Service-Orientierten Architecture and Web-Services

Introduction and Basic Concepts

Material created by Claus P. Priese

Web-Services and Service-Orientierted Architecture (WS & SOA)

Web Services are developed since 1999 by several working groups of the World Wide Web Consortium

Most important basic specification documents are:

- [W3C04g] World Wide Web Consortium: Web Services Architecture Requirements, W3C Working Draft, 11. Februar 2004, http://www.w3.org/TR/wss-reqs/
- [W3C04h] World Wide Web Consortium: Web Services Architecture (WSA), W3C Working Draft, 11. Februar 2004, http://www.w3.org/TR/ws-arch
- [W3C04i] World Wide Web Consortium: Web Services Glossary, W3C Working Draft, 11. Februar 2004, http://www.w3.org/TR/ws-gloss/

WS & SOA – terms & concepts

What is a Web Service ?

The Web Services Glossary document [W3C04i], created by the web services architecture working group, says:

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

WS & SOA – terms & concepts

Web Service: is an abstract decription functionality and interfaces

Agent: is a concrete realization of a Web Service, written in any programming language

Provider: is the person or organisation that provides a concrete agent to implement a Web Service

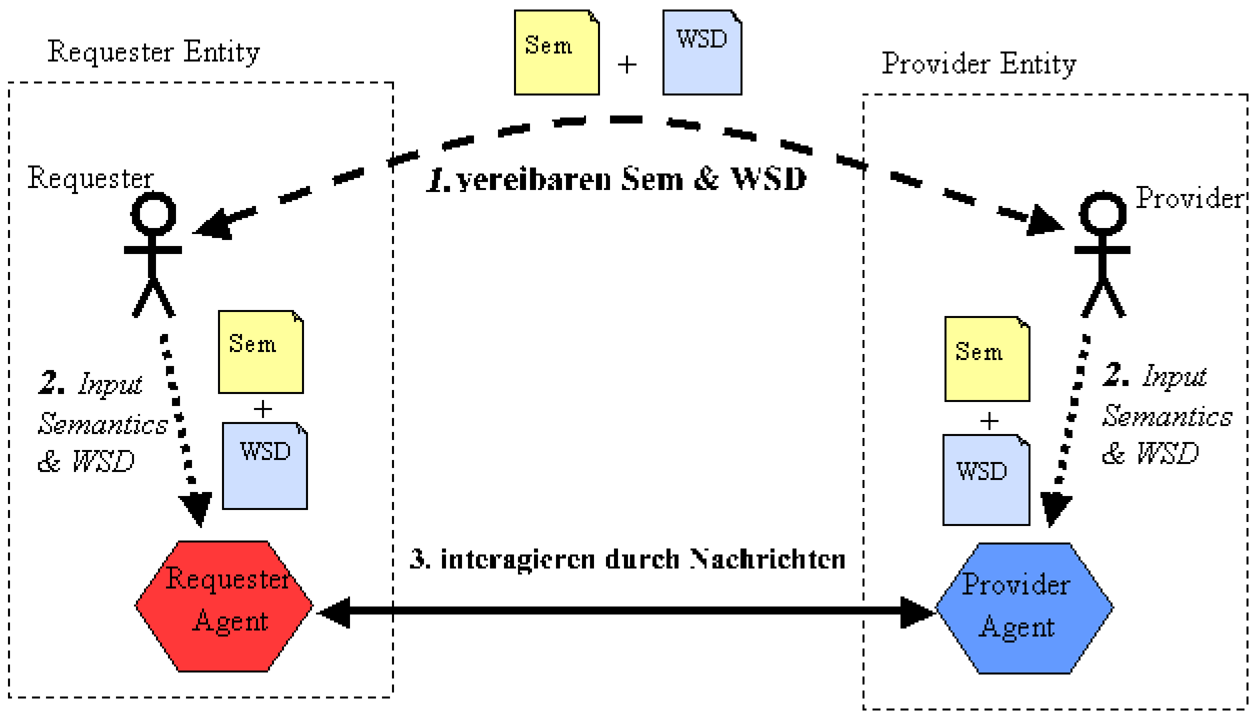
Requester: is the person or organisation that wishes to use a providers agent. For the exchange of messages a requester agent will be used

WS & SOA – terms & concepts

Service Description (WSD): is a machine readable specification of web service interfaces in WSDL with accompanying information about datatypes, protocols, concrete endpoints and msg.X.patterns

Semantics: is the conctract between reqester und provider about the meaning and purpose of agent-interaction not already covered in the WSD. The way of expressing semantics - oral, informal or strict formal - is not specified.

