SOAP

Richard Hopkins
National e-Science Centre, Edinburgh
February 23 / 24 2005

Information Society

OUTLINE

- **Goals**
  - To understand the structure and meaning of SOAP messages
  - To understand how SOAP messages are standardly used for RPC over HTTP
- **Outline**
  - SOAP architecture
    - What soap is
    - Message structure
    - Processing Model
    - Faults
  - SOAP Mappings
    - Serialisation
    - Bindings
    - RPC

"SOAP" what it is

- **Name**
  - Originally – Simple Object Access Protocol
  - Temporarily – Service Oriented Architecture Protocol ?
  - Now (SOAP 1.2) – Not an acronym
- **Purpose**
  - A extensible protocol to enable the exchange of
    - structured and typed information
    - between peers
    - in a decentralised, distributed environment
- **Status**
  - SOAP 1.2 – http:www.w3.org/TR/soap12-part0
    - W3C recommendation, June 2003
  - **SOAP 1.1** – http://www.w3.org/TR/NOTE-SOAP-20000508
    - W3C submission May 2000 – but that's what people use currently

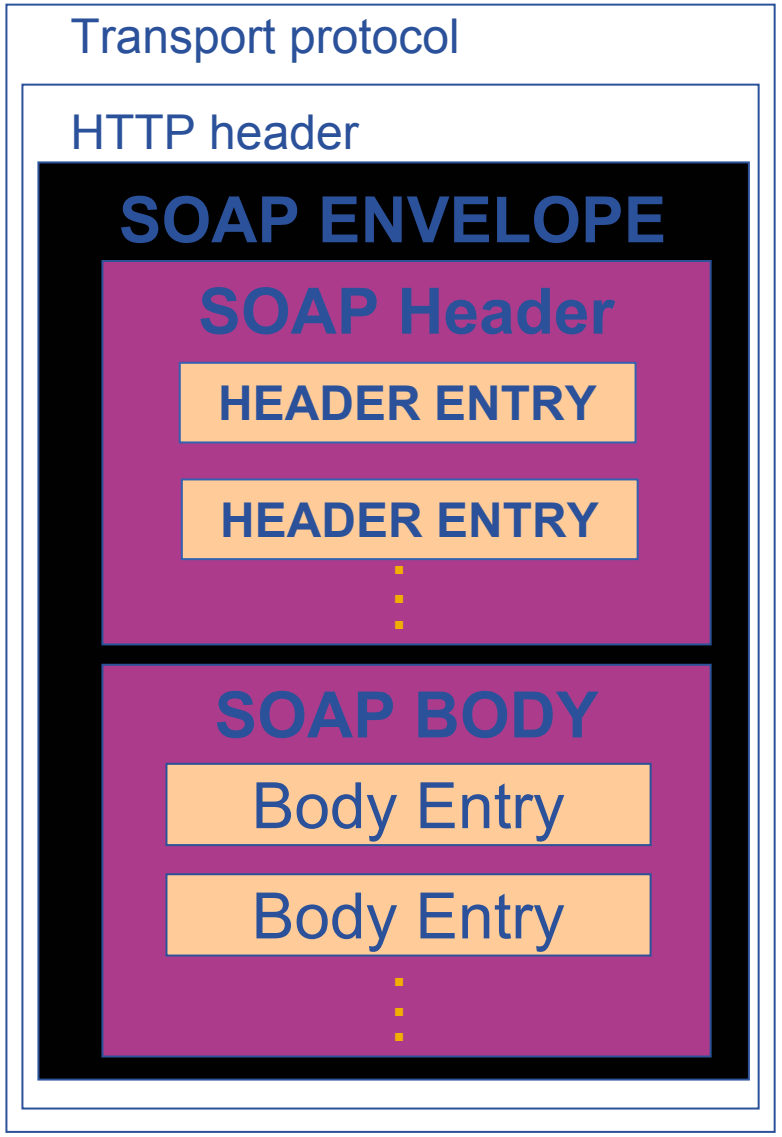
Main Architectural Features

- **XML based (defined as an infoset – assume XML 1.0)**
- **Higher order Protocol –**
  - Built on some underlying protocol - binding
    - Extensibility – can define binding for any underlying protocol
    - Usually HTTP – a specific standard extension
- **Single Message Protocol**
  - Multi-message conversations require a means to associate one message with another
    - Via underlying protocol (e.g. use of same connection)
    - Via the application (specific message-id information as part of the soap message)
- **Multi-stage message processing –**
  - The soap Processing model

Message Structure

Each SOAP message will have:

