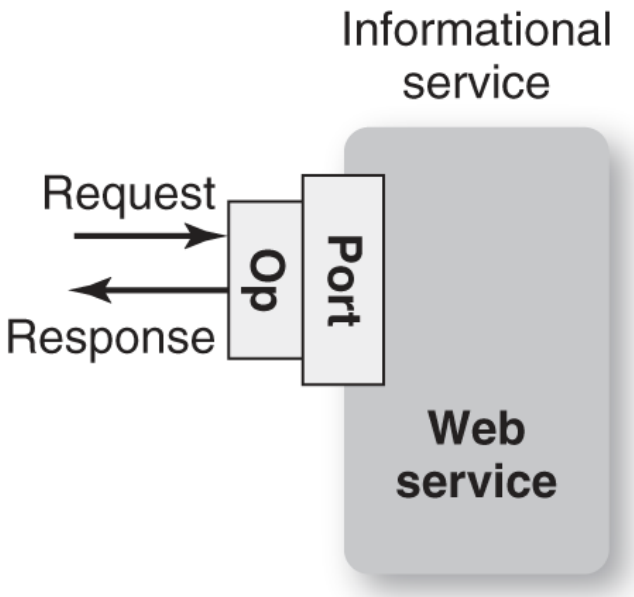
cphbusiness

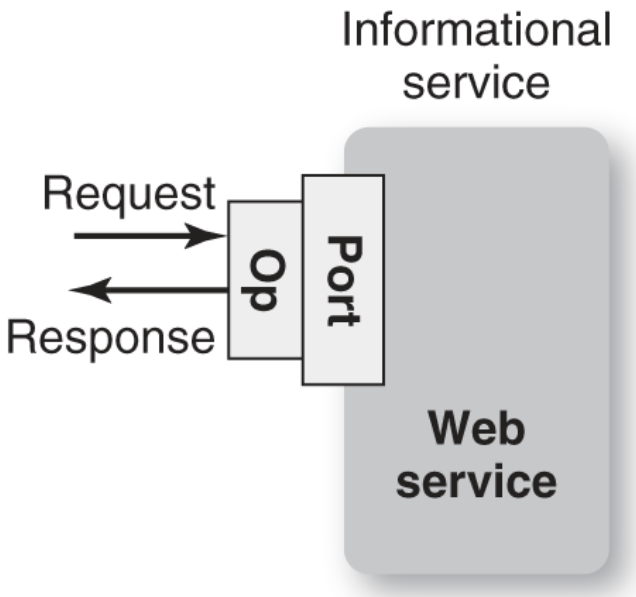
Types of Web Services

Informational services



- Services of **simple** nature
- Provide **access** to content interacting with an end-user by means of simple request/response sequences
- Or **expose** back-end business applications to other applications

Types of Web Services



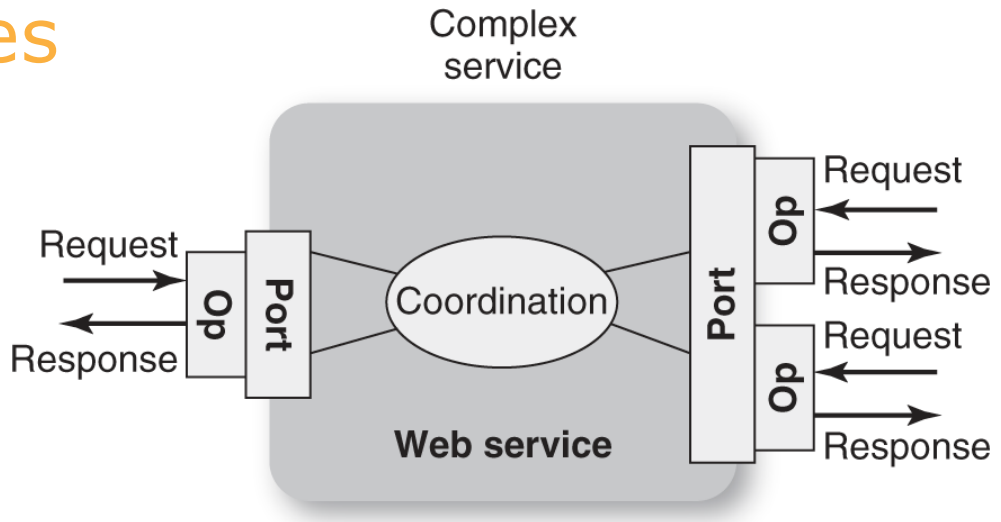
Informational services – Examples:

- ***Content services*** such as weather forecasts, stock quotes, news items, etc.
- ***Simple trading services*** providing a seamless aggregation of information across disparate systems and data sources, e.g., logistic services
- ***Information syndication services***: value-added info web services plugged into commerce sites, e.g., reservation services on travel site

Types of Web Services

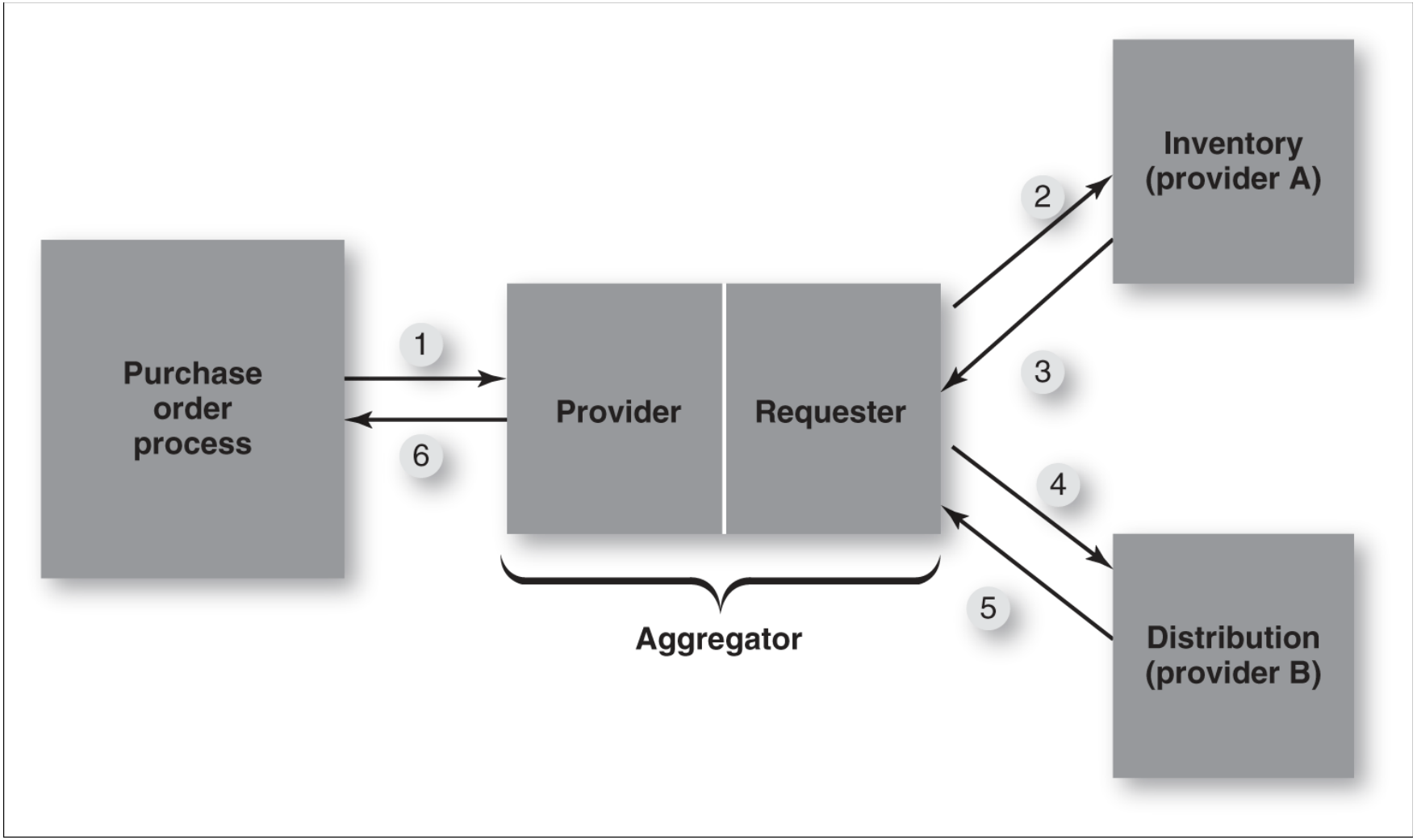
Complex services

(Composite Services)



- typically involve the **assembly** and **invocation** of **many pre-existing services** possibly found in diverse enterprises to complete a multi-step business interaction that requires coordination.
- Example:
  - a supply-chain application involving order taking, stocking orders, sourcing, inventory control, financials, and logistics

Types of Web Services – Complex Services



cphbusiness

XML & SOAP

cphbusiness

Outline

- XML
  - XML document structure
  - XML schemas reuse
  - Document navigation and transformation
- Simple Object Access Protocol (SOAP)
  - Inter application communication
  - SOAP as a messaging protocol
  - Structure of a SOAP message
  - SOAP communication model
  - SOAP fault message
  - SOAP over HTTP
  - Advantages and disadvantages of SOAP

EXtensible Markup Language (XML)

Sådan bruges PowerPoint i Cphbusiness

cphbusiness

Markup

- A method of distinguishing text from instructions in typesetting systems.
- Markup = instructions to computerized typesetting systems.
- Special characters are used to show where a markup instruction starts and stops.

Example:

```
<center> This is a <italics> very serious </italics> matter.</center>
```

This is a *very serious* matter

Why XML?

- XML is a standard for data exchange
- XML is closely related to object-oriented and so-called semi-structured data
- Is extensible, i.e., has a schema defining documents
- Is machine readable
  - All major database products have been retrofitted with facilities to store and construct XML documents

cphbusiness

Book Catalogue in XML

```
<book>
  <title>
      Web Services: Principles
and Technology
  </title>
  <author>
    Mike P. Papazoglou
  </author>
  <date>
    2007
  </date>
  <publisher>
    Prentice Hall
  </publisher>
</book>
```

XML tags allow computers and humans to ‘easily’ extract, e.g., publisher data

cphbusiness

XML structure

- **Document type:** XML documents are regarded as having types.
  - XML's constituent parts and their structure formally define the type of a document.
- An XML document is composed of named **containers** and their contained data values.
  - containers are represented as declarations, elements, and attributes.
- An XML document is also known as an **instance or XML document instance** to signify that it represents one possible set of data for a particular markup language.

cphbusiness

Example of an XML Instance

```
<?xml version:"1.0" encoding:"UTF-8"?>

<BillingInformation>
  <Name> Right Plastic Products </Name>
  <BillingDate> 2012-09-15 </BillingDate>
  <Address>
    <Street> 158 Edward st. </Street>
    <City> Brisbane </City>
    <State> QLD </State>
    <PostalCode> 4000 </PostalCode>
  </Address>
</BillingInformation>
```

cphbusiness

Layout of a typical XML document

cphbusiness

XML Elements

- Elements are fundamental units of content comprising element name and element content
  - An **element** is a sequence of characters that begins with a start tag and ends with an end tag and includes everything in between

```
<chapter number="1">
     Text for Chapter 1
</chapter>
```

- **What is content?** The characters in between the tags (rendered in green in this presentation) constitute the **content**
- The topmost element of the XML document is a single element known as the **root element**
- Elements contained in other elements are referred to as **nested elements**
  - The containing element is the **parent element** and the nested element is called the **child element**

XML Attributes

- Another way to insert data into an XML document is by adding attributes to start tags.
  - An attribute specification is a name–value pair that is associated with an element.
  - Attributes are used to better specify the content of an element on which they appear by adding information about a defined element.

```
<?xml version:"1.0" encoding:"UTF-8"?>

<BillingInformation customer-type="manufacturer">
  <Name> Right Plastic Products </Name>
  <BillingDate> 2012-09-15 </BillingDate>
  <Address>
    ...
  </Address>
</BillingInformation>
```

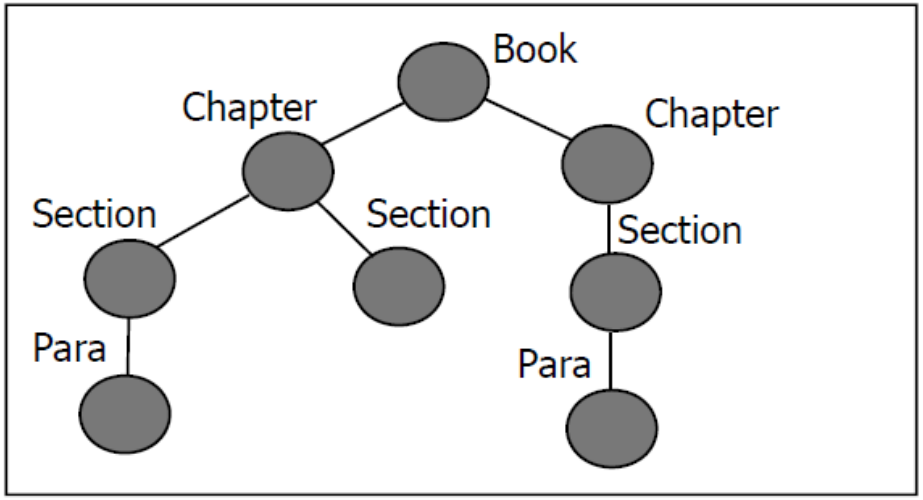
XML Structure

- An XML document **must have a root** tag
- An XML document is an information unit that can be seen in two ways:
  - As a **linear sequence of characters** that contain characters data and markup
  - As an abstract data structure that is a **tree of nodes**

Linear sequence

```
<book>
  <chapter n="1">  Title 1 </chapter>
  <section n="1.1"> Section 1.1 </section>
  <paragraph> .... <paragraph>
  <section n="1.2"> Section 1.2 </section>
  <chapter n="2">  Title 2 </chapter>
  <section n="2.1"> Section 2.1 </section>
  < paragraph > .... < paragraph >
</book>
```

Tree



cphbusiness

XML Namespaces

- Namespaces in XML provide a facility for associating the elements and/or attributes in all or part of a document with a particular schema and avoiding name clashes.
- Namespace declarations have a scope.
  - A namespace declaration is in scope for the element on which it is declared and of that element's children.
- The namespace name and the local name of the element together form a globally unique name known as a *qualified* name

```
<?xml version:"1.0"  encoding:"UTF-8"?>

<BillingInformation  customer-type="manufacturer"
    xmlns="http://www.plastics.com/BillInfo">
  <Name> Right Plastic Products </Name>
  <BillingDate> 2012-09-15 </BillingDate>
  <Address xmlns="http://www.plastics.com/Address">
    ...
  </Address>
</BillingInformation>
```

Uniform Resource Identifiers (URI)

XML Schema

- **XML schema** refers to a document that defines the content of and structure for expressing XML documents.
- Schemas provide support for additional meta-data characteristics such as structural **relationships**, **cardinality**, **valid values**, and **data types**.
- Each type of schema acts as a method of describing data characteristics and applying **rules** and **constraints** to a referencing XML document.
- An XML schema describes the elements and attributes that may be contained in a schema conforming document and the ways that the elements may be arranged within a document structure.

cphbusiness

XML Schema & Instance Document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="First" type="xs:string"/>
        <xs:element name="Middle" type="xs:string" minOccurs="0"/>
        <xs:element name="Last" type="xs:string"/>
        <xs:element name="Age" type="xs:integer"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Person.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Person.xsd">
  <First>Sophie</First>
  <Last>Jones</Last>
  <Age>34</Age>
</Person>
```

Person.xml

cphbusiness

XML Schema Components – Complex Types

```
<xsd:complexType name="PersonType">
  <xsd:sequence>
    <xsd:element name="First" type="xsd:string"/>
    <xsd:element name="Last" type="xsd:string"/>
    <xsd:element name="Title" type="xsd:string" minOccurs="0"/>
        <xsd:element name="PhoneExt" type="xsd:int"/>
        <xsd:element ref="EMail"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Person" type="PersonType"/>
<xsd:element name="VIP" substitutionGroup="Person">
  <xsd:complexType>
    <xsd:complexContent>
          <xsd:extension base="PersonType">
            <xsd:attribute name="IQ" type="xsd:int"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

cphbusiness

XML Schema Reuse

Complex Type Extensions

cphbusiness

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:PO="http://www.plastics_supply.com/PurchaseOrder"
  targetNamespace="http://www.plastics_supply.com/PurchaseOrder">

          <xsd:complexType name="Address">
             <xsd:sequence>
                 <xsd:element name="Number" type="xsd:decimal"/>
                 <xsd:element name="Street" type="xsd:string"/>
                 <xsd:element name="City" type="xsd:string" minOccurs="0"/>
             </xsd:sequence>
          </xsd:complexType>

          <xsd:complexType name="AustralianAddress">
              <xsd:complexContent>
                  <xsd:extension base="PO:Address">
                    <xsd:sequence>
                      <xsd:element name="State" type="xsd:string"/>
                      <xsd:element name="PostalCode" type="xsd:decimal"/>
                      <xsd:element name="Country" type="xsd:string"/>
                    </xsd:sequence>
                  </xsd:extension>
              </xsd:complexContent>
          </xsd:complexType>
</xsd:schema>
```

Complex Type Restrictions

