

SCA *Service Component Architecture*

Client and Implementation Model Specification for WS-BPEL

SCA Version 1.00, March 21 2007

Technical Contacts:	Martin Chapman	Oracle
	Sabin Ielceanu	TIBCO Software Inc.
	Dieter Koenig	IBM Corporation
	Michael Rowley	BEA Systems, Inc.
	Ivana Trickovic	SAP AG
	Alex Yiu	Oracle

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 and display 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 this 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 AG., 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 COMPONENT 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.

Table of Contents

Copyright Notice	ii
License	ii
Status of this Document	iii
1. WS-BPEL Client and Implementation Model	1
1.1. Goals	1
1.2. WS-BPEL Processes as Component Implementations	1
1.3. Component Types defined by WS-BPEL Processes	2
1.3.1. Services and References	2
1.3.2. PartnerLinkTypes and SCA Interfaces	3
1.3.3. Specifying an SCA interface with a partnerLinkType	4
1.3.4. Handling of Local PartnerLinks	5
1.3.5. Support for conversational interfaces	5
1.4. SCA Extensions to WS-BPEL	5
1.4.1. Properties	6
1.4.2. Multi-Valued References	6
1.5. Using BPEL4WS 1.1 with SCA	8
2. Appendix	9
2.1. XML Schema	9
2.2. References	11

1. WS-BPEL Client and Implementation Model

1.1. Goals

The SCA WS-BPEL Client and Implementation model specifies how WS-BPEL 2.0 can be used with SCA. The goal of the specification is to address the following scenarios.

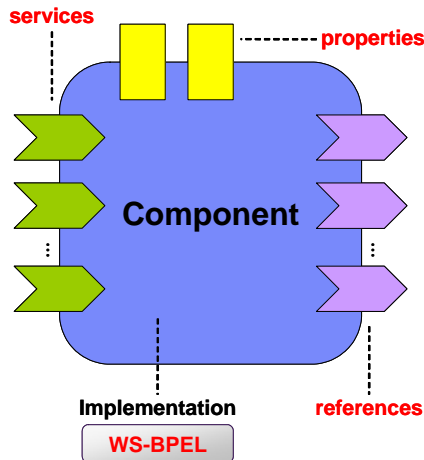
Start from WS-BPEL process. It should be possible to use any valid WS-BPEL process definition as the implementation of a component within SCA. In particular, it should be possible to generate an SCA Component Type from any WS-BPEL process definition and use that type within an SCA assembly. Most BPEL4WS 1.1 process definitions may also be used with SCA by using the backward compatibility approach described in section 1.5.

Start from SCA Component Type. It should be possible to use WS-BPEL to implement any SCA *Component Type* that uses only WSDL interfaces to define services and references, possibly with some SCA specific extensions used in process definition.

Start from WS-BPEL with SCA extensions. It should be possible to create a WS-BPEL process definition that uses SCA extensions and generate an SCA Component Type and use that type within an SCA assembly. Some SCA capabilities (such as properties and multi-party references) can only be used by WS-BPEL process definitions that use SCA extensions.

1.2. WS-BPEL Processes as Component Implementations

A WS-BPEL process definition may be used as the implementation of an SCA component.



Such a component definition has the following form:

```
<component name="xs:NCName">*
  <implementation.bpel process="xs:QName"/>
  <property name="xs:NCName" source="xs:string"? file="xs:anyURI"?>*
    property-value?
  </property>
  <reference name="xs:NCName"/>*
    wire-target-URI
  </reference>
</component>
```

36 The only aspect of this that is specific to WS-BPEL is the `<implementation.bpel>` element. The `process`
 37 attribute of that element specifies the target QName of some executable WS-BPEL process.

