

Universität Stuttgart

Fakultät Informatik, Elektrotechnik und Informationstechnik

**WS-BPEL Extension for Semantic Web
Services (BPEL4SWS), Version 1.0**

Dimka Karastoyanova, Tammo van Lessen,
Frank Leymann, Jörg Nitzsche, Daniel Wutke

Report 2008/03
April, 2008



**Institut für Architektur von
Anwendungssystemen**

Universitätsstr. 38
70569 Stuttgart
Germany

CR: D.2.12, D.2.13, D.3.3, H.4.1, I.2.4

WS-BPEL Extension for Semantic Web Services (BPEL4SWS), Version 1.0

April 2008

Editor

Jörg Nitzsche

Authors (in alphabetical order)

Dimka Karastoyanova
Tammo van Lessen
Frank Leymann
Jörg Nitzsche
Daniel Wutke

Licence

Permission to copy and display the WS-BPEL Extension for Semantic Web Services Specification (the "Specification", which includes WSMO, (SA)WSDL and schema documents), in any medium without fee or royalty is hereby granted, provided that you include the following on ALL copies of the WS-BPEL Extension for Semantic Web Services Specification, or portions thereof, that you make:

1. A link or URL to the Specification at one of the Authors' websites.
2. The copyright notice as shown in the Specification.

The Authors agree to grant you a license, under royalty-free and otherwise reasonable, non-discriminatory terms and conditions, to their respective essential patent claims that they deem necessary to implement the Specification.

THE SPECIFICATION IS PROVIDED "AS IS," AND THE AUTHORS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR TITLE; THAT THE CONTENTS OF THE SPECIFICATION ARE SUITABLE FOR ANY PURPOSE; NOR THAT THE IMPLEMENTATION OF SUCH CONTENTS WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

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 SPECIFICATION.

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

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

Abstract

The Web Services Business Process Execution Language, version 2.0 (WS-BPEL 2.0 or BPEL for brevity) introduces a model for business processes based on Web services. A BPEL process orchestrates interactions among different Web services. The language encompasses features needed to describe complex control flows, including error handling and compensation behavior.

BPEL for Semantic Web Services (BPEL4SWS) uses Semantic Web Service Frameworks to define a communication channel between two partner services instead of using the partner link which is based on WSDL 1.1. It enables describing activity implementations in a much more flexible manner based on ontological descriptions of service requesters and providers.

Status

BPEL4SWS is provided as-is and for review and evaluation only. The authors hope to solicit your contributions and suggestions in the near future. The authors make no warranties or representations regarding the specifications in any manner whatsoever.

Table of Contents

1	<i>Introduction</i>	5
2	<i>Language Design</i>	6
2.1	Dependencies on Other Specifications	6
2.2	Notational Conventions	6
2.3	Namespaces	6
2.4	Language Extensibility	7
3	<i>Defining a BPEL4SWS process</i>	8
3.1	Initial Example	8
3.2	Overall Language Structure	20
4	<i>Data Handling</i>	23
4.1	Ontological Data Types	23
4.2	Mediation	24
4.3	Reasoning in BPEL4SWS	25
5	<i>WSDL-less interaction model</i>	29
5.1	Conversation	29
5.2	InteractionActivity	29
5.3	Pick	31
5.4	EventHandler	33
5.5	Partner	34
6	<i>Describing Interactions using the RO4SSOA</i>	36
7	<i>Grounding</i>	37
7.1	Syntax	37
7.2	Properties	37
7.3	Operational behaviour	38
8	<i>Acknowledgements</i>	40
9	<i>References</i>	40
10	<i>Non-normative references</i>	41
	<i>Appendix A – Standard Faults</i>	42
	<i>Appendix B – BPEL4SWS Schema</i>	43

1 Introduction

This specification introduces an extension to BPEL to enable describing interaction using semantic Web service Frameworks instead of using WSDL 1.1 [WSDL 1.1]. Semantic Web services (SWS) can be considered an integration layer on top of Web services; they use ontologies as data model and they have a rich conceptual model. There are efforts towards standardizing this conceptual model within the Reference Ontology for Semantic Service Oriented Architectures (RO4SSOA) [RO4SSOA].

The RO4SSOA is as an extension of the SOA-RM [SOARM], informed by existing semantic Web service approaches such as the Web Service Modelling Ontology (WSMO) [WSMO] and the OWL Services Ontology (OWL-S) [OWL-S].

It formalises the concept of a *service* described in terms of its *interfaces* and its *capability*, as in the SOA-RM. In contrast to the SOA-RM it also explicitly models the required capability and possible interactions of the service consumer. The concept used to contain these descriptions is called a *goal*.

In addition to the SWS based interaction, BPEL4SWS makes use of annotated data types to enhance data handling by means of ontological mediators and uses ontological reasoning to evaluate conditions.

2 Language Design

The BPEL4SWS extension is defined in a way that it is layered on top of BPEL so that its features can be composed with BPEL features whenever needed. All elements and attributes introduced in this extension are made available to both BPEL executable processes and abstract processes.

This extension introduces a set of elements and attributes to enable defining interaction that is independent of WSDL and ontological mediation.

2.1 Dependencies on Other Specifications

BPEL4SWS utilizes the following specifications:

- WS-BPEL 2.0: BPEL4SWS extends the WS-BPEL 2.0 process model and uses existing WS-BPEL 2.0 capabilities.
- SAWSDL: BPEL4SWS uses SAWSDL to annotate data types of variables used in a process definition.

2.2 Notational Conventions

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

2.3 Namespaces

This specification uses a number of namespace prefixes throughout; they are listed in Table 1. Note that the choice of any namespace prefix is arbitrary and not semantically significant (see [XML Namespaces]).

Prefix	Namespace
bpel	http://docs.oasis-open.org/wsbpel/2.0/process/executable
b4s	http://www.iaas.uni-stuttgart.de/bpel4sws/executable
prg	http://www.iaas.uni-stuttgart.de/bpel4sws/wsdgrounding
wSDL	http://schemas.xmlsoap.org/wSDL/
xsd	http://www.w3.org/2001/XMLSchema

Table 1 Prefixes and namespaces used in this specification

All information items defined by BPEL4SWS are identified by the XML namespace URI [XML Namespaces] <http://www.iaas.uni-stuttgart.de/bpel4sws/executable>. A normative XML Schema [XML Schema Part 1, Part 2] document can be obtained by dereferencing the XML namespace URI.

2.4 Language Extensibility

The BPEL4SWS specification extends the reach of the standard BPEL extensibility mechanism to BPEL4SWS elements. This allows:

- Attributes from other namespaces to appear on any BPEL4SWS element
- Elements from other namespaces to appear within BPEL4SWS elements

Extension attributes and extension elements **MUST NOT** contradict the semantics of any attribute or element from the BPEL4SWS namespace.

Standard BPEL element `<extension>` must be used to declare mandatory and optional extensions of BPEL4SWS.

3 Defining a BPEL4SWS process

3.1 Initial Example

Before describing the structure of business processes in detail, this section presents a simple example (inspired by the example given in the WS-BPEL specification) of a BPEL4SWS process for handling a purchase order. The aim is to introduce the most basic structures and some of the fundamental concepts of the extensions.

