

Data-centric XML

SOAP

What is SOAP

- <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>
 - 1.2: <http://www.w3.org/TR/soap12-part0/>, -part1, -part2
- “lightweight protocol for exchange of information in a decentralized, distributed environment”
- three parts:
 - envelope structure for defining messages
 - set of encoding rules for encoding application-specific data types
 - convention for doing RPC
- Current version 1.2 (W3C XML Protocols group)
 - Version 1.1 widely used (Don Box, DevelopMentor)
- Supports potentially multiple transport mechanisms
 - only HTTP specified

SOAP Companions

- **WSDL: Web Services Definition Language**
 - <http://www.w3.org/TR/wsdl>
 - interface definition: endpoints, operations, and messages
 - XML vocabulary for describing services (SOAP, HTTP, MIME)
- **UDDI: Universal Description Discovery and Integration**
 - <http://uddi.org/pubs/uddi-v3.0.1-20031014.htm>
 - repository service for discovery of services
 - XML schemas for various information models
 - not covered here
- **Web Services Interoperability**
- **Web Services Security**

Usage Scenarios

- Fire-and-forget to single/multiple receiver (notifications)
- Request/response asynchronous communication
- RPC
- Request with acknowledgement
- Request with encrypted payload
- (Multiple) Third party intermediary
- Multiple asynchronous responses
- Caching
- Routing
- ...

SOAP Non-Goals

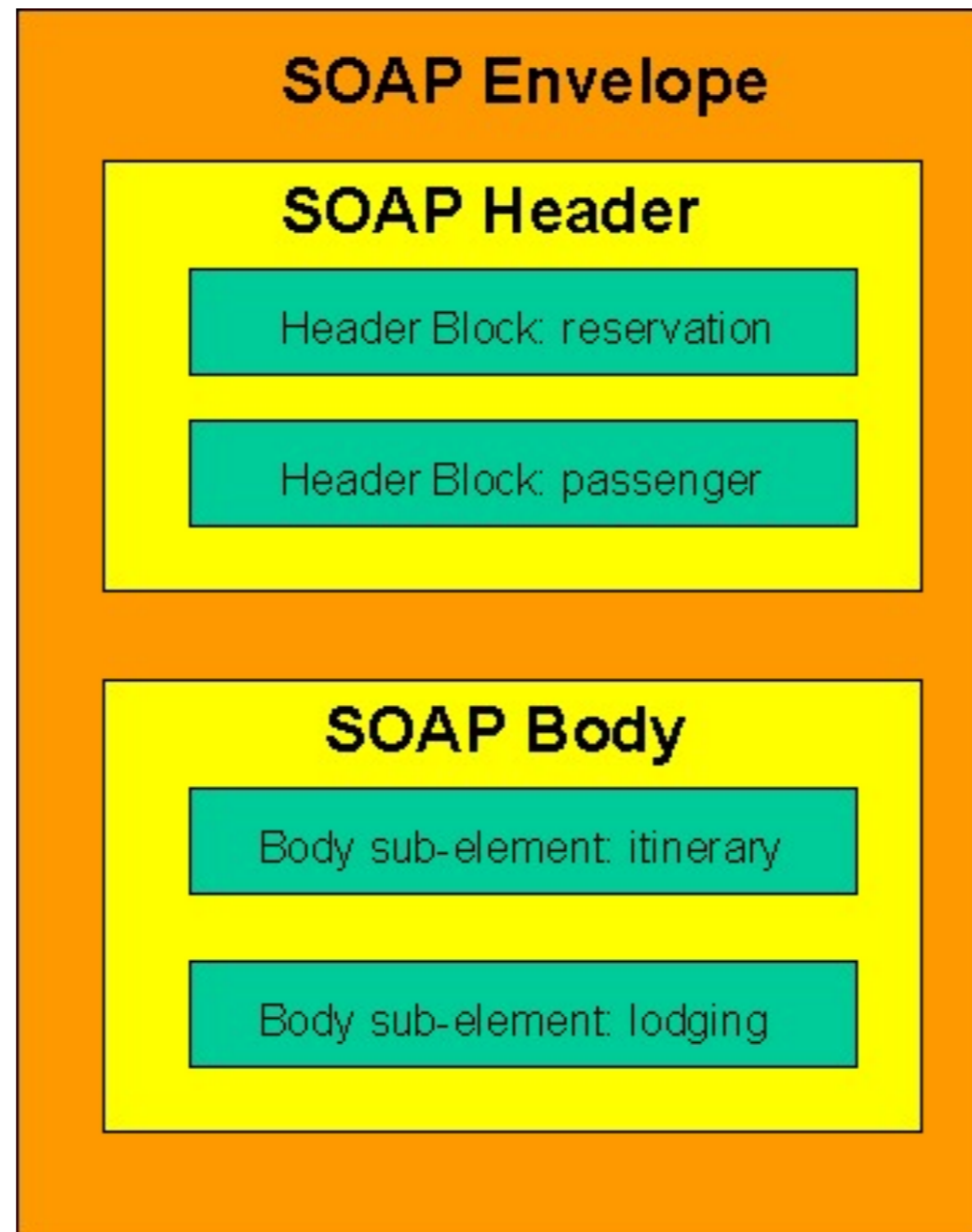
- Distributed Garbage Collection
- Boxcarring/batching of messages
- objects-by-reference
- activation