38

39 **1.3. Component Types defined by WS-BPEL Processes**

40 While a WS-BPEL process definition provides an implementation that can be used by a component, the
 41 process definition also determines the `ComponentType` of any SCA component that uses that
 42 implementation. The component type represents the aspects of the implementation that SCA needs to be
 43 aware of in order to support assembly and deployment of components that use that implementation. The
 44 generic form of a component type is defined in the SCA Assembly Specification [1] as follows:

45

```
46 <componentType xmlns="http://www.oesa.org/xmlns/sca/1.0" >
47
48   <service name="xs:NCName">*
49     <interface/>
50   </service>
51
52   <reference name="xs:NCName" override="sca:OverrideOptions"?
53     multiplicity="0..1 or 1..1 or 0..n or 1..n"?>*
54     <interface/>
55   </reference>
56
57   <property name="xs:NCName" (type="xs:QName" | element="xs:QName")
58     many="xs:boolean"? required="xs:boolean"?>*
59     default-property-value?
60   </property>
61
62 </componentType>
```

63

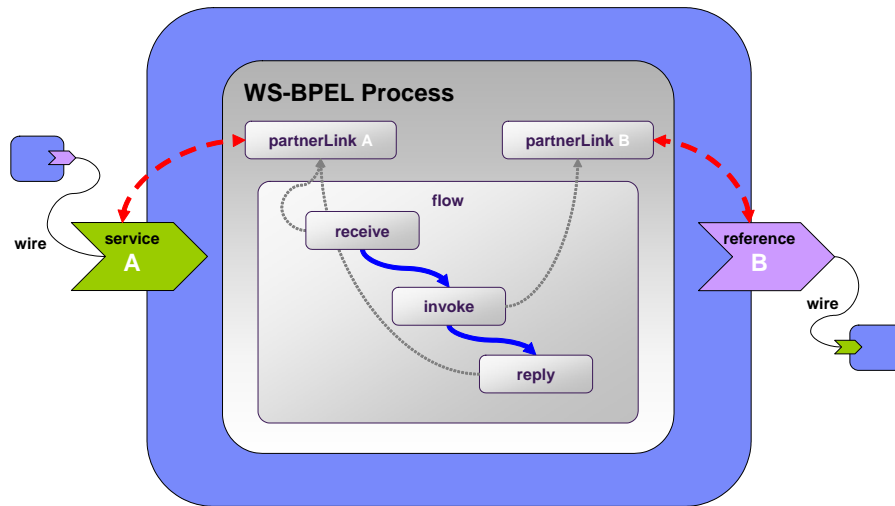
64 The component type MAY be generated from a WS-BPEL process definition by introspection.

65

66 **1.3.1. Services and References**

67 In SCA, both *services* and *references* correspond to WS-BPEL's concept of partner link. In SCA, the
 68 difference between a service and a reference is determined by which party sends the first message in a
 69 conversation. No matter of how many messages a bi-directional conversation involves or how long it takes,
 70 there is always a first message. The sender of the first message is considered to be the *client* and the
 71 receiver is the *service provider*. Messages that go from the service provider to the client are called *callback*
 72 *messages*.

73 WS-BPEL's partner links are not differentiated based on who sends the first message. So, in order to map a
 74 WS-BPEL process to an SCA Component Type, it is necessary to determine which role sends the first
 75 message. A simple static analysis of the control flow, which does not involve determining the values of any
 76 expressions, will be used to determine which role can send the first message.



77

78

79 **Services:** If a static analysis of the process determines that it is possible that the first message for a
 80 partner link will be received in a <receive> activity, the <onMessage> element of a <pick> activity or
 81 the <onEvent> element of an event handler then the partner link has a corresponding SCA service in the
 82 component type. If the partner link declaration has `initializePartnerRole="yes"`, then the service
 83 must be wired using a binding that knows the identity of the partner as soon as the partner link becomes
 84 active (e.g. the binding cannot depend on using a "reply-to" field as the mechanism to initialize partner
 85 role.).

86 **References:** If a static analysis of the process does not determine that the partner link should map to an
 87 SCA service, then the partner link is mapped to an SCA reference in the component type.

