Web Service

IF3110 Web-Based Application Development
School of Electrical Engineering and Informatics
Institut Teknologi Bandung

Objective

- Students understand the basic idea of web service and its basic principles
- Students understand various usages and approaches of Web Service
- Students know various techs to develop a Web Service
- Students are able to develop/consume a Web Service

Motivation

- We need to use a service offered by others; and they are connected to Internet
- We intend to develop an application and it reuses/composes other running components
- We intend to access data that are provided by the other application

Definition

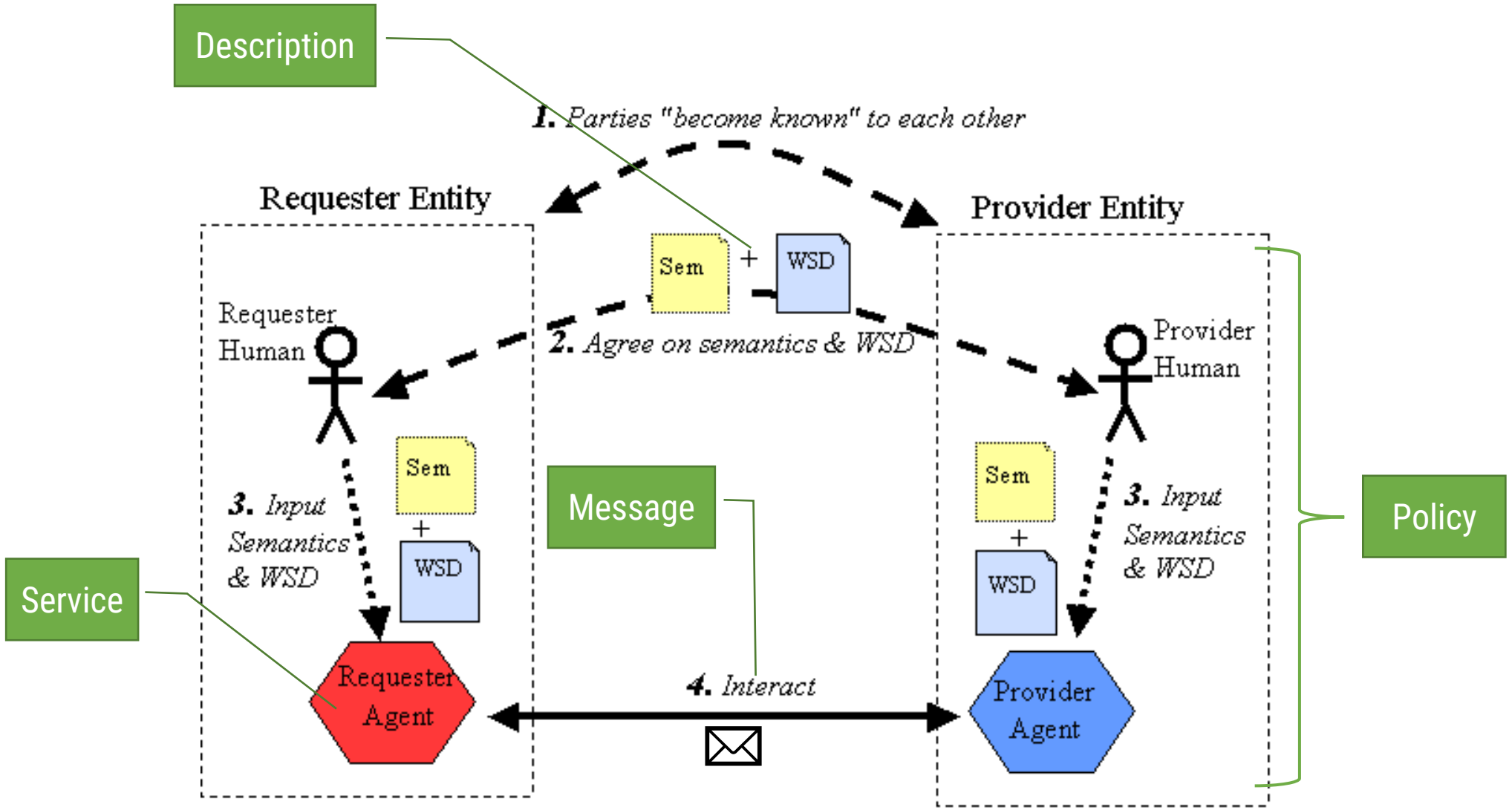
A web service is

- a software system designed
- to support interoperable machine-to-machine interaction over a network.
- It has an interface described in a machine-processable format (specifically WSDL).
- Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.
- — *W3C, Web Services Glossary*

Characteristics

- Simple: application service delivered via Web protocol
- Web friendly: HTTP, XML based format, TCP/IP
- Contract-oriented:
  - Two approaches:
    - SOAP Based (WS-*) Web Services
    - REST style web services
- Technology/language agnostic
- Discoverable

