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

License

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.

Table of Contents

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.

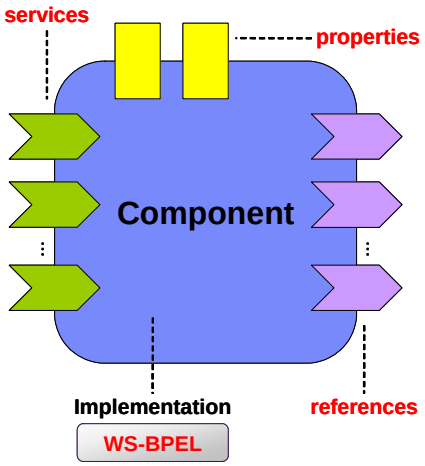
Start from WS-BPEL process. It should be possible to use any valid WS-BPEL process definition as the implementation of a component within SCA. In particular, it should be possible to generate an SCA Component Type from any WS-BPEL process definition and use that type within an SCA assembly. Most BPEL4WS 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>
```

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.

1.3. 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 [1] as follows:

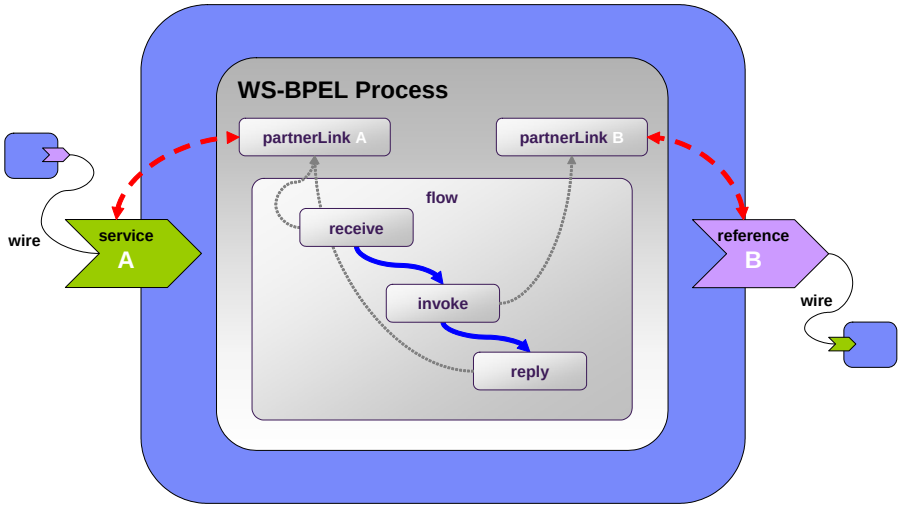
```
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0" >

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

   <reference name="xs:NCName" override="sca:OverrideOptions"?
               multiplicity="0..1 or 1..1 or 0..n or 1..n"?>*
         <interface/>
   </reference>

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

</componentType>
```

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

1.3.1. 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 has a corresponding SCA service in the component type. If the partner link declaration has `initializePartnerRole="yes"`, then the service must be wired 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:multiRefFrom="aVariableName"` extension, the multiplicity of the SCA reference will be determined by the `multiplicity` attribute of `sca: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 1.4.2.
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).

1.3.2. 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.

1.3.3. 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.

1.3.4. 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)
- 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 the aliases to avoid collision.

1.3.5. Support for conversational interfaces

WS-BPEL can be used to implement an SCA Component with *conversational* services. See the SCA Assembly Specification [1] 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 [1], 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.

1.4. SCA Extensions to WS-BPEL

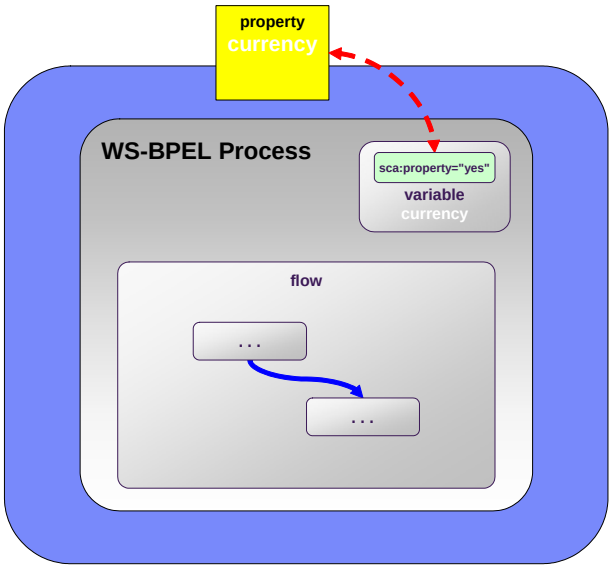
It is possible to use WS-BPEL processes in conjunction with SCA, while the processes have no knowledge of SCA. A few SCA concepts are only available to WS-BPEL processors that support SCA specific extensions. The capabilities that require knowledge of SCA are provided by an SCA extension, which must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

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

