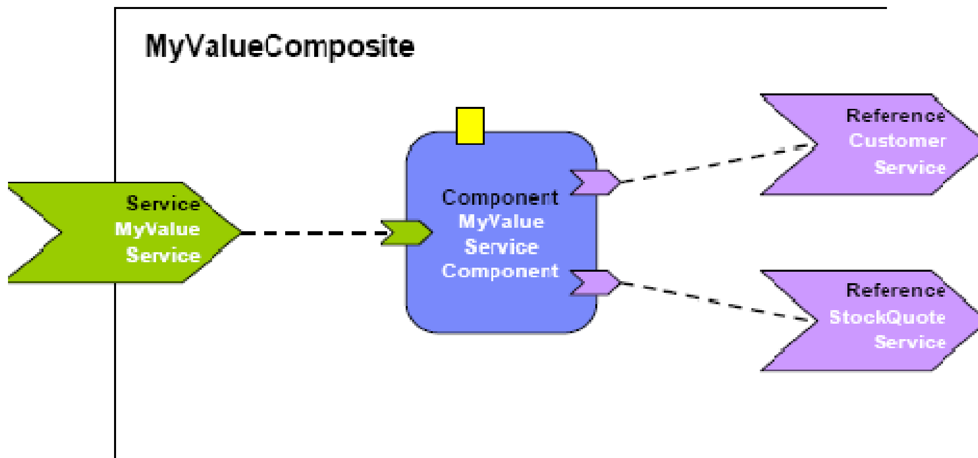

Figure 8: MyValueComposite showing References

如下代码片段为MyValueComposite.composite文件内容，其中包含CusomterService和StockQuoteService引用元素的MyValueComposite。CustomerService引用用SCA binding限定。StockQuoteService引用用Web service binding限定。绑定的端点地址可以被指定，比如使用绑定的uri属性（细节查看绑定节），或在组合构件中覆盖。这种情况下，尽管StockQuoteService引用被限定为Web service，但是它的接口是用java接口定义的，该java接口创建自目标web服务的WSDL portType 。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_3 example -->
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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)"/>

   </reference>
```

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

1.6.3服务

通过提升其内部构件所定义的服务，来定义组合构件的服务。构件服务以组合构件服务的形式被提升。

组合构件服务用composite元素的service子元素来描述。composite里有0个或多个service元素。如下片段展示带有service子元素schema的composite schema：

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Servicee schema snippet -->
<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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:QName"?
               requires="list of xs:QName" policySets="list of xs:QName"?/>*
         <callback>?
               <binding uri="xs:anyURI"? name="xs:QName"?
                     requires="list of xs:QName"?
                     policySets="list of xs:QName"?/>+
        </callback>
   </service>

   ...

</composite>
```

service元素有如下属性：

- ***name (必须)*** – 服务的名字，该名字在组合构件的所有服务中必须是唯一的。组合构件服务名可以与被提升的构件服务名不同。
- ***promote (必须)*** – 标识被提升的服务，其值的形式为<component-name>/<service-name>。如果目标构件只有一个服务，service-name是可选的。
- ***requires (可选)*** – 策略意图列表。该属性的描述查看策略框架规范。
- ***policySets (optional)*** – 策略集列表。该属性的描述查看策略框架规范。

组合构件服务可选地指定接口（***interface***），必须的意图(***required intents***)以及绑定。只要是没指定的方面，都默认为被提升构件服务的配置。

如果指定了一个接口，则该接口必须是一个相同的或与被提升构件服务定义的接口兼容的子集（subset）。比如，提供一个由构件服务定义的操作的子集。接口用零个或一个interface元素描述，interface元素为service元素的子元素。Interface元素的细节请查看接口节。

指定的required intents添加到或更进一步限定，被提升的构件服务上定义的必须的意图（required intents）

如果指定了绑定，从组合构件服务的视角，它们会覆盖被提升的构件服务所指定的绑定定义。对于其组合构件中的局部连线而言，构件服务上的绑定定义仍然有效。service元素可以0个或多个binding子元素。binding元素的细节在绑定节描述。更多关于连线的细节，请查看连线节。

如果接口有一个定义的回调**，**service元素有一个可选的***callback***元素,该元素有一个或多个binding子元素。如果有必要指定用于处理回调的绑定细节，可以指定***callback***及其binding子元素。如果不存在callback元素，其行为依赖于运行时的实现。

相同的构件服务可以被多个组合构件服务所提升。

1.6.3.1 服务例子

下图展示了在装配图中用于描述服务的服务图标。

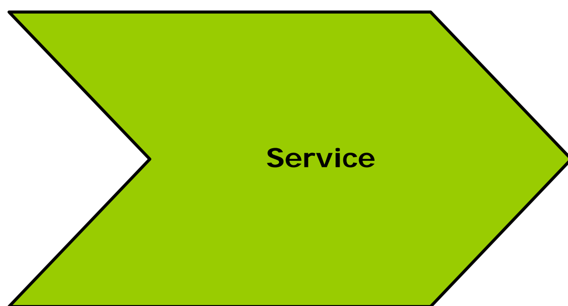

Figure 9: Service symbol

下图给出了包含MyValueService服务的MyValueComposite的装配图

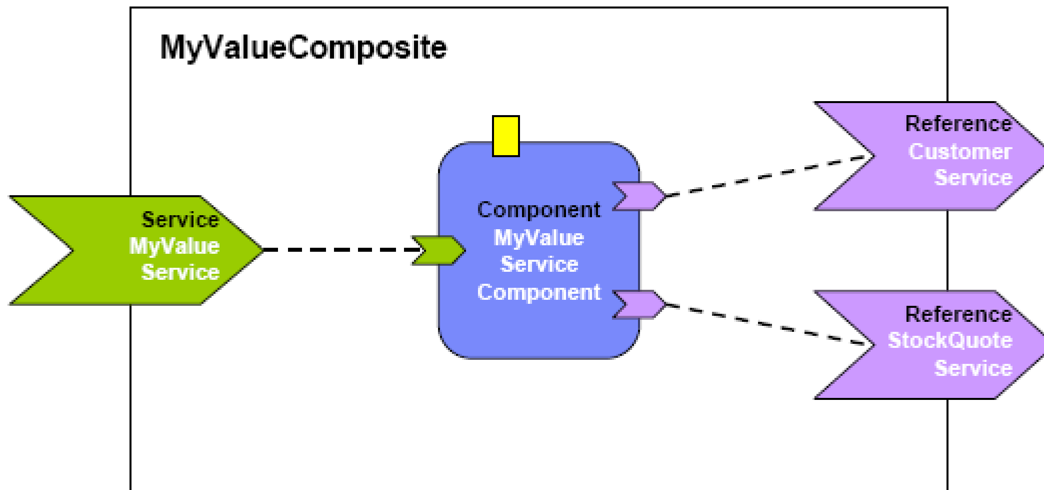

Figure 10: MyValueComposite showing Service

如下代码片段为包含MyValueService service元素的MyValueComposite，展示了MyValueComposite.composite文件内容。MyValueService是对MyValueServiceComponent所提供的服务的一个提升。因为MyValueServiceComponent只提供了一个服务，所以省略了被提升的服务名。MyValueService组合构件服务用Web service binding限定。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite_4 example -->
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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>

   ...

</composite>
```

1.6.4 Wire

组合构件中的SCA连线将源构件引用连接到目标构件服务。

定义连线的方法之一是通过使用target属性来配置一个构件引用。用解析引用的服务连线目标URI配置reference元素。当引用的multiplicity为0..n或1..n时，多个目标服务是有效的。

定义连线的另一个方法是通过作为composite元素的wire子元素来定义。在composite中有0个或多个wire元素。这种定义的方法在从连接的元素中分离出连线有助于简化开发或可操作性的情况下是有用的。下面是一个例子：SCA域中的组件相对来说是比较稳定的，但是在具体的应用当中某些部件又经常需要变动，此时就需要修改以前的组件。如果使用单独的连线元素，就可以以最小的代价修改连线。

注意：通过wire元素指定的连线与通过引用的target属性指定的连线效果是一样的。 禁止将target属性指定的wire与引用的binding子元素中对端点（endpoint）的指定混用，这条规则也适用于通过单独的wire元素指定的连线。

如下片段展示了带有构件的reference元素、组合构件服务以及wire子元素schema的composite schema

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Wires schema snippet -->
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
                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>
```

构件的reference元素和服务的reference元素对于target有一个值列表，列表元素连线目标的URI形式如下，多个值以空格分隔：

- <component-name>/<service-name>
    - 目标是一个构件的服务。如果目标构件只有一个带有兼容接口的服务，那么服务名的指定是可选的。

wire 元素有如下属性:

- ***source (必须)*** –标识源构件引用名. 有效的URI schemes为:
    - <component-name>/<reference-name>
        - source所指的地方是一个构件的引用。如果源构件只有一个引用，那么对reference-name的指定是可选的。

- ***target (必须)*** – 标识目标构件的服务。 有效的URI schemes为
  - <component-name>/<service-name>
    - target所指的地方是一个构件的服务。如果目标构件只有一个带有兼容接口的服务，那么对service-name的指定是可选的。

对于一个作为构件实现的组合构件而言（译者注：即将组合构件作为一种构件的实现形式），连线仅仅能连接包含于同一组合构件中的源与目标（与哪个文件用于描述组合构件无关）。通过更高一级组合构件定义的连线，完成本组合构件的服务和引用与本组合构件的外部实体之间的连线。

只有target实现了一个与source要求的接口相兼容的接口，连线才能将source连接到target。source与target只有以下条件成立才兼容：

1. source接口与target接口必须都为远程接口或都为本地接口
2. target接口上的操作必须与source上指定接口的操作要么相同要么前者是后者的超集。
3. 对于单独操作的兼容是指操作签名的兼容，也就是操作名、输入参数类型和输出参数类型必须都相同。
4. 输入参数类型与输出参数类型的顺序也必须相同。
5. source期望的Fault和Exception集合必须与target所指定的要么相同要么前者是后者的超集。
6. 两个接口的其他指定的属性必须匹配，包括作用域（scope）和回调接口。

只要两个接口定义的操作是等效的，连线可以在各个方向上对不同接口语言（比如Java接口和WSDL的portType）进行连接。如果操作、参数、返回值以及错误/异常互相都可以映射，则它们就是等效的。

服务客户程序不能（便捷地）询问运行时关于服务实现所提供的附加接口（比如，在Java里的"instance of"结果是不便捷的）。对于一个SCA实现拥有对所有的连线的代理是有效的，比如传递给实现的一个引用对象也许只有引用的逻辑接口，而不是某个实现了目标服务的（Java）类实例，甚至不管接口是本地的且目标服务运行于同一进程中。

注意：部署一个含有未被连线的引用的组合构件是允许的。对于带有属性multipicity为1..1或1..n的未被连线的引用，SCA运行时提供的部署进程应该发布一个警告信息。

1.6.4.1 连线例子

下图为含有服务间连线、构件以及引用的MyValueComposite2展示了装配图。

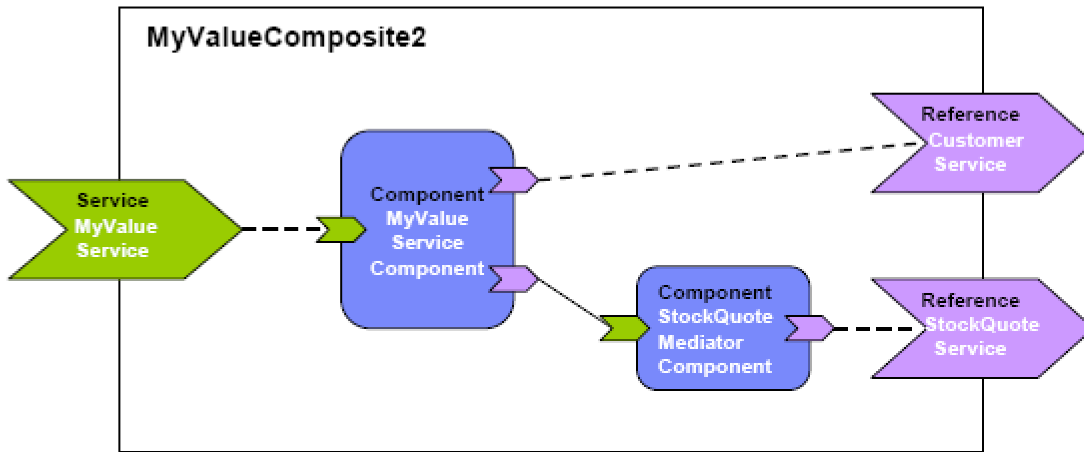

Figure 11: MyValueComposite2 showing Wires

如下片段为包含已配置的构件和服务引用的MyValueComposite2展示了MyValueComposite2.composite文件内容。MyValueService服务被连线到MyValueServiceComponent。MyValueServiceComponent的customerService引用被连线到组合构件的CustomerService引用（译者注：这里其实是指提升）。MyValueServiceComponent的stockQuoteService引用被连线到StockQuoteMediatorComponent，而StockQuoteMediatorComponent的引用被连线到组合构件的StockQuoteService引用（译者注：这里其实是指提升）。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- MyValueComposite Wires examples -->
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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"
               target="StockQuoteMediatorComponent"/>
   </component>

   <component name="StockQuoteMediatorComponent">
         <implementation.java class="services.myvalue.SQMediatorImpl"/>
         <property name="currency">EURO</property>
         <reference name="stockQuoteService"/>
   </component>

   <reference name="CustomerService"
         promote="MyValueServiceComponent/customerService">
         <interface.java interface="services.customer.CustomerService"/>
         <binding.sca/>
   </reference>
```

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

</composite>
```