88 The *multiplicity* of the reference is determined by the following algorithm:

- 89 1. **Multi-Reference.** If the partner link is declared with `sca:multiRefFrom="aVariableName"`
 90 extension, the multiplicity of the SCA reference will be determined by the `multiplicity` attribute
 91 of `sca:multiReference` extension used in the corresponding variable. The multiplicity
 92 declaration of the variable which is either `0..n` or `1..n`. Details of these extensions are described in
 93 section 1.4.2.
- 94 2. **Required Reference.** If not (1) and the partner link has `initializePartnerRole="yes"`, then
 95 the multiplicity is `1..1` (i.e. it's a *required reference*).
- 96 3. **Stub Reference.** If not (1) or (2) and if the analysis of the process determines that the first use of
 97 the partner link by any activity is in an assign activity that sets the partner role, then the
 98 multiplicity is `"0..1"` and the attribute `wiredByImpl` is set to `"true"`. A reference with
 99 `wiredByImpl="true"` is referred to as a *stub reference*. Although the target can't be set for such
 100 a reference, SCA can still apply bindings and policies to it and may need to set the endpoint address
 101 for callbacks, if the interface is bi-directional.
- 102 4. **Optional Reference.** If not (1) or (2) or (3) then the `multiplicity="0..1"`.

103

104 For both services and references, the name of the service or reference is the name partner link, when that
 105 name is unique (see the "Handling Local Partner Links" section below, for how to handle ambiguous cases).

106

107 1.3.2. PartnerLinkTypes and SCA Interfaces

108 When a partner link is determined to correspond to an SCA service, the type of the service is determined by
 109 the partner link type of the partner link. The role that the partner link specified as *myRole* provides the
 110 WSDL port type of the service. If the partner link type has two roles, then the *partnerRole* provides the
 111 WSDL port type of the callback interface.

112 Consider an example that uses one of the partner link types used as an example in the WS-BPEL
113 specification. The partner link type definition is:

```
114 <plnk:partnerLinkType name="invoicingLT">
115     <plnk:role name="invoiceService"
116         portType="pos:computePricePT" />
117     <plnk:role name="invoiceRequester"
118         portType="pos:invoiceCallbackPT" />
119 </plnk:partnerLinkType>
```

120

121 The "invoiceProcess", which provides invoice services, would define a partner link that uses that type with a
122 declaration that would look like:

```
123     <partnerLink name="invoicing"
124         partnerLinkType="lns:invoicingLT"
125         myRole="invoiceService"
126         partnerRole="invoiceRequester" />
```

127

128 Somewhere in the process, a start activity would use that partner link, which might look like:

```
129     <receive partnerLink="invoicing"
130         portType="pos:computePricePT"
131         operation="initiatePriceCalculation"
132         variable="PO"
133         createInstance="yes" />
```

134

135 Because the partner link is used in a start activity, SCA maps that partner link to a service for on the
136 component type. In this case, the service element of the component type would be:

```
137     <service name="invoicing">
138         <interface.wSDL
139             interface="http://manufacturing.org/wSDL/purchase#
140                 wsdl.interface(computePricePT)"
141             callbackInterface="http://manufacturing.org/wSDL/purchase#
142                 wsdl.interface(invoiceCallbackPT)" />
143     </service>
```

144 Conversely, when a partner link is determined to correspond to an SCA reference, the role that the partner
145 link specified as *partnerRole* provides the WSDL port type of the reference. If the partner link type has two
146 roles, then the *myRole* provides the WSDL port type of the callback interface.

147 1.3.3. Specifying an SCA interface with a partnerLinkType

148 In the approach described above, the SCA definition of service and reference uses the <interface.wSDL>
149 which restates the association between the interface and the callback interface that is already present in the
150 WS-BPEL partnerLinkType. A partnerLinkType defines the relationship between two services by specifying
151 roles the services play in the conversation. A partnerLinkType specifies at least one role.

152 For users that prefer this WS-BPEL element, it is also possible to define interfaces with an alternative
153 partnerLinkType form of an interface type. This form does not provide any more information than is
154 present in the <interface.wSDL> element. The example above would look like the following:

```
155     <interface.partnerLinkType type="lns:invoicingLT"
156         serviceRole="invoiceService" />
```

157

158 The generic form of this interface type definition is as follows:

```
159     <interface.partnerLinkType type="xs:QName"
160         serviceRole="xs:NCName" ? />
```

161

162 The `type` attribute is mandatory and references a partner link type. In case the partner link type has two
 163 roles, the optional attribute `serviceRole` MUST be used to specify which of the two roles is used as the
 164 interface. The other role is used as the callback. If the `partnerLinkType` has only one role, it cannot be a
 165 callback. Moreover, the `serviceRole` attribute MAY be omitted.

