Service Component Architecture WS-BPEL Client and Implementation Specification Version 1.1

Committee Draft 01

17 January 2008

Specification URIs:

This Version:

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

Previous Version:

N/A

Latest Version:

http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec.html
http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec.doc
http://docs.oasis-open.org/opencsa/sca-bpel/sca-bpel-1.1-spec.pdf

Technical Committee:

OASIS SCA-BPEL TC

Chair(s):

Anish Karmarkar, Oracle
Sanjay Patil, SAP

Editor(s):

Najeeb Andrabi, TIBCO Software
Martin Chapman, Oracle
Dieter König, IBM
Michael Rowley, BEA Systems
Ivana Trickovic, SAP
Alex Yiu, Oracle

Related work:

This specification is related to:

- Service Component Architecture – Assembly Model Specification – Version 1.1
- Service Component Architecture – Policy Framework Specification – Version 1.1
- Web Services – Business Process Execution Language – Version 2.0 – http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

Declared XML Namespace(s):

- **sca** – http://docs.oasis-open.org/ns/opencsa/sca/200712
- **sca-bpel** (defined here) – http://docs.oasis-open.org/ns/opencsa/sca-bpel/200801
- **bpel** – http://docs.oasis-open.org/wsbpel/2.0/process/executable
- **plnk** – http://docs.oasis-open.org/wsbpel/2.0/plnktype
- **sref** – http://docs.oasis-open.org/wsbpel/2.0/serviceref
- **wsdl** – http://schemas.xmlsoap.org/wsdl/
- **xsd** – http://www.w3.org/2001/XMLSchema

Abstract:

The Service Component Architecture (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 4.

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**.**

Status:

This document was last revised or approved by the [TC name | membership of OASIS] on the above date. The level of approval is also listed above. Check the "Latest Version" or "Latest Approved Version" location noted above for possible later revisions of this document.

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

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

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

Notices

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

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

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

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

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

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

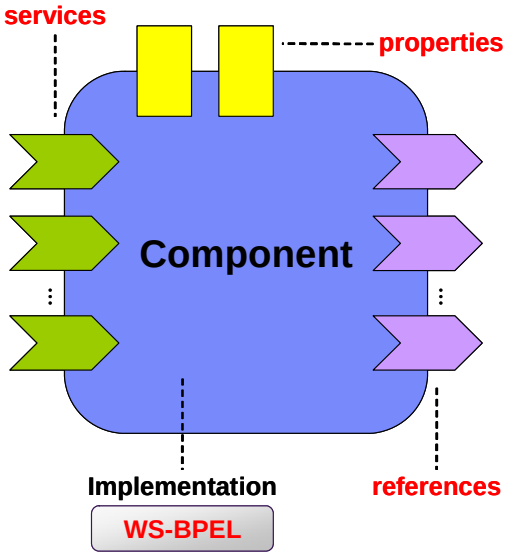
OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither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

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

Table of Contents

1 Introduction

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



Such a component definition has the following form:

```
<component ... >

   ...

   <implementation.bpel process="xs:QName" />

   ...

</component>
```

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

1.1 Terminology

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

1.2 Normative References

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, http://www.ietf.org/rfc/rfc2119.txt, IETF RFC 2119, March 1997.
[SCA-Assembly]	*Service Component Architecture – Assembly Model Specification – Version 1.1*, (insert link here)
[WS-BPEL]	*Web Services – Business Process Execution Language – Version 2.0*, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
[Reference]	[Full reference citation]

1.3 Non-Normative References

[Reference] [Full reference citation]