Namespaces

- `xmlns:ENV="http://schemas.xmlsoap.org/soap/envelope/"`
 - SOAP 1.2: `http://www.w3.org/2003/05/soap-envelope`
- `xmlns:ENC="http://schemas.xmlsoap.org/soap/encoding/"`
 - SOAP 1.2: `http://www.w3.org/2003/05/soap-encoding`

Messages

- Emitted by *sender*, targeted at *ultimate receiver*, through *intermediaries*



Envelope

```
<?xml version='1.0' ?>  
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">  
  <env:Header> ... </env:Header>  
  <env:Body> ... </env:Body>  
</env:Envelope>
```


Header

```
<env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
    env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
    env:mustUnderstand="true">
    <n:name>Áke Jógvan Øyvind</n:name>
  </n:passenger>
</env:Header>
```

Body

```
<env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2001-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
      <p:departureDate>2001-12-20</p:departureDate>
      <p:departureTime>mid-morning</p:departureTime>
      <p:seatPreference/>
    </p:return>
  </p:itinerary>
</env:Body>
```

Roles

- Target roles specified in ENV:role of headers
 - SOAP 1.1: ENV:actor
- Possible roles
 - "http://www.w3.org/2003/05/soap-envelope/role/next"
 - "http://www.w3.org/2003/05/soap-envelope/role/none"
 - "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"
- ENV:mustUnderstand specifies whether target is required to process the header
- Intermediary may drop, modify, or add headers
 - headers targeted at intermediary are dropped, unless ENV:relay is true (and unless processing of the header reinserts it)

Body Processing

- Targeted at ultimate receiver
- Structure completely application-defined, except for faults
- Structure of body specified in ENV:encodingStyle
 - 1.1: typically specified on ENV:Envelope
 - 1.2: specified on header block, child of body that is not a Fault element, child of a Detail element, or descendants thereof

Faults

- ENV:Fault
- Mandatory children
 - ENV:Code
 - ENV:Value, with values Env:{VersionMismatch, MustUnderstand, DataEncodingUnknown, Sender, Receiver}
 - Additional Header elements may provide more detail
 - Optional ENV:Subcode child
 - ENV:Reason, with optional ENV:Text children
- Optional children
 - ENV:Node, with URI content
 - ENV:Role, with URI content
 - ENV:Detail, with arbitrary attributes and child elements

SOAP Data Model

- Optional part of SOAP
- Intended for RPC, accompanied by encoding style
- Data are represented as directed edge-labeled graph of nodes
 - outbound edges may be distinguished either by label or position
 - labels are QNames
 - nodes may be referenced just once, or have multiple references
- Structs are nodes with all outbound edges distinguished by label
- Arrays are nodes with all outbound edges distinguished by position

