Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Web Service Technologies: SOAP + WSDL + UDDI

SOAP

- Message Exchange
- SOAP Message Structure
- SOAP Encoding
- Programming Issues

WSDL

- Purpose & Scope
- Structure
- Programming & Use

UDDI

- Concept
- Data Structure

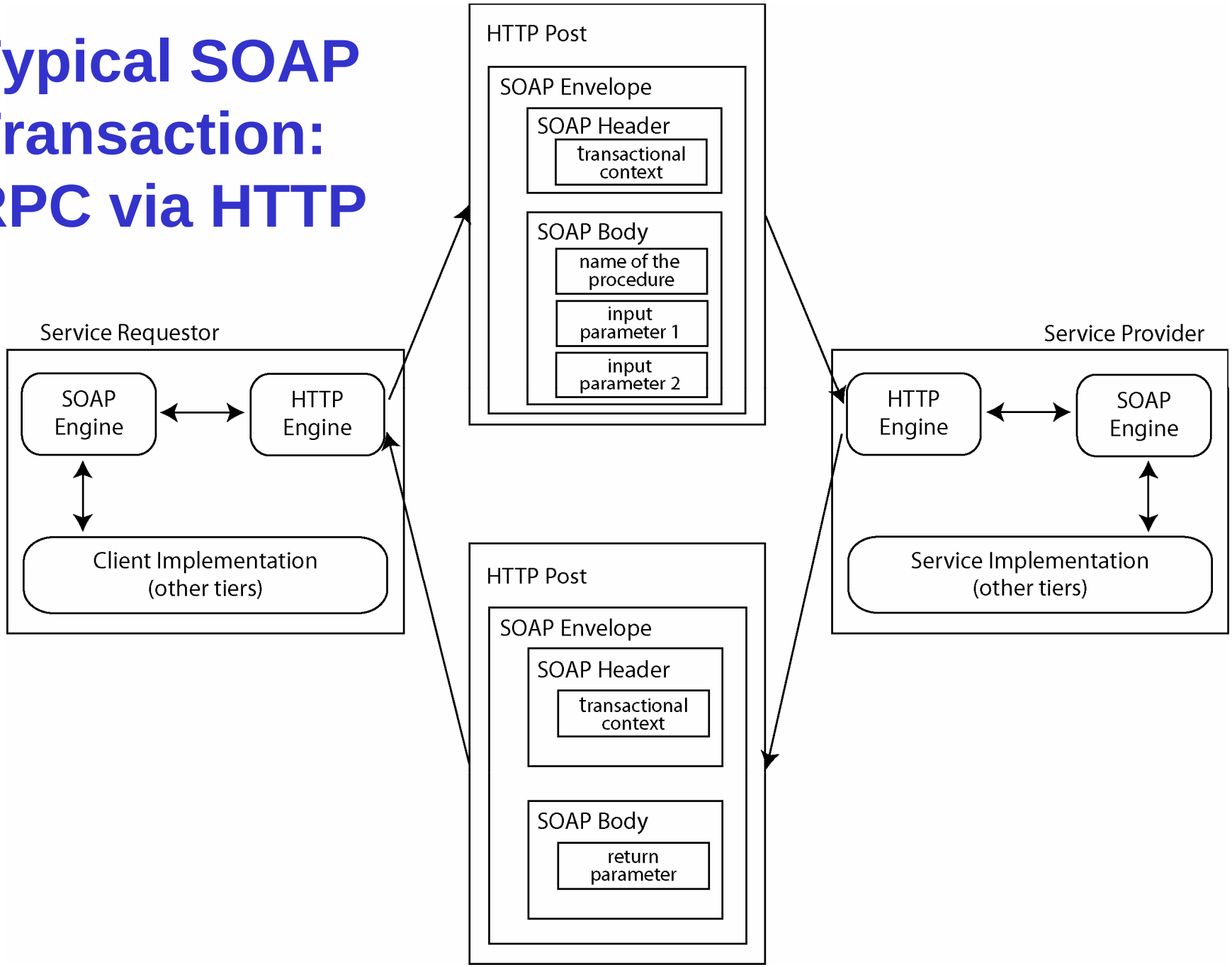
SOAP Message Exchange Model

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

A SOAP message in principle is a one-way transmission of an envelope from sender to receiver, but

- Messages may pass through various intermediate processors.
- Each intermediate processor may transform/enrich the previously received message (pipelining).
- A Message Path (routing) may be defined in the envelope header
- Processors along the path are called Actors

Typical SOAP Transaction: RPC via HTTP



SOAP via http: SOAPAction Header

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- http Header specified by the SOAP WG (depreciated)
- Design to indicate SOAP intent to http server
- Major use: blocking of (unwanted) SOAP requests (firewalls …)
- Now: ‘action + URI’ optional parameter of the **`application/soap_xop+xml`** Media Type

```
HTTP/1.0
…
Content-Type: text/xml; charset=utf-8
Content-Length: 456
SOAPAction: “urn:mymethod#myaction”
```

Asynchronous SOAP Message Exchange

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

RPC-style interactions result in a tight inter-dependence of components. To generate a loose coupling of actors, SOAP may be

- bound to an asynchronous transport protocol s.a. SMTP
  - See W3C: SOAP Version 1.2 Email Binding
- implemented via asynchronous RPC handling by
  - placing SOAP communication in concurrent threads
  - placing SOAP messages in buffer queues

Session States of SOAP Services

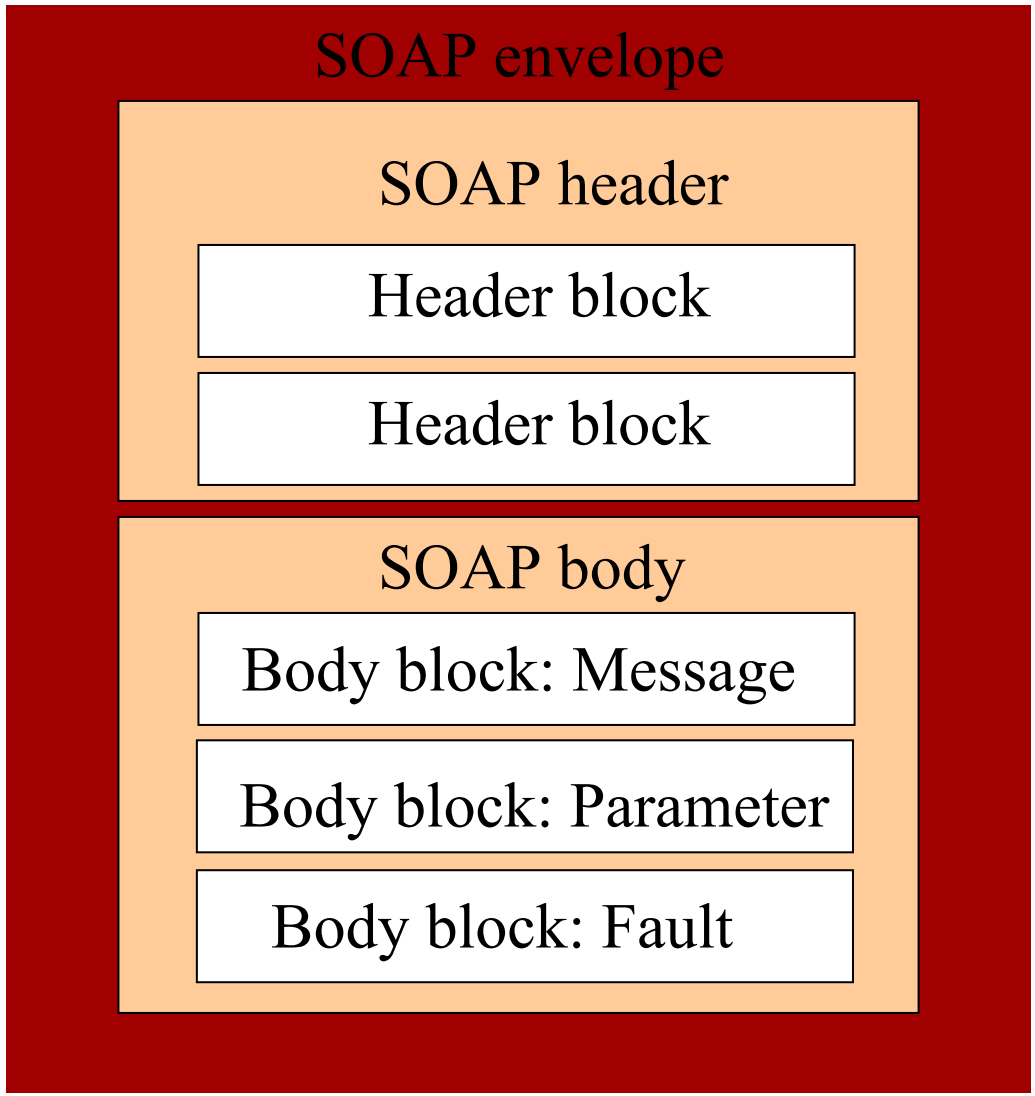
Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

