



Web Engineering

Web Services

Prof. Dr. Alexander Knapp

Dr. Nora Koch

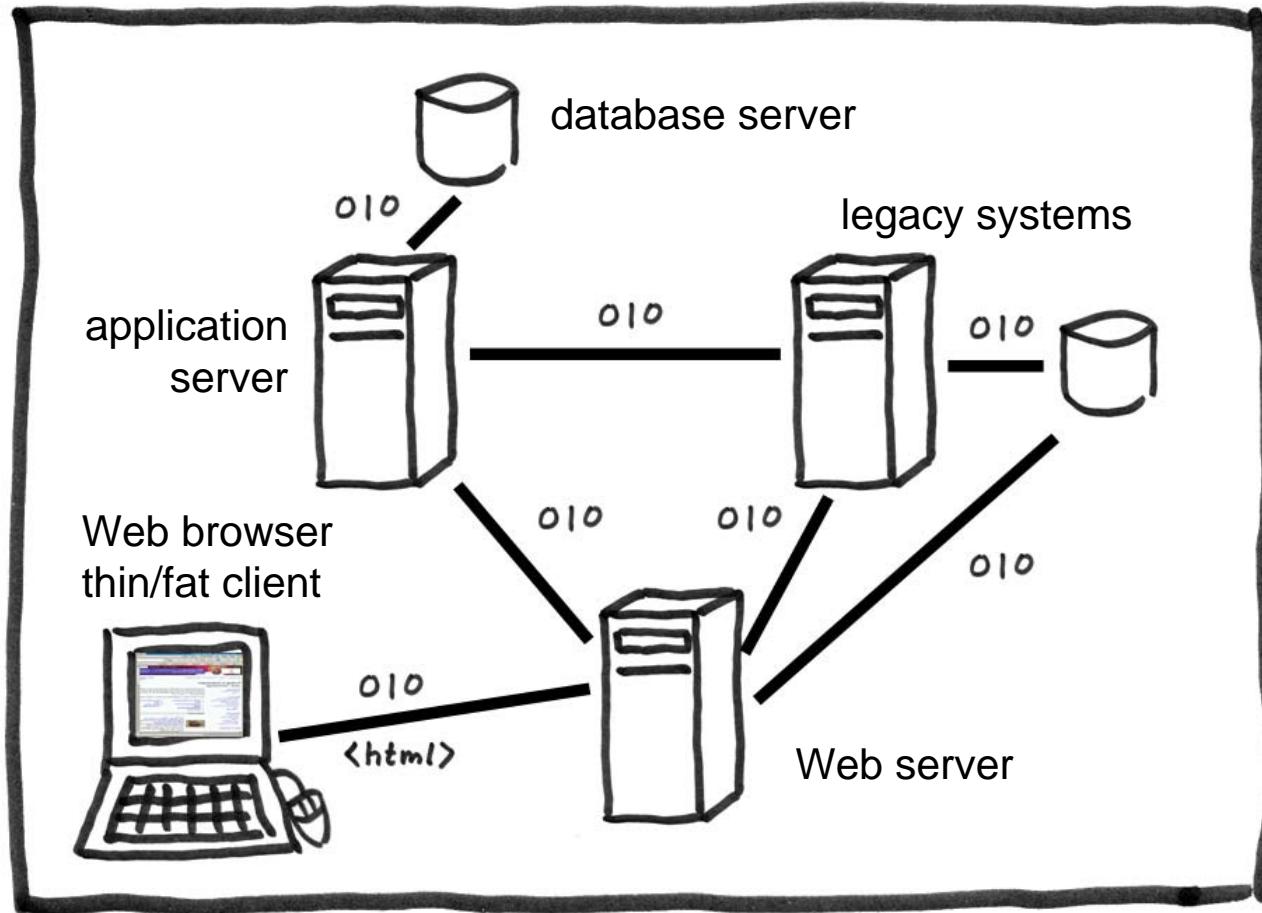
SS 07 (10) – 16.07.07

Ludwig-Maximilians-Universität München

- **IT Integration**

- **Web Services**
 - SOAP
 - WSDL
 - UDDI

- **Service-Oriented Architecture**



Distributed IT: Integration (1)

- **IT integration within a company**
 - Enterprise Application Integration (EAI)
 - **middleware**: technology for integrating different systems
 - Remote Procedure Call (RPC): Sun RPC, RMI
 - object broker: CORBA, COM/DCOM, .NET, EJB
 - transaction management: TP monitor, object monitor
 - message queues/message broker: MQSeries, Java
 - message service: JMS

- **IT integration between IT systems of different companies**
 - global computing
 - loose coupling
 - Business-to-Business (B2B)

- Possible solution: **providing services**
 - Service-Oriented Architecture (SOA)

■ Technical problems with inter-company IT integration

- firewalls
 - block most of TCP/IP ports for security reasons
 - but HTTP (Web) / SMTP (e-mail) are mostly open
- different existing component / middleware technologies
 - CORBA
 - EJB
 - .NET
 - COM/DCOM

■ Possible solution: **Web services**

- firewalls: communication via HTTP using XML messages – SOAP
- different component / middleware technologies: use of Internet standards, only n interfaces instead of n^2 for inter-service communication

Web Services (1)

- **Definition** (<http://www.w3.org/2002/ws/Activity>)

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterised by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. They can be combined in a loosely coupled way in order to achieve complex operations. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services.