WS & SOA – terms overview



WS & SOA – Web Services architecture

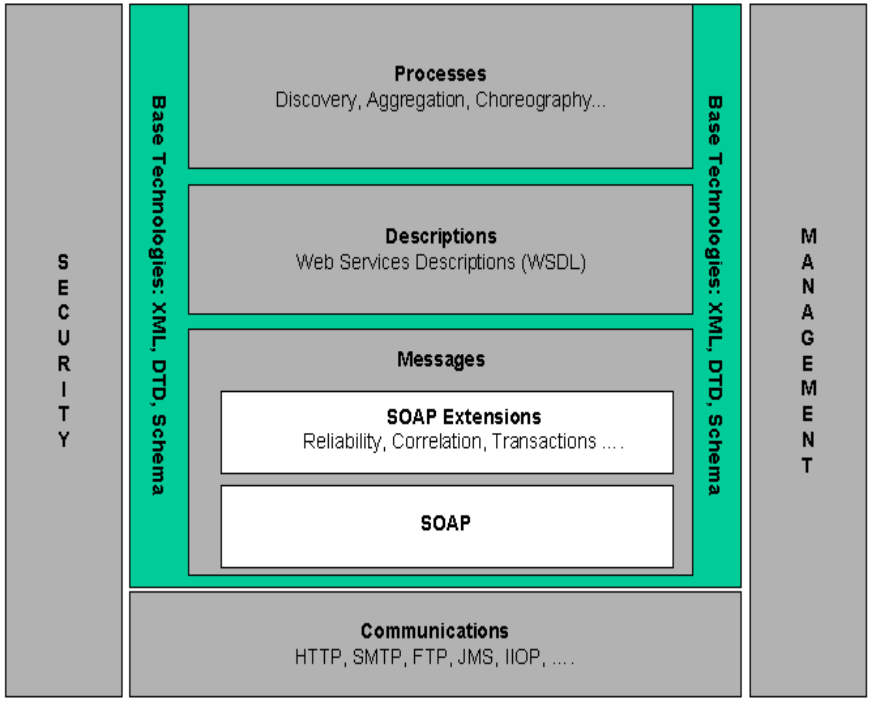
7 goals:

- Interoperability
- Reliability
- Integration with the World Wide Web
- Security
- Scalability and Extensibility
- Team Goals
- Management and Provisioning

4 Architectural Models (Views):

- Message Oriented Model
- The Service Oriented Model
- The Resource Oriented Model
- The Policy Model