SOAP message exchange is per default stateless. Session states may be preserved according to deployment parameters:

- Within `<service>` tag use
  - `<parameter name="scope" value="request"/>` (default) every request causes a new instance of the service class
  - `<parameter name="scope" value="session"/>` placing an instance of the service class is preserved during a session (derived from `SimpleSessionHandler`)
  - `<parameter name="scope" value="application"/>` only one instance of the service class is initiated
- Client code might account for states by re-using call object

SOAP Message Structure

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



SOAP Message Envelope

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Mandatory root element of any SOAP messages
- Defines SOAP version via a namespace
- Two possible child elements:
  - Soap Header
  - Soap Body

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
  “http://schemas.xmlsoap.org/soap/envelope”>
   <SOAP-ENV:Body>
      …
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Header

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Optional element
- Flexible framework to specify additional attributes, e.g. transaction management, AAA (transactional context)
- Two predefined header attributes:
  - Actor attribute: defines message path for chaining SOAP service nodes
  - mustUnderstand attribute: indicates whether a header element is optional or mandatory, i.e. must be processed

SOAP Header Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<SOAP-ENV:Header>
  <my:ServiceAccount xmlns:my=„urn:my.org„
      SOAP-ENV: mustUnterstand=„true“>
     bogus-7
  </my:ServiceAccount>
 </SOAP-ENV:Header>
```

Implementation:

Create/manipulate a header with `SOAPHeader`, `SOAPHeaderElement` from `org.apache.axis.message`

SOAP Body

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Mandatory element
- Contains messages and data in user defined blocks (XML encoded payload)
- One predefined optional body element
  - SOAP-ENV:Fault: specifies SOAP error conditions

SOAP Body

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<SOAP-ENV:Body>
  <my:ServiceRequest xmlns:my=„urn:my.org„>
     <my:ServiceType>Test</my:ServiceType>
  </my:ServiceRequest>
  <my:ServiceParams xmlns:my=„urn:my.org„>
      Parameters everywhere …
  </my:ServiceParams>
 </SOAP-ENV:Body>
```

Implementation:

- At Client: `setOperationName` and `addParameter` (Methods of `org.apache.axis.client.Service.call`)

SOAP Fault

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- A Webservice error will be returned with a `Fault` element in the body

Information carried in the `Fault` block:

- faultCode: text code indicating the class of errors
- faultString: human readable explanation of the error
- faultActor: string indicating the fault causing server
- detail: element to carry application specific detail related to processing the body of the message, may contain child elements (detail entries)

SOAP FaultCodes

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

SOAP `FaultCodes` may attain the following values:

- VersionMismatch: Indicates that the SOAP-ENV included an invalid namespace.
- MustUnderstand: Indicates that the recipient is unable to process a header with MustUnderstand attribute.
- Server: Indicates a server error, which is not directly related to the message processed.
- Client: Indicates that the server is unable to process the client request, e.g. a method is nonexistent or parameters are invalid.

SOAP Fault Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<SOAP-ENV:Body>
   <s:Fault>
      <faultcode>Client</faultcode>
      <faultstring> Invalid credentials</faultstring>
      <faultactor>http://myservice.org</faultactor>
      <details>
         <!-- application specific details -->
      </details>
   </s:Fault>
</SOAP-ENV:Body>
```

Implementation:

- At Client: Catch `javax.xml.soap.SOAPException`
- At Server: Extend exception handling with class
  `org.apache.axis.message.SOAPFault`

SOAP Encoding

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- In principle SOAP envelopes are designed to carry any well-formed XML document
- To agree on common data types SOAP allows to set an encodingStyle attribute – SOAP1.2:
  `SOAP-ENV:encodingStyle=`
  `“http://www.w3.org/2001/09/soap-encoding”`
- SOAP encoding includes simple types (taken from XSD Schema) and compound types
- Implementation of types:
  `org.apache.axis.encoding.*`

SOAP Compound Types

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Array:

```
…xsi:type="Array" arrayType="xsd:string[2]">
     <name> Charly Brown </name>
     <name> Snoopy Dog </name>
```

- Struct:

```
   …xsi:type="Name">
<firstname xsi:type="xsd:string">Charly</firstname>
<lastname xsi:type="xsd:string">Dog</lastname>
```

- Bytearrays:

```
     <binarydata xsi:type="base64">
             sdkIJWSNjnsjfdoi234sdi
     </ binarydata>
```

Array Service

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public double[] getSinusArray(double[] array){
        double[] retVal = (double[]) array.clone();

        for(int i =0 ; i<array.length; i++){
            retVal[i] = Math.sin(retVal[i]);
        }
        return(retVal);
    }
```

Array Client

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
private void look4Sinus() {
      Call call;
      Service service = new Service(); //create service

      try {
         call = (Call)service.createCall();
         call.setReturnType(javax.xml.rpc.encoding.XMLType.SOAP_ARRAY);   //specify the (array) return value

         call.setOperationName("getSinusArray");                          //specify invoked webservice method
         call.setTargetEndpointAddress( HOST + SERVICE_PATH );            //specify ws url

         //specify passed String parameter (name of enumeration)
         call.addParameter("array",  XMLType.SOAP_ARRAY, ParameterMode.IN);

         double[] numbers = new double[]{1.0,20.0,50.0};                  //create input value
        Object values[] = (Object[])call.invoke(new Object[] {numbers} );  //invoke web service

         System.out.println("Size of return ["+values.length+"]");        //output result
         for(int i =0; i< values.length;i++){
            System.out.println("value ["+i+"] ["+values[i]+"]");
         }
...
```

Literals/XML

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Instead of using SOAP encoded data one can transmit entire XML documents by

- encoding data as literalxml

```
SOAP-ENV:encodingStyle=
http://xml.apache.org/xml-soap/literalxml
```

- appending serialized XML to the envelope
- as attached XML file

Service with XML as String Parameter

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public String stringTransport(String xml){

      String ret = null;

      if(xml != null){
          try {
              //process  XML data
              ….(new StreamSource(new StringReader(xml)),result);

              //put processed data into return value
              ret = result.getWriter().toString();

          } catch (Exception ex) {
...
```

Client sending XML as String Parameter

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public void transportXMLString() throws Exception {
        …
        //exit if no sources are specified
        if (dsXML == null ) return;
        //create handler for data sources
        String stringXML = getStringFromDataSource(dsXML);
        call = (Call) service.createCall();          //create new call
        //register (passed) parameter for XML string
        call.addParameter("xml", XMLType.XSD_STRING, ParameterMode.IN);

        call.setReturnType(XMLType.XSD_STRING); //specify expected return type of WS-Method
        call.setOperationName(new QName("stringTransport")); //specify web service method
        call.setTargetEndpointAddress(HOST + SERVICE_PATH); //specify web service URI

        //invoke web service with passing the XML strings as parameters.
        String ret = (String) call.invoke(new Object[] { stringXML });
…
```

Service with XML as Attachment

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public DataHandler transport(DataHandler xml){
      DataHandler dhRet = null;

      if(xml != null){
          try {
              …
              … new StreamSource(xml.getInputStream()),result);


              …
              dhRet = new DataHandler(dsResult);
    } catch (Exception ex)
```

Client sending XML as Attachment

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public void transportXMLAttachments() throws Exception {
    …
    //exit if no sources are specified
    if (dsXML == null || dsXSL == null)  return;