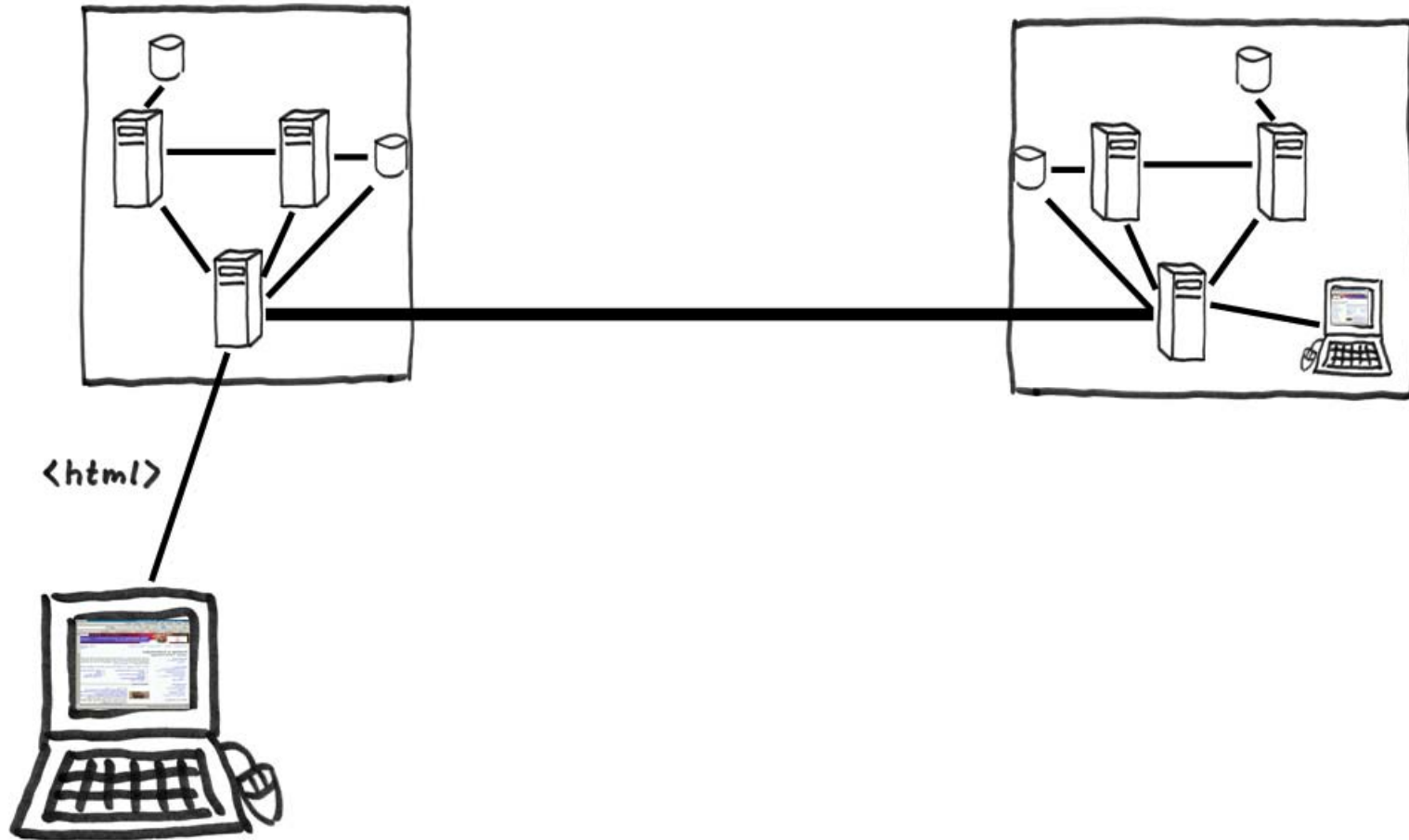
- **Characteristics**

- providing services to other computer programs
- interoperability between software applications running on different computers
- loosely coupled
- machine processable
- use of standards: XML, HTTP, SOAP, WSDL, ...