166 This form has a couple advantages over the `interface.wSDL` form. It is more concise. It also doesn't restate
 167 the link between the interface and the `callbackInterface`, so with this form, the `partnerLinkType` could
 168 change the `portType` used to define one of the roles and all of the SCA `componentTypes` that use that
 169 `partnerLinkType` would remain accurate without having to also change the interface definitions for those
 170 `componentTypes`. This form also may be more familiar to some users.

171

172 1.3.4. Handling of Local PartnerLinks

173 It is possible to declare `partnerLinks` local to a `<scope>` in WS-BPEL, besides declaring `partnerLinks` at the
 174 `<process>` level. The names of `partnerLink` declared in different `<scope>` may potentially share the
 175 identical name. In case of this name sharing situation, the following scheme is used to disambiguate
 176 different occurrences of `partnerLink` declaration:

- 177 • Suppose "originalName" is the original NCName used in multiple `partnerLink` declarations
- 178 • When these `partnerLinks` are exposed to SCA assembly, these `partnerLinks` will given aliases from
 179 "_originalName_1" to "_originalName_N" regardless of how `partnerLink` participate in SCA assembly
 180 (i.e. services vs references)
- 181 • If any "_originalName_i" (where $1 \leq i \leq N$) is already taken by existing `partnerLink` declaration
 182 in the process definition, additional underscore characters may be added at the beginning of the
 183 aliases to avoid collision.

184

185 1.3.5. Support for conversational interfaces

186 WS-BPEL can be used to implement an SCA Component with *conversational* services. See the SCA Assembly
 187 Specification [1] for a description of conversational interfaces. When an interface that has been marked as
 188 conversational is used for a role of a partner link, no other mechanism (such as the WS-BPEL correlation
 189 mechanism) is needed to correlate messages on that partner link, although it is still allowed. This means
 190 the SCA conversational interface is used as an implicit correlation mechanism to associate all messages
 191 exchanged (in either direction) on that partner link to a single conversation. When the EPR of the
 192 `partnerRole` is initialized a new conversation MUST be used for an operation of the conversational service.

193 Any process which, through static analysis, can be proved to use an operation on a conversational interface
 194 after an *endsConversation* operation has completed SHOULD be rejected. In cases where the static analysis
 195 cannot determine that such a situation could occur, then at runtime a `sca:ConversationViolation` fault would
 196 be generated when using a conversational partner link after the conversation has ended. See the SCA
 197 Assembly Specification [1], section 1.5.3 for a description of this fault.

198 It is important to point out that the WS-BPEL correlation mechanism is not restricted to a single partner
 199 link. It can be used to associate messages exchanged on different partner links to a particular WS-BPEL
 200 process instance.

201

202 1.4. SCA Extensions to WS-BPEL

203

204 It is possible to use WS-BPEL processes in conjunction with SCA, while the processes have no knowledge of
 205 SCA. A few SCA concepts are only available to WS-BPEL processors that support SCA specific extensions.
 206 The capabilities that require knowledge of SCA are provided by an SCA extension, which must be declared
 207 in any process definition as follows:

```
208 <process ...>
209     <extensions>
210         <extension namespace="http://www.osea.org/xmlns/sca/bpel/1.0"
211             mustUnderstand="yes"/>
```