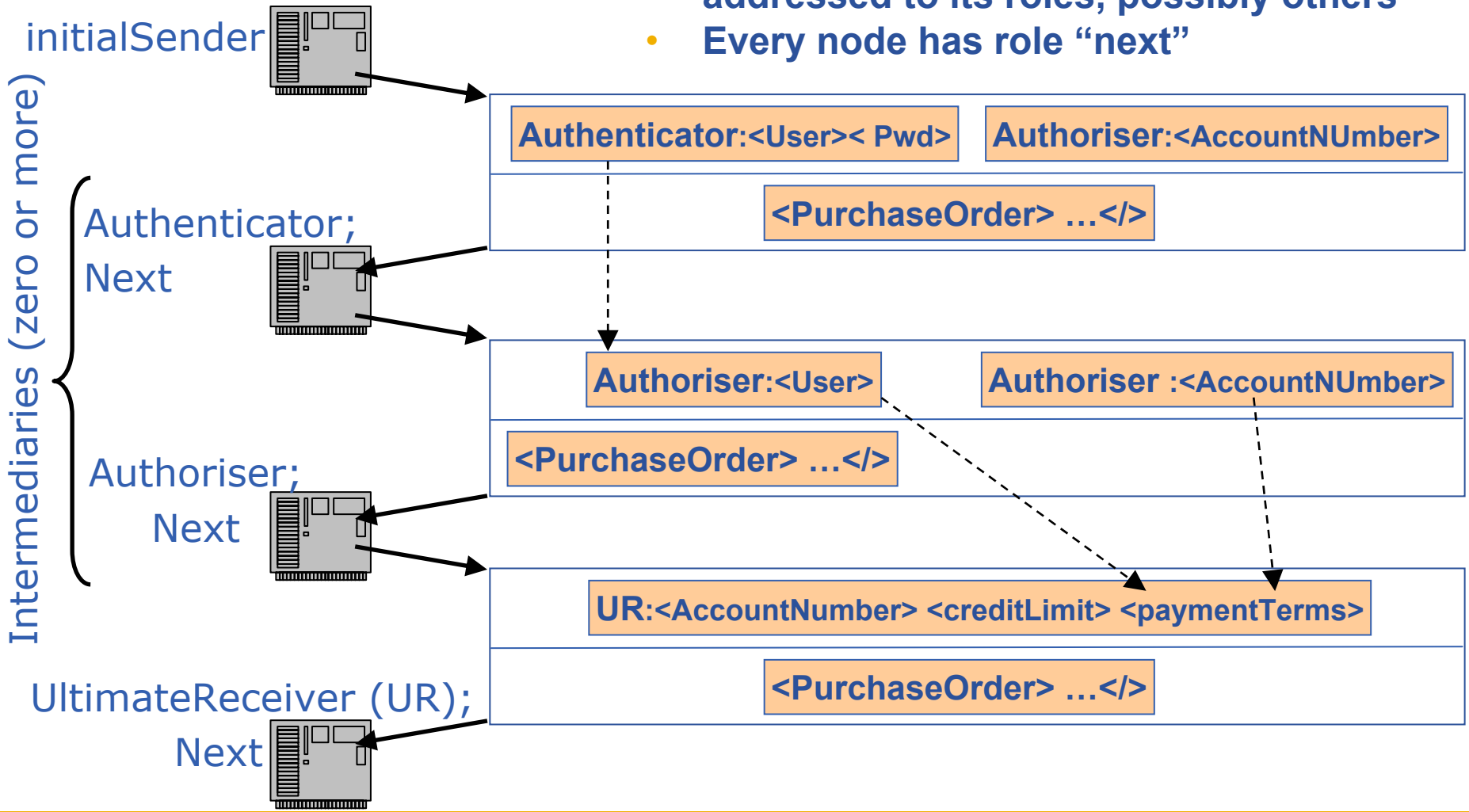
- **Envelope (XML root element)**
- **Header (optional)**
  - Multiple header blocks/entries
  - For different purposes – factorisation
  - For different processing stages
- **Body (mandatory)**
  - The payload
  - Zero or more XML elements
  - maybe a Fault element
    - Specific fault reporting standard



Multi-Stage Header Processing

SOAP Processing Node Roles/Actors

- Node adopts one or more roles
- Node attempts to process headers addressed to its roles, possibly others
- Every node has role “next”