Application Interaction with Web Service



Rationale

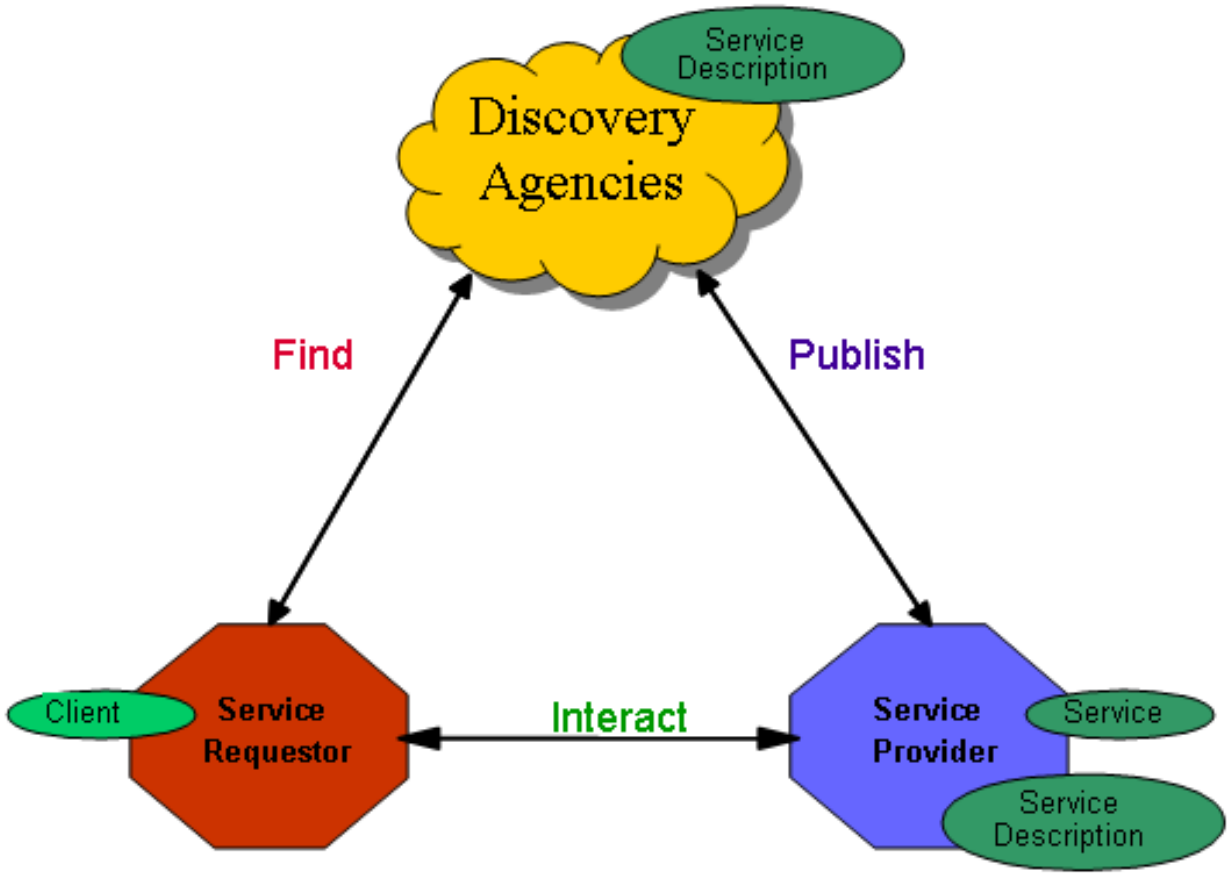
- Web protocol (HTTP, TCP/IP) is popular
- Text based
- loosely coupled
  - each component runs independently/autonomously

Web services

web services is a service-oriented **middleware/technology**

- standards (WSDL, SOAP, WS-Security, WS-Policy, WS-*)
- loose coupling:
  - is a service providing independent functionalities
  - service defined in WSDL without any dependency to the implementation
  - based on asynchronous messaging
- using existing internet technology (HTTP, SMTP)

Service-Oriented Architecture



Service-Oriented Middleware

Middleware: broker

- object-oriented: based on distributed object, synchronous access
- message-oriented: based on message exchange
- event-oriented: similar to message-oriented, but transient

- service-oriented middleware

  based on message exchange, and has clear definition of the service (contract)

Using Internet Technology

- service discovery: why not using DNS?
- interaction:
  - synchronous communication: HTTP
  - asynchronous communication: SMTP

- HTTP & SMTP is often non-blocked traffic by a firewall

XML

```
<?xml version="1.1" encoding="UTF-8" ?>
<!DOCTYPE greeting [
   <!ELEMENT greeting (#PCDATA)>
   ]>

<greeting>Hello, world!</greeting>
```