1.6.4.2 自动连线

SCA提供了一个特性叫做自动连线，该特性可以帮助简化组合构件的装配。自动连线可以让构件引用自动连线到满足的构件服务上，而不需要创建引用于服务间显式的连线。当使用自动连线特性时，未被提升同时未被显式连接到组合构件中某个服务的构件引用会自动连线到同一组合构件中的某一目标服务。自动连线通过在组合构件中查找匹配引用接口的服务接口来完成该工作。

默认情况下，自动连线特性是不使用的。通过设置autowire属性为"true"来开启自动连线功能。通过设置autowire属性为"false "关闭自动连线功能。autowire属性可以用于组合构件中的如下任意元素：

- reference
- component
- composite

没有显式设置autowire属性的元素，该属性的设置继承自其父元素的设置。所以，reference元素从包含它的component元素继承设置。component元素从包含它的composite元素继承设置。在任意级别都不存在设置，默认autowire="false"。

举例来说，如果composite元素的autowire设置为"true"，则意味着对于该组合构件中的所有构件引用都开启了自动连线功能。在这个例子中，可以通过在关注的构件以及引用上设置autowire="false"，为特定的构件以及引用关闭自动连线功能。

对于每个开启了自动连线的构件引用而言，自动连线处理过程会在组合构件中查找与引用相兼容的目标服务。这里的“兼容”是指：

- 目标服务接口必须是引用接口的兼容超集（在连线节描述）。
- 应用于服务的意图、绑定以及策略必须与引用上的相应设置相兼容– 以便引用到服务的连线不会由于绑定与策略不匹配而引起错误。（详情查看策略框架规范）

如果对于一个特定的引用，查找到了多于1个的有效目标服务，那么行为依赖于引用的multiplicity设置：

- 对于multiplicity为0..1和1..1的，SCA运行时以运行时相关的模式选择其中一个目标服务并将引用连线到该目标服务。
- 对于multiplicity为0..n和1..n的，将引用连线到所有的目标服务。

如果对于一个特定的引用，没有查找到有效的目标服务，那么行为依赖于引用的multiplicity设置：

- 对于multiplicity为0..1和0..n的，不存在问题 – 没有服务被连线，也不存在错误。

- 对于multiplicity为1..1和1..n的，因为引用必须被连线，所以由SCA运行时抛出一个错误。

1.6.4.3 自动连线例子

这个例子展示了相同构件的两个版本 – 第一个版本没有用自动连线，显式连线完成，第二个版本使用自动连线完成。这两种情况的最终结果都是一样的 – 将引用连接到服务的连线一样

下图是一个组合构件图：

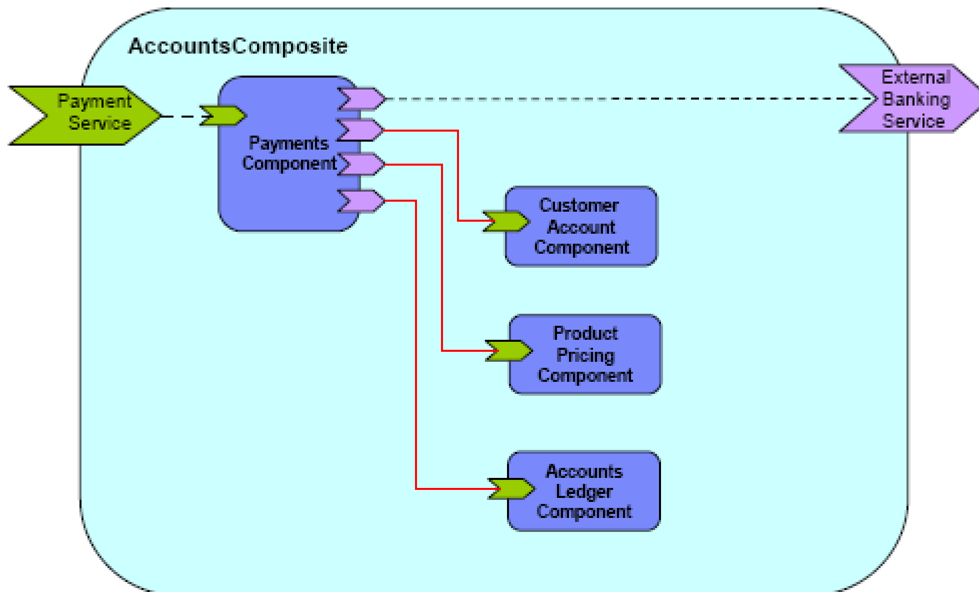

Figure 12: Example Composite for Autowire

第一个，使用显式连线：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Autowire Example - No autowire    -->
<composite     xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.osoa.org/xmlns/sca/1.0"
    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">
        <implementation.java class="com.foo.accounts.CustomerAccount"/>
```

```
    </component>

    <component name="ProductPricingComponent">
        <implementation.composite class="com.foo.accounts.ProductPricing"/>
    </component>

    <component name="AccountsLedgerComponent">
        <implementation.composite class="com.foo.accounts.AccountsLedger"/>
    </component>

    <reference name="ExternalBankingService"
        promote="PaymentsComponent/ExternalBankingService"/>

</composite>
```

第二个，使用自动连线的组合构件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Autowire Example - With autowire -->
<composite    xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://foo.com"
    name="AccountComposite">

    <service name="PaymentService" promote="PaymentsComponent">
         <interface.java class="com.foo.PaymentServiceInterface"/>
    </service>
    <component name="PaymentsComponent" autowire="true">
        <implementation.java class="com.foo.accounts.Payments"/>
        <service name="PaymentService"/>
        <reference name="CustomerAccountService"/>
        <reference name="ProductPricingService"/>
        <reference name="AccountsLedgerService"/>
        <reference name="ExternalBankingService"/>
    </component>
    <component name="CustomerAccountComponent">
        <implementation.java class="com.foo.accounts.CustomerAccount"/>
    </component>
    <component name="ProductPricingComponent">
        <implementation.composite class="com.foo.accounts.ProductPricing"/>
    </component>
    <component name="AccountsLedgerComponent">
        <implementation.composite class="com.foo.accounts.AccountsLedger"/>
    </component>
    <reference name="ExternalBankingService"
        promote="PaymentsComponent/ExternalBankingService"/>
</composite>
```

在第二种情况下，为PaymentsComponent设置了自动连线，并且其任意一个引用都不存在显式的连线 – 通过自动连线自动创建了连线。

在第二个例子中，也可以为PaymentsComponent省略所有的服务与引用。它们存在只是为了清晰的目的，但如果省略了它们，构件的服务与引用仍然存在，因为它们是由该构件所使用的实现提供的。

1.6.5 将组合构件作为构件实现使用

在更高层级的组合构件中，组合构件可以以构件实现的形式存在 – 换言之，更高层级的组合构件可以含有由组合构件实现的构件。

当一个组合构件用作构件实现时，其定义了可见性的边界。此组合构件中的构件不能被使用构件直接引用。使用构件只能连线到被使用组合构件的服务和引用，并可以为此组合构件的任意属性设置值。此组合构件的内部结构对使用它的构件是不可见的。

用作构件实现的组合构件也遵循完整的契约。组合构件的服务、引用以及属性形成了被使用构件所依赖的完全契约。组合构件的完全概念意味着：

- 组合构件必须有至少一个服务或至少一个引用。没有服务以及引用的构件在SCA中是没有意义的，因为其不能被连线到任何东西 – 此种构件既没有提供也没有消费任何服务。
- 组合构件提供的每个服务必须被连线到一个构件的服务或一个组合构件的引用。如果服务没有被连线，那么就意味着如果服务被调用，运行期将发生某个异常。

组合构件的component type是由一系列的service元素、reference元素以及property元素定义的。这些元素都是composite元素的子元素。

通过使用component元素的***implementation.composite***子元素，将组合构件用作构件实现。Implementation.composite元素的schema片段是：

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

   ...

   <component name="xs:NCName" autowire="xs:boolean"?
         requires="list of xs:QName"? policySets="list of xs:QName"?>*
         <implementation.composite name="xs:QName"/>?
         <service name="xs:NCName" requires="list of xs:QName"?
               policySets="list of xs:QName"?>*
               <interface/>?
               <binding uri="xs:anyURI" name="xs:QName"?
                     requires="list of xs:QName"
                     policySets="list of xs:QName"?/>*
               <callback>?
                     <binding uri="xs:anyURI"? name="xs:QName"?
                           requires="list of xs:QName"?
                           policySets="list of xs:QName"?/>+
         </callback>
         </service>
           <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
```

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

    ...

</composite>
```

implementation.composite元素有如下属性：

- ***name (必须)*** – 用作实现的组合构件名

*1.6.5.1*将组合构件作为构件实现使用的例子

在如下包含两个构件的组合构件的例子中，每个构件都由一个组合构件实现：

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

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

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

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

   <component name="AccountServiceComponent">
```

```
        <implementation.composite name="foo:AccountServiceComposite1"/>

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

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

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

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

</composite>
```

1.6.6 通过包含使用组合构件

为了辅助团队开发，组合构件也许以多个物理工件的形式开发，然后再将这些物理工件合并为单个的逻辑单元。

组合构件定义于一个***xxx.composite***文件中，并且组合构件也许通过包含其他的composite文件以接受附加的内容。

被包含的组合构件的语义是：通过在使用它们的组合构件中，将被包含的组合构件内容嵌入到使用它们的组合构件的**xxx.composite**文件中。其效果是文本的内含 – 即，用被包含的组合构件的文本内容替换掉使用它们的组合构件的include语句。在此处理过程中，被包含的composite元素本身被丢弃 – 只有其内容被包含进去。

用于内含（其它组合构件的）的composite文件可以有任意内容，但其总是包含单个***composite***元素。composite元素可以包含composite元素的任意有效的子元素，命名的component、service、reference、wire以及include元素。对于一个被包含的composite，其内容可以不完全，只要在使用此组合构件的composite中或在其他相关的被包含的composite文件中定义的工件可以被引用就可以了。比如，在一个composite文件中有两个构件，同时一个连线指定一个构件作为源，另一个构件（此构件定义于第二个被包含的composite文件中）作为目的，这是允许的。

如果由于内含（inclusion）导致组合构件无效，这是一个错误。比如，如果在容器组合构件（此处指包含组合构件的组合构件）中有完全相同的元素（如，带有相同uri的两个服务，而这两个服务由不同的被包含组合构件contribute出来）

如下片段为include元素展示了部分schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Include snippet -->
<composite    xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="xs:anyURI"
```

```
            name="xs:NCName" local="xs:boolean"? autowire="xs:boolean"?
            constrainingType="QName"?
            requires="list of xs:QName"? policySets="list of xs:QName"?>

    ...

    <include name="xs:QName"/>*

    ...

</composite>
```

include元素有如下属性：

- ***name (required)*** – the name of the composite that is included.
- ***name (required)*** – 被包含的组合构件名

1.6.6.1 被包含的组合构件例子

下图为包含四个组合构件的MyValueComposite2展示了装配图。***MyValueServices composite***包含了MyValueService service。***MyValueComponents composite***包含了MyValueServiceComponent、StockQuoteMediatorComponent以及它们之间的连线。***MyValueReferences composite***包含了CustomerService和StockQuoteService引用***MyValueWires composite***包含了将MyValueService service连接到MyValueServiceComponent的连线、将MyValueServiceComponent的customerService引用连接到CustomerService引用的连线以及将StockQuoteMediatorComponent的stockQuoteService引用连接到StockQuoteService引用的连线。注意，由一系列被包含的组合构件构建出MyValueComposite2只有一种可能的方式。

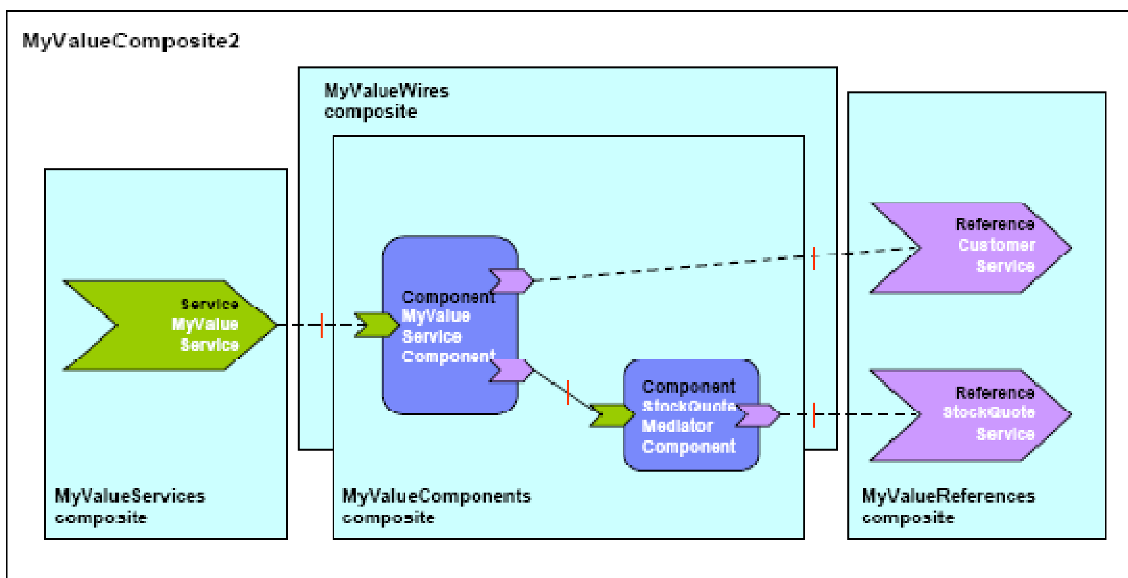

Figure 13 MyValueComposite2 built from 4 included composites

如下片段为通过使用被包含的组合构件创建的MyValueComposite2展示了MyValueComposite2.composite文件的内容。在这个例子中，其仅提供了组合构件名。composite文件自身可以用于使用被包含的组合构件来定义构件、服务、引用以及连线的场合。

```
<?xml version="1.0" encoding="ASCII"?>
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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>
```