XML Message Representation

```
HTTP    …
SOAP    <?xml version="1.0"?>
        <env:Envelope
                xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
                xlmns:m="http://company" >
          <env:Header>
                <m:authenticate env:actor="http://company/authenticator">
                  <m:username>Fred</>
                  <m:password>yhjik154</> </>
                <m:authorise env:actor="http://company/authoriser">
                  <m:accountNumber>17-365-37a</>  </></env:Header>
          <env:Body>
                <m:purchaseOrder> …. </> …. <env:Body> </env:Envelope>
```

Identifies Soap's Namespace – and the SOAP version used

Identifies the applications namespace – probabaly really several

Globally unique keyword for application-specific actor names

Header Attributes (Actor)

```
<env:Header>
  <m:authenticate
        …. application-specific-attribute="…" ….
          env:actor="http://company/authenticator"
          env:mustUnderstand="true">
      <m:username>Fred</><m:password>yhjik154</> ….  </>
```

Attributes - application-specific attributes and
standardised attributes- env:encodingStyle (see later)
processing flow – actor, mustUnderstand

Actor Attribute

One actor per message;
Multiple messages with same actor;
Multiple nodes with actor role;
One node adopting multiple actor roles

- User-defined e.g. authenticator
- Next env:actor="http://schemas.xmlsoap.org/soap/actor/next"
  - the next node should process it (including the UltimateReceiver)
- Default – no actor means actor is final recipient

Must Understand

```
<env:Header>
  <m:authenticate
        env:role="http://company/authenticator"
        env:mustUnderstand="1">
    <m:username>Fred</><m:password>yhjik154</> ….  </>
```

- **mustUnderstand="1" means "mandatory"**
- **A processing node can dynamicaly determine its set of user-defined roles for a particular message (+ next + possibly ultimateReceiver), E.g.**
  - {next, authenticator}
  - {next, ultimateReceiver, authenticator}
  - {next, ultimateReceiver}

Processing Rules

```
<env:Header>
  <m:authenticate
        env:role="http://company/authenticator"
        env:mustUnderstand="1">
    <m:username>Fred</><m:password>yhjik154</> ….  </>
```

- **For each message, the node has to consistently act in those roles**
  - Must not receive any headers not targeted at one of those roles
  - Must receive all headers targeted at one of those roles
  - Must process mandatory received headers
  - May process non-mandatory received headers
- **Receive means remove it**
  - may insert a similar one,
    - but that is a contract with this node,
    - not with the node inserting the original header
- **Processing means either**
  - deal with it according to its semantics
  - report an error
- **Body is as amandatory header with no actor (final recipient)**

Multiply-targeted information

```
<env:Header xlmns:m="http://company" >
  <m:multiUse>
      <m:username id= "userNameValue">Fred</>
        ....  </>
  <m:authenticate
                    env:role="http:/…/authenticator">
          <m:username href="userNameValue">
          <m:password>yhjik154</>  </>
<m:authorise
                    env:role="http:/…/authoriser">
          <m:username href="userNameValue"></>
```