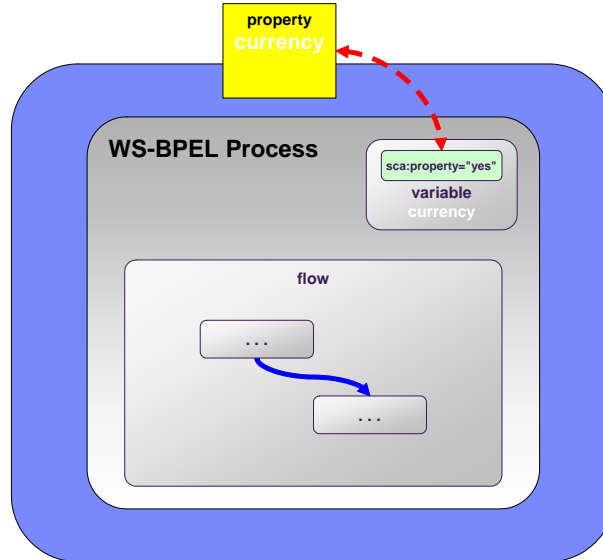
212 </extensions>

213

214 1.4.1. Properties

215

216 A WS-BPEL variable declaration may include an SCA extension that says that the variable represents an
 217 SCA property for the component represented by the WS-BPEL process.



218

219

220 The declaration looks like the following:

```
221 <variable name="currency" type="xsd:string" sca:property="yes" />
```

222

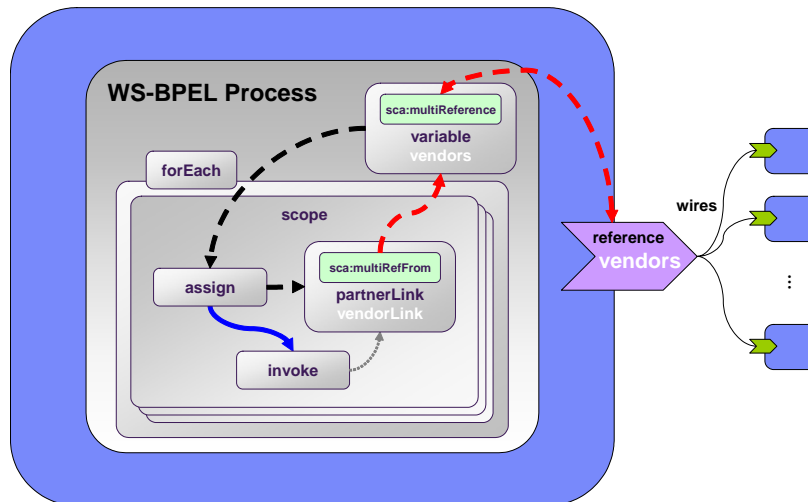
223 When `sca:property="yes"` is used on a variable declaration, the name of the variable is used as the
 224 name of a property of the component type represented by the WS-BPEL process. The name of the variable
 225 must be unique within the process.

226 At runtime, the variable will be initialized with the value provided by the component definition that uses the
 227 WS-BPEL process.

228

229 1.4.2. Multi-Valued References

230 Component types may declare references with a multiplicity that allows a single reference to be wired to
 231 multiple targets. An example use of this capability is a purchasing component wired to a list of accepted
 232 vendors. SCA assumes that each programming language binding will provide its own approach for making
 233 the list of targets available within that programming language.



234

235

236 In WS-BPEL, a variable may include an `sca:multiReference` extension element that declares that the
 237 variable represents a multi-valued reference. The type of the variable must be an element of
 238 `sca:serviceRefList`. However, since that type only specifies that the variable holds a list of endpoint
 239 references, the `sca:multiReference` element also has attributes to specify the partner link type and
 240 partner role of the target of the reference. An example of a variable that represents a list of references to
 241 vendors would look like:

```
242 <variable name="vendors" element="sca:serviceReferenceList">
243   <sca:multiReference partnerLinkType="pos:vendorPT" partnerRole="vendor"/>
244 </variable>
```

245

246 Syntax of this extension:

```
247 <sca:multiReference partnerLinkType="xs:QName" partnerRole="xs:NCName"
248   multiplicity="0..n or 1..n"?/>
```

249

250 The default value of multiplicity is "1..n".

251 The `sca:serviceReferenceList` element declaration is the following:

```
252 <xsd:element name="serviceReferenceList">
253   <xsd:complexType>
254     <xsd:sequence>
255       <xsd:element ref="bpel:service-ref" minOccurs="0" maxOccurs="unbounded"/>
256     </xsd:sequence>
257   </xsd:complexType>
258 </xsd:element>
```

259

260 A typical use of a variable that holds a multi-valued reference would be to have a `<forEach>` activity with
 261 an iteration for each element in the list. The body of the `<forEach>` activity would declare a local partner
 262 link and assign one of the list elements to the local partner link. Such a local partner link is typically
 263 categorized as the "References" case 1 listed in section 1.3.1.