如下片段展示了MyValueServices.composite文件的内容

```
<?xml version="1.0" encoding="ASCII"?>
<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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>
```

如下片段展示了MyValueComponents.composite文件的内容

```
<?xml version="1.0" encoding="ASCII"?>
<composite  xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="http://foo.com"
               xmlns:foo="http://foo.com"
               name="MyValueComponents" >

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

   <component name="StockQuoteMediatorComponent">
         <implementation.java class="services.myvalue.SQMediatorImpl"/>
         <property name="currency">EURO</property>
   </component>

<composite>
```

如下片段展示了MyValueReferences.composite文件的内容

```
<?xml version="1.0" encoding="ASCII"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="http://foo.com"
               xmlns:foo="http://foo.com"
               name="MyValueReferences" >

   <reference name="CustomerService"
          promote="MyValueServiceComponent/CustomerService">
          <interface.java interface="services.customer.CustomerService"/>
          <binding.sca/>
    </reference>

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

  </composite>
```

如下片段展示了MyValueWires.composite文件的内容

```
<?xml version="1.0" encoding="ASCII"?>
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               targetNamespace="http://foo.com"
               xmlns:foo="http://foo.com"
               name="MyValueWires" >

   <wire source="MyValueServiceComponent/stockQuoteService"
         target="StockQuoteMediatorComponent"/>

</composite>
```

1.6.7 包含多种类型构件实现的组合构件

包含多种类型构件的组合构件可能有多种构件实现的类型。比如，一个组合构件可以包含一个以Java POJO作为其实现的构件，以及另一个以BPEL流程作为其实现的构件。

1.6.8强制类型

SCA允许一个构件及其相关的实现被一个***constrainingType***限定。在SCA的自顶向下开发案例中，constrainingType元素提供了辅助功能。此开发案例中，架构师或装配者可以在任意实现被开发出来之前，定义组合构件的结构，包括构件实现的要求形式。

用一个带有service、reference以及property子元素的元素来描述一个constrainingType，并且其子元素上可以应用意图（intent）。constrainingType与任意的实现无关。因为与实现无关，所以其不能包含任意实现相关的配置信息或默认值。特别地，其不包含绑定（binding）、策略集（policySet）、属性值以及默认的连线信息。

constrainigType通过component上的constrainingType属性应用于此构件。

constrainingType为构件及其实现提供了一个“形”。任意指向了一个constrainingType的构件配置都受该“形”的限制。constrainingType指定服务、引用以及属性，且它们都必须被实现。这为实现人员提供了编程一系列特定的由constraingType定义的服务、引用以及属性的能力。所以，构件是已配置的实现实例，且受限于相关的constraingType。

如果此构件的配置或其实现与constraingType不一致，则是错误的。

constrainingType由***constrainingType***元素描述。如下片段为composite元素展示了伪schema

```
<?xml version="1.0" encoding="ASCII"?>
<!-- ConstrainingType schema snippet -->
<constrainingType xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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"?>*
         default-property-value?
   </property>

</constrainingType>
```

constrainingType元素有如下属性：

- name (必须) –constrainingType名。constrainingType名是一个XML QName的形式，其所在命名空间为targetNamespace属性所标识的命名空间。
- targetNamespace (可选) – constraingType声明所在目标命名空间的标识符
- requires (optional) –策略意图列表。此属性的描述请查看《策略框架规范》

constrainingType元素包含0个或多个property、service、reference子元素。

当一个实现被constrainingType限定，那么此实现必须定义constrainingType中指定的所有服务、引用以及属性。constrainingType的引用和服务有指定的接口也可能有指定的意图（intent）。一个实现可能包含附加的服务、附加可选的引用以及附加可选的属性，但不能包含附加却非可选的引用或附加却非可选的属性。（一个非可选的属性是一个没有默认值的属性）

当一个构件受限于一个constraingType（通过"constrainingType"属性）时，与此构件相关的整个componentType及其实现对于其组合构件（这里指包含受限于constraingType的构件的组合构件）是不可见的。其组合构件只能看见构件相关的componentType以及受限于constrainingType的构件实现的一个投影。比如，实现提供的一个附加服务（附加服务指不在构件相关的constraingType中的服务），不能被其组合构件提升。这个要求确保constrainingType契约不被组合构件（composite）改变。

constrainingType可以包含任意元素上的必须意图（required intent）。那些意图可以应用于使用此constraingType的任意构件。换言之，如果在constrainType元素或其service、reference子元素上存在requires="reliability"，那么被限定的构件或其实现必须在构件或实现或相应的服务、引用上包含requires="reliability"。注意：构件或实现可能会使用意图（intent）的限定形式（qualified form），而此意图在constrainingType中指定为非限定形式，但如果constrainigType使用了限定形式，那么构件或实现必须也用限定形式，否则就存在错误。（译者注：即要求构件或实现至少要受constraingType的限制，当然可以比constrainingType的限制更严格，意图的限定形式比非限定形式更进一步说明也更严格）

constraingType可以应用于一个实现。这种情况下，实现的componentType有一个设置为constrainngType QName的constraingType属性。

*1.6.8.1*强制类型例子

如下片段展示了名为"MyValueServiceComponent"的构件的内容，此构件受限于myns:CT的constraingType。与实现相关的constraingType（译者注：原文为componentType是错误的）也显示出来了。

```
<component name="MyValueServiceComponent" constrainingType="myns:CT>
  <implementation.java class="services.myvalue.MyValueServiceImpl"/>
  <property name="currency">EURO</property>
  <reference name="customerService" target="CustomerService">
    <binding.ws ...>
  <reference name="StockQuoteService"
      target="StockQuoteMediatorComponent"/>
</component>

<constrainingType name="CT"
            targetNamespace="http://myns.com">
  <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"/>
</constrainingType>
```

MyValueServiceComponent构件受限于constrainingType CT，即意味这必须提供：

- 带有services.myvalue.MyValueService 接口的***MyValueService***服务
- 带有services.customer.CustomerService(译者注：原文有误）接口的***customerService***引用
- 带有services.stockquote.StockQuoteService 接口的***stockQuoteService***引用

- 类型为xsd:string 的***currency***属性.

1.7 绑定

绑定被服务以及引用所使用。引用使用绑定来描述用于调用服务的访问机制（此服务可以是另一个SCA组合构件提供的服务）。服务使用绑定来描述客户端（可以是来自于另一个SCA组合构件的客户）要调用服务必须使用的访问机制。

SCA支持多种不同绑定类型的使用。举例来说，包括***SCA service,Web service, stateless session EJB, data base stored procedure***以及***EIS service。***SCA运行时必须为***SCA service***以及***Web service***绑定类型提供支持。SCA提供扩展机制。通过此扩展机制，SCA运行时可以添加对附加绑定类型的支持。附加绑定类型如何定义的细节，请查看扩展模型节。

绑定由binding元素定义，此binding元素为组合构件中service元素或reference元素的子元素。如下片段展示了带有binding元素schema的composite schema。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Bindings schema snippet -->
<composite     xmlns="http://www.osoa.org/xmlns/sca/1.0"
               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:QName"?
               requires="list of xs:QName"? policySets="list of xs:QName"?/>*
         <callback>?
               <binding uri="xs:anyURI"? name="xs:QName"?
                     requires="list of xs:QName"?
                     policySets="list of xs:QName"?/>+
         </callback>
   </service>

   ...

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

```
                        policySets="list of xs:QName"?/>+
            </callback>
      </reference>

      ...

</composite>
```

binding元素的元素名是构造的；其本身就是一个限定名。第一个限定符总是"binding"，第二个限定符分别为绑定类型（比如binding.composite、binding.ws、binding.ejb以及binding.eis）

binding元素有如下属性：

- ***uri (可选) –***有如下语义。
  - 对于reference的binding，此URI属性定义引用的目标URI（对于SCA域内部，端点的连线为component/service或SCA域外部某个端点的可访问地址）。对于在用作构件实现的组合构件中定义的引用来说，uri是可选的，但对于被contribute到SCA域的组合构件中定义的引用来说，uri则是必须的。组合构件的引用的URI属性可以被其上层组合构件（译者注：上层组合构件指将某个组合构件作为其实现的组合构件）中的某个构件重新配置。某些绑定类型也许要求目标服务的地址使用多个简单URI（比如一个WS-Addressing端点引用）。那些情形下，绑定类型将定义附加的属性或必要的用于标识服务的子元素。
  - 对于service的binding，此URI属性定义相对于构件的URI，此构件contribute服务到SCA域中。此URI的默认值为绑定的name属性值。
- ***name （可选） –***绑定实例名（一个QName）。name属性用于区分单个服务或引用上的多个binding元素。其默认值为服务名或引用名。但一个服务或引用有多个绑定，只能有一个绑定有默认值。所有其他的绑定name值在服务或引用中必须唯一。name属性也允许别处引用绑定实例 – 对于某些绑定类型特别有用，这些绑定类型在定义文档中作为模板声明，并且被其它的绑定实例引用，从而简化更复杂绑定实例的定义（此引用例子请查看《JMS绑定规范》[11]）。
- ***requires (可选)*** –策略意图列表。此属性描述请查看《策略框架规范》
- ***policySets (optional)*** –策略集列表。此属性描述请查看《策略框架规范》

当一个服务存在多个绑定时，则意味着以任何指定的绑定形式服务都是有效的。SCA运行时选择哪个有效绑定的技术则留给实现，且可能包含附加（或不标准）的配置。不管使用了什么技术都应该记录下来。

服务和引用总是在SCA域级别覆盖它们的绑定，除非被应用于它们的意图所限定了。

1.7.1 包含了并未在服务接口中定义的数据的消息

对于一个消息，可能包含并未在接口中定义的信息，此接口定义服务。比如实例信息可能被包含于SOAP头部信息中或作为MIME附件。

实现类型可能在其执行上下文中使得该信息对构件实现有效。这些实现类型必须指明此信息是如何被访问且以何种

形式呈现的。

1.7.2已部署绑定的 URI 形式

1.7.2.1 构建分层的URI

使用分层URI的绑定schema用如下片段的组合构建有效的URI：

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

每部分的附加定义如下：

Base Domain URI for a scheme.一个SCA域应该为每个提供服务的分层URI schema定义一个基URI（base URI）。
比如：HTTP和HTTPS schema，每个schema都有它们自己的为域而定义的基URI（base URI）。因为是非分层sheme，所以没有基URI（base URI）的例子有"jms:" schema。

Component URI. 上面的component URI是对于被部署于SCA域中的一个构件而言的。一个构件的URI默认为此构件的名字（name of the component），其作为相对URI使用。构件也可以有一个指定的URI值。此指定的URI值也许是一个绝对URI。当为绝对URI时，此URI对于所有属于该构件的所有服务而言则成为了基URI（Base URI）。如果此指定的URI值是一个相对URI，则用作如上的Component URI值。

Service Binding URI. Service Binding URI是相对URI，其在服务的binding元素的"uri"属性中被指定。此属性的默认值是binding的name属性值（此属性值看作是相对URI）。如果单个服务的多个绑定使用了相同的scheme（如HTTP），那么这些绑定中只有一个绑定依赖于uri属性的默认值，例如，只有一个使用默认的绑定名。Service Binding URI也可以是绝对URI，这时绝对URI完全地指定服务的完整URI（地址）。某些部署环境也许不支持在服务绑定中使用绝对URI。

在构件只有单个服务的地方，Service Binding URI的默认值是null，所以有效的URI为：

Base Domain URI for a scheme / Component URI

这种URI的简化形式与连线引用到服务时使用的连线target URI的简化形式一致。

被部署于域中的服务（而不是构件中的服务）有一个不包含构件名的URI，比如：

Base Domain URI for a scheme / Service Binding URI

上层组合构件（译者注：指包含构件的组合构件）的名字不contribute到任意服务的URI。

举个例子，基URI（Base URI）为"http://acme.com"，构件名为"stocksComponent"且服务绑定名为"getQuote"的一个服务，其URI看上去象这样：

http://acme.com/stocksComponent/getQuote

一个绑定的相对URI可以与服务名不同，允许不依赖于域组织而设计服务的URI层次。

将URI层次设计得与域组织无关是一个很好的实践，但很多时候域都是用默认URI层次初始化创建的。
这种情形下，通过为所选元素的uri属性设置相应值，可以改变域组织，同时管理URI层次的形式。

要将一个构件的服务子集（假设是"foo"）移动到一个新的构件（假设是"bar"），后者构件应该为移动来的服务的绑定指定一个URI "../foo/MovedService"。

为了为某些端点创建更简短的URI，也可以使用URI属性。而这些端点中，构件名根本不存在于URI中。 比如，如果绑定有一个"../myService"的uri属性，构件名并不存在于URI中。

1.7.2.2 非分层URI

使用非分层URI scheme的绑定（比如jms:或mailto:）可以可选地使用"uri"属性，此属性为服务绑定URI的完整描述。在绑定没有使用"uri"属性的地方，绑定必须为指定服务地址提供一个不同的机制。