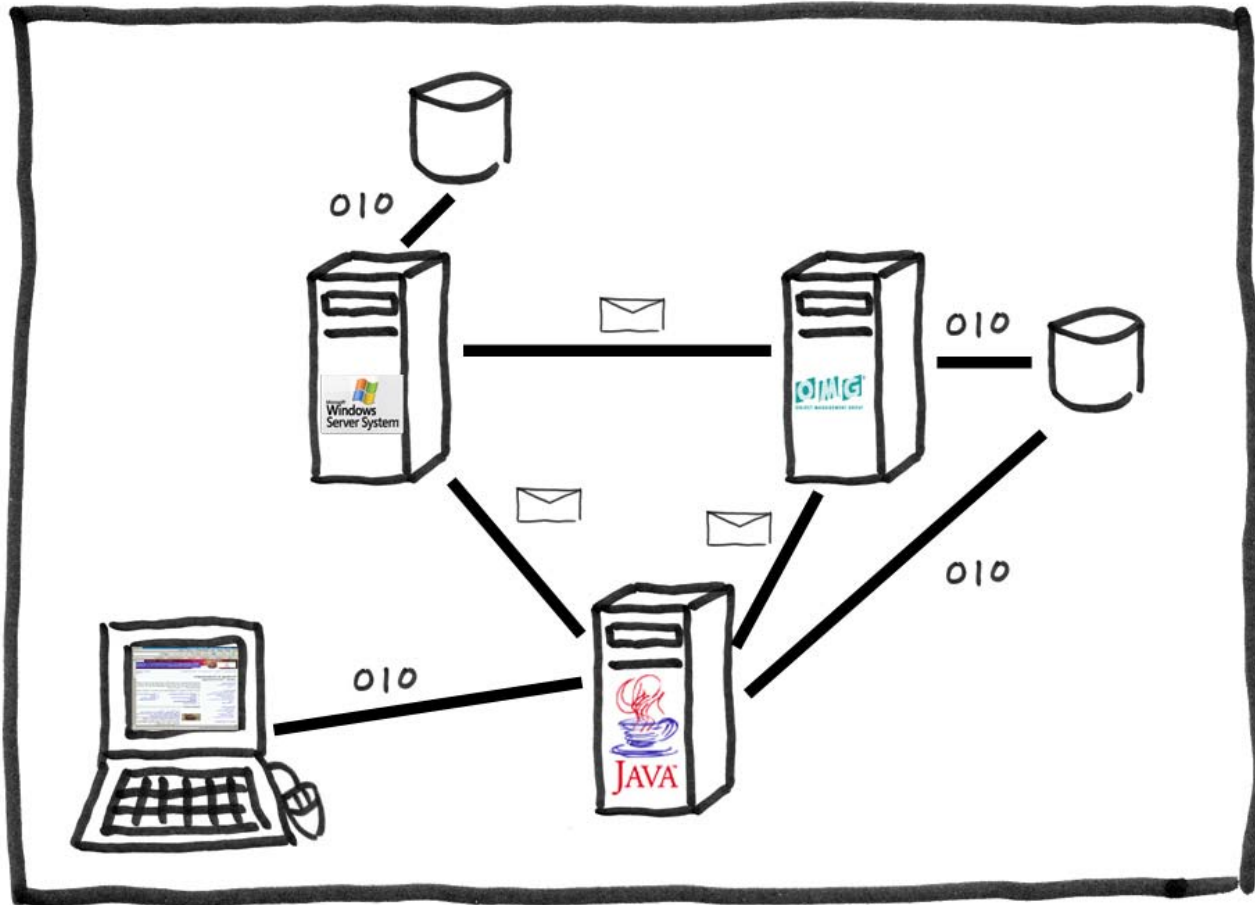
Web Services (2)

- **Major use as integration technology**
 - within a business
 - across businesses
- **Message exchange**
 - messages are exchanged using XML (SOAP)
 - using standard protocols to transport messages (HTML, SMTP, ...)
 - **messaging paradigms**
 - one-way
 - request/response (most common style)
 - solicit/response
 - notification
- **Issues**
 - description
 - discovery
 - composition, coordination
 - transaction, security

Web Services: Integration Scenarios (1)



Web Services: Integration Scenarios (2)



Web Services: Technological Factors

- **SOAP based on XML**
 - only XML parser needed for parsing the messages
 - transported by HTTP or SMTP

- **Standardisation of message contents and data encoding**
 - WSDL and XML Schema help with that

- **Language bindings**
 - have been developed for almost any programming language
 - easy to provide Web services in any programming language
 - easy to use Web services in any programming language and operating system

- **Web server technology can be reused for Web services**
 - e.g. CGI, servlets

■ SOAP “defines” Web services

- SOAP 1.1 (2000)
 - “Simple Object Access Protocol”, “Service-Oriented Architecture Protocol”
 - namespace URI: `http://schemas.xmlsoap.org/soap/envelope/`
- SOAP 1.2 (2003)
 - simply “SOAP”
 - namespace URI: `http://www.w3.org/2003/05/soap-envelope`

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
              xmlns:ns="urn:example-delayed-quotes" >
  <env:Header>
    <wsa:To>http://example.org/StockExchange</wsa:To>
    <wsa:Action>http://example.org/StockExchange/Infos</wsa:Action>
  </env:Header>
  <env:Body>
    <ns:getQuote>
      <ns:symbol>NYSE:IBM</ns:symbol>
    </ns:getQuote>
  </env:Body>
</env:Envelope>
```

```
<Envelope encodingStyle="uri" >
```

```
  <Header encodingStyle="uri" xmlns:ns1="uri" >
```

SOAP header block: optional

encoding style, e.g., <http://www.w3.org/2003/05/soap-encoding>,
can also be used with most body elements

```
<ns1:...>
```

Attributes:

```
  role = next | none | ultimateReceiver | ...
```

identifies which SOAP nodes needs to act on this header information
the role takes the form of a URI

(e.g., <http://www.w3.org/2003/05/soap-envelope/role/next>)

```
  mustUnderstand = true | false
```

it is mandatory that the header information is processed or optional

