# Service-Oriented Architecture

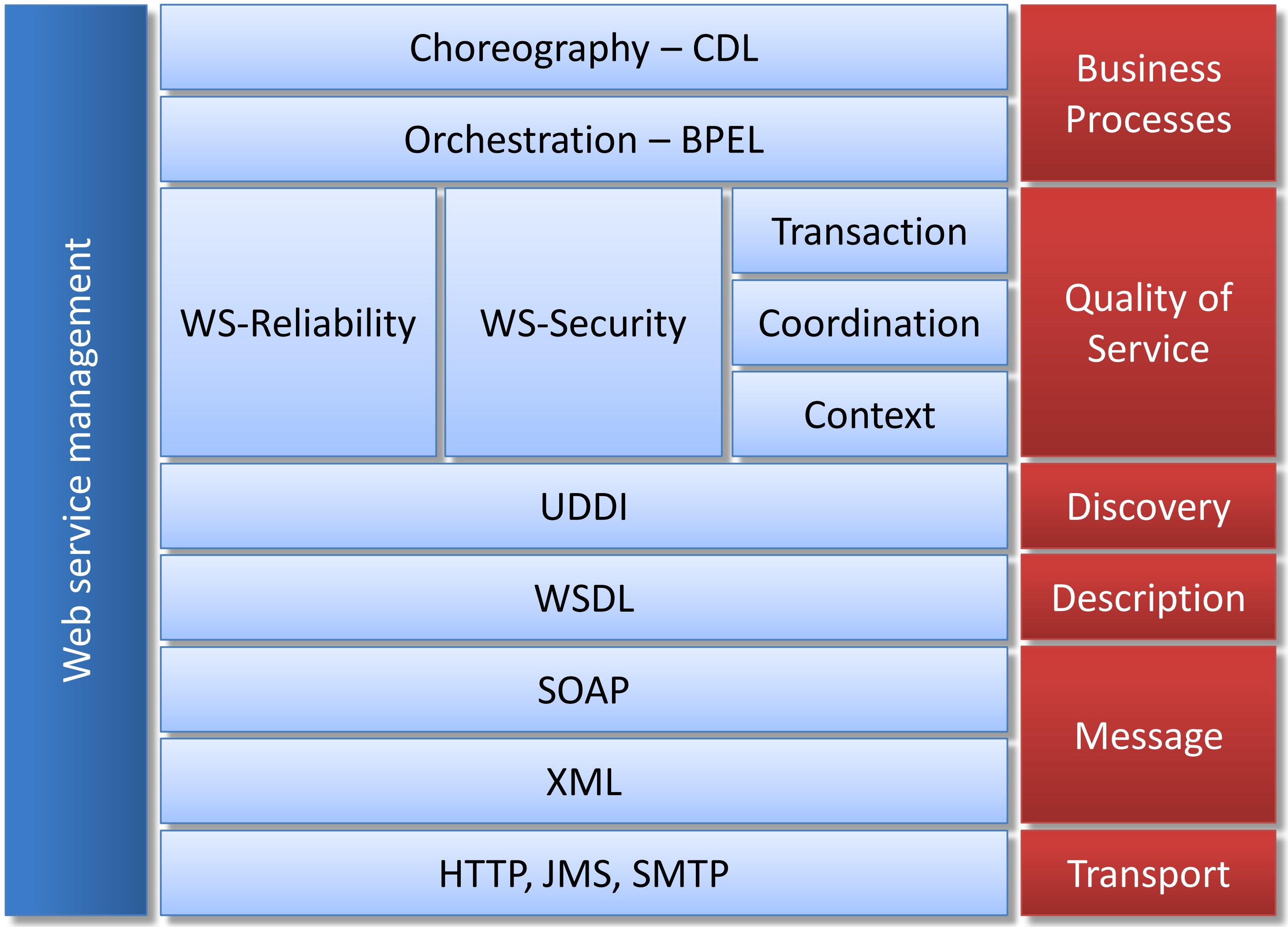
## Simple Object Access Protocol (SOAP)

Lecture 03

Henry Novianus Palit

hnpalit@petra.ac.id

# Web Service Technology Stack



## SOAP in a Nutshell *(1)*

- SOAP is an XML-based communication protocol for <u>exchanging messages between computers regardless of their OS, programming environment, or object model framework</u>
- It provides the definition of XML-based information which can be used for <u>exchanging structured & typed information between peers</u> in a decentralized, distributed environment
- Its primary application is <u>inter-application communication</u>, e.g., e-Business integration
- A *SOAP method* is simply an HTTP request or response that complies with the SOAP encoding rules
- A *SOAP endpoint* is simply an HTTP-based URL that identifies a target for method invocation
- It possesses <u>two fundamental properties</u>:
  - Send and receive HTTP (or, other transport protocol) packets
  - Process XML messages

# SOAP in a Nutshell *(2)*

- Wire (messaging) vs. transport protocol
  - Wire protocol specifies the form or shape of the data to be exchanged
  - Transport protocol specifies the method by which data is transferred
- A wire protocol has to meet specific design criteria
  - *Compactness* $\rightarrow$ refer to how concise a network package becomes while conveying the same information; small degree of compactness is usually best
  - *Protocol efficiency* $\rightarrow$ examine the overhead required to send the payload; the less overhead required, the more efficient the protocol is
  - *Coupling* $\rightarrow$ rate flexibility and adaptability of the protocol to changes; loosely coupled protocols are better
  - *Scalability* $\rightarrow$ address the ability of a protocol to work with a large number of potential recipients
  - *Interoperability* $\rightarrow$ refer to the ability of the protocol to work with a variety of computing platforms

# SOAP in a Nutshell *(3)*

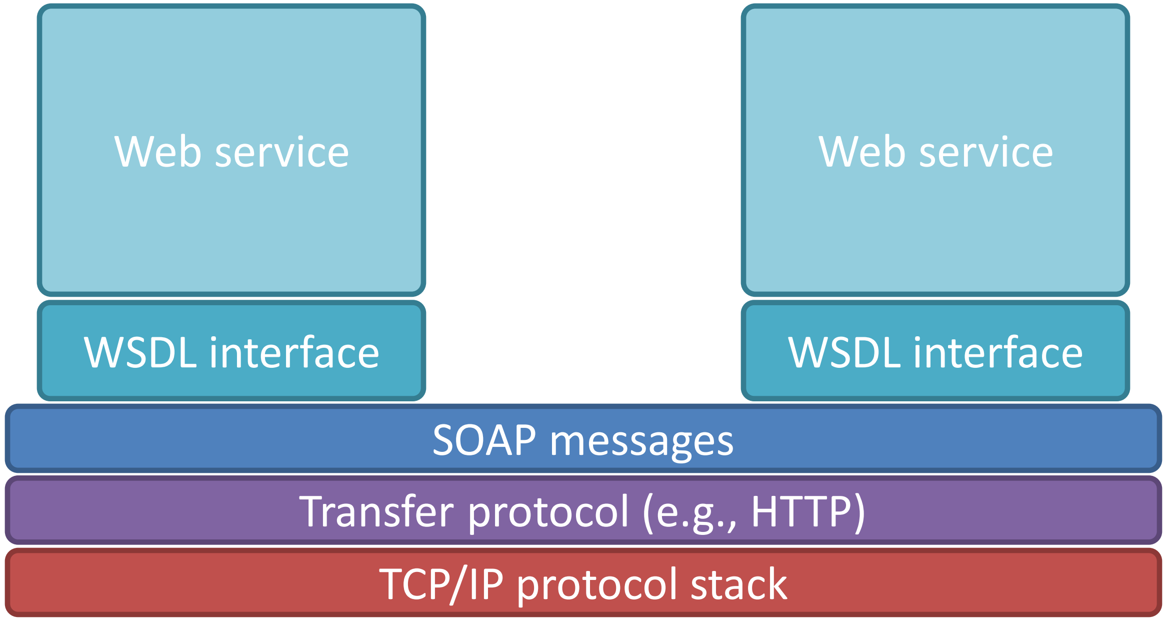
- No single protocol can achieve all the above criteria
  - XML and SOAP are both loosely coupled and interoperable
  - Those two criteria adversely affect compactness and efficiency of both protocols
  - SOAP uses HTTP to achieve scalability
- As a wire protocol, SOAP commonly <u>uses HTTP to transport XML-encoded serialized method argument data</u> from system to system
  - The serialized argument data is used on the remote end to execute a client's method call on that remote system, rather than on a local system