header entry holding multi-used information

- **This can be used to pass graph structure in the body**

SOAP Faults

Faults reported in the body – single element

Zero or more header entries –

for detail error information pertaining to original header entries

Body

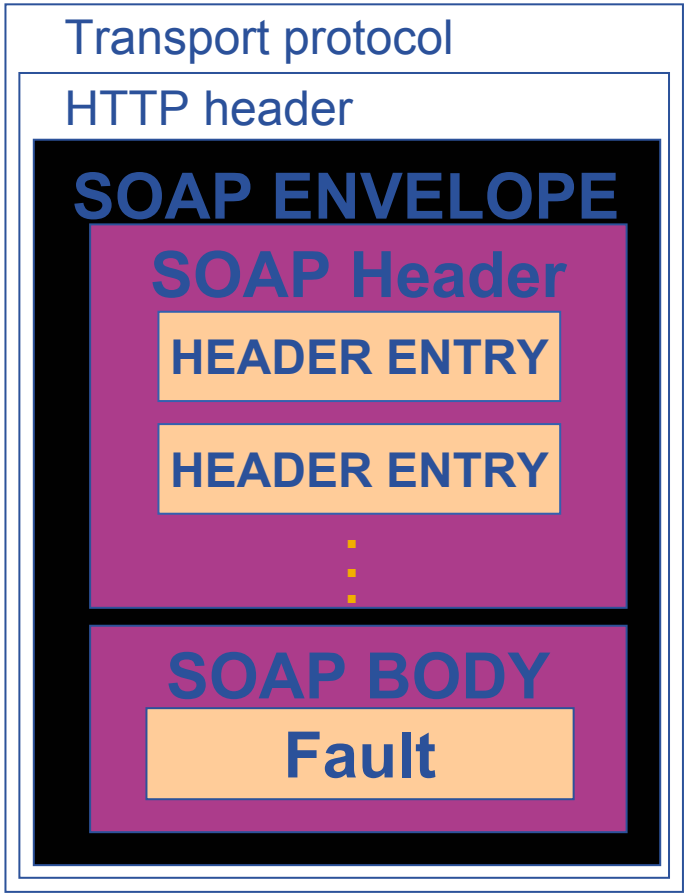
Fault

faultcode — a Qname, e.g env:mustUnderstand

faultstring — Human readable text

faultactor ? — Actor that was operating (URI) (default = ultimate destination, Mandatory otherwise)

detail ? — Any structure of further application-specific information. Its presence means body was processed



Fault Message Example

```
<env:Envelope xlmns:env="… .org/soap/envelope">
    <env:Body >
      <env:Fault>
        <env:faultcode>env:Server</>
        <env:faultstring>Server Error</></>
        <env:detail
              xlmns:m="http://company"
              env:encodingStyle="…">
          <m:faultdetail1>
              <m:faultcode>129</>
              <m:excuse>
                  not my fault really </> </>
          <m:faultdetail2> …. </>
                  </></></></></>
```

} Standard error code

} Explanation

} Application-specific Error code

} Explanation

The Standard Fault Codes

- **env:VersionMismatch**
  - Un-recognised namespace for the env:Envlope
- **env:MustUnderstand**
  - A mandatory header entry was not understood
- **env:Client**
  - It's your fault (e.g. wrong info. In body); re-send won't work.
  - Must have detail element
- **env:Server**
  - It's our fault (e.g an upstream processing node not responding).
  - Might succeed if sent later.
  - Can have detail element

MAPPINGS

- **Goals**
  - To understand the structure and meaning of SOAP messages
  - To understand how SOAP messages are standardly used for RPC over HTTP
- **Outline**
  - SOAP architecture
    - What soap is
    - Message structure
    - Processing Model
    - Faults
  - SOAP Mappings
    - Serialisation
    - Bindings
    - RPC

Intended to be orthogonal – mix and match

Encoding Styles

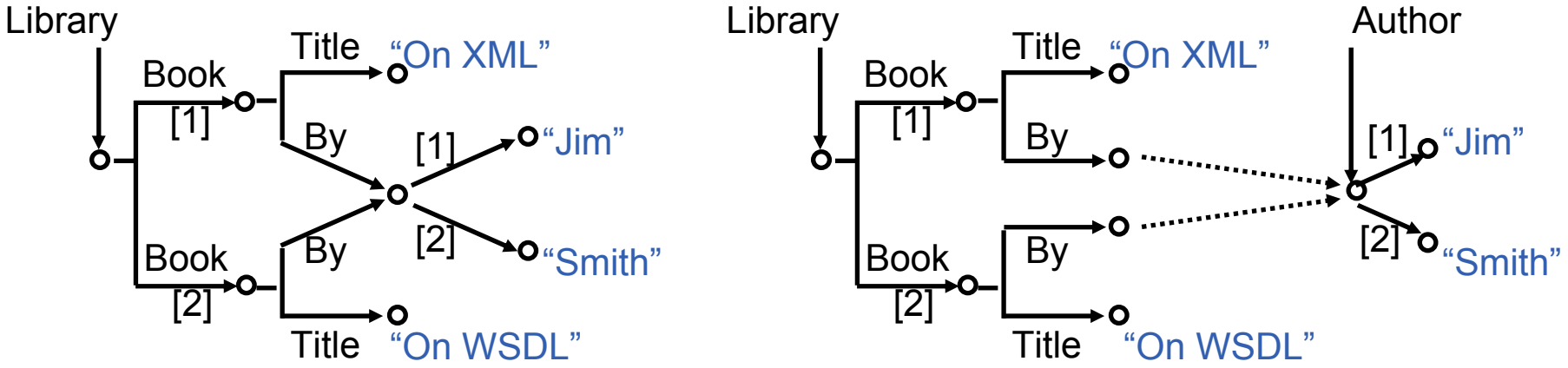
```
<env:Envelope       xmlns:env="http://www.w3.org/2003/05/soap-envelope"
                    xlmns:m="http://company"
                     env:encodingStyle= "http://schemas.xmlsoap.org/soap/encoding/" >
   <env:Header> … </>
   <env:Body>
       <m:purchaseOrder
          env:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/    http://company/encodeStyle1">
                    </></></></>
```