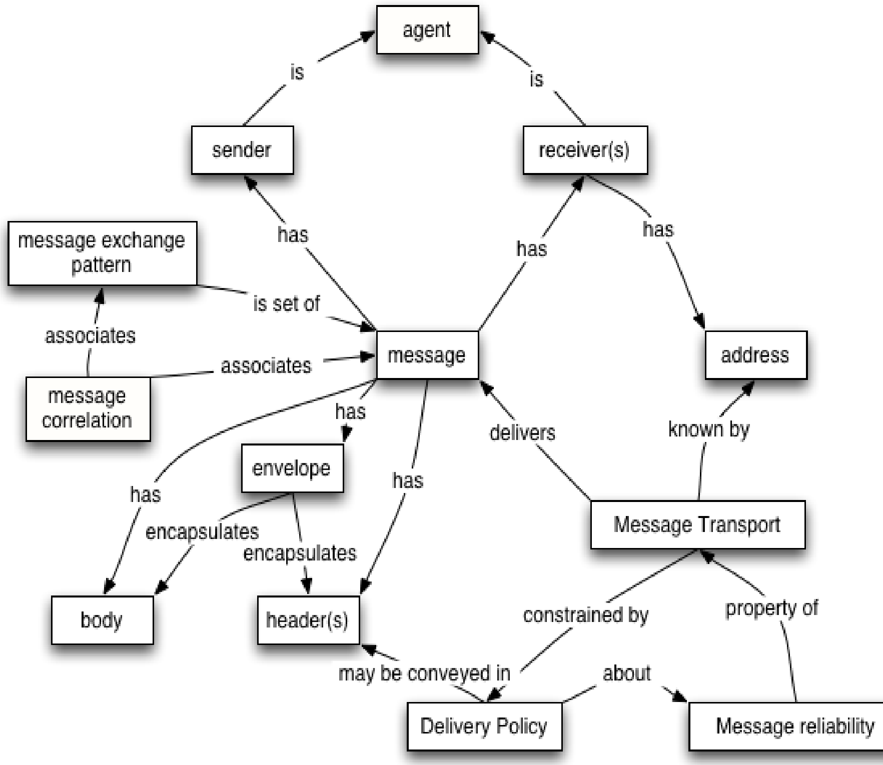
technologies:


SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...
MANAGEMENT

Claus P. Priese

WS & SOA – Message Oriented Model

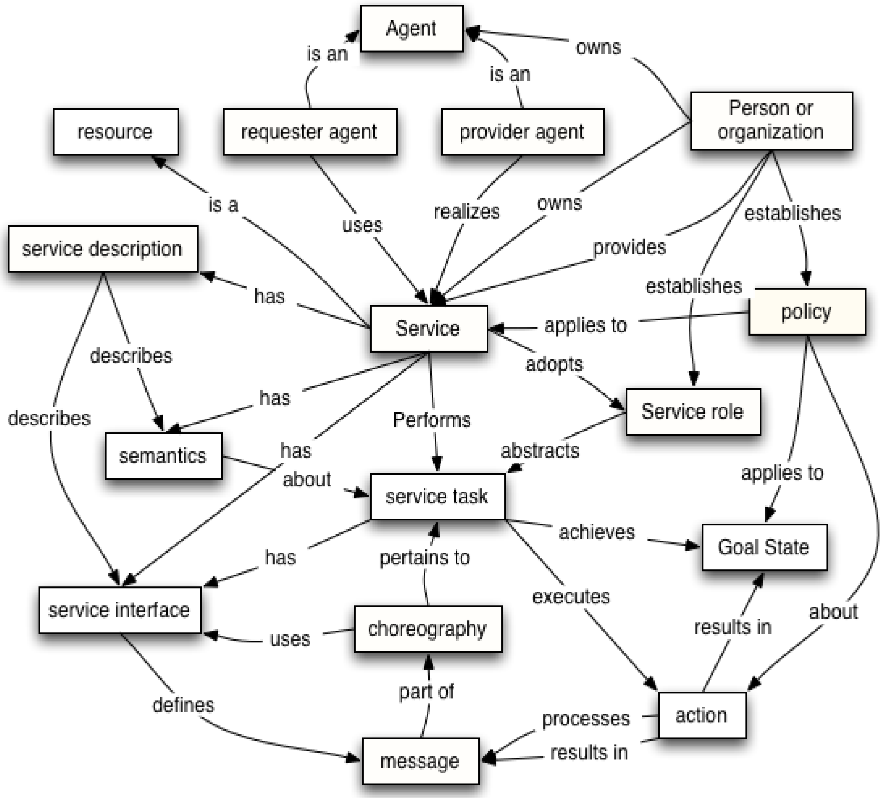
- message := unit of data sent from agent to agent, data-structure described in service-description-language
- Subject: relationship between sender and receiver
- Receiver always has transportmechnism-complient identifier (can be an URI)
- Message Exchange Patterns MEP describe groups of messages between agents