```
  relay = true | false
```

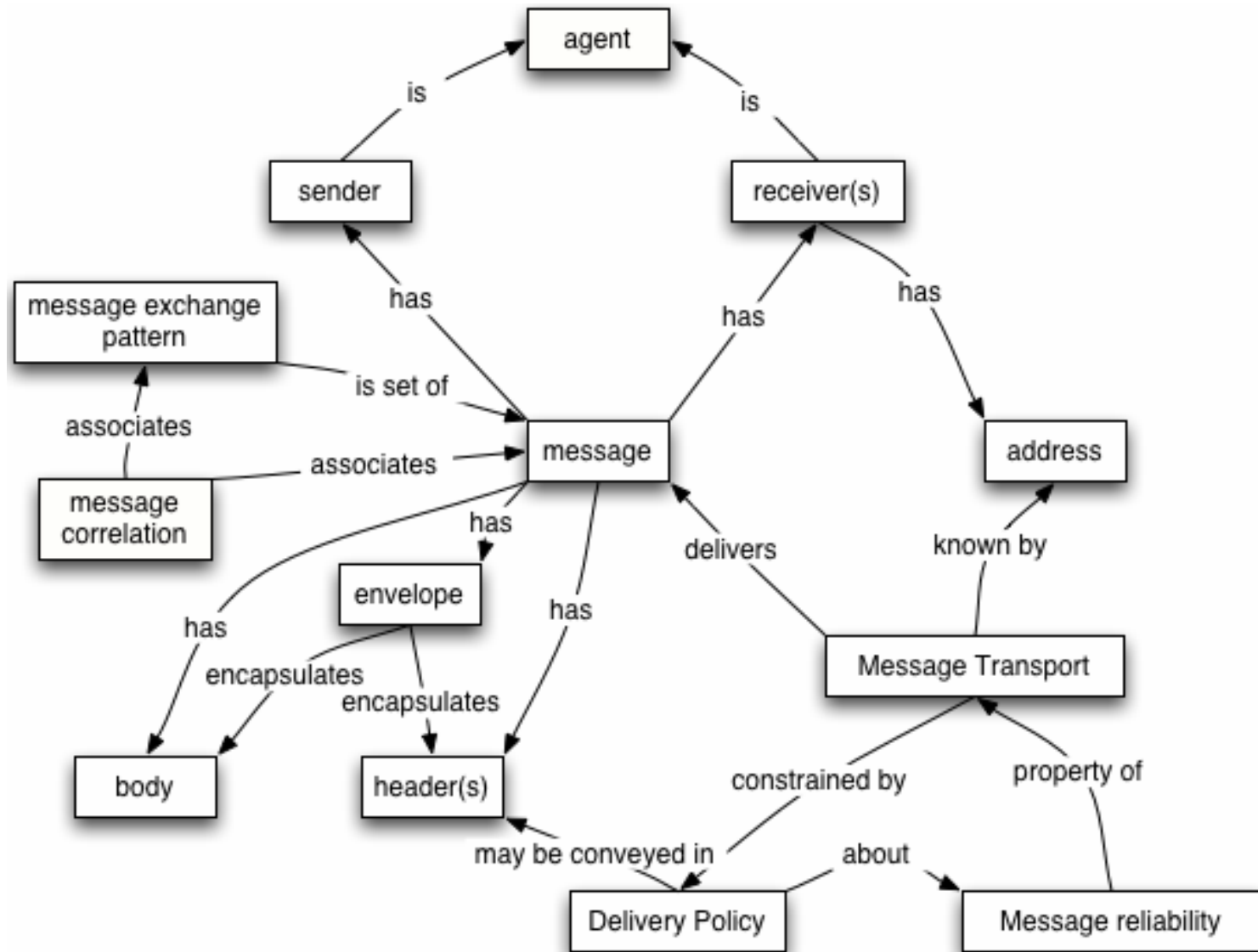
header information must be relayed if not processed

