Introduction to SOAP, WSDL & Web Services

RESTFULL Web services, WEB API, SSL and OpenID

Why we need web services

- Exposing the existing function on to network:
- Connecting Different Applications (Interoperability)
- Standardized Protocol

Characteristics of Web services

XML-Based

- Web Services uses XML at data representation
- highly interoperable application at their core level.

Loosely Coupled

- A consumer of a web service is not tied to that web service directly.
- A tightly coupled system implies that the client and server logic are closely tied to one another, implying that if one interface changes, the other must be updated.
- software systems more manageable and allows simpler integration between different systems.

Coarse-Grained

- Object-oriented technologies such as Java expose their services through individual methods.
- Building a Java program from scratch requires the creation of several fine-grained methods that are then composed into a coarse-grained service that is consumed by either a client or another service.
- Web services technology provides a natural way of defining coarse-grained services that access the right amount of business logic.

Ability to be Synchronous or Asynchronous

- In synchronous invocations, the client blocks and waits for the service to complete its operation before continuing.
- Asynchronous operations allow a client to invoke a service and then execute other functions.

Supports Remote Procedure Calls(RPCs)

- Web services allow clients to invoke procedures, functions, and methods on remote objects using an XML-based protocol. Remote procedures expose input and output parameters that a web service must support.
- A web service supports RPC by providing services of its own, equivalent to those of a traditional component, or by translating incoming invocations into an invocation of an EJB or a .NET component.

Supports Document Exchange

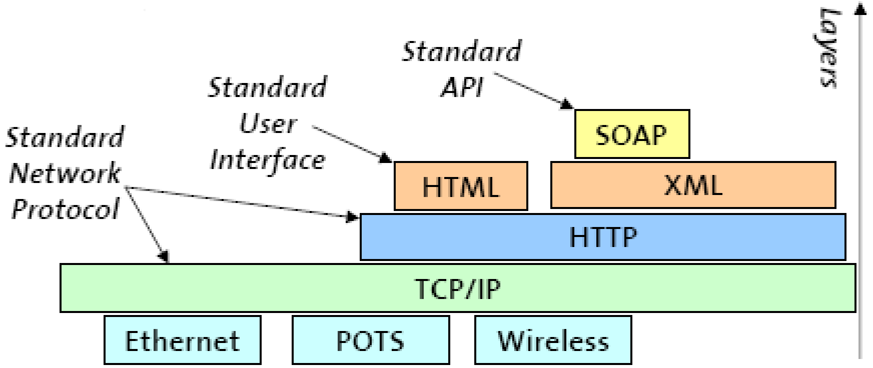
- One of the key advantages of XML is its generic way of representing not only data, but also complex documents.

What is SOAP

- RPC is an obvious and popular paradigm for implementing the client-server model of distributed computing.
- Different RPC protocols exist today.
  - Sun RPC, DCE/RPC, DCOM, CORBA
- Most of the RPC protocols are too low level and certainly not compatible among each other (gateways are needed)
- Since RPC represents a compatibility and security problems, firewalls and proxy servers will normally block this kind of traffic.
- To address this problem, XML was used to define SOAP.
- Soap uses HTTP/SMTP as the communication protocol

Why is SOAP

- XML-based protocol for exchanging information between applications
- Primary focus of SOAP is Remote Procedure Calls transported via HTTP
- Similar to DCOM, CORBA, and Java RMI, the main difference is that SOAP messages are written entirely in XML
- SOAP is therefore uniquely platform and language independent
- For example, a SOAP Java client running on Linux or a Perl client running on Solaris can connect to a Microsoft SOAP server running on Windows 2000

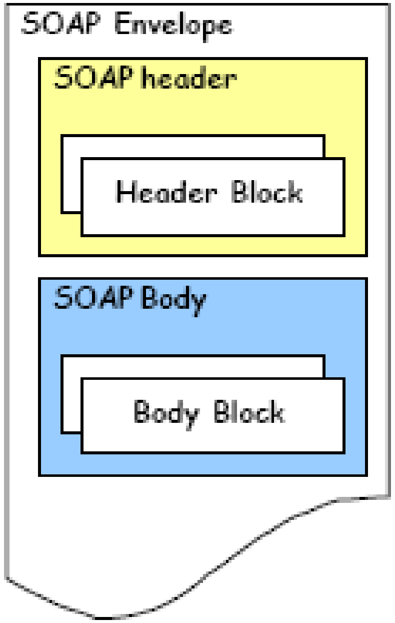


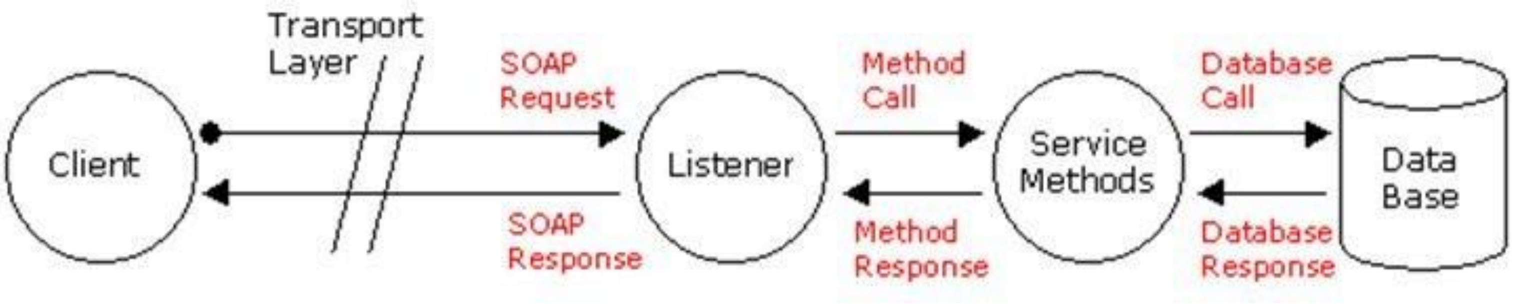
How SOAP works

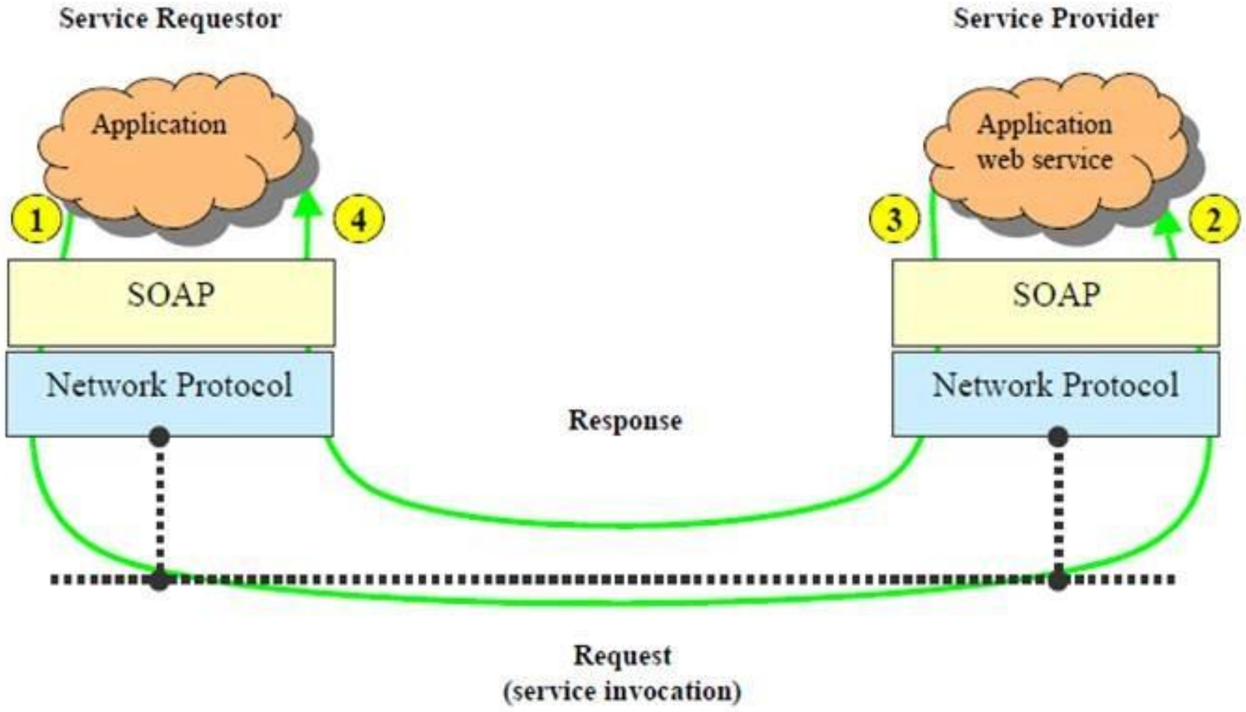
- RPC Using HTTP
  - At the Client
    - Turns the RPC call into a XML document
  - Transports the XML data over HTTP (or SMTP) (Client to Server)
  - At the Server
    - Turns the XML document into a procedure call
    - Turns the procedure's response into a XML document
  - Transports the XML data over HTTP (or SMTP) (Server to Client)
  - At the Client
    - Turns the XML data into the response to the RPC