“On receiving the purchase order from a customer, the process initiates two paths concurrently: calculating the final price for the order, selecting a shipper [...], and scheduling the production and shipment for the order. While some of the processing can proceed concurrently, there are control and data dependencies between the three paths. In particular, the shipping price is required to finalize the price calculation, and the shipping date is required for the complete fulfillment schedule. When the three concurrent paths are completed, invoice processing can proceed and the invoice is sent to the customer.” [WS-BPEL 2.0]

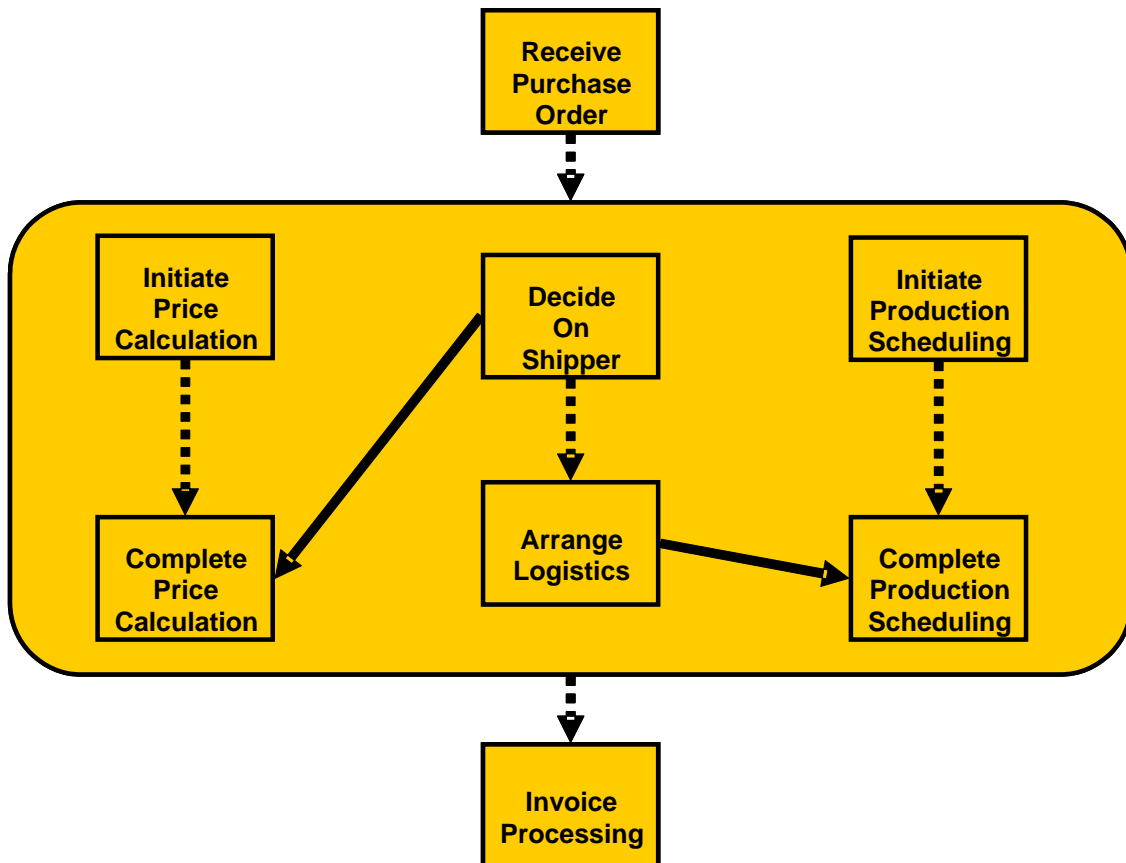


Figure 1: Purchase Order Process – Outline [WS-BPEL 2.0]

In contrast to the WS-BPEL process defined in the BPEL specification, in the BPEL4SWS process presented below not all communication channels are modelled using a `partnerLink`. The communication with the shipper is modelled using WSDL-less interaction activities forming a *conversation*. A *goal* description is attached which describes the goal of this particular conversation in terms of its requested *capability* and its message exchange (*interface*, according to the RO4SSOA). The goal matches a *service* in case (i) the capability the service provides and the capability the goal requests match and (ii) the message exchange the service can involve in matches the message exchange the goal requests. The message exchange of both, the goal and the service, is grounded to WSDL port types in a manner that is compliant with the Basic Profile 1.1 [WS-I Basic Profile] of the WS-Interoperability organization. The first two messages, i.e. the `shippingRequestMessage` and the `shippingInfoMessage`, are not grounded to a particular port type. They are sent to the port type of the service that has been discovered during runtime. The third message, the `scheduleMessage` is grounded to the `shippingCallbackPT`. This port type is used by the service to provide the shipping date to the process. The process grounding binds operations of the `shippingCallbackPT` to activities in the process model.

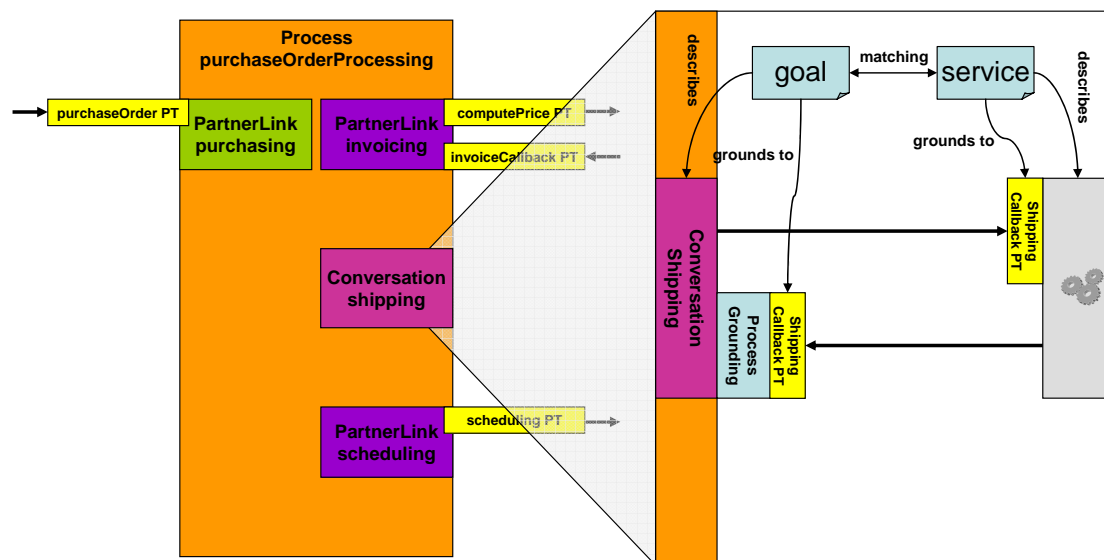


Figure 2: Purchase Order Process – Communication Channels

```

<process name="purchaseOrderProcess"
  targetNamespace="http://example.org/bpel4sws/purchase"
  xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:lns="http://manufacturing.org/wsd/purchase"
  xmlns:b4s=" http://www.iaas.uni-stuttgart.de/BPEL4SWS/executable">

  <documentation xml:lang="EN">
    A simple example of a BPEL4SWS process for handling a purchase
    order.
  </documentation>

  <extensions>
    <extension namespace="http://www.iaas.uni-stuttgart.de/
      bpel4sws/executable"
      mustUnderstand="yes"/>
  </extensions>

  <b4s:conversations>
    <b4s:conversation name="shipping"
      hasGoal="http://example.org/bpel4sws/goal"/>
  </b4s:conversations>

  <partnerLinks>
    <partnerLink name="purchasing"
      partnerLinkType="lns:purchasingLT"
      myRole="purchaseService" />
    <partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
      myRole="invoiceRequester" partnerRole="invoiceService" />
    <partnerLink name="scheduling"
      partnerLinkType="lns:schedulingLT"
      partnerRole="schedulingService" />
  </partnerLinks>

  <variables>
    <variable name="PO" messageType="lns:POMessage" />
    <variable name="Invoice" messageType="lns:InvMessage" />
    <variable name="shippingRequest"
      messageType="lns:shippingRequestMessage" />
    <variable name="shippingInfo"
      messageType="lns:shippingInfoMessage" />
    <variable name="shippingSchedule"
      messageType="lns:scheduleMessage" />
  </variables>

```

```

<faultHandlers>
  <catch faultName="lns:cannotCompleteOrder"
    faultVariable="POFault"
    faultMessageType="lns:orderFaultType">
    <reply partnerLink="purchasing"
      portType="lns:purchaseOrderPT"
      operation="sendPurchaseOrder" variable="POFault"
      faultName="cannotCompleteOrder" />
    </catch>
</faultHandlers>

<sequence>
  <receive partnerLink="purchasing"
    portType="lns:purchaseOrderPT"
    operation="sendPurchaseOrder" variable="PO"
    createInstance="yes">
  </receive>

  <flow>
    <links>
      <link name="ship-to-invoice" />
      <link name="ship-to-scheduling" />
    </links>
    <sequence>
      <assign>
        <copy>
          <from>$PO.customerInfo</from>
          <to>$shippingRequest.customerInfo</to>
        </copy>
      </assign>
      <extensionActivity>
        <b4s:interactionActivity
          name="decideOnShipper"
          conversation="shipping"
          inputVariable="shippingRequest"
          outputVariable="shippingInfo">
          <sources>
            <source linkName="ship-to-invoice" />
          </sources>
        </b4s:interactionActivity>
      </extensionActivity>
      <extensionActivity>
        <b4s:interactionActivity