Claus P. Priese

WS & SOA – Service Oriented Model

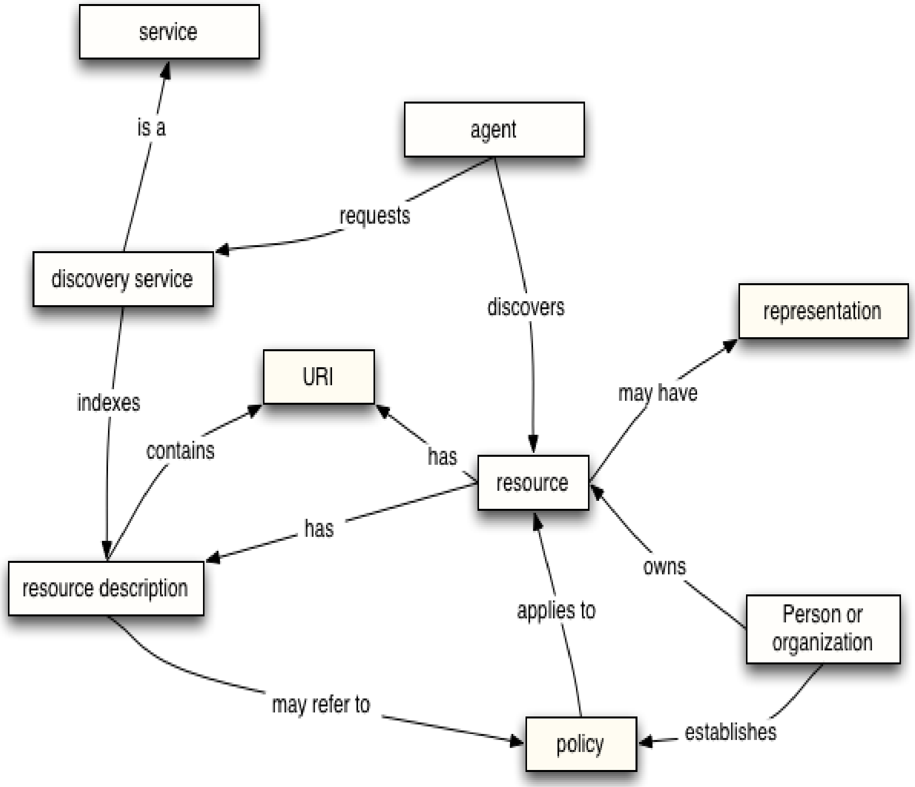
- Service := abstract resource executing task for person or organisation
- Service has interface, description and semantics
- Choreographie describes sequence and condition for collaboration of multiple agents in choreographie-description-language (WS-CDL)



Claus P. Priese

WS & SOA – Resource Oriented Model

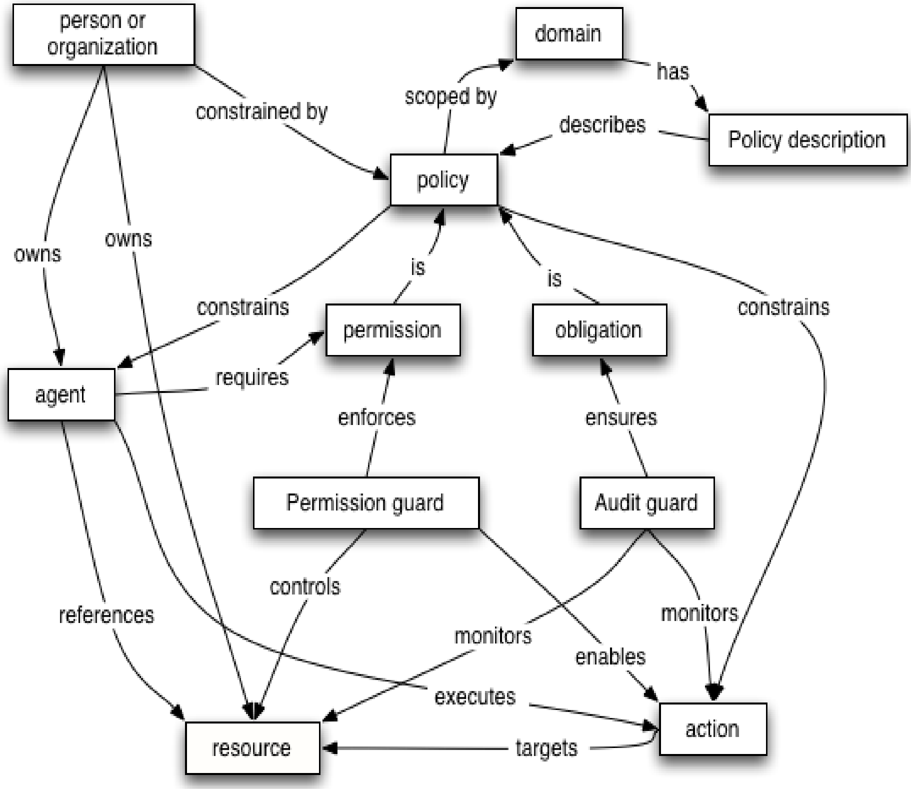
- Resource always has identifier (URI) and description
- Resources can be discovered by agents with use of discovery-service
- Resources are owned by person or organization and policies can be set on it