```
<!-- Uses the data type declarations from Listing on page 17 -->
 <xsd:complexType name="AustralianPostalAddress">
    <xsd:complexContent>
       <xsd:restriction base="PO:AustralianAddress">
              <xsd:sequence>
                 <xsd:element name="Number" type="xsd:decimal"/>
                 <xsd:element name="Street" type="xsd:string"/>
                 <xsd:element name="City" type="xsd:string" minOccurs="0" maxOccurs="0"/>
                 <xsd:element name="State" type="xsd:string"/>
                 <xsd:element name="PostalCode" type="xsd:decimal"/>
              <xsd:element name="Country" type="xsd:string"/>
           </xsd:sequence>
       </xsd:restriction>
    </xsd:complexContent>
 </xsd:complexType>
```

cphbusiness

Complex Type Polymorphism

```
<!-- Uses the data type declarations from Listing on page 17 -->
  <xsd:complexType name="PurchaseOrder">
    <xsd:sequence>
      <xsd:element name="Name" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="shippingAddress" type="PO:Address" minOccurs= "1"
                                                            maxOccurs="1"/>
      <xsd:element name="billingAddress" type="PO:Address" minOccurs= "1"
                                                            maxOccurs="1"/>
        <xsd:choice minOccurs="1" maxOccurs="1">
            <xsd:element name="BillingDate"  type="xsd:date"/>
            <xsd:element name="ShippingDate" type="xsd:date"/>
        </xsd:choice>
    </xsd:sequence>
  </xsd:complexType
```

Variant of the PurchaseOrder type that uses the base type Address for its billingAddress and shippingAddress elements.

cphbusiness

Complex Type Polymorphism

```
<?xml version="1.0" encoding="UTF-8"?>
<PO:PurchaseOrder
xmlns:PO="http://www.plastics_supply.com/PurchaseOrder">

    <Name> Plastic Products </Name>
    <shippingAddress xsi:type="PO:AustralianAddress">
      <Number> 459 </Number>
      <Street> Wickham st. </Street>
      <City> Fortitude Valley </City>
      <State> QLD </State>
      <PostalCode> 4006 </PostalCode>
      <Country> Australia </country>
    </shippingAddress>

    <billingAddress xsi:type=="PO:AustralianPostalAddress">
      <Number> 158 </Number>
      <Street> Edward st. </Street>
      <State> QLD </State>
      <PostalCode> 4000 </PostalCode>
      <Country> Australia </Country>
    </billingAddress>
      <BillingDate> 2002-09-15 </BillingDate>
</PO:PurchaseOrder>
```

An instance document can now use any type derived from base type Address for its billingAddress and shippingAddress elements.

PurchaseOrder type uses the derived AustralianAddress type as its billingAddress and the derived AustralianPostalAddress type as its shippingAddress elements.

cphbusiness

Document Navigation and Transformation

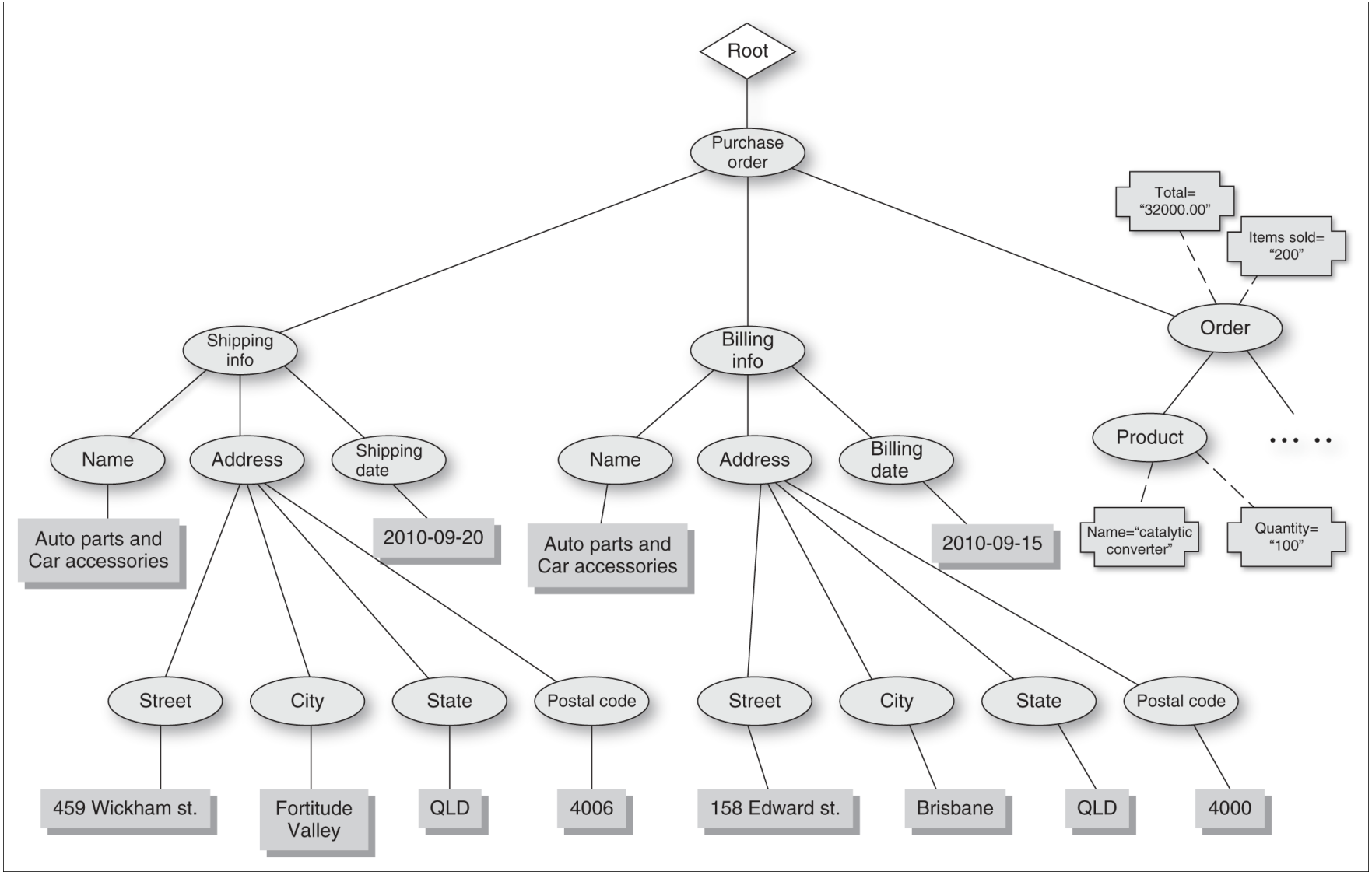
cphbusiness

Document Navigation and Transformation

- XML is primarily used to describe and contain data.
- The **eXtensible Stylesheet Language Transform** (**XSLT**) can be used to format or transform XML documents
- The XSLT process transforms an XML structure into other markup languages, such as, HTML or into any other required forms and structures
- XSLT uses the XML Path Language (**Xpath)** to address and locate sections of XML documents
- XPath is a standard for creating expressions that can be used to find specific pieces of information within an XML document
  - XPath is an abstract language that defines a tree model that codifies the logical structure of an XML document against which all expressions are evaluated

cphbusiness

XPath tree model for instance document



cphbusiness

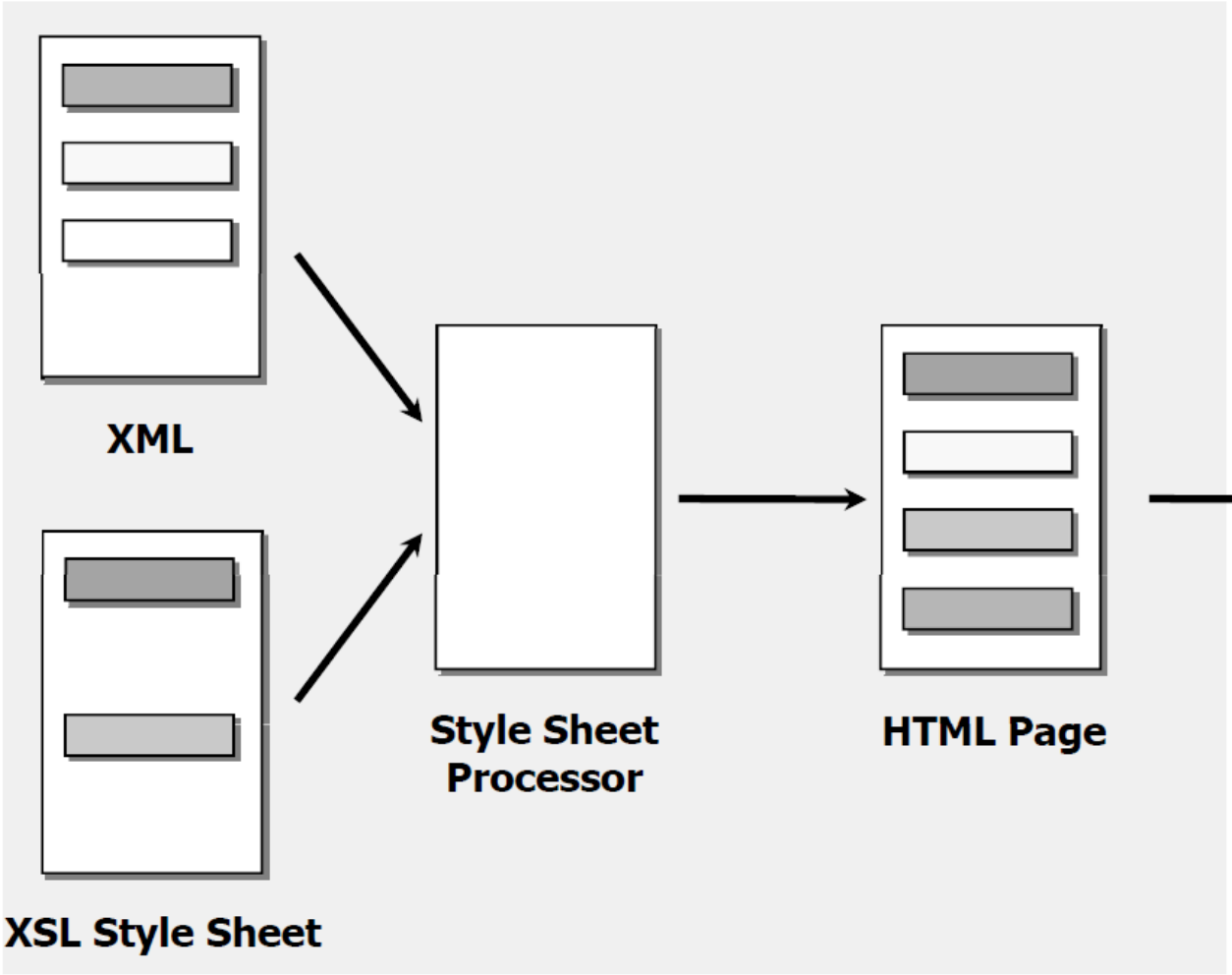
```
<?xml version="1.0" encoding="UTF-8"?>

<ns1:purchaseOrder
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:ns1='http://tempuri.org/po.xsd'
    xsi:schemaLocation='http://tempuri.org/po.xsd file:/.../XMLExamples/PurchaseOrder.xsd'>
    <ns1:shipTo country="US">
        <ns1:name>Alice Smith</ns1:name>
        <ns1:street>123 Maple Street</ns1:street>
        <ns1:city>Mill Valley</ns1:city>
        <ns1:state>CA</ns1:state>
        <ns1:zip>90952</ns1:zip>
    </ns1:shipTo>
    <ns1:billTo country="US">
        <ns1:name>Robert Smith</ns1:name>
        <ns1:street>8 Oak Avenue</ns1:street>
        <ns1:city>Old Town</ns1:city>
        <ns1:state>PA</ns1:state>
        <ns1:zip>95819</ns1:zip>
    </ns1:billTo>
    <ns1:comment>Hurry, my lawn is going wild!</ns1:comment>
    <ns1:items>
        <ns1:item partNum="872-AA">
            <ns1:productName>Lawnmower</ns1:productName>
            <ns1:quantity>1</ns1:quantity>
            <ns1:USPrice>148.95</ns1:USPrice>
            <ns1:comment>Confirm this is electric</ns1:comment>
        </ns1:item>
        <ns1:item partNum="926-AA">
            <ns1:productName>Baby Monitor</ns1:productName>
            <ns1:quantity>1</ns1:quantity>
            <ns1:USPrice>39.98</ns1:USPrice>
            <ns1:shipDate>1999-05-21</ns1:shipDate>
        </ns1:item>
    </ns1:items>
</ns1:purchaseOrder>
```

Xpath expression

```
/purchaseOrder/items/
item[@partNum='872-AA']/
child::*
```

cphbusiness

XML Style Sheet Processing



cphbusiness

Example of document transformation using XSLT

```
<PurchaseOrder>
  <Name> Plastic Products </Name>
  <billingAddress>
      <Number> 158 </Number>
      <Street> Edward st. </Street>
      <State> QLD </State>
      <PostalCode> 4000 </PostalCode>
      <Country> Australia </country>
  </billingAddress>
<PurchaseOrder>
```

Source XML application

Show in NetBeans!

Transformation service

