Middleware and the Internet

**Middleware today**
- Designed for special purposes (e.g. DCOM) or with „overloaded“ specification (e.g. CORBA)
- Specifying own protocols – integration in real world network?
- Non-performant runtime behaviour
- Security questions: how to deal with firewalls?
- …

**Internet today**
- Web designed for applications to be used by human beings
- Enabling B2B e-commerce and non-automated B2B interactions
- Build upon Internet protocols (HTTP, together with HTML)
- No assumptions made about platforms

**Problem today**
- How to enable automatic application-to-application interaction in the Web?
- E-Marketplaces, business process integration, resource sharing, …
Example: Shopping Service

Solution:
What we need is a Web site that provides a programmatic interface.
What could be possible?

**CORBA Component Model:**
- Building modular, distributed applications
- Ad-hoc creation of complex applications
- Reuse and integration of existing applications/components

**CORBA services, esp. Trading Service**
- Strong mechanism for dynamic binding of components

Service-oriented architecture
- Applications/components are seen as services, characterising their functionality
- Application construction be composing services
- Easy realisation of business processes
- Three roles: service provider, service requestor, service registry
Service Oriented Architecture

- Manage service descriptions and provide search facilities

- Find binding information in service descriptions to locate a service
- Bind to that service to invoke it

- Publish services by advertising service descriptions in the registry
Web Services

What are Web Services?

- Web services is an effort to build a distributed computing platform for the Web
- Web service applications are encapsulated, loosely coupled Web “components” that can bind dynamically to each other
- SUN: “a Web service is a modular piece of code on the Internet that provides one or more business functions, and that can be discovered and used on demand.”

Goals and requirements:

- Enable universal interoperability
- Enable widespread adoption
- Enable dynamic binding (service oriented architecture)
- Support Web environment efficiently
- Base on open, extensible standards
- Assume minimal amount of required infrastructure
- Focus on messages and documents, not on APIs
Web Service Definition

A software component that can be

- **Described** using a service-description language, which
  - is in formal XML notation,
  - covers all the details necessary to interact with the service (message formats for operations, transport protocols and location), and
  - hides the implementation details of the service
- **Published** to a registry of services
- **Discovered** through a standard mechanism
- **Invoked** through a declared API, usually through a network
- **Composed** with other services
  - enabling loosely coupled, component-oriented, cross-technology application implementations.
Web Services Example

Often-cited examples of a Web Service:

- Stock quote service, in which the request asks for the current price of a specified stock, and the response gives the stock price. This is one of the simplest forms of a Web service in that the request is filled almost immediately, with the request and response being parts of the same method call.

- Instead of including credit card transaction processing functions in an online retail application, an application can access a Web service that provides those functions.
Web Services Framework

Publish, Find, Use Services:

*Universal Description, Discovery and Integration (UDDI)*

Formal Service Descriptions:

*Web Services Description Language (WSDL)*

Service Interactions:

*Simple Object Access Protocol (SOAP)*

Universal Data Format: XML

Ubiquitous Communications: TCP/IP, HTTP

What allows us to find these descriptions: Discovery of services.

What describes what goes on the wire: Description languages.

What goes “on the wire”: Formats and protocols.
SOAP, WSDL and UDDI

**Discovery**
- WSDL URIs
- Publish WSDL URLs

**Description**
- WSDL
- Proxy

**Interaction**
- SOAP request
- SOAP response

**HTTP**
- Automatic generation of WSDL from server code
- Generate proxy, service location stored in WSDL
- Publish service description
- Client can locate the service querying UDDI
- Stubs can be generated from WSDL automatically
- Client invokes service (almost) like a local method
Web Services and Distributed Objects

**Similarities:**
- Both have some sort of description language
  - Operations, signatures, return types, exceptions
  - Compilers generate client proxy and server skeleton
  - Run-time middleware mediates the client-server interaction
- Both have well-defined network interactions
- Both have a similar mechanism for registering and discovering available components

**Differences:**
- Distributed objects enable stateful computing; the server lifecycle depends on the client-server interaction. Web services - at their basic incarnation - are designed for stateless computing.
- A client object can hold a reference to a server and access the server state through the server’s lifetime. There is no web-service reference mechanism.
- Distributed objects were designed mainly for within an intranet, and were conceived as decentralisation technology. Web services are intended as a technology in support of integration on the web.
Simple Object Access Protocol

SOAP...