Claus P. Priese

WS & SOA – The Policy Model

- Policies contrain the behaviour of agents
- Policies are related to and derived from an application-domain
- Policies are set and owned by persons or organisations
- Two types of policies: permissions and obligations with two enforcement guard



Claus P. Priese

WS&SOA -ServiceOrientedArchitecture

Summary - 6 characteristics of SOAs from [W3C04h]:

- **Logical view**: service is abstracted, *logical* view, defined in terms of what it *does*
- **Message orientation**: service formally defined in terms of msg.exchange between provider agents and requester agents; service is not definend by properties of agents themselves
- **Description orientation**: services described in machine-readable form.
- **Granularity**: services use small number of operations with large and complex messages.
- **Network orientation**: services normally use over network; local use possible too
- **Platform neutral**: Messages in platform-neutral, standardized format delivered through interfaces. (language: XML)

WSDL

WSDL stands for **Web Serives Description Language** and is developed from working groups of W3C in conjuntion with the Web Services Architecture.

Most important basic specification documents are:

- [W3C04j] World Wide Web Consortium: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Working Draft, 3. August 2004, http://www.w3.org/TR/wsdl20
- [W3C04k] World Wide Web Consortium: Web Services Description Language (WSDL) Version 2.0 Part 2: Predefined Extensions, W3C Working Draft, 3. August 2004, http://www.w3.org/TR/wsdl20-extensions
- [W3C04l] World Wide Web Consortium: Web Services Description Language (WSDL) Version 2.0 Part 3: Bindings, W3C Working Draft, 3. August 2004, http://www.w3c.org/TR/wsdl20-bindings

WSDL - Characteristics

- Own WSDL-component model
- Own WSDL-Namespace(s): *wsdl, wsdli, wsdls, wrpc, wsoap and whttp*
- Independence from any serialization format (i.e. external representation), default is XML/XML-Schema
- Own simple Types: *wsdls:string, wsdls:Token, wsdls:NCName, wsdls:anyURI, wsdls:QName, wsdls:boolean, wsdls:int*

WSDL – Components 1 – definitions

```
<definitions
      targetNamespace="xs:anyURI" >
  <documentation />?
  [ <import /> | <include /> ]*
  <types />?
  [ <interface /> | <binding /> | <service /> ]*
</definitions>
```

A definitions-element is

- a container for all other WSDL-elements (interfaces, bindings, services)
- the place for targetNamespace-specifiation of contained elements
- The place for type-specification in element types
- The place for modularisation of WSDL-Documents by use of import/include

WSDL – Components 2 – interfaces

```
<definitions> <interface
        name="xs:NCName"
        extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> | <property /> ]*
  </interface> </definitions>
```

A interface-element

- Groups together for a service a set of message-exchanges described in operations
- Can be extended / be contained in an inheritance-hierarchy

WSDL – Components 3 – interface faults

```
<definitions>  <interface>
    <fault
          name="xs:NCName"
          element="xs:QName"? >
      <documentation />?
      [ <feature /> | <property /> ]*
    </fault>
  </interface> </definitions>
```