```

```

        name="arrangeLogistics"
        conversation="shipping"
        outputVariable="shippingSchedule">
        <sources>
            <source linkName="ship-to-scheduling" />
        </sources>
    </b4s:interactionActivity>
</extensionActivity>
</sequence>
<sequence>
    <invoke partnerLink="invoicing"
        portType="lns:computePricePT"
        operation="initiatePriceCalculation"
        inputVariable="PO">
        <documentation>
            Initial Price Calculation
        </documentation>
    </invoke>
    <invoke partnerLink="invoicing"
        portType="lns:computePricePT"
        operation="sendShippingPrice"
        inputVariable="shippingInfo">
        <documentation>
            Complete Price Calculation
        </documentation>
        <targets>
            <target linkName="ship-to-invoice" />
        </targets>
    </invoke>
    <receive partnerLink="invoicing"
        portType="lns:invoiceCallbackPT"
        operation="sendInvoice" variable="Invoice" />
</sequence>
<sequence>
    <invoke partnerLink="scheduling"
        portType="lns:schedulingPT"
        operation="requestProductionScheduling"
        inputVariable="PO">
        <documentation>
            Initiate Production Scheduling
        </documentation>
    </invoke>
    <invoke partnerLink="scheduling"
        portType="lns:schedulingPT"

```

```

        operation="sendShippingSchedule"
        inputVariable="shippingSchedule">
        <documentation>
            Complete Production Scheduling
        </documentation>
        <targets>
            <target linkName="ship-to-scheduling" />
        </targets>
    </invoke>
</sequence>
</flow>
<reply partnerLink="purchasing" portType="lns:purchaseOrderPT"
    operation="sendPurchaseOrder" variable="Invoice">
    <documentation>Invoice Processing</documentation>
</reply>
</sequence>
</process>

```

Listing 1: Purchase Order Process

“The WSDL port type offered by the service to its customers (purchaseOrderPT) is shown in the following WSDL document. Other WSDL definitions required by the business process are included in the same WSDL document for simplicity; in particular, the port types for the Web services providing price calculation, shipping, and production scheduling functions are also defined there” [WS-BPEL 2.0]. Note that in contrast to the example in the BPEL 2.0 specification, no partnerLinkType is defined for the shipping. The messages used during the shipping conversation are annotated using SAWSDL annotated data types.

```

<wsdl:definitions
  targetNamespace="http://manufacturing.org/wsdl/purchase"
  xmlns:sns="http://manufacturing.org/xsd/purchase"
  xmlns:pos="http://manufacturing.org/wsdl/purchase"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <wsdl:types>
    <xsd:schema>
      <xsd:import namespace="http://manufacturing.org/xsd/
        purchase"
        schemaLocation="http://manufacturing.org/xsd/
        purchase.xsd" />
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="POMessage">
    <wsdl:part name="customerInfo" type="sns:customerInfoType" />
    <wsdl:part name="purchaseOrder" type="sns:purchaseOrderType" />
  </wsdl:message>

  <wsdl:message name="InvMessage">
    <wsdl:part name="IVC" type="sns:InvoiceType" />
  </wsdl:message>

  <wsdl:message name="orderFaultType">
    <wsdl:part name="problemInfo" element="sns:OrderFault" />
  </wsdl:message>

  <wsdl:message name="shippingRequestMessage"
    sawsdl:modelReference="http://example.org/shippingOntology#
    shippingRequest"
    sawsdl:loweringSchemaMapping="http://example.org/..."
    sawsdl:liftingSchemaMapping="http://example.org/...">
    <wsdl:part name="customerInfo" element="sns:customerInfo" />
  </wsdl:message>

  <wsdl:message name="shippingInfoMessage"
    sawsdl:modelReference="http://example.org/shippingOntology#
    shippingInfo"
    sawsdl:loweringSchemaMapping="http://example.org/..."
    sawsdl:liftingSchemaMapping="http://example.org/...">

```

```

    <wsdl:part name="shippingInfo" element="sns:shippingInfo" />
</wsdl:message>
<wsdl:message name="scheduleMessage"
  sawsdl:modelReference="http://example.org/shippingOntology#
    shippingSchedule"
  sawsdl:loweringSchemaMapping="http://example.org/..."
  sawsdl:liftingSchemaMapping="http://example.org/...">
  <wsdl:part name="schedule" element="sns:scheduleInfo" />
</wsdl:message>

<!-- portTypes supported by the purchase order process -->
<wsdl:portType name="purchaseOrderPT">
  <wsdl:operation name="sendPurchaseOrder">
    <wsdl:input message="pos:POMessage" />
    <wsdl:output message="pos:InvMessage" />
    <wsdl:fault name="cannotCompleteOrder"
      message="pos:orderFaultType" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="invoiceCallbackPT">
  <wsdl:operation name="sendInvoice">
    <wsdl:input message="pos:InvMessage" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="shippingCallbackPT">
  <wsdl:operation name="sendSchedule">
    <wsdl:input message="pos:scheduleMessage" />
  </wsdl:operation>
</wsdl:portType>

<!-- portType supported by the invoice services -->
<wsdl:portType name="computePricePT">
  <wsdl:operation name="initiatePriceCalculation">
    <wsdl:input message="pos:POMessage" />
  </wsdl:operation>
  <wsdl:operation name="sendShippingPrice">
    <wsdl:input message="pos:shippingInfoMessage" />
  </wsdl:operation>
</wsdl:portType>

```



```

<!-- portType supported by the shipping service -->
<wsdl:portType name="shippingPT">
  <wsdl:operation name="requestShipping">
    <wsdl:input message="pos:shippingRequestMessage" />
    <wsdl:output message="pos:shippingInfoMessage" />
  </wsdl:operation>
</wsdl:portType>

<!-- portType supported by the production scheduling process -->
<wsdl:portType name="schedulingPT">
  <wsdl:operation name="requestProductionScheduling">
    <wsdl:input message="pos:POMessage" />
  </wsdl:operation>
  <wsdl:operation name="sendShipingSchedule">
    <wsdl:input message="pos:scheduleMessage" />
  </wsdl:operation>
</wsdl:portType>

<plnk:partnerLinkType name="purchasingLT">
  <plnk:role name="purchaseService"
    portType="pos:purchaseOrderPT" />
</plnk:partnerLinkType>

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

<plnk:partnerLinkType name="schedulingLT">
  <plnk:role name="schedulingService"
    portType="pos:schedulingPT" />
</plnk:partnerLinkType>
</wsdl:definitions>

```

Listing 2: WSDL defintion

The process grounding binds operations of the `shippingCallbackPT` to activities in the process model.

```
<prg:grounding
  processName="http://example.org/bpel4sws/purchase#
    purchaseOrderProcess"
  xmlns="http://www.iaas.uni-stuttgart.de/bpel4sws/grounding"
  xmlns:prg="http://www.iaas.uni-stuttgart.de/bpel4sws/grounding"
  xmlns:pos="http://manufacturing.org/wsdl/purchase">
  <prg:conversation name="shipping">
    <prg:activity name="arrangeLogistics"
      portType="pos:shippingCallbackPT"
      operation="sendSchedule"/>
  </prg:conversation>
</prg:grounding>
```

Listing 3: Process Grounding

According to the RO4SSOA, a goal, in this example a WSMO goal defined using WSML [WSML], is used to describe the `shipping` conversation of the `purchaseOrderProcess`.

```

namespace {_"http://example.org/bpel4sws#",
          so _"http://example.org/shippingOntology#"}

goal _"http://example.org/bpel4sws/goal"
...
capability
  precondition
    ...
  postcondition
    ...
  assumption
    ...
  effect
    ...

interface purchaseOrderProcess
  choreography
    stateSignature
    ...
  out
    concept so#shippingRequest
  in
    concept so#shippingInfo
    concept so#shippingSchedule withGrounding {
      _"http://manufacturing.org/wsdl/purchase#wsdl.
      interfaceMessageReference(shippingCallbackPT/
      sendSchedule/In)"
    }
  ...