1.7.2.3 判定一个被部署绑定的URI scheme

当创建一个被部署绑定的有效URI（比如，端点）时，需要判定的信息之一就是URI scheme。判定端点URI scheme的过程与绑定类型相关。

如果绑定类型支持单一协议，那么只有一个与之相关的URI scheme。这种情形下，就用那个URI scheme。

如果绑定类型支持多种协议，绑定类型的实现通过内省绑定的配置判定URI scheme。此绑定配置也许包含于绑定相关的策略集。

支持多种协议的绑定类型的好例子就是binding.ws。可以通过引用一个"抽象"WSDL元素（如portType或interface）或一个"具体"WSDL元素（如binding、port或endpoint）来配置binding.ws。当绑定引用了一个portType或Interface，那么协议以及URI scheme则派生自附加于绑定上的意图/策略集（intents/policy set）。当绑定引用了一个"具体"WSDL元素，则有两种情况：

1) 被引用的WSDL binding元素唯一地标识一个URI scheme. 这是最常见的情况。在这种情形下, URI scheme由在WSDL binding元素中指定的protocol/transport给定。

2) 被引用的WSDL binding并不唯一地标识一个URI scheme。例如，当在SOAP binding元素中指定的@transport 属性是HTTP时，“http”和“https”都可以用作有效的URI scheme。这种情形下,通过查找附加到绑定上的策略集来判定URI scheme。

值得注意的是：绑定类型所支持的意图可能完全改变绑定的行为。比如，当要求HTTP绑定的意图为"confidentiality/transport"时，SSL就开启了。这基本上将绑定的URI scheme从“http”改变为“https”。

1.7.3 SCA Binding

SCA binding元素由如下schema定义

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

SCA binding可以用于SCA域中引用和服务间的服务交互。SCA规范并未定义这种绑定类型实现的方式，不同的SCA运行时可以以不同的方式实现。仅有的要求是对于SCA binding类型必须实现要求的服务质量。SCA binding类型并不打算作为一种互操作性的绑定类型。对于互操作性，会使用可互操作的绑定类型，比如Web service绑定。

无binding元素的service或reference定义使用SCA binding。在覆盖的情况下，或当你在服务或引用的定义上指定了一系列绑定且SCA binding必须是其中之一时，才必须指定<binding.sca/>
（**译者注**：原文为：A service or reference 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 or reference definition and the SCA binding should be one of them.
）

如果服务或引用的接口是本地的，那么使用SCA binding的本地变种。如果服务或引用的接口是远程的，那么依赖于源与目的是否是协同定位的来使用SCA binding的本地或远程变种。（译者注：即根据判断源与目的是否在同一个JVM中来判断使用SCA binding的本地或远程变种）

如果一个引用通过其uri属性指定一个URI，那么这就提供了一个默认到服务的连线，而此服务由另一个域级别的构件提供。URI的值必须按如下形式：

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

（**译者注**：原文为：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:）

1.7.3.1 SCA binding例子

如下片段为包含MyValueService service元素以及StockQuoteService reference元素的MyValueComposite展示了MyValueComposite.composite文件。此服务以及引用都使用SCA binding。引用的target在这种绑定中没有定义，必须由其上层组合构件（此上层组合构件指使用了MyValueComposite组合构件的组合构件）提供。

```
<?xml version="1.0" encoding="ASCII"?>
<!-- Binding SCA example -->
<composite      xmlns="http://www.osoa.org/xmlns/sca/1.0"
                targetNamespace="http://foo.com"
                name="MyValueComposite" >

   <service name="MyValueService" promote="MyValueComponent">
         <interface.java interface="services.myvalue.MyValueService"/>
         <binding.sca/>
         …
   </service>

   …

   <reference name="StockQuoteService"
promote="MyValueComponent/StockQuoteReference">
         <interface.java interface="services.stockquote.StockQuoteService"/>
         <binding.sca/>
   </reference>

</composite>
```

1.7.4 Web Service 绑定

SCA定义了Web service绑定。其在单独的规范[9]中描述。

1.7.5 JMS 绑定

SCA定义了JMS绑定。其在单独的规范[11]中描述。

1.8 SCA 定义

存在多种SCA工件，它们一般都是有用的，且并没有指定到特定的组合构件或构件。这些共享的工件包括意图、策略集、绑定、绑定类型定义以及实现类型定义。

SCA域中所有的这些工件定义在一个全局的名为definintions.xml的SCA域范围的文件中。definitions.xml文件包含一个与如下伪schema片段一致的definitions元素：

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

```
<definitions    xmlns="http://www.osoa.org/xmlns/sca/1.0"
                targetNamespace="xs:anyURI">

   <sca:intent/>*

   <sca:policySet/>*

   <sca:binding/>*

   <sca:bindingType/>*

   <sca:implementationType/>*

</definitions>
```

definitions元素有如下属性：

- ***targetNamespace (required)*** – 该definitions元素的子元素所在的命名空间（用于工件解析过程）

definitions元素包含可选的子元素 – intent、policySet、binding、bindingtype以及implementationType。这些元素在此规范的其他地方或《SCA策略框架规范》[10]中描述。在《SCA策略规范》和《JMS绑定规范》[11]中描述definitions元素中声明的这些元素的使用。

1.9 扩展模型

装配规范可以被扩展，以支持新的接口类型、实现类型以及绑定类型。扩展模型基于XML schema置换组。在SCA命名空间中存在三个XML schema置换组的头部：***interface***、***implementation***以及***binding***，分别相应于接口类型、实现类型以及绑定类型。

《SCA客户程序以及实现规范》和《SCA绑定规范》[1]运用这些XML schema置换组定义某些基本的接口类型、实现类型以及绑定类型，而其他类型可以根据需求定义，为运行时提供支持。SCA规范定义的接口类型元素、实现类型元素以及绑定类型元素在它们各自的schema中都在SCA命名空间（"http://www.osoa.org/xmlns/sca/1.0"）中。新的以此扩展模型定义的接口类型、实现类型以及绑定类型，不属于这些SCA规范部分的，必须定义在非SCA命名空间中。

"."号用于命名SCA规范定义的元素（比如<implementation.java ... />、<interface.wsdl ... />、<binding.ws ... />），其并不是一种并行扩展的方式，而是一种改善SCA装配语言用法的命名约定。

注意：如何contribute SCA模型扩展和它们的运行时功能到一个SCA运行时由此规范的后续版本定义。

1.9.1定义一个接口类型

如下片段为***sca-core.xsd*** 中的interface元素以及接口类型展示一个基本的定义，至于完整的schema，请查看附录。

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

   ...

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

   ...

</schema>
```

如下片段为我们展示了如何扩展此基本定义以支持Java接口。该片段展示了***interface.java***元素的定义以及***sca-interface-java.xsd***中的***JavaInterface***类型。

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

  <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>
```

如下片段为我们展示了一个如何用其他规范扩展此基本定义以支持SCA规范中未定义的新的接口类型的案例。该片段展示了***my-interface-extension***元素的定义以及***my-interface-extension-type***类型。

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

1.9.2定义一个实现类型

如下片段为***implementation***元素以及***sca-core.xsd***中的***Implementation***类型展示了基本定义。至于完整的schema请查看附录。

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

   ...

    <element name="implementation" type="sca:Implementation" abstract="true"/>
    <complexType name="Implementation"/>

   ...

</schema>
```

如下片段为我们展示了如何扩展基本定义以支持Java实现。该片段展示了***implementation.java***元素以及***sca-implementation-java.xsd***中的***JavaImplementation***类型。

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

   <element name="implementation.java" type="sca:JavaImplementation"
                              substitutionGroup="sca:implementation"/>
   <complexType name="JavaImplementation">
         <complexContent>
               <extension base="sca:Implementation">
                     <attribute name="class" type="NCName" use="required"/>
               </extension>
         </complexContent>
   </complexType>
</schema>
```

如下片段为我们展示了一个如何用其他规范扩展此基本定义以支持SCA规范中未定义的新的实现类型的案例。该片段展示了***my-impl-extension***元素的定义以及***my-impl-extension-type***类型。

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

```
        xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
       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>
```

除了新实现实例元素的定义外，还必须存在相关的为新实现类型提供元数据的implementationType元素。implementationType元素的伪schema显示如下：

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

implementation type有如下属性：

- ***type (必须) –实现***（此实现应用了此implementationType元素）的类型。对于实现类型，要求为implementation元素的QName，比如"sca:implementation.java"
- ***alwaysProvides (可选)*** – 实现类型总能提供的意图集。细节请查看《策略框架规范》
- ***mayProvide (可选) –***实现类型也许能提供的意图集。细节请查看《策略框架规范》

1.9.3定义一个绑定类型

如下片段为***binding***元素以及***sca-core.xsd***中的Bding类型展示了基本定义。完整的schema请查看附录。

```
<?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://www.osoa.org/xmlns/sca/1.0"
        xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
        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>
```

如下片段为我们展示了如何扩展基本定义以支持Web service绑定。该片段展示了***binding.ws***元素以及***sca-binding-webservice.xsd***中***WebServiceBinding***类型的定义。

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

   <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>
```

如下片段为我们展示了一个如何用其他规范扩展此基本定义以支持SCA规范中未定义的新的绑定的案例。该片段展示了***my-binding-extension***元素的定义以及***my-binding-extension-type***类型。

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

   <element name="my-binding-extension" type="tns:my-binding-extension-type"

        substitutionGroup="sca:binding"/>
  <complexType name="my-binding-extension-type">
        <complexContent>
              <extension base="sca:Binding">
                    ...
              </extension>
        </complexContent>
  </complexType>
</schema>
```

除了新绑定实例元素的定义外，还必须存在相关的为新绑定类型提供元数据的bindingType元素。bindingType元素的伪schema显示如下：

```
<bindingType type="xs:QName"
          alwaysProvides="list of intent QNames"?
          mayProvide = "list of intent QNames"?/>
```

binding type有如下属性：

- ***type (必须) –绑定***（此绑定应用了该bindingType元素）的类型。对于绑定类型，要求为binding元素的QName，比如"sca:binding.ws"
- ***alwaysProvides (可选)*** – 绑定类型总能提供的意图集。细节请查看《策略框架规范》
- ***mayProvide (可选)*** –绑定类型也许能提供的意图集。细节请查看《策略框架规范》

1.10 打包以及部署

1.10.1 域

一个***SCA Domain***描述一个完整的运行时配置，潜在地可以被部署到一系列内连的运行时节点上。

一个SCA单域为所有的SCA机制定义了可见性边界。比如，SCA连线只能用于连接SCA单域中的构件。连接到域外部的服务必须使用特定的有服务寻址功能的绑定机制（比如WSDL端点URI）。SCA机制，比如意图策略集，只能用在SCA单域的上下文中。一般说来，用SCA开发以及部署的服务，其外部客户应该不能区分出服务是使用SCA实现的– 这是实现细节。

一个SCA域的范围大小以及配置不受SCA装配规范的限制，并认为是高度可变性的。典型地，一个SCA域代表一个由单个组织控制的业务功能的区域。比如，一个SCA域可以是一个商务的全部，也可能是商务里的一个部门。

举例来说，对于商务中的会计部门，SCA域可能覆盖所有金融相关的功能，也可能包含一系列处理特定帐务领域的组合构件，一个处理客户帐户，另一个处理应付帐户。

一个SCA域有如下内容：

- 一个虚拟的域级组合构件，其构件被部署且运行
- 一系列被安装的contribution，其包含实现、接口以及执行构件必须的其它工件
- 一系列操作contriubtion集合以及虚拟域级组合构件的逻辑服务

SCA域相关信息有多种存储方式，可以是但不限定于特定的文件系统或者存储库。

1.10.2 Contributions

一个SCA域也许要求许多不同的工件才能运行。这些工件包括SCA定义的工件以及其它工件，如目标代码文件和接口定义文件。SCA定义的工件类型全都是XML文档。不同的SCA定义文档的根元素是：composite、componentType、constrainingType以及definitions。非SCA定义的但一个SCA域又需要的XML工件包括XML schema文件、WSDL文件以及BPEL文件。SCA构造（construct），象其它的XML定义的构造(construct)，使用XML限定名标识自己（比如名称空间+本地名）。

一个SCA域内也要求非XML的工件。这些非XML工件最明显的例子就是java、C++以及其它对于构件实现必须的其他编程语言文件。因为SCA是可扩展的，所以可以需要其它的XML或非XML的工件。

SCA为contribution定义了一个可互操作的打包格式（ZIP），此格式在后面指定。此格式不是SCA运行时可以使用的唯一打包格式。SCA允许多种不同的打包格式，但要求支持ZIP格式。当使用ZIP格式部署一个contribution时，此规范并不指定在被部署后是否还保留该格式。比如，基于SCA运行时的Java EE也许会将ZIP包转换为一个EAR包。SCA预期任何包的某些特性如下：

- 将包工件作为基于单根的资源层级呈现给SCA，必须是可能的。
- 目录资源必须存在于名为"META-INF"的层级根目录下
- META-INF目录下必须直接存在一个名为sca-contribution.xml的文档，此文档列出可运行的contribution中的SCA组合构件。

同一个文档（译者注：此处指sca-contribution.xml文档）也可选地列出contribution中定义的构造（construct）的命名空间，且其他contribution可能用到这些命名空间。可选地，附加的元素可能存在于contribution需要的构造（construct）的命名空间中，必须在其它地方发现，比如在其它的contribution中。这些可选的元素可能并不物理地存在于包中，但可能基于已存在的定义或索引而产生，或者如果不存在可解析的引用，它们根本就不存在。
（译者注：原文为：The same document also optionally lists namespaces of constructs that are defined within the contribution and which may be used by other contributions Optionally, additional elements may exist that list the namespaces of constructs that are needed by the contribution and which must be found elsewhere, for example in other contributions. These optional elements may not be physically present in the packaging,but may be generated based on the definitions and references that are present, or they may not exist at all if there are no unresolved references.）