264 To assist a more effective SCA modeling, another SCA extension is introduced to associate a multi-valued
 265 reference, manifested as a "sca:serviceReferenceList" variable with a partner link. This extension is
 266 in an attribute form attached to the partner link declaration. Syntax of this extension is:

```
267 sca:multiRefFrom="xs:NCName"
```

268

269 The attribute value must refer to the name of a variable manifesting an SCA multi-valued reference. The
 270 `partnerLinkType` and `partnerRole` attributes of the partner link and multi-valued reference variable

271 must be matched. Also, there must be at least one code-path that values from the multi-valued reference
 272 variable are copied to the `partnerRole` of the partner link.

273 If any above constraints are violated, it will be considered an error during static analysis.

274 When this `sca:multiRefFrom` extension is applied to pair up a multi-valued reference variable and a
 275 partner link which is categorized as the "References" case 1 (as described in section 1.3.1), the partner link
 276 and variable are manifested as a single multi-valued reference entity in SCA assembly model using the
 277 name of the variable. If the interface involved is bi-directional, this implies the wiring of the bi-directional
 278 interface as a single reference in SCA.

279

280 For example:

```

281 <variable name="vendors" element="sca:serviceReferenceList">
282   <sca:multiReference partnerLinkType="pos:vendorPT" partnerRole="vendor"/>
283 </variable>
284 ...
285 <forEach counterName="idx" ...>
286   <startCounterValue>1</startCounterValue>
287   <finalCounterValue>count($vendors/bpel:service-ref)</finalCounterValue>
288   ...
289   <scope>
290     ...
291     <partnerLink name="vendorLink"
292                 partnerLinkType="pos:vendorPT"
293                 partnerRole="vendor"
294                 myRole="quoteRequester"
295                 sca:multiRefFrom="vendors" />
296     ...
297     <assign>
298       <copy>
299         <from>$vendors/bpel:service-ref[$idx]</from>
300         <to partnerLink="vendorLink" />
301       </copy>
302     </assign>
303     ...
304   </scope>
305 </forEach>
306
```

307 A multi-valued reference named "vendors" is declared in the example above. The partner link named
 308 "vendorLink", which is categorized as the "References" case 1, is not manifested directly into the SCA
 309 Assembly Model. The extra `sca:multiRefFrom="vendors"` extension associates the "vendorLink"
 310 partner link with multi-valued reference variable "vendors". Consequently, the partner link and variable
 311 are manifested as a single multi-valued reference named "vendors" in SCA. This makes the SCA Assembly
 312 modeling easier to follow.

313

314 **1.5. Using BPEL4WS 1.1 with SCA**

315 A BPEL4WS 1.1 process definition may be used as the implementation of an SCA component. The syntax
 316 introduced in section 1.2 is used to define a component having a BPEL4WS 1.1 process as the
 317 implementation. In this case, the `process` attribute specifies the target QName of a BPEL4WS 1.1
 318 executable process.