```
<Body>
```

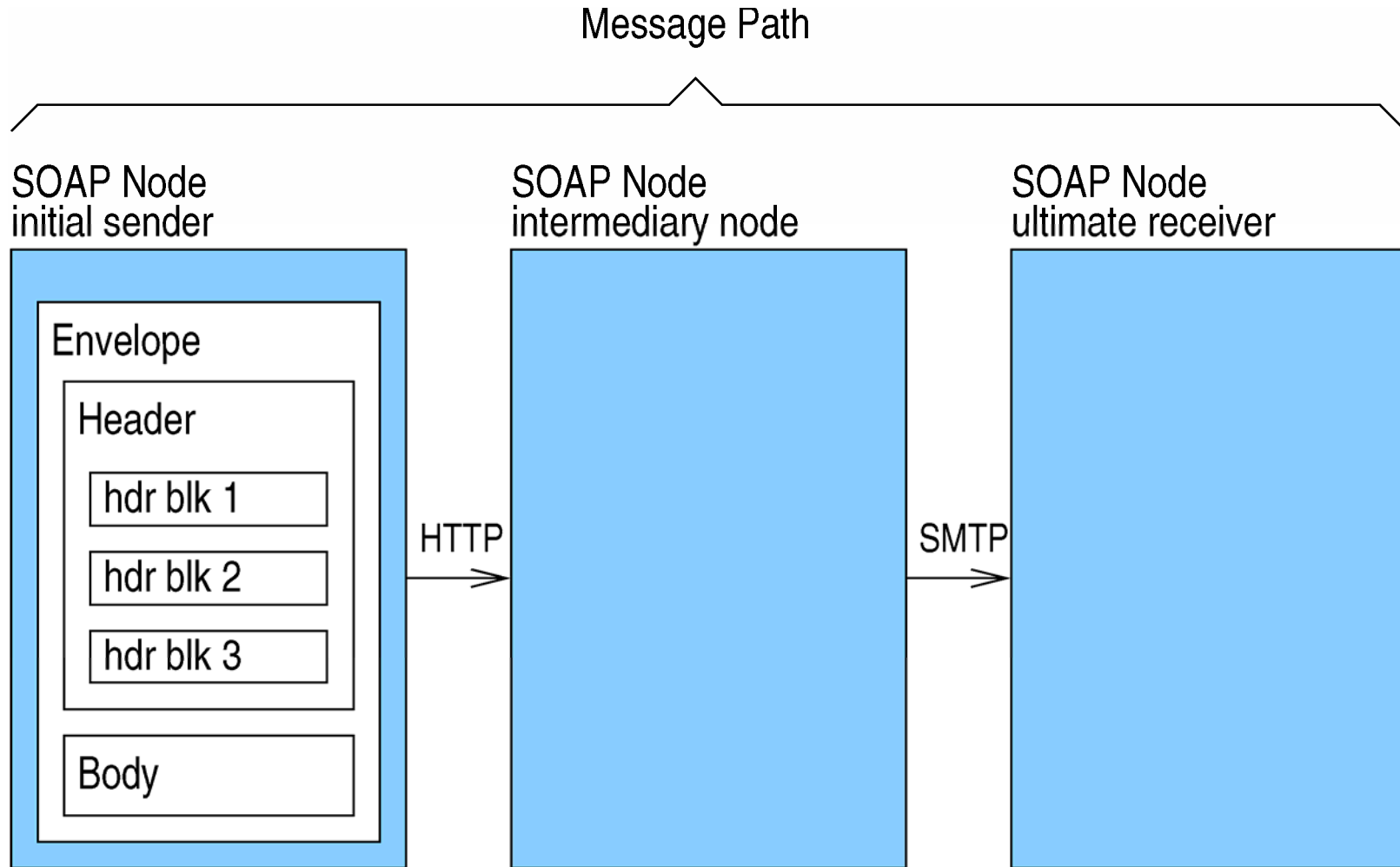
```
  <ns2:...>
```

```
  or <fault>
```

SOAP: Messaging Model



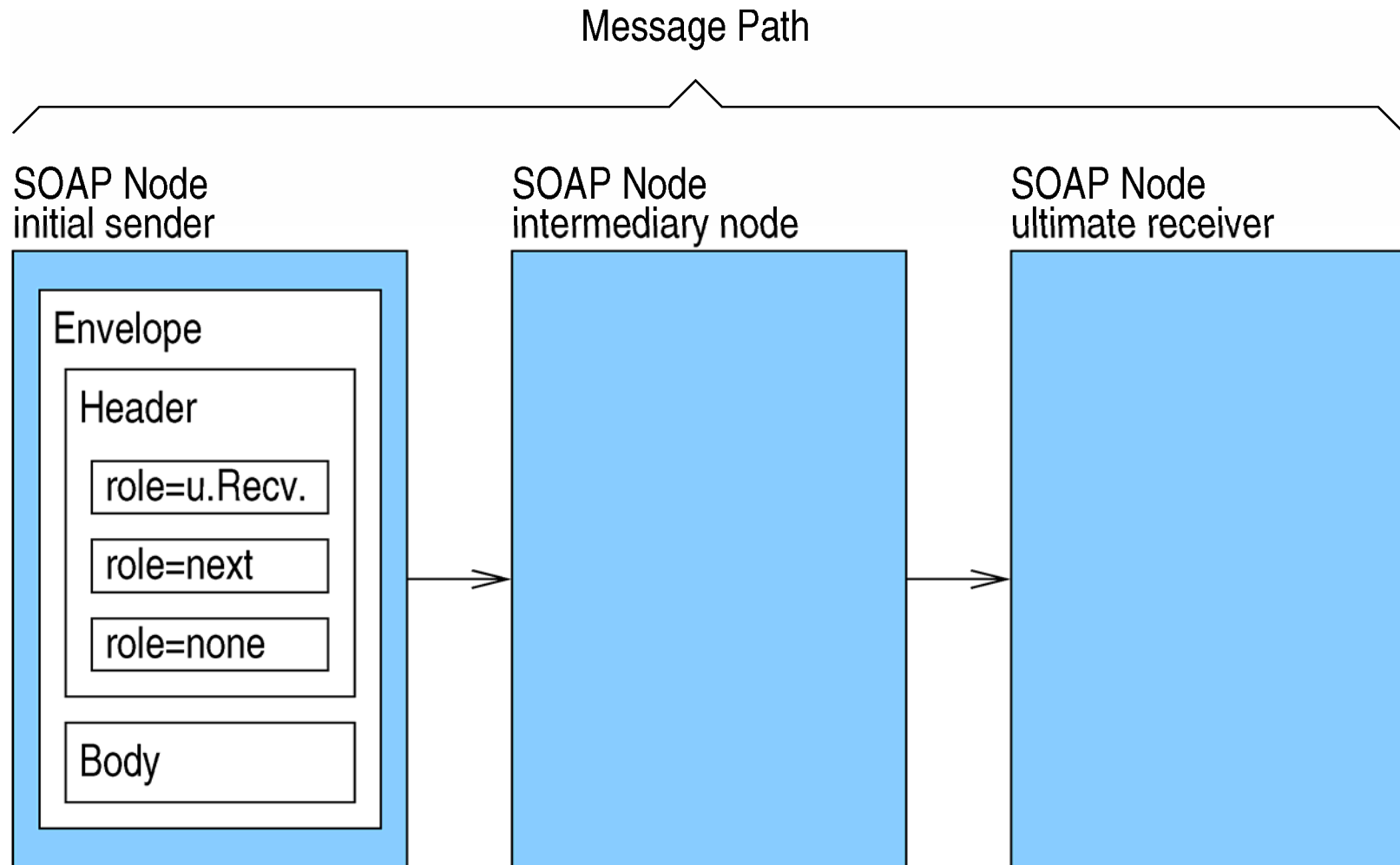
SOAP Messaging Model: Logical View



SOAP: Headers

- **Include processing instructions for the service intermediaries**
 - like signing a message
 - logging a message
- **Routing of messages**
 - who should be dealing with that message?
- **Context- / meta data and transaction management**
 - e.g., name of the person for a travel arrangement
 - transaction identifier / start of transaction / end of transaction / common data
- **Correlation information**
 - ensuring that, e.g., a certain answer belongs to a certain request if the transport layer does not ensure this

SOAP Messaging Model: Roles



SOAP over HTTP

■ HTTP request

- method POST
- content type text/html
- HTTP header contains SOAPAction
 - identifies action/service and allows firewalls to detect and handle SOAP messages