# SOAP – Messaging Protocol *(1)*

- Goal of SOAP is to <u>diffuse the barriers of heterogeneity</u> that separate distributed computing platforms
- SOAP follows the same recipe as other successful Web protocol: *simplicity*, *flexibility*, *firewall friendliness*, *platform neutrality*, and *XML messaging (text) based*
- Instead of being a new technological advancement, SOAP <u>codifies the usage of existing Internet technologies to standardize distributed communications over the Web</u>
- SOAP has a clear purpose – exchanging data over networks
  - Encapsulate & encode XML data
  - Define the rules for transmitting & receiving that data
  - ☞ SOAP describes how a message is formatted, but it does not specify how it is delivered; that is the task of a transport protocol

# SOAP – Messaging Protocol *(2)*

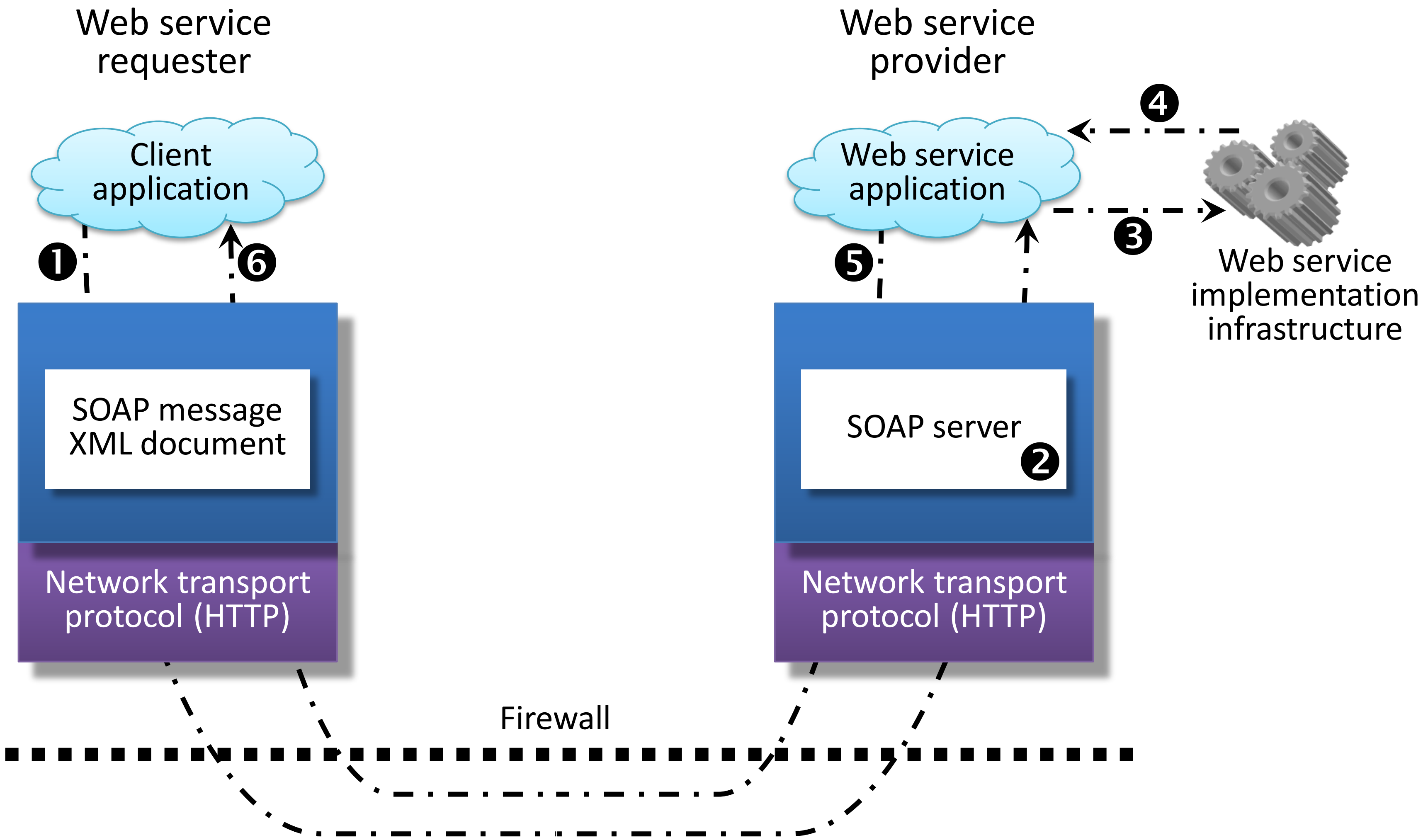
- SOAP may use different protocols to transport messages, locate the remote system, and initiate communications: HTTP (the natural choice), FTP, SMTP, or RMI
- SOAP is a network application protocol used to transfer messages between service instances described by WSDL



# SOAP – Messaging Protocol *(3)*

- The SOAP message will be the body of an HTTP message; the HTTP message becomes data in a TCP stream sent over a connection to the other end
- At the other end, an HTTP listener passes the HTTP body on to a SOAP processor that understands the syntax and is able to process the message
- SOAP is a *stateless, one-way message exchange paradigm*
  - Applications can create more complex interaction patterns (e.g., request/response, request/multi-responses, etc.) by combining such one-way exchanges with features provided by an underlying protocol and/or application specific information
- SOAP allows a flexible binding;  e.g., a Web service might provide two bindings and the client may submit a SOAP request using either HTTP or SMTP (as an email)

# SOAP – Messaging Protocol *(4)*


Web service requester
Web service provider
Client application
Web service application
❶
❷
❸
❹
❺
❻
Web service implementation infrastructure
SOAP message XML document
SOAP server
Network transport protocol (HTTP)
Network transport protocol (HTTP)
Firewall