```
<PurchaseOrder>
<Name> Plastic Products </Name>
  <billingAddress>
      <Number> 158 </Number>
       <Street> Edward st. </Street>
       <PostalCode> QLD 4000 </PostalCode>
       <Country> Australia </country>
  </billingAddress>
<PurchaseOrder>
```

Target XML application

http://www.w3schools.com/xsl/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog

cphbusiness

Simple Object Access Protocol (SOAP)

Inter-application Communication

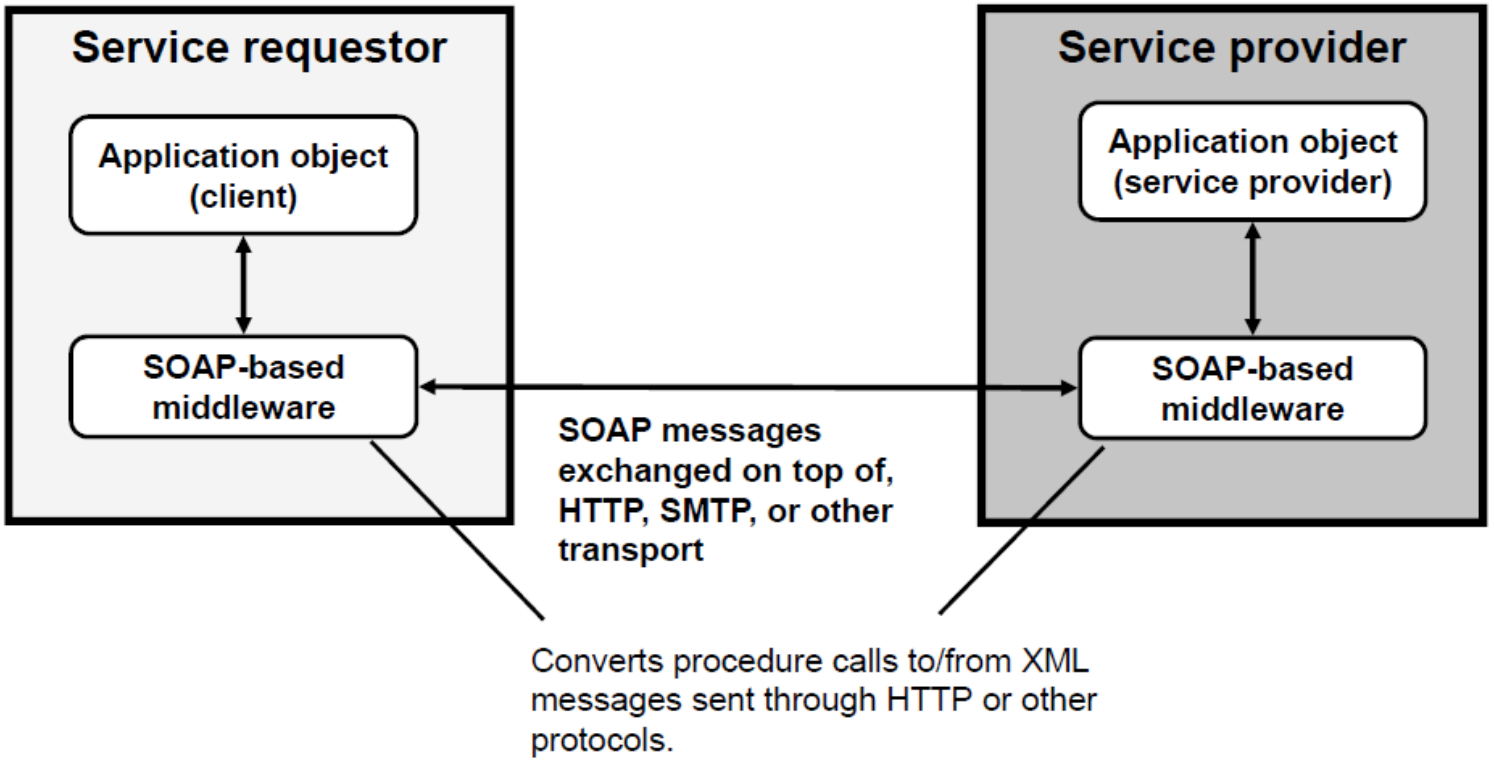
- Conventional distributed applications use distributed communication technologies, e.g., CORBA, Java/RMI, etc. based on object RPC (ORPC) protocols that attempted to marry object orientation and network protocols.
  - **ORPC request is an identifier** or symbolic name that the server could use to **locate** the target object **inside** the **server** process.
- Weaknesses:
  - **Both ends** of the communication link would need to be **implemented** under the **same** distributed **object model** (Java/RMI or CORBA/IIOP)
  - **Difficulty** of getting these protocols to work over **firewalls** or proxy servers, e.g., most firewalls are configured to allow hypertext transfer protocol (HTTP) to pass across.

Inter-application Communication

- One solution for heterogeneous infrastructures:
  - Simple Object Access Protocol (SOAP) over **HTTP**
  - **XML-based** communication protocol
  - for exchanging messages between computers **regardless** of their **operating systems**, **programming framework,** or **object model** framework

What is SOAP? – A Messaging Protocol

- SOAP … standard messaging protocol for web services
- Primarily for inter-application communication
- SOAP encodes request and response parameters in XML via HTTP for transport



What is SOAP? – A Messaging Protocol

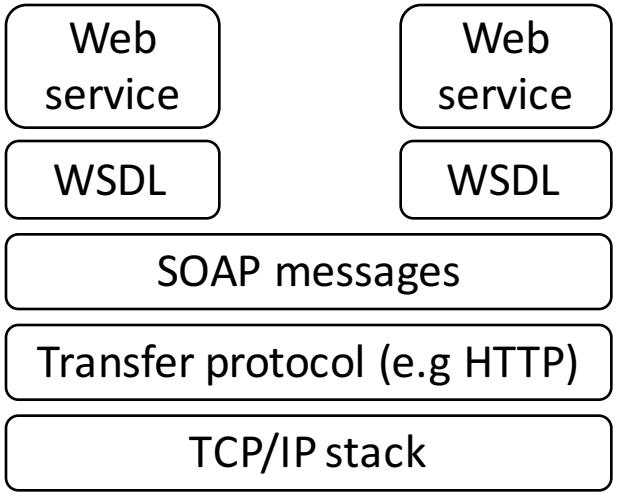
SOAP covers the following four main areas:

- **A message format** for one-way communication describing how a message can be packed into an XML document.
- A **description** of how a SOAP message should be transported using HTTP (for Web-based interaction) or SMTP (for e-mail-based interaction).
- A **set of rules** that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message.
- A **set of conventions** on how to turn an RPC call into a SOAP message and back.

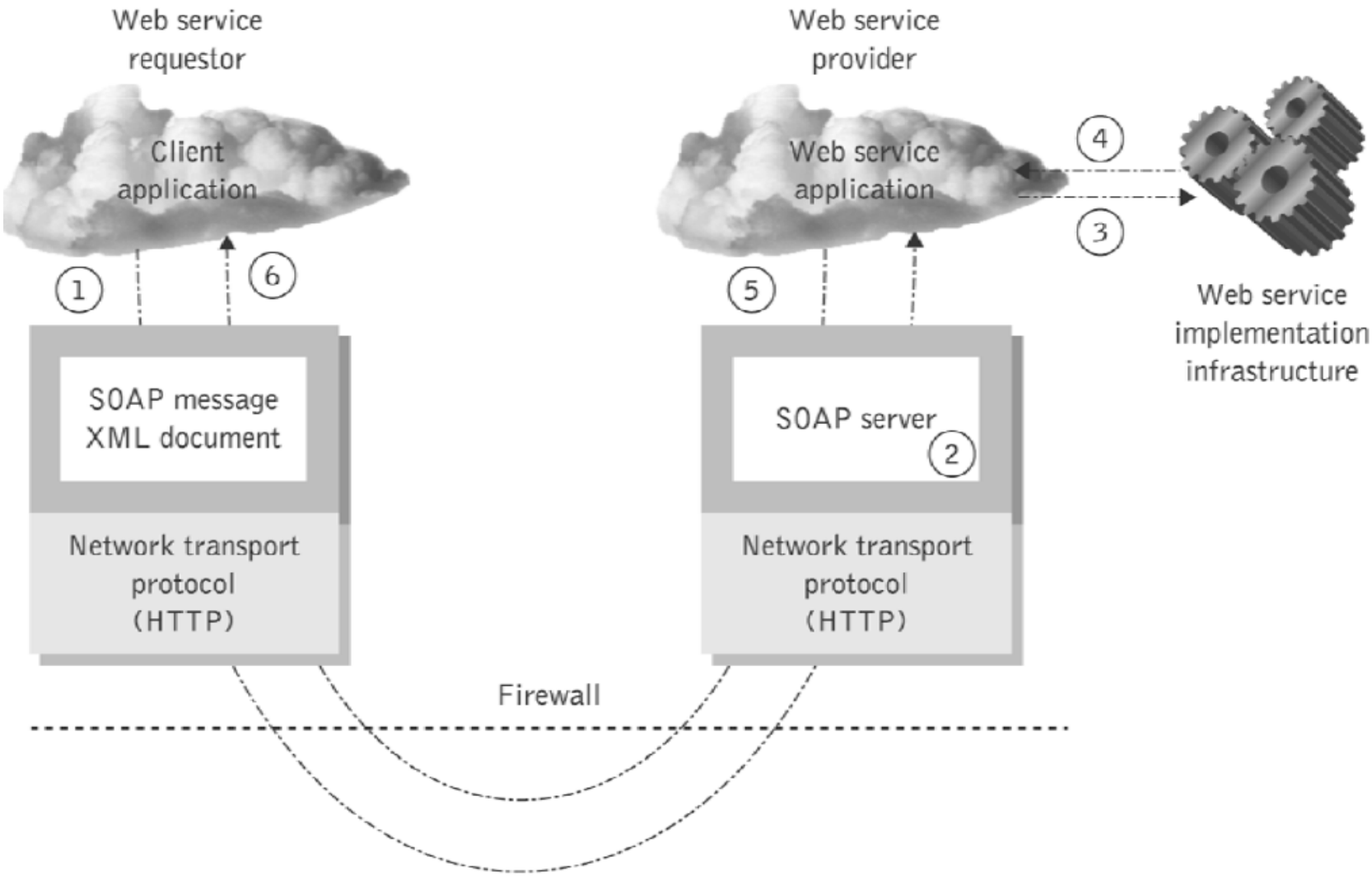
cphbusiness

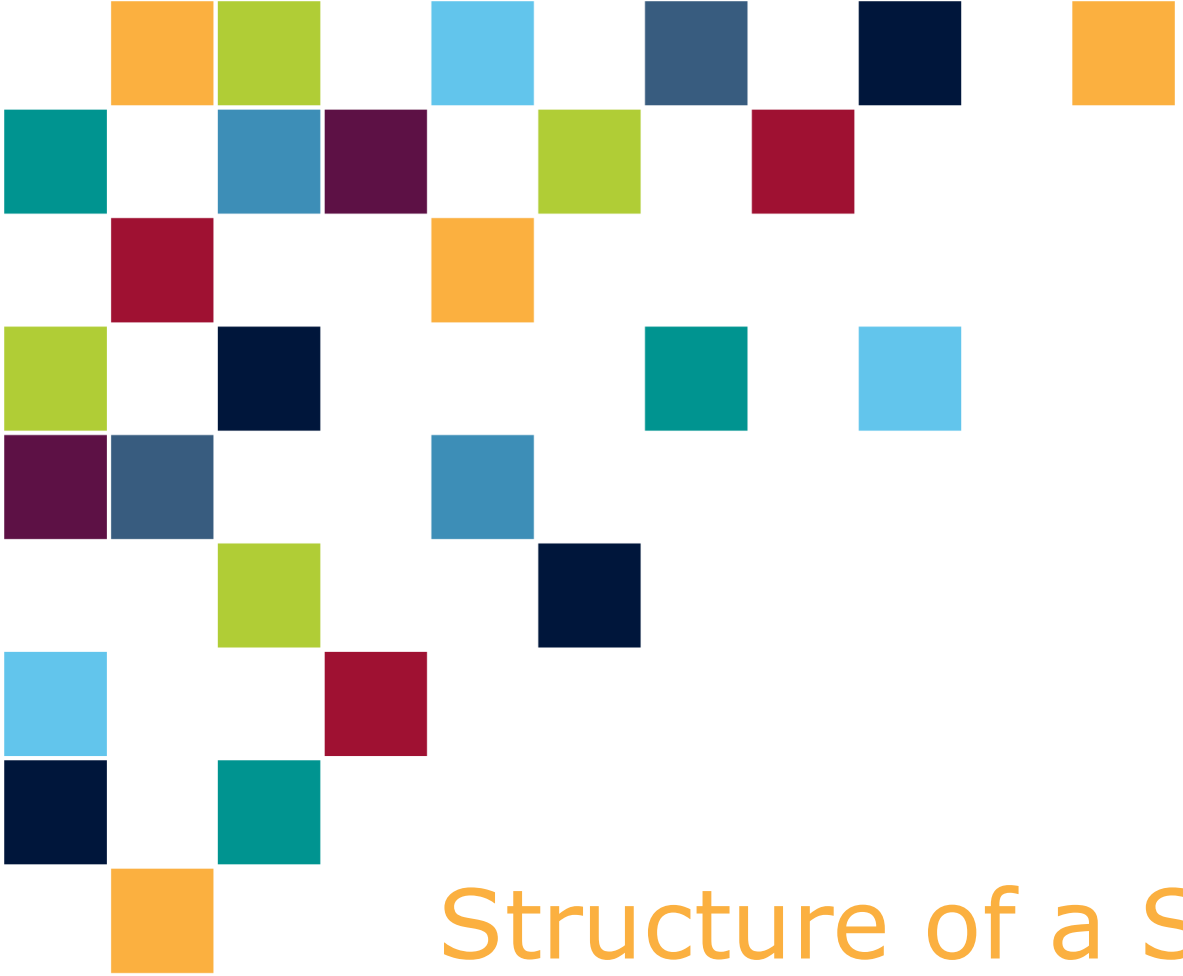
SOAP, a lightweight protocol

- SOAP allows applications to pass **messages** and **data** back and forth between disparate systems in a distributed environment, enabling remote method invocation
- Lightweight … SOAP possesses only two fundamental properties:
  - It can send and receive HTTP (or other) transport protocol packets
  - It can process XML messages
- In contrast, distributed object architecture (heavyweight) such as Internet Inter-ORB protocol, wire protocol for CORBA, specify the form or shape of data to be exchanged disparate applications/systems



Distributed Messaging Using SOAP

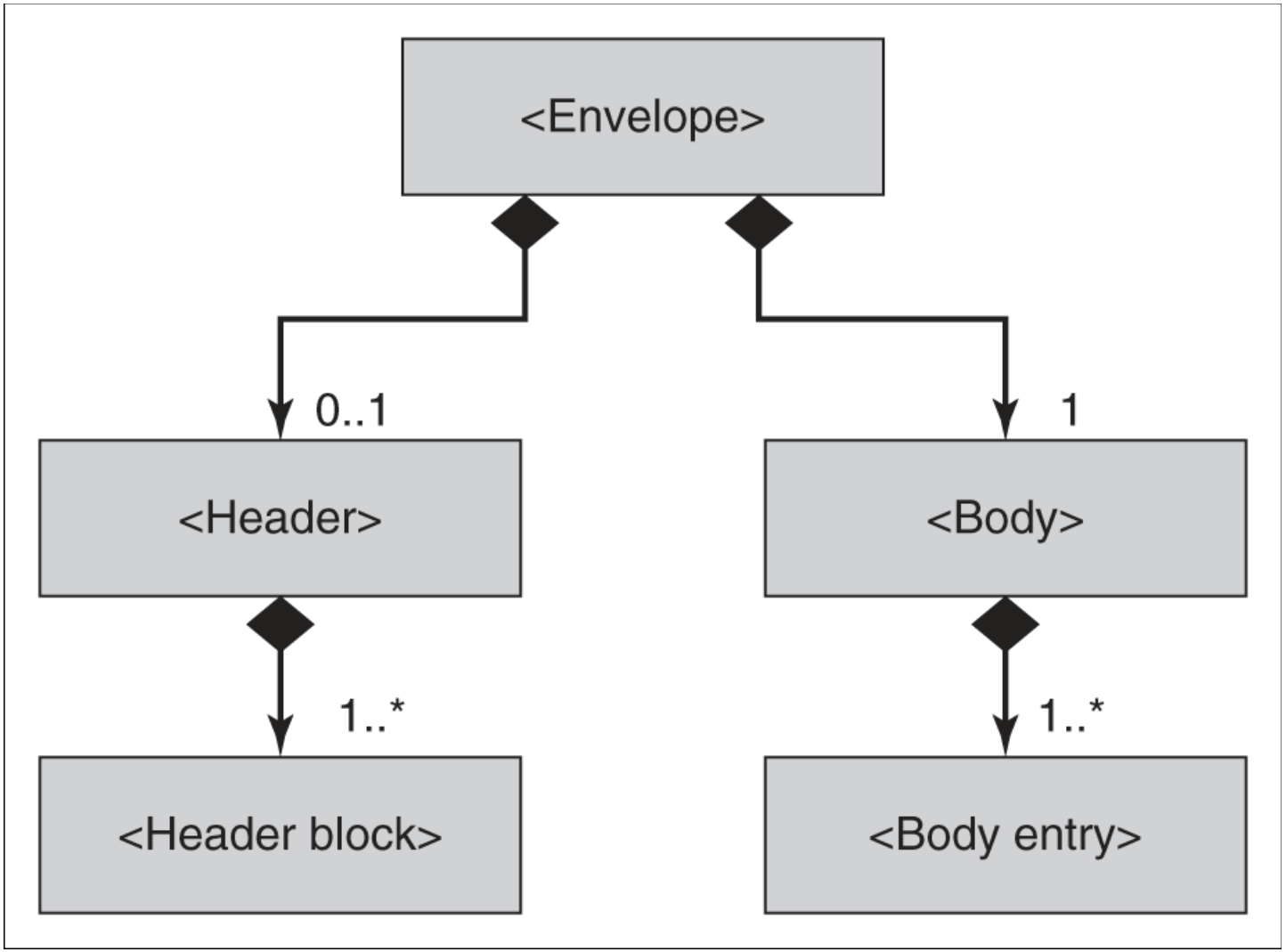




cphbusiness

Structure of a SOAP message

SOAP Messages



cphbusiness

SOAP Messages

- Messages are seen as envelopes containing the data to be sent
- A SOAP message consists of an <Envelope> element containing an optional <Header> and a mandatory <Body> element
- The contents of these elements are defined by applications.
- They are not a part of the SOAP specifications!
- A SOAP <Header> contains blocks of information relevant to how the message is to be processed. This helps pass information in SOAP messages that is not application payload
- The SOAP <Body> is where the main end-to-end information conveyed in a SOAP message must be carried

SOAP Envelope and Header

cphbusiness

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
        .....
</env:Envelope>
```

Example of SOAP envelope

<Envelope> element describes what's in the message and how to process it

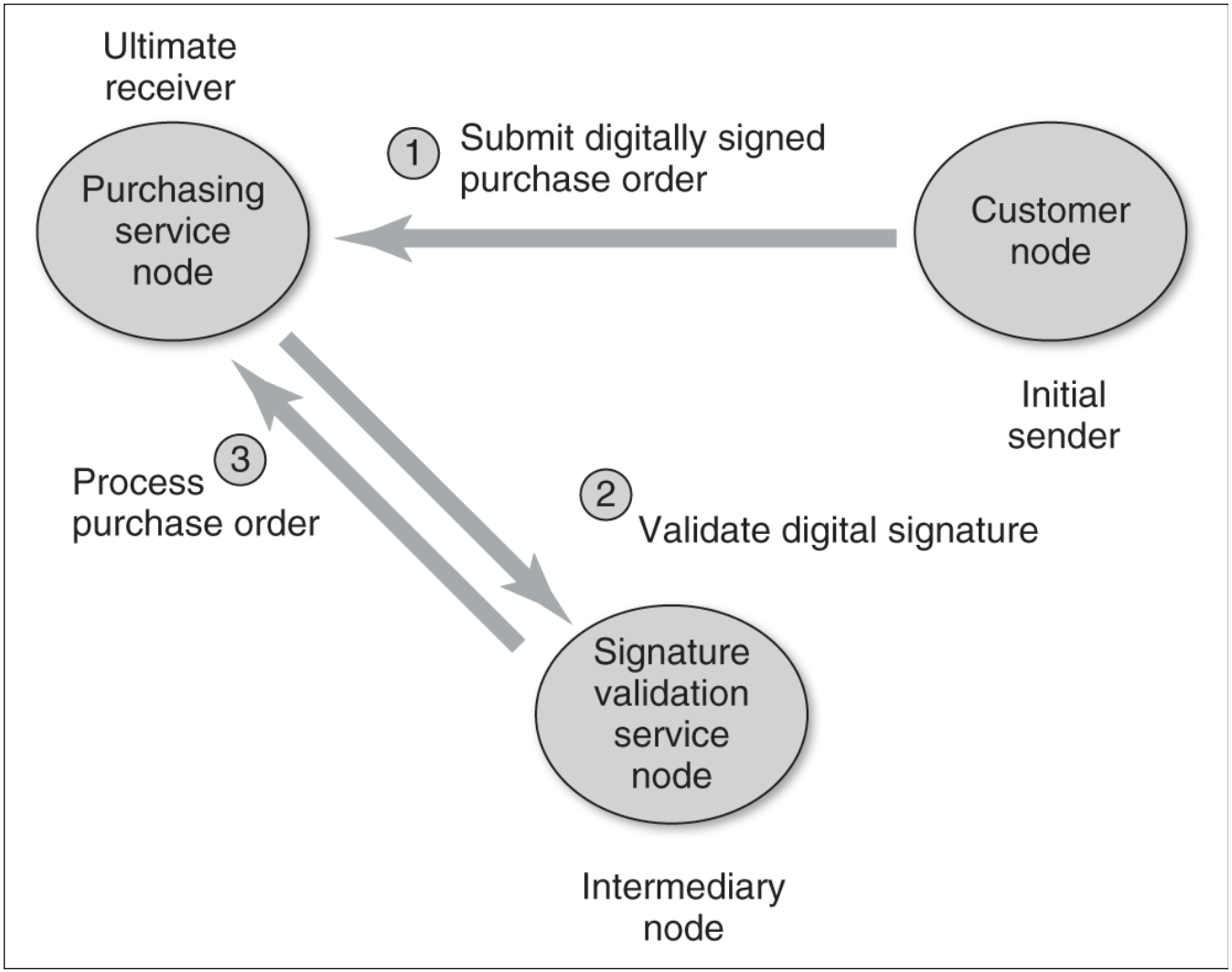
```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
      ...
    <env:Header>
       <tx:transaction-id
           xmlns:tx="http://www.transaction.com/transaction"
                 env:mustUnderstand="true">
               512
         </tx:transaction-id>
       <notary:token xmlns:notary="http://www.notarization-services.com/token"
             env:mustUnderstand="true">
                GRAAL-5YF3
       </notary:token>
    </env:Header>
        ..............
</env:Envelope>
```

