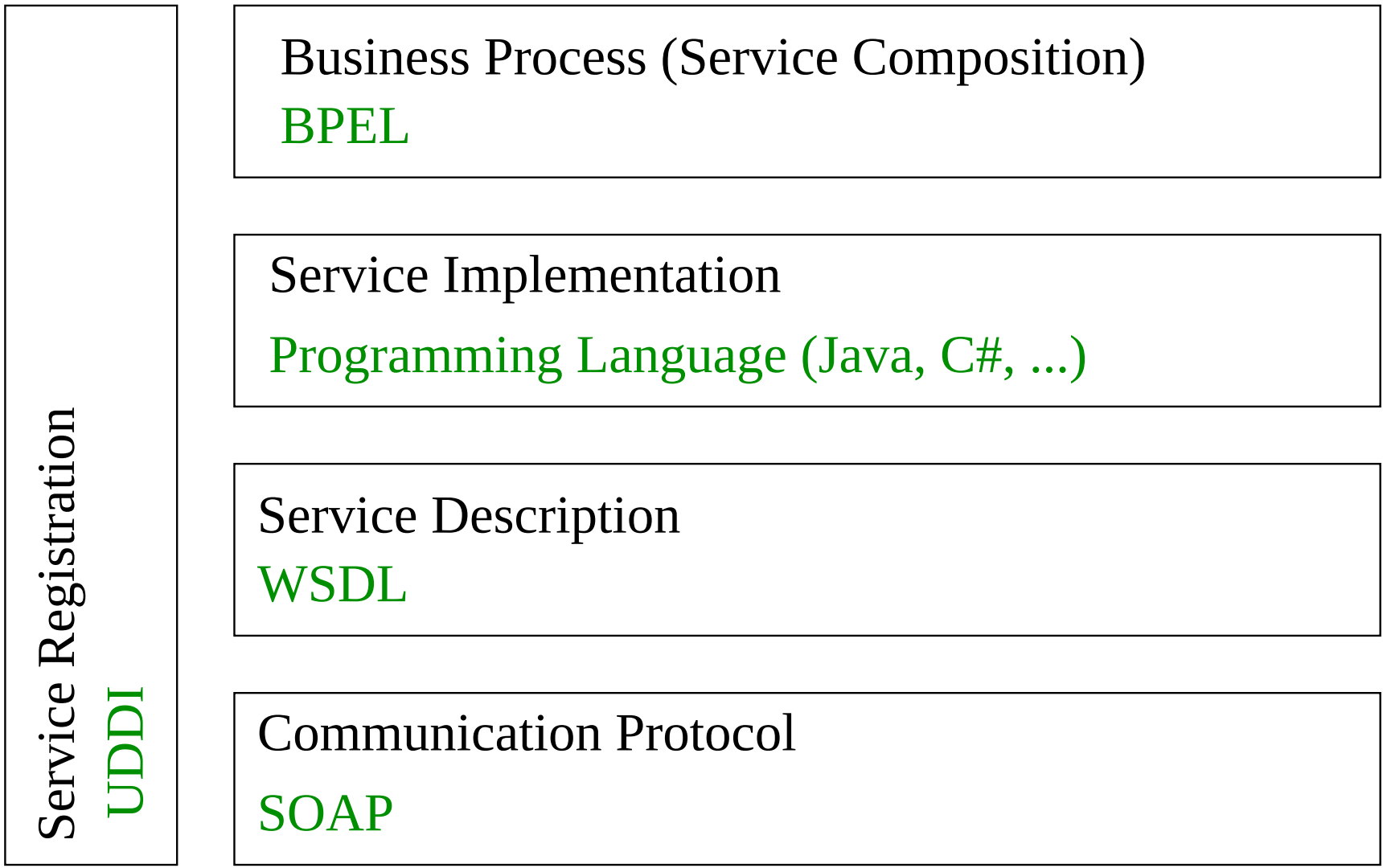
Part VII

Business Process Execution Language

Motivation

- Observation: business processes are frequently changed
- this implies frequent and quick software changes
- Idea:
  - structure software into system of web services ($\rightarrow$ SOA)
  - compose services flexibly
- for this: Business Process Execution Language (BPEL, WS-BPEL, BPEL4WS)

Overview


Service Registration
UDDI
Business Process (Service Composition)
BPEL
Service Implementation
Programming Language (Java, C#, ...)
Service Description
WSDL
Communication Protocol
SOAP

BPEL

- universal imperative programming language ("Turing complete")
- tailored to the composition of web services
  ($\rightarrow$ domain specific language, DSL)
- synchronous and asynchronous web-service calls
- assignments to variables
- block structured
- control structures: sequence, loop, branching, ...

