UNIVERSITY *of* STIRLING

www.cs.stir.ac.uk

Web Services

SOAP

Simple Object Access Protocol

UNIVERSITY *of* STIRLING

Slide 1

SOAP

- Simple Object Access Protocol
- Not a programming language!

- A structured XML **message format**
- A protocol for **exchanging messages**
- An **encoding scheme** for representing data types in those messages
- Uses an underlying transport protocol (HTTP, SMTP etc) through **binding**

UNIVERSITY *of* STIRLING

Slide 2

SOAP

- SOAP provides platform neutral:
  - Message and Information exchanging
  - Invocation of remote functionality
- SOAP enables:
  - Distributed applications
  - Business-to-Business integration
  - Web Services
- SOAP version 1.2
  - W3C Recommendation (standard), April 2007
  - From XML Protocol Working Group
  - http://www.w3.org/TR/soap/

UNIVERSITY *of* STIRLING

Slide 3

Why SOAP

- Many applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA.
- RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this traffic.
- A better way to communicate between applications is over HTTP, because HTTP is supported by all Internet browsers and servers. SOAP was created to accomplish this.
- SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages.

UNIVERSITY *of* STIRLING

Slide 4

SOAP messages

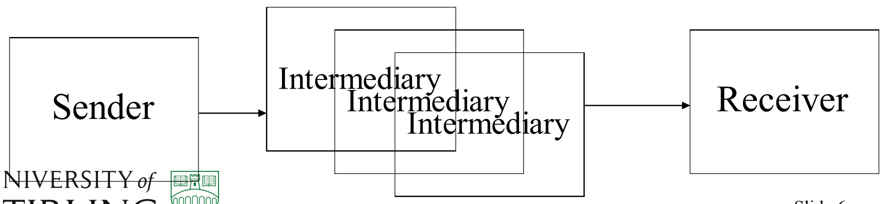
- SOAP messages are
  - Stateless
  - One-way
  - Composable, e.g. WSDL operation types
    - One-way
    - Request-response
    - Solicit-response
    - Notification
  - Transferred between SOAP nodes (apps)

UNIVERSITY *of* STIRLING

Slide 5

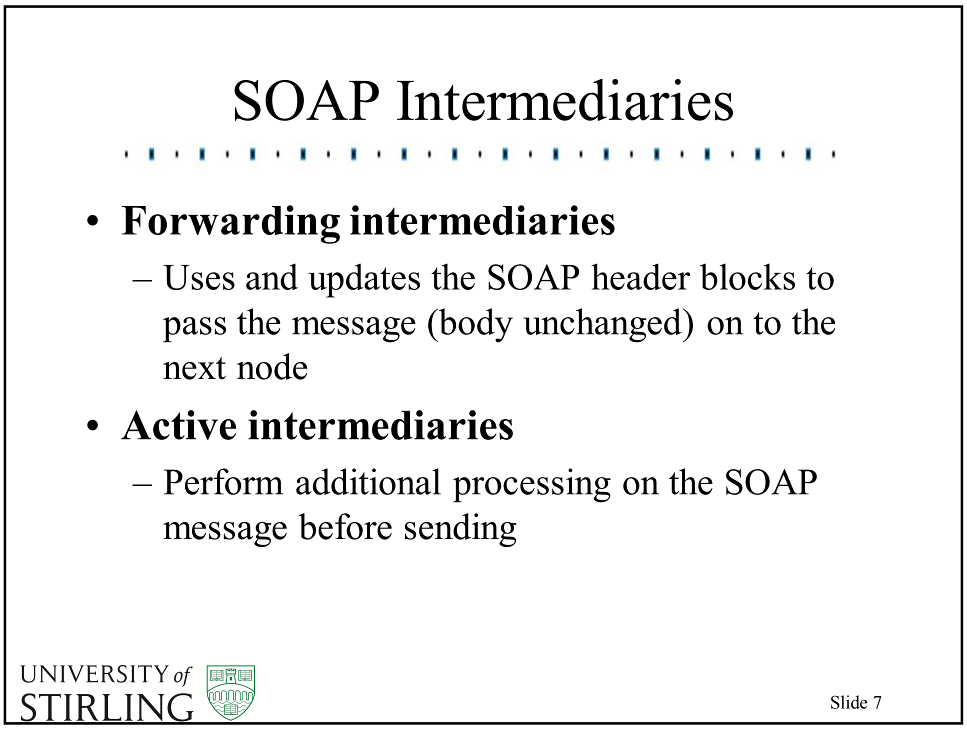
SOAP nodes

- **SOAP Sender**
  - Generates & sends the message
- **SOAP Receiver**
  - Ultimately receives and processes the message
  - May generate a SOAP response, message or fault as a result
- **SOAP Intermediary**
  - Zero or more
  - Receives, processes (e.g. routes) and resends the message