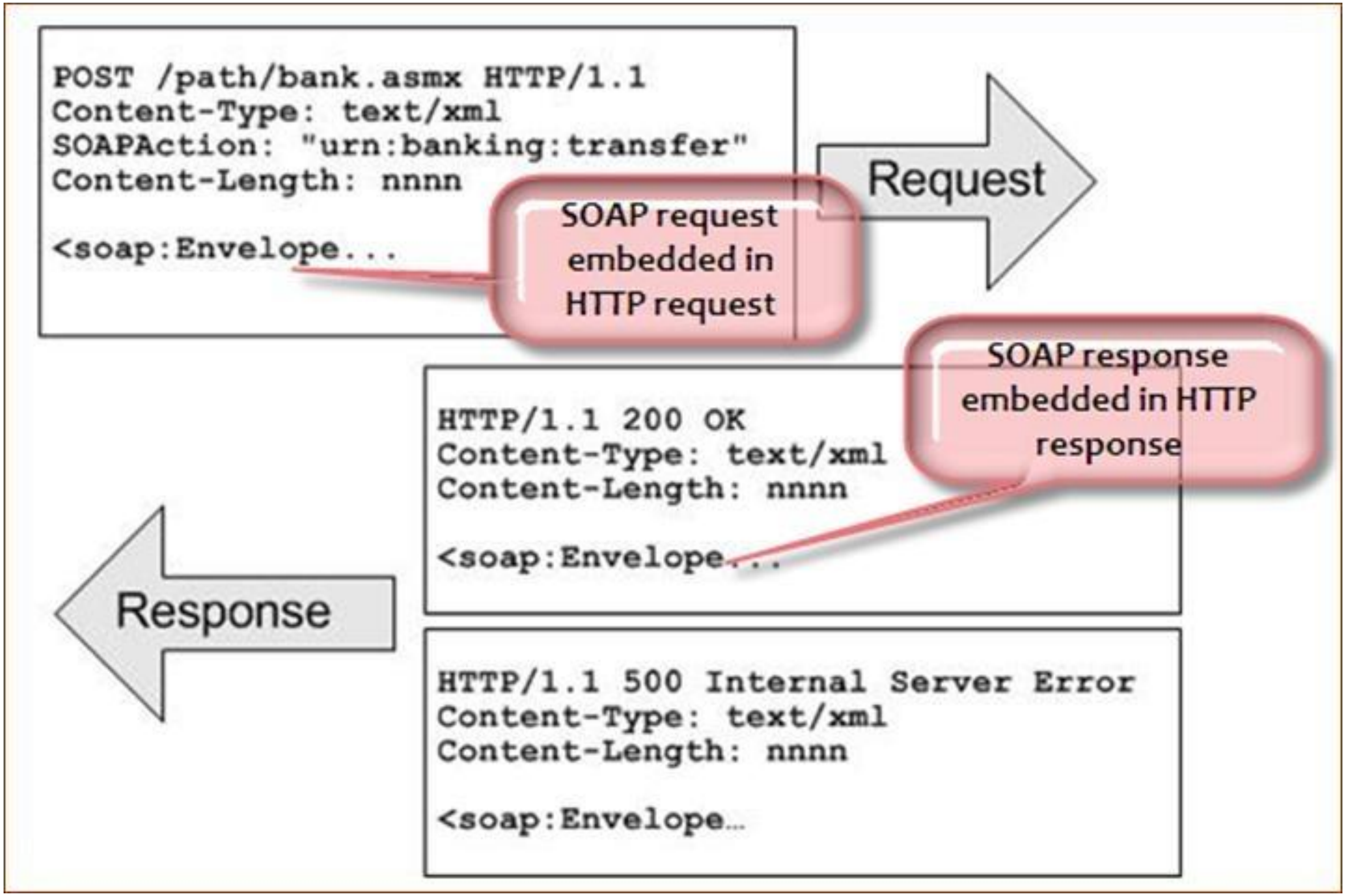
Structure of a SOAP Message

- SOAP is based on message exchanges.
- Messages are seen as envelopes where the application encloses the data to be sent.
- A message has two main parts; header and body, which both can be divided into blocks.
- SOAP does not specify what to do in the body, it only states that the header is optional and the body is mandatory.
- The use of header and body, however, is implicit. The body is for application level data. The header is for infrastructure level data.




Transport Layer
SOAP Request
Method Call
Database Call
Client
Listener
Service Methods
Data Base
SOAP Response
Method Response
Database Response


Service Requestor
Service Provider
Application
Application web service
1
4
3
2
SOAP
SOAP
Network Protocol
Network Protocol
Response
Request
(service invocation)



```
POST /path/bank.asmx HTTP/1.1
Content-Type: text/xml
SOAPAction: "urn:banking:transfer"
Content-Length: nnnn

<soap:Envelope...
```

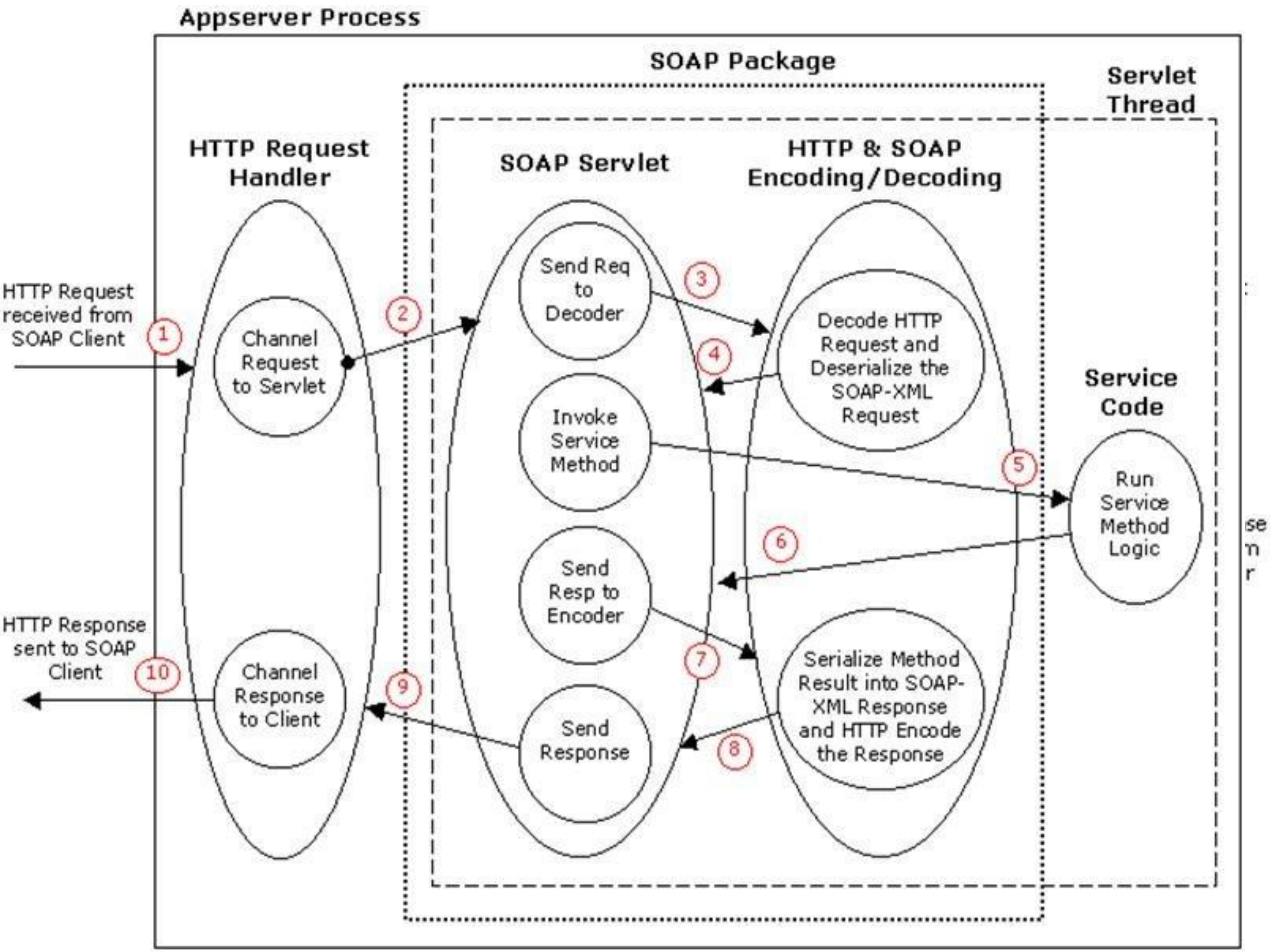
```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<soap:Envelope...
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml
Content-Length: nnnn

<soap:Envelope…
```