```

Listing 4: WSMO goal

A service, also specified using WSMO/L, matches the requirements of the goal in terms of capability and message exchange.

```

namespace {_"http://example.org/bpel4sws#",
            so _"http://example.org/shippingOntology#"}

webService _"http://example.org/bpel4sws/service"
...
capability
  precondition
    ...
  postcondition
    ...
  assumption
    ...
  effect
    ...

interface shippingService
  choreography
    stateSignature
      ...
    in
      concept so#shippingRequest withGrounding {
        _"http://manufacturing.org/wsd/purchase#wsdl.
        interfaceMessageReference( shippingPT/requestShipping/In) "
      out
        concept so#shippingInfo withGrounding {
          _"http://manufacturing.org/wsd/purchase#wsdl.
          interfaceMessageReference( shippingPT/requestShipping/Out) "
        concept so#shippingSchedule
      ...

```

Listing 5: WSMO service

3.2 Overall Language Structure

This section provides a quick summary of BPEL4SWS extension elements, including the new activity type interaction activity, conversation and partner.

```
<bpel:process ...
  ...
  xmlns:b4s=" http://www.iaas.uni-stuttgart.de/BPEL4SWS/executable"
  ...
  <bpel:extensions>
    <bpel:extension
      namespace="http://www.iaas.uni-stuttgart.de/BPEL4SWS"
      mustUnderstand="yes"/>
  </bpel:extensions>
  ...
  <b4s:partners?>
    <b4s:partner name="NCName"
      businessEntity="QName">+
      <b4s:conversation name="NCName"/>+
    </b4s:partner>
  </b4s:partners>
  ...
  <b4s:conversations?>
    <b4s:conversation name="NCName">+
  </b4s:conversations>
  ...
  <bpel:assign validate="yes|no"? standard-attributes>
    standard-elements
    (
      <bpel:copy keepSrcElementName="yes|no"?>from-spec to-spec
    </bpel:copy>
      |
      <bpel:extensionAssignOperation>
        <b4s:mediate name="NCName"
          mediatorURI="anyURI"?
          inputVariable="NCName"
          outputVariable="NCName"/>
      </bpel:extensionAssignOperation>
    )+
  </bpel:assign>
  ...
  <b4s:eventHandlers?>
    <b4s:eventHandler name="NCName"
      <b4s:onEvent name="NCName"
        variable="NCName"
        conversation="NCName">
```

```

        <bpel:correlations .../>?
        scope
        </b4s:onEvent>*
        <bpel:onAlarm .../>*

    </b4s:eventHandler>
</b4s:eventHandlers>
...
<bpel:extensionActivity>
    <b4s:interactionActivity name="NCName"
        inputVariable="NCName"?
        outputVariable="NCName"?
        conversation="NCName"
        createInstance="yes|no"?
        standard-attributes>
        standard-elements
        <bpel:correlations .../>?
    </b4s:interactionActivity>
</bpel:extensionActivity>
...
<bpel:extensionActivity>
    <b4s:pick name="NCName"
        createInstance="yes|no"?
        standard-attributes>
        standard-elements
        (<b4s:onMessage name="NCName"
            variable="NCName"
            conversation="NCName">
                <bpel:correlations .../>?
            activity
        </b4s:onMessage> |
        <bpel:onMessage .../>)+
        <bpel:onAlarm .../>*
    </b4s:pick>
</bpel:extensionActivity>
...
</bpel:process>

```

Listing 6: Overall Language Structure

A BPEL4SWS process must use BPEL4SWS extension elements. Therefore elements from the namespace BPEL4SWS MUST be understood.

BPEL4SWS uses SAWSDL to annotate variable types with ontological concepts. These annotations enable data manipulation on an ontological level. The `mediate` operation which makes use of the ontological knowledge is introduced as a child element of the `extensionAssignOperation`. The syntax and semantics of the `mediate` element is introduced in section 4.2.

BPEL4SWS introduces an interaction model that is independent of WSDL. The WSDL-less interaction model is based on the concept of a conversation which plays the role of a WSDL-less partnerLink. A <conversation> is defined in a <conversations> element like a <partnerLink> is defined in a <partnerLinks> element.

The new activity types, <b4s:interactionActivity> and <b4s:pick> as well as the <b4s:eventHandler> reference a conversation and this way are used to model interactions of a BPEL process with other services or processes. The new activities types are included in the BPEL activity <bpel:extensionActivity> which is used as wrapper.

BPEL4SWS enables constraining that several conversations have to be established with one single partner: a <partner> element may group several conversations. The <partner> elements are defined in a <partners> element.

The syntax and semantics of the elements forming the new interaction model are introduced in section 5.

All BPEL4SWS elements may use the element <b4s:documentation> to provide annotation for users. The content could be a plain text, HTML, and so on. The <b4s:documentation> element is optional and has the following syntax:

```
<b4s:documentation xml:lang="xsd:language">
...
</b4s:documentation>
```

Listing 7: Documentation element

4 Data Handling

4.1 Ontological Data Types

In BPEL4SWS variables can be defined as either WSDL message types, xsd elements, simple types or complex types like described in the WS-BPEL 2.0 specification, section 8.1.

```
<bpel:variables>?
  <bpel:variable name="BPELVariableName"
    messageType="QName" ?
    type="QName" ?
    element="QName" ?>+
  from-spec?
</bpel:variable>
</bpel:variables>
```

Listing 8: Variable Definition

SAWSDL annotations for XML data types is used to provide an ontological representation of the data [SAWSDL].

```
<xsd:element name="OrderRequest"
  sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/
    spec/ontology/purchaseorder#OrderRequest"
  sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/
    sawsdl/spec/mapping/
    RDFOnt2Request.xml"
  sawsdl:liftingSchemaMapping="http://www.w3.org/2002/ws/
    sawsdl/spec/mapping/
    Request2RDFOnt.xml">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="customerNo" type="xsd:integer" />
      <xsd:element name="orderItem" type="item" minOccurs="1"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xs:element>
```

Listing 9: SAXSD Sample

The attribute `modelReference` specifies the ontological concept, the XML data is assigned to. The `LiftingSchemaMapping` defines rules how to transform an XML instance of the given type into an ontological instance which is member of the specified concept. The `loweringSchemaMapping` defines rules how to transform an ontological instance which is member of the specified concept into an XML instance of the given type.

The information given in SAWSDL/XSD documents is used to transform XML instance data into its ontological representation and vice versa. In case an error occurs during lifting or lowering of data, a `liloFault` has to be thrown.

4.2 Mediation

Data manipulation in BPEL4SWS can be either implemented using copy statements like described in the WS-BPEL 2.0 specification, section 8.4 or using mediation. Especially, ontological mediation is defined as an `extensionAssignOperation`.

4.2.1 Syntax

Definition of `mediate`:

```
<bpel:assign validate="yes|no"? standard-attributes>
  standard-elements
  (
    <bpel:copy keepSrcElementName="yes|no"?>from-spec to-spec</copy>
    |
    <bpel:extensionAssignOperation>
      <b4s:mediate name="NCName"
        mediatorURI="anyURI"?
        inputVariable="NCName"
        outputVariable="NCName"/>
    </bpel:extensionAssignOperation>
  )+
</bpel:assign>
```

Listing 10: extensionAssignOperation: mediate

4.2.2 Properties

The `<b4s:mediate>` element has the following attribute:

- `name`: This attribute is used to unambiguously identify a `mediate` element in a BPEL4SWS process. This attribute is mandatory.
- `mediatorURI`: This attribute references a mediator that performs the data manipulation. This attribute is optional.
- `inputVariable`: This attribute specifies the input for the `mediate` operation. This attribute is mandatory.
- `outputVariable`: This attribute specifies the output of the `mediate` operation. This attribute is mandatory.