UNIVERSITY *of* STIRLING

Slide 6

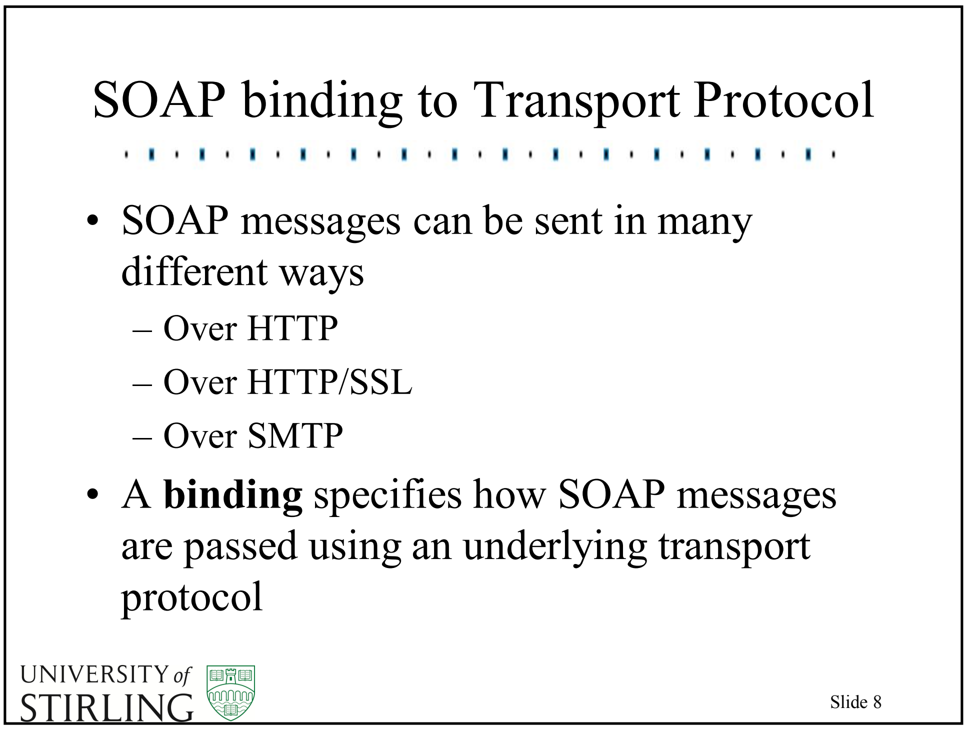


SOAP Intermediaries

- **Forwarding intermediaries**
  - Uses and updates the SOAP header blocks to pass the message (body unchanged) on to the next node
- **Active intermediaries**
  - Perform additional processing on the SOAP message before sending

UNIVERSITY *of* STIRLING

Slide 7



SOAP binding to Transport Protocol

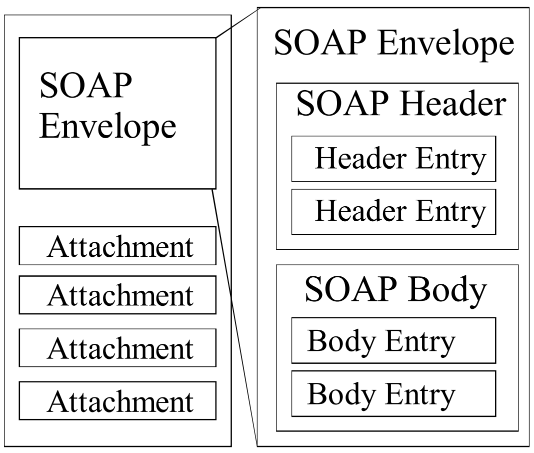
- SOAP messages can be sent in many different ways
  - Over HTTP
  - Over HTTP/SSL
  - Over SMTP
- A **binding** specifies how SOAP messages are passed using an underlying transport protocol