2 Component Types defined by WS-BPEL Processes

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

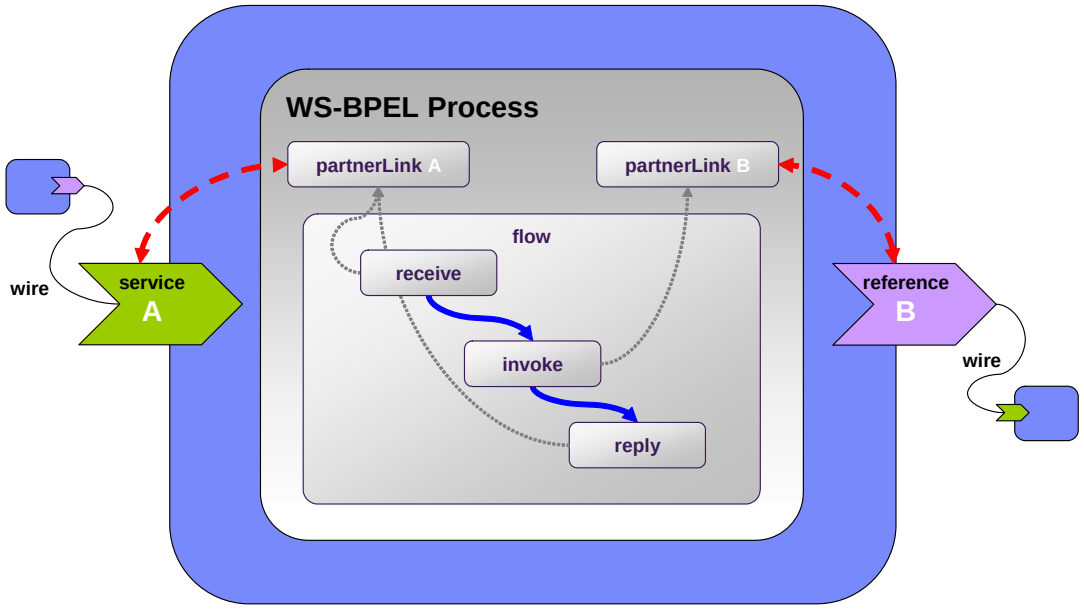
```
<componentType ... >

   <service name="xs:NCName" ... > ... </service>

   <reference name="xs:NCName" ... > ... </reference>

   <property name="xs:NCName" ... > ... </property>

   <implementation ... />

</componentType>
```

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

2.1 Services and References

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

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



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

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

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

1. **Multi-Reference**. If the partner link is declared with sca-bpel:multiRefFrom="aVariableName" extension, the multiplicity of the SCA reference will be determined by the multiplicity attribute of sca-bpel:multiReference extension used in the corresponding variable. The multiplicity declaration of the variable which is either 0..n or 1..n. Details of these extensions are described in section 3.2.
2. **Required Reference**. If not (1) and the partner link has `initializePartnerRole="yes"`, then the multiplicity is 1..1 (i.e. it's a *required reference*).
3. **Stub Reference**. If not (1) or (2) and if the analysis of the process determines that the first use of the partner link by any activity is in an assign activity that sets the partner role, then the multiplicity is "0..1" and the attribute `wiredByImpl` is set to `"true"`. A reference with `wiredByImpl="true"` is referred to as a *stub reference*. Although the target can't be set for such a reference, SCA can still apply bindings and policies to it and may need to set the endpoint address for callbacks, if the interface is bi-directional.
4. **Optional Reference**. If not (1) or (2) or (3) then the `multiplicity="0..1"`.

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

2.2 PartnerLinkTypes and SCA Interfaces

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

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

```
<plnk:partnerLinkType name="invoicingLT">
   <plnk:role name="invoiceService"
      portType="pos:computePricePT" />
   <plnk:role name="invoiceRequester"
      portType="pos:invoiceCallbackPT" />
</plnk:partnerLinkType>
```

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

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

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

```
<receive partnerLink="invoicing"
   portType="pos:computePricePT"
   operation="initiatePriceCalculation"
   variable="PO"
   createInstance="yes" />
```

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

```
<service name="invoicing">
   <interface.wsdl
      interface="http://manufacturing.org/wsdl/purchase#
                           wsdl.interface(computePricePT)"
      callbackInterface="http://manufacturing.org/wsdl/purchase#
                              wsdl.interface(invoiceCallbackPT)" />
</service>
```

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

2.3 Specifying an SCA interface with a partnerLinkType

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

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

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

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

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

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

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

2.4 Handling of Local PartnerLinks

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

- Suppose "originalName" is the original NCName used in multiple partnerLink declarations
- When these partnerLinks are exposed to SCA assembly, these partnerLinks will given aliases from "_orginalName_1" to "_orginalName_N" regardless of how partnerLink participate in SCA assembly (i.e. services vs. references) and the number suffixes are based on the lexical order of the corresponding partnerLink occurrences in the process definition.
- If any "_orginalName_i" (where 1 <= i <= N) is already taken by existing partnerLink declaration in the process definition, additional underscore characters may be added at the beginning of all aliases consistently to avoid collision.