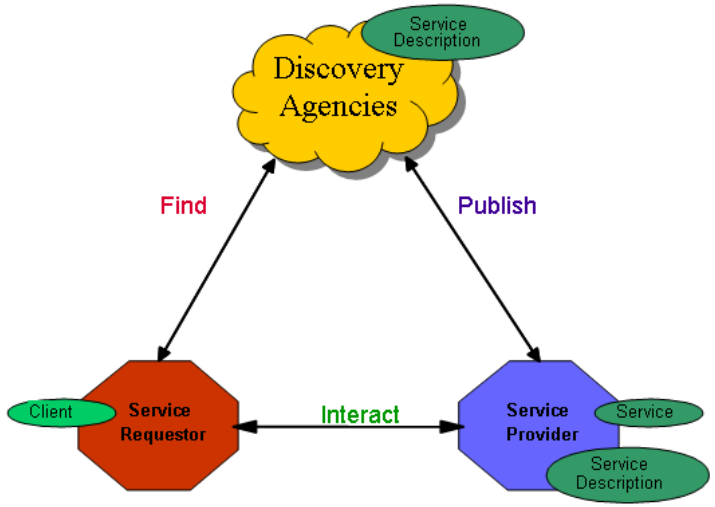
Web Service Standard

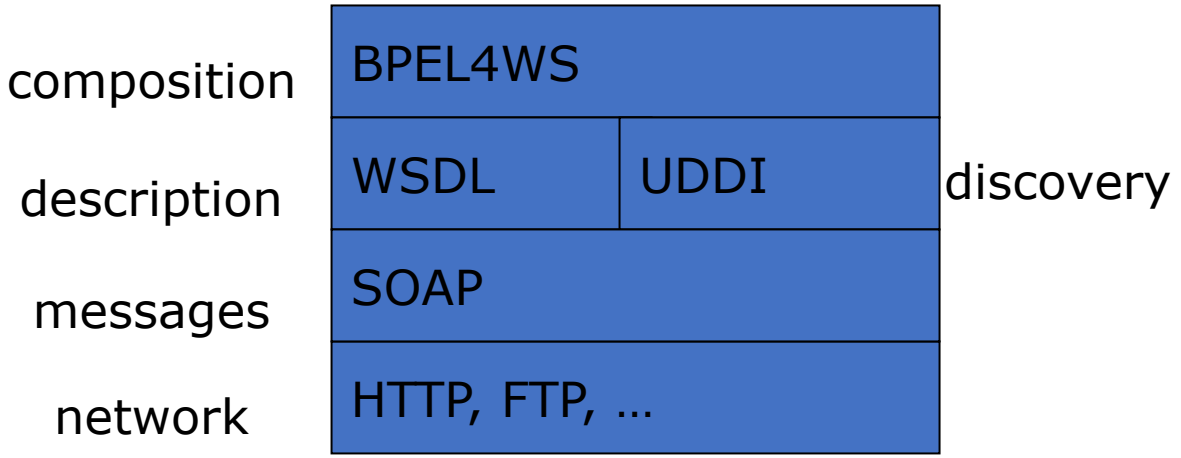
- Most (if not all) Web Service standard defined in XML; thus refer to XML Schema definition
- Example
  - Info about offered goods → uses XML tag …. (XSD of SOAP)
  - Define how to use a web service via WSDL

- XML Schema Definition (.xsd)
  - Verify XML document – supported XML tags and the order of usage
  - Develop a program consuming a service without defining which is the service producer

Web services

- Implementation means to realize services
- Based on open standards:
  - XML
  - SOAP: Simple Object Access Protocol
  - WSDL: Web Services Description Language
  - UDDI: Universal Description, Discovery and Integration
  - BPEL4WS: Business Process Execution Language for Web Services
- Main standardization
- bodies: OASIS, W3C



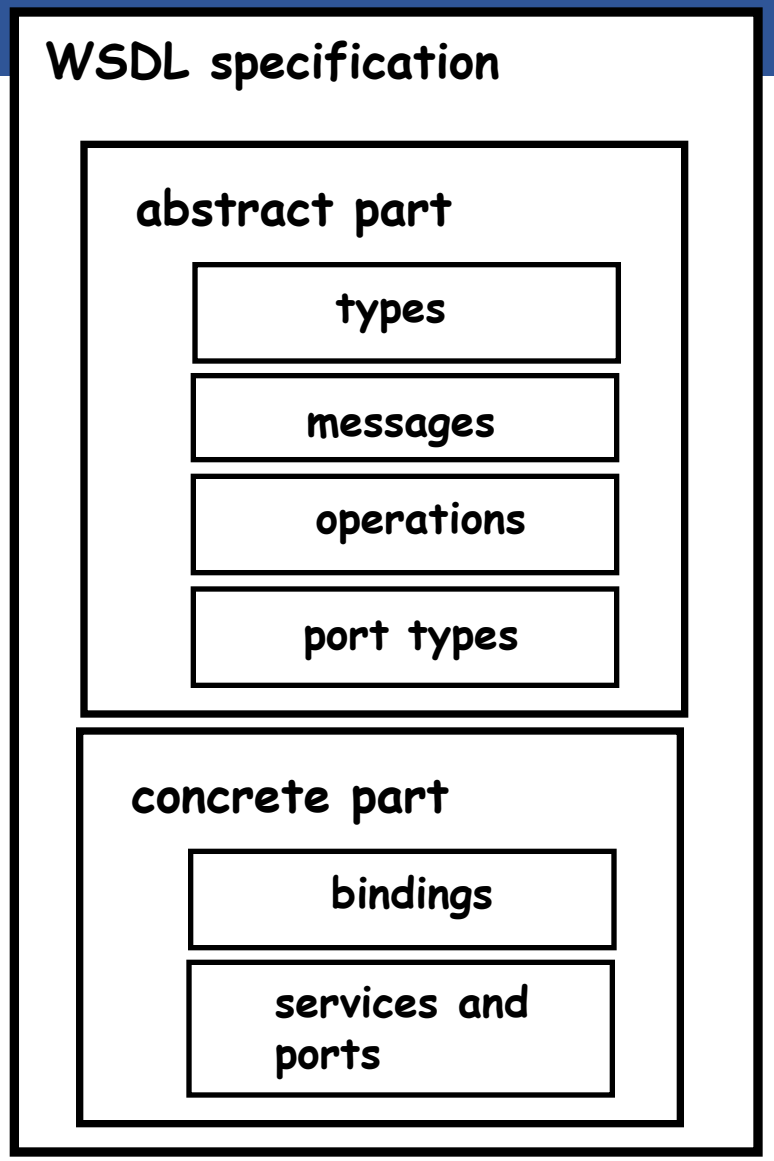


WSDL specification

- WSDL: defined service provided by a service provider
- WSDL contains 6 elements