BPEL (continued)

- error and event handling, compensation
- concurrency
- BPEL processes are web services themselves
  ($\rightarrow$ recursion possible but unusual)
- syntax based on XML tags
  ($\rightarrow$ badly readable for humans, graphic tools)
- based on: XML schema, XPath, WSDL, ...

What BPEL Does Not Offer

- user interaction ($\rightarrow$ project BPEL4People)
- procedures, methods
- object orientation (inheritance)

Comparison: BPEL vs. Java as Coordination Language

- What can BPEL do, what Java cannnot? nothing!
- BPEL suited, if coordination logic is simple
- Java more appropriate, if coordination logic is complex

BPEL Engines

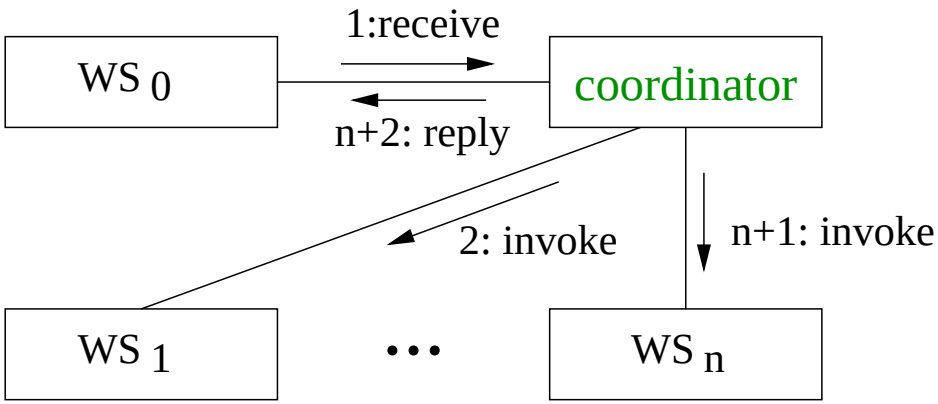
- BPEL programs are executed by a BPEL engine
  (i.e. BPEL runtime environment)
- available BPEL engines:
  - ActiveBPEL Engine (Active Endpoints, open source)
  - Websphere Business Integration Server Foundation (IBM)
  - BPEL Process Manager (Oracle)
  - BizTalk Server (Microsoft)
  - NetWeaver Exchange Infrastructure (SAP)
  - WebLogic Integration / AquaLogic (BEA)
  - ...

Similar Languages and Approaches

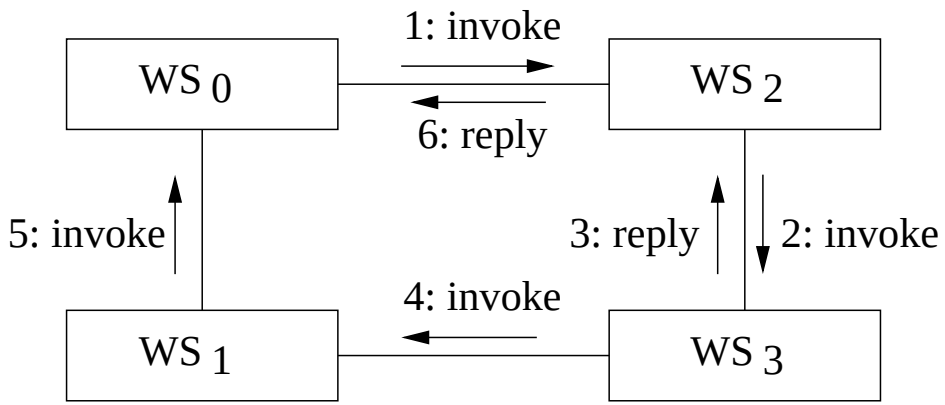
- XLANG/s (Microsoft) incorporated in BPEL
- WSFL (Web Services Flow Language, IBM) incorporated in BPEL
- BPML (Business Process Modeling Language, BPMI.org)
- WSCI (Web Service Choreography Interface, by Sun, SAP, BEA)
- BPSS (Business Process Specification Schema, part of ebXML)
- WS-CDL (Web Service Collaboration Definition Language)
- jBPM (JBoss)
- ...
- BPEL seems to prevail