Interface faults

- Appear while invocation of interface-operations, i.e. by network-connection-loss or operation-abort
- are defined at interface-level to be reusable for different operations

WSDL – Components 3 - operation

```
<definitions> <interface> <operation
          name="xs:NCName"
          pattern="xs:anyURI"
          style="list of xs:anyURI"?
          safe="xs:boolean"? >
      <documentation />?
      [ <feature /> | <property /> |  [ <input /> | <output /> | <infault /> | <outfault /> ]+  ]*
    </operation> </interface> </definitions>
```

Operations

- model for an interface a set of message exchanges(inputs, outputs) of a service
- Connects to a Message-Exchange-Pattern (MEP) which specifies exact message-sequences and cardinality
- Allows the specification of Rules for the contained message references (<input />, <output /> ..)

WSDL – RPC-Style

RPC-Style is one special operation-style (*http://www.w3.org/2004/08/wsdl/style/rpc*) for use with XML-Schema as the Message-Format-Description-Language and these rules:

- the content of input and output elements MUST be a complex type containing a sequence
- the sequence MUST only contain elements and no other structures such as xs:choice
- the sequence MUST contain only local element children. child elements MAY have attributes: nillable, minOccurs and maxOccurs.
- the LocalPart of input element's QName MUST be same as Interface operation component's name.
- the LocalPart of output element's QName is name of operation concatenated with "Response".
- Input and output elements MUST both be in the same namespace.
- complex type that defines body of input or output element MUST NOT have attributes.
- childelements of input and output with same qual. name, MUST use same type.
- input or output sequence MUST NOT contain multiple childelements with same name.

WSDL - Signatures

When RPC-Style is used: signature extension defines mathematical function ***f*** of given interface operations

Signature extension is an additional attribute containing a list of pairs (q, t) with types *wsdls:Qname* and *wsdls:Token, under these conditions:*

- Only tokens *#in, #out, #inout* and *#return* are allowed
- value of the first component of each pair *(q, t)* MUST be unique in list
- For each child element of the input and output messages of the operation, a pair *(q, t)* whose first component *q* is equal to the qualified name of that element MUST be present in the list
- For each pair *(q, #in)*, there MUST be a child element of the input element with a name of *q* and there MUST NOT be a child element of the output element with the same name.
- For each pair *(q, #out)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.
- For each pair *(q, #inout)*, there MUST be a child element of the input element with a name of *q* and there MUST be a child element of the output element with the same name. Furthermore, those two elements MUST have the same type.
- For each pair *(q, #return)*, there MUST be a child element of the output element with a name of *q* and there MUST NOT be a child element of the input element with the same name.

Then for the operation of an interface:

The input parameter of the mathematical function ***f*** are the message references marked with *#in, #out, #inout*

- The values of the mathematical function ***f*** are the message references marked with *#return*

WSDL – Components 4 – message reference

```
<definitions> <interface> <operation>
   <input messageLabel="xs:NCName"?
         element="union of xs:QName, xs:Token"? >
    <documentation />?
    [ <feature /> | <property /> ]*
   </input>
   <output messageLabel="xs:NCName"?
          element="union of xs:QName, xs:Token"? >
    <documentation />?
    [ <feature /> | <property /> ]*
   </output>
  </operation> </interface> </definitions>
```

- Connect message references in MEP with concrete datatypes defined in global „types“-Element of surrounding definition

WSDL – Components 5 – fault reference

```
<definitions> <interface> <operation>
   <infault ref="xs:QName"
          messageLabel="xs:NCName"? >
    <documentation />?
    [ <feature /> | <property /> ]*
   </infault>*
   <outfault ref="xs:QName"
           messageLabel="xs:NCName"? >
    <documentation />?
    [ <feature /> | <property /> ]*
   </outfault>*
  </operation> </interface> </definitions>
```

- Connect fault message references in MEP with concrete interface fault of surrounding interface
- Two possible Fault-Message-Exchange-Patterns are: „fault-replaces-message“ and „message-triggers-fault“