Example of SOAP header

SOAP Intermediaries

- SOAP headers have been designed in anticipation of participation of other SOAP processing nodes – called **SOAP intermediaries** – along a **message's path** from an initial SOAP sender to an ultimate SOAP receiver.
- A SOAP message travels along the message path from a sender to a receiver.
- All SOAP messages start with an initial sender, which creates the SOAP message, and end with an ultimate receiver

SOAP Intermediaries



Example: SOAP Header with Message Routing

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:order
    xmlns:m="http://www.plastics_supply.com/purchase-order"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
   <m:order-no >uuid:0411a2daa</m:order-no>
   <m:date>2004-11-8</m:date>
  </m:order>
  <n:customer xmlns:n="http://www.supply.com/customers"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
     <n:name> Marvin Sanders </n:name>
  </n:customer >
 </env:Header>
 <env:Body>
    <-- Payload element goes here -->
 </env:Body>
</env:Envelope>
```

cphbusiness

The SOAP Body

- The SOAP body is the area of the SOAP message, containing the application specific XML data (payload)
- **<Body> element**
  - must be present
  - is immediate child of the envelope
  - may contain a number of child elements, called body entries
  - may also be empty
- **<Body> element** contains either:
  - **Application-specific data**
    - the information exchanged with a web service
    - the method call information and its related arguments
    - the response to a method call
  - A **fault message** is used only when an error occurs
- A SOAP message may carry either application-specific data or a fault, but not both!

Example: SOAP Message

cphbusiness

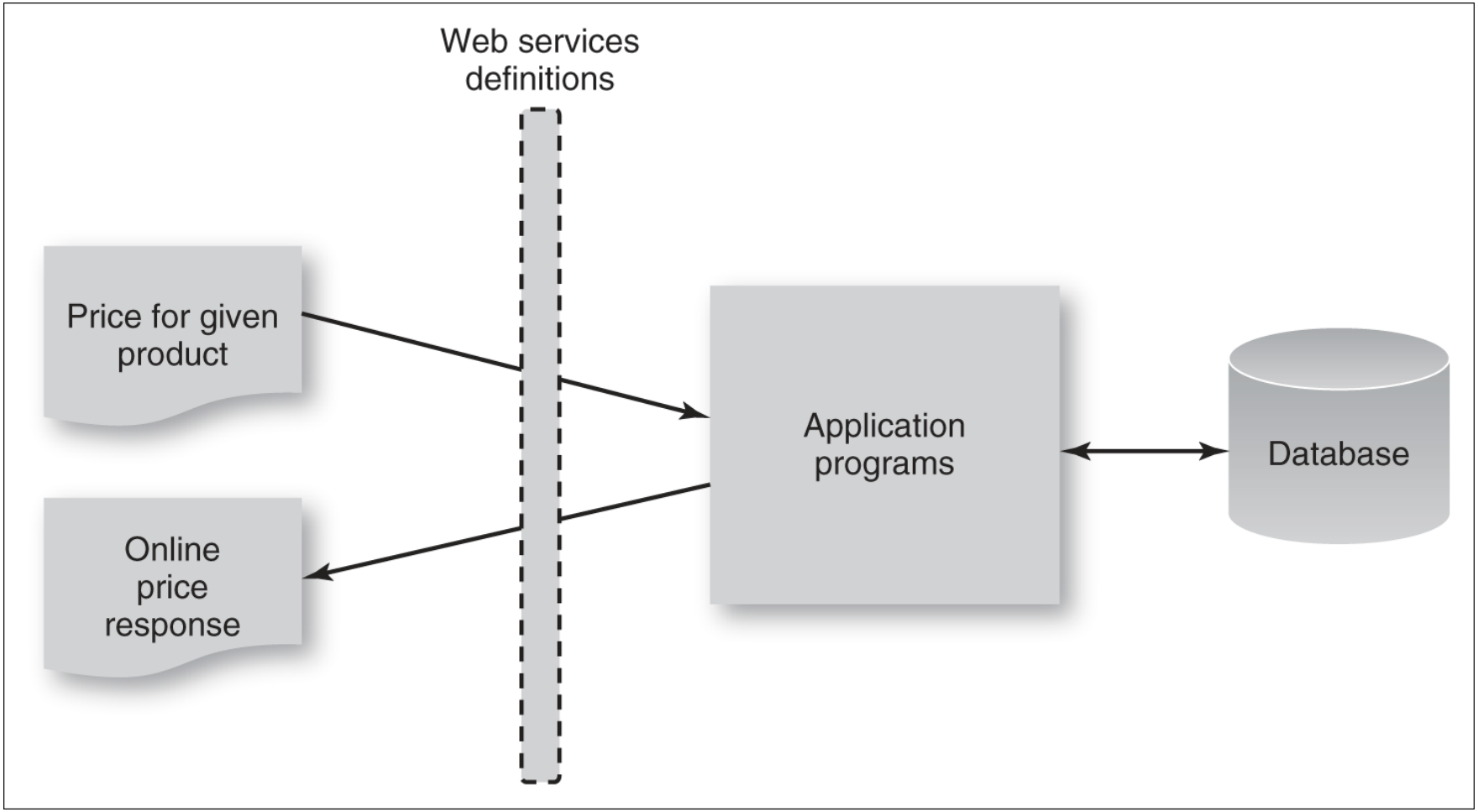
SOAP Communication Model

The SOAP Communication Model

- SOAP supports two possible communication styles:
  - remote procedure call (RPC) and
  - document (or message)

cphbusiness

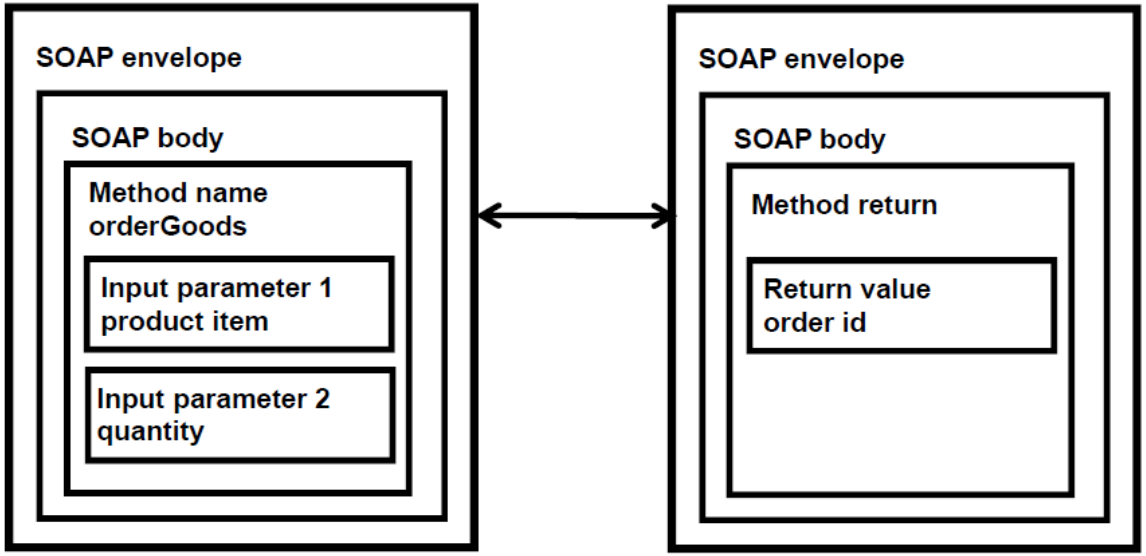
RPC-style SOAP Services



cphbusiness

RPC-style SOAP Services

- An RPC-style web service appears as a remote object to a client application.
- Interaction between a client and an RPC-style web service via a service-specific interface
- Clients express request as a method call with a signature responding with a return value



```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
          <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
              env:mustUnderstand='1'>
               512
          </tx:Transaction-id>
    </env:Header>
    <env:Body>
          <m:GetProductPrice>
          <product-id>  450R6OP  </product-id >
          </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

Method name

Method parameter

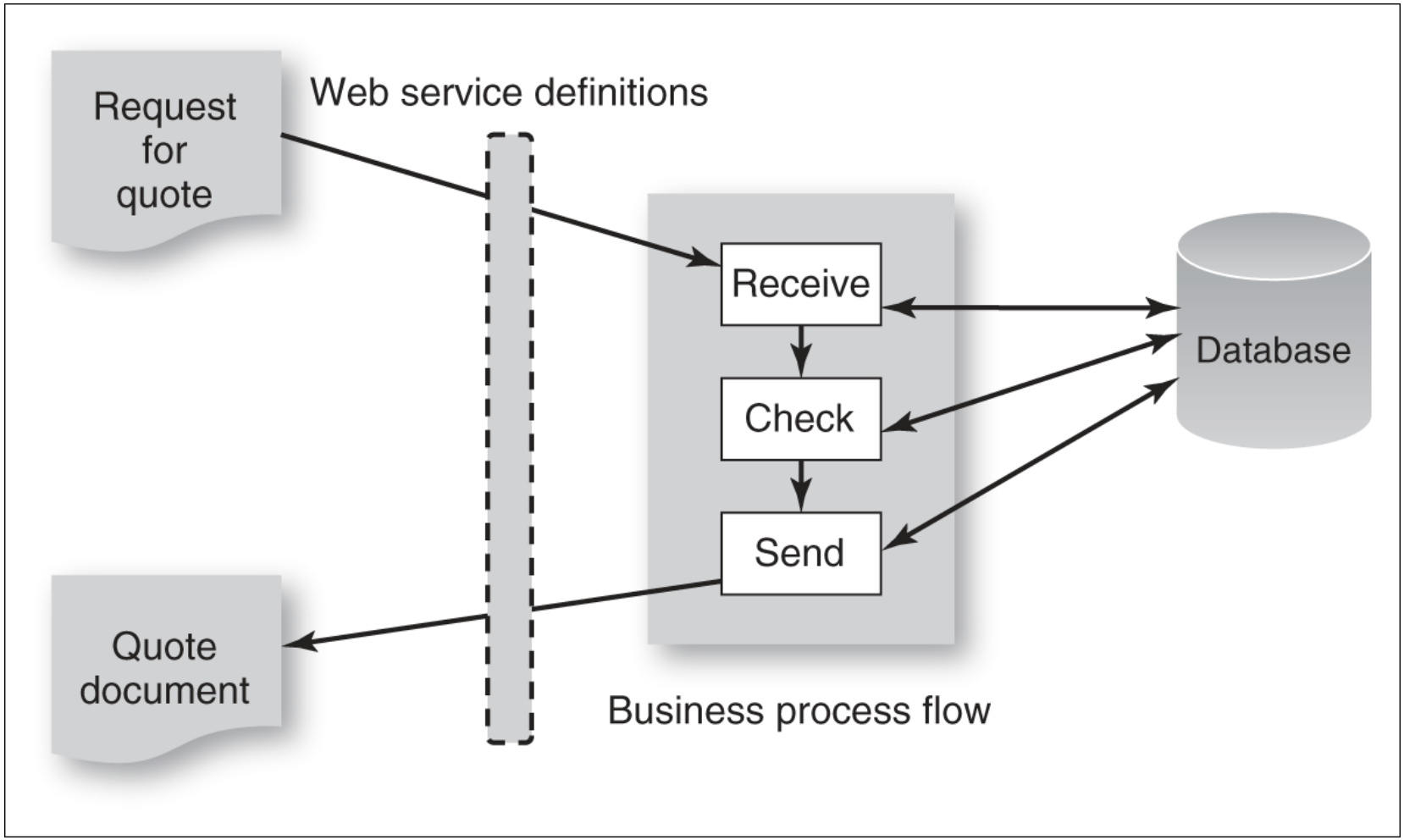
Listing 4.5: Example of RPC-style SOAP body

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
          <--! - Optional context information -->
    </env:Header>
    <env:Body>
          <m:GetProductPriceResponse>
             <product-price>  134.32  </product-price>
          </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

Returned value

Listing 4.6: Example of RPC-style SOAP response message

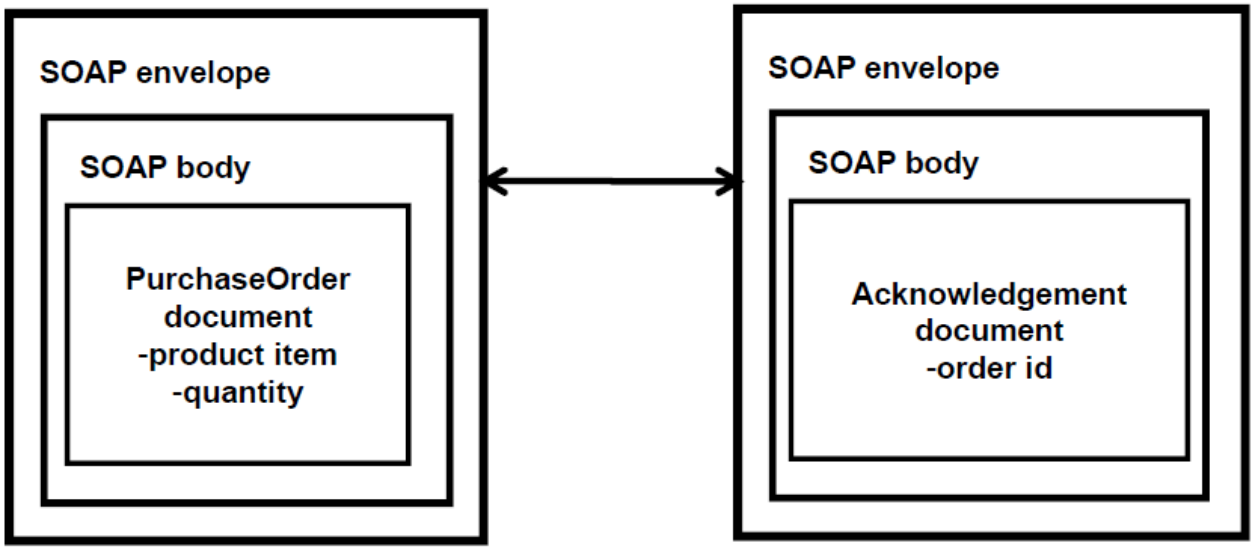
Document (Message)-style SOAP Services



cphbusiness

Document (Message)-style SOAP Services

- SOAP <Body> contains an XML document fragment without explicit XML structure.
- SOAP run-time environment accepts the SOAP <Body> as it
- Delivers it unchanged to the target application
- Response to messages are optional



cphbusiness

Example: of Document-style SOAP Service

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>
          <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
              env:mustUnderstand='1'>
               512
    </env:Header>
    <env:Body>
          <po:PurchaseOrder oderDate="2004-12-02"
           xmlns:m="http://www.plastics_supply.com/POs">
           <po:from>
             <po:accountName>   RightPlastics   </po:accountName>
                <po:accountNumber>   PSC-0343-02  </po:accountNumber>
           </po:from>
           <po:to>
             <po:supplierName>  Plastic Supplies Inc.  </po:supplierName>
             <po:supplierAddress>  Yara Valley Melbourne  </po:supplierAddress>
           </po:to>
           <po:product>
               <po:product-name> injection molder </po:product-name>
               <po:product-model> G-100T </po:product-model>
               <po:quantity> 2 </po:quantity>
           </po:product>
          </ po:PurchaseOrder >
    </env:Body>
</env:Envelope>
```

Example of document-style SOAP body

cphbusiness

SOAP Fault Message

cphbusiness

SOAP Fault Message

- SOAP provides a model for error handling
- It distinguishes between:
  - Conditions that result in an error
  - Information about the the type of error for message source or another node
- Error messages are placed in the SOAP <Body>
  - In special-purpose element called <env:Fault> (predefined in specification)
- <env:Fault> element contains two sub-elements:
  - <env:Code>
    - Required <env:Value> with type of error
    - Optional <env:Subcode> futher details about error type
  - <env:Reason>
    - human-readable error description
  - optional <env:Detail> containing application-specific error information

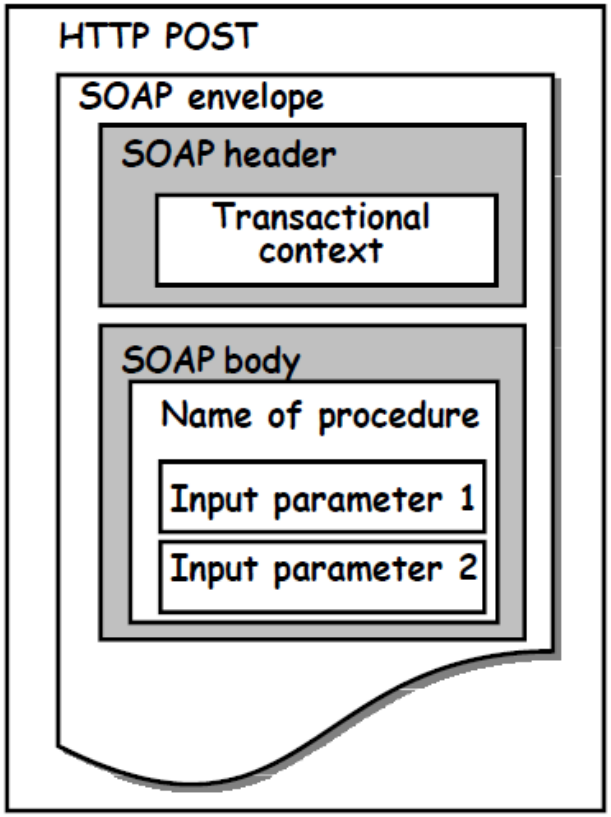