http://schemas.xmlsoap.org/wsdl/

- Note: WSDL uses XML, defined by an XML Schema



WSDL example

```
<?xml version="1.0"?>
<definitions name="Procurement"
   targetNamespace="http://example.com/procurement/definitions"
   xmlns:tns="http://example.com/procurement/definitions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns="http://schemas.xmlsoap.org/wsdl/" >

   <message name="OrderMsg">
      <part name="productName" type="xs:string"/>
      <part name="quantity" type="xs:integer"/>
   </message>

   <portType name="procurementPortType">
      <operation  name="orderGoods">
         <input message = "OrderMsg"/>
      </operation>
   </portType>

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

   <service name="ProcurementService">
      <port name="ProcurementPort" binding="tns:ProcurementSoapBinding">
         <soap:address location="http://example.com/procurement"/>
      </port>
   </service>

</definitions>
```

abstract part

messages

operation and port type

concrete part

binding

port and service

Communication patterns in WSDL <portType>

there are 4 classes of operations:

- one way (the client initiates this)
- request-response (the client initiates this)
- solicit-response (the service initiates this)
- notification (the service initiates this)

- in the previous example, a request-response type was used (input before output)

WSDL <binding>

WSDL binding defines how SOAP engine does the service binding, e.g., to HTTP

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

SOAP (and SOAP engines)

SOAP is a standard to define:

- Message exchange form
- a set of rules for SOAP engines to exchange SOAP messages
- Convention to implement RPC

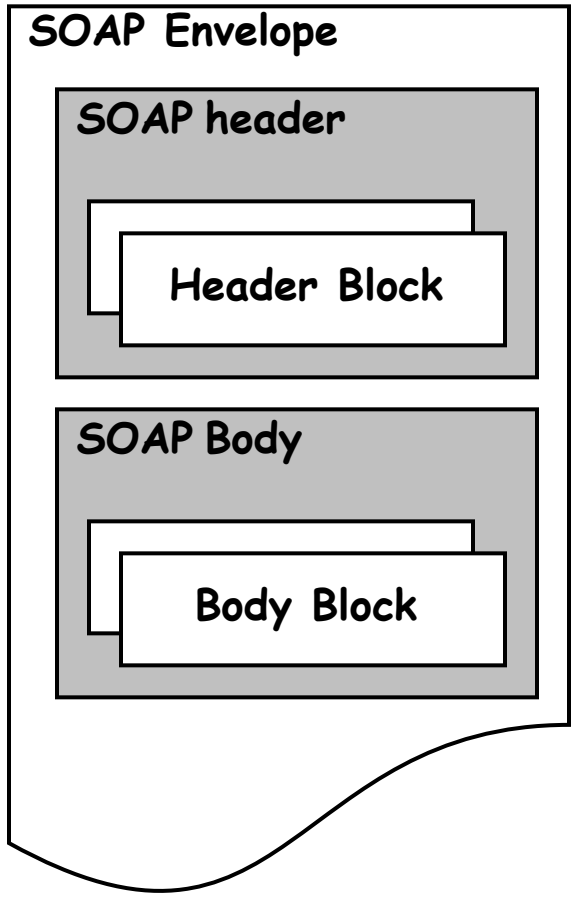
- SOAP uses HTTP or SMTP to deliver the message
- Defined in term of XML Schema

SOAP messages

SOAP envelop wrap data to be sent:

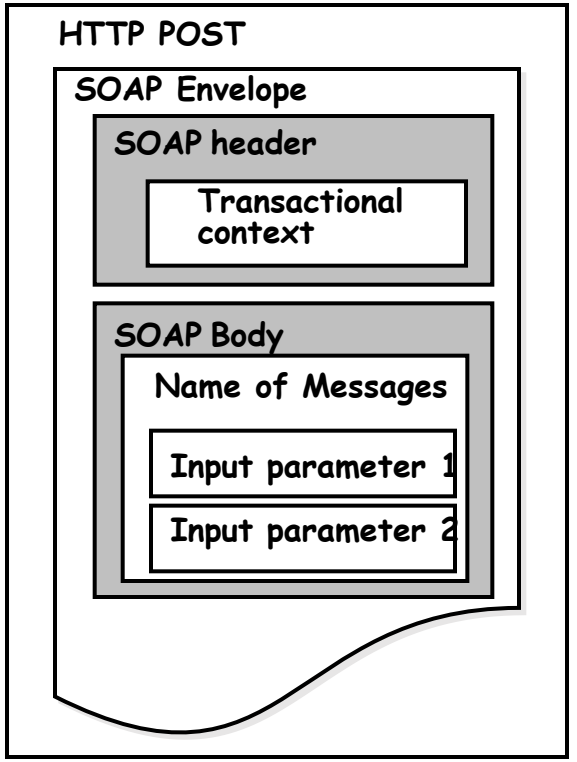
- defined **encoding rules**
- contains **SOAP header**
  - optional
  - infrastructure-level data (e.g., security tokens, routing info, transaction IDs, ...)
- contains **SOAP body**
  - mandatory
  - application-level data (e.g., XML messages according to the WSDL of a service)

note: SOAP message is an XML document, refer to SOAP XML Schema http://schemas.xmlsoap.org/soap/