    //create handler for data source
    DataHandler dhXML = new DataHandler(dsXML);
    …
    call = (Call) service.createCall();                    //create a new call
    //create qualified name for attachment type (DataHandler).
    QName qnameAttachment = new QName("DataHandler");
    //Add (default) serializer for attachments.
    call.registerTypeMapping(dhXML.getClass(), qnameAttachment, JAFDataHandlerSerializerFactory.class,
    JAFDataHandlerDeserializerFactory.class);

    //Register (passed) parameter for XML file.
    call.addParameter("xml", qnameAttachment, ParameterMode.IN);
    …
    //invoke web service with passing XML datahandler as parameter.
    Object ret = call.invoke(new Object[] { dhXML });
…
```

Java Beans

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

For more complex objects it is simplest working with beans:

- Provide the bean class at client and server
- SOAP will transport the state of the bean
- Encoding of beans is eased by built-in bean serializer classes (transform to XML)

```
or.apache.axis.encoding.ser.*
```

Bean Service

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public MyBean scramble(MyBean bean){
      //create return value
      MyBean retVal = new MyBean();

      //write text backwards (scramble text value)
      String text = "";
      for(int i =bean.getText().length();i>0;i--)
          text+=bean.getText().substring(i-1,i);

      //set new text value
      retVal.setText(text);


     //return scrambled object
      return(retVal);
 }
```

Bean Client

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
...
        bean = new MyBean();                                    //create bean
        call = (Call)service.createCall();                      //create call

        //create qualified name for attachment type (DataHandler).
        QName qname = new QName("http://myBean.org","MyBean",“my");

        //add (de-)serializer for bean
        call.registerTypeMapping(MyBean.class, qname, BeanSerializerFactory.class,
              BeanDeserializerFactory.class);
        call.addParameter("bean", qname, ParameterMode.IN);       //register (passed) parameter for bean

        call.setReturnType(qname);                        //specify expected return type of web service method
        call.setOperationName(new QName("scramble"));                    //specify web service method
        call.setTargetEndpointAddress(HOST + SERVICE_PATH);              //specify web service URI

      Object ret = call.invoke(new Object[] {bean});     //invoke web service with passing bean as parameter.
} catch (Exception ex) {
```

Own Objects

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

To include own objects in SOAP messages:

- Provide the object class at client and server
- Provide object (de-)serializer
- Provide (de-)serializer factory
- Register serializer with attachment
- Handle object in bean analogy

Programming: Service Versioning

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
public abstract class BaseService {
    /**
     * The version of the web service. Should be re-set on derviation via setVersion().
     * You can overwrite this string in your derviated class, too.
     */
    protected String VERSION = "To set correct Version, use setVersion() in constructor of web service.";

/**
     * Resolve the version of web service.     */
    public String getVersion() {
        return this.VERSION;
    }

    /**Internal way to (re-)set version string. */

    protected void setVersion(String version){
        this.VERSION = version;
    }

}
```

Programming: Client Checking Version

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
Service service = new org.apache.axis.client.Service();
        try {
            Call call = (Call) service.createCall();              //create call
            call.setReturnType(XMLType.XSD_STRING);  //specify expected return-type of call

           call.setOperationName("getVersion");               //specify called method

            //specify URL of web service
            call.setTargetEndpointAddress(HOST + SERVICE_PATH);

            //invoke remote method
            String ret = (String) call.invoke((Object[]) null);

          System.out.println("Version of WebService [" + ret + "]");

        } catch (ServiceException ex)
```

Programming: Obtaining a Service Logger

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
/**
  * Obtained Logger from JBoss.
 */
protected static Log log = LogFactory.getLog(MyService.class.getName());
```

Sample Codes

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- SessionbasedService
- OwnObjectService

Web Service Description Language (WSDL)

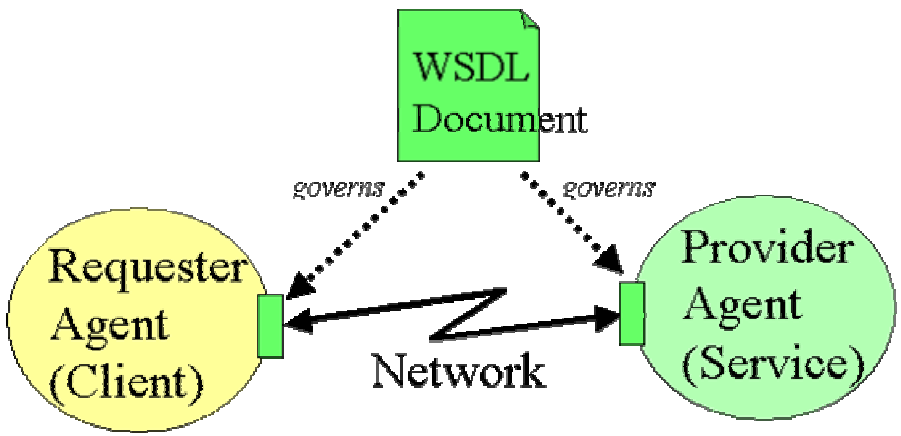
Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Interface description language for web services (like IDL in Corba).
- Additionally defines *service access* mechanisms and *locations* (→ no central middleware).
- XML encoded.
- Originally a proposal of IBM, Microsoft, Ariba, +++, now WSDL 1.1 W3C note (not recommendation)
- WSDL 2.0 work of W3C – on last call status (Oct. 04).

WSDL Purpose & Scope

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Governs the interaction between service provider and requester.
- Viewpoint: service provider.
- Describes ‘the mechanics’, i.e. syntax and behaviour (WSDL 2.0) of the service
- No description of the intended semantics.
- Limited to describing individual Web Service, not service chains.



WSDL

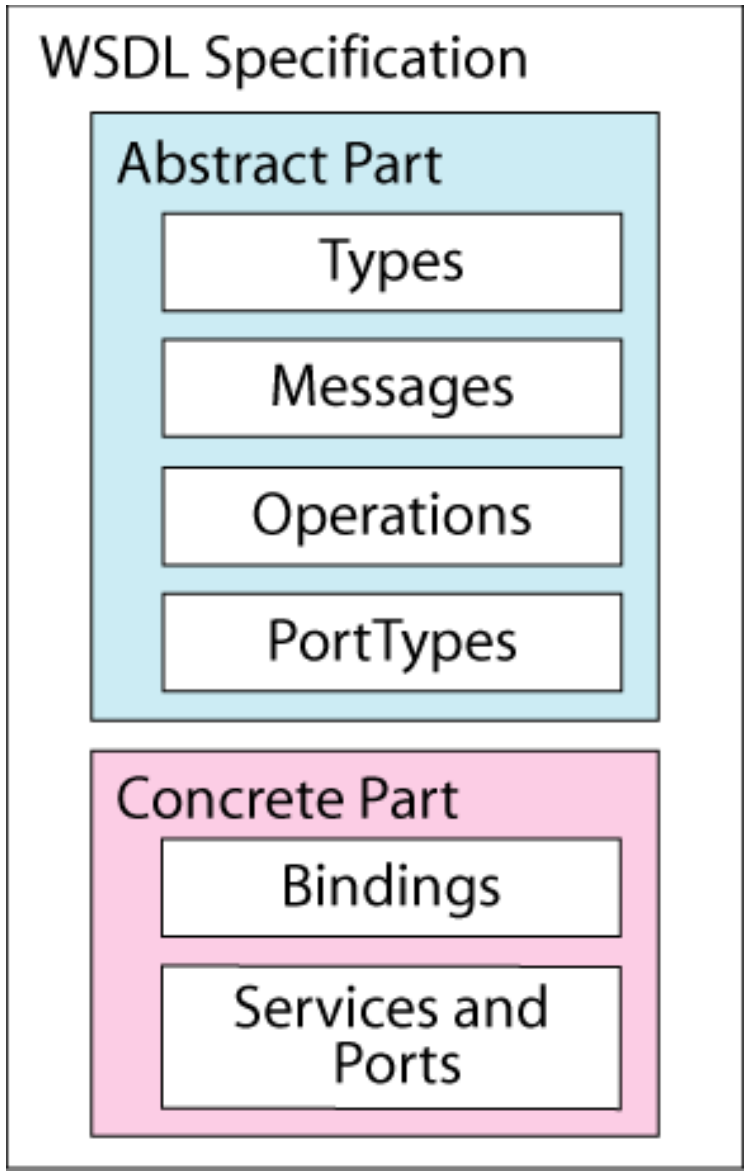
Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Abstract Part:

- Analogue to IDL.
- Defines Port Types as a collection of message exchange operations.