```
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
          <tx:Transaction-id
           xmlns:t="http://www.transaction.com/transactions"
              env:mustUnderstand='1'>
               512
          </tx:Transaction-id>
    </env:Header>
    <env:Body>
     <env:Fault>
       <env:Code>
            <env:Value>env:Sender</env:Value>
            <env:Subcode>
               <env:Value>m:InvalidPurchaseOrder</env:Value>
            </env:Subcode>
       </env:Code>
       <env:Reason>
            <env:Text xml:lang="en-UK">Specified product did not exist</env:Text>
       </env:Reason>
       <env:Detail>
          <err:myFaultDetails
             xmlns:err="http://www.plastics_supply.com/faults">
            <err:message> Product number contains invalid characters
</err:message>
            <err:errorcode> 129 </err:errorcode>
          </err:myFaultDetails>
       </env:Detail>
     </env:Fault>
    </env:Body>
</env:Envelope>
```

cphbusiness

SOAP over HTTP

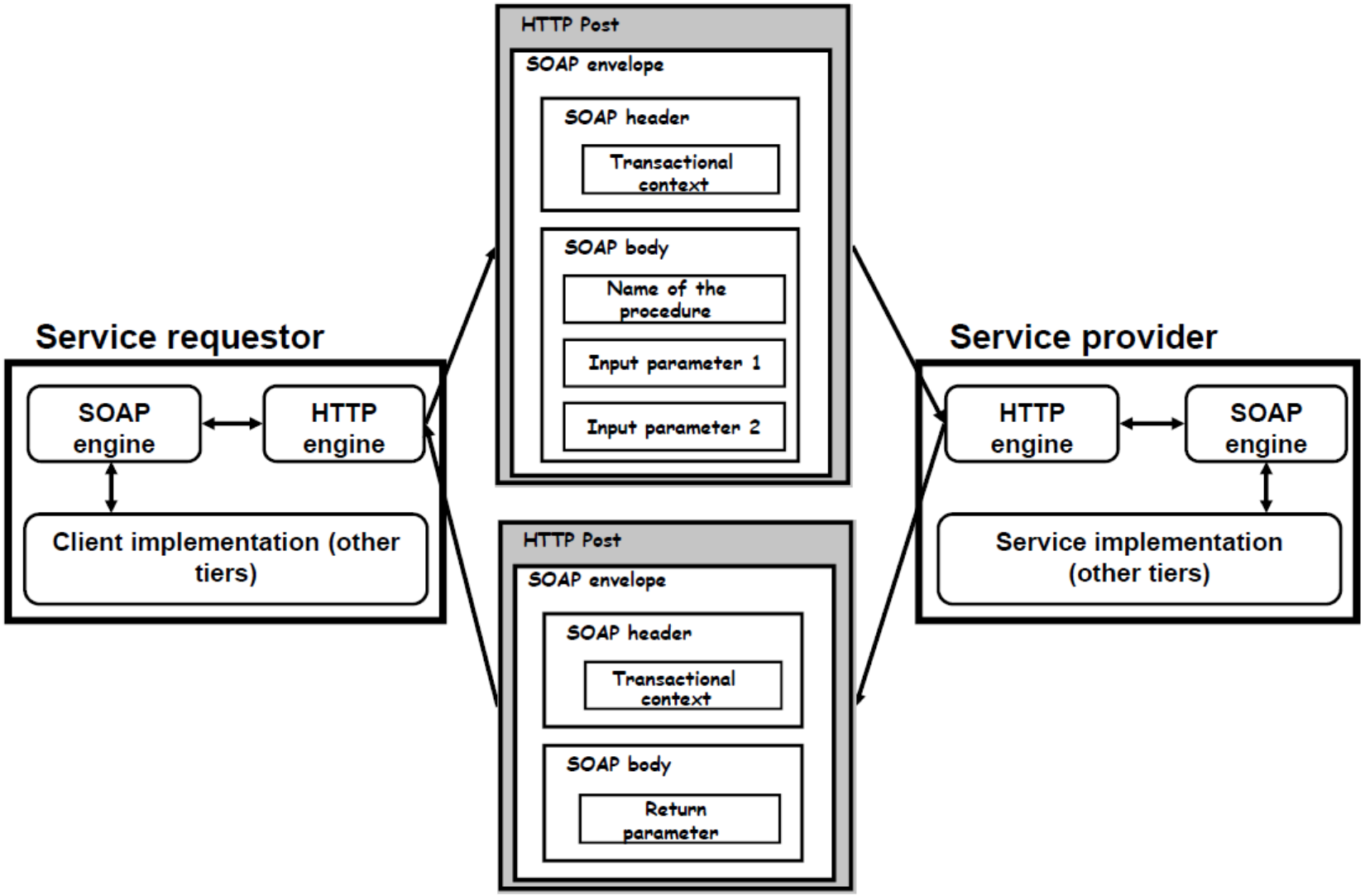
SOAP and HTTP

- A binding of SOAP to a transport protocol is a description of how a SOAP message is to be sent using that transport protocol
- Typical binding for SOAP is HTTP
- SOAP can use HTTP GET or POST requests
- GET request:
  - the request is not a SOAP message
  - the SOAP response is a SOAP message
- POST request:
  - both request and response are SOAP message
- SOAP uses the same error and status codes as those used in HTTP
  - HTTP responses can be directly interpreted by a SOAP module



cphbusiness

SOAP RPC via HTTP Post Request



```
POST /Purchase Order HTTP/1.1
Host: http://www.plastics_supply.com   <! - Service provider -- >
Content-Type:application/soap+xml;
charset = "utf-8"
Content-Length: nnnn

<?xml version="1.0" ?>
<env:Envelope
 xmlns:SOAP="http://www.w3.org/2003/05/soap-envelope"
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
        <tx:Transaction-id
            xmlns:t="http://www.transaction.com/transactions"
             env:mustUnderstand='1'>
                512
        </tx:Transaction-id>
    </env:Header>
    <env:Body>
        <m:GetProductPrice>
           <product-id>  450R6OP  </product-id >
        </m:GetProductPrice >
    </env:Body>
</env:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type:application/soap+xml;
charset = “utf-8”
Content-Length: nnnn

<?xml version="1.0" ?>
<env:Envelope
 xmlns:SOAP=“http://www.w3.org/2003/05/soap-envelope”
 xmlns:m="http://www.plastics_supply.com/product-prices">
    <env:Header>
       <--! - Optional context information -->
    </env:Header>
    <env:Body>
       <m:GetProductPriceResponse>
          <product-price>   134.32  </product-price>
       </m:GetProductPriceResponse>
    </env:Body>
</env:Envelope>
```

cphbusiness

SOAP Pros & Cons

cphbusiness

Advantages

- Simplicity: Based on highly-structured format of XML
- Portability: No dependencies on underlying platform
- Firewall friendliness: By posting data over HTTP
- Use of open standards: text-based XML standard
- Interoperability: Built on open technologies (XML and HTTP)
- Universal acceptance. Most widely accepted message communication standard … hmm

cphbusiness

Disadvantages

- Too much reliance on HTTP: limited only to request/response model
- Statelessness: difficult for transactional and business processing applications
- Serialization by value and not by reference: impossible to refer or point to external data source
- Slow HTTP protocol causes bad performance

Application	*"calculator"*		*"data"*		*"large data"*	
Technology	**Client CPU**	**Server CPU**	**Client CPU**	**Server CPU**	**Client CPU**	**Server CPU**
WS	15.0s	6s	22.8s	8.4s	1087s	551s (436s)
Java-RMI	2.3s	0.8s	4s	1.1s	148s	212s (97s)
CORBA	3.2s	1.9s	3.6s	2.1s	54.2s	250s (136s)

http://mercury.it.swin.edu.au/ctg/AWSA04/Papers/gray.pdf

cphbusiness

WSDL & UDDI

cphbusiness

Outline

- Why is a service description needed?
- Web Service Description Language (WSDL)
- Difference between web service interface definition and web service implementation
- WSDL message exchange patterns
- Service registries and discovery
- Universal Description, Discovery and Integration (UDDI)

cphbusiness

Why is a service description needed?

Why is a service description needed?

- Web services must be **defined** in a **consistent** manner to be (automatically) **discovered** and **used** by other services and applications
- Web service consumers must determine the precise XML interface of a web service

Why is a service description needed?

- **Service description** reduces the amount of required common understanding and custom programming and integration:
  - It is a machine understandable standard describing the operations of a web service.
  - It specifies the wire format and transport protocol that the web service uses to expose this functionality.
  - It can also describe the payload data using a type system.
- **Service description + SOAP infrastructure isolates** all technical details, e.g., machine- and implementation language-specific elements, away from the service requestor's application and the service provider's web service.

cphbusiness

Web Service Description Language (WSDL)

cphbusiness

Web Services Description Language

- … is an XML-based language for service representation
- … used to describe the details of the complete a web services' interfaces
- … provides means to access a web service
- … allows for encapsulation: neither the service requestor nor the provider are aware of each other's technical infrastructure, programming language, etc.

cphbusiness

Web Services Description Language

- … is platform and language independent
- … primarily used to describe SOAP-enabled services
- Essentially, WSDL is used to describe precisely:
  - ***What*** a service does, i.e., the operations the service provides,
  - ***Where*** it resides, i.e., details of the protocol-specific address, e.g., a URL
  - ***How*** to invoke it, i.e., details of the data formats and protocols necessary to access the service's operations

cphbusiness

```
<definitions  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.web.credit.bank.org/"  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"  targetNamespace="http://service.web.credit.bank.org/"
name="CreditScoreService">
  <types>
    <xsd:schema>
      <xsd:import  namespace="http://service.web.credit.bank.org/"
schemaLocation="http://139.59.154.97:8080/CreditScoreService/CreditScoreService?xsd=1"/>
    </xsd:schema>
  </types>
  <message  name="creditScore">
    <part  name="parameters"  element="tns:creditScore"/>
  </message>
  <message  name="creditScoreResponse">
    <part  name="parameters"  element="tns:creditScoreResponse"/>
  </message>
  <portType  name="CreditScoreService">
    <operation  name="creditScore">
      <input  wsam:Action="http://service.web.credit.bank.org/CreditScoreService/creditScoreRequest"
message="tns:creditScore"/>
      <output  wsam:Action="http://service.web.credit.bank.org/CreditScoreService/creditScoreResponse"
message="tns:creditScoreResponse"/>
    </operation>
  </portType>
  <binding  name="CreditScoreServicePortBinding"  type="tns:CreditScoreService">
    <soap:binding  transport="http://schemas.xmlsoap.org/soap/http"  style="document"/>
      <operation  name="creditScore”>
        <soap:operation  soapAction=""/>
          <input>
            <soap:body  use="literal"/>
          </input>
          <output>
            <soap:body  use="literal"/>
          </output>
      </operation>
  </binding>
  <service  name="CreditScoreService">
    <port  name="CreditScoreServicePort"  binding="tns:CreditScoreServicePortBinding">
      <soap:address  location="http://139.59.154.97:8080/CreditScoreService/CreditScoreService"/>
    </port>
  </service>
</definitions>
```

What is provided?

How is it invoked?

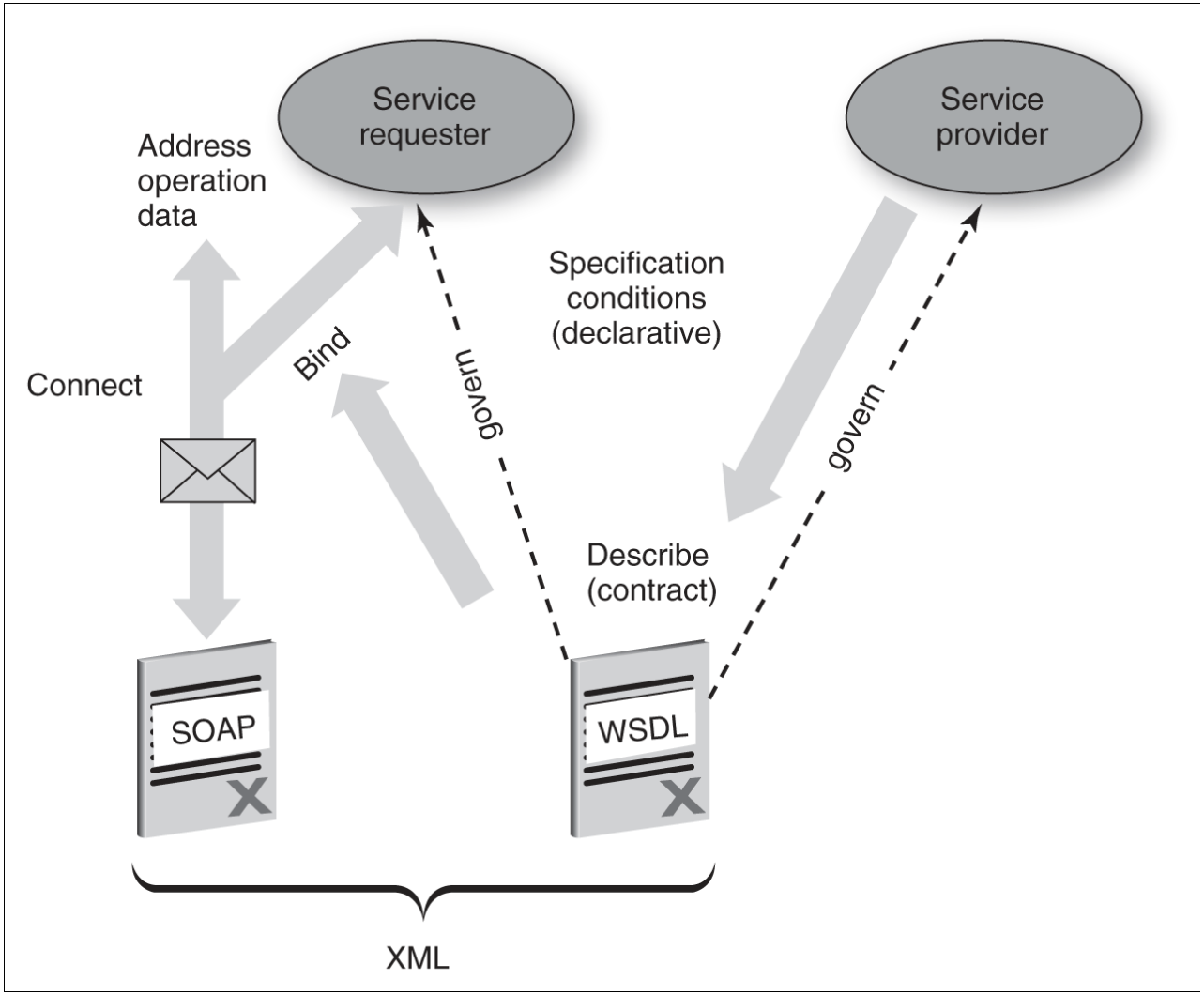
Where is it to be found?

WSDL Descriptions as Contracts

- A WSDL web service description defines the way of interacting with a particular web service.
- WSDL web service descriptions constrain both the service provider and the service requestor.
- Consequently, it represents a "**contract**" between the service requestor and the service provider

cphbusiness

WSDL Descriptions as Contracts



cphbusiness

Difference between web service interface definition and web service implementation

cphbusiness

Structure of WSDL documents

- WSDL documents can be separated into distinct sections:
  - The ***service-interface definition*** (abstract interface) describes the general Web service interface structure. This contains all the operations supported by the service, the operation parameters, and abstract data types.
  - The ***service implementation part*** (concrete endpoint) binds the abstract interface to a concrete network address, to a specific protocol, and to concrete data structures.
- A web service client may bind to such an implementation and invoke the service in question.
- This enables each part to be defined separately and independently, and **reused** by other parts.
- The combination of these two parts contains **sufficient information** to describe to the service requestor how to invoke and interact with the web service at a provider's site.
  - Using WSDL, a requestor can locate a web service and invoke any of the publicly available operations.

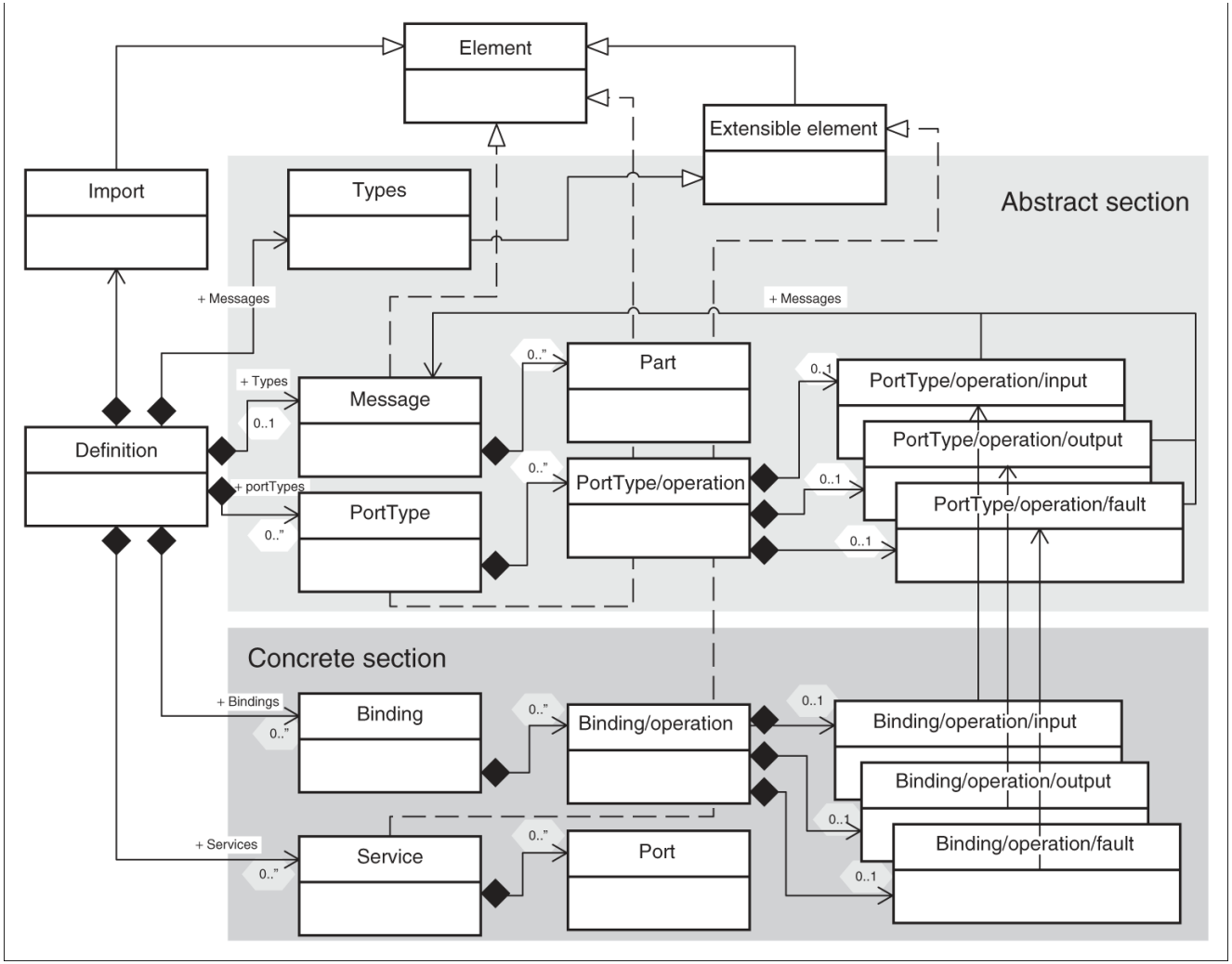
cphbusiness

WSDL Document Contents

- Abstract (interface) definitions
  - <types> data type definitions
  - <message> operation parameters
  - <portType> set of operation definitions
    - <operation> abstract description of service actions
- Concrete (implementation) definitions
  - <binding> operation bindings
  - <port> association of an endpoint with a binding
  - <service> location/address for each binding
- Also:
  - <import> used to reference other XML documents

cphbusiness

WSDL Abstract Sytnax



cphbusiness