UNIVERSITY *of* STIRLING

Slide 8

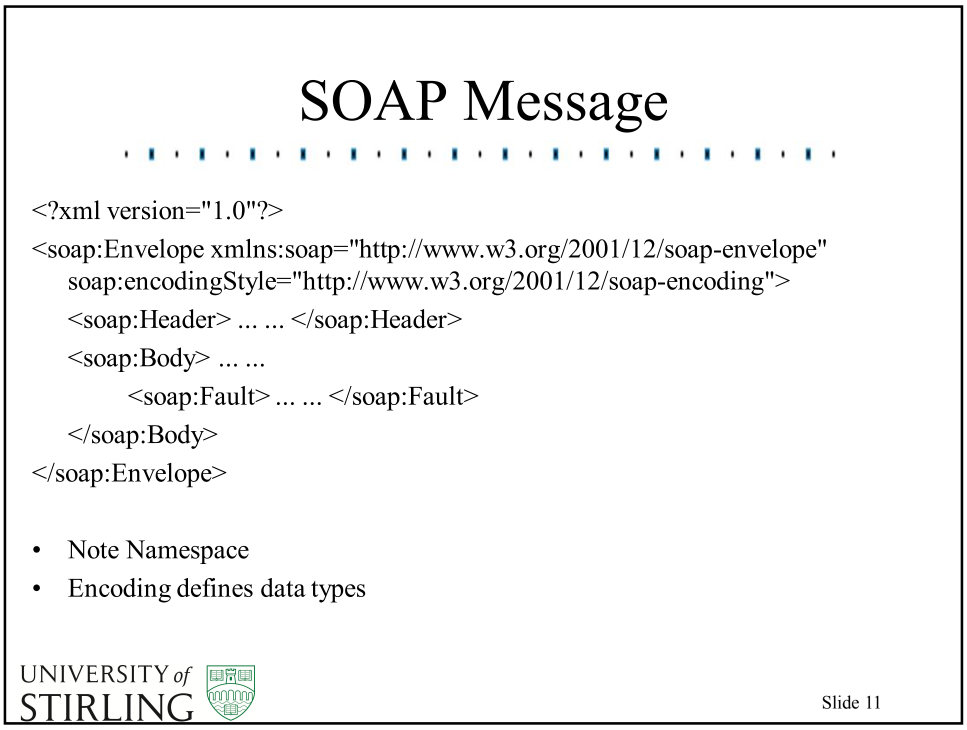
SOAP Message structure

- **Envelope**
  - Identifies that this is a SOAP message.
- • **Header**
  - Optional & application specific
  - Entries may be addressed to a particular SOAP node
- • **Body**
  - Mandatory
  - Contains message "payload"



SOAP message structure

- Additional components:
- **Faults**
  - Details of what and where something went wrong
- **Attachments**
  - E.G. Binary Data (GIF, JPEG, MP3 etc)
  - Typically carried outside envelope
  - Uses Multipurpose Internet Mail Extensions (MIME)



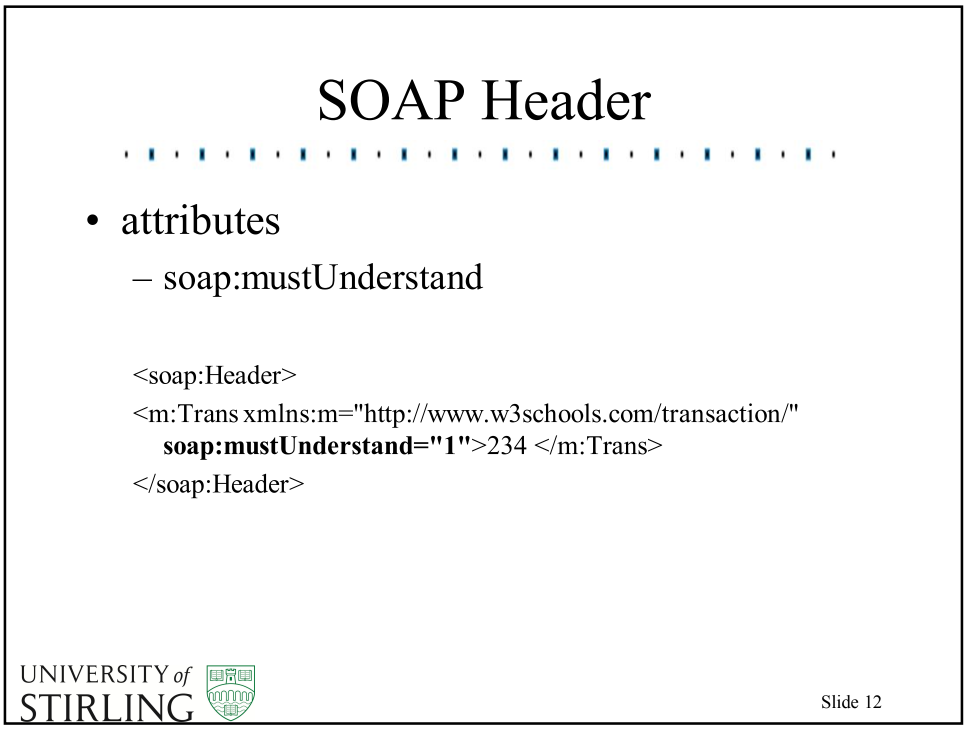
SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header> ... ... </soap:Header>
    <soap:Body> ... ...
            <soap:Fault> ... ... </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