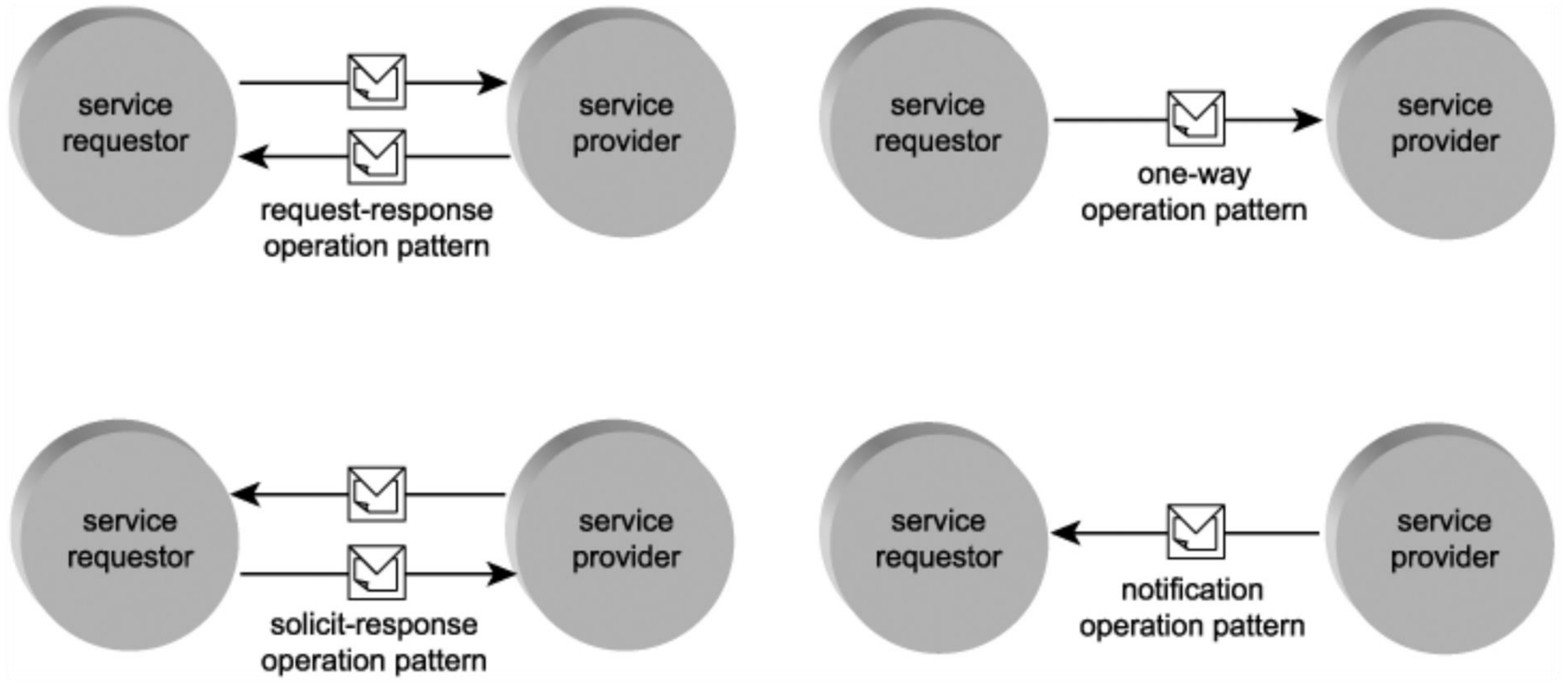
## SOAP – Messaging Protocol *(5)*

- Basic steps of distributed application processing with SOAP:
  1. A service client <u>creates a SOAP message to invoke a desired Web service operation</u> hosted by a remote service provider. The method request & its arguments are XML-encoded and placed in the body of the SOAP request. The service requester <u>forwards the SOAP message together with the provider's URI to the network infrastructure</u>.
  2. The network infrastructure <u>delivers the message to the service provider's SOAP run time system (e.g., a SOAP server)</u>. The SOAP server is simply special code that listens for SOAP messages and acts as a distributor & interpreter of SOAP documents.
  3. The SOAP server <u>routes the request message to the service provider's WS implementation code</u>. The SOAP server <u>ensures that documents received over an HTTP/SOAP connection are converted from XML to programming language specific objects</u> required by the WS app. The conversion is governed by the encoding scheme found within the SOAP message envelope. The SOAP server also <u>ensures that the parameters included in the SOAP document are passed to the appropriate methods</u> in the WS implementation infrastructure.

# SOAP – Messaging Protocol *(6)*

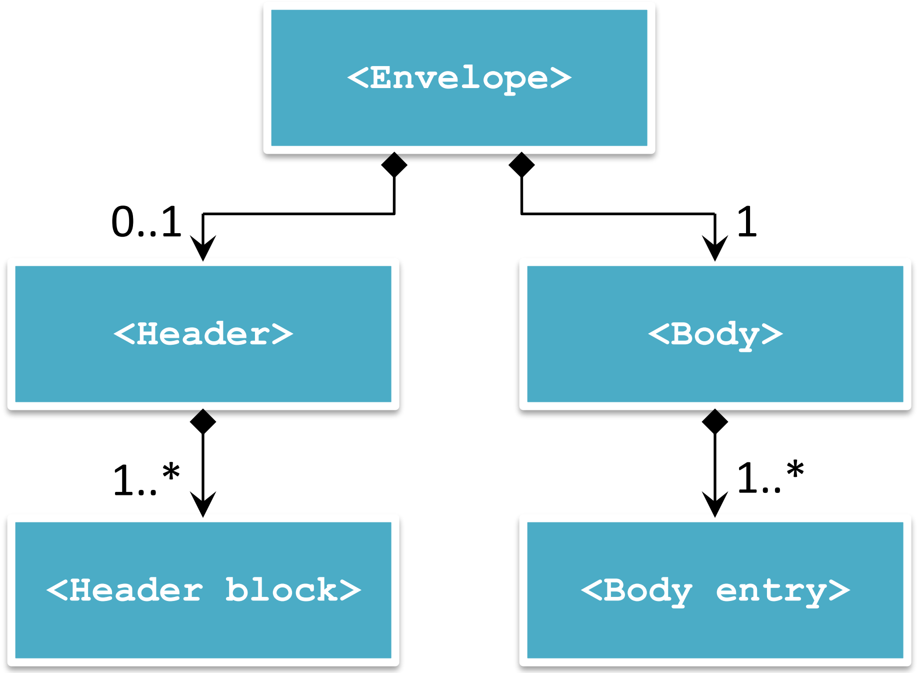
- Basic steps … : (cont'd)
  4. The Web service is responsible for <u>processing the request and formulating a response</u> as a SOAP message. <u>The response SOAP message is presented to the SOAP run time system</u> at the provider's site <u>with the service requester's URI</u> as its destination.
  5. The SOAP server <u>forwards the response SOAP message to the service requester</u> over the network.
  6. The response message is received by the network infrastructure on the service requester's node. <u>The message is routed through the SOAP infrastructure</u>, potentially <u>converting the XML response into objects understood by the source (service requester's) application</u>.
- Web services can use different message exchange patterns:
  - One-way → the service expects a single message
  - Request/response → upon receiving a message, the service responds
  - Solicit/response → upon submitting a message, the service expects a response
  - Notification → the service sends a message

# SOAP – Messaging Protocol *(7)*



# Structure of SOAP Message

- SOAP specification v1.2 describes how the data types defined in associated XML schemas are serialized over HTTP or other transport protocols
  - Both the provider and requester of SOAP messages must have access to the same XML schemas to exchange information correctly
  - The schemas are normally posted on the Internet, and may be downloaded by any party in an exchange of messages
- A SOAP message consists of an **`<Envelope>`** element containing an optional **`<Header>`** and a mandatory **`<Body>`** element



# SOAP Envelope *(1)*

- The SOAP envelope wraps any XML document interchange and provides a mechanism to augment the payload with additional information required to route it to its destination
- It is the single root of every SOAP message and must be present for the message to be SOAP compliant
- If a **`<Header>`** element is used, it must be the immediate child of the **`<Envelope>`** element and precede the **`<Body>`** element