此文件（译者注：此处指sca-contribution.xml文件）的细节，请查看" SCA contribution元数据文档"节

为了举出各种SCA使用的各种包格式，可能用于将SCA工件以及元数据打包为一个contribution的格式例子：
- 一个文件系统目录
- 一个OSGi bundle
- 一个压缩的目录（zip, gzip等）
- 一个JAR文件（或其变种 – WAR, EAR等）

contribution不包含其它的contriubtion。如果包格式是一个包含其它JAR文件的JAR文件，那么其内部文件（译者注：这里指被包含于其中的其它JAR文件）不作为单独的SCA contribution对待。由实现决定内部文件是否应该描述为contribution层级中的单一工件或者其内容的全部（译者注：此处指内部文件内容的全部）是否应该描述为隔离的工件（译者注：即其内部文件中的各个文件一一对应各自的工件）。

SCA部署方式的目标是： contribution的内容不需要被修改就可以在一个域中安装并使用contribution的内容。

1.10.2.1 SCA工件解析

contribution可能是自包含的，即在该contribution自身中就可以找到运行contribution内容所有必须的工件。然而，也有情况是contribution的内容引用了不包含于其contribution中的一个或多个工件。这些引用可能是对SCA工件的引用，也可能是对其它工件的引用，比如WSDL文件、XSD文件或代码工件，如Java类文件和BPEL脚本。

一个contribution可能使用一些工件相关的或包相关的方式来解析工件的引用。这些机制的例子包括：

- wsdlLocation和schemaLocation属性分别对WSDL和XSD schema工件的引用
- OSGi bundle解析Java类和相关资源依赖关系的机制

如果存在类似的场景，这些机制用来解析工件的依赖关系。

SCA也提供一个工件的解析机制。SCA工件解析机制用于无其它有效机制或同一SCA域中被不同的contribution使用的机制不同的情况下。后者情况的一个例子是OSGI Bundle用于第一个contribution，而此contribution使用的第二个contriubtion并非使用OSGI实现的 – 比如第二个contribution是一个大型机的COBOL服务，其接口以WSDL声明用于第一个contribution的访问。

SCA工件解析对于包含异构contribution的SCA域来说可能是最有用处的。在这种SCA域中工件相关或包相关机制不可能跨越不同种类的contribution工作。

SCA工件解析的工作原则是一个需要用到在其它地方定义的工件的contribution在属于此contribution的元数据中运用import语句来表达这种依赖关系。一个contribution通过附加到此contribution的元数据中的***export***语句来控制其工件对其它contribution的可见性（译者注：即将其工件导出，以便其它contribution可以引用这些工件）。

1.10.2.2 SCA contribution元数据文档

contribution可选地包含一个文档，且此文档声明了可运行组合构件、导出定义以及导入定义。此文档在相对contribution根目录的META-INF/sca-contriubtion.xml路径下。经常地，有些SCA元数据需要手工指定，而有些元数据由工具生成（比如下面讲到的<import>元素）。为此，可能在META-INF/sca-contribution-generated.xml路径下存在一个结构化的文档。如果此文档存在（或按需生成的），其内容将合并到sca-contribution.xml文件的内容，如果两个文件内容存在任何的冲突声明，sca-contribution.xml里的条目（entry）优先。

此文档的格式是：

```
<?xml version="1.0" encoding="ASCII"?>
<!-- sca-contribution pseudo-schema -->
<contribution xmlns=http://www.osoa.org/xmlns/sca/1.0>

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

```
</contribution>
```

deployable元素：标识contribution中的一个组合构件，此组合构件潜在地被包含进虚拟域级组合构件。此contribution中的其它组合构件则不试图被包含进虚拟域级组合构件，而只是被其它组合构件使用而已。在一个contribution被安装后，通过使用增加部署组合构件（add Deployment Composite）功能以及增加到域级组合构件（add To Domain Level Composite）功能可以创建新的组合构件。

- ***composite (必须)*** – contribution中组合构件的QName。

export元素：声明属于一个特定命名空间的工件被导出以便其它contribution使用。一个contribution中的一个export声明指定一个命名空间，而其所有的定义被认为是要被导出的。默认情况下，定义不被导出。

对于包含异构的contribution包和技术的SCA域来说，（这里工件相关或包相关机制不可能跨越不同种类的contribution工作），SCA工件的导出（export）是有用处的。

- ***namespace (必须)*** – 对于XML定义，用QName标识，namespace应该是被导出的定义的命名空间URI。对于定义了在一个命名空间中可以使用多个符号空格（*symbol space*）的XML技术来说（比如WSDL port type是与WSDL binding不同的符号空格），所有符号空格的所有定义被导出。

  使用naming scheme而不是QName的技术必须使用一个与同一置换组不同的export元素作为SCA <export>元素。使用的元素标识此技术，且可能使用为此技术使用相应的命名空间值。比如，<export.java>可以用于导出java定义，这时命名空间应该是一个全限定的包名。

Import element: Import声明指定导入此contribution中定义以及实现所需要的定义的命名空间，而这些命名空间不存在于此contribution中。大多数情况下，认为基于对contribution内容的内省，产生import声明。这种情形下，import声明可以在META-INF/ sca-contribution-generated.xml文档中找到。

- ***namespace (必须)* –** 对于由QName标识的XML定义，namespace应该是被导入定义的命名空间URI。对于定义了在一个命名空间中可以使用多个符号空格（symbol space）的XML技术来说（比如WSDL port type是与WSDL binding不同的符号空格），所有符号空格的所有定义被导入。

  使用naming scheme而不是QName的技术必须使用一个与同一置换组不同的import元素作为SCA <import>元素。使用的元素标识此技术，且可能使用为此技术使用相应的命名空间值。比如，<import.java>可以用于导入java定义，这时命名空间应该是一个全限定的包名。

- ***location (optional)*** – 一个解析此import定义的URI。SCA对此URI的形式以及解析的方式没有规定。也许（通过此URI）指向另一个contribution，或完全指向SCA域外部的某个位置。

期望SCA运行时也许会定义实现相关的方式，解析contribution间工件解析的位置信息。然而这些机制通常受限于一种运行时技术和一种宿主环境（hosting environment）的contribution集合。

为了满足完全异构的运行时技术的contribution间工件的导入，强烈推荐将SCA contribution URI作为位置指定。

支持跨contribution解析SCA工件的contribution URI的SCA运行时，应该在使用了@schemaLocation、@wsdlLocation以及其它的工件位置指定时简单地完成解析工作。

import语句导入的顺序在此机制中可能发挥作用。因为一个命名空间的定义可以跨工件分发，所以多import声明可以组成一个命名空间。

location值只能有一个值，且如果存在冲突，在installContribution调用中列出的依赖的contribution应该可以覆盖其值。然而，解析contribution（这些contribution存在冲突的定义）间冲突的机制与实现相关。

如果location属性的值是一个SCA contribution URI，那么contribution包可能依赖于部署的环境。为了避免此依赖，只能在部署或升级contribution（这些操作在对contribution的操作节指定）时指定依赖的contribution。

1.10.2.3 使用ZIP打包的contribution

SCA允许SCA运行时支持多种不同的打包格式，但SCA要求所有运行时支持contribution的ZIP打包格式。此格式允许存在SCA contribution元数据文档节指定的元数据。特别地，此包中可能包含一个"META-INF"的顶级目录以及一个"META-INF/sca-contribution.xml"的文件，同时也可能存在一个可选的" META-INF/sca-contribution-generated.xml"文件。SCA定义的工件和非SCA定义的工件，如目标文件、WSDL定义、Java类都可以存在于ZIP包的任何地方。
ZIP文件格式的最新定义由PKWARE发布，位置为an Application Note on the .ZIP file format [12].。

1.10.3 已安装的 contribution

以上章节提到了，contribution的内容应该不需要被修改就可以在一个域中安装并使用。被安装的contribution是带有所有相关必要信息以执行contribution中部署的组合构件的一个contribution。

一个已安装的contribution由如下内容组成:

- contribution包装 – 此contribution将作为解析所有引用的起点
- contribution 基础URI（base URI）
- 依赖的contribution: 一系列其它contribution的快照（snapshot）。这些contribution用于从根组合构件以及其它依赖的contribution中解析import语句。
  - o 依赖的contribution可能被其它安装的contribution共享也可能不被其它安装的contribution共享
  - o 当任意contribution的快照是实现定义的，广泛搜索contribution被安装到执行期间的时间。
- 部署时期的组合构件

这些是被安装的contribution已经部署后添加到该contribution的那些组合构件。它使得不需要修改contribution而提供最终配置以及访问contribution中的实现成为可能。这些是可选的，因为已存在于contribution中的组合构件也可以用于部署。
（译者注：原文为：

An installed contribution is made up of the following things:

- Contribution Packaging – the contribution that will be used as the starting point for resolving all references
- Contribution base URI
- Dependent contributions: a set of snapshots of other contributions that are used to resolve the import statements from the root composite and from other dependent contributions
    - o Dependent contributions may or may not be shared with other installed contributions.
    - o When the snapshot of any contribution is taken is implementation defined, ranging from the time the contribution is installed to the time of execution
- Deployment-time composites.

These are composites that are added into an installed contribution after it has been deployed. This makes it possible to provide final configuration and access to implementations within a contribution without having to modify the contribution. These are optional, as composites that already exist within the contribution may also be used for deployment.
）
被安装的contribution提供了一个上下文。在此上下文中解析限定名（如XML中的QName，java中的全限定的类名（译者注：此处指含全包名的java类名））

如果多个依赖的contribution有含有冲突限定名的导出定义，那么用于判定限定名的算法依赖于实现。如果存在冲突的名字， SCA的实现也可以生成一个错误。

1.10.3.1 已安装的工件URI

当安装一个contribution时，contribution中的所有工件都分配了URI，这些URI由此contribution的基URI开头，后加上每个工件的相对URI构造而成（回忆下，SCA要求任意包格式都能在一个单层次中贡献其工件）

1.10.4 对 contribution 的操作

SCA域提供与contribution相关的如下逻辑功能（意味着此功能也许不以可寻址的服务方式呈现，也意味着以其它方式提供等效的功能）。此功能是可选的，意味着某些SCA运行时可以选择不以任意方式提供该功能。

1.10.4.1 contribution的安装和更新

用一个提供的根contribution创建或更新一个被安装的contribution，并安装在一个给定的基URI（base URI）上。一个给定的依赖的contribution列表指定了用于解析根contribution以及其它依赖的contribution的依赖关系的contribution（译者注：此contribution列表指定了被根contribution和其他依赖的contribution所依赖的contribution）。这些（译者注：这里指此contribution列表）覆盖通过contribution的import语句中location属

性显式列出的任意依赖的contribution。

SCA允许简单的假设。此假设是使用一个解析任意信息的contribution也就意味着从该contribution导出的所有其它工件都能被使用。因此，该依赖的contribution列表只是一个被安装contribution URI的一个列表。没有必要指出互相所使用的是什么。

每个依赖的contribution也是一个被安装的contribution，同时带有其自身依赖的contribution。默认情况下，这些依赖的contribution的所依赖的contribution（我们称之为间接依赖的contribution）作为被安装contribution的依赖的contribution被包含。然而，如果在依赖的contribution列表中列出的一个contribution导出了任意与间接依赖的contribution相冲突的定义，则不包含间接依赖的contribution（比如，显式列表覆盖间接依赖的contribution的默认包含）。也有两个间接依赖的contribution之间存在冲突的情况，那么必须由依赖的contribution列表中的一个显式条目（entry）来解决冲突（译者注：即指以显式条目指定的内容为准）。
（译者注：原文为：
Each dependent contribution is also an installed contribution, with its own dependent contributions. By default these dependent contributions of the dependent contributions (which we will call indirect dependent contributions) are included as dependent contributions of the installed contribution. However, if a contribution in the dependent contribution list exports any conflicting definitions with an indirect dependent contribution, then the indirect dependent contribution is not included (i.e. the explicit list overrides the default inclusion of indirect dependent contributions). Also, 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.
）

注意：许多情况下，依赖的contribution列表可以被生成。特别地，如果一个域关注避免创建相同限定名的重复定义，那么通过工具产生此列表则很容易。
（译者注：原文为：
Note that in many cases, the dependent contribution list can be generated. In particular, if a domain is careful to avoid creating duplicate definitions for the same qualified name, then it is easy for this list to be generated by tooling.
）

1.10.4.2 部署组合构件的添加和更新

通过使用一个给定的组合构件（"composite by value" – 一个数据结构，并不是域中的某个已存在的资源）将一个部署组合构件添加或更新到由一个给定的contribution URI标识的contribution中。被添加或更新的部署组合构件被给定了一个相对URI，匹配组合构件的@name属性，并以“.composite”为后缀。因为所有的组合构件必须运行于一个被安装的contribution的上下文中（此contribution中解析任意构件实现或其它定义），这个特性使得对于部署者来说，无需修改根contribution的内容就能创建一个带有最终配置信息以及连线信息的组合构件并将其添加到一个被安装的contribution中成为可能。

也有，在某些用例中，contribution可能只包含实现代码（比如PHP脚本）。那么由一个组合构件（可能是生成的）为那些（实现代码）给出构件名，而无需修改包是可能的。此组合构件被添加到被安装的contribution中。

1.10.4.3 删除contribution

删除由给定contribution URI所标识的被部署的contribution。

1.10.5 对已存在（非 SCA）的工件解析机制的使用

对于某些工件种类，存在已有的通用机制以索引一个特定的具体位置，此位置上的工件可以被解析。

这些机制的例子包括：

- 对于WSDL文件， ***@wsdlLocation***属性指示了一个指向持有WSDL自身内容的地址URI值
- 对于XSD, ***@schemaLocation***属性指示了一个匹配XSD地址URI的命名空间

注意：这种情形下，运行时必须使用位置信息且URI必须是可索引的。

SCA允许这些机制的使用。当存在这些机制时，这些机制优先于SCA机制。然而，不鼓励使用这些机制，因为以这种方式寻址装配集会使得装配集不够灵活，且当整个SCA域发生变化时更容易出错。（译者注：这些机制是指已存在的工件解析机制）

注意：如果这些机制之一存在，但当使用此机制时查找指定资源出现了错误（比如，URI不正确或无效），那么SCA运行时必须抛出一个错误，且必须不去试图将SCA资源机制作为后备机制。

1.10.6 域级组合构件

域级组合构件是一个虚拟的组合构件，其并非由一个composite定义文档所定义。相反地，其通过域上的操作进行建立并修改。然而，在其它方面，它很象是一个组合构件，因为它包含构件、连线、服务以及引用。

修改域级组合构件的抽象域级功能如下，尽管一个运行时可能以不同的形式提供等效的功能：

1.10.6.1 添加到域级组合构件中

此功能添加由一个给定的URI标识的组合构件到域级组合构件中。该给定的组合构件URI必须索引到一个被安装的contribution中的一个组合构件。由此组合构件所在的被安装的contribution决定如何解析该组合构件的工件（直接或间接地）。给定的组合构件被添加到域级组合构件中，其语境是域级组合构件有一个引用了该给定的组合构件的<include>语句。此组合构件（译者注：此处指被添加到域级组合构件中的组合构件）的所有构件成为顶级构件，且服务变为了外部可见的服务（比如，它们可以存在于一个域的WSDL描述中）。

1.10.6.2 从域级组合构件中删除

从域级组合构件中删除一个元素，此元素由一个给定的组合构件URI标识。这意味着通过被标识的组合构件将原

先被添加到域级组合构件中的构件、连线、服务以及引用进行删除。

1.10.6.3 获取域级组合构件

返回一个<composite>定义，此定义对于每个被添加到域级组合构件的组合构件都有一个<include>行。特别注意，必须根据那个被安装的组合构件来解析被包含的组合构件以及任意被引用的工件。

1.10.6.4 获取QName定义

为了使得域级组合构件（通过获取域级组合构件的返回值）有意义，必须使得获取被包含的组合构件中的命名工件的定义成为可能。这个功能使用一个被安装的contribution（其提供上下文）的相应URI、一个要查找的定义的相应限定名以及给定的符号空格（作为一个QName，如wsdl:PortType）。其结果是一个单一定义，不管以任何形式都适合那个定义类型。

注意：这个，象所有其它域级操作一样，都是一个逻辑操作。其功能应该以某种形式存在，但并非必要是带有此精确签名（译者注：操作的签名指操作名、参数类型以及参数顺序以及返回值）的一个服务操作。

2. 附录 1

2.1 XML Schemas

2.1.1 sca.xsd

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

    <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"/>
```

```
</schema>
```