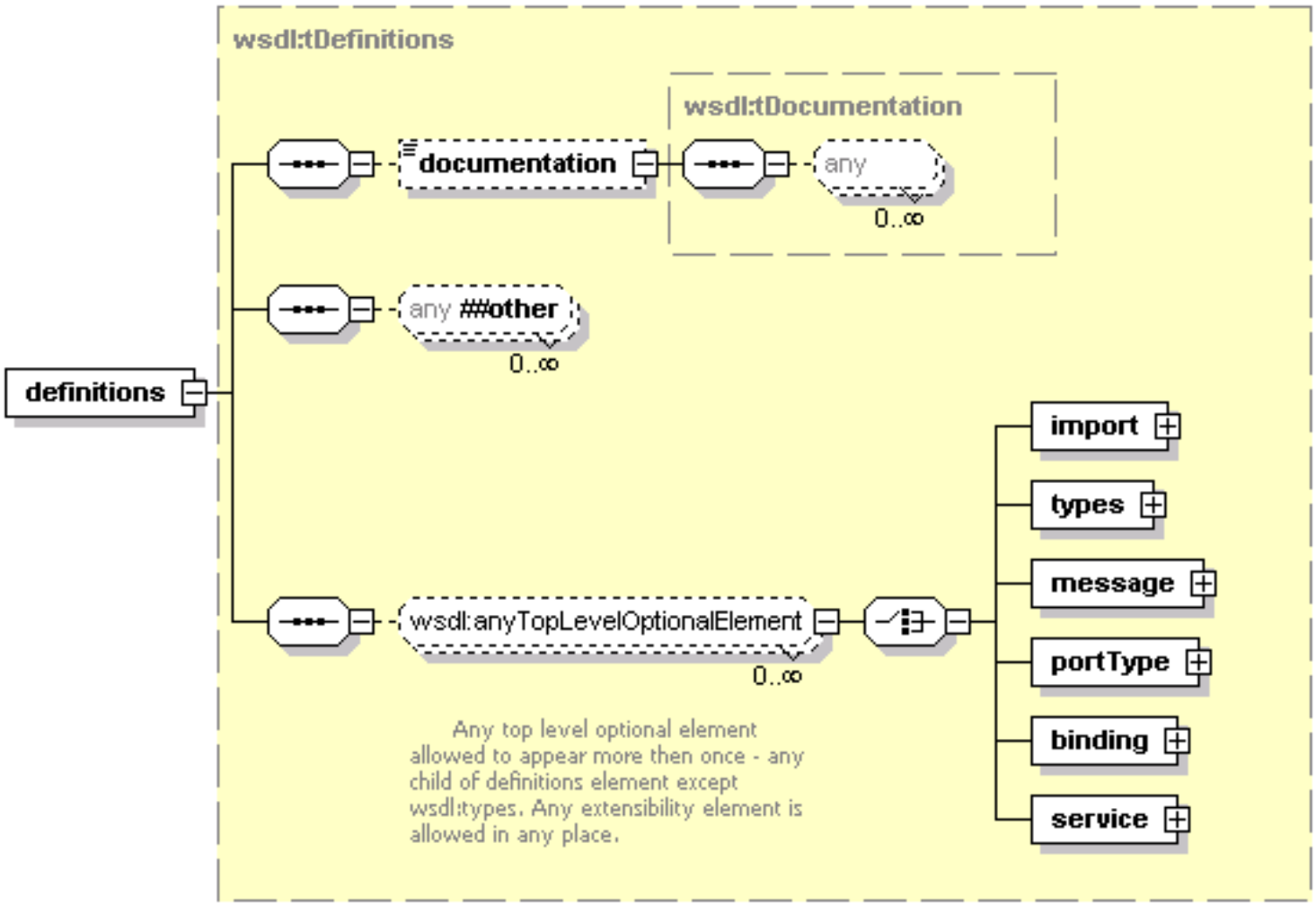
Concrete Part:

- Defines service access, i.e. encoding, protocol binding.
- and EndPoints as described by URIs.



WSDL Structure

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



All structural parts may contain human readable descriptions (documentation).

Types

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Describes all data types used between client and server.
- XML Schema types default.

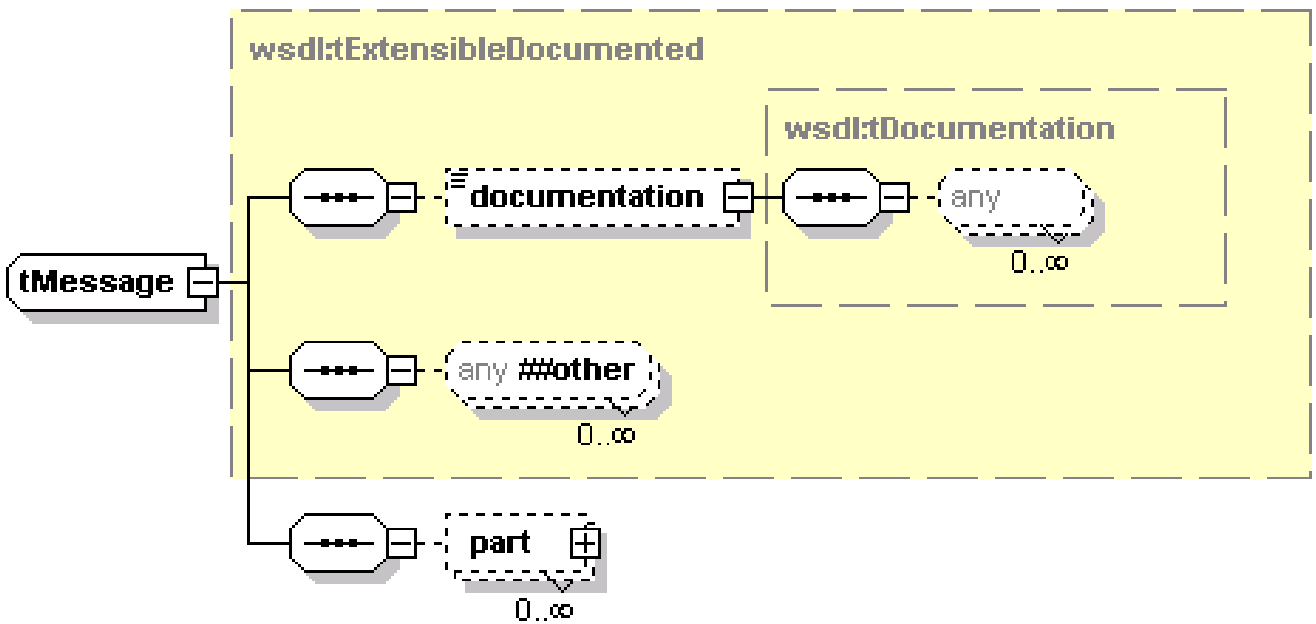
Example:

```
<types>
 <schema targetNamespace=http://www.fhtw-berlin.de
         xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="HylosObject">
     <sequence>
        <element name="aggregationLevel" type="xsd:int"/>
        <element name="keywords" nillable="true" type="soapenc:Array"/>
        <element name="relations" nillable="true" type="soapenc:Array"/>
     </sequence>
   </complexType>
   <complexType name="Relation">
     <sequence>
        <element name="reference" nillable="true" type="xsd:string"/>
        <element name="relation" nillable="true" type="xsd:string"/>
     </sequence>
   </complexType>
 </schema>
</types>
```

WSDL Message

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Describes a single, one-way message.
- Defines the name of the message.
- Parameters or return values are encoded as parts.