- **Encoding style is the serialization scheme,**
  - how logical structure is physically represented
- **Soap-encoding is standard, but can use**
  - A completely different one
  - An extension of soap encoding
  - A combination of encodings
- **Can define encoding on any element - usual scoping rules**

SOAP Serialisation

- **Encoding style is the serialization rules –**
- **For soap encoding this is**
  - 1. mapping
    - From a SOAP data model, a directed graph, with typed nodes
    - To a serial representation as a tree.
  - 2. Defining how to represent that tree in XML

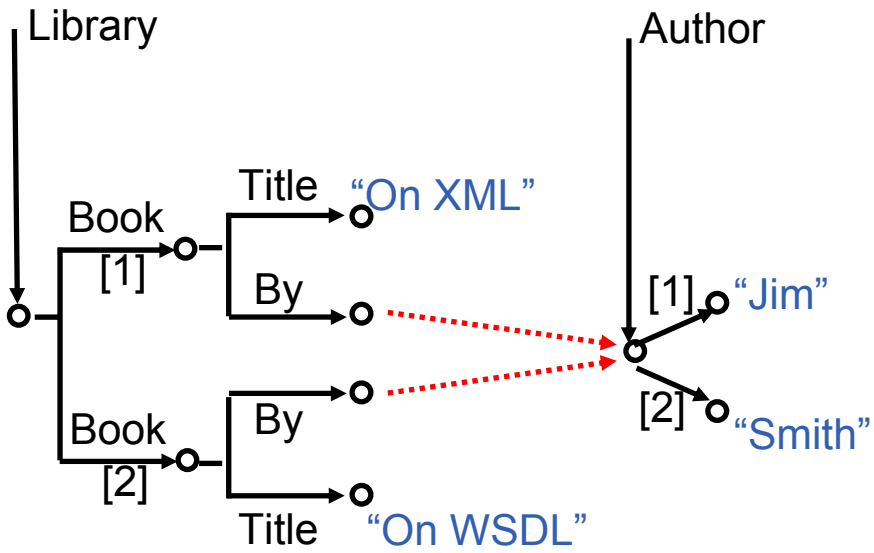
Tree-ifying A value Graph



- **Value Node**
  - Simple – character data – as can be defined in a Schema
  - Struct – outgoing edges distinguished by role name (its accessor)
  - Array - outgoing edges distinguished by position (its accessor)
  - Otherwise – by role name and position (its accessor)
  - Every node has a type – explicit or determined by associated schema
- **Serialisation – to a forest with reference links**
  - A node with N incoming edges becomes
    - A top level node
    - N leaf nodes referencing it and having no components

Tree-ifying A value Graph

```
<env:Envelope
        xmlns:env="…/soap/envelope"
        xlmns:m="http://company"
        env:encodingStyle="…encoding/" >
  <env:Body>
    <m:Library se:root="1">
      <book>  <Title> On XML</>
              <By href="A1"/>  </>
      <book>  <Title>On WSDL</>
              <By href="A1"/>  </>
    <m:Author id="A1" se:root="0">
      <Name>Jim</>
      <Name>Smith</> </></></></>
```



- **No attributes for values; all values as**
  - Child elements, for complex types
  - Character data for simple types
- **Unqualified names for local;**
- **Otherwise qualified**

- **Use href and id for cross-tree links**
- **Linked-to value must be top-level body entry**
- **Link can cross resource boundaries –**

  href is full URL

Simple Types

- **Every simple value has a type which is a (derivation of a) primitive type, as defined in Schemas standard, which defines their lexical form – (Review)**
- **Primitive Types**
  - **string**
  - **Boolean**
  - **Float**
  - **Double**
  - **Decimal**
  - **hexBinary**
  - **base64Binary**
  - **anyURI**
  - **QName**
  - **NOTATION**
  - **duration**
  - **dateTime**
  - **time**
  - **date**
  - **gYearMonth**
  - **gYear**
  - **gMonthDay**
  - **gDay**
  - **gMonth**
- **Derivations**
  - Lengths - length, maxLength,minLength
  - Limits – minInclusive, maxInclusive, minExclusive, maxExclusive
  - Digits – totalDigits, fractionalDigits – (value range and accuracy)
  - pattern – regular expression [A-Z]
  - enumeration – list of allowed values