- Note Namespace
- Encoding defines data types

UNIVERSITY *of* STIRLING

Slide 11



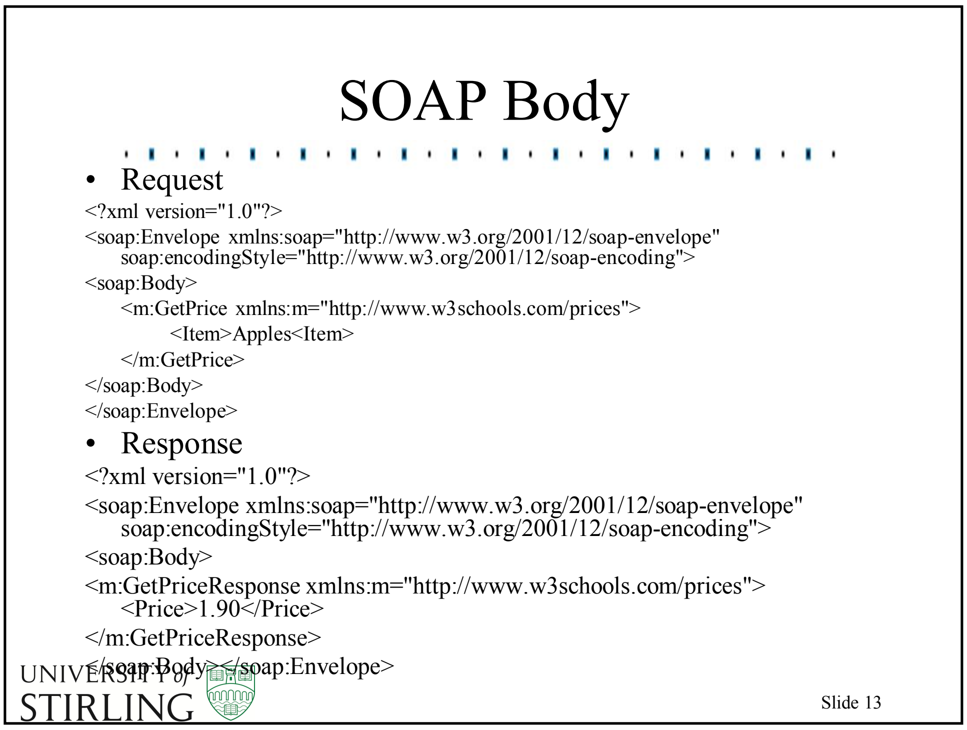
SOAP Header

- attributes
  - soap:mustUnderstand

```
<soap:Header>
<m:Trans xmlns:m="http://www.w3schools.com/transaction/"
    soap:mustUnderstand="1">234 </m:Trans>
</soap:Header>
```