Request
SOAP request embedded in HTTP request
SOAP response embedded in HTTP response
Response


Appserver Process
SOAP Package
Servlet Thread
HTTP Request Handler
SOAP Servlet
HTTP & SOAP Encoding/Decoding
Service Code
HTTP Request received from SOAP Client
1
Channel Request to Servlet
2
Send Req to Decoder
3
Decode HTTP Request and Deserialize the SOAP-XML Request
4
Invoke Service Method
5
Run Service Method Logic
6
Send Resp to Encoder
7
Serialize Method Result into SOAP-XML Response and HTTP Encode the Response
8
Send Response
9
Channel Response to Client
10
HTTP Response sent to SOAP Client

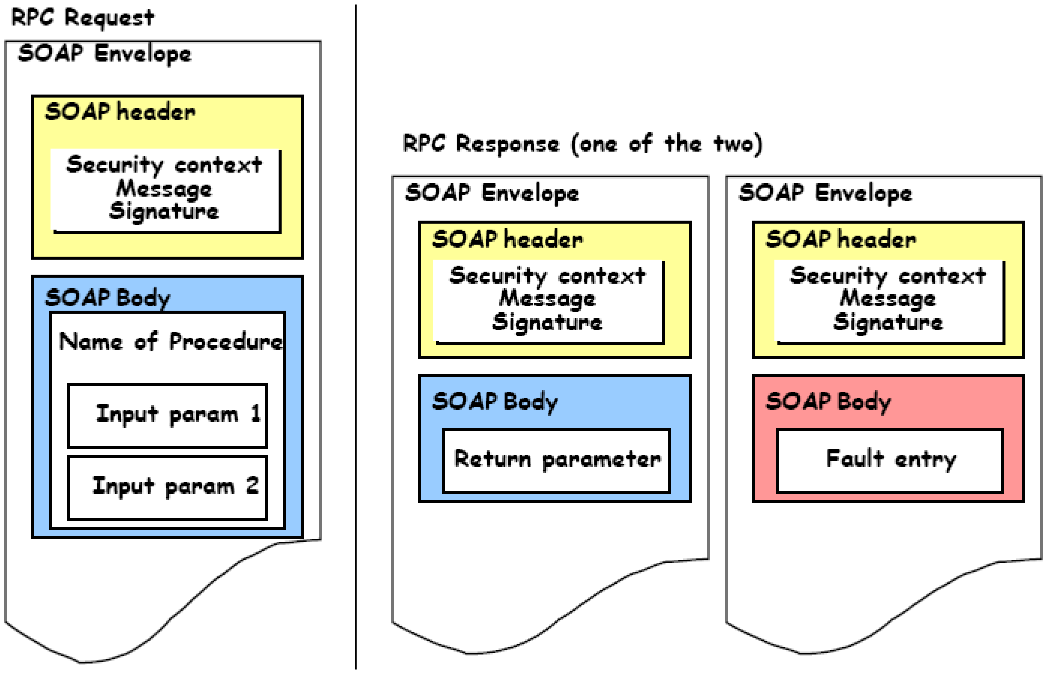
SOAP Header

- The header is intended as a generic place holder for information that is not necessarily application dependent (the application may not even be aware that a header was attached to the message).

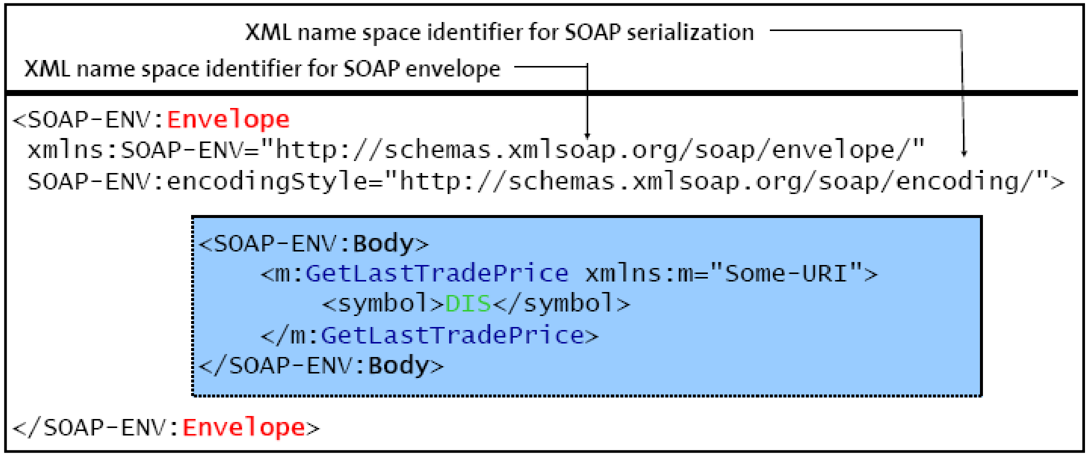
- Typical uses of the header are: coordination information, identifiers (for, e.g.: transactions) and security information (e.g.: certificates)

SOAP Body

- The body is intended for the application specific data contained in the message.
- Also it may contain a Fault entry Section (for reporting errors in processing a SOAP message)



SOAP Message



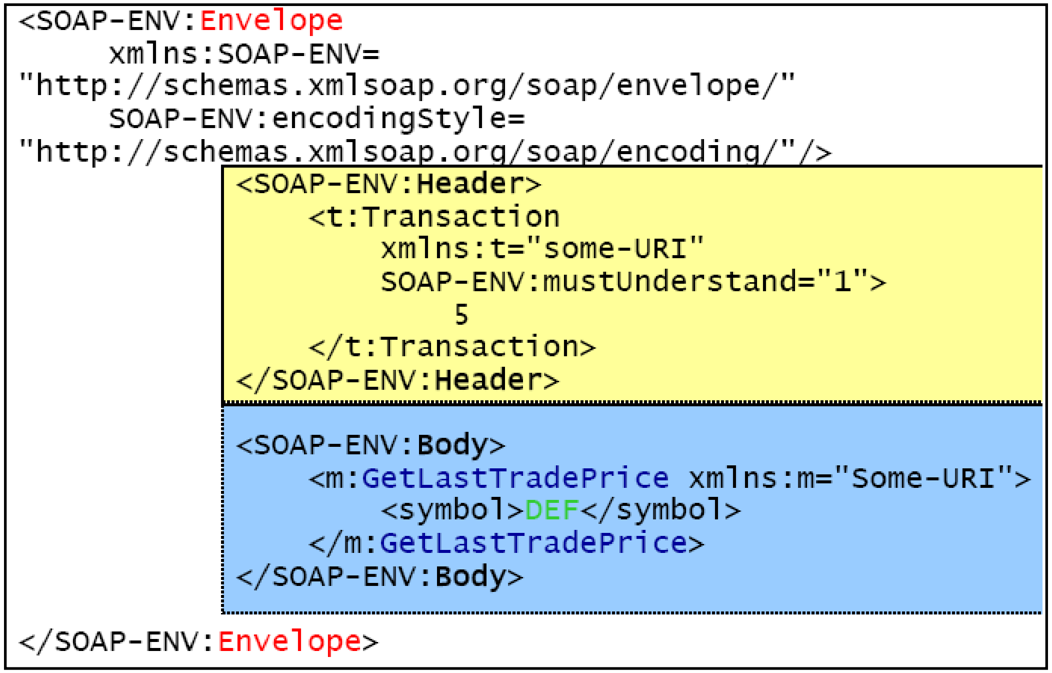
XML name space identifier for SOAP serialization