2.5 Support for conversational interfaces

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

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

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

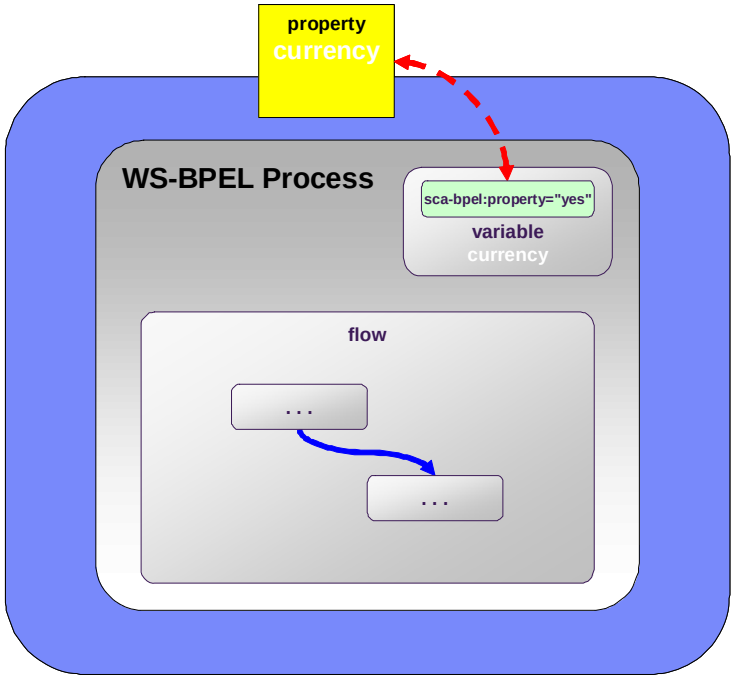
3 SCA Extensions to WS-BPEL

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

```
<process ...>
   <extensions>
      <extension namespace="http://www.osoa.org/xmlns/sca/bpel/1.0"
         mustUnderstand="yes" />
   </extensions>
   ...
</process>
```

3.1 Properties

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



The declaration looks like the following:

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

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

If the variable has an initialization from-spec, then that becomes the default value for the variable in cases where the SCA component does not provide a value for that property.

If the from-spec is a literal value, where it has the following form:

```
    <from><literal>literal value</literal></from>
```

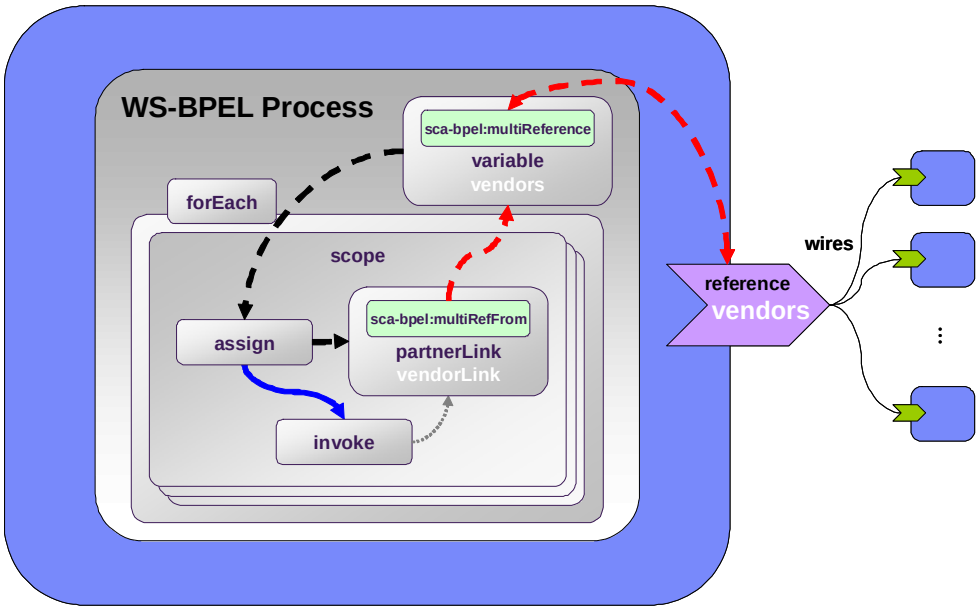
then the literal value will be represented as the default value in the component type for the process. Any other kind of initialization from-spec will not be represented in the component type. However, even though the other kinds of initialization from-spec are not represented in the component type, they would still be computed and used as the default value for the property when the component does not provide a value for that property.

If a value is provided for the property, any initialization from-spec MUST still be evaluated, but the value of the variable will be changed to the provided property value immediately after the initialization is evaluated, and specifically, before any following variable initialization from-spec is evaluated. Thus, any side effects that result from the execution of the initialization from-spec will occur irrespective of whether the property is set.

If a BPEL variable that is used as a property has an initialization from-spec then mustSupply="false" must be specified on the component type property declaration, even if the default value is not literal and therefore not represented in the component type.