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <env:Header> <!-- optional -->
        <!-- header blocks go here ... -->
    </env:Header>
    <env:Body>
        <!-- payload or Fault element goes here ... -->
    </env:Body>
</env:Envelope>
```

## SOAP Envelope *(2)*

- All elements of the SOAP envelope are defined using W3C XML Schema (i.e., **`http://www.w3.org/2003/05/soap-envelope`**)
- A SOAP envelope can specify a set of encoding rules (i.e., defined in an XML schema) serializing the application-defined XML data over the network
- Two or more communicating parties desiring to express their agreement on a specific encoding style in the SOAP messages can use the global **`encodingStyle`** attribute
- To give more flexibility, SOAP allows applications to define their own encoding style, e.g., if two apps use an array as argument in a SOAP message, one app may serialize the array as a sequence of rows using one set of tags, while another may serialize it as a sequence of columns using a different set of tags

## SOAP Header *(1)*

- A SOAP `<Header>` element contains blocks of information relevant to how the message is to be processed (e.g., where the document shall be sent, where it originated, who sent it)
- The header's purpose is to encapsulate extensions to the message format without having to couple them to the payload or to modify the fundamental structure of SOAP; hence, it facilitates additional features like security, transactions, object references, billing, QoS attributes, etc.
- The schema for the optional SOAP header element allows for an unlimited number of child elements in the header
  - The immediate child elements are called *header blocks*, representing a logical grouping of data that can individually be targeted at SOAP nodes that might be encountered in the path of a message from a sender to an ultimate receiver
  - Each header block should have its own namespace for easy identification and processing

# SOAP Header *(2)*

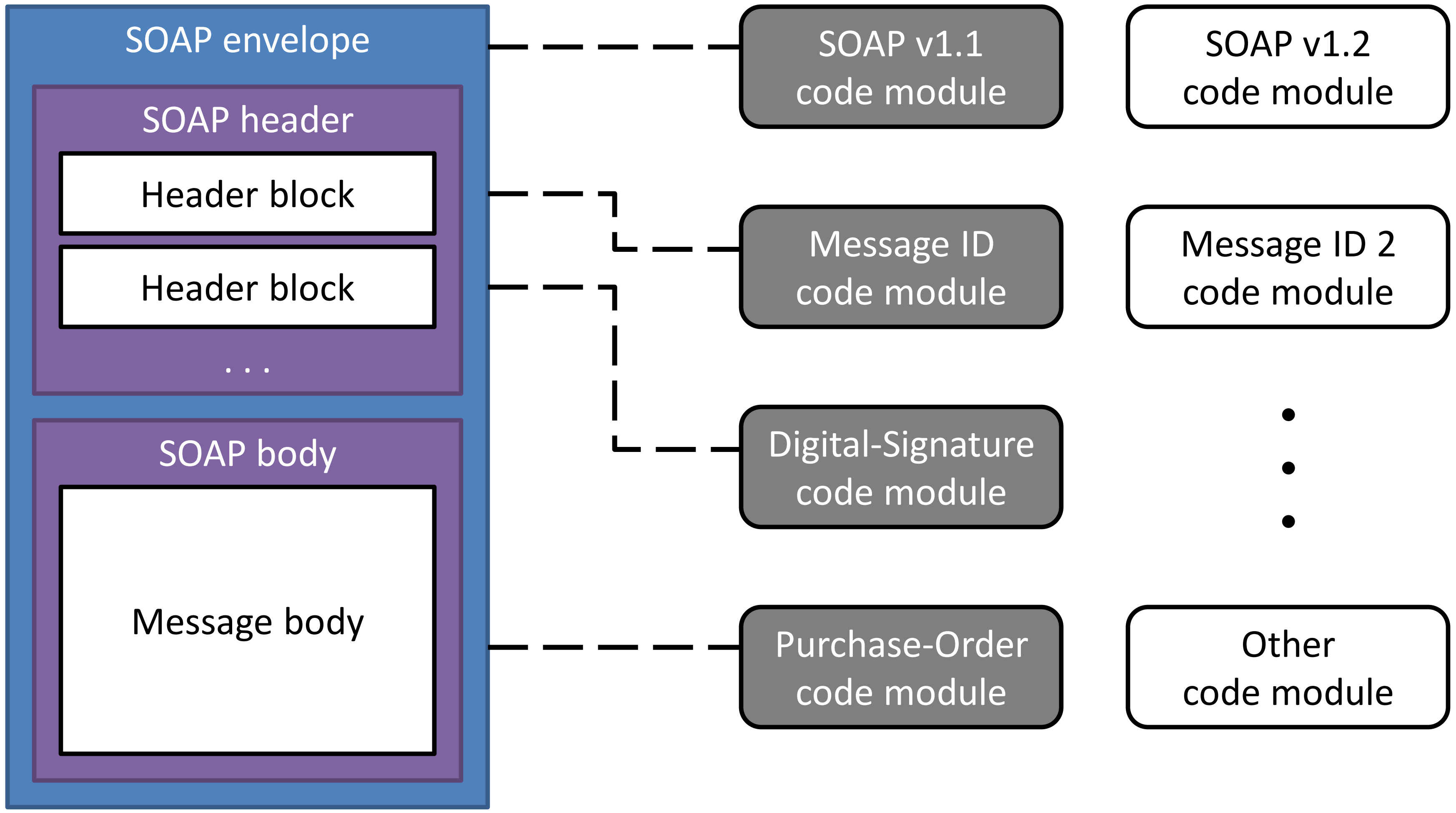
- Example of a SOAP header