SOAP Simple Types

- SOAP encoding allows all elements to have id and href attributes
- So have SOAP types that extends primitive types with those attributes
- Fragments from the SOAP encoding schema,

```
<xs:schema targetNamespace=
"http://schemas.xmlsoap.org/soap/encoding/">
…
<xs:attributeGroup name="commonAttributes">
  <xs:attribute name="id" type="xs:ID"/>
  <xs:attribute name="href" type="xs:anyURI"/>
  <xs:anyAttribute namespace="##other"
                   processContents="lax"/>
</xs:attributeGroup>
…
```

```
…
<xs:complexType name="integer">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
     <xs:attributeGroup
          ref="tns:commonAttributes"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

- Example usage – schema for a soap message

```
<xsd:schema xmlns:SEnc= "http://schemas.xmlsoap.org/soap/encoding/">
<import location= "http://schemas.xmlsoap.org/soap/encoding/">
….. <xsd:element name=anInt type="SEnc:integer"> ….
```

Compound Types

- **If the order is significant, encoding must follow that required order**
  - For Schema **sequence** – order is significant
  - For Schema **any** – order is not significant
- **Soap encoding schema provides two compound types**
- **Se:Struct – components are uniquely named**
- **Se:Array – components are identified by position**
- **Both have href and id atributes**
- **Arrays have further attributes**

Compound Types - Arrays

- **Array is of type SEnc:Array or some derivative thereof**
  - Attibutes SEnc:href SEnc:id for referencing
- **Can specify shape and component type**

```
<element name="A" type="se:Array"/>
```

Schema

```
<A se:arrayType="xsd:integer [2,3] [2]">
      <A1>
            <n>111</n> <n>112</n> <n>113</n>
            <n>121</n> <n>122</n> <n>123</n>  </>
      <A2>
            <n>211</n> <n>112</n> <n>213</n>
            <n>221</n> <n>122</n> <n>223</n>  </>
</>
```

Message

- **[2] - An array of 2 elements -**
- **[2,3] Each is a 2 x 3 array of**
- **Xsd:integer**

Partial Arrays

- **Partially transmitted array, offset at which it starts**

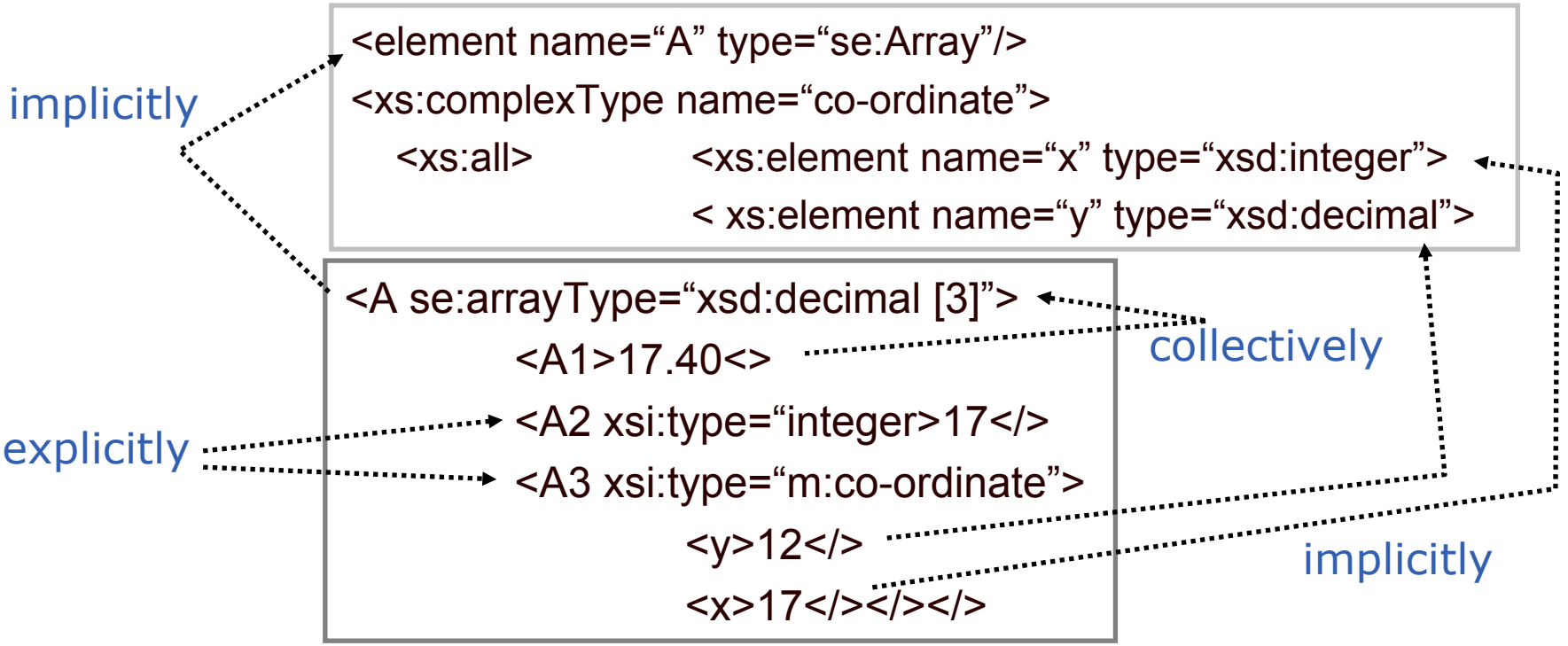
```
<se:Array    se:arrayType="xsd:integer [5]"     se:offset="[2]" >
            <! - - omitted elements 0, 1 and 2-- >
            <i>3</>  <i>4</>   </>
```

- **Sparse Array – each element says its position**