3.2 Multi-Valued References

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



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

```
<variable name="vendors" element="sca-bpel:serviceReferenceList">
   <sca-bpel:multiReference partnerLinkType="pos:vendorPT"
      partnerRole="vendor" />
</variable>
```

Syntax of this extension:

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

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

The sca-bpel:serviceReferenceList element declaration is the following:

```
<xsd:element name="serviceReferenceList">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element ref="sref:service-ref"
            minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

A typical use of a variable that holds a multi-valued reference would be to have a <forEach> activity with an iteration for each element in the list. The body of the <forEach> activity would declare a local partner link and assign one of the list elements to the local partner link. Such a local partner link is typically categorized as the “References” case 1 listed in section 2.1.

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

```
<partnerLink ... sca-bpel:multiRefFrom="xs:NCName" />
```

The attribute value must refer to the name of a variable manifesting an SCA multi-valued reference. The partnerLinkType and partnerRole attributes of the partner link and multi-valued reference variable must be matched. Also, there must be at least one code-path that values from the multi-valued reference variable are copied to the partnerRole of the partner link.

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

When this sca-bpel:multiRefFrom extension is applied to pair up a multi-valued reference variable and a partner link which is categorized as the “References” case 1 (as described in section 2.1), the partner link and variable are manifested as a single multi-valued reference entity in SCA assembly model using the name of the variable. If the interface involved is bi-directional, this implies the wiring of the bi-directional interface as a single reference in SCA.

For example:

```
<process>
   ...
   <variable name="vendors" element="sca-bpel:serviceReferenceList">
      <sca-bpel:multiReference partnerLinkType="pos:vendorPT"
         partnerRole="vendor" />
   </variable>
   ...
   <forEach counterName="idx" ...>
      <startCounterValue>1</startCounterValue>
      <finalCounterValue>
         count($vendors/bpel:service-ref)
      </finalCounterValue>
      ...
      <scope>
```

```
            ...
            <partnerLink name="vendorLink"
               partnerLinkType="pos:vendorPT"
               partnerRole="vendor"
               myRole="quoteRequester"
               sca-bpel:multiRefFrom="vendors" />
            ...
            <assign>
               <copy>
                  <from>$vendors/bpel:service-ref[$idx]</from>
                  <to partnerLink="vendorLink" />
               </copy>
            </assign>
            ...
         </scope>
      </forEach>
      ...
</process>
```

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

4 Using BPEL4WS 1.1 with SCA

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

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

5 Conformance

(tbd.)

A. Acknowledgements

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

Members of the SCA-BPEL Technical Committee:

Najeeb Andrabi, TIBCO Software Inc.
Graham Barber, IBM
William Barnhill, Booz Allen Hamilton
Charlton Barreto, Adobe Systems
Hanane Becha, Nortel
Michael Beisiegel, IBM
Jeffrey Bik, Active Endpoints, Inc.
David Burke, TIBCO Software Inc.
Fred Carter, AmberPoint
Martin Chapman, Oracle Corporation
Eric Clairambault, IBM
James Bryce Clark, OASIS
Mark Combellack, Avaya, Inc.
Kevin Conner, Red Hat
Robin Cover, OASIS
Jean-Sebastien Delfino, IBM
Jacques Durand, Fujitsu Limited
Mike Edwards, IBM
Raymond Feng, IBM
Mark Ford, Active Endpoints, Inc.
Genadi Genov, SAP AG
Alejandro Guizar, Red Hat
Uday Joshi, Oracle Corporation
Khanderao Kand, Oracle Corporation
Anish Karmarkar, Oracle Corporation
Jason Kinner, Oracle Corporation
Dieter Koenig, IBM
Rich Levinson, Oracle Corporation
Mark Little, Red Hat
Ole Madsen, OIOXML eBusiness Standardization Group
Ashok Malhotra, Oracle Corporation
Keith McFarlane, Avaya, Inc.
Mary McRae, OASIS
Jeff Mischkinsky, Oracle Corporation
Simon Moser, IBM

Sanjay Patil, SAP AG
Michael Pellegrini, Active Endpoints, Inc.
Luciano Resende, IBM
Michael Rowley, BEA Systems, Inc.
Clifford Thompson, Individual
Ivana Trickovic, SAP AG
Danny van der Rijn, TIBCO Software Inc.
Mark Walker, Avaya, Inc.
Prasad Yendluri, Software AG, Inc.
Alex Yiu, Oracle Corporation

OSOA Contributors:

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

B. Non-Normative Text

C. Revision History

[optional; should not be included in OASIS Standards]

Revision	Date	Editor	Changes Made
2	2007-10-10	Dieter König	Issue resolutions BPEL-4, BPEL-7 New section “5. Conformance” List of XML namespaces Table of Contents formatting References formatting Syntax and Examples formatting
3	2007-10-10	Dieter König	Reduced component/composite syntax in sections 1 and 2
4	2007-12-05	Dieter König	Issue resolutions BPEL-5, BPEL-6, BPEL-9, BPEL-13 Document title according to OASIS rules
5	2008-01-11	Michael Rowley	Issue resolution for BPEL-11
6	2008-01-17	Dieter König	Approved Committee Draft