```
<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>
```

## SOAP Header *(3)*

- Identifying the header block version by its namespace enables a SOAP receiver to switch processing models, or to reject messages if it does not support the specified version
  - This *modularity* enables different parts of a SOAP message to be processed independently of other parts and to evolve separately, e.g., the version of SOAP **`<Envelope>`** or **`<Header>`** block may change over time while the structure of the application specific contents in the **`<Body>`** element remains the same
  - This *modularity* enables developers to use different code libraries to process different parts of a SOAP message
- The SOAP **`<Header>`** element also provides for *extensibility*, in which additional information required for a particular service (e.g., authenticating the requester before a method is invoked) can be added to SOAP; this can be accomplished without changing the message body

# Processing Parts of a SOAP Message

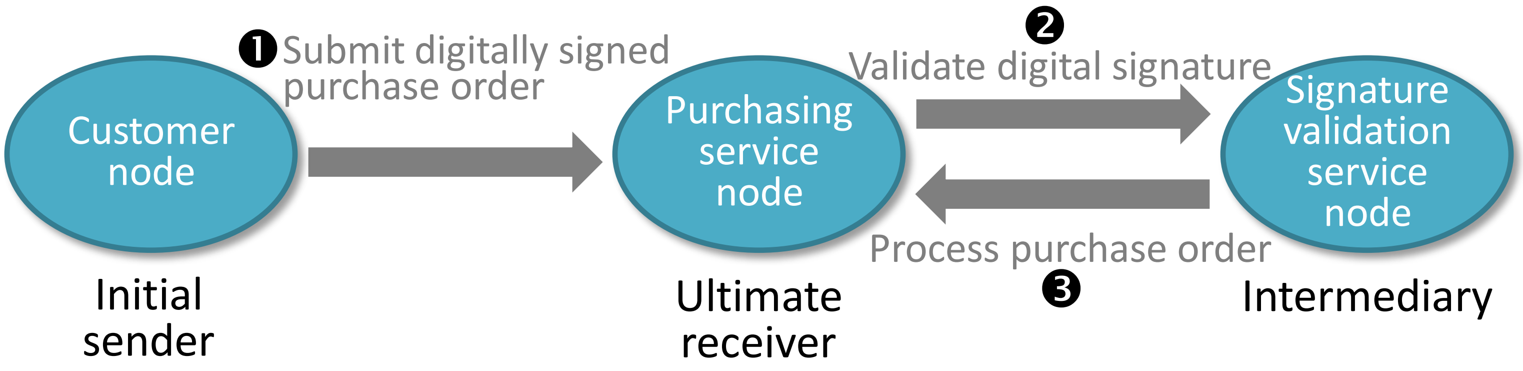


# SOAP Intermediaries *(1)*

- SOAP headers have been designed to anticipate the participation of other SOAP processing nodes – called *SOAP intermediaries* – along a message's path from an initial SOAP sender (point of origin) to an ultimate SOAP receiver (final destination)
- The route taken by a SOAP message, passing through all intermediaries, is called the *SOAP message path*
- Along the message path, <u>the header blocks of a SOAP message may be intercepted and processed by a number of SOAP intermediaries</u>
- The applications along the message path are commonly referred to as *SOAP nodes*
- <u>Headers may be inspected, inserted, deleted, or forwarded by SOAP nodes</u> encountered along a SOAP message path

# SOAP Intermediaries *(2)*

- Three key use cases define the need for SOAP intermediaries
  - Crossing trust domains
  - Ensuring scalability
  - Providing value-added services along the message path



- Example: validating a purchase order SOAP message
  - The intermediary service verifies that the customer's digital signature header block embedded in the SOAP message is actually valid
  - The SOAP message is automatically routed to the intermediary node, which extracts the digital signature, validates it, and add a new header block telling the purchasing service whether the signature is valid

# SOAP Body

- The SOAP **`<Body>`** element is the mandatory element within the SOAP **`<Envelope>`** and implies that this is where the application-specific XML data (payload) is placed
- It may contain an arbitrary number of child elements, called *body entries*, but it may also be empty
- All body entries that are immediate children of the **`<Body>`** element must be namespace qualified
- The **`<Body>`** element contains either the application-specific data (i.e., can be arbitrary XML data or parameters to a method call) or a fault message, but not both
- The **`<Body>`** element has one distinguished root element, which is either the request or the response object

## SOAP Communication Model

- SOAP communication model is defined by its encoding and its communication style

- The SOAP encoding style conveys information about how the contents of a particular element in the header blocks, or the `<Body>` element of a SOAP message, are encoded

- SOAP supports two possible communication styles:
  - RPC
  - Document (or message)

## RPC Style Web Services *(1)*

- An RPC style Web service appears as a remote object to a client application
- The interaction between a client and an RPC style Web service centers around a service-specific interface
- Clients express their request as a method call with a set of arguments, and the expected response contains a return value; these are represented as sets of XML elements embedded within a SOAP message
- RPC style supports automatic serialization/deserialization of messages
- RPC/Literal messaging is used to expose traditional components – e.g., a servlet, a stateless session bean, a Java RMI object, a CORBA object, or a DCOM component – as Web services

## RPC Style Web Services *(2)*

- The rules for packaging an RPC/Literal request in a SOAP envelope:
  - A URI identifying the transport address for the call is required
  - An RPC request message contains the method name and the input parameters of the call
  - The names and the physical order of the parameters must correspond to those in the method being invoked
  - An RPC response message contains the return value and any output parameters (or a fault)
- Once a SOAP message containing a call body (a method element and arguments in the `<Body>` element) has been sent, it is reasonable to expect that a response message will ensue
- A useful feature of the HTTP binding when RPC style messages are transmitted is that it provides a way automatically to associate the request with the corresponding response

# RPC Style Web Services *(3)*

- Example of an RPC-style SOAP request

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.auto-parts.com/product-prices">
    <env:Header>
        <tx:Transaction-id
          xmlns:tx="http://www.transaction.com/transactions"
          env:mustUnderstand="true">
            512
        </tx:Transaction-id>
    </env:Header>
    <env:Body>
        <m:GetProductPrice>
          <product-id>450R60P</product-id>
        </m:GetProductPrice>
    </env:Body>
</env:Envelope>
```

## RPC Style Web Services *(4)*

- Example of an RPC-style SOAP response

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

## Document Style Web Services *(1)*

- SOAP provides no means for encoding source and destination information into the envelope; it is up to the client of SOAP to decide where and how the information is to be transmitted
- Sending non-coded XML content in the body is known as *document style* SOAP, as it focuses on the message as an XML document rather than an abstract data model encoded in XML
- Document style Web services are message-driven and asynchronous (i.e., client can continue with its computation without waiting for a response)
  - A client typically sends an entire document, such as a purchase order, rather than a discrete set of parameters
  - The Web service, which processes the document, may or may not return a response message

## Document Style Web Services *(2)*

- The document style does not support automatic serialization/ deserialization of messages; it assumes that the contents of the SOAP message are well formed XML documents
- In the Document/Literal mode of messaging, a SOAP **`<Body>`** element contains an XML document fragment – a well formed XML element containing arbitrary application data that belongs to an XML schema & namespace, separate from that of the SOAP message
- The SOAP run time environment accepts the SOAP **`<Body>`** element as it stands and hands it over to the destined application unchanged

# Document Style Web Services *(3)*

- Example of a document style SOAP message

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
        <tx:Transaction-id
          xmlns:tx="http://www.transaction.com/transactions"
          env:mustUnderstand="true">
            512
        </tx:Transaction-id>
    </env:Header>
    <env:Body>
        <po:PurchaseOrder orderDate="2009-12-20"
          xmlns:po="http://www.auto-parts.com/PurchaseOrder">
            <po:from>
                <po:accountName> Nifty Cars Inc. </po:accountName>
                <po:accountNumber> PSC-0343-02 </po:accountNumber>
            </po:from>
            <po:to>
                <po:supplierName>
                    Auto Parts & Car Accessories
                </po:supplierName>
```

(to be continued)

## Document Style Web Services *(4)*

- Example of a document style SOAP message (cont'd)