SOAP over HTTP

- SOAP standard defined how to send SOAP message via HTTP;
- SOAP (also SOAP engines) follows HTTP error codes
- HTTP POST
  - request using SOAP
  - response using SOAP, but in asynchronous messaging only containing the receipt acknowledgment HTTP 202-Accepted



SOAP header and body: example

```
<?xml version='1.0' ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2002/06/soap-envelope" >
 <soap:Header>
  <t:transactionID
       xmlns:t="http://intermediary.example.com/procurement"
       soap:role="http://www.w3.org/2002/06/soap-envelope/role/next"
       soap:mustUnderstand="true" >
     57539
  </t:transactionID>
 </soap:Header>

<soap:Body>
  <m:ex
      soap:encodingStyle="http://www.w3.org/2002/06/soap-encoding"
      xmlns:m="http://example.com/procurement">
   <m:productItem>
      <m:name>ACME Softener</m:name>
   </m:productItem>
   <m:quantity>
      35
   </m:quantity>
  </m:orderGoods>
 </soap:Body>

</soap:Envelope>
```

envelope

header

blocks

body

Web Service Access in PHP

```
<?php

  $wsdl_url =
    'http://services.xmethods.net/soap/urn:
     xmethods-delayed-quotes.wsdl';

  $client = new SOAPClient($wsdl_url);
  $quote = $client->getQuote('EBAY');

  print $quote;
?>
```

Web Service Provider in PHP

```
<?php
  class pc_SOAP_return_time {
   public function return_time() {
     return date('Ymd\THis');
   }
  }

  $server = new SOAPServer(null,
      array('uri'=>'urn:pc_SOAP_return_time'));
  $server->setClass('pc_SOAP_return_time');
  $server->handle();
?>
```

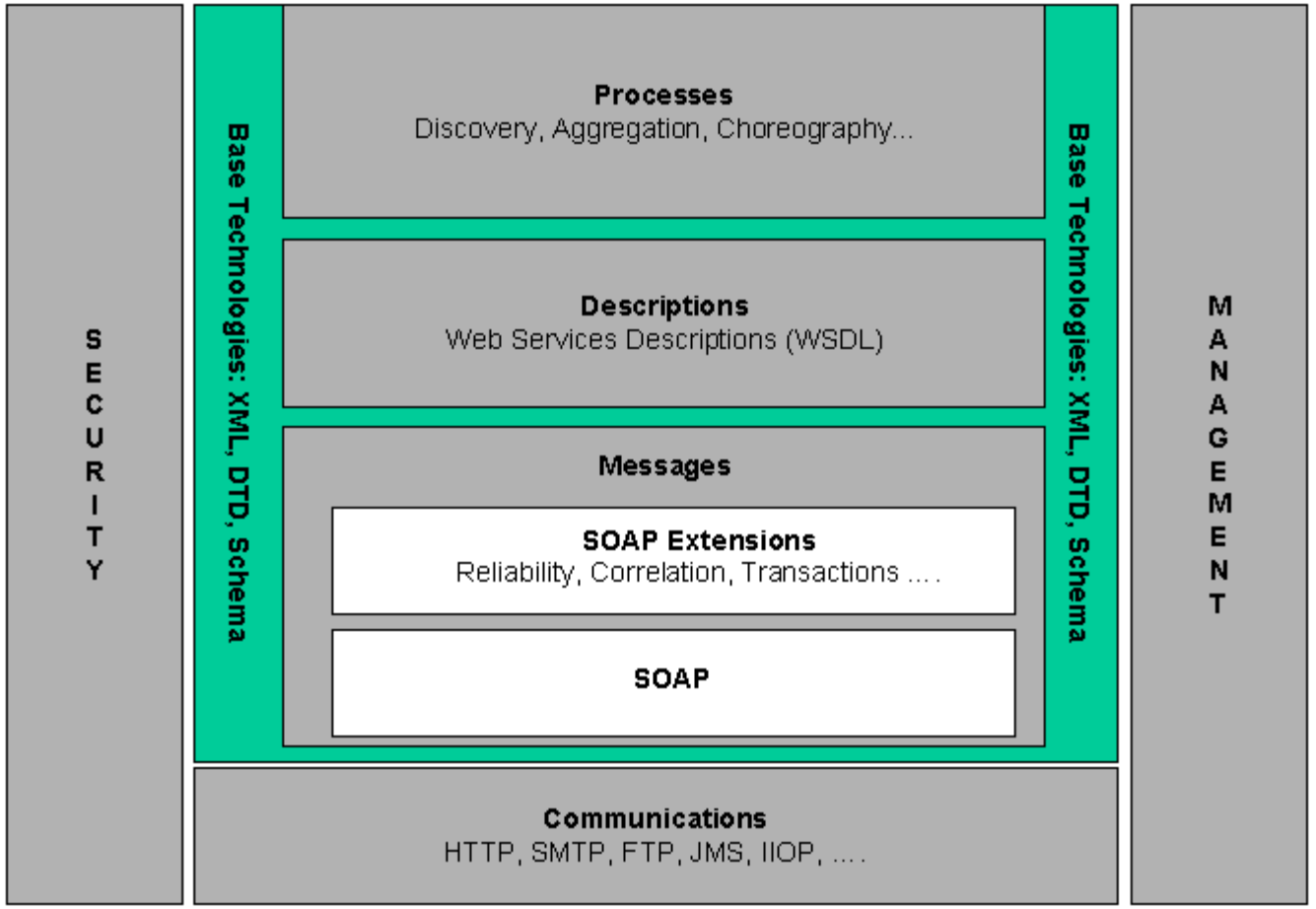
```php
<?php
$opts = array('location' => 'http://localhost/contoh/soapserver.php',
              'uri' => 'http://pc_SOAP_return_time');

$client = new SOAPClient(null, $opts);

$result = $client->__soapCall('return_time', array());

print "The local time is $result.\n";
?>
```