```
SOAPAction "http://electrocommerce.org/abc#MyMessage"
```

```
SOAPAction "axis/EchoString.jws"
```

```
SOAPAction ""
```

means that the URI of the HTTP request is the SOAPAction URI

```
POST /axis/EchoString.jws HTTP/1.1
SOAPAction: ""
Content-type: text/xml
User-Agent: Java/1.5.0_07
Host: 127.0.0.1:8070
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*;
Connection: keep-alive
Content-Length: 190
```

SOAP for RPCs

■ Remote procedure call information

- URI of target object
- method name
- method parameters

$r\ op(in\ p_1, \dots, in/out\ q_1, \dots, out\ r_1, \dots)$

Request

```
<soap:Body>
  <op>
    <p1>v1</p1>
    ...
    <q1>y1</q1>
    ...
  </op>
</soap:Body>
```

Response

```
<soap:Body>
  <opResponse>
    <opReturn>v</opReturn>
    <q1>v1</q1>
    ...
    <r1>y1</r1>
    ...
  </opResponse>
</soap:Body>
```

Web Service Client: Example (1)

```
public String echo(String string) {
    URL endPoint = "http://localhost:8070/axis/EchoString.jws";
    HttpURLConnection connection =
        (HttpURLConnection) endPoint.openConnection();
    connection.setRequestMethod("POST");
    connection.setRequestProperty("SOAPAction", "\"\"");
    connection.setRequestProperty("Content-type", "text/xml");
    connection.setDoOutput(true);
    connection.connect();
    SoapMessage message = SoapMessage.createCallResponseMessage("echo",
        "string", string);
    sendSoapRequest(connection, message);
    SoapMessage answer = receiveSoapResponse(connection);
    connection.disconnect();
    return answer.getCallResponseMessage().getArgument("echoReturn");
}
```

Web Service Client: Example (2)

```
private SoapMessage receiveSoapResponse(HttpURLConnection connection)
    throws IOException {
    InputStream result = (InputStream) connection.getContent();
    SoapMessage answer = SoapMessage.newInstance(result);
    return answer;
}

private void sendSoapRequest(HttpURLConnection connection,
                             SoapMessage message)
    throws IOException {
    Writer writer =
        new OutputStreamWriter(connection.getOutputStream());
    writer.write(message.asXML());
    writer.close();
}
```

```
public void doPost(HttpServletRequest request,
                  HttpServletResponse response)
    throws IOException, ServletException {
    InputStream stream = request.getInputStream();
    SoapMessage message = SoapMessage.newInstance(stream);
    CallResponseMessage payload = message.getCallResponseMessage();
    if ("echo".equals(payload.getMessageName())) {
        String input = payload.getArgument("string");
        EchoString echo = new EchoString();
        String output = echo.echo(input);
        SoapMessage result =
            SoapMessage.createCallResponseMessage("echoResponse",
                                                  "echoReturn", output);