```
<definitions  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns:wsp="http://www.w3.org/ns/ws-policy"  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://service.web.credit.bank.org/"  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/"  targetNamespace="http://service.web.credit.bank.org/"
name="CreditScoreService">
  <types>
    <xsd:schema>
      <xsd:import  namespace="http://service.web.credit.bank.org/"
schemaLocation="http://139.59.154.97:8080/CreditScoreService/CreditScoreService?xsd=1"/>
    </xsd:schema>
  </types>
  <message  name="creditScore">
    <part  name="parameters"  element="tns:creditScore"/>
  </message>
  <message  name="creditScoreResponse">
    <part  name="parameters"  element="tns:creditScoreResponse"/>
  </message>
  <portType  name="CreditScoreService">
    <operation  name="creditScore">
      <input  wsam:Action="http://service.web.credit.bank.org/CreditScoreService/creditScoreRequest"
message="tns:creditScore"/>
      <output  wsam:Action="http://service.web.credit.bank.org/CreditScoreService/creditScoreResponse"
message="tns:creditScoreResponse"/>
    </operation>
  </portType>
  <binding  name="CreditScoreServicePortBinding"  type="tns:CreditScoreService">
    <soap:binding  transport="http://schemas.xmlsoap.org/soap/http"  style="document"/>
      <operation  name="creditScore”>
        <soap:operation  soapAction=""/>
          <input>
            <soap:body  use="literal"/>
          </input>
          <output>
            <soap:body  use="literal"/>
          </output>
      </operation>
  </binding>
  <service  name="CreditScoreService">
    <port  name="CreditScoreServicePort"  binding="tns:CreditScoreServicePortBinding">
      <soap:address  location="http://139.59.154.97:8080/CreditScoreService/CreditScoreService"/>
    </port>
  </service>
</definitions>
```

Abstract Definitions

Concrete Definitions

cphbusiness

Web Service Interface Definition

- WSDL specifies a grammar and syntax that describes Web services as a collection of communicating endpoints.
- A complete WSDL definition contains all of the information necessary to invoke a web service.
- Exchanged data (between the **endpoints**) is specified as part of messages
- Processing activity at endpoints is considered as an operation
- WSDL provides the means to group messages into operations and operations into interfaces.
  - Collections of permitted operations at an endpoint are grouped together into **port types.**

```
<wsdl:definitions name="PurchaseOrderService"
   targetNamespace="http://supply.com/PurchaseService/wsdl"
   xmlns:tns="http://supply.com/ PurchaseService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://supply.com/PurchaseService/wsdl"
      <xsd:complexType name="CustomerInfoType">
          <xsd:sequence>
             <xsd:element name="CusNamer" type="xsd:string"/>
             <xsd:element name="CusAddress" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="POType">
          <xsd:sequence>
             <xsd:element name="PONumber" type="integer"/>
             <xsd:element name="PODate" type="string"/>
          </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="InvoiceType">
          <xsd:all>
             <xsd:element name="InvPrice" type="float"/>
             <xsd:element name="InvDate" type="string"/>
          </xsd:all>
       </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
  </wsdl:message>
  <wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
  </wsdl:message>
  <wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
       </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Abstract data type definitions

Listing 1:
Example of WSDL interface definition describing a purchase order service

Data that is sent

Data that is returned

Port type with one operation

An operation with request (input) & response (output) message

<types> Element

- The WSDL **<types>** element serves as a container that contains all abstract data types that define a Web service interface.
- A <types> element in WSDL is comparable to a data type in Java
  - WSDL uses a few primitive data types that XML schema definition (XSD) defines, e.g., **int, float, long, short, string, boolean**,
  - Or allows developers to build complex data types based on primitive types before using them in messages.
  - The data types and elements defined in the <types> element are used by message definitions when declaring the parts (payloads) of messages.
  - Any complex data type that the service uses must be defined using a <types> element.
- Listing illustrates three complex types :

  `CustomerInfoType, POType and InvoiceType`

```
<wsdl:definitions name="PurchaseOrderService"
   targetNamespace="http://supply.com/PurchaseService/wsdl"
   xmlns:tns="http://supply.com/ PurchaseService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://supply.com/PurchaseService/wsdl"
      <xsd:complexType name="CustomerInfoType">
          <xsd:sequence>
             <xsd:element name="CusNamer" type="xsd:string"/>
             <xsd:element name="CusAddress" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="POType">
          <xsd:sequence>
             <xsd:element name="PONumber" type="integer"/>
             <xsd:element name="PODate" type="string"/>
          </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="InvoiceType">
          <xsd:all>
             <xsd:element name="InvPrice" type="float"/>
             <xsd:element name="InvDate" type="string"/>
          </xsd:all>
       </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
  </wsdl:message>
  <wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
  </wsdl:message>
  <wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
       </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Abstract data type definitions

Listing 1:
Example of WSDL interface definition describing a purchase order service

Data that is sent

Data that is returned

Port type with one operation

An operation with request (input) & response (output) message

<message> Element

- … describes the payload of a message used by a web service.
- … consists of <part> elements, which are linked to <types> elements
  - Each message can consist of one or more parts.
  - Parts are similar to parameters of a function call in a programming language
- In Listing, the PurchaseOrder service defines <message> elements to describe the parameters and return values of that service.
  - POMessage describes the input parameters of the service, while
  - InvMessage represents the return (output) parameters.

```
<wsdl:definitions name="PurchaseOrderService"
   targetNamespace="http://supply.com/PurchaseService/wsdl"
   xmlns:tns="http://supply.com/ PurchaseService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://supply.com/PurchaseService/wsdl"
      <xsd:complexType name="CustomerInfoType">
          <xsd:sequence>
             <xsd:element name="CusNamer" type="xsd:string"/>
             <xsd:element name="CusAddress" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="POType">
          <xsd:sequence>
             <xsd:element name="PONumber" type="integer"/>
             <xsd:element name="PODate" type="string"/>
          </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="InvoiceType">
          <xsd:all>
             <xsd:element name="InvPrice" type="float"/>
             <xsd:element name="InvDate" type="string"/>
          </xsd:all>
       </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
  </wsdl:message>
  <wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
  </wsdl:message>
  <wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
       </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Abstract data type definitions

Listing 1:
Example of WSDL interface definition describing a purchase order service

Data that is sent

Data that is returned

Port type with one operation

An operation with request (input) & response (output) message

cphbusiness

<message> Element

```
<!-- message elements that describe input and output parameters for the
    PurchaseOrderService -->
<!--input message  -->
<wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      <wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
</wsdl:message>
<! -- outputput message  -->
<wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
</wsdl:message>
```

RPC-style message

```
<!-- message element that describes input and output parameters -->
<wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" element="tns:PurchaseOrder"/>
</wsdl:message>
```

Document-style message

<portType>, <operation> elements

- A **<portType>** element defines an abstract type and its operations but not an implementation.
- A **<portType>** element is a logical grouping of **<operation>**s in a web service
  - It describes the kinds of operations that a web service supports –the messaging mode and payloads– without specifying the Internet protocol or physical address used.
  - **The <portType> element is central to a WSDL description; the rest of the elements in the definition are essentially details that the <portType> element depends upon.**
  - The <portType> element can be compared to a **function library** (or a module, or a class) in a programming language

cphbusiness

<operation> elements

- Operations in WSDL represent the methods exposed by the service: they include the name of the method and the input and output parameters.
  - A typical <operation> element is composed of at most one <input> or <output> element and any number of <fault> elements.
- The WSDL example in Listing 1 contains a <portType> named PurchaseOrderPortType that supports a single <operation> called SendPurchase

```
<wsdl:definitions name="PurchaseOrderService"
   targetNamespace="http://supply.com/PurchaseService/wsdl"
   xmlns:tns="http://supply.com/ PurchaseService/wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://supply.com/PurchaseService/wsdl"
      <xsd:complexType name="CustomerInfoType">
          <xsd:sequence>
             <xsd:element name="CusNamer" type="xsd:string"/>
             <xsd:element name="CusAddress" type="xsd:string"/>
          </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="POType">
          <xsd:sequence>
             <xsd:element name="PONumber" type="integer"/>
             <xsd:element name="PODate" type="string"/>
          </xsd:sequence>
       </xsd:complexType>
       <xsd:complexType name="InvoiceType">
          <xsd:all>
             <xsd:element name="InvPrice" type="float"/>
             <xsd:element name="InvDate" type="string"/>
          </xsd:all>
       </xsd:complexType>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
  </wsdl:message>
  <wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
  </wsdl:message>
  <wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
       </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Abstract data type definitions

Listing 1:
Example of WSDL interface definition describing a purchase order service

Data that is sent

Data that is returned

Port type with one operation

An operation with request (input) & response (output) message

cphbusiness

WSDL Implementation

- The purpose of WSDL is to specify a Web service abstractly **and then to define how the WSDL developer will reach the implementation of these services**.
- The service implementation part of WSDL contains the elements **<binding>**, **<port>**, and **<service>** and describes how a particular service interface is implemented by a given service provider.
- The service implementation describes
  - **where** the service is located, or more precisely;
  - **which network address** the message must be sent to in order to invoke the web service;
  - a **WSDL service element**.
- A service implementation document can contain references to more than one service interface document by means of <import> elements.

```
<wsdl:definitions> . …
   <import namespace="http://supply.com/PurchaseService/wsdl"
    location="http://supply.com PurchaseService/wsdl/PurchaseOrder-interface.wsdl"/>
 <!--  location of WSDL PO interface from Listing-1-->
 <!-- wsdl:binding states a serialisation protocol for this service -->
 <!--  type attribute must match name of portType element in Listing-1-->
 <wsdl:binding name="PurchaseOrderSOAPBinding"
                  type="tns:PurchaseOrderPortType">

   <!-- leverage off soapbind:binding synchronous style -->
   <soapbind:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http/"/>

   <wsdl:operation name="SendPurchase">

    <!-- again bind to SOAP -->
    <soapbind:operation
      soapAction="http://supply.com/ PurchaseService/wsdl/ SendPurchase" style="rpc"/>

    <!-- furthur specify that the messages in the wsdl:operation use SOAP -->
     <wsdl:input>
       <soapbind:body  use="literal"
        namespace="http://supply.com/PurchaseService/wsdl"/>
     </wsdl:input>
     <wsdl:output>
        <soapbind:body  use="literal"
          namespace="http://supply.com/ PurchaseService/wsdl"/>
      </wsdl:output>

  </wsdl:operation>
 </wsdl:binding>

 <wsdl:service name="PurchaseOrderService">
    <wsdl:port  name="PurchaseOrderPort" binding="tns:PurchaseOrderSOAPBinding">
       <!-- give the binding a network endpoint address or URI of service  -->
       <soapbind:address location="http://supply.com:8080/PurchaseOrderService"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Bind an abstract operation to this implementation and

map the abstract input and output messages to these concrete messages

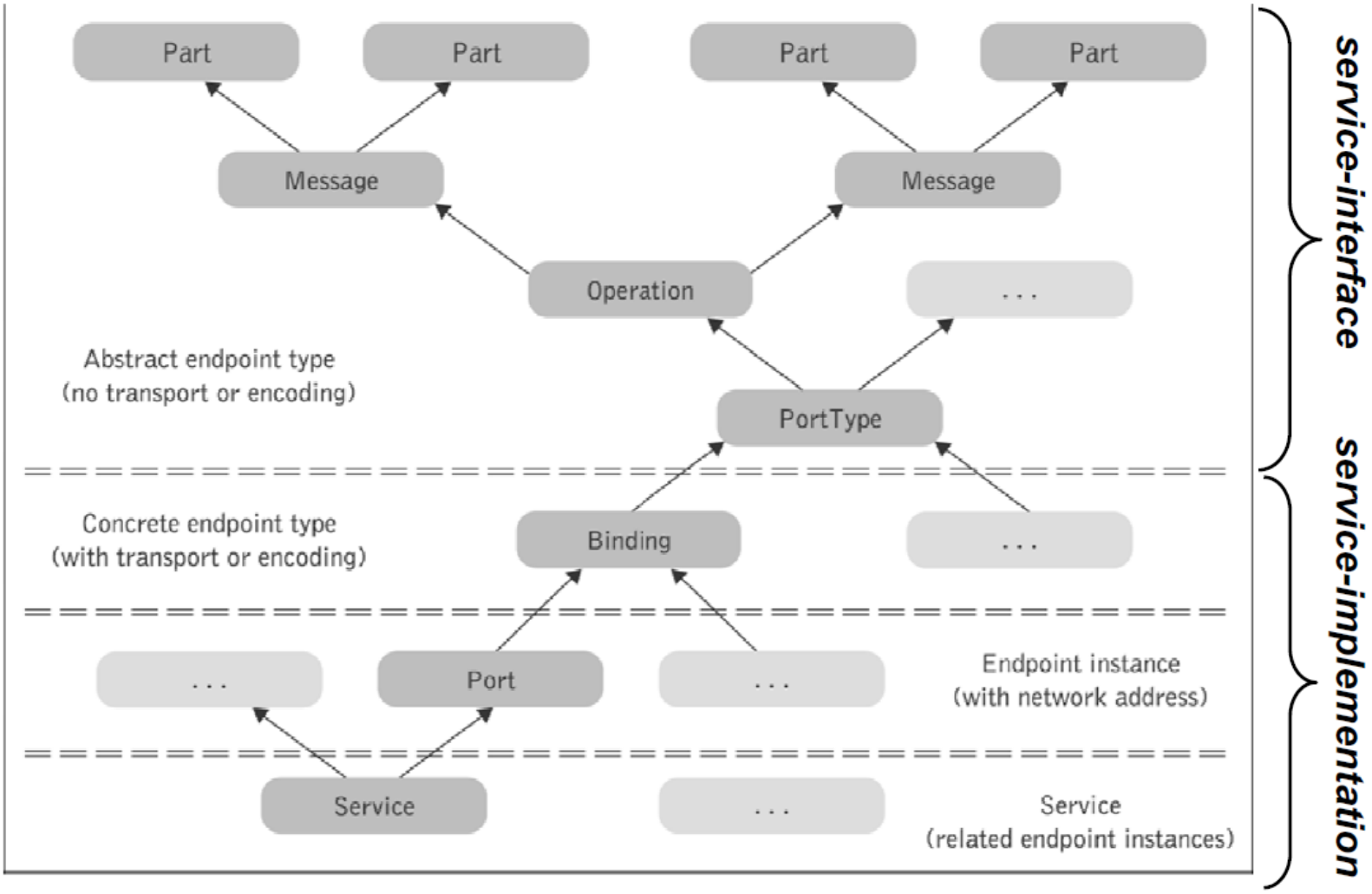
Service name

Network address of service

Listing 2: Example of WSDL implementation

cphbusiness

WSDL Elements Hierarchy



cphbusiness

<binding> Element

- … the central element of the implementation description
- … specifies how the client and web service exchange message
- The client uses this information to access the web service
- … contains information of how the elements in an abstract service interface (<portType> element) are converted into a concrete representation in a particular combination of
  - concrete protocols, e.g., SOAP or HTTP,
  - messaging styles, e.g., RPC or document style,
  - formatting (encoding) styles, e.g., literal or SOAP encoding.