```
        </extensions>
```

1.4.1. 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:

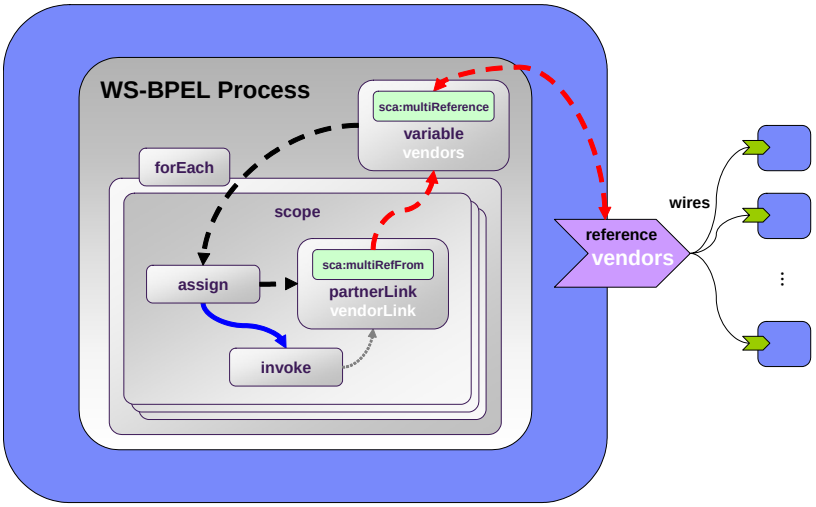
```
    <variable name="currency" type="xsd:string" sca:property="yes"/>
```

When `sca: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.

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

1.4.2. 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:multiReference` extension element that declares that the variable represents a multi-valued reference. The type of the variable must be an element of `sca:serviceRefList`. However, since that type only specifies that the variable holds a list of endpoint references, the `sca: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:serviceReferenceList">
   <sca:multiReference partnerLinkType="pos:vendorPT" partnerRole="vendor"/>
</variable>
```

Syntax of this extension:

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

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

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

```
<xsd:element name="serviceReferenceList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="bpel: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 1.3.1.

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

```
sca: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: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 1.3.1), 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:

```
<variable name="vendors" element="sca:serviceReferenceList">
   <sca:multiReference partnerLinkType="pos:vendorPT" partnerRole="vendor"/>
</variable>
 ...
<forEach counterName="idx" ...>
   <startCounterValue>1</startCounterValue>
   <finalCounterValue>count($vendors/bpel:service-ref)</finalCounterValue>
   ...
   <scope>
     ...
     <partnerLink name="vendorLink"
                  partnerLinkType="pos:vendorPT"
                  partnerRole="vendor"
                  myRole="quoteRequester"
                  sca:multiRefFrom="vendors" />
      ...
      <assign>
         <copy>
            <from>$vendors/bpel:service-ref[$idx]</from>
            <to partnerLink="vendorLink" />
         </copy>
      </assign>
      ...
   </scope>
</forEach>
```

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: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.

1.5. 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 1.2 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.

2. Appendix

2.1. XML Schema

XML Schema sca-implementation-bpel.xsd

```
<xsd:schema xmlns="http://www.osoa.org/xmlns/sca/1.0"
   targetNamespace="http://www.osoa.org/xmlns/sca/1.0"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified">

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

   <xsd:element name="implementation.bpel"
      type="BpelImplementation"
      substitutionGroup="implementation"/>
   <xsd:complexType name="BpelImplementation">
      <xsd:complexContent>
         <xsd:extension base="Implementation">
               <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="process" type="xsd:QName"
               use="required"/>
         <xsd:anyAttribute namespace="##any" processContents="lax"/>
      </xsd:extension>
   </xsd:complexContent>
   </xsd:complexType>

   <xsd:element name="interface.partnerLinkType"
      type="BpelPartnerLinkType"
      substitutionGroup="interface"/>
   <xsd:complexType name="BpelPartnerLinkType">
      <xsd:complexContent>
         <xsd:extension base="Interface">
               <xsd:sequence>
            <xsd:any namespace="##other" processContents="lax"
                  minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
         <xsd:attribute name="type" type="xsd:QName"
               use="required"/>
         <xsd:attribute name="serviceRole" type="xsd:NCName"
               use="optional"/>
         <xsd:anyAttribute namespace="##any" processContents="lax"/>
      </xsd:extension>
   </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

XML Schema sca-bpel.xsd

```
<xsd:schema xmlns="http://www.osoa.org/xmlns/sca/bpel/1.0"
```

```
        targetNamespace="http://www.osoa.org/xmlns/sca/bpel/1.0"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
        elementFormDefault="qualified">

        <xsd:import namespace="http://docs.oasis-open.org/wsbpel/2.0/serviceref"
         schemaLocation="ws-bpel_serviceref.xsd" />

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

        <xsd:simpleType name="tBoolean">
             <xsd:restriction base="xsd:string">
                 <xsd:enumeration value="yes"/>
                 <xsd:enumeration value="no"/>
             </xsd:restriction>
        </xsd:simpleType>

        <xsd:attribute name="property" type="tBoolean"/>

        <xsd:attribute name="multiRefFrom" type="xsd:NCName"/>

        <xsd:element name="multiReference" type="tMultiReference"/>
        <xsd:complexType name="tMultiReference">
           <xsd:attribute name="partnerLinkType" type="xsd:QName"
                    use="required"/>
              <xsd:attribute name="partnerRole" type="xsd:NCName"
                    use="required"/>
              <xsd:attribute name="multiplicity" type="tMultiplicity"
                    use="optional" default="1..n"/>
        </xsd:complexType>

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

2.2. References

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

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

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

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

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

http://dev2dev.bea.com/technologies/webservices/BPEL4WS.jsp
http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
http://msdn2.microsoft.com/en-us/library/aa479359.aspx
https://www.sdn.sap.com/irj/sdn/developerareas/esa/standards
http://www.siebel.com