        response.setContentType("text/xml");
        PrintWriter writer = response.getWriter();
        writer.println(result.asXML());
    }
}
```

Web Service Description

- **For use in a loosely coupled system**
 - Web service description
 - Web service discovery

- **Abstract description**
 - available services
 - which messages are needed for a service
 - description of the messages used
 - description of data types used

- **Concrete description**
 - binding to the message layer (e.g., SOAP)
 - binding to the transport layer (e.g., HTTP)
 - where to find messages

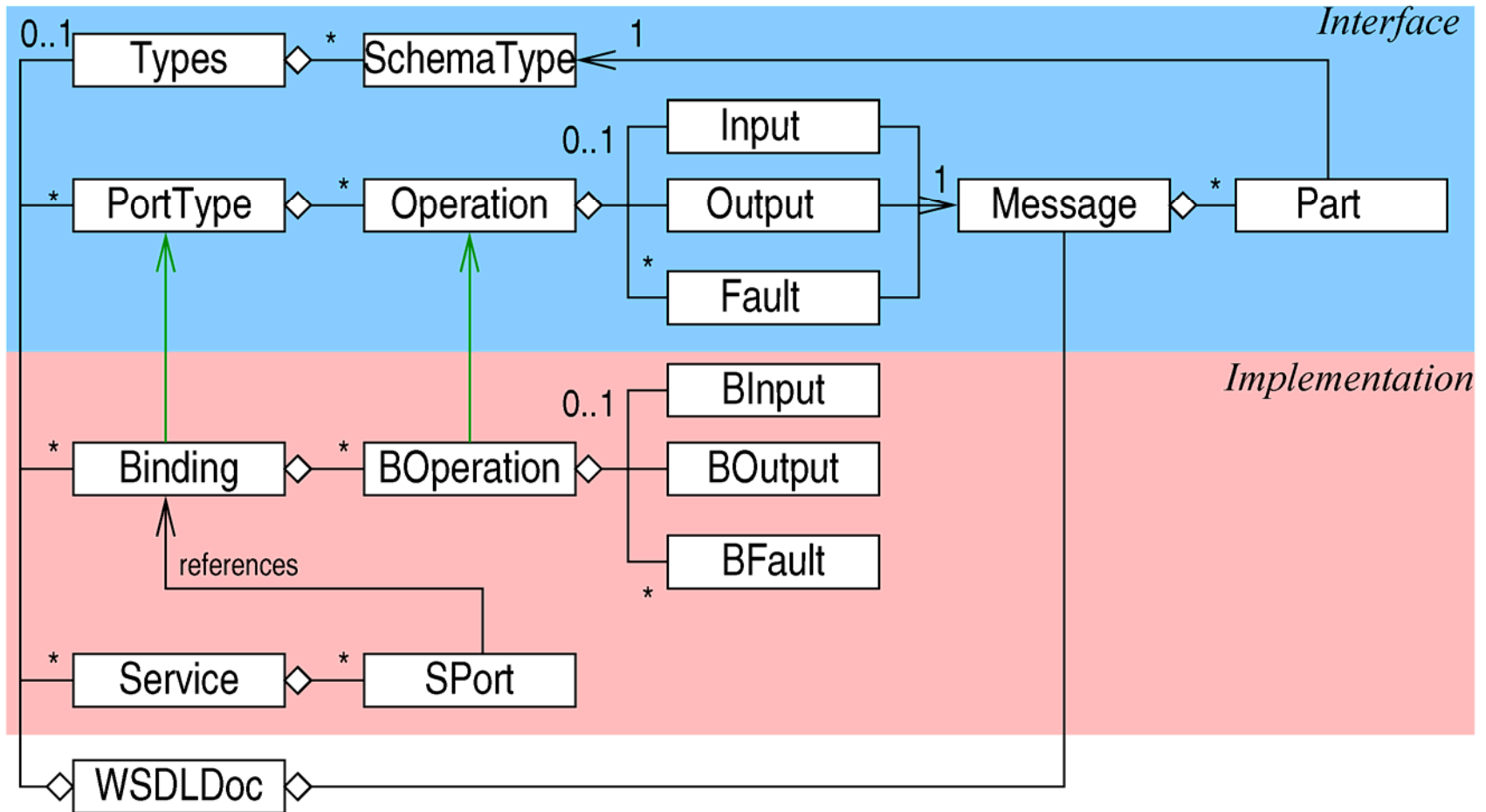
Web Service Description Language (WSDL)

- **WSDL 1.1**
 - de-factor standard, not endorsed by W3C
- **WSDL 2.0**
 - recommendation by W3C
- **Namespaces**
 - WSDL 1.1: `http://schemas.xmlsoap.org/wsdl/`
 - SOAP binding: `http://schemas.xmlsoap.org/wsdl/soap/`
 - XML Schema for user defined datatypes: `http://www.w3.org/2001/XMLSchema`

WSDL: Document Structure (1)

- `types`
 - provides data type definitions used to describe the messages exchanged
- `message`
 - represents an **abstract** definition of the data being transmitted; a message consists of logical parts each of which is associated with a definition within some type system
- `portType`
 - set of **abstract** operations; each operation refers to an input message and output messages
- `binding`
 - specifies **concrete** protocol and data format specifications for the operations and messages defined by a particular `portType`
- `service`
 - used to aggregate a set of related ports

WSDL: Document Structure (2)



WSDL: Example (1)

```
<wsdl:definitions
  targetNamespace="http://localhost:8086/axis/EchoString.jws">

  <wsdl:message name="echoResponse">
    <wsdl:part name="echoReturn" type="xsd:string"/>
  </wsdl:message>
  <wsdl:message name="echoRequest">
    <wsdl:part name="str" type="xsd:string"/>
  </wsdl:message>
  <wsdl:portType name="EchoString">
    <wsdl:operation name="echo" parameterOrder="str">
      <wsdl:input message="impl:echoRequest" name="echoRequest"/>
      <wsdl:output message="impl:echoResponse" name="echoResponse"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="EchoStringSoapBinding" type="impl:EchoString">
    <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
  </wsdl:binding>
</wsdl:definitions>
```

WSDL: Example (2)

```
<wsdl:operation name="echo">
  <wsdlsoap:operation soapAction="" />
  <wsdl:input name="echoRequest">
    <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://DefaultNamespace" use="encoded" />
  </wsdl:input>
  <wsdl:output name="echoResponse">
    <wsdlsoap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      namespace="http://localhost:8086/axis/EchoString.jws"
      use="encoded" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="EchoStringService">
  <wsdl:port binding="impl:EchoStringSoapBinding" name="EchoString">
    <wsdlsoap:address
      location="http://localhost:8086/axis/EchoString.jws" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

WSDL: Message Formats

■ Document style

- paradigm: exchange of self-contained documents
- possibly one-way operation
- Web service description to specify document schema

■ Remote Procedure Call (RPC) style

- paradigm: Web service client passes parameters to procedure, receives result (request/response operation)
- message body contains an XML representation of a method call, message parts represent the parameters to the method
- Web service message must include identification for “method”

WSDL: Message Exchange Patterns

■ One way

- the endpoint receives a message

```
<operation name="nmtoken">  
  <input name="nmtoken"? message="qname">  
  <fault name="nmtoken" message="qname">*
```

■ Request-response

- the endpoint receives a message, and sends a correlated message

```
<operation name="nmtoken" parameterOrder="nmtokens"?>*  
  <input name="nmtoken"? message="qname">  
  <output name="nmtoken"? message="qname">  
  <fault name="nmtoken" message="qname">*
```

■ Solicit-response

- the endpoint sends a message, and receives a correlated message

■ Notification

- the endpoint sends a message.