Message Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

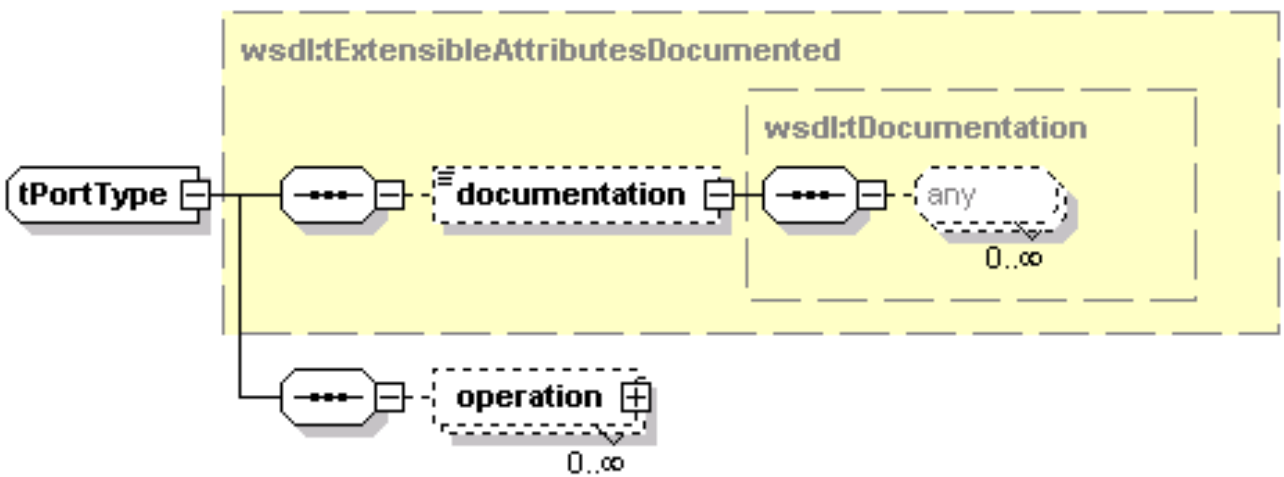
```
<message name="searchObjectsRequest">
   <part name="filter" type="xsd:string"/>
</message>

<message name="searchObjectsResponse">
   <part name="searchObjectsReturn"
         type="soapenc:Array"/>
</message>
```

WSDL PortType

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Combines multiple message elements to form an operation.
- May define several operations.
- Operations must conform to pre-defined patterns.



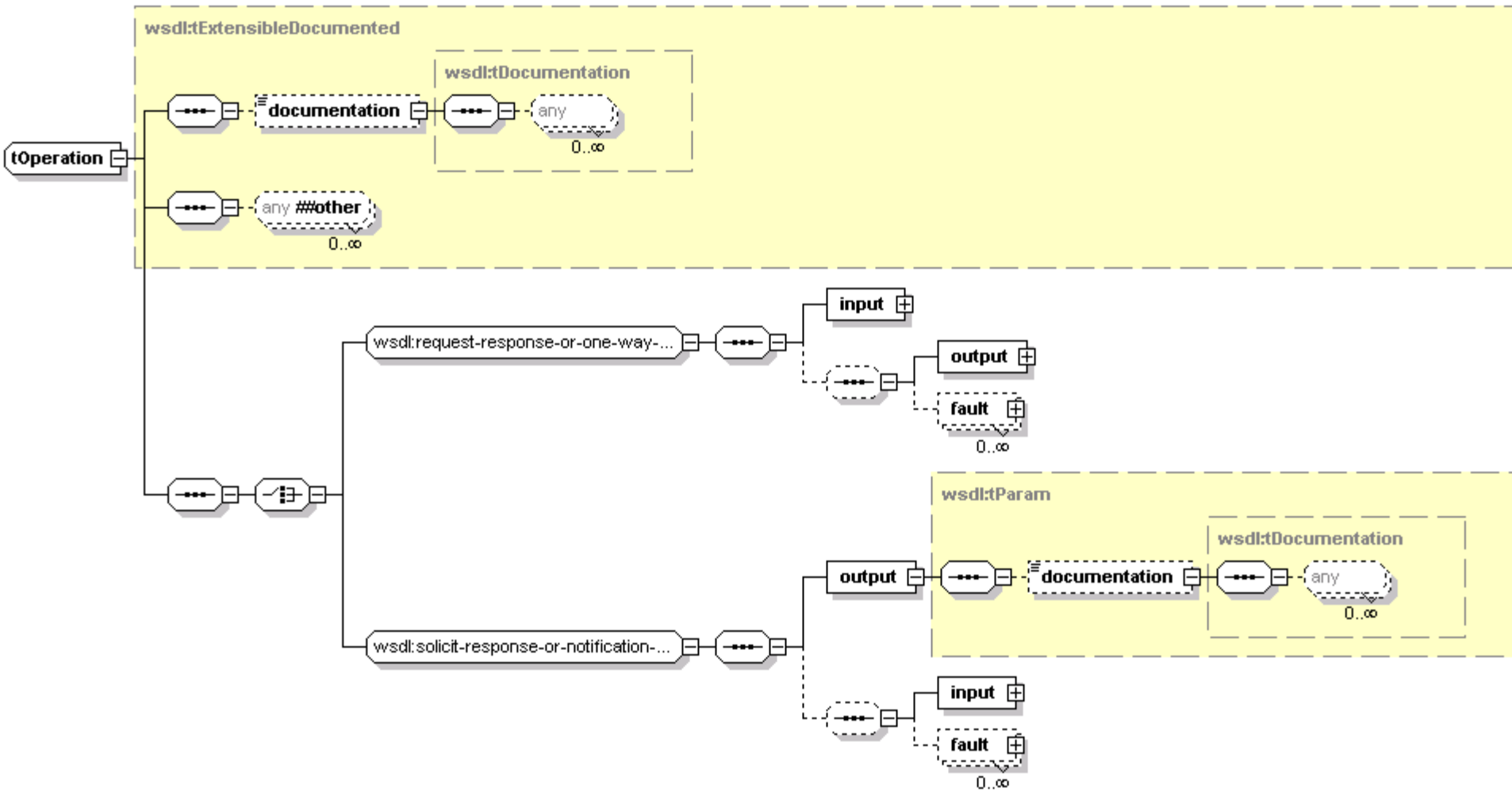
PortType Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<portType name="HylosObjectService">
  <operation name="fetchObject" parameterOrder="path">
      <input message="impl:fetchObjectRequest"
            name="fetchObjectRequest"/>
      <output message="impl:fetchObjectResponse“
            name="fetchObjectResponse"/>
  </operation>
  <operation name="searchObjects"
            parameterOrder="filter">
      <input message="impl:searchObjectsRequest“
            name="searchObjectsRequest"/>
      <output message="impl:searchObjectsResponse“
            name="searchObjectsResponse"/>
  </operation>
</portType>
```

WSDL Operations

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



WSDL 2.0 – Message Exchange Pattern (MEP)

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Abstract description of the sequence and cardinality of messages within operations

In-bound MEPs

- In-Only
- Robust In-Only
- In-Out
- In-Optional-Out

Out-bound MEPs

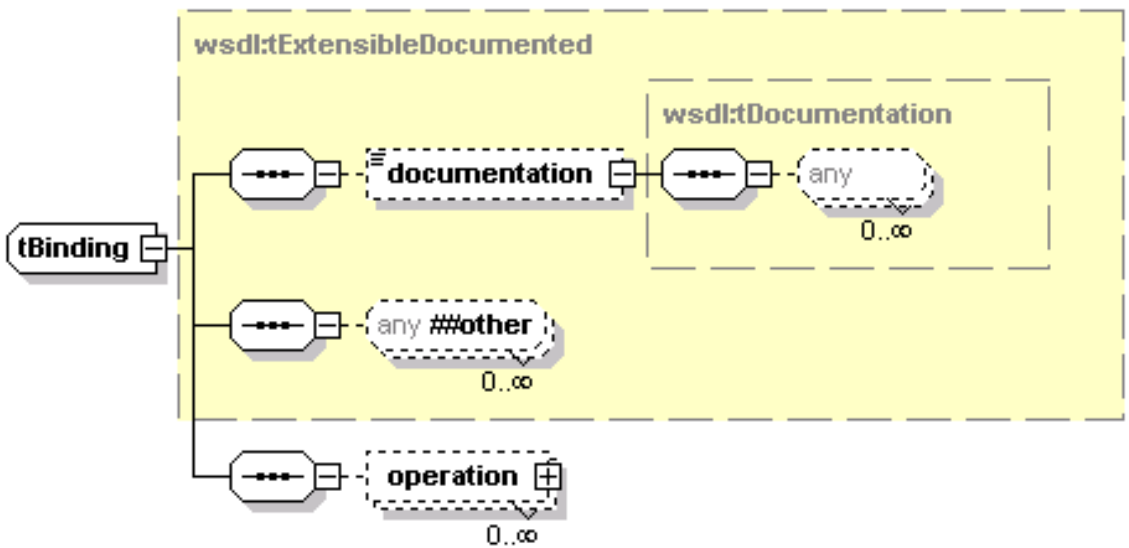
- Out-Only
- Robust Out-Only
- Out-In
- Out-Optional-In

MEPs purpose is merely the complete specification of functionality at the end point (e.g. out-bound descriptions)

WSDL Binding

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Makes abstract service definitions concrete with respect to
  - RPC protocols: assigns protocol actions to operations.
  - Transport protocols: specifies RPC transport
  - Encodings: fills messages/parameters into protocol payload



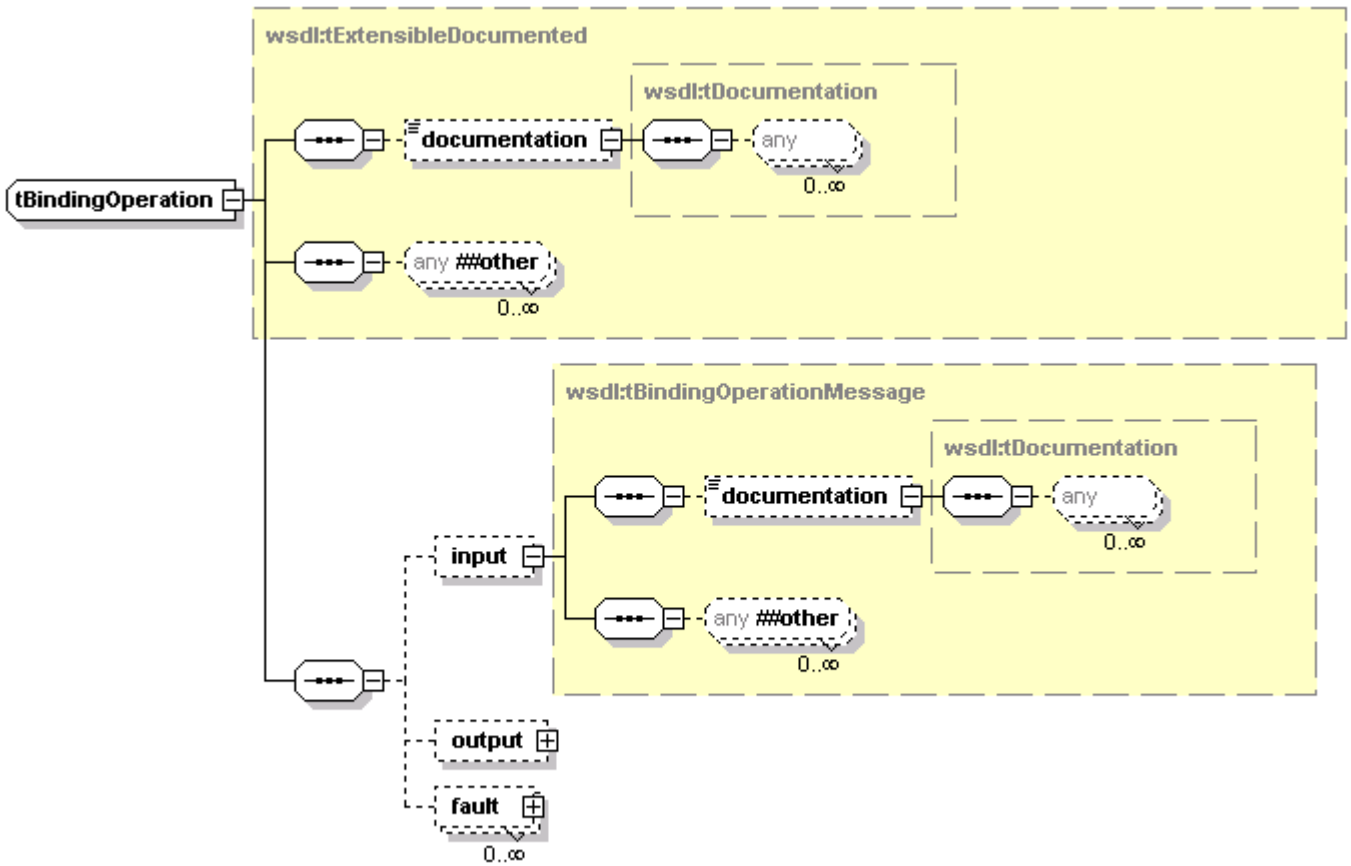
Binding Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<binding name="HylosObjectServiceSoapBinding" type="impl:HylosObjectService">
 <wsdlsoap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
 <operation name="fetchObject">
   <wsdlsoap:operation soapAction=""/>
   <input name="fetchObjectRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://hylos.de" use="encoded"/>
   </input>
   <output name="fetchObjectResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="…/HylosObjectService" use="encoded"/>
   </output>
 </operation>
 <operation name="searchObjects">
   <wsdlsoap:operation soapAction=""/>
   <input name="searchObjectsRequest">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://hylos.de" use="encoded"/>
   </input>
   <output name="searchObjectsResponse">
     <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="…/HylosObjectService" use="encoded"/>
   </output>
 </operation>
</binding>
```