XML name space identifier for SOAP envelope

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

            <SOAP-ENV:Body>
                <m:GetLastTradePrice xmlns:m="Some-URI">
                    <symbol>DIS</symbol>
                </m:GetLastTradePrice>
            </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```



```
<SOAP-ENV:Envelope
     xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
     SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"/>
            <SOAP-ENV:Header>
                <t:Transaction
                    xmlns:t="some-URI"
                    SOAP-ENV:mustUnderstand="1">
                        5
                </t:Transaction>
            </SOAP-ENV:Header>

            <SOAP-ENV:Body>
                <m:GetLastTradePrice xmlns:m="Some-URI">
                    <symbol>DEF</symbol>
                </m:GetLastTradePrice>
            </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

Introduction to WSDL

```
<?xml version="1.0" encoding="U
<definitions name="AktienKurs":
  targetNamespace="http://loca
  xmlns:xsd="http://schemas.xmlsoap.or
  xmlns="http://schemas.xmlsoap.org/wsd
  <service name="AktienKurs">
    <port name="AktienSoapPort" binding
      <soap:address location="http://loc
    </port>
    <message name="Aktie.HoleWert">
      <part name="body" element="xsd:Tra
    </message>
    …
  </service>
</definitions>
```

What is WSDL?

- WSDL stands for Web Services Description Language

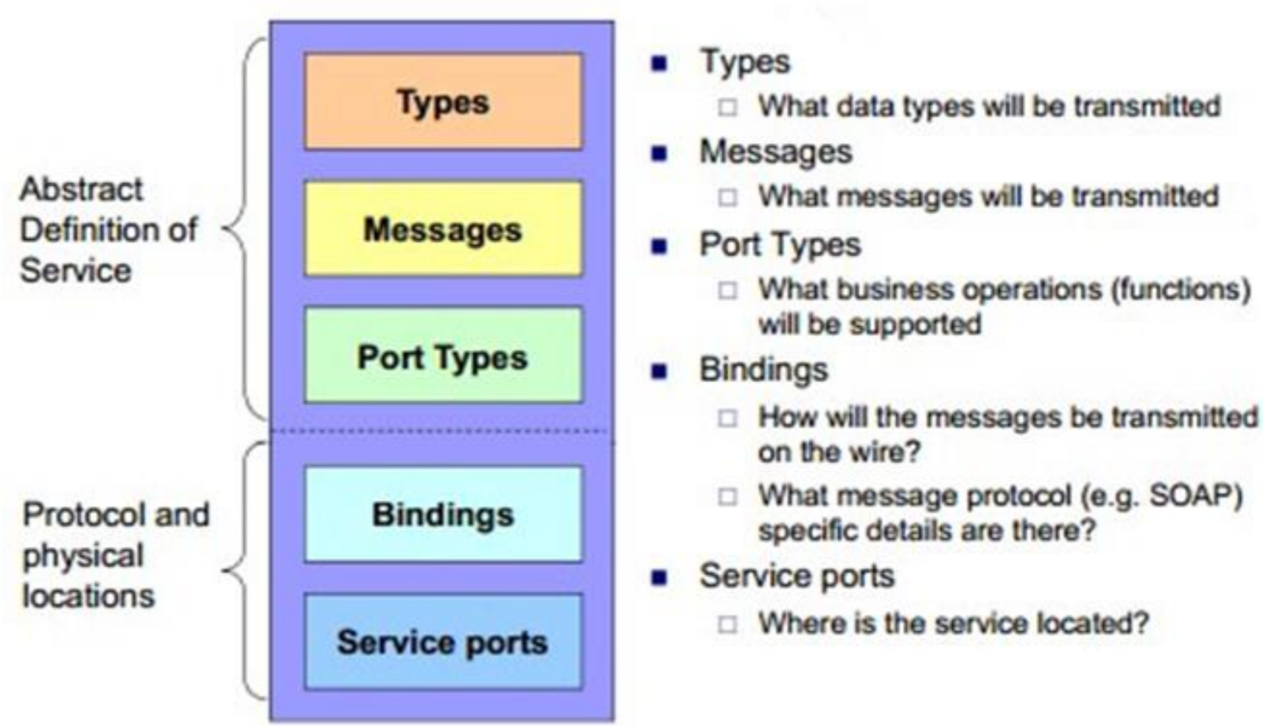
- WSDL is as XML document

- WSDL is used to describe Web services
  - It specifies the location of the service and the operations (or methods) the service exposes

- WSDL is also used to locate Web services

The WSDL Document Structure

- A WSDL document describes a web service using these major elements:


Abstract Definition of Service
Protocol and physical locations
Types
Messages
Port Types
Bindings
Service ports

- Types
  - □ What data types will be transmitted
- Messages
  - □ What messages will be transmitted
- Port Types
  - □ What business operations (functions) will be supported
- Bindings
  - □ How will the messages be transmitted on the wire?
  - □ What message protocol (e.g. SOAP) specific details are there?
- Service ports
  - □ Where is the service located?

WSDL Types

- The **<types>** element defines the data types that are used by the web service.

- For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

WSDL Messages

- The **<message>** element defines the data elements of an operation.

```
<message name="newTermValues">
        <part name="term" type="xs:string"/>
        <part name="value" type="xs:string"/>
</message>
```

- Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.

WSDL Port

- A WSDL port describes the interfaces (legal operations) exposed by a web service.

- It defines **a web service**, the **operations** that can be performed, and the **messages** that are involved.

- Operation Types
  - One-way
  - Request-response
  - Solicit-response
  - Notification

One-Way Operation

```
<message name="newTermValues">
      <part name="term" type="xs:string"/>
      <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
      <operation name="setTerm">
        <input name="newTerm" message="newTermValues"/>
      </operation>
</portType >
```

Request-Response Operation