2.1.2 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://www.osoa.org/xmlns/sca/1.0"
        xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
        elementFormDefault="qualified">

    <element name="componentType" type="sca:ComponentType"/>
    <complexType name="ComponentType">
      <sequence>
            <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
            <choice minOccurs="0" maxOccurs="unbounded">
                  <element name="service" type="sca:ComponentService" />
                  <element name="reference" type="sca:ComponentReference"/>
                  <element name="property" type="sca:Property"/>
            </choice>
            <any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
        </sequence>
        <attribute name="constrainingType" type="QName" use="optional"/>
        <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <element name="composite" type="sca:Composite"/>
    <complexType name="Composite">
         <sequence>
             <element name="include" type="anyURI" minOccurs="0"
                  maxOccurs="unbounded"/>
             <choice minOccurs="0" maxOccurs="unbounded">
                 <element name="service" type="sca:Service"/>
                 <element name="property" type="sca:Property"/>
                 <element name="component" type="sca:Component"/>
                 <element name="reference" type="sca:Reference"/>
                 <element name="wire" type="sca:Wire"/>
             </choice>
             <any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
        </sequence>
        <attribute name="name" type="NCName" use="required"/>
        <attribute name="targetNamespace" type="anyURI" use="required"/>
        <attribute name="local" type="boolean" use="optional" default="false"/>
        <attribute name="autowire" type="boolean" use="optional" default="false"/>
        <attribute name="constrainingType" type="QName" use="optional"/>
        <attribute name="requires" type="sca:listOfQNames" use="optional"/>
        <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
        <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>

    <complexType name="Service">
```

```
  <sequence>
        <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
        <element name="operation" type="sca:Operation" minOccurs="0"
              maxOccurs="unbounded" />
        <choice minOccurs="0" maxOccurs="unbounded">
              <element ref="sca:binding" />
              <any namespace="##other" processContents="lax"
                    minOccurs="0" maxOccurs="unbounded" />
        </choice>
        <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
        <any namespace="##other" processContents="lax" minOccurs="0"
              maxOccurs="unbounded" />
  </sequence>
  <attribute name="name" type="NCName" use="required" />
  <attribute name="promote" type="anyURI" use="required" />
  <attribute name="requires" type="sca:listOfQNames" use="optional" />
  <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
  <anyAttribute namespace="##any" processContents="lax" />
</complexType>

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

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

<complexType name="SCAPropertyBase" mixed="true">
  <!-- mixed="true" to handle simple type -->
  <sequence>
        <any namespace="##any" processContents="lax" minOccurs="0"
              maxOccurs="1" />
        <!-- NOT an extension point; This xsd:any exists to accept
               the element-based or complex type property
               i.e. no element-based extension point under "sca:property" -->
  </sequence>
</complexType>

<!-- complex type for sca:property declaration -->
```

```
<complexType name="Property" mixed="true">
   <complexContent>
         <extension base="sca:SCAPropertyBase">
            <!-- extension defines the place to hold default value -->
            <attribute name="name" type="NCName" use="required"/>
            <attribute name="type" type="QName" use="optional"/>
            <attribute name="element" type="QName" use="optional"/>
            <attribute name="many" type="boolean" default="false"
              use="optional"/>
            <attribute name="mustSupply" type="boolean" default="false"
              use="optional"/>
            <anyAttribute namespace="##any" processContents="lax"/>
            <!-- an extension point ; attribute-based only -->
         </extension>
   </complexContent>
</complexType>

<complexType name="PropertyValue" mixed="true">
   <complexContent>
      <extension base="sca:SCAPropertyBase">
         <attribute name="name" type="NCName" use="required"/>
         <attribute name="type" type="QName" use="optional"/>
         <attribute name="element" type="QName" use="optional"/>
         <attribute name="many" type="boolean" default="false"
              use="optional"/>
         <attribute name="source" type="string" use="optional"/>
         <attribute name="file" type="anyURI" use="optional"/>
         <anyAttribute namespace="##any" processContents="lax"/>
         <!-- an extension point ; attribute-based only -->
      </extension>
   </complexContent>
</complexType>

<element name="binding" type="sca:Binding" abstract="true"/>
<complexType name="Binding" abstract="true">
  <sequence>
        <element name="operation" type="sca:Operation" minOccurs="0"
              maxOccurs="unbounded" />
  </sequence>
    <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>

<element name="bindingType" type="sca:BindingType"/>
<complexType name="BindingType">
  <sequence minOccurs="0" maxOccurs="unbounded">
        <any namespace="##other" processContents="lax" />
  </sequence>
  <attribute name="type" type="QName" use="required"/>
  <attribute name="alwaysProvides" type="sca:listOfQNames" use="optional"/>
  <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
  <anyAttribute namespace="##any" processContents="lax"/>
</complexType>

<element name="callback" type="sca:Callback"/>
<complexType name="Callback">
  <choice minOccurs="0" maxOccurs="unbounded">
```

```
        <element ref="sca:binding"/>
        <any namespace="##other" processContents="lax"/>
    </choice>
    <attribute name="requires" type="sca:listOfQNames" use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
    <anyAttribute namespace="##any" processContents="lax"/>
</complexType>

<complexType name="Component">
    <sequence>
         <element ref="sca:implementation" minOccurs="0" maxOccurs="1"/>
         <choice minOccurs="0" maxOccurs="unbounded">
              <element name="service" type="sca:ComponentService"/>
            <element name="reference" type="sca:ComponentReference"/>
            <element name="property" type="sca:PropertyValue" />
         </choice>
         <any namespace="##other" processContents="lax" minOccurs="0"
              maxOccurs="unbounded"/>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="autowire" type="boolean" use="optional" default="false"/>
    <attribute name="constrainingType" type="QName" use="optional"/>
    <attribute name="requires" type="sca:listOfQNames" use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
    <anyAttribute namespace="##any" processContents="lax"/>
</complexType>

<complexType name="ComponentService">
  <complexContent>
        <restriction base="sca:Service">
                <sequence>
                     <element ref="sca:interface" minOccurs="0" maxOccurs="1"/>
                     <element name="operation" type="sca:Operation" minOccurs="0"
                          maxOccurs="unbounded" />

                      <choice minOccurs="0" maxOccurs="unbounded">
                         <element ref="sca:binding"/>
                           <any namespace="##other" processContents="lax"
                                 minOccurs="0" maxOccurs="unbounded"/>
                      </choice>
                      <element ref="sca:callback" minOccurs="0" maxOccurs="1"/>
                      <any namespace="##other" processContents="lax" minOccurs="0"
                           maxOccurs="unbounded"/>
                 </sequence>
                 <attribute name="name" type="NCName" use="required"/>
                 <attribute name="requires" type="sca:listOfQNames"
                     use="optional"/>
                 <attribute name="policySets" type="sca:listOfQNames"
                     use="optional"/>
                 <anyAttribute namespace="##any" processContents="lax"/>
         </restriction>
   </complexContent>
 </complexType>

 <complexType name="ComponentReference">
   <complexContent>
         <restriction base="sca:Reference">
               <sequence>
                     <element ref="sca:interface" minOccurs="0" maxOccurs="1" />
```

```
                        <element name="operation" type="sca:Operation" minOccurs="0"
                              maxOccurs="unbounded" />
                        <choice minOccurs="0" maxOccurs="unbounded">
                              <element ref="sca:binding" />
                              <any namespace="##other" processContents="lax" />
                        </choice>
                        <element ref="sca:callback" minOccurs="0" maxOccurs="1" />
                        <any namespace="##other" processContents="lax" minOccurs="0"
                              maxOccurs="unbounded" />
                  </sequence>
                  <attribute name="name" type="NCName" use="required" />
                  <attribute name="autowire" type="boolean" use="optional"
                        default="false"/>
                  <attribute name="wiredByImpl" type="boolean" use="optional"
                        default="false"/>
                  <attribute name="target" type="sca:listOfAnyURIs" use="optional"/>
                  <attribute name="multiplicity" type="sca:Multiplicity"
                        use="optional" default="1..1" />
                  <attribute name="requires" type="sca:listOfQNames" use="optional"/>
                  <attribute name="policySets" type="sca:listOfQNames"
                        use="optional"/>
                  <anyAttribute namespace="##any" processContents="lax" />
            </restriction>
      </complexContent>
  </complexType>

  <element name="implementation" type="sca:Implementation"
    abstract="true" />
  <complexType name="Implementation" abstract="true">
    <attribute name="requires" type="sca:listOfQNames" use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
  </complexType>

  <element name="implementationType" type="sca:ImplementationType"/>
  <complexType name="ImplementationType">
    <sequence minOccurs="0" maxOccurs="unbounded">


          <any namespace="##other" processContents="lax" />
    </sequence>
    <attribute name="type" type="QName" use="required"/>
    <attribute name="alwaysProvides" type="sca:listOfQNames" use="optional"/>
    <attribute name="mayProvide" type="sca:listOfQNames" use="optional"/>
    <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>

  <complexType name="Wire">
      <sequence>
          <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
      </sequence>
      <attribute name="source" type="anyURI" use="required"/>
      <attribute name="target" type="anyURI" use="required"/>
      <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>

  <element name="include" type="sca:Include"/>
  <complexType name="Include">
          <attribute name="name" type="QName"/>
          <anyAttribute namespace="##any" processContents="lax"/>
```

```
    </complexType>

  <complexType name="Operation">
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="requires" type="sca:listOfQNames" use="optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use="optional"/>
    <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>

  <element name="constrainingType" type="sca:ConstrainingType"/>
  <complexType name="ConstrainingType">
      <sequence>
          <choice minOccurs="0" maxOccurs="unbounded">
              <element name="service" type="sca:ComponentService"/>
              <element name="reference" type="sca:ComponentReference"/>
              <element name="property" type="sca:Property" />
          </choice>
          <any namespace="##other" processContents="lax" minOccurs="0"
                maxOccurs="unbounded"/>
      </sequence>
      <attribute name="name" type="NCName" use="required"/>
      <attribute name="targetNamespace" type="anyURI"/>
      <attribute name="requires" type="sca:listOfQNames" use="optional"/>
      <anyAttribute namespace="##any" processContents="lax"/>
  </complexType>


  <simpleType name="Multiplicity">
      <restriction base="string">
          <enumeration value="0..1"/>
          <enumeration value="1..1"/>
          <enumeration value="0..n"/>
          <enumeration value="1..n"/>
      </restriction>
  </simpleType>

  <simpleType name="OverrideOptions">
      <restriction base="string">

         <enumeration value="no"/>
         <enumeration value="may"/>
         <enumeration value="must"/>
     </restriction>
 </simpleType>

 <!-- Global attribute definition for @requires to permit use of intents
      within WSDL documents -->
 <attribute name="requires" type="sca:listOfQNames"/>

 <!-- Global attribute defintion for @endsConversation to mark operations
      as ending a conversation -->
 <attribute name="endsConversation" type="boolean" default="false"/>