4.2.3 Operational behaviour

Prior to execution of the mediate operation, a mediation service that is able to mediate between the concept the `inputVariable` is annotated with and the concept the `outputVariable` is annotated **MUST** be discovered. In case a `mediatorURI` is specified the URI **MAY** be used/resolved to discover the mediation service. In case no mediation service was discovered or an error during mediation occurred a `mediationFailed` error **MUST** be thrown.

4.3 Reasoning in BPEL4SWS

In BPEL4SWS ontological reasoning can be used to evaluate expressions. In particular boolean expressions that are used to evaluate conditions, e.g. `joinConditions`, `transitionConditions`, exit conditions in `<while>` or `<repeatUntil>` activities, and conditions that decide which branch to take in an `<if>` activity.

The following example is used to illustrate the usage of reasoning in BPEL4SWS.

```
ontology _ "http://www.example.org/ontologies/example"

concept Human
  nonFunctionalProperties
    dc:description hasValue "concept of a human being"
  endNonFunctionalProperties
  ...

concept Man subConceptOf Human
  nfp
    dc#relation hasValue ManDisjointWoman
  endnfp
  ...

concept Woman subConceptOf Human
  nfp
    dc#relation hasValue ManDisjointWoman
  endnfp
  ...

axiom ManDisjointWoman
  definedBy
    !- ?x memberOf Man and ?x memberOf Woman.
```

Listing 11: Human Ontology

The variable `customer` is annotated with the concept `human` that is defined in the ontology presented above. This concept is further refined into `man` and `woman`, i.e. the variable may also contain instances of type `man` and `woman`.

```

<xsd:schema
  targetNamespace="http://example.org/XSD"
  ... >
  ...
  <xsd:element name="Customer"
    sawsdl:modelReference="http://www.example.org/
      ontologies/example#Human"
    sawsdl:liftingSchemaMapping="http://www.example.org/
      ..."
    ...>
  ...
</xsd:element>
</xsd:schema>

```

Listing 12: XSD Human

The expression language of the transition conditions in the links "receive-to-man" and "receive-to-woman" are set to WSML4BPEL which in turn uses WSML [WSML] to evaluate logical expressions.

```

<process ...
  xmlns:ns="http://example.org/XSD">
  ...
  <scope>
    <variables>
      <variable name="customer" element="ns:Customer" />
      ...
    </variables>
    ...
    <flow>
      <links>
        <link name="receive-to-man" />
        <link name="receive-to-woman" />
      </links>
      ...
      <extensionActivity>
        <b4s:interactionActivity
          name="receiveCustomerDetails"
          conversation="shipping"
          outputVariable="customer">

```

```

        <sources>
            <source linkName="receive-to-man" >
                <transitionCondition
                    expressionLanguage="http://www.iaas.uni-
                        stuttgart.de/bpel4sws/
                            wsm14bpel ">
                        customer; ?x memberOf Man
                    </transitionCondition>
                </source>
            <source linkName="receive-to-woman" >
                <transitionCondition
                    expressionLanguage="http://www.iaas.uni-
                        stuttgart.de/bpel4sws/
                            /wsm14bpel ">
                        customer; ?x memberOf Woman
                    </transitionCondition>
                </source>
            </sources>
        </b4s:interactionActivity>
    </extensionActivity>
    ...
</flow>
</scope>
</process>

```

Listing 13: Ontological Reasoning in BPEL4SWS

WSML4BPEL is defined as follows:

```

wsm14bpel = varList ';' http://www.wsm1.org/wsm1/wsm1-syntax#log_expr
varList = var ' ' varList | var

```

WSML4BPEL is an expression language that uses WSML logical expressions to reason over a knowledge base. The knowledge base consists of:

- ontological instances that represent the values of the variables that are specified as input of the query by listing them in the first part of the expression (varList). The instances are generated using the LiftingSchemaMapping defined for the variable types.
- ontologies that define the concepts that are used to annotate the variables that are specified as input of the query .

All variables that are specified as input of the query MUST be visible in the surrounding scope. They MUST be annotated with a modelReference and a liftingSchemaMapping.

In case the query results in a empty set the condition is considered false. In case the query results in a non-empty set, the condition is considered true.

The expressions evaluate to true in case the variable represents a man (link `receive-to-man`) or a woman (link `receive-to-woman`).

5 WSDL-less interaction model

In BPEL the interaction with partner services is based on WSDL. A WSDL extension called partner link type defines an abstract channel between two partners by binding two roles together. The roles are defined in terms of port types each role has to implement. In order to establish a contract between two partners in BPEL, a partner link references a partner link type, and defines which role is taken by the partner service and which role is taken by the process itself. Thus, the WSDL definition of partner services is an integral part of the process definition; BPEL4SWS improves this by allowing to specify the process model independent of WSDL.

BPEL4SWS abstracts from interface definitions, i.e. port Types, and provides for a "WSDL-less interaction model". It is based on the newly introduced concept of a conversation. A conversation is a WSDL-less abstraction of a partner link: Instead of referring to a partner link type, a conversation refers to either a goal or a Web service description defined according to the RO4SSOA.

BPEL4SWS defines new activity types to define the message exchange with partners: the `<b4s:interactionActivity>` that represent WSDL-less receive, reply and invoke activities; a WSDL-less `<b4s:pick>` activity; and a WSDL-less `<b4s:eventHandler>`. The activities can be grouped using a `<b4s:conversation>` element. This way a BPEL4SWS process can be involved in long-running multi-message interactions with partners. Conversations can be grouped using a `<b4s:partner>` element. This way it can be defined that multiple conversations have to be conducted with a single partner.

5.1 Conversation

A `<b4s:conversation>` specifies a message exchange between two partners. It is defined within a `<b4s:conversations>` element which is a child element of `<process>`.

5.1.1 Syntax

Definition of conversation:

```
<b4s:conversations>?  
  <b4s:conversation name="NCName"  
                    hasGoal="anyURI" ?>+  
</b4s:conversations>
```

Listing 14: Conversation Syntax

5.1.2 Properties

The `<b4s:conversation>` element has the following attribute:

- `name`: This attribute is used to unambiguously identify a conversation in a BPEL4SWS process. This attribute is mandatory.
- `hasGoal`: This attribute references a goal description that describes which requirements a BPEL4SWS process imposes on a service that is invoked via this conversation. This attribute is optional.

5.2 InteractionActivity

An `<b4s:interactionActivity>` specifies that a process sends and/or receives (a) message(s). It is defined as a child element of BPELs `<extensionActivity>`.

5.2.1 Syntax

Definition of `b4s:interactionActivity`:

```
<bpel:extensionActivity>
  <b4s:interactionActivity name="NCName"
    inputVariable="NCName"?
    outputVariable="NCName"?
    conversation="NCName"
    createInstance="yes|no"?
    standard-attributes>
    standard-elements
  <bpel:correlations .../>?
</b4s:interactionActivity>
</bpel:extensionActivity>
```

Listing 15: WSDL-less interactionActivity Syntax

5.2.2 Properties

The `<b4s:interactionActivity>` element has the following attributes and elements:

- `name`: This attribute is used to unambiguously identify an `interactionActivity` in a BPEL4SWS process. This attribute is mandatory.
- `inputVariable`: This attribute refers to a process variable which is used as input of this activity. The process variable **MUST** be of type `xsd:element`, `xsd:type` or `wsdl:messageType`. This attribute is optional. This attribute **MUST NOT** be used if the activity receives a message only.
- `outputVariable`: This attribute refers to a process variable which is used as output of this activity. The process variable **MUST** be of type `xsd:element`, `xsd:type` or `wsdl:messageType`. This attribute is optional. This attribute **MUST NOT** be used if the activity sends a message only.
- `conversation`: This attribute indicates in which conversation the activity is participating. This attribute is mandatory.
- `createInstance`: This attribute indicates whether this activity creates a new process instance. This attribute is optional. The default value is `no`. This attribute **MUST NOT** be used if the `outputVariable` is not specified or if the attribute mode has the value `out-in`.
- `standard-attributes`: The activity makes available all BPEL's standard attributes.
- `standard-elements`: The activity makes available all BPEL's standard elements
- `correlations`: This element is used to define correlation. This attribute is optional. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 9.2.