```
<message name="getTermRequest">
    <part name="term" type="xs:string"/
</message>
<message name="getTermResponse">
    <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
    <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
    </operation>
</portType>
```

WSDL Binding

- WSDL bindings defines the message format and protocol details for a web service.

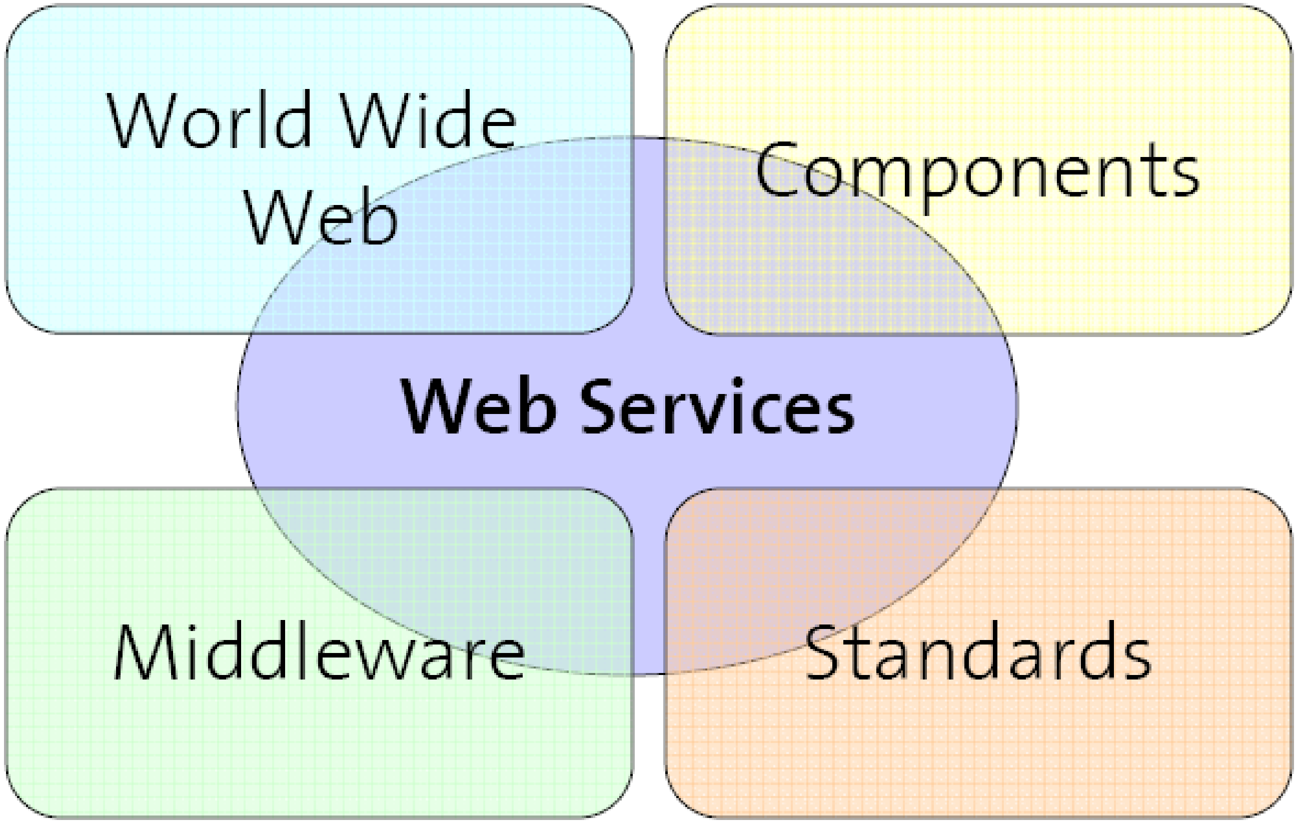
- The **binding** element has two attributes - name and type.
  - The name attribute defines the name of the binding, and the type attribute points to the port for the binding.

- The **soap:binding** element has two attributes - style and transport.

- The **operation** element defines each operation that the port exposes
  - For each operation the corresponding SOAP action has to be defined. You must also specify how the input and output are encoded. .

Introduction to Web Services

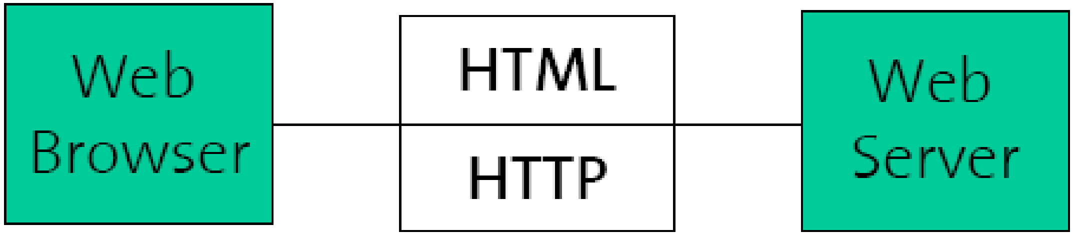
Web Services



Web-based Services & Web Services

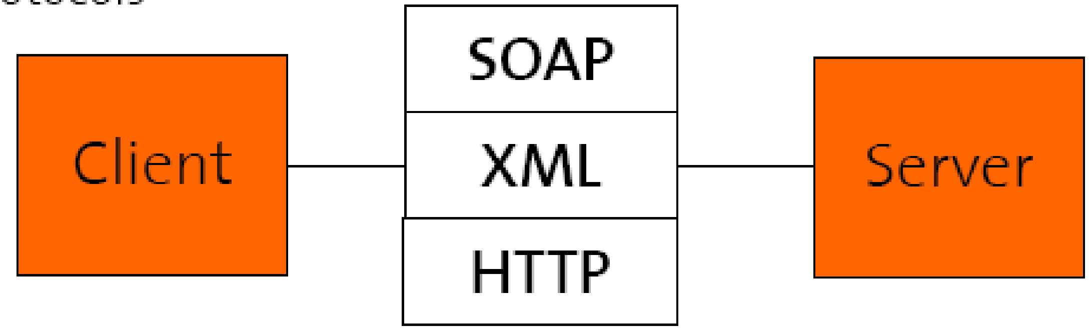
Web-based Services

- □ Services offered through a Web site



Web Services

- □ Services offered through Web-wide standardized protocols



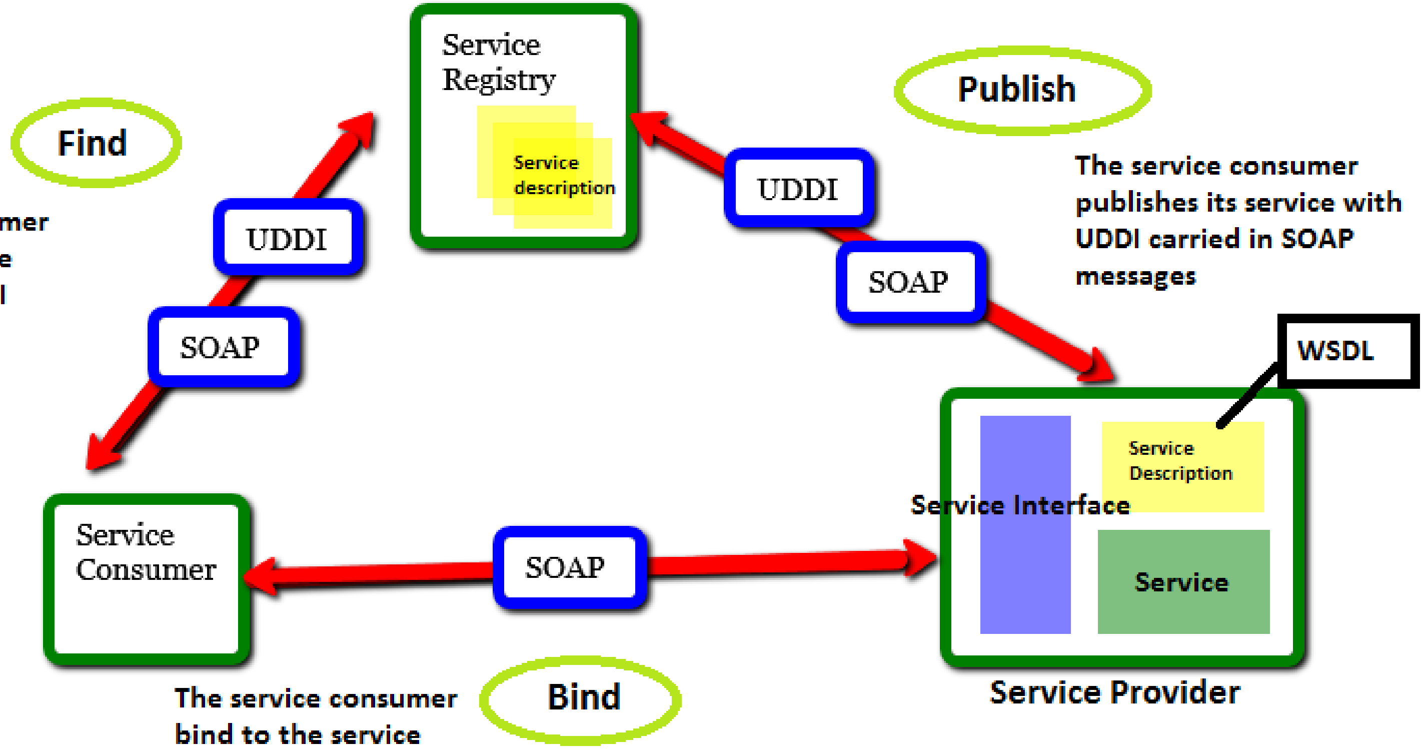
Web Service (W3C Definition)

- “A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via the Internet based protocols”

Web Service

- Web Service definition by W3C emphasizes different aspects:
  - In order to be accessible, a service should be defined, described and discovered.
  - XML is the foundation for all standards that are going to be used (SOAP, WSDL, UDDI)
  - Web services are components that can be readily integrated into more complex distributed applications.
  - Web services are meant for software based consumption (while Web-based applications are meant to be used by humans equipped with a WWW browser)

Web Service Architecture



The service consumer looks up a suitable service using UDDI carried in SOAP messages

The service consumer publishes its service with UDDI carried in SOAP messages

The service consumer bind to the service provider by sending a SOAP request

Web Service Roles

- Service Provider

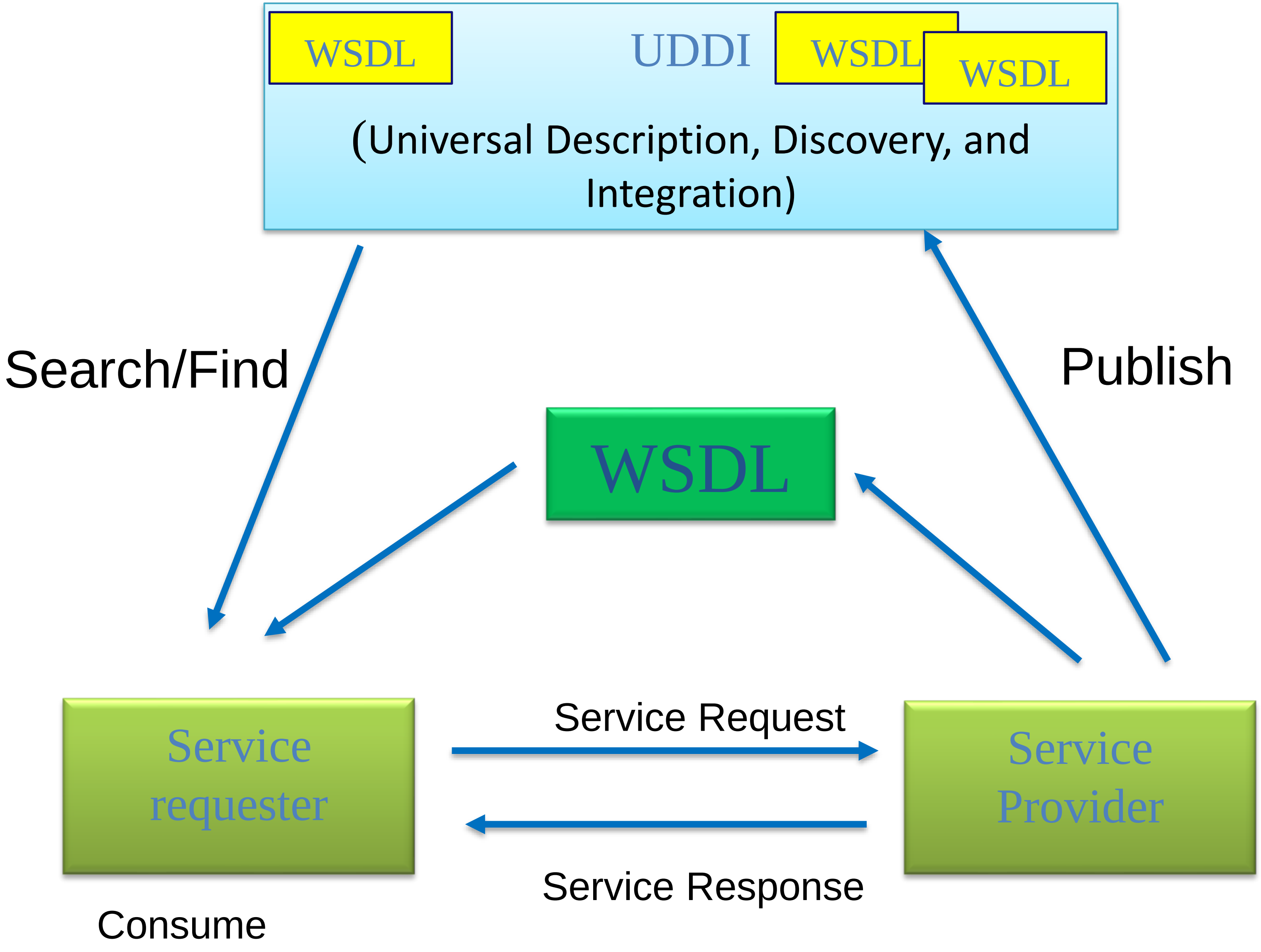
This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

- Service Requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

- Service Registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearing house for companies and their services.