<port> Element

- … defines the location of a web service
- … the URL where the service can be found
- A <port> associates an endpoint, for instance, a network address location or URL, with a specific WSDL <binding> element.
  - It is possible for two or more <port> elements to assign different URLs to the same <binding> element.
  - This might be, for instance, useful for load balancing or fail-over purposes.

```
<wsdl:definitions> . …
   <import namespace="http://supply.com/PurchaseService/wsdl"
    location="http://supply.com PurchaseService/wsdl/PurchaseOrder-interface.wsdl"/>
 <!--  location of WSDL PO interface from Listing-1-->
 <!-- wsdl:binding states a serialisation protocol for this service -->
 <!--  type attribute must match name of portType element in Listing-1-->
 <wsdl:binding name="PurchaseOrderSOAPBinding"
                  type="tns:PurchaseOrderPortType">

   <!-- leverage off soapbind:binding synchronous style -->
   <soapbind:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http/"/>

   <wsdl:operation name="SendPurchase">

    <!-- again bind to SOAP -->
    <soapbind:operation
      soapAction="http://supply.com/ PurchaseService/wsdl/ SendPurchase" style="rpc"/>

    <!-- furthur specify that the messages in the wsdl:operation use SOAP -->
     <wsdl:input>
       <soapbind:body  use="literal"
        namespace="http://supply.com/PurchaseService/wsdl"/>
     </wsdl:input>
     <wsdl:output>
        <soapbind:body  use="literal"
          namespace="http://supply.com/ PurchaseService/wsdl"/>
      </wsdl:output>

  </wsdl:operation>
 </wsdl:binding>

 <wsdl:service name="PurchaseOrderService">
    <wsdl:port  name="PurchaseOrderPort" binding="tns:PurchaseOrderSOAPBinding">
       <!-- give the binding a network endpoint address or URI of service  -->
       <soapbind:address location="http://supply.com:8080/PurchaseOrderService"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Bind an abstract operation to this implementation and

map the abstract input and output messages to these concrete messages

Service name

Network address of service

Listing 2: Example of WSDL implementation

cphbusiness

<service> Element

- … contains a collection (usually one) of WSDL <port> elements.
- … Each <service> element is named, and each name must be unique among all services in a WSDL document.

```
<wsdl:definitions> . …
   <import namespace="http://supply.com/PurchaseService/wsdl"
    location="http://supply.com PurchaseService/wsdl/PurchaseOrder-interface.wsdl"/>
 <!--  location of WSDL PO interface from Listing-1-->
 <!-- wsdl:binding states a serialisation protocol for this service -->
 <!--  type attribute must match name of portType element in Listing-1-->
 <wsdl:binding name="PurchaseOrderSOAPBinding"
                  type="tns:PurchaseOrderPortType">

   <!-- leverage off soapbind:binding synchronous style -->
   <soapbind:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http/"/>

   <wsdl:operation name="SendPurchase">

    <!-- again bind to SOAP -->
    <soapbind:operation
      soapAction="http://supply.com/ PurchaseService/wsdl/ SendPurchase" style="rpc"/>

    <!-- furthur specify that the messages in the wsdl:operation use SOAP -->
     <wsdl:input>
       <soapbind:body  use="literal"
        namespace="http://supply.com/PurchaseService/wsdl"/>
     </wsdl:input>
     <wsdl:output>
        <soapbind:body  use="literal"
          namespace="http://supply.com/ PurchaseService/wsdl"/>
      </wsdl:output>

  </wsdl:operation>
 </wsdl:binding>

 <wsdl:service name="PurchaseOrderService">
    <wsdl:port  name="PurchaseOrderPort" binding="tns:PurchaseOrderSOAPBinding">
       <!-- give the binding a network endpoint address or URI of service  -->
       <soapbind:address location="http://supply.com:8080/PurchaseOrderService"/>
    </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Bind an abstract operation to this implementation and

map the abstract input and output messages to these concrete messages

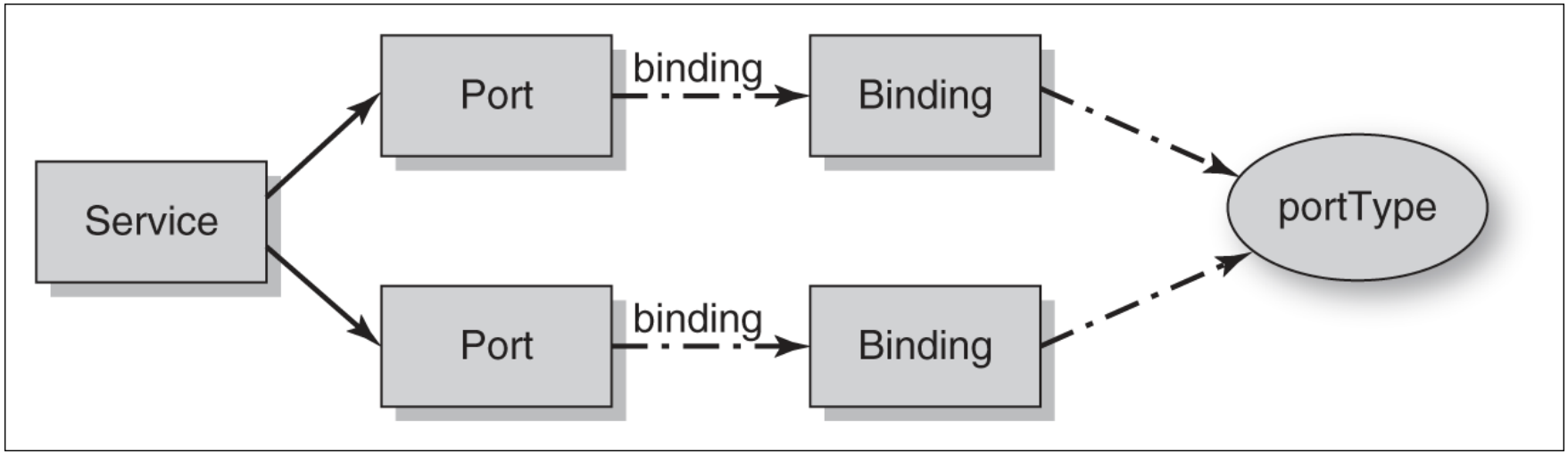
Service name

Network address of service

Listing 2: Example of WSDL implementation

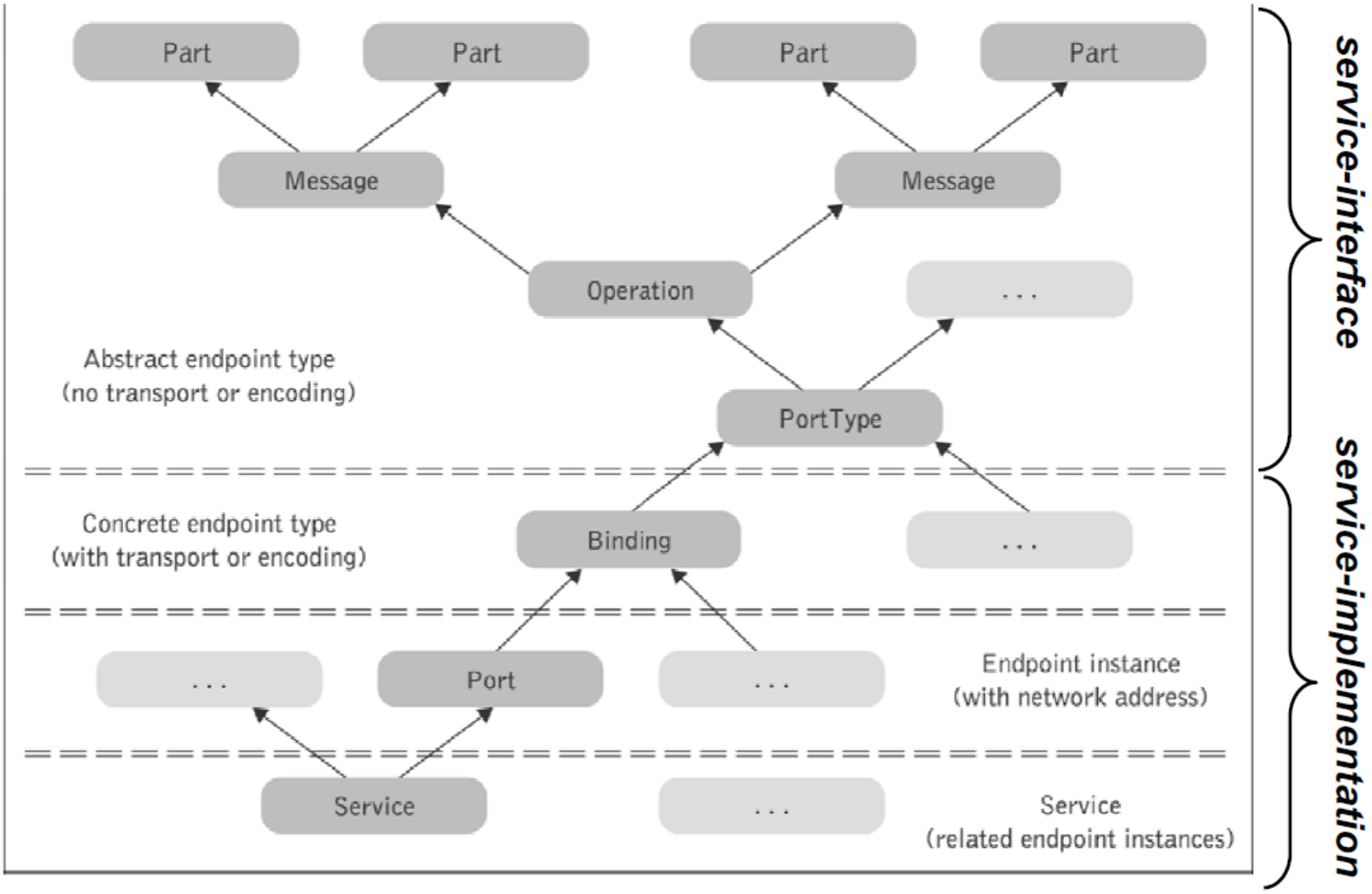
Connecting the Service Interface with the Service Implementation

- A single service may contain multiple ports that all use the same <portType>
- This implies that there could be multiple service implementations for the same service interface provided by different service providers



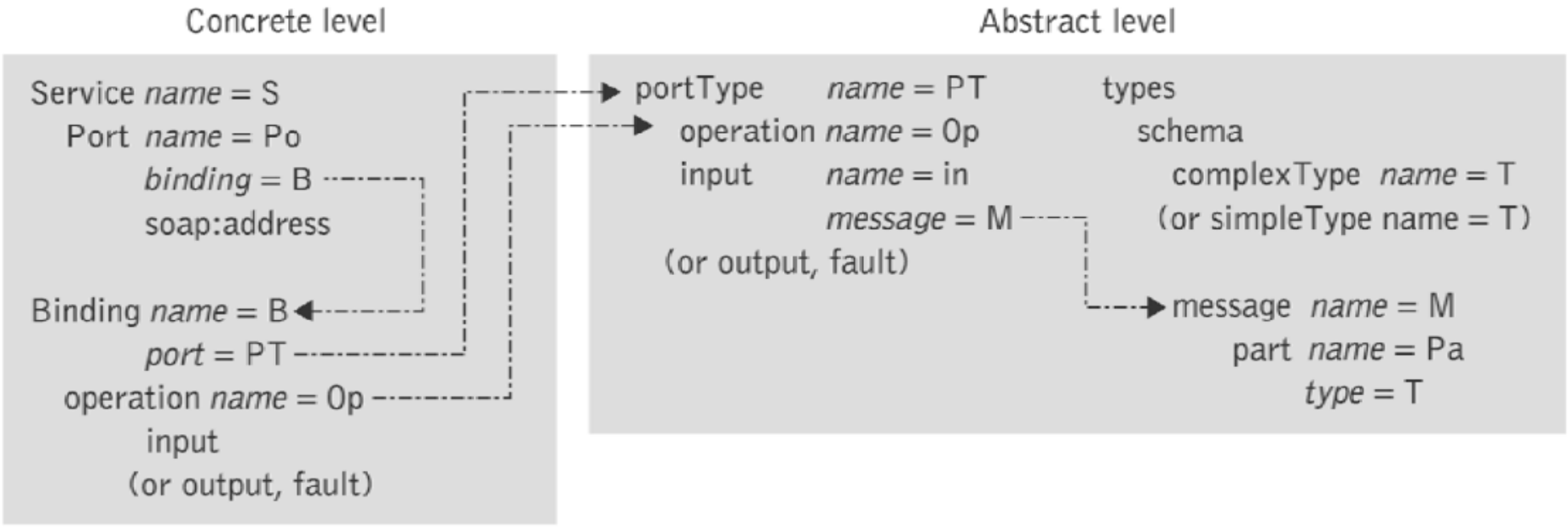
cphbusiness

WSDL Elements Hierarchy



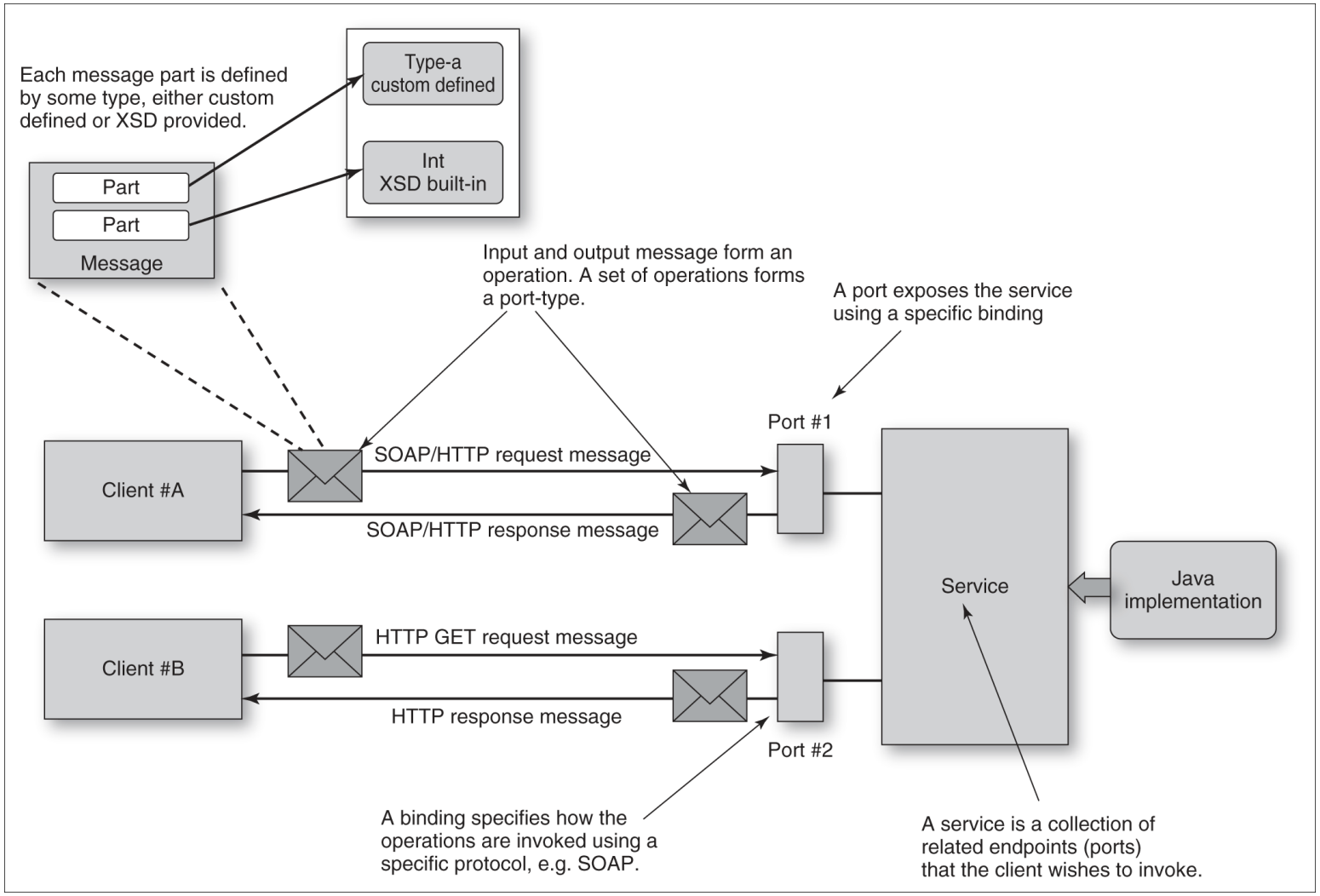
cphbusiness

Connecting the Abstract and Concrete Levels of a Web Service



cphbusiness

Elements of WSDL wrt. Requestor–Service Interaction



```
<wsdl:message name="POMessage">
      <wsdl:part name="PurchaseOrder" type="tns:POType"/>
      < wsdl:part name="CustomerInfo" type="tns:CustomerInfoType"/>
</wsdl:message>
<wsdl:message name="InvMessage">
      <wsdl:part name="Invoice" type="tns:InvoiceType"/>
</wsdl:message>

<wsdl:portType name="PurchaseOrderPortType">
      <wsdl:operation name="SendPurchase">
            <wsdl:input message="tns:POMessage"/>
            <wsdl:output message="tns:InvMessage"/>
      </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="POMessageSOAPBinding"
                type="tns:PurchaseOrderPortType">

   <soapbind:binding style="rpc"
          transport="http://schemas.xmlsoap.org/soap/http/"/>
   <wsdl:operation name="SendPurchase">

   <soapbind:operation style="rpc"
              soapAction="http://supply.com/ PurchaseService/wsdl/ SendPurchase"/>

     <wsdl:input>
        <soapbind:body use="literal"
         namespace="http://supply.com/PurchaseOrderService/wsdl"/>
     </wsdl:input>
     <wsdl:output>
        <soapbind:body use="literal"
          namespace="http://supply.com/ PurchaseOrderService/wsdl"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```