5.2.3 Operational behaviour

In case the `inputVariable` is not specified, the activity behaves like a WS-BPEL `<receive>` activity. It waits for a matching message to arrive and completes when the message arrives.

If only the `inputVariable` is specified, the activity sends the message to a partner service that is associated with the conversation, the activity references. This behaviour corresponds to a `<reply>` or *one-way* `<invoke>` activity of WS-BPEL 2.0. After sending the message the activity completes.

In case both variables are specified, the activity first sends a message and then waits for a matching incoming message which corresponds to a *request-response* `<invoke>` activity in WS-BPEL 2.0. After receiving the message, the activity completes.

5.3 Pick

A `<b4s:pick>` specifies that a process receives (a) message(s). It is defined as a child element of BPELs `<extensionActivity>`.

5.3.1 Syntax

Definition of `pick`:

```
<bpel:extensionActivity>
  <b4s:pick name="NCName"
    createInstance="yes|no"?
    standard-attributes>
  standard-elements
  ( <b4s:onMessage name="NCName"
    variable="NCName"?
    conversation="NCName">
    <bpel:correlations .../>?
    activity
  </b4s:onMessage> |
  <bpel:onMessage partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    variable="BPELVariableName"?
    messageExchange="NCName"?>*
    <bpel:correlations .../>?
    <bpel:fromParts .../>?
    activity
  </onMessage> )+
  <bpel:onAlarm .../>*
</b4s:pick>
</bpel:extensionActivity>
```

Listing 16: WSDL-less pick Syntax

5.3.2 Properties

The `<b4s:pick>` element is enclosed in the BPEL `extensionActivity` and has the following attributes and elements:

- `name`: This attribute is used to unambiguously identify a pick activity in a BPEL4SWS process. This attribute is mandatory.

- `createInstance`: This attribute indicates whether this activity creates a new process instance. This attribute is optional. The default value is `no`.
- `standard-attributes`: The activity makes available all BPEL's standard attributes.
- `standard-elements`: The activity makes available all BPEL's standard elements
- `b4s:onMessage`: This element represents an incoming message that triggers a child activity. For each `<b4s:pick>` activity, at least one `b4s:onMessage` or one `bpel:onMessage` element has to be specified. The `b4s:onMessage` element has the following attributes and elements.
 - `name`: This attribute is used to unambiguously identify an `onMessage` element in a BPEL4SWS process. This attribute is mandatory.
 - `variable`: This attribute refers to a process variable which is used as output of this activity. The process variable **MUST** be of type `xsd:element` or `xsd:type`. This attribute is mandatory.
 - `bpel:correlations`: This element is used to define correlation. This attribute is optional. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 9.2.
 - `bpel:activity`: This element defines the child activity that is triggered by the incoming message. This element is mandatory.
- `bpel:onMessage`: This element represents an incoming message that triggers a child activity. For each `<b4s:pick>` activity, at least one `b4s:onMessage` or one `bpel:onMessage` element has to be specified. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 11.5.
- `bpel:onAlarm`: This element allows defining timing constraints for the scope of the `pick` activity. Any number of `<bpel:onAlarm>` elements may appear as children of the `b4s:pick` activity. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 11.5.

5.3.3 Operational behaviour

The operational behaviour of the `<b4s:pick>` activity is similar to the `<pick>` activity in WS-BPEL 2.0 described in [WS-BPEL 2.0] section 11.5.

It "waits for the occurrence of exactly one event from a set of events, then executes the activity associated with that event. After an event has been selected, the other events are no longer accepted by that `<pick>`. If a race condition occurs between multiple events, the choice of the event is implementation dependent." [WS-BPEL 2.0]

Similar to the WS-BPEL 2.0 `pick`, the `<b4s:pick>` activity's events come in two forms:

- Message events, namely the `<b4s:onMessage>` which is similar to a `<b4s:interactionActivity>` with only the `outputVariable` specified and the `<bpel:onMessage>` is similar to a `<bpel:receive>` activity. Both wait for the receipt of a matching message.
- The `<onAlarm>` corresponds to a timer-based alarm.

Similar to the WS-BPEL 2.0 `<bpel:pick>` which **MUST** include at least one message event, each `<b4s:pick>` activity **MUST** include at least one `<b4s:onMessage>` or `<bpel:onMessage>`.

In case, the `createInstance` attribute is set to "yes", the events in the `<b4s:pick>` MUST all be message events.

5.4 EventHandler

A `<b4s:eventHandler>` specifies that a process receives (a) message(s). It is defined as a child element of `<b4s:eventHandlers>` which can occur as a child element of `<process>` or `<scope>`.

5.4.1 Syntax

Definition of `eventHandler`:

```
<b4s:eventHandlers>?
  <b4s:eventHandler name="NCName">+
    <b4s:onEvent name="NCName"
      variable="NCName"
      conversation="NCName">
      <bpel:correlations .../>?
    scope
  </b4s:onEvent>*
  <bpel:onAlarm .../>*
</b4s:eventHandler>
</b4s:eventHandlers>
```

Listing 17: WSDL-less eventHandler Syntax

5.4.2 Properties

The `<eventHandler>` element has the following attributes and elements:

- `name`: This attribute is used to unambiguously identify an `eventHandler` in a BPEL4SWS process. This attribute is mandatory.
- `onEvent`: This element represents an incoming message that triggers a child activity. Any number of `<onEvent>` elements may appear as children of the `eventHandler` element. The `onEvent` element has the following attributes and elements:
 - `name`: This attribute is used to unambiguously identify an `onEvent` element in a BPEL4SWS process. This attribute is mandatory.
 - `variable`: This attribute refers to a process variable which is used as output of this activity. The process variable MUST be of type `xsd:element` or `xsd:type`. This attribute is mandatory.
 - `conversation`: This attribute indicates in which conversation the activity is participating. This attribute is mandatory.
 - `correlations`: This element is used to define correlation. This attribute is optional. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 9.2.
 - `scope`: This element defines the child scope that is triggered by the incoming message.
- `onAlarm`: This element allows defining timing constraints for the scope of the `pick` activity. Any number of `<onAlarm>` elements may appear as children of the `eventHandler` element. Its syntax and semantics are introduced in the WS-BPEL 2.0 specification, section 12.7.2.

5.4.3 Operational behaviour

Similar to event handlers in WS-BPEL 2.0, `<b4s:eventHandler>`s can be defined for each scope, including the process scope. The behaviour is described in the WS-BPEL 2.0 specification in section 12.7.

Similar to the ws-BPEL 2.0 event handler, `<b4s:eventHandler>`s MUST contain at least one `<onAlarm>` or `<b4s:onEvent>` element.

The `<onEvent>` element indicates that the specified event waits for a message to arrive. The interpretation of this element and its attributes is very similar to a `<interactionActivity>` with only the `outputVariable` specified.

The `<onAlarm>` element marks a time-driven event like described in the WS-BPEL 2.0 specification section 12.7.2.

The enablement and processing of events is defined like in section 12.7.3 and 12.7.4 of the WS-BPEL 2.0 specification.

5.5 Partner

A `<b4s:partner>` is used to specify that multiple `<b4s:conversation>`s have to take place with one single partner. It is defined as a child element of `<b4s:partners>` which can occur as a child element of `<process>` or `<scope>`.

5.5.1 Syntax

Definition of `partner`:

```
<b4s:partners>?  
  <b4s:partner name="NCName"  
              businessEntity="QName" ?>+  
    <b4s:conversation name="NCName" />+  
  </b4s:partner>  
</b4s:partners>
```

Listing 18: partner Syntax

5.5.2 Properties

The `<partner>` element is enclosed in a `<partners>` element and has the following attributes and elements:

- `name`: This attribute is used to unambiguously identify a partner element in a BPEL4SWS process. This attribute is mandatory.
- `businessEntity`: This attribute is used to specify a `businessEntity`, i.e. a concrete organisation or unit. This attribute is optional.
- `conversation`: This element represents a conversation that has to be established with the same partner then all other conversations enclosed in the partner element. For each `<b4s:partner>` element, at least one `conversation` element has to be specified. The `conversation` element has the following attributes and elements:
 - `name`: This attribute is used to identify the conversation that has to be established with the partner the conversation is enclosed with. This attribute is mandatory.