• … is an application layer protocol, carried within HTTP messages (also can be used with other application layer protocols, e.g. FTP, SMTP)
• … defines a messaging framework for exchanging structured and typed information (formatted XML data) across the Internet
• … is neutral with respect to operating systems, programming languages, and computing platforms
• … adopts RPC to transfer complete XML documents
• … also allows one-way transfer in a document-oriented approach
• … bridges heterogeneous implementations
SOAP Message Structure

SOAP defines

- An XML envelope for XML messaging
- A HTTP binding for SOAP messaging
- A convention for doing RPC
- An XML serialisation format for structured data

- HTTP headers
  - Pass through web server
  - Specifies global web service handler

- SOAP envelope
  - Names XML schemas in use, encoding style

- SOAP header
  - Metadata about the message
  - Attributes for runtime services

- SOAP body
  - Data, method call, response, faults
The SOAP Envelope

```xml
<SOAP-ENV:Envelope
 xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>

  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>

  ...
</SOAP-ENV:Envelope>
```

- **Additional information**: instructing run-time environment, target application, coordinator, ...
- **Real content**: Method calls, parameters, responses, error codes, ...
SOAP Example Message

HTTP headers
- POST /ocs/contentService HTTP/1.1
- Host: www.wns.com
- ... SOAPAction: "http://wns.com/#searchRequest"

SOAP envelope
- xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
- xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
- xmlns:xsd="http://www.w3.org/1999/XMLSchema"
- SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"

SOAP header
- <t:Transaction xmlns:t="http://wns.com/acl"
  SOAP:mustUnderstand="1">3</t:access>

SOAP body
- <fd:search xmlns:fd="http://wns.com/search">
  <article-id xsi:type="xsd:string">1245</article-id>
</fd:search>
Parts of SOAP Message

POST /ocs/contentService HTTP/1.1
Host: www.wns.com

... SOAPAction: "http://wns.com/#searchRequest"
<SOAP-ENV:Envelope
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
  <SOAP-ENV:Header>
    <t:Transaction xmlns:t="http://wns.com/acl"
      SOAP:mustUnderstand="1">
      3
    </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <fd:search xmlns:fd="http://wns.com/search">
      <article-id xsi:type="xsd:string">1245</article-id>
    </fd:search>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

- Specifying the SOAPAction, i.e. the target application
- Naming the XML schemas used for coding
- Defining the encoding style for data transfer
Parts of SOAP Message

Information “describing” the SOAP request:

- Request belongs to a transaction
- Transaction is assigned the TID 3
- mustUnderstand is set to 1 – instructing the server to either process the request as a transaction or fail to process it

```xml
POST /ocs/contentService HTTP/1.1
Host: www.wns.com
SOAPAction: "http://wns.com/#searchRequest"

<SOAP-ENV:Envelope
    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema"
    SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
>
    <SOAP-ENV:Header>
        <t:Transaction xmlns:t="http://wns.com/acl"
            SOAP:mustUnderstand="1">3</t:Transaction>
    </SOAP-ENV:Header>

    <SOAP-ENV:Body>
        <fd:search xmlns:fd="http://wns.com/search"
            article-id xsi:type="xsd:string">1245</article-id>
    </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```
Parts of SOAP Message

```
POST /ocs/contentService HTTP/1.1
Host: www.wns.com

 Simply a method call search(article-id)
SOAPAction:  "http://wns.com/#searchRequest"