SOAP Encodings

- Support for multiple encoding styles
- Only one standard encoding
 - "<http://www.w3.org/2003/05/soap-encoding>"
- Graph edges represented by elements
 - label of the edge is element name
 - ENC:ref="id" (type IDREF) can be used for multiple references
 - ENC:id="id" (type ID) specifies target of the reference
- Simple values are encoded in element content
- Compound values are encoded through child elements
 - outgoing labelled edges again denoted through child element name

SOAP Encodings (2)

- ENC:nodeType can be used to specify one of “simple”, “struct”, “array”
- Arrays: Element names of child elements are not significant
 - ENC:itemType specifies the array element type
 - ENC:arraySize allows the specification of multi-dimensional arrays
- Integration with XML Schema
 - xsi:type can specify the node type
 - if not specified, ENC:itemType may apply
 - otherwise, graph node has unspecified type

Using SOAP for RPC

- Multiple bindings to protocols, only HTTP binding specified
- RPC usage must identify RPC resource
 - e.g. encoding of identified resources in URL query parameters
 - pure GET operations may not include any SOAP envelope
- RPC Invocation is a single struct, with outbound edges for parameters
 - name of struct node equals operation name
- RPC Response is a single struct, whose name is irrelevant
- Faults should be used to indicate errors in the RPC invocation

WSDL

- <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- Definition of *services* as a collection of *ports* (network endpoints)
- Definition of *messages*: data that are exchanged
- Definition of *port types*: abstract collections of operations
- Definition of *bindings*: specification of protocol and data format for a type

Example

```
<?xml version="1.0"?>
<definitions name="StockQuote" targetNamespace="http://example.com/
stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>...</types>
  <message>...</message>
  <portType>...</portType>
  <binding>...</binding>
  <service>...</service>
</definitions>
```

Example: Types

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

Example: Messages

```
<message name="GetLastTradePriceInput">  
  <part name="body" element="xsd1:TradePriceRequest"/>  
</message>
```

```
<message name="GetLastTradePriceOutput">  
  <part name="body" element="xsd1:TradePrice"/>  
</message>
```

Example: Port Types

```
<portType name="StockQuotePortType">  
  <operation name="GetLastTradePrice">  
    <input message="tns:GetLastTradePriceInput"/>  
    <output message="tns:GetLastTradePriceOutput"/>  
  </operation>  
</portType>
```

Example: Bindings

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Example: Services

```
<service name="StockQuoteService">  
  <documentation>My first service</documentation>  
  <port name="StockQuotePort" binding="tns:StockQuoteBinding">  
    <soap:address location="http://example.com/stockquote"/>  
  </port>  
</service>
```


WSDL Document Structure

- Specification uses informal grammar to describe document structure, e.g.

```
<definitions .... >
```

```
  <types>
```

```
    <xsd:schema .... />*
```

```
  </types>
```

```
</definitions>
```

means: The “definitions” element contains a “types” element, which can contain multiple “xsd:schema” elements

WSDL Definitions

- **Types:** List of schema types
- **Messages:** Consist of multiple parts
 - each part specified either as an element or a type
 - actual (XML) representation depends on the binding
- **port types:** List of operations
 - Each operation has optional input, output, faults
- **Bindings:** Lists of inputs, outputs, faults, mixed with extension elements
- **Service:** List of ports
 - Each port has name, binding, optional extensions

WSDL SOAP Binding

- Extensions soap:binding, soap:operation, soap:body, soap:header, soap:headerFault, soap:address, ...
- soap:binding specifies transport= (default HTTP) and style= (“document” | “rpc”)
- soap:operation specifies soapAction= target URL (SOAPAction: header), and optional style=
- soap:body specifies parts= included in the body, use= type of encoding (“literal” | “encoded”), optional encodingStyle=, and optional namespace=
- soap:address specifies location= of HTTP server

SOAP Styles

```
<message name="fooRequest">  
  <part name="param1" type="xsd:int"/> <!-- encoded requires to use type= -->  
</message>
```

```
<portType name="PT">  
  <operation name="foo">  
    <input message="fooRequest"/>  
  </operation>  
</portType>
```