319 A BPEL4WS 1.1 process definition may be used to generate an SCA Component Type.

320

321 2. Appendix

322

323 2.1. XML Schema

324

325 XML Schema sca-implementation-bpel.xsd

```

326 <xsd:schema xmlns="http://www.oesa.org/xmlns/sca/1.0"
327   targetNamespace="http://www.oesa.org/xmlns/sca/1.0"
328   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
329   elementFormDefault="qualified">
330
331   <xsd:include schemaLocation="sca-core.xsd"/>
332
333   <xsd:element name="implementation.bpel"
334     type="BpelImplementation"
335     substitutionGroup="implementation"/>
336   <xsd:complexType name="BpelImplementation">
337     <xsd:complexContent>
338       <xsd:extension base="Implementation">
339         <xsd:sequence>
340           <xsd:any namespace="##other" processContents="lax"
341             minOccurs="0" maxOccurs="unbounded"/>
342         </xsd:sequence>
343         <xsd:attribute name="process" type="xsd:QName"
344           use="required"/>
345         <xsd:anyAttribute namespace="##any" processContents="lax"/>
346       </xsd:extension>
347     </xsd:complexContent>
348   </xsd:complexType>
349
350   <xsd:element name="interface.partnerLinkType"
351     type="BpelPartnerLinkType"
352     substitutionGroup="interface"/>
353   <xsd:complexType name="BpelPartnerLinkType">
354     <xsd:complexContent>
355       <xsd:extension base="Interface">
356         <xsd:sequence>
357           <xsd:any namespace="##other" processContents="lax"
358             minOccurs="0" maxOccurs="unbounded"/>
359         </xsd:sequence>
360         <xsd:attribute name="type" type="xsd:QName"
361           use="required"/>
362         <xsd:attribute name="serviceRole" type="xsd:NCName"
363           use="optional"/>
364         <xsd:anyAttribute namespace="##any" processContents="lax"/>
365       </xsd:extension>
366     </xsd:complexContent>
367   </xsd:complexType>
368 </xsd:schema>

```

369

370

371 XML Schema sca-bpel.xsd

```

372 <xsd:schema xmlns="http://www.oesa.org/xmlns/sca/bpel/1.0"
373

```

```

374 targetNamespace="http://www.oesa.org/xmlns/sca/bpel/1.0"
375 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
376 xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
377 elementFormDefault="qualified">
378
379 <xsd:import namespace="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
380   schemaLocation="ws-bpel_serviceref.xsd" />
381
382 <xsd:simpleType name="tMultiplicity">
383   <xsd:restriction base="xsd:string">
384     <xsd:enumeration value="0..n"/>
385     <xsd:enumeration value="1..n"/>
386   </xsd:restriction>
387 </xsd:simpleType>
388
389 <xsd:simpleType name="tBoolean">
390   <xsd:restriction base="xsd:string">
391     <xsd:enumeration value="yes"/>
392     <xsd:enumeration value="no"/>
393   </xsd:restriction>
394 </xsd:simpleType>
395
396 <xsd:attribute name="property" type="tBoolean"/>
397
398 <xsd:attribute name="multiRefFrom" type="xsd:NCName"/>
399
400 <xsd:element name="multiReference" type="tMultiReference"/>
401 <xsd:complexType name="tMultiReference">
402   <xsd:attribute name="partnerLinkType" type="xsd:QName"
403     use="required"/>
404   <xsd:attribute name="partnerRole" type="xsd:NCName"
405     use="required"/>
406   <xsd:attribute name="multiplicity" type="tMultiplicity"
407     use="optional" default="1..n"/>
408 </xsd:complexType>
409
410 <xsd:element name="serviceReferenceList"
411   type="tServiceReferenceList"/>
412 <xsd:complexType name="tServiceReferenceList">
413   <xsd:sequence>
414     <xsd:element ref="sref:service-ref"
415       minOccurs="0" maxOccurs="unbounded"/>
416   </xsd:sequence>
417 </xsd:complexType>
418 </xsd:schema>

```

419

420

2.2. References

421

422 [1] SCA Assembly Model Specification, Version 1.0, February 2007

423

424 http://www.osoa.org/download/attachments/35/SCA_AssemblyModel_V100.pdf

425

426 [2] Web Services Business Process Execution Language Version 2.0, November 2006

427

<http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>

428

429 [3] Business Process Execution Language for Web Services Version 1.1, May 2003

430

<http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp>

431

<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>

432

<http://msdn2.microsoft.com/en-us/library/aa479359.aspx>

433

<https://www.sdn.sap.com/irj/sdn/developerareas/esa/standards>

434

<http://www.siebel.com>

435