UNIVERSITY *of* STIRLING

Slide 12



SOAP Body

- Request

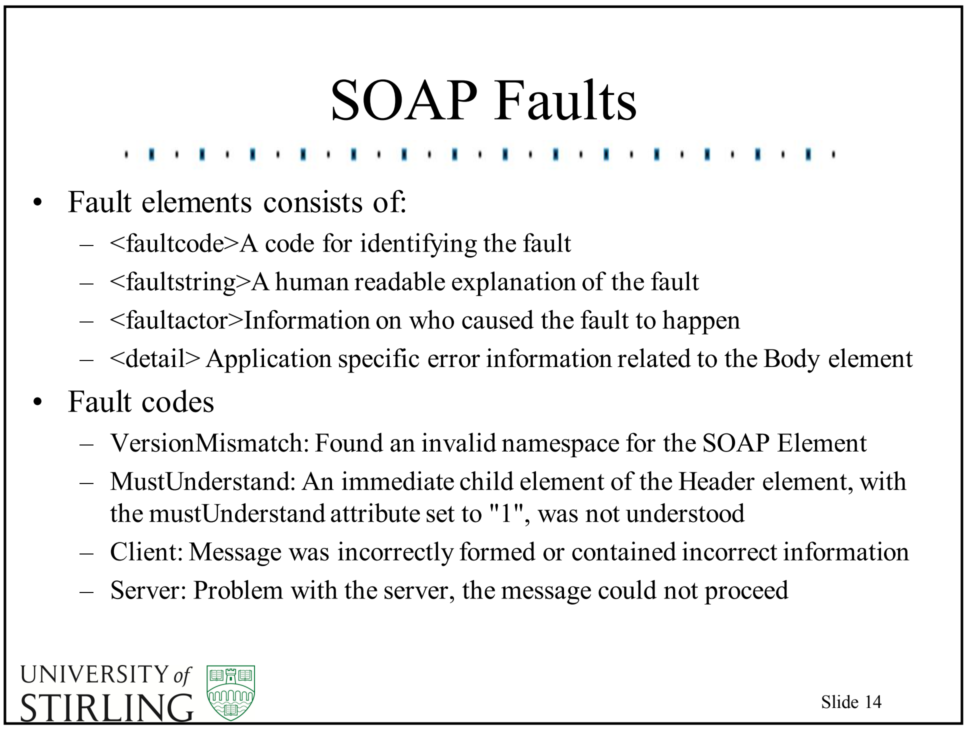
```
<?xml version="1.0"?>
<soap:Envelope  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
    <m:GetPrice  xmlns:m="http://www.w3schools.com/prices">
        <Item>Apples<Item>
    </m:GetPrice>
</soap:Body>
</soap:Envelope>
```

- Response

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body>
<m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <Price>1.90</Price>
</m:GetPriceResponse>
</soap:Body></soap:Envelope>
```

UNIVERSITY *of* STIRLING

Slide 13



SOAP Faults

- Fault elements consists of:
  - <faultcode>A code for identifying the fault
  - <faultstring>A human readable explanation of the fault
  - <faultactor>Information on who caused the fault to happen
  - <detail> Application specific error information related to the Body element
- Fault codes
  - VersionMismatch: Found an invalid namespace for the SOAP Element
  - MustUnderstand: An immediate child element of the Header element, with the mustUnderstand attribute set to "1", was not understood
  - Client: Message was incorrectly formed or contained incorrect information
  - Server: Problem with the server, the message could not proceed

UNIVERSITY *of* STIRLING

Slide 14

Communication

- SOAP provides two communication models:
- **SOAP RPC**
  - Synchronous request-response
  - Request encodes method & arguments
  - Response encodes result value or fault
- **SOAP Messaging (document)**
  - Document-driven: XML
  - Normal XML description e.g. of products can be sent
  - No reference to operation names
  - Operations must have a single element

SOAP RPC

- The Request body describes
  - The name of the method to invoke
  - Optional arguments to pass to that method
- Includes the WSDL operation
- Parameters are based on WSDL types
- WSDL operations can include one or more parts
- May be identified by order and/or by name
- The Response body describes
  - The return value(s) from the method or
  - SOAP fault

SOAP Message (document)

- Each message body is an XML document or “literal XML”
  - can be validated against pre-defined XML schema document
  - A body element type typically identifies the message type
  - And therefore how/by what it should be handled
- No Operation name in SOAP message
- Parts of a message are based on schema element definitions rather than WSDL types
- Operations have a single part

UNIVERSITY *of* STIRLING

RPC vs Message

- RPC is **function-centric**
  - RPC has **tight coupling** between the message and the implementation
- Messaging is **data-centric**
  - Messaging has **loose coupling** between the message and the implementation

UNIVERSITY *of* STIRLING

Data encoding

- **Literals**: XML fragments, defined in XML Schema
  - Commonly used in XML messaging scenarios
- **Encoded values**: defined in SOAP Encoding
  - A set of rules for representing data types (not supported in Axis2)
- Defines standard XML encoding for commonly observed programming language types
  - Simple types, Enumerations
  - Compound types, e.g. structs, objects
  - Arrays, References

UNIVERSITY *of* STIRLING

Slide 19

Examples

- suppose a service supports an add operation that accepts two integers (i, j) and returns their sum; it may also report a ‘result too large’ fault

UNIVERSITY *of* STIRLING

Slide 20

rpc/literal