RPC/encoded

```
<ENV:Envelope ...>  
  <ENV:Body>  
    <foo>  
      <param1 xsi:type="xsd:int">5</param1>  
    </foo>  
  </ENV:Body>  
</ENV:Envelope>
```

RPC/literal

```
<ENV:Envelope ...>  
  <ENV:Body>  
    <foo>  
      <param1>5</param1>  
    </foo>  
  </ENV:Body>  
</ENV:Envelope>
```

Document/encoded

- not specified; meaningless

Document/literal

```
<types>
  <schema>
    <element name="IntValue" type="xsd:int"/>
  </schema>
</types>

<message name="fooRequest">
  <part name="param" element="IntValue"/>
  <!-- type= is discouraged for document-style operations in WS-I -->
</message>

<portType name="PT">
  <operation name="foo">
    <input message="fooRequest"/>
  </operation>
</portType>
```


Document/literal

```
<ENV:Envelope ...>
```

```
  <ENV:Body>
```

```
    <IntValue>5</IntValue>
```

```
  </ENV:Body>
```

```
</ENV:Envelope>
```

- with `<part name="param" type="xsd:int"/>`, the Body child would become `<param>5</param>`
 - confusing, since there is no element definition for a “param” element

Document/wrapped

- WSDL style convention
- Create an XML element for each message, use that in element=for each part
 - allows to identify the operation name from the body

Interoperability

- Sources of non-interoperability
 - Under-specification (optional features, implementation-defined behavior)
 - Non-compliance of implementation
- Web-Services Interoperability Organization aims at improving interoperability
 - through specifications
 - through testing technology
- Multiple profiles of WS-I, depending on application domain

Basic Profile

- Basic Profile Version 1.0a
 - <http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.html>
- Requirements on XML, SOAP, WSDL, and UDDI usage
 - UDDI out of scope for this presentation

Basic Profile Requirements

- An INSTANCE MUST be described by a WSDL 1.1 service description, by a UDDI binding template, or both.
- Conformance claims
 - A MESSAGE MAY contain conformance claims, as specified in the conformance claim schema
 - MESSAGE's conformance claims MUST be carried as SOAP header blocks.
 - A MESSAGE MAY contain conformance claims for more than one profile.
 - A SENDER MUST NOT use the soap:mustUnderstand attribute when sending a SOAP header block containing a conformance claim.

Basic Profile Requirements (2)

- XML Representation

- A RECEIVER MUST accept messages that include the Unicode Byte Order Mark (BOM)
- When a MESSAGE contains a soap:Fault element, that element MUST NOT have element children other than faultcode, faultstring, faultactor and detail.
- When a MESSAGE contains a soap:Fault element its element children MUST be unqualified.
 - Violation of SOAP 1.2?
- A RECEIVER MUST accept fault messages that have any number of elements, including zero, appearing as children of the detail element. Such children can be qualified or unqualified.
- ...

Basic Profile Requirements (3)

- XML Usage
 - A MESSAGE MUST NOT contain a Document Type Declaration.
 - A MESSAGE MUST NOT contain Processing Instructions.
 - A RECEIVER MUST accept messages that contain an XML Declaration.
 - A MESSAGE MUST NOT have any element children of soap:Envelope following the soap:Body element.
 - unspecified in SOAP 1.1
 - A MESSAGE MUST be serialized as either UTF-8 or UTF-16.
 - The media type of a MESSAGE's envelope MUST indicate the correct character encoding, using the charset parameter.
 - A MESSAGE containing a soap:mustUnderstand attribute MUST only use the lexical forms "0" and "1".
 - The children of the soap:Body element in a MESSAGE MUST be namespace qualified.

Basic Profile Requirements (4)

- XML Requirements

- A RECEIVER MUST generate a fault if they encounter a message whose document element has a local name of "Envelope" but a namespace name that is not "http://schemas.xmlsoap.org/soap/envelope/".
- A RECEIVER MUST NOT mandate the use of the xsi:type attribute in messages except as required in order to indicate a derived type (see XML Schema Part 1: Structures, Section 2.6.1).