Orchestration vs. Choreography

Orchestration



Choreography



- BPEL can model both
- a BPEL engine can only execute orchestration

Orchestration

- central coordination by a web service
- execution by BPEL engine
- participating web services don't know,
  how and where they are used
- thus: new web services can be integrated more easily
- typically used in in-house applications
- easy to supervise and monitor
- error handling easy

Choreography

- distributed coordination
- no central execution and monitoring by BPEL engine
- participating web services know,
  how they collaborate with their neighbors
- typically used for cross-company applications (e.g. SCM)
- monitoring and error handling difficult

Executable vs. Abstract Process

- Executable Process:
  - based on orchestration
  - coordinator executed by BPEL engine
  - business process is new web service
- Abstract Process:
  - describes the distributed interaction between web services
  - not executed by BPEL engine
  - typically used for choreography

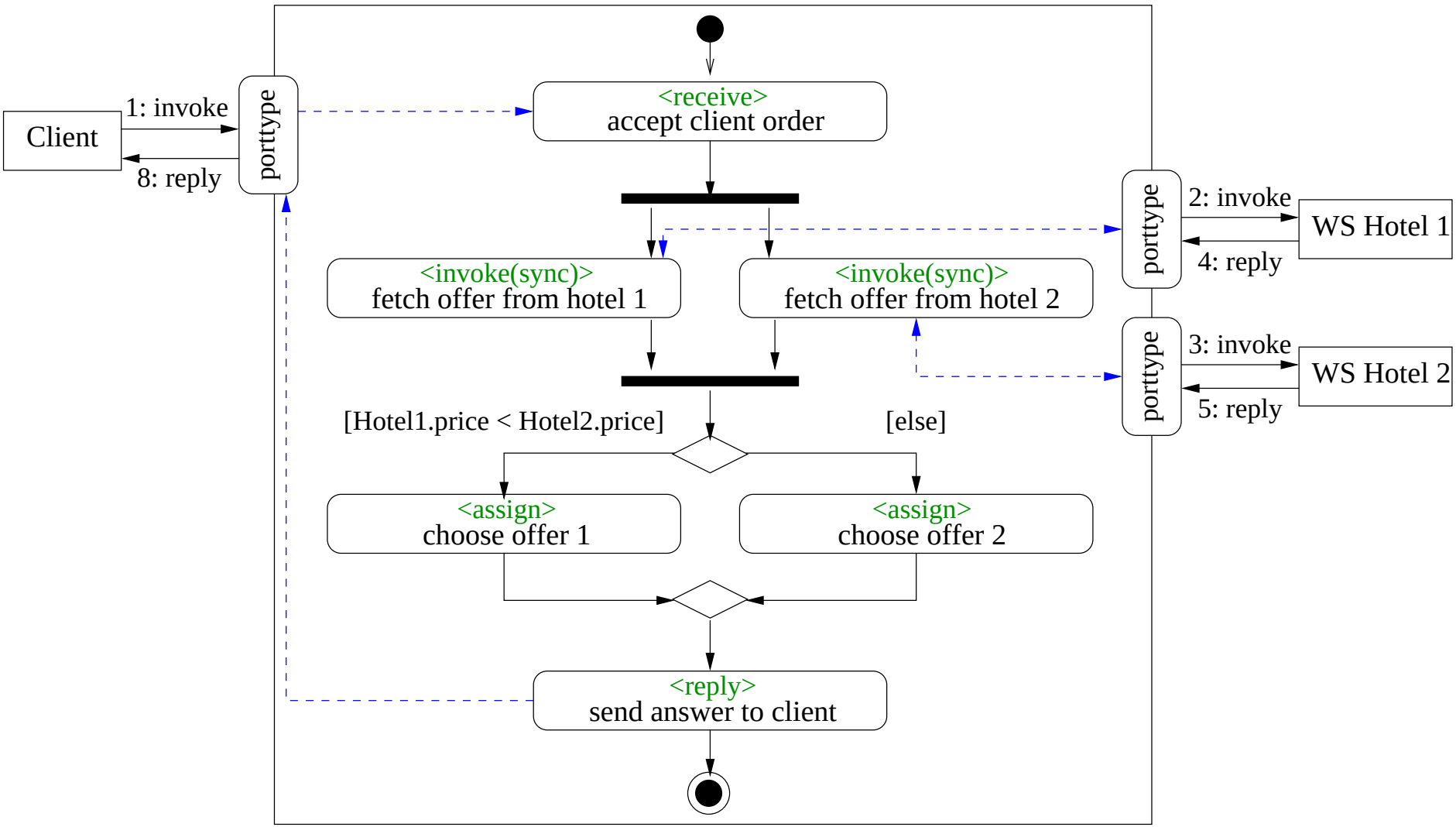
BPEL Features

- Declarations:
  - `<variable>`: variable
  - `<partnerLink>`: link to other web service
- Basic Activities:
  - `<invoke>`: synchronous or asynchronous web service call
  - `<receive>`: reception of message from client or from asynchronously called service
  - `<reply>`: send answer to client
  - `<assign>`: assignment
  - `<throw>`: trigger exception handling
  - `<wait>`: wait for some time (`for`) or until point in time (`until`)
  - `<terminate>`: terminate

BPEL Features (continued)

- Compound Activities:
  - `<sequence>`: sequential composition of activities
  - `<flow>`: parallel composition
  - `<switch>`: branching
  - `<while>`, `<repeatUntil>`, `<forEach>`: usual loops
  - `<pick>`: waits for event