WSDL

UDDI

WSDL

WSDL

(Universal Description, Discovery, and Integration)

Search/Find

Publish

WSDL

Service requester

Service Request

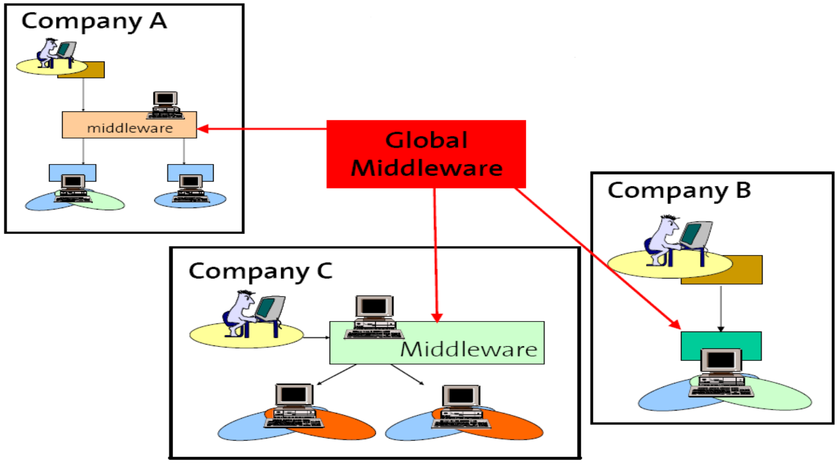
Service Response

Service Provider

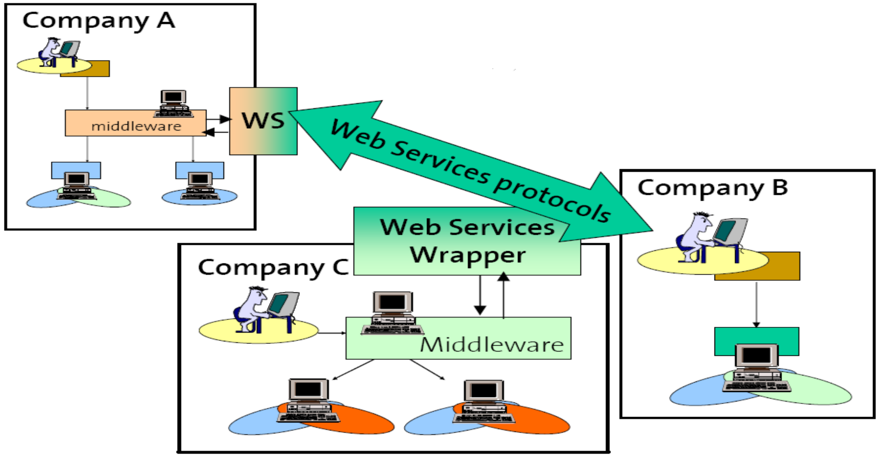
Consume

Benefits of Web Services

- The Web services architecture represented by SOAP, UDDI, and WSDL is a direct descendant of conventional middleware platforms

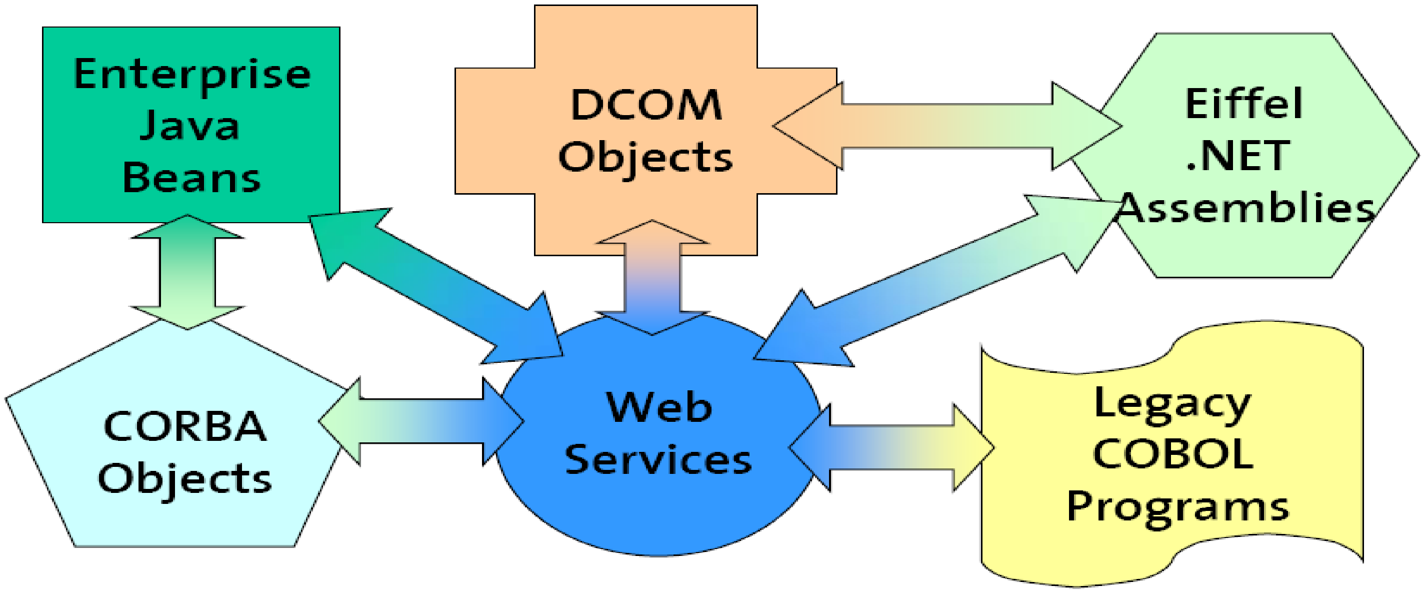


- They can be seen as the most basic extensions that are necessary to allow conventional synchronous (RPC=based) middleware to achieve interoperability



Benefits of Web Services

- Platform independence (Hardware, OS)
- Programming language neutrality
- Portability across Vendor/Middleware tools



Benefits of Web Services

- Low Cost Communication

Web services use SOAP over HTTP protocol, so you can use your existing low-cost internet for implementing web services. Besides SOAP over HTTP, web services can also be implemented on other reliable transport mechanisms like FTP.

- Interoperability

Web services allow various applications to talk to each other and share data and services among themselves. Other applications can also use the web services

- Standardized Protocol

Web services use standardized industry standard protocol for the communication. All the four layers (Service Transport, XML Messaging, Service Description, and Service Discovery layers) use well-defined protocols in the web services protocol stack. This standardization of protocol stack gives the business many advantages such as a wide range of choices, reduction in the cost due to competition, and increase in the quality.

What Is RESTFULL Web Service

RESTful Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications.

the life cycle of a RESTful web service is **per request** so the service does not have to worry about concurrency and can use instance variables safely.

HTTP methods All client server communication on the World Wide Web are done using the following simple HTTP methods:

- **GET**= "give me some info" (Retrieve)
- **POST**= "here's some info to new Info” (Creat)
- **PUT**= "here's info to update" (Update)
- **DELETE**= "delete some info" (Delete)

Two Type of Web Services

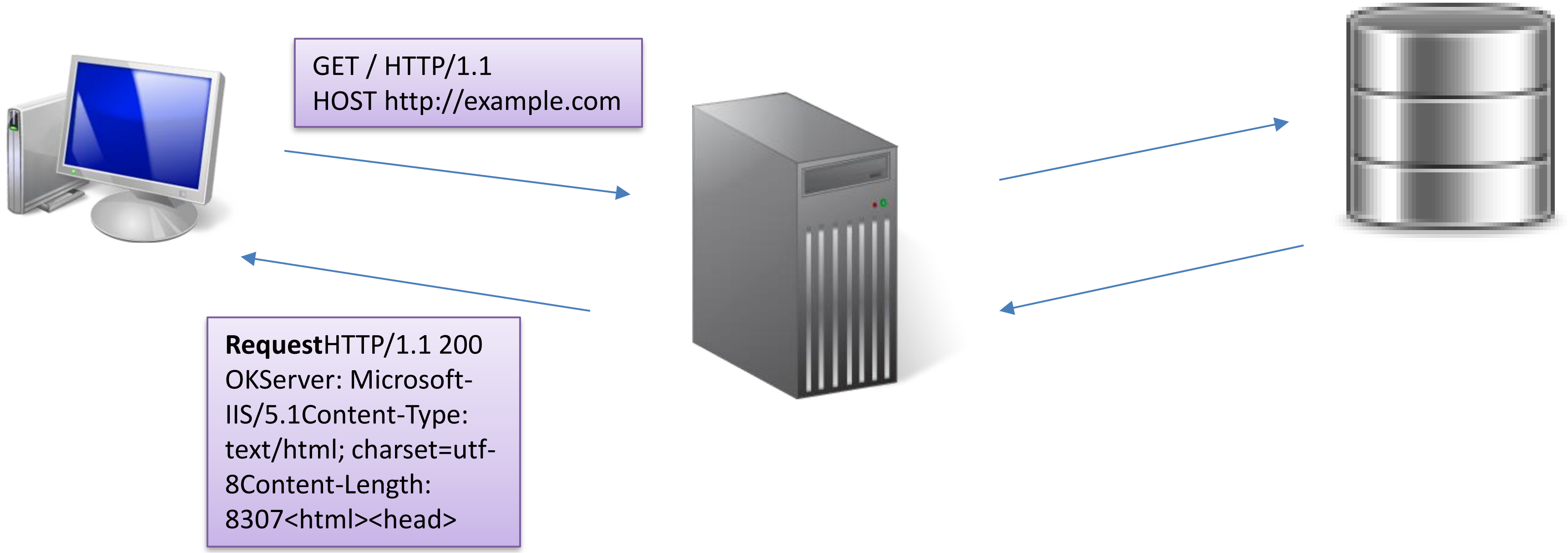