```
<se:Array            se:arrayType="xsd:integer [ , ] [4]">
    <se:Array        se:position="[2]"
                     se:arrayType="xsd:integer[10,10]">
            <i se:position="[0,0]">11</>
            <i se:position="[3,8]">49</>    </></>
```

Typing

- **Type of a value** must **be determined, either –**
  - Explicitly - as xsi:type attribute for the element itself
  - Collectively - via type of containing compound value
  - Implicitly - by name and schema definition



MAPPINGS

- **Goals**
  - To understand the structure and meaning of SOAP messages
  - To understand how SOAP messages are standardly used for RPC over HTTP
- **Outline**
  - SOAP architecture
    - What soap is
    - Message structure
    - Processing Model
  - SOAP Mappings
    - Serialisation
    - Bindings
    - RPC

Protocol Binding – SOAP 1.1 over HTTP

```
POST /invoices?InvNo=165-983 HTTP/1.1
HOST: company.org
Content-Type: text/xml; charset="utf-8"
Content-Length: 561
SOAPAction: http://company.org/pay-invoice

  <?xml version="1.0" ?>
  <env:Envelope ….> … </>
```

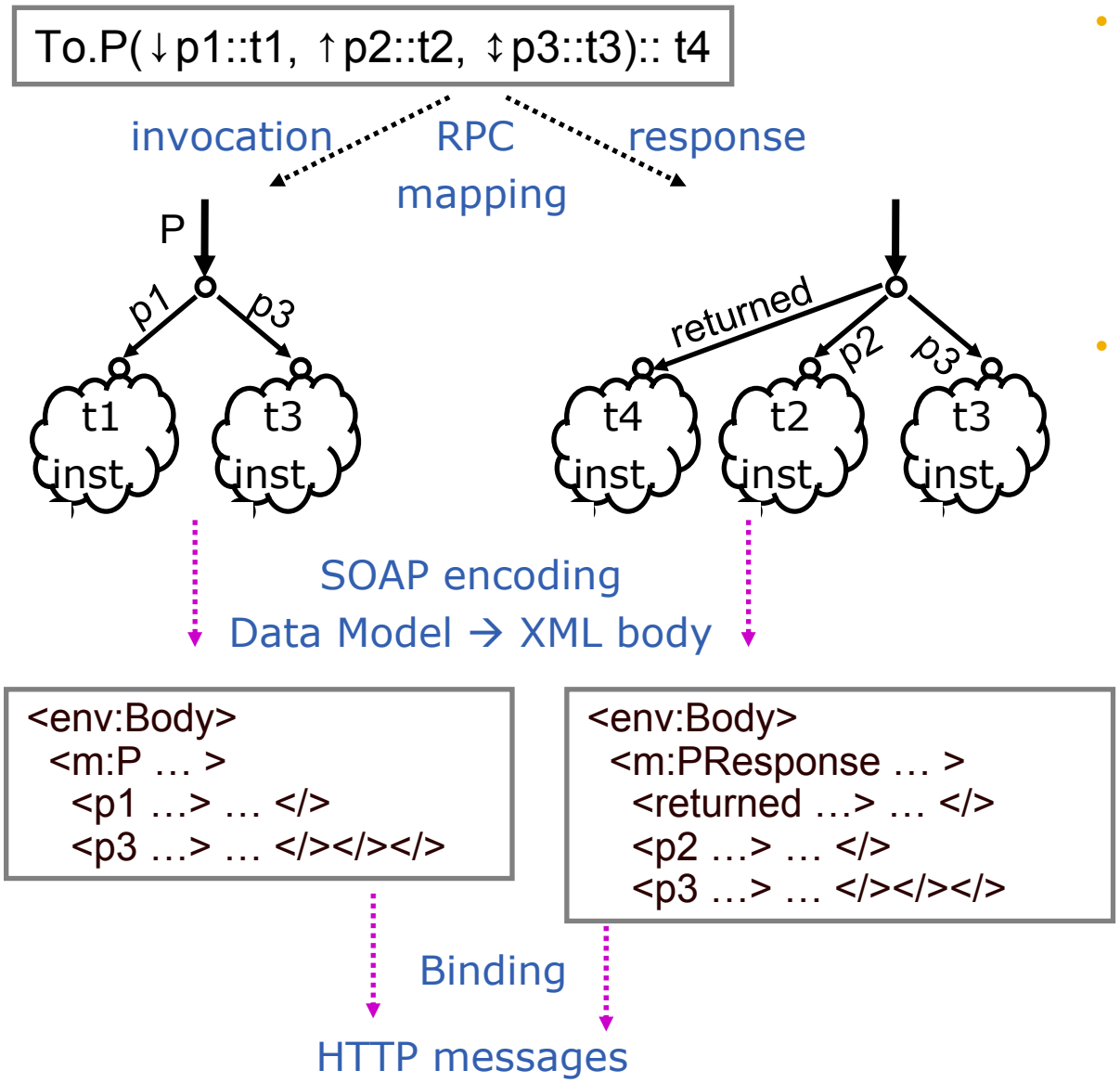
- **Request –**
  - Must have media type text/xml
  - Must use SOAPAction to indicate intention of message
  - This binding requires POST