- Structuring Tags:
  - `<process>`: surrounds BPEL program
  - `<variables>`, `<partnerlinks>`, ...: embrace declarations

Example: Hotel Agency Service



Example: Partner Links for Hotel Agency Service

```
<partnerLinks>
  <partnerLink name = "client"
               partnerLinkType = "clt:agencyLT"
               myRole = "hotelAgencyService"/>
  <partnerLink name = "hotel1"
               partnerLinkType = "hotel:hotelLT"
               myRole = "hotelClient"/>
  <partnerLink name = "hotel2"
               partnerLinkType = "hotel:hotelLT"
               myRole = "hotelClient"/>
</partnerLinks>
```

Example: Variables for Hotel Agency Service

```
<variables>
  <variable name = "AgencyQuery"
            messageType = "hotel:HotelQueryMessage"/>
  <variable name = "Hotel1Answer"
            messageType = "hotel:HotelAnswerMessage"/>
  <variable name = "Hotel2Answer"
            messageType = "hotel:HotelAnswerMessage"/>
  <variable name = "AgencyAnswer"
            messageType = "hotel:HotelAnswerMessage"/>
</variables>
```

Example: BPEL Process for Hotel Agency Service

```
<?xml version="1.0" encoding="utf-8"?>
<process name= "HotelAgencyProcess"
        targetNameSpace="http://hotelagency.com/bpel/hotelexample/"
        xmlns="http//schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:clt="http://hotelagency.com/bpel/client/"
        xmlns:hotel="http://hotelagency.com/bpel/hotel/">
  <partnerLinks> ... see above ... </partnerLinks>
  <variables> ... see above ... </variables>

  <sequence>
    <receive partnerLink = "client" ... />
    <flow>
      <invoke partnerLink = "hotel1" ... />
      <invoke partnerLink = "hotel2" ... />
    </flow>
    <switch> ... </switch>
    <reply partnerLink = "client" ... />
  </sequence>
</process>
```

