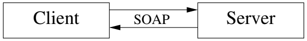
Part V

Web Services

Web Services

- services which communicate with clients based on messages in XML format SOAP
- description of web service in XML format WSDL (Web Services Description Language)
- search desired service with XML format UDDI (Universal Description, Discovery, and Integration)
- potential: leading integration infrastructure (“least common denominator”)

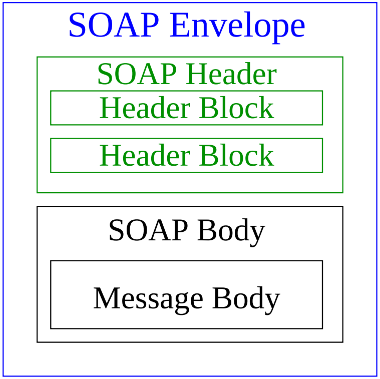


Web−Service Protocol Stack

Discovery	UDDI
Description	WSDL
Packing	SOAP
Transmission	HTTP, SMTP, FTP

1. SOAP

- XML format for transmitting web-service calls and their answers



Details about SOAP

- on error: special fault answer ($\hat{=}$ exception)
- header: optional access control, transaction processing,. . .
- web services are typically stateless
- thus: state (e.g. session id) as parameter (if needed)
- clumsy XML handling is hidden by proxy classes
- implementations: e.g. Apache Axis ($\rightarrow$ Tomcat), JAX-WS

Example: SOAP Request

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
  xmlns:xsi="http//www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getWeather
       xmlns:ns1="urn:examples:weatherservice"
       SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding/">
       <zipcode xsi:type="xsd:string">48149</zipcode>
    </ns1:getWeather>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(based on: E. Cerami: Web Services, O'Reilly, 2002)

Example: SOAP Answer

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/"
  xmlns:xsi="http//www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
     <ns1:getWeatherResponse
       xmlns:ns1="urn:examples:weatherservice"
       SOAP-ENV:encodingStyle="http://www.w3.org/2001/09/soap-encoding/">
       <return xsi:type="xsd:int">23</return>
    </ns1:getWeatherResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Message Formats

- document: SOAP body is an arbitrary XML document (default)
- rpc: SOAP body contains method name and parameters
- literal: data structured according to XML schema (XSD)(default)
- encoded: uses predefined XML tags for usual basic types (e.g. int, double) as well as arrays and structs
- typical combinations: document/literal (recommended) and rpc/encoded
- no distributed object model (in contrast to e.g. CORBA)

2. WSDL

- XML format for describing web services

<definitions>	
<types>	data types
<messages>	messages
<porttype>	operations
<binding>	transmission protocols
<service>	address

Example: WSDL Description of the Wheather Web Service

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="WeatherService"
   targetNamespace="http://www.ecerami.com/wsdl/WeatherService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/WeatherService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="getWeatherRequest">
      <part name="zipcode" type="xsd:string"/>
   </message>
   <message name="getWeatherResponse">
      <part name="temperature" type="xsd:int"/>
   </message>

   <portType name="Weather_PortType">
      <operation name="getWeather">
         <input message="tns:getWeatherRequest"/>
         <output message="tns:getWeatherResponse"/>
      </operation>
   </portType>
```

WSDL Description of the Wheather Web Service (continued)

```
  <binding name="Weather_Binding" type="tns:Weather_PortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="getWeather">
      <soap:operation soapAction=""/>
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice"  use="encoded"/>
      </input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:weatherservice"  use="encoded"/>
      </output>
    </operation>
  </binding>

  <service name="Weather_Service">
    <documentation>WSDL File for Weather Service</documentation>
    <port binding="tns:Weather_Binding" name="Weather_Port">
      <soap:address location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

3. Web Services with .NET and WCF

- Windows Communication Foundation (WCF):
  framework for service-oriented communication
- messages exchanged between endpoints
- endpoint: Address, Binding, Contract (ABC)
- contracts describe behaviour (service c.) and structure (data c.)
- WCF supports different bindings (transport protocol, encoding),
  e.g. HTTP+SOAP for web services

Service Contract

- Service contract: interface describing service operations
- annotations `[ServiceContract], [OperationContract]`
- structure: `[DataContract], [DataMember]`

```
namespace AdderWebService.Contracts {
  [ServiceContract]
  public interface IAdder {
    [OperationContract]
    int Add(int value1, int value2);
  }
}
```

Implementation and Hosting

```
namespace AdderWebService.Implementation {
  class Adder :  IAdder {
    public int Add(int value1, int value2) {
      return value1 + value2;
    }
  }
}
```

- implementation of interface in ordinary class
- hosting in arbitrary .NET process
  (web server (!), console application, windows service)