WSDL BindingOperation

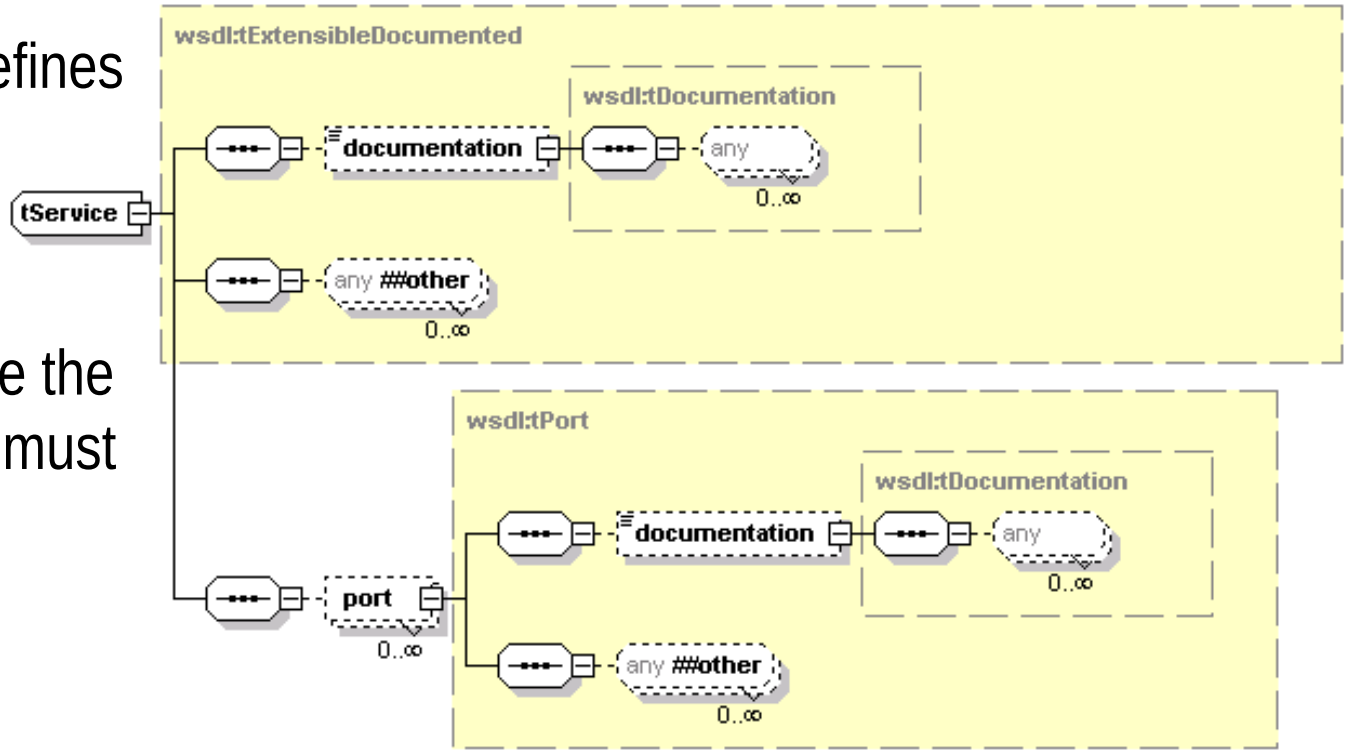
Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



WSDL Service

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Service element defines the address for invoking the service.
For a SOAP service the **`address`** element must specify an URL.