Control Flow in Example in Detail (1)

```
<sequence>
  <receive partnerLink = "client"
           portType = "clt:HotelAgencyServicePT"
           operation = "BrokerHotel"
           variable = "AgencyQuery"
           createInstance = "yes"/>
  <flow>
    <invoke partnerLink = "hotel1"
            portType = "hotel:ComputeHotelPricePT"
            operation = "ComputeHotelPrice"
            inputVariable = "AgencyQuery"
            outputVariable = "Hotel1Answer"/>

    <invoke partnerLink = "hotel2"
            portType = "hotel:ComputeHotelPricePT"
            operation = "ComputeHotelPrice"
            inputVariable = "AgencyQuery"
            outputVariable = "Hotel2Answer"/>
  </flow>   ...
```

Control Flow in Example in Detail (2)

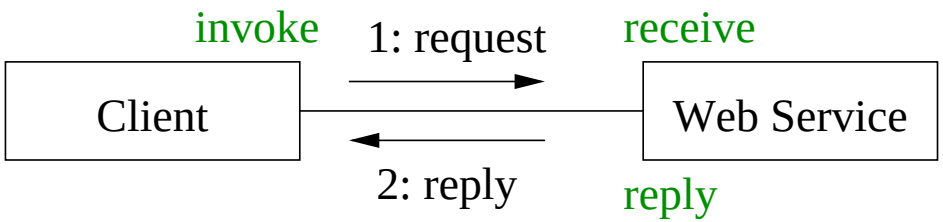
```
<switch>
  <case condition = "bpws:getVariableData('Hotel1Answer',
                'ConfirmationData', '/ConfirmationData/hotel:Amount')
                &lt; bpws:getVariableData('Hotel2Answer',
                'ConfirmationData', '/ConfirmationData/hotel:Amount')">
    <assign>
      <copy>  <from variable = "Hotel1Answer"/>
              <to variable = "AgencyAnswer"/>
      </copy>
    </assign>
  </case>
  <otherwise>
    <assign>
      <copy>  <from variable = "Hotel2Answer"/>
              <to variable = "AgencyAnswer"/>
      </copy>
    </assign>
  </otherwise>
</switch>    ...
```

Control Flow in Example in Detail (3)

```
...
  <reply partnerLink = "client"
         portType = "clt:HotelAgencyPT"
         operation = "ChooseHotel"
         variable = "AgencyAnswer"/>
</sequence>
</process>
```

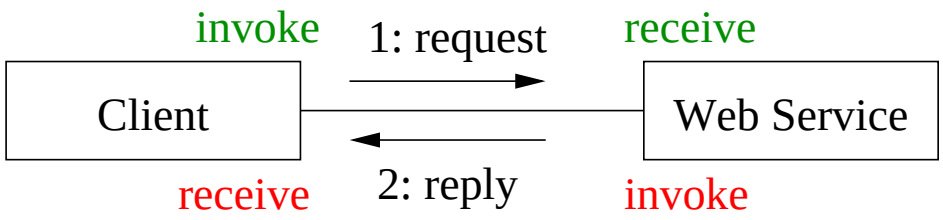
Synchronous vs. Asynchronous Web-Service Calls

synchronous:



- `<invoke>` waits for result
- and accepts it

asynchronous:



- `<invoke>` does not wait
- `<receive>` accepts result later
- precondition: 2 porttypes (one-way) in WSDL

Partner-Link Types

- partner-link types allow to specify the role of a web service in an interaction

Example:

```
<partnerLinkType name = "HotelLT"
            xmlns="http://schemas.xmlsoap.org/ws/2003/05/partner-link"/>
            <role name = "HotelAgencyService">
    <portType name = "hotel:ComputeHotelPricePT"/>
  </role>

  <role name = "HotelClient">
    <portType name = "hotel:ComputeHotelPriceCallBackPT"/>
  </role>
</partnerLinkType>
```

- partner-link types are specified in WSDL rather than in BPEL

Example: Partner Links

```
<partnerLink name = "hotel1"
             partnerLinkType = "hotel:HotelLT"
             myRole = "HotelClient"
             partnerRole = "Hotel"/>
```

- reference to neighbor web service
- attributes:
  - `name`: name of partner link
  - `partnerLinkType`: type of partner link
  - `myRole`: own role in interaction
  - `partnerRole`: role of partner in interaction

Variables

Sorts of Variables:

- `messagetype`: variable stores message
- `element`: ... XML schema element
- `type`: ... value of XML schema basic type (int, double, string, ...)