WS Architecture for SOA-system



Discovery

- the act of locating a machine-processable description of a Web service-related resource that may have been previously unknown and that meets certain functional criteria.
- It involves matching a set of functional and other criteria with a set of resource descriptions.

WS Composition

Orchestration

- Executable Process
- the arrangement of **execution sequence of independent agent** to support a business process
- defines the sequence of actions performed by a set of agents
- Language: WS-BPEL

Choreography

- Multi-party collaboration
- the sequence and conditions under which multiple cooperating **independent agents exchange messages** in order to perform a task to achieve a goal state.
- defines the pattern of possible interactions between a set of agents.
- Language: WS-CDL

Web service & distributed objects

- Distributed object systems problems:
  - Problems introduced by latency and unreliability of the underlying transport.
  - The lack of shared memory between the caller and object.
  - The numerous problems introduced by partial failure scenarios.
  - The challenges of concurrent access to remote resources.
  - The fragility of distributed systems if incompatible updates are introduced to any participant.

Web service & distributed objects

- web service is suitable for:
  - platform/vendor neutrality is important; and overhead on performance is trade-able
  - service needs to operate via Internet where reliability and speed un-assured
  - loosely coupled – each component is developed independently
  - multiplatform & vendor
  - has an application that is accessible via network

WS Standard

- W3 Consortium, OASIS,
- Orchestration: WS-Choreography, BPEL
- Transactions: WS-Transaction, RosettaNet
- Reliability: WS-ReliableMessaging, WS-Reliability
- Registries: ebXML, UDDI

ReSTful webservice

Web Service Alternatives

- REST: Representation State Transfer
  - Roy Fielding and his doctoral thesis on“Architectural Styles and the Design of Network-based Software Architectures”.
- resource is accessible via URI
- standard operation/method: create, retrieve, update and delete (similar to HTTP)
- interaction: stateless, simple

Why REST?

- What makes the Web scale?
- How can I apply the architecture of the Web to my own applications?
- Fielding (along with Tim Berners-Lee) designed HTTP and URI's.
- The question he tried to answer in his thesis was "Why is the web so viral"? What is its architecture? What are its principles?

Notes from "RESTFul Java with JAX-RS" by Bill Burke.

Understanding REST

- REST is not protocol specific. It is usually associated with HTTP but its principles are more general.
- SOAP and WS-* use HTTP strictly as a transport protocol.
- But HTTP may be used as a rich application protocol.
- Browsers usually use only a small part of HTTP.
- HTTP is a synchronous request/response network protocol used for distributed, collaborative, document based systems.
- Various message formats may be used – XML, JSON,..
- Binary data may be included in the message body.

REST vs SOAP

- So far we have:
  1. Created a SOAP based web service.
  2. Tested it and retrieved its WSDL.
  3. Generated code based on the WSDL.
  4. Called that code from a client.