Example:

```
<service name="HylosObjectServiceService">
   <port binding="impl:HylosObjectServiceSoapBinding“
                        name="HylosObjectService">
        <wsdlsoap:address location="http://satan.rz.fhtw-
          berlin.de:8080/ws4ee/services/HylosObjectService"/>
   </port>
</service>
```

Implications of the WSDL Model

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Abstract definition part remains independent of any concrete binding

→ re-use of interfaces

- WSDL can describe services, which proactively invoke operations

→ peer-to-peer behaviour

- WSDL combined with SOAP forms a general RPC service description, which work with different XML encodings and transport protocols

→ standard for defining interactions

Using WSDL

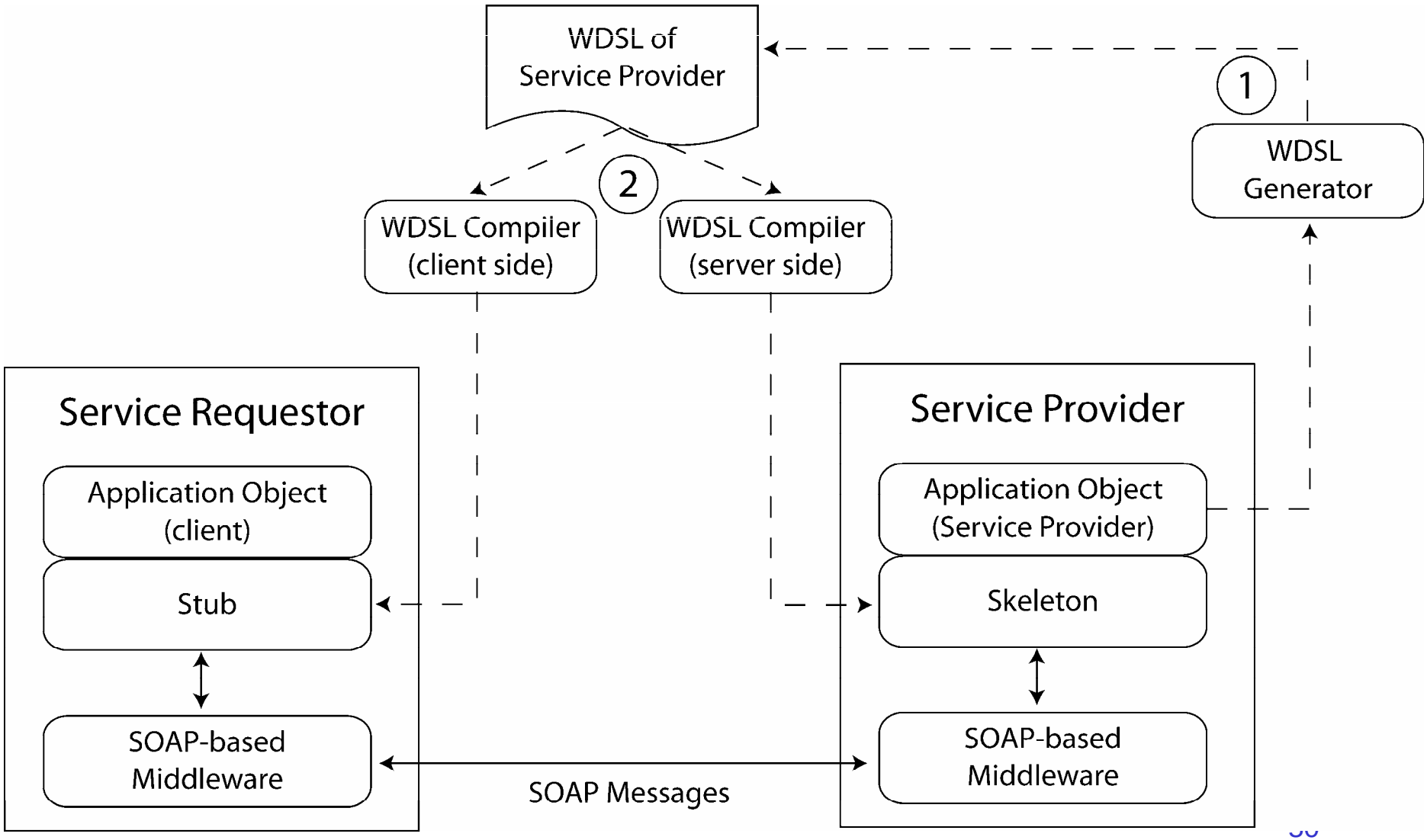
Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

WSDL is ‘just another IDL’, but its generality, its encoding in XML, thus its potential for pervasive automation lead to a widespread acceptance … for

1. Traditional service description contract between provider and requestor.
2. Input to stub compilers / automated processing tools for auto-generation of clients.
3. Service information offer, which eventually supplies descriptions on service behaviour and service meaning to client program designers –
   but: WSDL is not a semantic description layer.

Operations with WSDL

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



Programming: WSDL Generation

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

A WSDL file can be generated from a deployed Web Service.

Example:

```
java org.apache.axis.wsdl.Java2WSDL -o my.wsdl \
 -l “http://localhost:8080/axis/services/MyService” \
 -n "urn:MyNS“ p “mypackages.services“ "urn:MyNS“ \
      mypackages.services.MyService
```

Will produce the WSDL-file “my.wsdl”

Programming: Generating Stubs from WSDL

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

From a given WSDL file Client Stubs and Server Skeletons may be generated. The generic code generation for Clients produces:

- Bean classes for every types section
- an interface for every portType
- an interface and service implementation (locator) for every service section
- Client stubs for every binding section

Example:

```
import org.apache.axis.wsdl.WSDL2Java;
public class WSDL2JavaTests {

    public static void main(String[] args) {
      //create client side classes
      WSDL2Java.main(new String[]{"-a","-T1.1",“
               http://my.de/HylosObjectService.wsdl"});
    }
}
```

Programming: Generating Skeletons from WSDL

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

When generating generic server code (skeletons) from a given WSDL file, the following is produced:

- Bean classes for every types section, as they are needed at the server side
- deployment descriptors
- Server skeletons for every binding section, including implementations of predefined operations

Example:

```
import org.apache.axis.wsdl.WSDL2Java;
public class WSDL2JavaTests {

    public static void main(String[] args) {
      //create client side classes
      WSDL2Java.main(new String[]{"--server-side","-Strue","-a","-T1.1",“
                  http://my.de/HylosObjectService.wsdl"});
    }
}
```

Universal Description, Discovery and Integration (UDDI)

Thomas Schmidt
schmidt@informatik.haw-hamburg.de

- Framework for describing and discovering Web services (‘Business registry’)
- Advanced naming and directory service
- Technically a Web service
- Originally by IBM, Microsoft, Ariba, …
- Version 2.0 by OASIS org, Vers. 3.0 announced draft