```
<?xml version= "1.0"  encoding= "UTF-8" ?>
<soap:Envelope
xmlns:soapbind="http://schemas.xmlsoap.org/soap/envelope"
  xmlns:tns="http://supply.com/ PurchaseService/wsdl ">
     <soap:Body>
        <tns:SendPurchase>
          <POtype>
              <PONumber> 223451 </PONumber>
              <PODate> 10/28/2004 </PODate>
            </POtype>
              ……
          <tns:SendPurchase>
      </soap:Body>
</soap:Envelope>
```

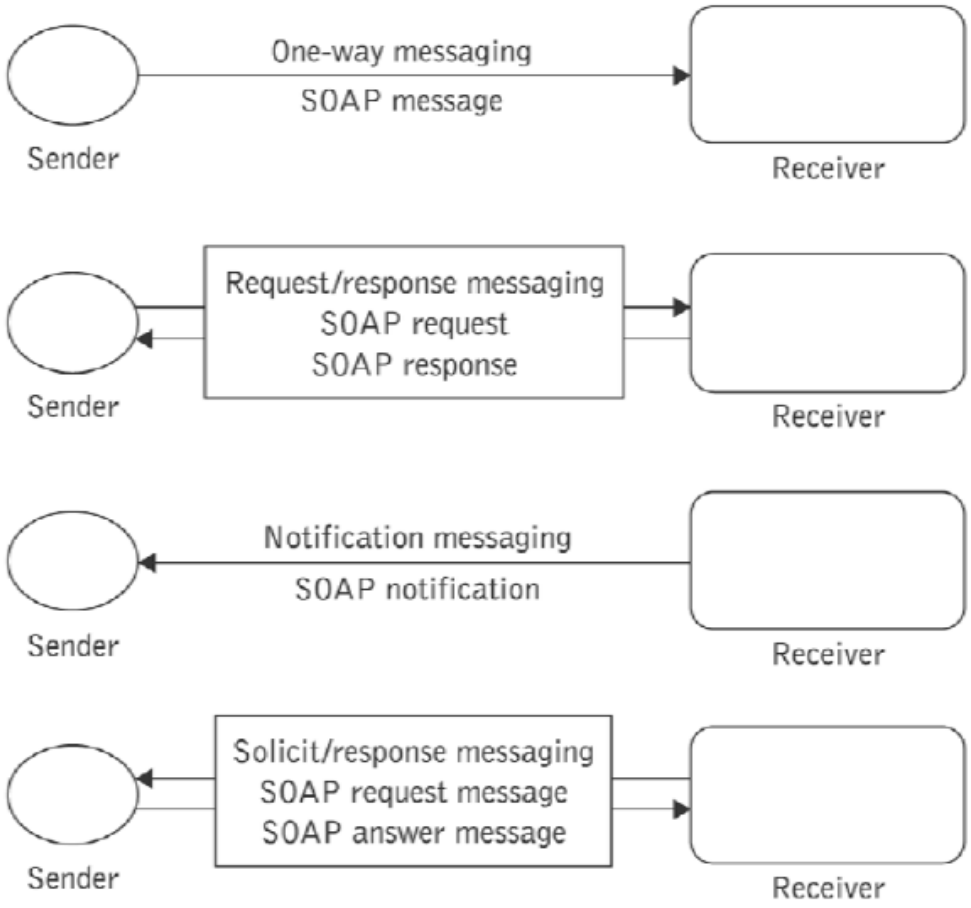
cphbusiness

WSDL Message Exchange Patterns

cphbusiness

WSDL Message Exchange Patterns

- WSDL interfaces support four common types of operations that represent possible combinations of input and output messages
- The WSDL operations correspond to the incoming and outgoing versions of two basic operation types:
  - an incoming single message passing operation and its outgoing counterpart ("one-way" and "notification" operations),
  - the incoming and outgoing versions of a synchronous two-way message exchange ("request/response" and "solicit/response").
- Any combination of incoming and outgoing operations can be included in a single WSDL interface:
  - these four types of operations provide support for both push and pull interaction models at the interface level



One-way Operation

cphbusiness

- A one-way operation is an operation in which the service endpoint receives a message, but does not send a response.
  - An example of a one-way operation might be an operation representing the submission of an order to a purchasing system. Once the order is sent, no immediate response is expected.
  - This message exchange pattern is typically thought of as asynchronous messaging. In an RPC environment, a one-way operation represents a procedure call to which no return value is assigned.
  - A one-way message defines only an input message. It requires no output message and no fault.

```
<!-- portTyepe element describes the abstract interface of a Web service -->
<wsdl:portType name="SubmitPurchaseOrder_PortType">
     <wsdl:operation name="SubmitPurchaseOrder">
          <wsdl:input name="order" message="tns:SubmitPurchaseOrder_Message"/>
     </wsdl:operation>
</wsdl:portType>
```

Request/Response Operation

cphbusiness

- A request/response operation is an operation in which the service end point receives a message and returns a message in response.
- If an <operation> element is declared with a single <input> element followed by a single <output> element, it defines a request/response operation. By listing the <input> tag first, the <operation> indicates that the Web service receives a message that is sent by the client. Listing the <output> tag second indicates that the Web service should respond to the message

```
<!-- portTyepe element describes the abstract interface of a Web service -->
<wsdl:portType name="PurchaseOrder_PortType">
     <wsdl:operation name="SendPurchase">
          <wsdl:input message="tns:POMessage"/>
          <wsdl:output message="tns:InvMessage"/>
     </wsdl:operation>
</wsdl:portType>
```

cphbusiness

Notification Operation

- A notification operation is an operation in which the service endpoint sends a message to a client, but it does not expect to receive a response.
- This type of messaging is used by services that need to notify clients of events.
  - Notification is when a <portType> element contains an <output> tag, but no <input> message definitions.
- Here the client (subscriber) has registered with the Web service to receive messages (notifications) about an event.
  - An example of this could be a service model in which events are reported to the service and where the endpoint periodically reports its status.
- No response is required in this case, as most likely the status data is assembled and logged and not acted upon immediately

cphbusiness

Solicit/Response Operation

- A solicit/response operation is an operation in which the service endpoint sends a message and expects to receive an answering message in response.
- This is the opposite of the request/response operation since the service endpoint is initiating the operation (soliciting the client), rather than responding to a request.
- Solicit/response is similar to notification messaging, except that the client is expected to respond to the Web service.
- With this type of messaging the <portType> element first declares an <output> tag and then a <input> message definition – exactly the reverse of a request/response operation.
  - An example of this operation might be a service that sends out order status to a client and receives back a receipt.

cphbusiness

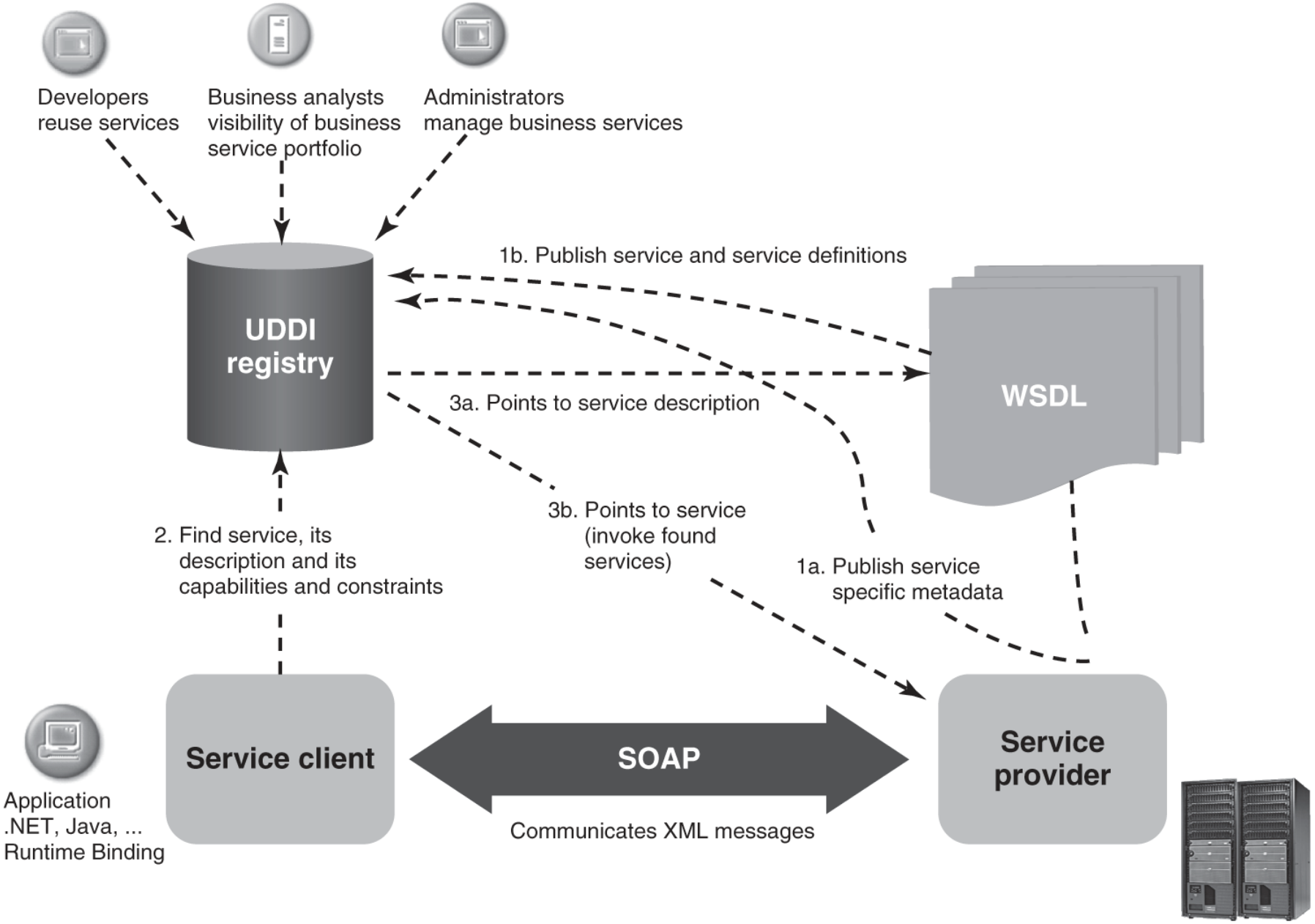
Service Registries and Discovery

RIP

http://uddi.xml.org/public-uddi-registry

SOA Interactions between Actors

- Problem to solve:
  - How to find the service a client wants among a large set of services and providers.
  - The client does not need to know which provider provides the service.


Developers
reuse services
Business analysts
visibility of business
service portfolio
Administrators
manage business services
UDDI
registry
1b. Publish service and service definitions
WSDL
3a. Points to service description
3b. Points to service
(invoke found
services)
2. Find service, its
description and its
capabilities and constraints
1a. Publish service
specific metadata
Application
.NET, Java, ...
Runtime Binding
Service client
SOAP
Communicates XML messages
Service
provider

cphbusiness

Service Registries

- To discover Web services, a service registry is needed. This requires describing and registering the Web service.
- Publication of a service requires proper description of a Web service in terms of business, service, and technical information.
- Registration deals with persistently storing the Web service descriptions in the Web services registry.
- Two types of registries can be used:
  - The document-based registry: enables its clients to publish information, by storing XML-based service documents such as business profiles or technical specifications (including WSDL descriptions of the service).
  - The meta-data-based service registry: captures the essence of the submitted document.

Service Discovery

- Service discovery is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.
- Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.
  - A query consists of search criteria such as:
    - the type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
  - Discovering Web services is a process that is also dependent on the architecture of the service registry.
- After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI) its capabilities, and how to interface with it.

cphbusiness

Types of Service Discovery

Static

- The service implementation details (network location and protocol) are bound at design time and a service retrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by a human designer and the service description returned by the retrieval operation is incorporated into the application logic.

Dynamic

- The service implementation details are left unbound at design time so that they can be determined at run-time.
- The Web service requestor has to specify preferences to enable the application to **infer/reason** which Web service(s) the requester is most likely to want to invoke.
- Based on application logic quality of service considerations such as best price, performance, security certificates, and so on, the application chooses the most appropriate service, binds to it, and invokes it.

cphbusiness

Universal Description, Discovery and Integration (UDDI)

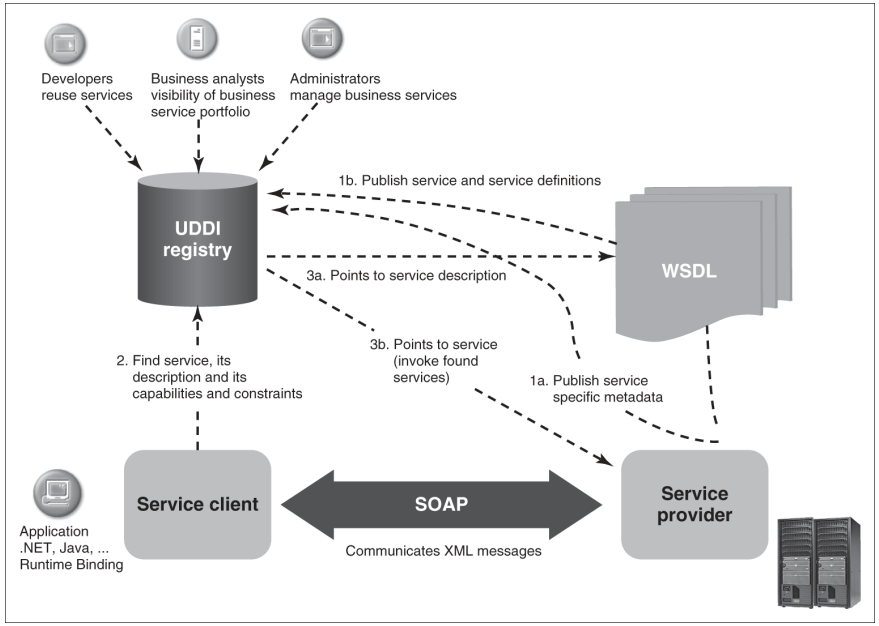
cphbusiness

What is UDDI?

- … a registry standard for Web service description and discovery facility that supports WS publishing and discovery processes
- UDDI enables a business to:
  - **describe** its business and its services;
  - **discover** other businesses that offer desired services;
  - **integrate (interoperate)** with these other businesses.
- Conceptually, a UDDI business registration consists of three inter-related components:
  - "white pages" (address, contact, and other key points of contact);
  - "yellow pages" classification info. based on standard industry taxonomies; and
  - "green pages", the technical capabilities and information about services.

The UDDI Usage Model

- An enterprise may set up multiple *private* UDDI registries in-house to support intranet and e-Business operations.
- Public UDDI registries can be set up by customers and business partners of an enterprise.
  - Services must be published in a public UDDI registry so that potential clients and service developers can discover them.



UDDI – Main Characteristics

- UDDI provides a mechanism to categorize businesses and services using **taxonomies**.
  - Service providers can use a taxonomy to indicate that a service implements a specific domain standard, or that it provides services to a specific geographic area
  - UDDI uses standard taxonomies so that information can be discovered on the basis of categorization.
- **UDDI business registration: an XML document** used to describe a business entity and its Web services.
- UDDI is not bound to any technology. In other words,
  - An entry in the UDDI registry can contain any type of resource, independently of whether the resource is XML based or not, e.g., the UDDI registry could contain information about an enterprise's electronic document interchange (**EDI**) system or even a service that, uses the **fax machine** as its primary communication channel.
  - While UDDI itself uses XML to represent the data it stores, it allows for other kinds of technology to be registered.