- Let's look at the REST design philosophy…

REST vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user
- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:body pb="http://www.acme.com/phonebook">
              <pb:GetUserDetails>
                    <pb:UserID>12345</pb:UserID>
              </pb:GetUserDetails>
        </soap:Body>
</soap:Envelope>
```

REST vs SOAP

- Simple web service as an example: querying a phonebook application for the details of a given user
- And with REST? The query will probably look like this:
  http://www.acme.com/phonebook/UserDetails/12345

- GET /phonebook/UserDetails/12345 HTTP/1.1
  Host: www.acme.com
  Accept: application/xml

- Complex query:
  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

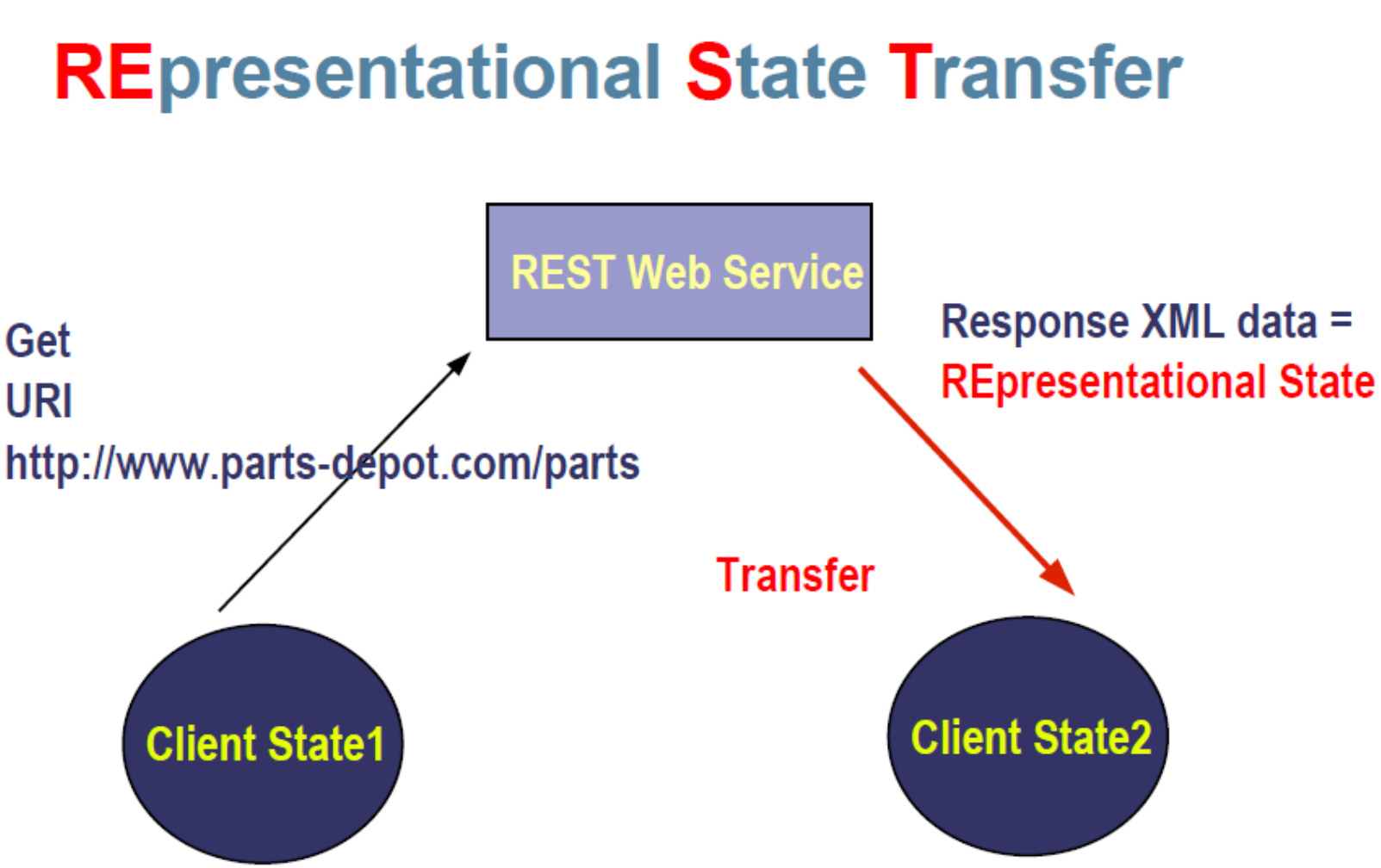
REST Style WS

POST /examples/stringer
Host: www.cdk4.net

Use a URI and an HTTP method to select what needs to be done.

Drop the SOAP and use name value pairs in the request. Use XML or JSON for the response. Don't provide a new set of operations - use HTTP methods instead. Use the same set of principles that made the web go viral!

What is REST?


REpresentational State Transfer
REST Web Service
Get
URI
http://www.parts-depot.com/parts
Response XML data =
REpresentational State
Transfer
Client State1
Client State2

HTTP Request/Response As REST

15

REST Architectural Principles

- The web has addressable resources.

  Each resource has a URI.
- The web has a uniform and constrained interface.
  - HTTP, for example, has a small number of methods. Use these to manipulate resources.
- The web is representation oriented – providing diverse formats.
  - Interaction with services using representations of that service.
  - Different platforms, different formats - browsers -> HTML, JavaScript -> JSON and a Java application -> XML?
- The web may be used to communicate statelessly – providing scalability
- Hypermedia is used as the engine of application state.
  - *Hypermedia As The Engine Of Application State (HATEOAS)*
  - Let your data formats drive state transitions in your applications.

Principle: Addressability

- Addressability (not restricted to HTTP)
  Each HTTP request uses a URI.
  The format of a URI is well defined:

  scheme://host:port/path?queryString#fragment

  The scheme need not be HTTP. May be FTP or HTTPS.
  The host field may be a DNS name or a IP address.
  The port may be derived from the scheme. Using HTTP implies port 80.
  The path is a set of text segments delimited by the "/".
  The queryString is a list of parameters represented as
  name=value pairs. Each pair is delimited by an "&".
  The fragment is used to point to a particular place in a document

Principle: Uniform Interface (1)

- A uniform constrained interface:
- No action parameter in the URI
- HTTP

  GET - read only operation
  - idempotent (once same as many)
  - safe (no important change to server's state)
  - may include parameters in the URI

    http://www.example.com/products?pid=123

Principle: Uniform Interface (2)

HTTP

PUT - store the message body

- insert or update
- idempotent
- not safe

Principle: Uniform Interface (3)

HTTP

POST
- Not idempotent
- Not safe
- Each method call may modify the resource in a unique way
- The request may or may not contain additional information
- The response may or may not contain additional information
- The parameters are found within the request body (not within the URI)

Principle: Uniform Interface (4)

HTTP
DELETE - remove the resource
- idempotent
- Not safe
- Each method call may modify the resource in a unique way
- The request may or may not contain additional information
- The response may or may not contain additional information

HTTP HEAD, OPTIONS, TRACE and CONNECT are less important.

Principle: Uniform Interface (5)

- Does HTTP have too few operations?

- Note that SQL has only four operations: SELECT, INSERT, UPDATE and DELETE
-
- JMS and MOM have, essentially, two
- operations: SEND and RECEIVE
- SQL and JMS have been very useful.

REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- Performed through HTTP methods + URI

CRUD Operations	4 main HTTP methods	
	Verb	**Noun**
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

Why a uniform interface?

Familiarity

We do not need a general IDL that describes a variety of method signatures.

We already know the methods.

Interoperability

WS-* has been a moving target.

HTTP is widely supported.

Scalability

Since GET is idempotent and safe, results may be cached by clients or proxy servers.

Since PUT and DELETE are both idempotent neither the client or the server need worry about handling duplicate message delivery.

Principle: Representation Oriented(1)

- Representations of resources are exchanged.
- GET returns a representation.
- PUT and POST passes representations to the server so that underlying resources may change.
- Representations may be in many formats: XML, JSON, YAML, etc., …

Principle: Representation Oriented(2)

- HTTP uses the CONTENT-TYPE header to specify the message format the server is sending.
- The value of the CONTENT-TYPE is a MIME typed string. Versioning information may be included.
- Examples:
  - text/plain
  - text/html
  - application/json
  - application/vnd+xml;version=1.1
- “vnd” implies a vendor specific MIME type

Principle: Representation Oriented(3)

- The ACCEPT header in content negotiation.
- An AJAX request might include a request for JSON.
- A Java request might include a request for XML.
- Ruby might ask for YAML.

Principle: Communicate Statelessly

- The application may have state but there is no client session data stored on the server.
- If there is any session-specific data it should be held and maintained by the client and transferred to the server with each request as needed.
- The server is easier to scale. No replication of session data concerns.

Principle: HATEOAS

- **H**ypermedia **A**s **T**he **E**ngine **O**f **A**pplication **S**tate
- Hypermedia is document centric but with the additional feature of links.
- With each request returned from a server it tells you what interactions you can do next as well as where you can go to transition the state of your application.