 <simpleType name="listOfQNames">
   <list itemType="QName"/>
 </simpleType>

 <simpleType name="listOfAnyURIs">
     <list itemType="anyURI"/>
 </simpleType>
```

```
</schema>
```

2.1.3 sca-binding-sca.xsd

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

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

    <element name="binding.sca" type="sca:SCABinding"
      substitutionGroup="sca:binding"/>
    <complexType name="SCABinding">
        <complexContent>
            <extension base="sca:Binding">
                  <sequence>
                        <element name="operation" type="sca:Operation" minOccurs="0"
                                 maxOccurs="unbounded" />
                  </sequence>
                    <attribute name="uri" type="anyURI" use="optional"/>
                    <attribute name="name" type="QName" use="optional"/>
                    <attribute name="requires" type="sca:listOfQNames"
                        use="optional"/>
                    <attribute name="policySets" type="sca:listOfQNames"
                        use="optional"/>
                <anyAttribute namespace="##any" processContents="lax"/>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

2.1.4 sca-interface-java.xsd

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

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

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

```
                <sequence>
                    <any namespace="##other" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded"/>
                </sequence>
                <attribute name="interface" type="NCName" use="required"/>
                <attribute name="callbackInterface" type="NCName" use="optional"/>
                <anyAttribute namespace="##any" processContents="lax"/>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

2.1.5sca-interface-wsdl.xsd

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

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

    <element name="interface.wsdl" type="sca:WSDLPortType"
            substitutionGroup="sca:interface"/>
    <complexType name="WSDLPortType">
        <complexContent>
            <extension base="sca:Interface">
                <sequence>
                    <any namespace="##other" processContents="lax" minOccurs="0"
                        maxOccurs="unbounded"/>
                </sequence>
                <attribute name="interface" type="anyURI" use="required"/>
                <attribute name="callbackInterface" type="anyURI" use="optional"/>
                <anyAttribute namespace="##any" processContents="lax"/>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

2.1.6sca-implementation-java.xsd

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

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

    <element name="implementation.java" type="sca:JavaImplementation"
      substitutionGroup="sca:implementation"/>
    <complexType name="JavaImplementation">
        <complexContent>
            <extension base="sca:Implementation">
                <sequence>
                    <any namespace="##other" processContents="lax"
                        minOccurs="0" maxOccurs="unbounded"/>
                </sequence>
                <attribute name="class" type="NCName" use="required"/>
                <attribute name="requires" type="sca:listOfQNames" use="optional"/>
                  <attribute name="policySets" type="sca:listOfQNames"
                        use="optional"/>
                <anyAttribute namespace="##any" processContents="lax"/>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

2.1.7 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://www.osoa.org/xmlns/sca/1.0"
    xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
    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="##any" processContents="lax"/>
            </extension>
        </complexContent>
    </complexType>
</schema>
```

2.1.8 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://www.osoa.org/xmlns/sca/1.0"
    xmlns:sca="http://www.osoa.org/xmlns/sca/1.0"
    elementFormDefault="qualified">

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

    <element name="definitions">
       <complexType>
          <choice minOccurs="0" maxOccurs="unbounded">
             <element ref="sca:intent"/>
             <element ref="sca:policySet"/>
             <element ref="sca:binding"/>
             <element ref="sca:bindingType"/>
             <element ref="sca:implementationType"/>
             <any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded"/>
          </choice>
       </complexType>
    </element>

</schema>
```

2.1.9 sca-binding-webservice.xsd

描述见 the SCA Web Services Binding specification [9]

2.1.10 sca-binding-jms.xsd

描述见 the SCA JMS Binding specification [11]

2.1.11 sca-policy.xsd

描述见the SCA Policy Framework specification [10]

2.2 SCA 概念

2.2.1 绑定

绑定被服务以及引用所使用。引用使用绑定来描述用于调用服务的访问机制（此服务可以是另一个SCA组合构件提供的服务）。服务使用绑定来描述客户端（可以是来自于另一个SCA组合构件的客户）要调用服务必须使用的访问机制。

SCA支持多种不同绑定类型。举例来说，包括***SCA service,Web service, stateless session EJB, data base stored procedure***以及***EIS service***。SCA提供扩展机制。通过此扩展机制，SCA运行时可以添加对附加绑定类型的支持。

2.2.2 构件

SCA构件是SCA实现的配置化实例，其提供服务和消费服务。SCA允许许多不同的实现技术，如Java、BPEL、C++。SCA定义了一个扩展机制，此扩展机制允许你引入新的实现类型。当前的规范不强制SCA运行时支持的实现技术，厂商可以选择支持他们认为重要的实现类型。单个SCA实现可以被多个构件使用，而每个构件都有不同的配置。

构件有一个实现（此构件就是该实现的实例）的引用、一系列属性值以及一系列服务引用值。属性值定义了由构件的实现所定义的属性值。引用值定义解析构件引用（其实现定义的引用）的服务。这些值要么是一个特定构件的一个服务，要么是其上层组合构件的一个引用（译者注：上层组合构件是指包含此构件的组合构件）。

2.2.3 服务

SCA服务用于声明一个实现的外部可访问的服务。对于一个组合构件，其服务一般由其内部的一个构件服务提供，或由此组合构件定义的一个服务（译者注：原文为引用）提供。后者情况允许一个带有新地址或新绑定机制的服务的重新公布。服务可以认为是外部客户端进入一个组合构件或实现的一个点。

服务象征一个实现的操作集合，设计这些操作集合暴露给其它实现使用或公开地暴露给其它地方使用（比如暴露给其它组织使用的公开的Web service）。一个服务提供的操作由一个接口指定，看作是服务客户端要求的操作（如果存在一个的话）。当可以分别寻址实现的服务时，一个实现可能包含多个服务。

服务可以提供为***SCA remote services, Web services, stateless session EJB's***以及 ***EIS service*** 等等。服务使用绑定描述服务被发布的方式。SCA提供一个扩展机制，使得为新的服务类型引入新的绑定类型成为可能。

2.2.3.1 远程服务

设计远程服务以便在松散耦合的SOA架构中被远程公布。比如，SCA实现的SCA服务可以定义为工业标准的Web service实现。远程服务对参数与返回结果使用传值方式。

2.2.3.2 本地服务

设计本地服务以便被其它实现“本地”使用。这些其它实现被同时部署于（与本地服务所在的）同一个操作系统进程的紧耦合架构中。

本地服务也许依赖于传引用调用约定，或可能采取细粒度的交互模式（此模式与远程分布式不兼容）。它们也能使用特定技术的数据类型。

当前，仅当被一个未标注@Remotable注解的Java接口定义的服务才是本地的服务。

2.2.4 引用

SCA引用描述一个实现对由某个其它实现所提供的服务的依赖。在其它实现中，通过配置指定要被使用的服务。换言之，一个引用是其业务功能执行期间实现可能会调用的某个服务。引用的类型是一个接口。

对于组合构件，组合构件的引用可以被该组合构件中的构件访问，就象是访问组合构件中构件所提供的任意服务。在配置构件时，组合构件的引用可以用作构件引用连线的目标。

组合构件的引用可以用于访问服务，如一个由其它SCA组合构件提供的SCA服务、一个web service、一个无状态会话EJB、一个数据库存储过程或一个EIS service等等。引用使用绑定描述使用服务的访问机制。SCA日工一个扩展机制以允许为引用引入新的绑定类型。

2.2.5 实现

实现是一个概念，其用于描述一块软件技术，如Java类、BPEL流程、XSLT转化或C++类，这些技术用于实现面向服务应用中的一个或多个服务。一个SCA组合构件也可以是一个实现。

实现定义变化点，包括属性，它们可以被设置，或是对其它服务的引用。由一个使用了此实现的构件来定义这些变化点。此规范将一个实现的可配置方面定义为***componentType***.

2.2.6 接口

接口定义了一个或多个业务功能。这些业务功能由服务提供并通过引用被构件使用。服务由实现的接口定义。SCA当前支持两种接口类型系统：

- Java interfaces
- WSDL portTypes

SCA也提供了一个扩展机制，通过此扩展机制一个SCA运行时候可以为添加对附加接口类型系统的支持。
接口也许是双向的。一个双向服务的服务操作必须是由服务通讯的每一端提供的服务 – 这种情况是一个特定的服务要求客户端的一个“回调”接口，此“回调”接口在客户对处理服务请求过程期间被调用。

2.2.7 组合构件

一个SCA组合构件是SCA域中组合的基本单元。SCA组合构件是一个构件、服务、引用以及内连它们的连线的集合体。使用组合构件向一个SCA域contribute元素。

组合构件有如下特性：
- 可以用于构件实现。当以这种方式使用时，其定义了构件可见性的边界。其内部构件也许不能直接被组合构件外界引用。
- 用于定义一个部署单元。使用组合构件contribute业务逻辑工件到SCA域中。

2.2.8 组合构件包含

一个组合构件可以用来提供其它组合构件定义的一部分，通过使用包含处理。这使得大型组合构件的团队开发更容易。部署期间，被包含的组合构件被一起合并到容器组合构件（此容器组合构件指包含被包含组合构件的组合构件）中以形成一个单一逻辑组合构件。

通过在容器组合构件中的<include.../>元素将组合构件包含进其它的组合构件中。SCA域以类似的方式使用组合构件，通过特定位置组合构件文件的部署。

2.2.9 属性

属性允许通过外部设置的数据值以配置实现。通过构件提供数据值，也可能源自此构件的上层组合构件的属性。

每个属性都是由实现定义的。属性可以直接通过实现语言定义，或通过实现的注解定义（当然要实现语言支持注解）或通过一个componentType文件定义。一个属性可以是简单数据类型或复杂数据类型。对于复杂数据类型，XML schema对于定义数据类型来说是优良的技术。

2.2.10 域

一个SCA域描述了一系列提供业务功能领域的服务，其由一个单一组织控制。比如，对于商务中的会计部门，SCA域也许会覆盖所有金融相关的功能，且其可能包含一系列处理帐务特定领域的组合构件，一个用于处理客户帐户，另一个用于处理可支付帐户。

一个域指定一系列构件（通过一个或多个composite文件提供）的初始化、配置以及连接。域，如同一个组合构件，也有服务和引用。域也包括连接构件、服务以及引用的连线。

2.2.11 连线

SCA连线将服务引用连接到服务。

在一个组合构件中，有效连线的源是构件的引用（注：原文可以是组合构件的服务，是错误的）。有效连线的目的是构件服务（注：原文可以是组合构件引用，是错误的）。

当使用被包含的组合构件时，连线的源和目的不必声明在包含连线的同一组合构件中，其它的被包含的组合构件可以定义源和目标。目标也能在SCA域的外部。

3. 附录 2：参考文献

[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_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

[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

[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

[10] SCA Policy Framework Specification

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

[12] ZIP Format Definition

http://www.pkware.com/documents/casestudies/APPNOTE.TXT

后记：SCA 中文规范项目

声明

SCA相关规范的中文文档得到 OSOA Chinese Community (OSOA中文社区) 的直接授权和大力的支持，其目的是在中文世界推广优秀的SOA相关技术标准。本次翻译活动由满江红开放技术研究组织 (http://www.redsaga.com) 和 OSOA 中文社区 (http://www.osoa.org/pages/viewpage.action?pageId=416) 共同发起、组织，将翻译SCA的绝大多数规范，并得到goCom社区 (http://www.gocom.cc/) 的赞助。我们在此郑重宣布，本次翻译遵循原Service Component Architecture Specification的授权协议。在完整保留全部文本包括本版权页，并不违反Service Component Architecture Specification协议的前提下，允许和鼓励任何人进行全文转载及推广。我们在此宣布所有参与人员放弃除署名权外的一切权利。

致谢

SCA 规范的翻译对很多人来说都是一个挑战，再次感谢所有参与翻译、评审工作的同学们，没有你们的辛勤劳动，也就不会有中文规范的面试。

参与人员

规范分配情况，最后的校对和统稿由管理员统一完成。

Specification	翻译	一审	二审
SCA Assembly Model V1.00	ligang1111	wangfeng	wangfeng
SCA Policy Framework V1.00	liang_ma,max	pesome	needle
SCA Transaction Policy V1.00	max	jiangxd	guangh
SCA Java Common Annotations and APIs V1.00	ligang1111	HiugongGwok	dc
SCA Java Component Implementation V1.00	ligang1111	pinelygao	nekesai
SCA Spring Component Implementation V1.00	ligang1111	pinelygao	Caozw

SCA BPEL Client and Implementation V1.00	jiangxd	hongsoft	hongsoft
SCA C++ Client and Implementation V1.00	SpringWater	SpringWater s	SpringWater s
SCA Web Services Binding V1.00	xichengmylove	hongsoft	hongsoft
SCA JMS Binding V1.00	YuLimin/Echo	hongsoft	pegasus
SCA EJB Session Bean Binding V1.00	max	hongsoft	yanfei
SCA JCA Binding V1.00 （暂不发布）	bsspirit	jiangxd	benja

参与人员列表：

网名	姓名	网名	姓名
billytree	冯博	guangh	光华
ligang1111	李刚	Caozw	小曹
xichengmylove	严永华	yanfei	晏斐
pesome	张俊	benja	老贾
max	孙浩	HiugongGwok	郭晓刚
needle	needle	bsspirit	张丹
YuLimin	俞丽敏	Echo	杨春花
wangfeng	王锋	pinelygao	高松
hongsoft	洪波	liang_ma	马亮
jiangxd	姜晓东	Dc	王葱权
SpringWater	张世富	pegasus	马捷
chrischengzh	chris	nekesai	蔡永保
jiaoly	老焦	keqiang	克强