Basic Profile Requirements (5)

- SOAP Requirements

- A RECEIVER MUST handle messages in such a way that it appears that all checking of mandatory header blocks is performed before any actual processing.
- A RECEIVER MUST generate a "soap:MustUnderstand" fault when a message contains a mandatory header block (i.e., one that has a soap:mustUnderstand attribute with the value "1") targeted at the receiver (via soap:actor) that the receiver does not understand.

Basic Profile Requirements (6)

- HTTP Requirements
 - A MESSAGE SHOULD be sent using HTTP/1.1.
 - A MESSAGE MUST be sent using either HTTP/1.1 or HTTP/1.0.
 - A RECEIVER MUST interpret SOAP messages containing only a soap:Fault element as a Fault.
 - I.e. do not rely on HTTP status to determine fault presence
 - A HTTP request MESSAGE MUST use the HTTP POST method.
 - A MESSAGE MUST NOT use the HTTP Extension Framework (RFC2774).
 - An INSTANCE MUST use a 2xx HTTP status code for responses that indicate a successful outcome of a request.
 - An INSTANCE MUST use a "500 Internal Server Error" HTTP status code if the response message is a SOAP Fault.
 - ...

Basic Profile Requirements (7)

- WSDL requirements
 - A DESCRIPTION using the WSDL namespace (prefixed "wsdl" in this Profile) MUST be valid according to the XML Schema found at "<http://schemas.xmlsoap.org/wsdl/2003-02-11.xsd>".
 - A DESCRIPTION MUST NOT use QName references to elements in namespaces that have been neither imported, nor defined in the referring WSDL document.
 - A wsdl:portType in a DESCRIPTION MUST have operations with distinct values for their name attributes.
 - ...

Basic Profile Requirements (8)

- SOAP Requirements

- A `wsdl:binding` element in a `DESCRIPTION` MUST use WSDL SOAP Binding as defined in WSDL 1.1 Section 3.
- A `wsdl:binding` element in a `DESCRIPTION` MUST specify the HTTP transport protocol with SOAP binding. Specifically, the `transport` attribute of its `soapbind:binding` child MUST have the value `"http://schemas.xmlsoap.org/soap/http"`.
- A `wsdl:binding` in a `DESCRIPTION` MUST use the value of `"literal"` for the `use` attribute in all `soapbind:body`, `soapbind:fault`, `soapbind:header` and `soapbind:headerfault` elements.
- A `wsdl:binding` in a `DESCRIPTION` that contains one or more `soapbind:body`, `soapbind:fault`, `soapbind:header` or `soapbind:headerfault` elements that do not specify the `use` attribute MUST be interpreted as though the value `"literal"` had been specified in each case.
 - WSDL 1.1 spec and schema are inconsistent as to whether `use=` is optional

Basic Profile Requirements (9)

- SOAP Requirements

- A DESCRIPTION SHOULD NOT have more than one wsdl:port with the same value for the location attribute of the soapbind:address element.
- For one-way operations, an INSTANCE MUST NOT return a HTTP response that contains a SOAP envelope. Specifically, the HTTP response entity-body must be empty.
- A CONSUMER MUST ignore a SOAP response carried in a response from a one-way operation.
- A MESSAGE MUST include all soapbind:headers specified on a wsdl:input or wsdl:output of a wsdl:operation of a wsdl:binding that describes it.
 - unspecified in WSDL 1.1

Basic Profile Requirements (10)

- HTTP Requirements

- A HTTP request MESSAGE MUST contain a SOAPAction HTTP header field with a quoted value equal to the value of the soapAction attribute of soapbind:operation, if present in the corresponding WSDL description.
- An INSTANCE MAY require the use of HTTPS.
- If an INSTANCE requires the use of HTTPS, the location attribute of the soapbind:address element in its wsdl:port description MUST be a URI whose scheme is "https"; otherwise it MUST be a URI whose scheme is "http".