5.5.3 Operational behaviour

Like the partner link in WS-BPEL, a conversation represents a conversational relationship between two partner processes. The `<b4s:partner>` element is used to express that more than a single conversational relationship has to be established with a business partner just like the partner element in [BPEL4WS 1.1]. The operational behaviour is defined in section 7.3 of the BPEL4WS 1.1 specification.

6 Describing Interactions using the RO4SSOA

The WSDL-less interaction model presented in the previous section builds the core of BPEL4SWS. BPEL4SWS uses the concept of a conversation to model the interaction of a process with a partner service. This conversation is concerned with solving a certain kind of problem, e.g. booking a flight. During this kind of conversation there are two roles: One partner provides the functionality to book a flight which is described via the concept of a service according to the RO4SSOA and the other partner wants to use/consume this functionality which is described via the concept of a goal. Therefore BPEL4SWS distinguishes between *providing* and *consuming* conversations. On a providing conversation the process offers functionality to partner services and on a consuming conversation a BPEL4SWS process wants to use functionality provided by a partner service.

Providing conversations are described using a service description. Consuming conversations are annotated with goal descriptions. This annotation can be made using the `hasGoal` attribute of the `<conversation>`-element.

When the process engine executes an activity that refers to a consuming conversation that has not been initialized yet, the attached goal description is used to discover a corresponding Web Service. In case no Web Service has been discovered, a `noServiceFound` Fault has to be thrown.

7 Grounding

The RO4SSOA builds a layer on top of Web services. It abstracts from technical details and provides a description of service requester and service provider that not only focusses on the interface but also describes the capability of a service in terms of assumptions and effects. To facilitate communication among services both, service as well as goal descriptions ground to WSDL. Frameworks that implement the RO4SSOA require a middleware component that provides the following functionalities:

- Goal-based service discovery
- Management of the life cycle of a communication between service requester and service provider
- Endpoint(s) for receiving data from both, service requester and service provider
- Means to invoke both, service provider and service requester

Conversations in BPEL4SWS are described using concepts of the RO4SSOA, but the processes endpoint is also described in terms of WSDL port types. The link between the WSDL-less interaction model and the WSDL description of a process is established via a separate grounding file.

7.1 Syntax

```
<grounding processName="QName"
  xmlns="http://www.iaas.uni-stuttgart.de/bpel4sws/
  wsdlGrounding">
  <conversation name="NCName"
    partnerLinkType="QName"?
    myRole="NCName"?
    partnerRole="NCName"?>
    <activity name="NCName"
      portType="QName"?
      operation="NCName"/>+
  </conversation>+
</grounding>
```

Listing 19: Process Grounding

7.2 Properties

The `<grounding>` document has the following attributes and elements:

- `processName`: This attribute is used to reference the process the grounding is specified for. This attribute is mandatory
- `xmlns`: This element defines the namespace the document is defined in. This attribute is mandatory
- `conversation`: This element reference a conversation in the given process the grounding is defined for. For each `<grounding>` element, at least one `conversation` element has to be specified. The `conversation` element has the following attributes and elements:

- `name`: This attribute is used to identify the conversation. This attribute is mandatory.
- `partnerLinkType`: This attribute references the `partnerLinkType` the conversation is grounded to. This attribute is optional.
- `myRole`: This attribute references the role the process takes. This attribute is optional. In case the `partnerLinkType` attribute is specified and the `partnerRole` attribute is not specified, this attribute must be specified.
- `partnerRole`: This attribute references the role the partner service takes. This attribute is optional. In case the `partnerLinkType` attribute is specified and the `myRole` attribute is not specified, this attribute must be specified.
- `activity`: This element references the activity or a event that is triggered by an incoming message respectively a grounding is defined for. For each `<conversation>` element, at least one `activity` element has to be specified. The `activity` element has the following attributes and elements:
 - `name`: This attribute is used to identify the activity the grounding is defined for. This attribute is mandatory.
 - `portType`: This attribute references the `portType` the activity is grounded to. This attribute is optional. In case the `partnerLinkType` attribute of the surrounding conversation is not specified this attribute must be specified.
 - `operation`: This attribute references the operation the activity is grounded to. This attribute is mandatory.

7.3 Operational behaviour

The process grounding binds WSDL operations the process provides to the WSDL-less interaction model.

At the beginning of a consuming conversation, i.e. the first occurrence of an interaction activity belonging to a conversation that references a goal description, the goal is submitted to the middleware component that executes goal-based discovery and creates an instance identifier of the conversation. This instance identifier **MUST** be included during the conversation between both, service consumer and middleware as well as middleware and service provider.

In case a message is received at the WSDL endpoint that is provided for a particular process, the grounding file is evaluated to determine which activity in the process model is the recipient of the message. The data types of the variables used as input/output container of an activity **MUST** match the data types used in the WSDL operations an activity is grounded to. In case the receiving activity belongs to an uninitialized providing conversation, the message has to be inspected for the instance identifier created by the middleware component that manages the life cycle of the communication. In case no instance identifier is available the process has been invoked directly by a partner service. Then, the EPR of the `partnerRole` of the `partnerLinkType` the conversation is grounded to has to be initialized. EPRs are further describe in the WS-BPEL 2.0 specification, section 6.3.

When sending a message, first the process grounding is determined whether the message belongs to a request-response operation, the processes endpoint provides.

If this is not the case and an instance identifier for the communication is available, the message in combination with the instance identifier is delivered to the middleware component which executes the invocation of the corresponding service on behalf of the process. In case no instance identifier is available and the activity belongs to a providing conversation the process grounding is evaluated to determine the WSDL operation of the partner service the message has to be sent to.

8 Acknowledgements

The work published on this specification was partly funded by the SUPER project (<http://www.ip-super.org/>) under the EU 6th Framework Programme Information Society Technologies Objective (contract no. FP6-026850).

The authors would like to thank the whole SUPER consortium and in particular Agata Filipowska, Armin Haller, Monika Kaczmarek and Barry Norton for their valuable contributions.

9 References

[BPEL4WS 1.1]

Business Process Execution Language for Web Services Version 1.1, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, May 2003, available via <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>, <http://ifr.sap.com/bpel4ws/>

[RFC 2119]

Key words for use in RFCs to Indicate Requirement Levels, [RFC 2119](#), available via <http://www.ietf.org/rfc/rfc2119.txt>

[WS-BPEL 2.0]

Web Service Business Process Execution Language Version 2.0, Working Draft, January 2006, OASIS Technical Committee, available via <http://www.oasis-open.org/committees/wsbpel>

[WSDL 1.1]

Web Services Description Language (WSDL) Version 1.1, W3C Note, available via <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>

[XML Namespaces]

Namespaces in XML 1.0 (Second Edition), W3C Recommendation, available via <http://www.w3.org/TR/REC-xml-names/>

[XML Schema Part 1]

XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via <http://www.w3.org/TR/xmlschema-1/>

[XML Schema Part 2]

XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via <http://www.w3.org/TR/xmlschema-2/>

[SAWSDL]

Semantic Annotations for WSDL and XML Schema, W3C Recommendation August 2007, available via <http://www.w3.org/TR/sawSDL/>

[WS-I Basic Profile]

Web Services Interoperability Organization, "Basic Profile Version 1.1", K. Ballinger, D. Ehnebuske, M. Gudgin, M. Nottingham, P. Yendluri, April 16, 2004, available via <http://www.ws-i.org/Profiles/BasicProfile-1.1.html>

[SOARM]

Reference Model for Service Oriented Architecture 1.0, Oasis Committee Specification, August 2006, available via <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>

10 Non-normative references

[RO4SSOA]

Reference Ontology for Semantic Service Oriented Architectures, Specification Draft, OASIS Semantic Execution Environment TC, March 2008, available via http://www.oasis-open.org/committees/document.php?document_id=25381&wg_abbrev=semantic-ex

[WSMO]

Web Service Modeling Ontology (WSMO), W3C Member Submission, June 2005, available via <http://www.w3.org/Submission/WSMO/>

[WSML]

Web Service Modeling Language (WSML) 0.3, WSML WG Draft, available via <http://www.wsmo.org/TR/d16/d16.1/v0.3/>

[OWL-S]

OWL-S: Semantic Markup for Web Services, W3C Member Submission, November 2004, available via <http://www.w3.org/Submission/OWL-S/>

Appendix A – Standard Faults

The following list specifies the standard faults defined within the BPEL4SWS specification. All standard fault names are qualified with the standard BPEL4SWS namespace.