1. Service Oriented web services – mostly used in Enterprise level
2. Resource oriented Web Service

Resource-Oriented Web Services

- Based on “resources”
- Resource -any directly accessible and distinguishable distributed component available on the network.
- RESTfulWeb Services

Architectural style in which clients and servers exchange representations of resources by using a standardized interface and protocol.

How this works



GET / HTTP/1.1
HOST http://example.com

RequestHTTP/1.1 200 OKServer: Microsoft-IIS/5.1Content-Type: text/html; charset=utf-8Content-Length: 8307<html><head>

Resources

- Every distinguishable entity is a resource.
- A resource may be a Web site, an HTML page, an XML document, a Web service, an image, a video *etc.*

Web Services (data, functionality on server side)implemented using HTTP + REST principles

Key elements of RESTfulWeb service are:

- The URI (path) of the Web Service
- The HTTP method supported by the web service.
- The MIME type of the request and response data supported by it.

SOAP vs REST

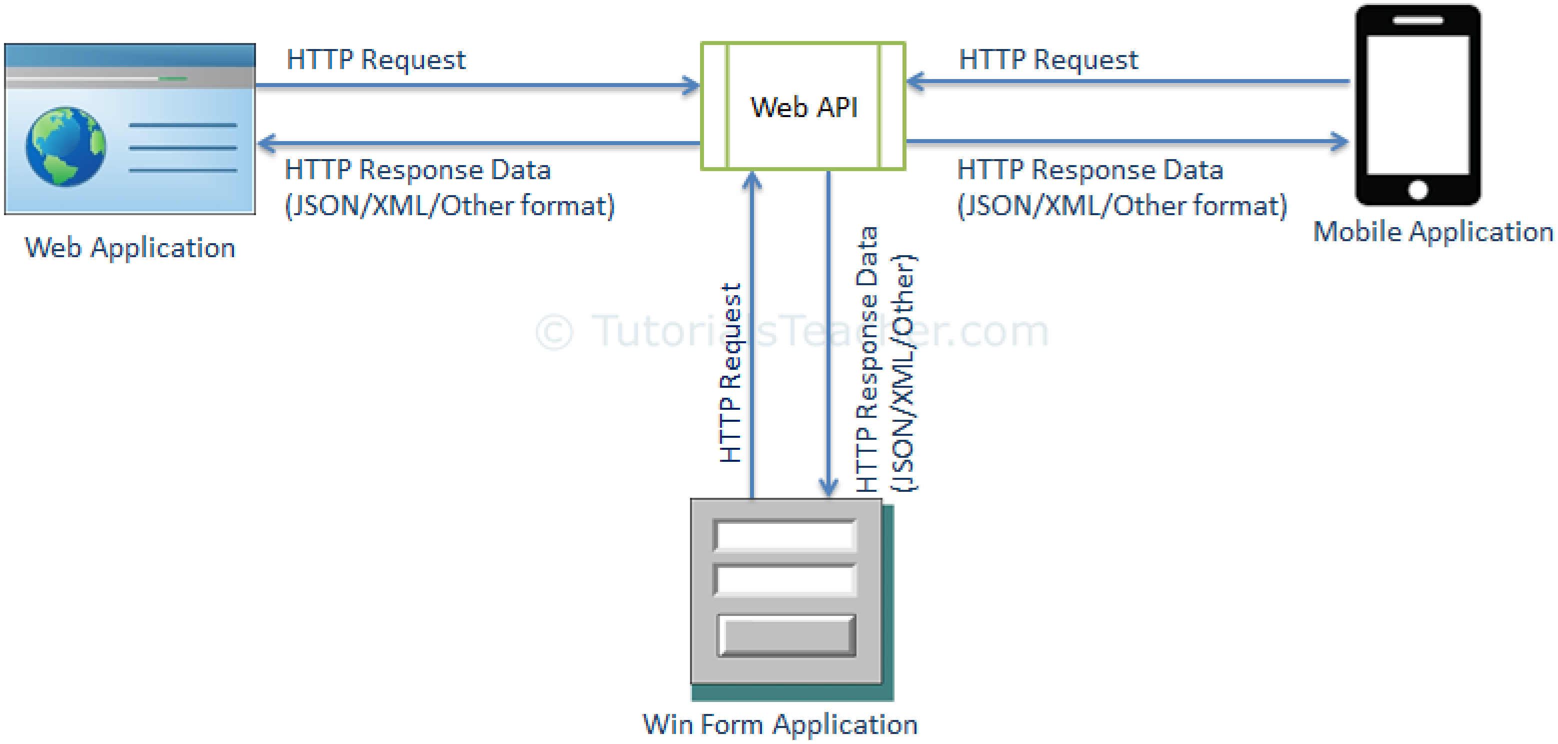
- REST is more dynamic, no need for creating and updating UDDI.
- REST is not restricted to XML format. REST web services can send plain text, JSON, and also XML.
- REST is protocol independent. It's not coupled to HTTP
- REST is as standardized as the parts you're using. Security and authentication in HTTP are standardized, so that's what you use when doing REST over HTTP.
- SOAP is a **protocol**. REST is an **architectural style**
- SOAP **can't use REST** because it is a protocol. REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
- SOAP **requires more bandwidth** and resource than REST. REST **requires less bandwidth** and resource than SOAP.
- REST has better performance and scalability. REST reads can be cached, SOAP based reads cannot be cached.

WEB - API

Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. Web API is an ideal platform for building RESTful applications

API (Application Programing Interface).

Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc.



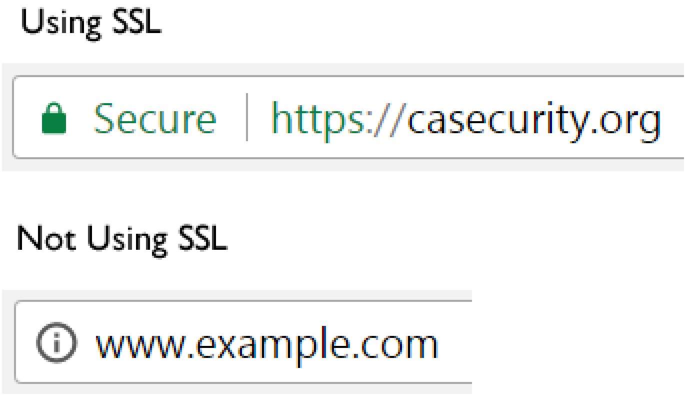
SSL

SSL (Secure Sockets Layer) is the standard **security technology** for establishing an **encrypted link** between a **web server and a browser**. This link ensures that all data passed between the web server and browsers **remain private and integral**. SSL is an **industry standard** and is used by millions of websites in the protection of their online transactions with their customers