```
<order id = “111”>
        <order-uri>http://…/order/111
    <customer>Alice
            <customer-uri>http://…/customers/3214
    <order-entries>
      <order-entry>
        <qty>5
        <product>Nooodle
                <product-uri>http://…/products/111
```

```
<order id = “111”>
    <customer>Alice
    <order-entries>
      <order-entry>
        <qty>5
        <product>Noodle
```

X

OpenAPI Specification (OAS)

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs.
An OpenAPI file allows you to describe your entire API, including:

- Available endpoints (e.g. /users) and operations on each endpoint (e.g. GET /users, POST /users)
- Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use, and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines.

https://github.com/OAI/OpenAPI-Specification/blob/main/versions/3.1.0.md

Basic Structure: Metadata

Metadata

- Every API definition must include the version of the OAS

```
openapi: 3.0.0
```

Info

- The info section contains API information: title, description (optional), version, contact information, license, terms of service, and other details.

```
info:
  title: Sample API
  description: Optional multiline or single-line description in
  [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.1.9
```

Basic Structure: Servers

Servers

- The servers section specifies the API server and base URL.

```yaml
servers:
  - url: http://api.example.com/v1
    description: Optional server description, e.g. Main (production) server
  - url: http://staging-api.example.com
    description: Optional server description, e.g. Internal staging server
    for testing
```

Basic Structure: Paths

Paths

- The paths section defines individual endpoints (paths) in your API, and the HTTP methods (operations) supported by these endpoints. For example, GET /users can be described as:

```yaml
paths:
  /users:
    get:
      summary: Returns a list of users.
      description: Optional extended description in CommonMark or HTML
      responses:
        "200":
          description: A JSON array of user names
          content:
            application/json:
              schema:
                type: array
                items:
                  type: string
```

OAS Complete Guide

- API Server and Base Path
- Media Types
- Paths and Operations
- Describing Parameters
- Parameter Serialization
- Describing Request Body
- Describing Responses
- Data Models
- Adding Examples
- Authentication
- Links
- Callbacks
- Components Section
- Using $ref
- API General Info
- Grouping Operations With Tags
- OpenAPI Extensions

Read here:

https://swagger.io/docs/specification/v3_0/about/

Misc

- Service vs API vs Endpoint vs Back End
  - What are their similarities?
  - What are their particularities?