```
HTTP/1.1 200 OK
HOST: company.org
Content-Type: text/xml; charset="utf-8"
Content-Length: 67

<?xml version="1.0" ?>
<env:Envelope ….> … </>
```

- **Response –**
  - Must have media type text/xml
  - Fault must use HTTP 500 response (internal server error)

RPC standard



- **Procedure P**
  - To

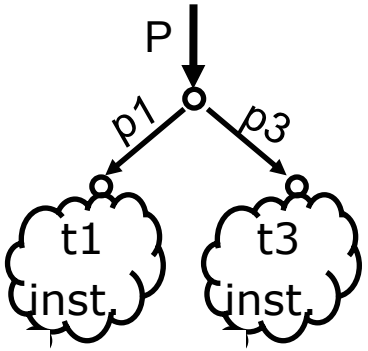
    identifies a target resource (the object in an O-O object invocation)

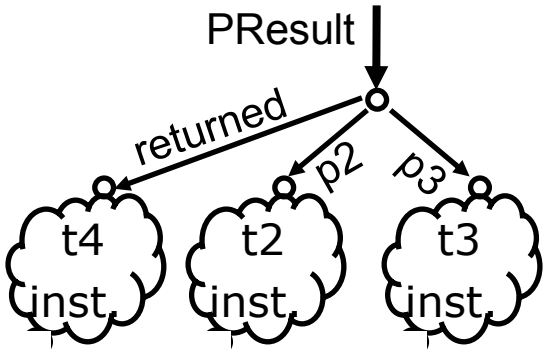
    In SOAP header
- **Parameters**
  - p1 – in
  - p2 – out
  - p3 – in/out
  - :: - of type

tN inst = instance of type tN

RPC Encoding - Rules

- **Invocation – one element (struct)**
  - name = the procedure/method name (P)
  - children named as the in and in/out parameter names,
  - in same order as in the signature
  - with same types
- **Response – one element (struct)**
  - Name insignificant (by convention PResult)
  - Children named as the output parameter names
  - Plus a result child if and only if non-void result
    - Must be first child
- **Additional Information**
  - Anything needed other than formal parameters/result may be expressed in the RPC encoding
  - If so it goes in the header entries, but not in the body
    - E.g resource (i.e. “object”) identifier – see WSRF





Example of RPC using SOAP encoding over HTTP

```
POST /SubmitPurchaseOrder HTTP/1.1
Host: www.company.org/ws
Content-Type: text/xml; charset="utf-8"
Content-Length: 356
SOAPAction: “www.company.org/ws/submitPO"

<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <env:Body>
      <m:PurchOrd>
        xmlns:m=“www.company.org/namespaces">
            <acc>17-A-53</>
            <items>
                <item><prodCode>15-56</>  <quantity>84</></>
                <item><prodCode>15-56</>  <quantity>84</></>
           </></>   </>
</>
```

Method interface not allow mixing of items and acc at same level So need This wrapper

END