```
                <po:supplierAddress>
                    Yara Valley Melbourne
                </po:supplierAddress>
            </po:to>
            <po:product>
                <po:product-name>
                    Catalytic Converter
                </po:product-name>
                <po:product-model>
                    GTTX-100
                </po:product-model>
                <po:quantity>
                    200
                </po:quantity>
            </po:product>
        </po:PurchaseOrder>
    </env:Body>
</env:Envelope>
```

# SOAP's Advantages

- *Simplicity* → as it bases on XML (i.e., highly structured and easy to parse)
- *Portability* → no dependency on the underlying platform, like byte-ordering issues or machine-word widths;  XML parsers exists for virtually any platform
- *Firewall friendly* → as it relies on HTTP that is widely available and able to get past firewalls
- *Use of open standards* → it uses the open standard of XML, so it is easily extendable and well supported
- *Interoperability* → it is built on open technologies and facilitates true distributed interoperability & loosely coupled applications
- *Acceptance* → it is widely accepted in the message communication domain
- *Resilience to changes* → changes to the SOAP infrastructure will likely not affect applications using the protocol

## SOAP's Disadvantages

- SOAP was <u>initially tied to HTTP</u>, and this mandated a request/ response architecture that may not be appropriate for some situations. In addition, HTTP is a relatively slow protocol, so the performance of SOAP may suffer. The latest version of the SOAP specification loosens this requirement.
- SOAP is <u>stateless</u>, so the requesting application must reintroduce itself when more connections are required. Maintaining the state of a connection is desirable when multiple Web services interact in the context of business processes and transactions.
- SOAP <u>serializes by value</u> and does not support serialization by reference. Consequently, an object's copy will, over time, contain state information that is not synchronized with other dislocated copies of the same object. Currently it is not possible for SOAP to refer to an external data source (in the form of an object reference).

# Service-Oriented Architecture

## Web Services Description Language (WSDL)

Lecture 03

Henry Novianus Palit

hnpalit@petra.ac.id

# Why is a Service Description Needed?

- Service definition is <u>an integral part of designing and developing SOA based applications</u>
- To support service assemblies, <u>Web services need to be described in a consistent manner</u>; thus, they can be published by service providers, discovered by service clients and developers, and assembled in a manageable hierarchy of composite services that are orchestrated to deliver value-added service solutions and composite application assemblies
- <u>Consumers must determine the precise XML interface of a Web service along with other miscellaneous message details</u> a priori
  - XML Schema can partially fill this need as it allows developers to describe the structure of XML messages understood by Web services
  - XML Schema alone cannot describe important additional details involved in communicating with a Web service, such as service functional and non-functional characteristics or service policies

# WSDL in a Nutshell *(1)*

- Web Services Description Language (WSDL) is an XML based specification language that has become *de facto* language for describing the public interface of a Web service (or, the contract that a service exposes to its clients)
- WSDL recognizes the need for rich type systems for describing message formats and supports the XML Schema specification (XSD) as its canonical type system
- WSDL supports extensibility to specify some technology specific binding; this allows enhancements in the area of network and message protocols without having to revise the base WSDL specification
- WSDL v1.1 is broadly implemented and supported by industry; WSDL v2.0 was recently (June 2007) defined as a recommendation of W3C

# WSDL in a Nutshell *(2)*

- WSDL document uses the following elements in the definition of services:
  - Types → a container for data type definitions using some type system (such as XSD)
  - Message → an abstract, typed definition of the data being communicated
  - Operation → an abstract description of an action supported by the service
  - Port type → an abstract set of operations supported by one or more endpoints
  - Binding → a concrete protocol and data format specification for a particular port type
  - Port → a single endpoint defined as a combination of a binding and a network address
  - Service → a collection of related endpoints

# Service Interface & Implementation *(1)*

- Essentially, WSDL is used to describe precisely three things:
  - What a service does → the operations the service provides
  - Where it resides → the details of the protocol specific address (e.g., a URL)
  - How to invoke it → the details of the data formats and protocols necessary to access the service’s operations
- WSDL provides a mechanism by which service providers can describe the basic format of Web requests over different protocols (e.g., SOAP) or encoding (e.g., MIME / Multipurpose Internet Messaging Extensions)
- Most of the WSDL elements are allowed to be extended with elements from other namespaces
- WSDL represents a service contract between the service requester and the service provider, in much the same way as an interface in an object oriented programming language

## Service Interface & Implementation *(2)*

- The partitioning into a distinct service interface and service implementation sections enables each part to be defined separately and independently, and be reused by other matching parts
  - *Service interface definition* (abstract service description) describes the general Web service interface structure in a language and platform independent manner; it contains all the operations supported by the service, the operation parameters, and abstract data types
  - *Service implementation* (concrete endpoint) binds the abstract interface to a concrete network address, to a specific protocol, and to concrete data structures; it describes the concrete details of how the abstract interface maps to messages on the wire, i.e., defines site specific matters such as serialization
- The combination of these two parts contains sufficient information to describe to the service requester how to invoke and interact with the Web service at a provider's site

## Service Interface & Implementation *(3)*

- When the service interface is represented as a collection of operations defined in WSDL, it does not indicate the order of execution of these operations
  - WSDL represents the service interface as a single monolithic fine grained (simple) artifact
  - It is the responsibility of BPEL managed orchestration to represent a service interface as an assembly (coarse grained / composite) of artifacts

## Definition Element *(1)*

- All the parts of the abstract and concrete sections of WSDL are housed within a root element called the **`<definitions>`** element
- The first attribute is **`name`**, which is used to name the entire WSDL document
- Another attribute is called **`targetNamespace`**, which identifies the logical namespace for elements defined within the WSDL document and characterizes the service
- A namespace prefix **`xmlns:tns`** (sometimes referred to as this namespace) is set to the value of **`targetNamespace`** and is used to qualify (scope) properties of this service definition
- Among the namespace declarations is one for the namespace of the WSDL XML schema **`http://schemas.xmlsoap.org/wsdl/`**

# Definition Element *(2)*

- The namespace definitions **`xmlns:soap`** and **`xmlns:xsd`** are used for specifying SOAP binding specific information as well as XSD data types, respectively

- Example of a definition element

```
<wsdl:definitions name="PurchaseOrderService"
  targetNamespace="http://wwww.auto-parts.com/PurchaseService/wsdl"
  xmlns:tns="http://www.auto-parts.com/PurchaseService/wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
```

## Abstract Service Description *(1)*

- The Web interface definition is used to describe a specific type of interface provided by the service and does not carry any deployment specific details
- It is an abstract service description that can be instantiated and referenced by multiple concrete service implementations
- The Web service public interface contains operational information relating to a Web service including:
  - Data type information for messages
  - All publicly available operations, their purpose and function
  - All message protocols supported by the Web service in order to engage the operations
  - Any constraints that govern the invocation of operations
- In WSDL, the abstract definition of the Web service interface is specified using the **`<types>`**, **`<message>`**, **`<part>`**, **`<operation>`**, and **`<portType>`** elements

# Abstract Service Description *(2)*

| | |
|---|---|
| Types | Defines the data that the Web service uses in its messages through an XML schema definition |
| Message | Defines the messages used by the service; each message refers to a data type |
| Part | Represents either incoming or outgoing operation parameter data |
| Operation | Represents a particular interaction (a Web service function) with the service and is used to describe the input, output, and exception messages that are possible during that interaction |
| PortType | Defines the service interface with an abstract set of operations |

- Each **`<message>`** definition describes the payloads of outgoing and incoming messages (i.e., sent or received by a Web service)
- Messages consist of **`<part>`** elements, each of which represents an instance of a particular type (typed parameter)
- Each **`<operation>`** element is declared by a **`<portType>`** element and contains a number of **`<message>`** definitions describing its input and output parameters as well as any faults

## Types Element *(1)*

- WSDL adopts as its basic type system the W3C XML Schema built-in types
- WSDL provides a specific element for the purpose of defining types for messages;  for this purpose, **`<types>`** element serves as a container that contains all abstract data types that define a Web service interface
- The WSDL **`<types>`** element also contains XML schemas or external references to XML schemas that describe the data type definitions used within the WSDL document
- The WSDL **`<types>`** element refers to an XSD **`simpleType`** or **`complexType`** using a **`QName`**
- Example of type element declarations

```
<wsdl:types>
    <xsd:schema
      targetNamespace="http://www.auto-parts.com/PurchaseService/wsdl">
```

(to be continued)

# Types Element *(2)*

- Example of type element declarations (cont'd)

```
            <xsd:complexType name="CustomerInfoType">
                <xsd:sequence>
                    <xsd:element name="CusName" 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="xsd:integer"/>
                    <xsd:element name="PODate" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
            <xsd:complexType name="InvoiceType">
                <xsd:all>
                    <xsd:element name="InvPrice" type="xsd:float"/>
                    <xsd:element name="InvDate" type="xsd:string"/>
                </xsd:all>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>
```

## Message Element *(1)*

- In WSDL, operation messages are represented by a **`<message>`** element that describes the payload of outgoing and incoming messages
- Each **`<message>`** element corresponds to a single piece of information moving between the invoker and a Web service; consequently, a regular round trip method call is modeled as two messages – one for the request and one for the response
- A **`<message>`** element can contain one or more input/output parameters that belong to an operation; each **`<part>`** element defines one such parameter and represents an instance of a particular type (typed parameter)
- The **`<message>`** element can describe contents of SOAP header blocks and fault detail elements

## Message Element *(2)*

- Example of RPC-style message declarations