WSDL – Components 6 – feature

```
<feature
     uri="xs:anyURI"
     required="xs:boolean"? >
  <documentation />?
</feature>
```

- feature enables to add external conditions and rules (specified by an URI) to be considered when messages are exchanged
- More than one feature can be present, all must be considered
- Required tells whether an requester MUST consider the rules

WSDL – Components 7 - property

```
<property
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
  [ <value /> | <constraint /> ]?
</property>
```

- Properties include with an URI named runtime-values into WSDL-Desriptions
- The named runtime-value can be constraint by <constraint />-Element
- Constants can be included by <value />

WSDL – Components 8 - binding

```
<definitions> <binding
        name="xs:NCName"
        interface="xs:QName"?
        type="xs:anyURI" >
    <documentation />?
    [ <fault /> | <operation /> | <feature /> | <property /> ]*
  </binding> </definitions>
```

A Binding specifies concrete details about the implementation of an service-interface and its operations about used protocols and used endpoints

If a concrete binding adds extension-elements the type-attribute contains the location for these

WSDL – Components 9 – binding fault

```
<definitions> <binding> <fault
        ref="xs:QName" >
     <documentation />?
     [ <feature /> | <property /> ]*
   </fault> </binding> </definitions>
```

- Describes the concrete binding of a fault-messageformat to a interface fault, which is identified by combination of interface-namespace and fault-name
- „ref“ contains the name specified by the fault-binding-component inside the interface specified by the surrounding binding-component

WSDL – Components 10 – binding operation

```
<definitions> <binding> <operation
          ref="xs:QName" >
      <documentation />?
      [ <input /> | <output /> | <feature /> | <property /> ]*
    </operation> </binding> </definitions>
```

binding of an interface operation for an endpoint to concrete messageformats and details of used protocol

„ref“ contains the name specified by the interface-binding-component inside the interface specified by the surrounding binding-component

WSDL – Components 11 – binding message reference

```
<definitions> <binding> <operation>
      <input messageLabel="xs:NCName"? >
        <documentation />?
        [ <feature /> | <property /> ]*
      </input>
      <output messageLabel="xs:NCName"? >
        <documentation />?
        [ <feature /> | <property /> ]*
      </output>
    </operation> </binding> </definitions>
```

- Descibes the concrete binding of message-formats to messages in interface-operations

WSDL – Components 12 – service

```
<definitions> <service
        name="xs:NCName"
        interface="xs:QName" >
    <documentation />?
    <endpoint />+
    [ <feature /> | <property /> ]*
  </service> </definitions>
```

- Describes a set of endpoints implementing the describes service
- „Interface“ contains the name of the interface this service represents

WSDL – Components 13 – endpoint

```
<definitions> <service> <endpoint
        name="xs:NCName"
        binding="xs:QName"
        address="xs:anyURI"? >
     <documentation />?
     [ <feature /> | <property /> ]*
   </endpoint> </service>+ </definitions>
```

- Endpoints contain the exact network adress (attribute „address“ of the implementation for a service within the binding specified in attribute „binding“.

WSDL – Components 14 – types

```
<definitions>
  <types>
    <documentation />?
    [extension elements]*
  </types>
</definitions>
```

- Contains all message- and fault-data types.
- If XML ist used as external representation language (as by default) the types are described as XML-Schema-Elements

WSDL – include and import

```
<definitions>
  <include
        location="xs:anyURI" >
    <documentation />?
  </include>
</definitions>
```

```
<definitions>
  <import
        namespace="xs:anyURI"
        location="xs:anyURI"? >
    <documentation />?
  </import>
</definitions>
```

- including WSDL-Descripition places the included elements in the same namespace as given by surrounding definition
- importing places the imported elements in a separate namespace specified in attribute „namespace“

Summary and Outlook

- SOAs provides the means of choice for todays distributed systems interconnection
- Web Services and WSDL are appropriate Architecture and infrastructure standards
- What to do with Web Services?

  Answer: systems for work- and business-processoriented combination of services
  - Modelling languages and process description standards
  - Petri Nets for secure logic-based formal modelling