```
<soap:Envelope>
    <soap:Body>
        <add>
                <i>12</i>
                <j>5</j>
        </add>
    </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope>
    <soap:Body>
        <addResponse>
                <res>17</res>
        </addResponse>
    </soap:Body>
</soap:Envelope>
```

- request to add 12 to 5 defines arguments by name, and wraps them in the operation (add)
- response must be a data structure even if a simple type is being returned; conventionally this is the operation name with Response appended

UNIVERSITY *of* STIRLING

Slide 21

rpc/literal –V2

```
<soap:Envelope>
    <soap:Body>
        <add>
            <operands>
                <i>12</i>
                <j>5</j>
            </operands>
        </add>
    </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope>
    <soap:Body>
        <addResponse>
                <res>17</res>
        </addResponse>
    </soap:Body>
</soap:Envelope>
```

- request now wraps two parameters in an operands element inside the operation (add)
- response unchanged

UNIVERSITY *of* STIRLING

Slide 22

Fault in rpc/literal

```
<soap:Envelope>
   <soap:Body>
        <soap:Fault>
           <soap:faultcode> soap:Sender</soap:faultcode>
          <soap:faultstring>Addition result too large</soap:faultstring>
          <soap:faultactor>http://aws.xyz.com/lists</soap:faultactor>
        </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

UNIVERSITY *of* STIRLING

Slide 23

Document /literal

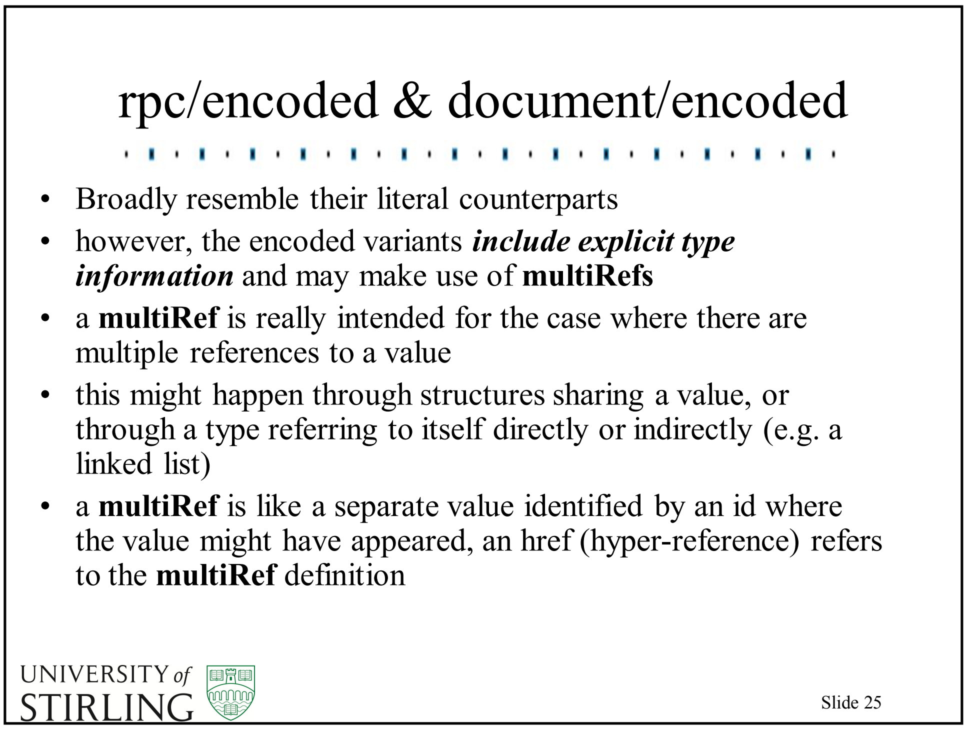
- Same request
- Missing operation
- Must have single element as parameter

```
<soap:Envelope>
   <soap:Body>
        <operands>
            <i>12</i>
            <j>5</j>
        </operands>
   </soap:Body>
</soap:Envelope>
```

```
<soap:Envelope>
   <soap:Body>
        <res>17</res>
   </soap:Body>
</soap:Envelope>
```

UNIVERSITY *of* STIRLING

Slide 24



rpc/encoded & document/encoded

- Broadly resemble their literal counterparts
- however, the encoded variants ***include explicit type information*** and may make use of **multiRefs**
- a **multiRef** is really intended for the case where there are multiple references to a value
- this might happen through structures sharing a value, or through a type referring to itself directly or indirectly (e.g. a linked list)
- a **multiRef** is like a separate value identified by an id where the value might have appeared, an href (hyper-reference) refers to the **multiRef** definition

UNIVERSITY *of* STIRLING

Slide 25