```
<!-- message element 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>
<!-- output message -->
<wsdl:message name="InvMessage">
    <wsdl:part name="Invoice" type="tns:InvoiceType"/>
</wsdl:message>
```

- Example of document-style message declarations

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

## Operation and PortType Elements *(1)*

- In WSDL, a typical operation element consists of a group of related input and output messages; execution of an operation requires the transmission or exchange of these messages between the service requester and the service provider
- Operations in WSDL are the equivalent of method signatures in programming languages; they represent the various methods being exposed by the service
- An operation defines a method on a Web service, including the name of the method and the input and output parameters
  - Each **`<operation>`** element is composed of, at most, one **`<input>`** and one **`<output>`** elements, and any number of **`<fault>`** elements
  - The **`<input>`** and **`<output>`** elements, and the order in which they are organized, establishes message exchange patterns (or MEPs) between a Web service and its clients

## Operation and PortType Elements *(2)*

- The central element externalizing a service interface description in WSDL is the `<portType>` element
- A `<portType>` element may have one or more `<operation>` elements, each of which defines an RPC style or document style Web service method
- A `<portType>` element describes the kinds of operation that a Web service supports – the messaging mode and payloads – without specifying the Internet protocol of physical address used
- A WSDL `<portType>` and its associated `<operation>` elements are analogous to a Java interface and its method declarations
- A `<portType>` element is used to bind the collection of logical operations to an actual transport protocol such as SOAP, thus providing the linkage between the abstract and concrete portions of a WSDL document

## Operation and PortType Elements *(3)*

- A WSDL definition can contain zero or more **`<portType>`** definitions; most WSDL documents typically contain a single **`<portType>`**; this convention separates out different Web service interface definitions into different documents
- This granularity allows each business process to have separate binding definitions, providing for reuse, significant implementation flexibility for different security, reliability, transport mechanism, and so on
- Example of PortType and Operation declarations

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

## Sample of Abstract Service Desc. *(1)*

```
<wsdl:definitions name="PurchaseOrderService"
  targetNamespace="http://www.auto-parts.com/PurchaseService/wsdl"
  xmlns:tns="http://www.auto-parts.com/PurchaseService/wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:types>
        <xsd:schema
          targetNamespace="http://www.auto-parts.com/PurchaseService/wsdl">
            <xsd:complexType name="CustomerInfoType">
                <xsd:sequence>
                    <xsd:element name="CusName" 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="xsd:integer"/>
                    <xsd:element name="PODate" type="xsd:string"/>
                </xsd:sequence>
            </xsd:complexType>
```

Abstract data type definitions

(to be continued)

## Sample of Abstract Service Desc. *(2)*

```
            <xsd:complexType name="InvoiceType">
                <xsd:all>
                    <xsd:element name="InvPrice" type="xsd:float"/>
                    <xsd:element name="InvDate" type="xsd: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>
```

Sent data

Returned data

Port type with one operation

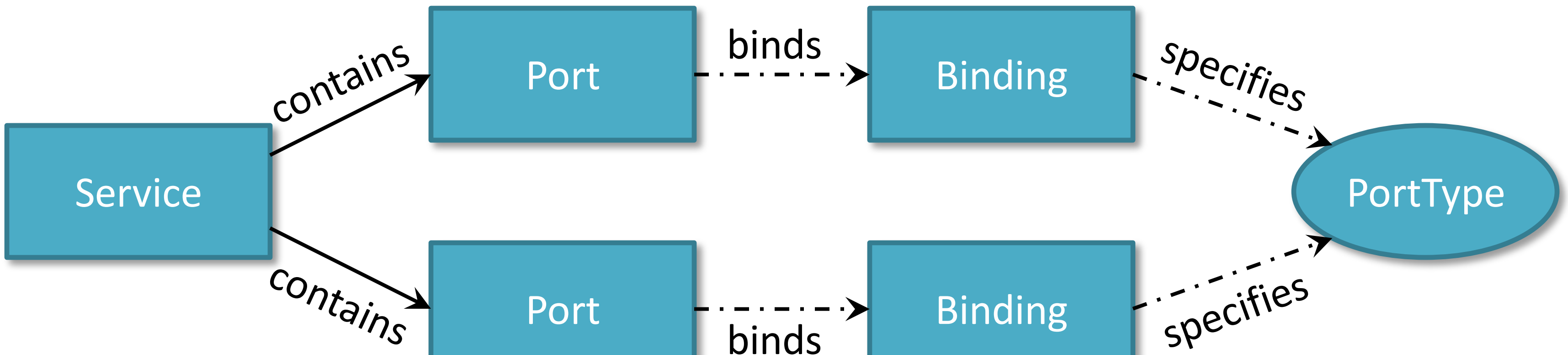
An operation with request (input) and response (output) message

# Concrete Service Description *(1)*

- This part of WSDL 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, to which network address the message must be sent in order to invoke the Web service
- Due to the modularity of WSDL definitions, an abstract description can be offered to different clients via multiple implementations, possibly residing in different locations
- The service implementation part of WSDL contains the elements **`<binding>`**, **`<service>`**, and **`<port>`**

# Concrete Service Description *(2)*

| | |
|---|---|
| Binding | Defines a concrete protocol and data format specification for a particular port type, and specifies binding of each operation in the port type section |
| Service | A collection of related endpoints at which the Web service can be accessed, each consists of a set of port elements, each of which references a binding element |
| Port | Associates between a binding and the network address at which it can be found, and contains endpoint data, including physical address and protocol information |



# Binding Element *(1)*

- The central element of the implementation description is the **`<binding>`** element
- It specifies how the client and Web service should exchange messages
- It maps an abstract **`<portType>`** to a set of concrete protocols (e.g., SOAP and HTTP), message styles (i.e., RPC or document), and encoding styles (e.g., literal or SOAP encoded)
- The structure of **`<binding>`** element resembles that of **`<portType>`** element; in fact, the operation construct that resides within the binding block resembles its counterpart in the interface section

## Binding Element *(2)*

- If a message is to be sent using SOAP over HTTP, the binding describes how the message parts are mapped to elements in the SOAP **`<body>`** & **`<header>`** and what the values of the corresponding attributes are
- Each type of protocol (e.g., MIME, SOAP, HTTP GET, or HTTP POST) has its own set of protocol specific elements and its own namespace
- Example of binding declaration

```
<wsdl:binding name="PurchaseOrderSOAPBinding"
              type="tns:PurchaseOrderPortType">

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

(to be continued)

# Binding Element *(3)*

- Example of binding declaration (cont'd)

```
    <wsdl:operation name="SendPurchase">
        <!-- bind to SOAP now -->
        <soap:operation soapAction=
          "http://www.auto-parts.com/PurchaseService/wsdl/SendPurchase"
          style="rpc"/>
        <!-- specify that the messages in wsdl:operation use SOAP -->
        <wsdl:input>
            <soap:body use="literal" namespace=
              "http://www.auto-parts.com/PurchaseService/wsdl"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" namespace=
              "http://www.auto-parts.com/PurchaseService/wsdl"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

## Binding Element *(4)*

- The binding element in the example is composed of two different namespaces:
  - There are elements that are members of the WSDL 1.1 namespace, indicated by prefix **`wsdl`**
  - There are SOAP specific elements that are members of the namespace for the SOAP-WSDL binding **`"http://schemas.xmlsoap.org/wsdl/soap/"`**, indicated by prefix **`soap`**
- Elements **`<soap:binding>`** and **`<soap:body>`** are to express the SOAP specific details of the Web service;  in the example, the messaging style is RPC & the transport service is SOAP/HTTP
- Element **`<soap:operation>`** is used to link the binding of a specific operation together with the mapping of its input and output messages from the abstract service interface description to a specific SOAP implementation

# Service and Port Elements *(1)*

- The `<service>` element binds the Web service to a specific network addressable location; a `<service>` is modeled as a collection of related `<port>` elements at which a service is made available
- The `<service>` element takes the bindings that were declared previously and ties them to one or more `<port>` elements, each of which represents a physical location for a Web service (often called an *endpoint*)
- A `<port>` element defines the location at which the operation of a specific `<portType>` using a particular transport protocol can be invoked
- Since the `<binding>` specifies a `<portType>`, the `<port>` element effectively ties the `<portType>` to an address

## Service and Port Elements *(2)*

- There could be multiple service implementations for the same service interface provided by different service providers; these all may have different bindings and/or addresses, allowing a service requester to choose the most convenient way to communicate with the service
- Example of service and port declaration