<SOAP-ENV:Envelope
  xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema"
  SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  />

<SOAP-ENV:Header>
  <t:Transaction xmlns:t="http://wns.com/acl"
    SOAP:mustUnderstand="1">3</t:access>
</SOAP-ENV:Header>

<SOAP-ENV:Body>
  <fd:search xmlns:fd="http://wns.com/search">
    <article-id xsi:type="xsd:string">1245</article-id>
  </fd:search>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```
Typical RPC:

```java
String strYear = objVehicle.getVehicleYear("739ADF984");
```

**XML-RPC**

**Request:**

```xml
<getVehicleYear>
  <VIN>739ADF984</VIN>
</getVehicleYear>
```

**Response:**

```xml
<getVehicleYearResponse>1996</getVehicleYearResponse>
```

XML-RPC:
- Encode and bind data structures into XML
- Encode an RPC call
Serialization

SOAP defines a serialisation for formatting data (e.g. parameters in a request) to XML structures:

```java
class VehicleDescription {
    String brand = "Ford";
    int doors = 3;
}
```

Serialiser

```xml
<VehicleDescription>
  <brand type="xsd:string">Ford</brand>
  <doors type="xsd:int">3</doors>
</VehicleDescription>
```
<?xml version="1.0" encoding="UTF-8"?>

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <!-- Header elements go here, and are optional. -->
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <!-- Message or method call elements go here.-->
        <getVehicleYear>
            <VIN>739ADF984</VIN>
        </getVehicleYear>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENV:Body"

    <!-- Response from web service. -->
    <getVehicleYearResponse>
        1996
    </getVehicleYearResponse>

    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Error Response

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Server</faultcode>
<faultstring>Server Error</faultstring>
<detail>
<e:myfaultdetails xmlns:e="http://www.ints.com/cars">
<message>
Server busy. Please try again later.
</message>
<errorcode>1001</errorcode>
</e:myfaultdetails>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Response contains result formatted in XML, or an error description
SOAP Message Exchange

There are three components that take part in a SOAP application:

- **Client application**: A program/Servlet/... that sends a SOAP request. Wants to use a service.
- **SOAP processor**: A program that can receive SOAP requests and act accordingly (e.g., call a method of the Application Server)
- **Application Server**: A program that supplies the Web service
What do we have to program?

- We won't directly read or write SOAP messages
- Instead, use Java methods that create request and analyse result
- Use a SOAP processor that is actually a Servlet
- Code the client application and the application server
- Your application server does not need anything special
- In fact, your application server does not have to "know" that it is being used as a Web Service
import javax.xml.soap.*;
import javax.xml.messaging.**;

//Create Soap Message
MessageFactory msgFactory = MessageFactory.newInstance();
SOAPMessage soapMsg = msgFactory.createMessage();

//Create Soap Envelope elements
SOAPPart soapPart = soapMsg.getSOAPPart();
SOAPEnvelope soapEnv = soapPart/getEnvelope();
SOAPHeader soapHeader = soapEnv.getHeader();
SOAPBody soapBody = soapEnv.getBody();
soapHeader.detachNode(); //SOAP Header is optional
Building the SOAP Message

//Create the request element Name
Name nameRequest = soapEnv.createName("getVehicleYear", "m”, "http://www.ints.com/cars");

//Add a request element to the SoapBody
SOAPBodyElement soapRequestElement = soapBody.addBodyElement(nameRequest);

//Set the request element's value
soapRequestElement.addTextNode("739ADF984");
Sending a SOAP Message

//Create Soap Connection
SOAPConnectionFactory scFactory =
    SOAPConnectionFactory.newInstance();
SOAPConnection sc = scFactory.createConnection();

//Create URLEndpoint and send Soap Message
URLEndpoint endPoint = new
    URLEndpoint("http://localhost/MyApp/SoapServlet");
SOAPMessage response = sc.call(soapMsg, endPoint);
sc.close();