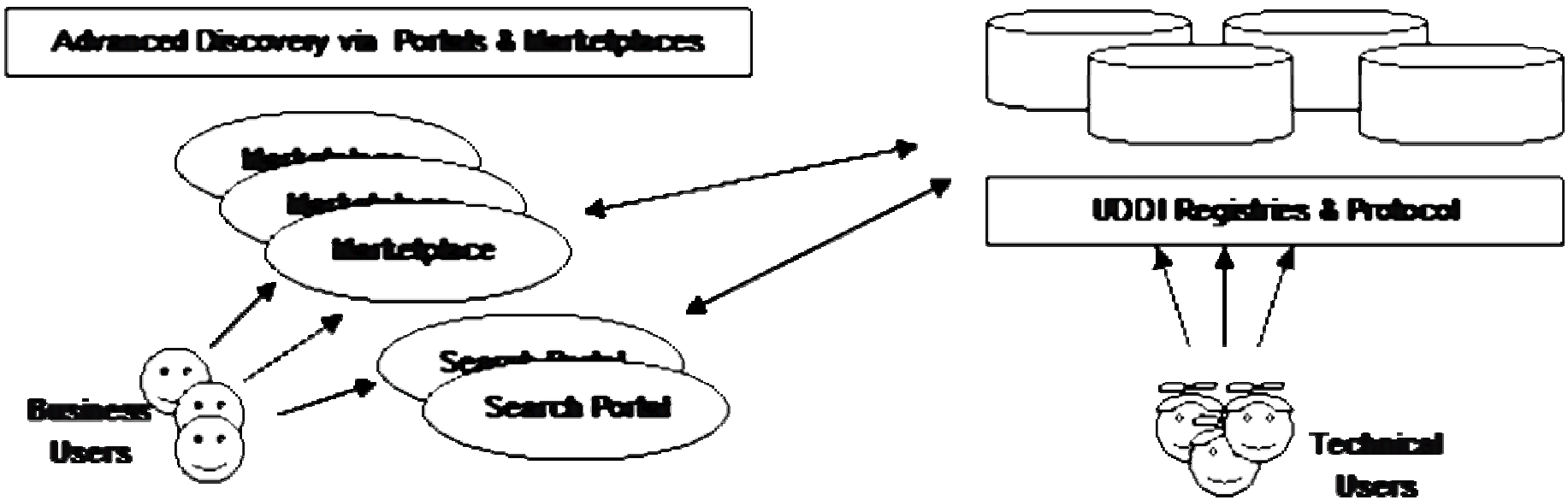
UDDI Objectives

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Supply information about services and their technical specification to client developers
  (→ unstructured information)
- Enable dynamic binding from clients, automatically searching the registry and obtain references to services of interest (→ structured information)
- Also: Idea of Universal Business Registry

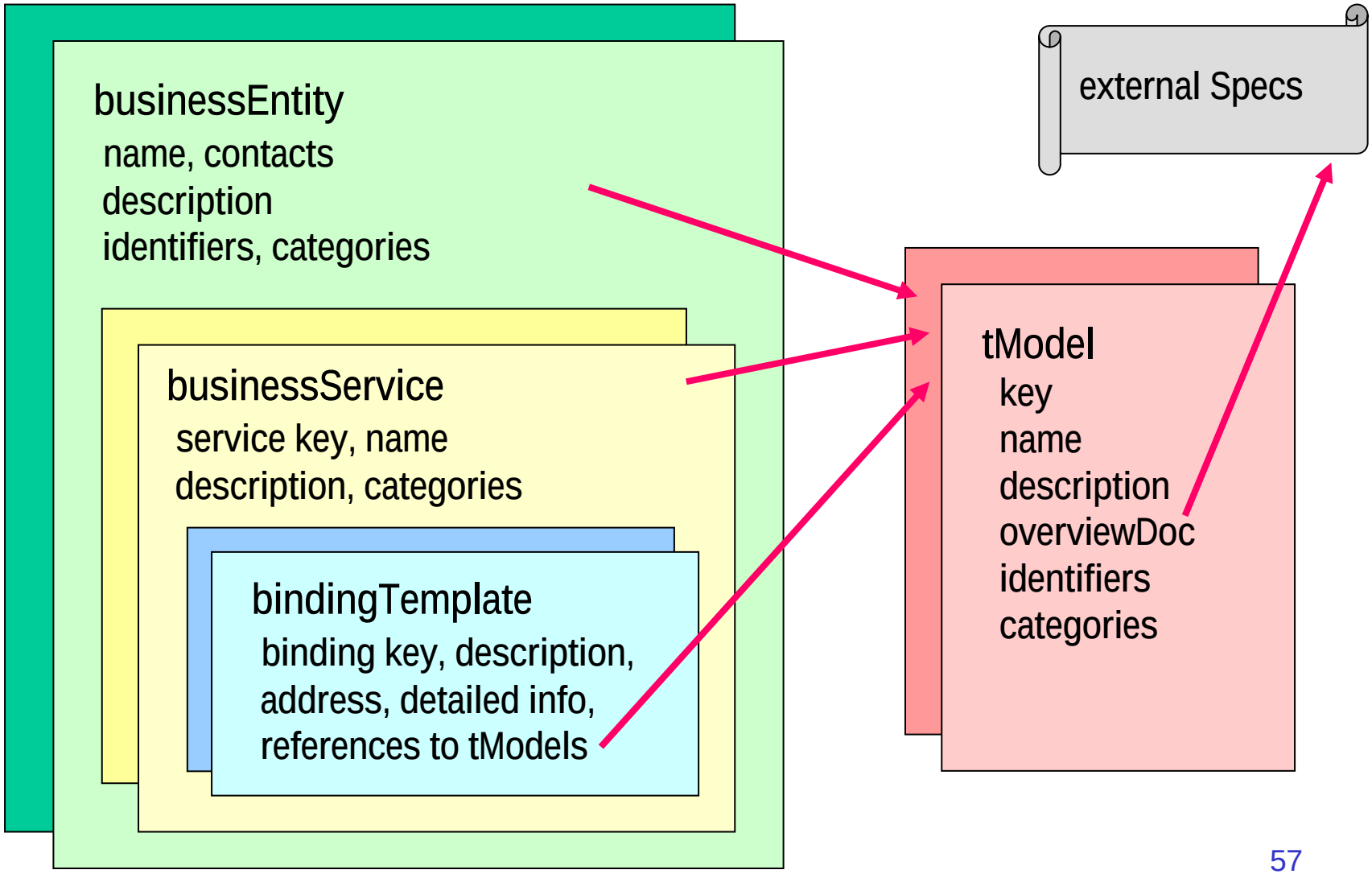
UDDI Use Cases

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



UDDI Data Model

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de



Data Entities

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

businessEntity:

Describes Web service providing organisation (unique businessKey)
Commonly categorised by several classification standards (NAICS for industry type, UNSPC for products and services, ISO 3166 for geography, ..)

businessService:

Describes group of related Web services

bindingTemplate:

Describes technical information to employ Web service

Technical Models (tModels)

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- Central data structure to supply technical and non-technical information
- Unique ID assigned to every published tModel
- Basic provision: Pointer to external document (overviewDoc), which describes the service

→ tModel allows for insight into a service

→ tModels can be re-used

→ Any service characterised by a ‘known’ tModel is already understood (e.g. by an appropriate client for that tModel)

tModel Example

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

```
<tModel tModelKey="uddi:uddi.org:v3_publication">
  <name>uddi-org:publication_v3</name>
  <description>UDDI Publication API V3.0</description>

  <overviewDoc>
    <overviewURL usedType="wsdlInterface">
        http://uddi.org/wsdl/uddi_api_v3_binding.wsdl
    </overviewURL>
  </overviewDoc>
  <overviewDoc>
    <overviewURL usedType="text">
        http://uddi.org/pubs/uddi_api_v3.html
    </overviewURL>
  </overviewDoc>

  <categoryBag>
   <keyedReference keyName="uddi-org:types:wsdl"
     keyValue="wsdlSpec"
     tModelKey="uddi:uddi.org:categorisation:types"/>
   <keyedReference keyName="uddi-org:types:specification"
     keyValue="specification"
     tModelKey="uddi:uddi.org:categorisation:types"/>
 </categoryBag
</tModel>
```

UDDI API

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

Inquiry API

→ find_* and get_*Detail

Publisher API

→ save_* and delete_*

Security API

→ get_authToken and discard_authToken

Custody and Ownership Transfer API

Subscription API

Replication API

see http://java.sun.com/xml/jaxr : javax.xml.registry.*

References

Thomas Schmidt
schmidt@informatik.
haw-hamburg.de

- The W3C standards http://www.w3.org/2002/ws/ , in detail:
- SOAP 1.2: http://www.w3.org/TR/soap12/
- WSDL 1.1: http://www.w3.org/TR/wsdl
- WSDL 2.0: http://www.w3.org/TR/wsdl2.0
- UDDI 2.0: http://www.uddi.org/specification.html
- *J. Snell et al.: Programming Web Services with SOAP, O'Reilly, Sebastopol, 2002.*
- E. Cerami: *Web Services Essentials,* O'Reilly, Sebastopol, 2002.
- G. Alonso et. al.: *Web Services*, Springer, Berlin, 2004.