```
<wsdl:service name="PurchaseOrderService">
    <wsdl:port name="PurchaseOrderSOAPPort"
               binding="tns:PurchaseOrderSOAPBinding">
        <!-- give the binding a network endpoint or URI of service -->
        <soap:address location=
          "http://www.auto-parts.com:8080/PurchaseOrderService"/>
    </wsdl:port>
</wsdl:service>
```

- Element `<soap:address>` is another SOAP extension to WSDL used to signify the URI of the service or network endpoint

# Sample of Concrete Service Def. *(1)*

```
<wsdl:definition> ...
    <xsd:import namespace="http://www.auto-parts.com/PurchaseService/wsdl"
      schemaLocation="http://www.auto-parts.com/PurchaseService/wsdl/
        PurchaseOrder-interface.wsdl"/>
    <!-- wsdl:binding states a serialization protocol for this service -->
    <!-- type attribute must match name of portType element -->
    <wsdl:binding name="PurchaseOrderSOAPBinding"
                  type="tns:PurchaseOrderPortType">
        <!-- leverage soap:binding synchronous style -->
        <soap:binding style="rpc"
                      transport="http://schemas.xmlsoap.org/soap/http/"/>
        <wsdl:operation name="SendPurchase">
            <!-- bind to SOAP now -->
            <soap:operation soapAction=
              "http://www.auto-parts.com/PurchaseService/wsdl/SendPurchase"
              style="rpc"/>
            <!-- specify that the messages in wsdl:operation use SOAP -->
            <wsdl:input>
                <soap:body use="literal" namespace=
                  "http://www.auto-parts.com/PurchaseService/wsdl"/>
            </wsdl:input>
```

Bind an abstract operation to this implementation

Map the abstract input message to this concrete message

(to be continued)

## Sample of Concrete Service Def. *(2)*

```
            <wsdl:output>
                <soap:body use="literal" namespace=
                  "http://www.auto-parts.com/PurchaseService/wsdl"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

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

Map the abstract output message to this concrete message

Service name

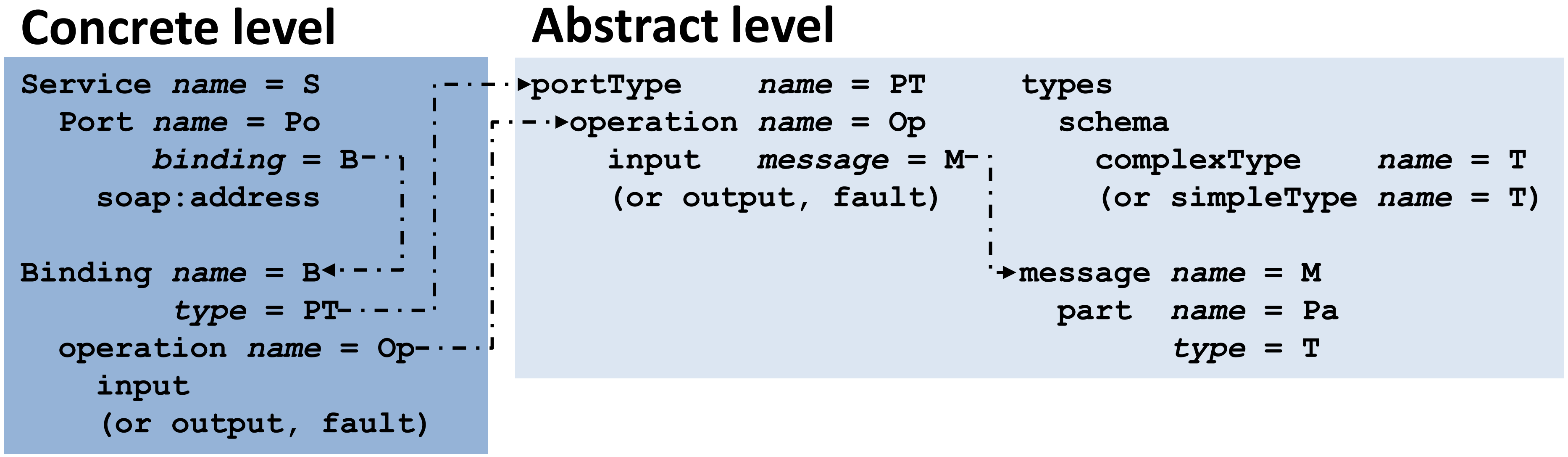
Network address of service

# Abstract & Concrete Mapping *(1)*

```
<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="PurchaseOrderSOAPBinding"
          type="tns:PurchaseOrderPortType">
    <soap:binding style="rpc"
              transport="http://schemas.xmlsoap.org/soap/http/"/>
    <wsdl:operation name="SendPurchase">
        <soap:operation soapAction=
          "http://www.auto-parts.com/PurchaseService/wsdl/SendPurchase"
          style="rpc"/>
        <wsdl:input>
            <soap:body use="literal" namespace=
              "http://www.auto-parts.com/PurchaseService/wsdl"/>
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" namespace=
              "http://www.auto-parts.com/PurchaseService/wsdl"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
 xmlns:soap=...
 xmlns:tns=...>
  <soap:Body>
    <tns:SendPurchase>
      <PurchaseOrder>
        <PONumber> 223451 </PONumber>
        <PODate> 10/28/2004 </PODate>
      </PurchaseOrder>
      ...
    </tns:SendPurchase>
  </soap:Body>
</soap:Envelope>
```

# Abstract & Concrete Mapping *(2)*



**Concrete level**

```
Service name = S
  Port name = Po
       binding = B
    soap:address

Binding name = B
          type = PT
  operation name = Op
    input
    (or output, fault)
```

**Abstract level**

```
portType    name = PT      types
  operation name = Op        schema
    input   message = M        complexType    name = T
    (or output, fault)         (or simpleType name = T)


                             message name = M
                               part  name = Pa
                                     type = T
```

- The above figure shows an overview of how the concrete and abstract levels of WSDL are connected
  - The indented items denote information represented by child elements
  - The attributes are italicized to distinguish them from elements

## WSDL 1.1 vs. WSDL 2.0 *(1)*

- Although WSDL 1.1 is the *de facto* standard, WSDL 2.0 is the version of WSDL that the W3C is currently standardizing
- WSDL 2.0 is a simpler and more usable language than WSDL 1.1; several improvements over WSDL 1.1 include language clarifications & simplifications, support for interoperation, and making it easier for developers to understand & describe services
- The WSDL 1.1 message element was mainly intended to serve as the bridge between message and RPC centric communication; it can be used to describe a document type message based on just one part element, and it can support an RPC type (parameter driven) message based on multiple parts

## WSDL 1.1 vs. WSDL 2.0 *(2)*

- Nevertheless, WSDL 1.1’s expressive power for RPC is limited; e.g., it cannot describe a variable number of input parameters or a choice of responses
- WSDL 2.0 addresses this gap by removing support for the message element altogether;  it simply allows an operation to reference a type directly, using the XML schema type system in the types element

# WSDL 1.1 vs. WSDL 2.0 *(3)*

- WSDL 2.0 introduces some noticeable differences to the document structure of a WSDL definition:
  - The **`definitions`** element is renamed to **`description`**
  - The **`portType`** element is renamed to **`interface`**
    - Support for interface inheritance is achieved by using the **`<extends>`** attribute in the interface element
    - The **`<extends>`** attribute also allows multiple **`<interface>`** declarations to be aggregated together and further extended to produce a completely new **`<interface>`** element
  - The **`port`** element is renamed to **`endpoint`**