Fault name	Description
noServiceFound	Thrown if no service is found
liloFault	Thrown if an error occurred during the lifting or lowering of data
mediationFault	Thrown if an error occurred during data mediation

Appendix B – BPEL4SWS Schema

XML schema for WS-BPEL Extension for Semantic Web Services (BPEL4SWS):

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  targetNamespace="http://www.iaas.uni-stuttgart.de/bpel4sws/
    executable"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.iaas.uni-stuttgart.de/bpel4sws/executable"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
    executable">

  <xsd:import
    namespace="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
    schemaLocation="http://docs.oasis-
open.org/wsbpel/2.0/OS/process/
      executable/ws-bpel_executable.xsd">

  </xsd:import>
  <xsd:annotation>
    <xsd:documentation>
      Schema for BPEL4SWS; Last modified date: 12th July, 2007
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="mediate"
    type="tMediate" />
  <xsd:complexType name="tMediate">
    <xsd:complexContent>
      <xsd:extension base="bpel:tExtensibleElements">
        <xsd:attribute name="name"
          type="xsd:NCName"
          use="required" />
        <xsd:attribute name="inputVariable"
          type="xsd:NCName"
          use="required" />
        <xsd:attribute name="outputVariable"
          type="xsd:NCName"
          use="required" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="conversations"
    type="tConversations" />
  <xsd:complexType name="tConversations">
    <xsd:complexContent>
      <xsd:extension base="bpel:tExtensibleElements">
        <xsd:sequence>
          <xsd:element ref="conversation"
            minOccurs="1"
            maxOccurs="unbounded" />
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <xsd:element name="conversation"
    type="tConversation" />
```

```

<xsd:complexType name="tConversation">
  <xsd:complexContent>
    <xsd:extension base="bpel:tExtensibleElements">
      <xsd:attribute name="name"
        type="xsd:NCName"
        use="required" />
      <xsd:attribute name="hasGoal"
        type="xsd:anyURI"
        use="required" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="tActivity">
  <xsd:complexContent>
    <xsd:restriction base="BPEL:tActivity">
      <xsd:attribute name="name"
        type="xsd:NCName"
        use="required" />
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="interactionActivity"
  type="tInteractionActivity" />
<xsd:complexType name="tInteractionActivity">
  <xsd:complexContent>
    <xsd:extension base="tActivity">
      <xsd:sequence>
        <xsd:element name="correlations"
          type="BPEL:tCorrelationsWithPattern"
          minOccurs="0" />
      </xsd:sequence>
      <xsd:attribute name="inputVariable"
        type="xsd:NCName" use="optional" />
      <xsd:attribute name="outputVariable"
        type="xsd:NCName"
        use="optional" />
      <xsd:attribute name="conversation"
        type="xsd:NCName"
        use="required" />
      <xsd:attribute name="createInstance"
        type="BPEL:tBoolean"
        use="optional" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="partners"
  type="tPartners" />
<xsd:complexType name="tPartners">
  <xsd:complexContent>
    <xsd:extension base="bpel:tExtensibleElements">
      <xsd:sequence>
        <xsd:element ref="partner"
          minOccurs="1"
          maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>

```

```

</xsd:complexType>

<xsd:element name="partner"
             type="tPartner" />
<xsd:complexType name="tPartner">
  <xsd:complexContent>
    <xsd:extension base="bpel:tExtensibleElements">
      <xsd:sequence>
        <xsd:element name="conversation"
                     type="xsd:NCName"
                     maxOccurs="unbounded"
                     minOccurs="1"/>
      </xsd:sequence>
      <xsd:attribute name="name"
                    type="xsd:NCName"
                    use="required" />
      <xsd:attribute name="businessEntity"
                    type="xsd:anyURI"
                    use="optional" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="pick"
             type="tPick" />
<xsd:complexType name="tPick">
  <xsd:complexContent>
    <xsd:extension base="bpel:tActivity">
      <xsd:sequence>
        <xsd:element ref="onMessage"
                     minOccurs="1"
                     maxOccurs="unbounded" />
        <xsd:element name="onAlarm"
                     type="bpel:tOnAlarmPick"
                     minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:attribute name="createInstance"
                    type="bpel:tBoolean"
                    default="no" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:element name="onMessage"
             type="tOnMessage" />
<xsd:complexType name="tOnMessage">
  <xsd:complexContent>
    <xsd:extension base="bpel:tExtensibleElements">
      <xsd:sequence>
        <xsd:element name="correlations"
                     type="bpel:tCorrelations"
                     minOccurs="0" />
        <xsd:element ref="bpel:fromParts"
                     minOccurs="0" />
        <xsd:group ref="bpel:activity"
                  minOccurs="1" />
      </xsd:sequence>
      <xsd:attribute name="name"
                    type="xsd:NCName"
                    use="required" />
      <xsd:attribute name="variable"
                    type="xsd:NCName"
                    use="optional" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

```

```

        type="bpel:BPMLVariableName"
        use="optional" />
    </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:element name="eventHandlers"
    type="tEventHandlers"/>
<xsd:complexType name="tEventHandlers">
    <xsd:complexContent>
        <xsd:extension base="bpel:tExtensibleElements">
            <xsd:sequence>
                <xsd:element ref="onEvent"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
                <xsd:element name="onAlarm"
                    type="bpel:tOnAlarmEvent"
                    minOccurs="0"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="onEvent"
    type="tOnEvent"/>
<xsd:complexType name="tOnEvent">
    <xsd:complexContent>
        <xsd:extension base="bpel:tExtensibleElements">
            <xsd:sequence>
                <xsd:element name="correlations"
                    type="bpel:tCorrelations"
                    minOccurs="0"/>
                <xsd:element ref="bpel:fromParts"
                    minOccurs="0"/>
                <xsd:element ref="bpel:scope"
                    minOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="name"
                type="xsd:NCName"
                use="required" />
            <xsd:attribute name="variable"
                type="bpel:BPMLVariableName"
                use="optional"/>
            <xsd:attribute name="messageType"
                type="xsd:QName"
                use="optional"/>
            <xsd:attribute name="element"
                type="xsd:QName"
                use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
</xsd:schema>

```

XML Schema for the grounding specification for BPEL4SWS:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
  targetNamespace="http://www.iaas.uni-stuttgart.de/bpel4sws/
    grounding"
  elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:prg="http://www.iaas.uni-stuttgart.de/bpel4sws/grounding">

  <xsd:complexType name="tGrounding">
    <xsd:sequence>
      <xsd:element name="conversation"
        type="prg:tConversation"
        minOccurs="1"
        maxOccurs="unbounded" />
    </xsd:sequence>

    <xsd:attribute name="processName"
      type="QName"
      use="required" />
  </xsd:complexType>

  <xsd:complexType name="tConversation">
    <xsd:sequence>
      <xsd:element name="activity"
        type="prg:tActivity"
        minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name"
      type="NCName"
      use="required" />
    <xsd:attribute name="partnerLinkType"
      type="QName"
      use="optional" />
    <xsd:attribute name="myRole"
      type="NCName"
      use="optional" />
    <xsd:attribute name="partnerRole"
      type="NCName"
      use="optional" />
  </xsd:complexType>

  <xsd:complexType name="tActivity">
    <xsd:attribute name="name"
      type="NCName"
      use="required" />
    <xsd:attribute name="operation"
      type="QName"
      use="required" />
  </xsd:complexType>

  <xsd:element name="grounding"
    type="prg:tGrounding" />
</xsd:schema>
```