Web Service Client with .NET

- Visual Studio can generate client proxies from WSDL descriptions

```
namespace AdderWebClient {
  public partial class AdderForm : System.Web.UI.Page {
    protected void ButtonCalculate_Click(object sender,
                                          EventArgs e) {
      // Read integer values from text fields
      int v1 = Convert.ToInt32(TextBox1.Text);
      int v2 = Convert.ToInt32(TextBox2.Text);

       // Create adder service proxy
      AdderServiceReference.Adder adderService =
        new AdderServiceReference.AdderClient();
       // Invoke service and display result
      ResultLabel.Text = "Result: " + adderService.Add(v1, v2);}}
}
```

Transformation of .NET Types and XML-Schema Types

.NET Type	XML-Schema Type
System.String	string
System.Boolean	boolean
System.Int16	short
System.Int32	int
System.Int64	long
System.Double	double
System.Byte	unsignedbyte
. . .	. . .

4. Web Services in Java

Example: Wheather Web Service in Java

```
@Stateless
@WebService
public class  WeatherService{
  @WebMethod
  public int  getWeather(String zipcode) {
    return 23;}
}
```

- deployed (e.g. in web container Tomcat)
- annotations `@WebService` and `@WebMethod` of classes and methods, respectively
- precondition: stateless (!) session bean (or servlet)

Additional Annotations

- annotation `@OneWay` for method with `void` result
- class annotations for determining the message format:
  - `@SOAPBinding(style= ..., use = ...)`
  - `style`: `SOAPBinding.Style.RPC` or `SOAPBinding.Style.DOCUMENT`
  - `use`: `SOAPBinding.Use.LITERAL` or `SOAPBinding.Style.ENCODED`

Java Client with Dependency Injection

Example: stateless session bean as client of a web service

```
@Stateless
@Resource(name="service/Adder",
          type ="javax.jws.WebService",
          mappedName="AdderService")
public class AdderClient implements SomeInterface{
  @WebServiceRef(name="java:comp/env/service/Adder")
  Adder service;

  public int add(int x, int y){
    int[] val = {x,y};
    return service.add(val);}
  }
}
```

Example: Session Bean as Web Service

```
package ejb;
import javax.jws.WebService;
...
@PermitAll
@Stateless
@Remote(AdderIF.class)
@WebService(endpointInterface = "ejb.AdderIF",
            name = "AdderIF")
@WebContext(contextRoot="/add4WS", secureWSDLAccess=false)
public class Adder implements AdderIF{

  public int add(int x, int y) throws RemoteException{
    return x+y;}
}
```

Example: Interface of Session Bean

```
package ejb;
import javax.jws.WebService;
...
@WebService
@SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL)
public interface AdderIF extends Remote {

  // rename due to .NET naming conventions
  @WebMethod(operationName="Add")
  public int add(int x, int y) throws RemoteException;
}
```

Example: Java Servlet as Web Service Client

```
package servlets;
import webservicesEJB.*;
...
public class AdderServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    resp.setContentType("text/html");
    int x = Integer.parseInt(req.getParameter("x"));
    int y = Integer.parseInt(req.getParameter("y"));
    int result = 0;
    try {
      AdderService adderService = new AdderService();
      AdderServiceSOAP adder = adderService.getAdderServiceSOAP();
      result = adder.add(x,y);}
    catch (Exception e) {e.printStackTrace();}
    // output of result ...
}}
```

Creating Proxy Classes

- the proxy classes for accessing a web service (including .NET) generated by (e.g.) `wsimport`
- `wsimport` is contained in JAX-WS
- execution via command line:
  1. (under Windows:) Start $\rightarrow$ Execute, then Open: cmd
  2. in the appearing window: move to the desired directory using `cd`
  3. `wsimport -keep` <WSDL-URL>
  4. e.g. `wsimport -keep http://localhost:8080/add4WS/Adder?wsdl`
  5. copy the generated directory into the src directory of the eclipse web-service client projects

5. RESTful Web Services

- Representational State Transfer (REST):
  web service interface directly based on HTTP(S) methods
- serialization of messages typically based on JSON
  (rather than SOAP)

REST Principles

- addressing: each service identified by URL
- representation: result is delivered in requested format,
  if supported by service (JSON, HTML, XML, text, . . .)
- stateless:
  - improves scalability and simplifies load balancing
  - (e.g.) cookies used for passing state information
- operations:
  - (just) create, read, update, delete (CRUD)
    mapped to HTTP methods POST, GET, PUT, DELETE
  - (should be) all idempotent
  - read/GET should not cause changes
  - moreover: HEAD, OPTIONS, CONNECT, TRACE
- hypermedia: allow to navigate between resources

JavaScript Object Notation (JSON)