- **Basic idea: Universal Business Registry (UBR)**
 - one global registry
 - companies register themselves with the UBR
 - companies register services they offer
 - Web services, but also other types of services
- **Information stored in a UDDI registry**
 - **White Pages:** list of organisations / business entities with contact information
 - **Yellow Pages:** classification of organisation and services; search by these categories
 - **Green Pages:** description of how a service can be called
- **OASIS specification** <http://www.oasis-open.org/specs>
 - UDDI 1.0 (2000)
 - UDDI 2.0 (2002)
 - UDDI 3.0 (2005)
- **Driving forces**
 - IBM, Microsoft, Ariba
 - IBM, Microsoft, and SAP discontinued their UDDI Business Registry in 2006

UDDI: Usage

- **Design time discovery**

- used at design time to find the appropriate service for a task

- **Run time discovery**

- type of Web services and its endpoint is discovered at run time
- service endpoint may have moved / node failure
- UDDI registry could also change: replicated registries
- common pattern: cache the endpoint of a service and only if the service is no longer available look for the new endpoint in the registry

- **Patterns**

- **browse pattern**
 - look for certain information (find xyz)
- **drill-down pattern**
 - use get xyz to get details on some entity
- **invocation pattern**
 - call the Web service

UDDI: Data Structures

- **BusinessEntity**
 - describes organisations and their relationship (in case of larger organisations)
- **BusinessService**
 - describes a set of (Web) services offered by an organisation
- **BindingTemplate**
 - describes how a business service is implemented
- **TModel (technical model)**
 - used by all other datastructures to store additional information

```
<businessService serviceKey='...' businessKey='...'>
  <name>Global Weather</name>
  <description>IMPLEMENTATION: msnet</description>
  <description>CONTACT EMAIL:...</description>
  <bindingTemplates>
    <bindingTemplate bindingKey='...' serviceKey='...'>
      <description>SOAP binding</description>
      <accessPoint URLType='http'>
        http://www.webservicex.com/globalweather.asmx</accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey='uuid:...'></tModelInstanceInfo> ...
      </tModelInstanceDetails>
    </bindingTemplate>
  </bindingTemplates>
</businessService>
```

UDDI Data Structures: TModel

- **Several overview docs, containing links to service descriptions**
 - can be textual API descriptions for services executed by humans
 - can also contain links to WSDL documents
- **Classification information**
 - different classification schemata are possible
 - e.g. "uddi-org:types"
 - based on location
 - based on business classification
- **Category bags**
 - classification of businesses, services, tModels etc.
 - consist of one or more `keyedReferences` each representing a classification

```
<categoryBag>
  <keyedReference keyName='uddi-org:types' keyValue='wsdlSpec'
    tModelKey='uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4' >
  </keyedReference>
</categoryBag>
```

■ Inquiry API

- find operations
 - business, related businesses, service, binding, tModel
 - can be searched by name, category bag, etc.
- get Operations
 - binding detail, business detail, service detail, tModel detail
 - Is accessed by a unique key

■ Publisher API

- login and logout: get and discard authorisation token
- business: delete, save
- publisher assertions (describes relationship between business entities): add, set, delete, get
- tModel: delete, save
- service: delete, save
- binding: delete, save

■ APIs accessible via SOAP

- **A set of principles for organising software**
 - loose coupling
 - services represent self contained units of logic (one function or a set of functions) which are relatively independent (reusability)
 - service description
 - encapsulation (autonomy and abstraction)
 - compositionality
 - statelessness
 - discoverability
 - publish/discover/bind

- Not restricted to the use of Web services
 - OSGI services
 - Grid services

- Additional for Web services
 - based on open standards
 - vendor neutrality, vendor diversity

SOA: Service Invocation vs. Function Calls



Service invocation	Function call
coarse grained	fine grained
across processes, computers, networks	within the same process
loosely coupled	tightly coupled
may fail	function is always available
takes time	takes almost no time
several invocation may form a dialog	focus on single calls
complex data or documents as parameters	simple data as parameters

SOA: Evolution

- **Document types**
 - standards, specification, extensions

- **Standardisation bodies**
 - W3C, Oasis, Web Service Interoperability Organisation (WS-I)

- **Vendor support**
 - Microsoft, IBM, Sun, ...

- **Community support**
 - Apache foundation (e.g., Tomcat, Axis), ...

SOA: Web Services

- **Web services are a revolution.**
 - “The best since the invention of sliced bread.”
 - change the way businesses operate
- **Web services are an evolution.**
 - “old wine in new skins”
 - same as already existing component technologies: EJB, CORBA, .NET, ...
 - SOAP = RPC
 - WSDL = IDL
 - SOAP message oriented
 - mostly request-response pattern is used
 - HTTP limiting factor
- Shift in thinking
 - RPC and message-oriented (asynchronous) communication
 - P2P instead of Client/Server
 - language support for dynamic discovery and binding, asynchronous message exchange, dynamic reconfiguration of service networks

References

- Thomas Erl. *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall, 2005.
- Jeffrey C. Jackson. *Web Technologies*. Pearson Prentice Hall, Upper Saddle River, 2007.
- David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, David Orchard. *Web Services Architecture*. W3C Working Group Note 11, 2004.
<http://www.w3.org/TR/ws-arch/>