下列人员报名，但由于种种原因没有参与，在此感谢：

网名	姓名	网名	姓名
jetyang_1	杨文明	cenwenchu	岑文初
lafay	叶江	Eric	孟庆余
laodingshan	丁跃斌		

项目历程

SCA 规范一期 翻译项目进度汇总

2008 年 03 月 17 日

3 月 17 日，SCA 规范一期的翻译项目正式启动，Wiki 中增加了相应的 SCA 板块和对应的几个栏目。在 JavaEye 和 goCom 上发了招募贴，在满江红邮件列表中发了封邮件。

希望有更多的同学能够参加到这次的项目中，积极报名，争取能早日高质量地完成所有的工作，让更多人能享受到我们的劳动成果～

2008 年 03 月 22 日

相当郁闷，我在 JavaEye 上的帖子竟然被隐藏了，也忒欺负人了。经过沟通之后，终于又恢复了，又在 JavaEye 发了一遍新闻，呵呵。

2008 年 03 月 28 日

通过这段调查，发现 SCA 规范和 Spring 比较，相当于 阳春白雪 VS 下里巴人， 研究规范的都是比较专业的人士。 看来我们的翻译工作很有必要啊，在国内推广 SOA。

2008 年 04 月 02 日

昨天是愚人节，结果自己被愚弄了一把，看到 Spring 被微软收购了，真是"很傻很天真"。

鉴于这次规范的翻译的特殊性，有相当多的规范已经有中文草稿，所以没有采取其他文档翻译按章节划分的做法。

截至今天为止参加人数已经有 10 几个人，一期计划翻译的 9 个规范都已经被领走了，ligang1111 同学一个人承包了 5 个规范，而且都是分量最重，而且内容最多的部分。最后一个 JMS binding 规范翻译是被 俞丽敏夫妇领走的，革命伴侣，令人羡慕！

希望有更多的人参与进来，现在正在招募一审，二审人员。

2008 年 04 月 06 日

CVS 服务器已经设置好，上传了规范英文文档，相关翻译人员分配了 CVS 帐号，大家可以开始干活了，呵呵。

2008 年 04 月 08 日

刚同学已经完成 SCA Spring Component Implementation Specification V100 翻译，在这里向他的辛勤劳动表示祝贺！

他为我们已经开了一个好头！

其他已经领取任务的同志们加油啊！

2008 年 04 月 10 日

昨天满江红网站出了故障访问不了，还好今天已经正常了。

xichengmylove 同学已经完成 SCA Web Services Binding V1.00 翻译，max 已经完成领取任务的 30%. 在这里向他门的辛勤劳动表示祝贺！

其他已经领取任务的同志们加油啊！请及时上报翻译进度。

2008 年 04 月 11 日

我们又有新鲜血液注入拉： 让我们欢迎 jiangxd，SpringWater 。他们分别承担了 SCA BPEL client，SCA C++ Client 的翻译工作，鼓掌！

SCA BPEL Client and Implementation V1.00	jiangxd			已领取	15	2008.4.11
SCA C++ Client and Implementation V1.00	SpringWater			已领取	70	2008.4.11

2008 年 04 月 21 日

前面一周出差了，回来发现很多文档都翻译完了，但是有些翻译文档没有上传，请大家及时上传，有问题直接和我联系。

众人拾柴火焰高！原定二期翻译的 SCA C++ Client and Implementation V1.00 规范已经完成 70%的翻译，向 SpringWater 致敬！

目前比较头疼的是，一审二审工作，还望群策群力。

2008 年 04 月 24 日

昨天下午去听了 SOA 中国的关键任务 上海站 报告，会上有个仁兄替我们 SCA 翻译作了个小广告，弱弱的问下，是那位啊？

感谢李刚同学！ 翻译完成 **SCA Assembly Model V1.00** ，分量最重的这块。

让我们欢迎 teamlet! 他将负责 **SCA Assembly Model V1.00** 的一审工作！

2008 年 04 月 28 日

SpingWater 今天将 SCA_ClientAndImplementationModel_Cpp-V100 翻译文档发给我了，我看了一下，其实我们计划的翻译阶段即将完成了！

再次感谢所有参与者的辛苦劳动！

现在面临的问题是 翻译文档的一审、二审工作，我将联系 OSOA 中文社区、Tuscany 中文社区、goCom 和 JavaEye 社区的 SOA 爱好者，也向所有的 SOA 爱好者发出邀请，希望大家热情参与！

2008 年 04 月 29 日

laodingshan 同学今天领取了 SCA ComponentImplementation_V100 和 SCA_SpringComponentImplementationSpecification-V100 的一审工作，虽然他是个老同志，但是参与 SOA 的热情很高！

对他的加入表示热烈欢迎！

2008年05月15日

今天有了个想法，这次翻译工作我们也能拉到一部分赞助，因此我想征求大家的意见，设立一个伯乐奖，对于推荐熟悉SOA相关技术，参与标准翻译"千里马"的"伯乐"也给与这次翻译活动的纪念品，希望大家积极推荐！

2008年05月19日

沉痛悼念汶川大地震死难同胞！

2008年05年20日

liang_ma 同学主动承担了 **SCA Policy Framework V1.00** 规范的翻译工作，这个规范比较长，有46页，让我们为他加油打气！

2008年05月27日

SCA Java Common Annotations and APIs V1.00 已经翻译完毕，祝贺 ligang1111；

HiugongGwok 同学将承担一审工作，欢迎 HiugongGwok ！

bsspirit 同学将承担 SCA_JCABindings v 1.0 的翻译，欢迎！

2008年06月05日

这一周，大家很忙啊，进度没有及时汇报，兄弟们要加油啊，一鼓作气！

提前预祝端午节快乐！

2008年06月12日

这是第一个法定的端午节假期，我们勤劳的 jiangxd 同志加班加点，完成了 **SCA Transaction Policy V1.00** 的一审工作。在此表示祝贺！

bsspirit 同学承担 SCA_JCABindings v 1.0 的翻译进度也很快，工作量已完成70%了。

2008年06月17日

bsspirit 同学完成 **SCA JCA Binding V1.00** 的翻译工作，在此表示祝贺！

由于 laodingshan 同志自身工作很忙，所以他负责的 SCA ComponentImplementation_V100 和 SCA_SpringComponentImplementationSpecification-V100 一审工作现在由 pinelygao 接替。

pinelygao 同志有着8年的J2ee开发、架构经历，将会给予我们很大的支援！

2008年06月20日

在 **SCA Policy Framework V1.00** 规范翻译中，liang_ma 同学可能遇到了困难。进度比较缓慢。

我们不得不承认，这对他是一个挑战，而且他已经翻译了 40%了 ！！！

这个规范是最后一个正在翻译的规范了。我们惊喜的发现，现在翻译的规范数比我们原来计划的要多很多了。

在我们为他加油打气的同时，我们也希望，期待援助：圈内人士，如果有对此感兴趣，欢迎积极加入，SCA Policy Framework V1.00 是一个很重要的规范，也是下一步研究的热点。 规范总共 46 页。

希望大家踊跃认领，支持！

补充：伟大的 jiangxd 同志又承担了 **SCA JCA Binding V1.00** 一审工作，我已经不知道说什么感谢好了。。。。。

2008 年 06 月 23 日

max 同志将承担起 **SCA JCA Binding V1.00** 的剩余翻译工作。这样我们的工作将可以 7 月份完成。届时组织一次集体规范的二审工作。

接下来，SCA 相关规范的中文版将面世。

请 liang_ma 同学尽快将你已经翻译的部分上传或者发给我。

SCA Policy Framework V1.00 一审已经被 pesome 领取了，感谢 pesome ！！！

2008 年 06 月 25 日

今天把 liang_ma 负责的 SCA_Policy_Framework_V100 翻译上传了，他已经翻译到了 1.4 节。

max 同志终于可以开始工作了。

感谢天，感谢地，看来原定计划 不会难产了。

2008 年 06 月 26 日

今天是个好日子。

喜报！

hongsoft 同学负责的 SCA BPEL Client and Implementation V1.00 ， SCA EJB Session Bean Binding V1.00 一审完毕。

pinelygao 同学负责的 SCA Java Component Implementation V1.00 一审完毕

而且，hongsoft 同学还将接替 needle,完成 SCA Web Services Binding V1.00 和 SCA JMS Binding V1.00 的一审工作。

向勤劳的 hongsoft 大厨表示致敬！！！

2008 年 07 月 07 日

新的规范出来了，**SCA Java EE Integration Specification V1.00** ，看了下，ORACLE 和 BEA 的人加起来比 IBM 还要多。没有看到 Primeton 的人，看来我们中国厂商要从标准跟随者到领导者，还要多多努力。

SCA Policy Framework V1.00 的翻译工作非常迅速，感谢 max 的辛苦工作，今天他告诉我只剩下 1.4 节和 liang_ma 交界的地方了。

今天是鄙人的生日，假公济私一下，哈哈。。

2008 年 07 月 10 日

HiugongGwok 承担的 SCA Java Common Annotations and APIs V1.00 一审工作已经完成。

而且，max 同志承担的 **SCA JCA Binding V1.00** 翻译已经高质量的完成，细心的他将在今晚自己审查后，上传。对 max 同志的高度责任心，一丝不苟的认真态度，值得我们学习。

再次感谢 2 位同志，胜利的曙光已经越来越近了。

还有一个好消息，我已经联系了相关赞助，将为我们这次的 SOA 规范翻译活动出专门的纪念 T 恤，以为各位同志的辛勤工作给少许心理上的安慰。

2008 年 07 月 11 日

max 同志承担的 **SCA Policy Framework** 已经上传 CVS，pesome 同志要接着进行"火炬"传递，完成一审工作。

SCA JCA Binding V1.00 一审工作在伟大的 jiangxd 同志的辛苦工作下已经完成，他自己本身工作也很忙，在经常加班的情况下，还认真审阅了规范的翻译，进行了大量认真、细致的修改、校正工作。

胜利的曙光。。。。。。

2008 年 07 月 14 日

wangfeng 同学已经完成了 **SCA Assembly Model V1.00** 这个最长规范的一审工作，鉴于该同志一如既往地兢兢业业，经研究决定，对该同志进行通报表扬，呵呵！

2008 年 07 月 21 日

pesome 同学负责的 **SCA Policy Framework V1.00** 一审工作已经完成 50%，预计将于本周末全部完成。

这样我们将在 8 月 2 号组织一次集中的二审工作，这样整个一期规范将于 8 月中旬正式发布。

2008 年 07 月 23 日

昨天当了一次黄世仁，呵呵。

感谢 pinelygao 同学提交了 **SCA Java Component Implementation V1.00** ，**SCA Spring Component Implementation V1.00** 一审文档。

2008 年 07 月 28 日

pesome 同学负责的 **SCA Policy Framework V1.00** 一审工作已经完成,祝贺阿

一个不好的消息是 SCA JCA 规范需要因为翻译质量不过关，需要重新翻译

好消息是二审工作已经全面铺开了

2008 年 08 月 18 日

刘翔退赛了。

我们的 SCA 规范翻译二审工作还在继续。。。

SCA 规范目前正在进行紧张的二审工作，在欣赏到众多志愿者翻译成果的同时，也发现了不少的问题。

在此，我们诚征志愿者，对有问题的中文规范进行修改，我们提供中文版规范和翻译中问题的批注供您参考。并且你将会署名在发布的规范中。
另外，对于参与翻译工作的人员，我们提供了一些纪念品。目前的想法包括：

1. **按翻译页数奖励 goCom 币，每页奖励 goCom 币 10 分。**

（用此 G 币可以在 goShop 商城购买商品，凡卓越网上能看到的商品都可以在 goCom 上通过获得的 G 币进行购买）

2. **打造成为 goCom 专家组成员。**

（goCom 近期将推出专家组概念，专家组成员将在 goCom 网站专家页面享有独立页面用以展示其个人内容，goShop 商城中享受更高折扣，各级评比中享受 10%的加分，北京用户免费参与 goCom 户外活动)

3. **邀请参加 goCom 在线技术日活动，每次活动可奖励 1500goCom 币。**
4. **赠送最新银弹杂志与 goCom 纪念 T 恤。**

2008 年 08 月 29 日

日志好久没有更新了。二审工作确实比较辛苦，跟每个规范的翻译质量有很大关系，而且每个人对规范的理解也不一样。

郁闷。。。。

我们需要专业的 SCA 人员，作校对和最后的把关！

- 但是不管怎样，我们还要前行，接下来，将陆续发布规范，首先是装配规范和 java 注解和实现规范。*

希望大家多提宝贵意见，在拍砖的同时能够体会我们翻译者的苦与痛。

讨论、建议和勘误

进行规范的翻译确实是一个挑战，错误和翻译不当之处在所难免，请在尊重作者辛勤劳动的基础上，提出你们的宝贵建议。

我们在goCom OSOA中文专区（http://www.gocom.cc/modules/osoa/）上专门开辟了一个板块，欢迎指出错误和讨论。

技术圈子

欢迎您加入 **SOA**技术圈子 http://groups.google.com/group/SOAer

SOAer 是一个 GoogleGroup,聚集了国内 SOA 方面的架构师、咨询师、技术专家和技术爱好者。主要活动包括：

1 SOA 相关开源的运作；
2 SOA 技术讨论；
3 SOA 相关热点问题的讨论；
4 SOA 规范：SCA/SDO 的翻译；
5 SOA 相关技术的高级培训；
6 特邀讲座及报道；
7 SOA 相关规范及书籍翻译；

我们本着“talk u like and do u like! ”、“一起分享，一起成长！”的宗旨，一起探讨 SOA 的方方面面。

招募

欢迎大家参与SCA、SDO后续规范的翻译。敬请关注：http://groups.google.com/group/SOAer