- data exchange format typically used in RESTful web services
- more compact than SOAP
- a JSON document is valid JavaScript
  (and could be interpreted by JS eval (→ risk))
- (simplified) syntax (in EBNF):

```
JSON ::= null | true | false | Number | "String" |
         [ ( JSON ( , JSON )* )? ] |                        // array
         { ( String : JSON ( , String : JSON )* )? }        // object
Number ::= ...
String ::= ...
```

JSON Example

```
{ "Title": "Enterprise Application Integration",
  "Authors": [ { "FirstName": "Stefan",
                 "LastName": "Conrad"},
               { "FirstName": "Wilhelm",
                 "LastName": "Hasselbring"}]
  "ISBN": 3827415721,
  "Year": 2006
}
```

Example: RESTful Web Service (1/6)

```
// imports ...
@Path("/")
public interface Root {

  @GET
  @Produces("application/json")
  public Map<String, Map<String, URI>> links(@Context UriInfo uriInfo);
}
```

- `@Path`: path, where service can be found
- `@GET`: HTTP method to support
- `@Produces`: MIME type of result

Example: RESTful Web Service (2/6)

```
// imports ...
public class RootImpl implements Root {

  public Map<String, Map<String, URI>> links(UriInfo uriInfo) {
    Map<String, URI> links = new HashMap<>();
    links.put("adder", adderUri(uriInfo));

    Map<String, Map<String, URI>> map = new HashMap<>();
    map.put("links", links);
    return map;
  }

  private URI adderUri(UriInfo uriInfo) {
    return uriInfo.getBaseUriBuild().path(AdderResource.class).build();
  }
}
```

Example: RESTful Web Service (3/6)

```
// imports ...
@Path("add")
public interface AdderResource {

  @GET
  @Produces("application/json")
  public Response redirectToSampleData();

  @POST
  @Consumes("application/json")
  public Response addTwoNumbers(AdderRequest adderRequest);

  @GET
  @Produces({ "application/json", "application/xml" })
  @Path("{first}/{second}")
  public AdderResult addTwoNumbers(
      @PathParam("first") int first,
      @PathParam("second") int second);
}
```

Example: RESTful Web Service (4/6)

```
// imports ...
public class AdderResourceImpl implements AdderResource {

  public Response redirectToSampleData() {
    AdderRequest adderRequest = new AdderRequest(42, 23);
    return Response.seeOther(adderRequestUri(adderRequest)).build();
  }

  private URI adderRequestUri(AdderRequest adderRequest) {
    return UriBuilder.fromResource(AdderResource.class)
        .segment("" + adderRequest.getFirst())
        .segment("" + adderRequest.getSecond()).build();
  }

  public Response addTwoNumbers(AdderRequest adderRequest) {
    Response response = Response.ok(new AdderResult(adderRequest)).build();
    response.getMetadata().add("Location", adderRequestUri(adderRequest));
    return response;
  }

  public AdderResult addTwoNumbers(final int first, final int second) {
    return new AdderResult(first, second);
  }
}
```

Example: RESTful Web Service (5/6)

```
// imports ...
public class AdderRequest {
  private int first;
  private int second;

  public AdderRequest(int first, int second) {
    this.first = first;
    this.second = second;
  }

  public int getFirst() {
    return first;
  }

  public int getSecond() {
    return second;
  }
}
```

Example: RESTful Web Service (6/6)

```
// imports ...
@XmlRootElement // needed to produce xml responses, not for JSON
public class AdderResult {
  private int first;
  private int second;

  public AdderResult(int first, int second) {
    this.first = first;
    this.second = second;}

  public AdderResult(AdderRequest adderRequest) {
    this(adderRequest.getFirst(), adderRequest.getSecond());}

  @XmlElement // needed to produce xml responses, not for JSON
  public int getFirst() {return first;}

  @XmlElement // needed to produce xml responses, not for JSON
  public int getSecond() {return second;}

  @XmlElement // needed to produce xml responses, not for JSON
  public int getResult() {return first + second;}
}
```

Example: Java Client of RESTful Web Service (1/2)

```
// imports ...
@Path("add")
public interface RestAdderClient {

  @POST
  @Consumes("application/json")
  @Produces("application/json")
  public AdderResult add(AdderRequest adderRequest);
}
```

Example: Java Client of RESTful Web Service (2/2)

```
// ...
String BASE_URI = "http://localhost:8080/AdderServiceResteasy/api";

RegisterBuiltin.register(ResteasyProviderFactory.getInstance());

AdderRequest adderRequest = new AdderRequest(41, 1);
RestAdderClient adderClient = ProxyFactory.create(RestAdderClient.class, BASE_URI);
result = adderClient.add(adderRequest).getResult();
// ...
```