Example:

```
<variables>
  <variable name = "AgencyQuery"
                messageType = "hotel:HotelQueryMessage"/>
  <variable name = "Address"
                element = "hotel:ClientInformation"/>
  <variable name = "Price"  type = "xs:int"/>
</variables>
```

- variables can have global or local scope ($\rightarrow$ `<scope>`)

Assignments

Example:

```
<assign>
   <copy>
      <from variable = "Hotel1Answer"/>
      <to variable = "AgencyAnswer"/>
   </copy>
</assign>
```

- several copy operations per `<assign>` possible
- also parts of message (see WSDL) and their components can be copied or changed
- a component is selected with (e.g.) XPath
- a `<from>` clause can also contain constants and arithmetic expressions

Example: Assignment

```
<assign>
  <copy>  <from expression="number(42)"/>
          <to variable = "Price"/>
  </copy>
  <copy>  <from expression = "bpws:getVariableData('Price')+1">
          <to variable = "HotelAnswer"
              part="Offer"
              query="/Offer/hotel:Price"/>
  </copy>
</assign>
```

Example: Assignment (continued)

corresponding WSDL:

```
<message name = "HotelAnswer">
  <part name="ClientData" type="hotel:ClientDataType"/>
  <part name="Offer" type="hotel:OfferType"/>
</message>
```

corresponding type declaration:

```
<xs:complexType name="OfferType">
  <xs:sequence>
    <xs:element name="Price" type="xs:int"/>
    <xs:element name="VAT"  type="xs:double"/>
  </xs:sequence>
</xs:complexType>
```

Condition

- used for branching and loop control
- arbitrary boolean XPath expression is allowed
- it may contain:
  - variables, constants
  - comparison operations: `&eq; &ne; > &ge; < &le;`
  - boolean operations: `&and; &or;`
- extension: `getVariableData` function for accessing values of variables

  ```
  bpws:getVariableData('VariableName', 'PartName', 'Path')
  ```

- the last two arguments are optional

Example: Condition

```
<switch>
  <case condition =
    "bpws:getVariableData(
                  'Hotel1Answer',
                  'ConfirmationData',
                  '/ConfirmationData/hotel:Amount')
    &lt; 100">
...
```

Activities <invoke>, <receive>, and <reply>

Attributes:

- partnerLink, portType, operation (from WSDL document)
- for <invoke> also: inputVariable, outputVariable
- for <receive> additionally:
  - variable: stores received message
  - createInstance (y/n):
    - determines whether a new instance of the web service shall be created when receiving a message
    - for initial <receive> from client: yes
    - otherwise (callback): no
- for <reply> additionally: variable with answer

Error Handling

Triggering Error Handling

- with `<throw>`
- results in fault answer, if not caught

Catching Errors

- `<catch>` allows to handle a specified error
- `<catchAll>` catches an arbitrary error

Example: Error Handling

```
<scope name="HotelCall">
  <faultHandlers>
    <catchAll>
      <!- if hotel not available: Price = 999999 ->
      <assign>
        <copy> <from expression="number(999999)"/>
               <to variable = "Hotel1Answer"
                   part="Offer"   query="/Offer/hotel:Price"/>
        </copy>
      </assign>
    </catchAll>
  </faultHandlers>

  <invoke partnerLink = "hotel1"
          portType = "hotel:ComputeHotelPricePT"
          operation = "ComputeHotelPrice"
          inputVariable = "AgencyQuery"
          outputVariable = "Hotel1Answer"/>
</scope>
```

Event Handling with <pick>

- waits for one of several possible events (including message reception)

```
<pick>
   <onMessage partnerLink="myPartner"
              portType="myServicePT"
              operation="myService"
              variable="Result">
      <!- process message ->
   </onMessage>

   <onAlarm ...>
      <!- process alarm due to timeout ->
   </onAlarm>
</pick>
```