One of the most important components of **online business is creating a trusted environment where potential customers feel confident in making purchases**. SSL certificates create a foundation of trust by establishing a secure connection and browsers give visual cues, such as a lock icon or a green bar, to help visitors know when their connection is secure.

How to identify the website is Secure with SSL

Technically, SSL is a transparent protocol which requires little interaction from the end user when establishing a secure session. In the case of a browser, you can tell if a site is using SSL when a padlock is displayed or the address bar shows the URL as HTTPS instead of HTTP.



Using SSL

🔒 Secure | https://casecurity.org

Not Using SSL

ⓘ www.example.com

With so much of our day to day transactions and communications happening online, there is very little reason for not using SSL. SSL supports the following information security principles:

Encryption: protect data transmissions (e.g. browser to server, server to server, application to server, etc.)

Authentication: ensure the server you're connected to is actually the correct server.

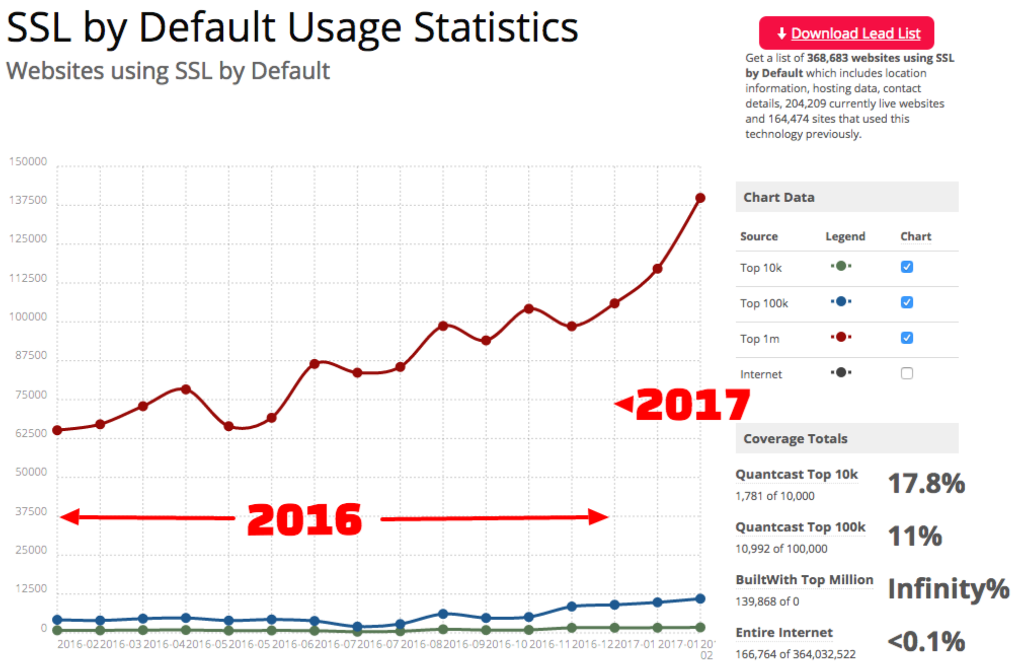
Data integrity: ensure that the data that is requested or submitted is what is actually delivered.

SSL can be used to secure:

- Online credit card transactions or other online payments.
- Intranet-based traffic, such as internal networks, file sharing, extranets and database connections.
- Webmail servers like Outlook Web Access, Exchangeand Office Communications Server.

How Do I Get SSL?

To adopt SSL in your business, you should purchase an SSL Certificate.



SSL by Default Usage Statistics

Websites using SSL by Default

Download Lead List

Get a list of **368,683 websites using SSL by Default** which includes location information, hosting data, contact details, 204,209 currently live websites and 164,474 sites that used this technology previously.

Chart Data

Source	Legend	Chart
Top 10k	●	☑
Top 100k	●	☑
Top 1m	●	☑
Internet	●	□

Coverage Totals

Quantcast Top 10k 1,781 of 10,000	**17.8%**
Quantcast Top 100k 10,992 of 100,000	**11%**
BuiltWith Top Million 139,868 of 0	**Infinity%**
Entire Internet 166,764 of 364,032,522	**<0.1%**

SSL Handshake

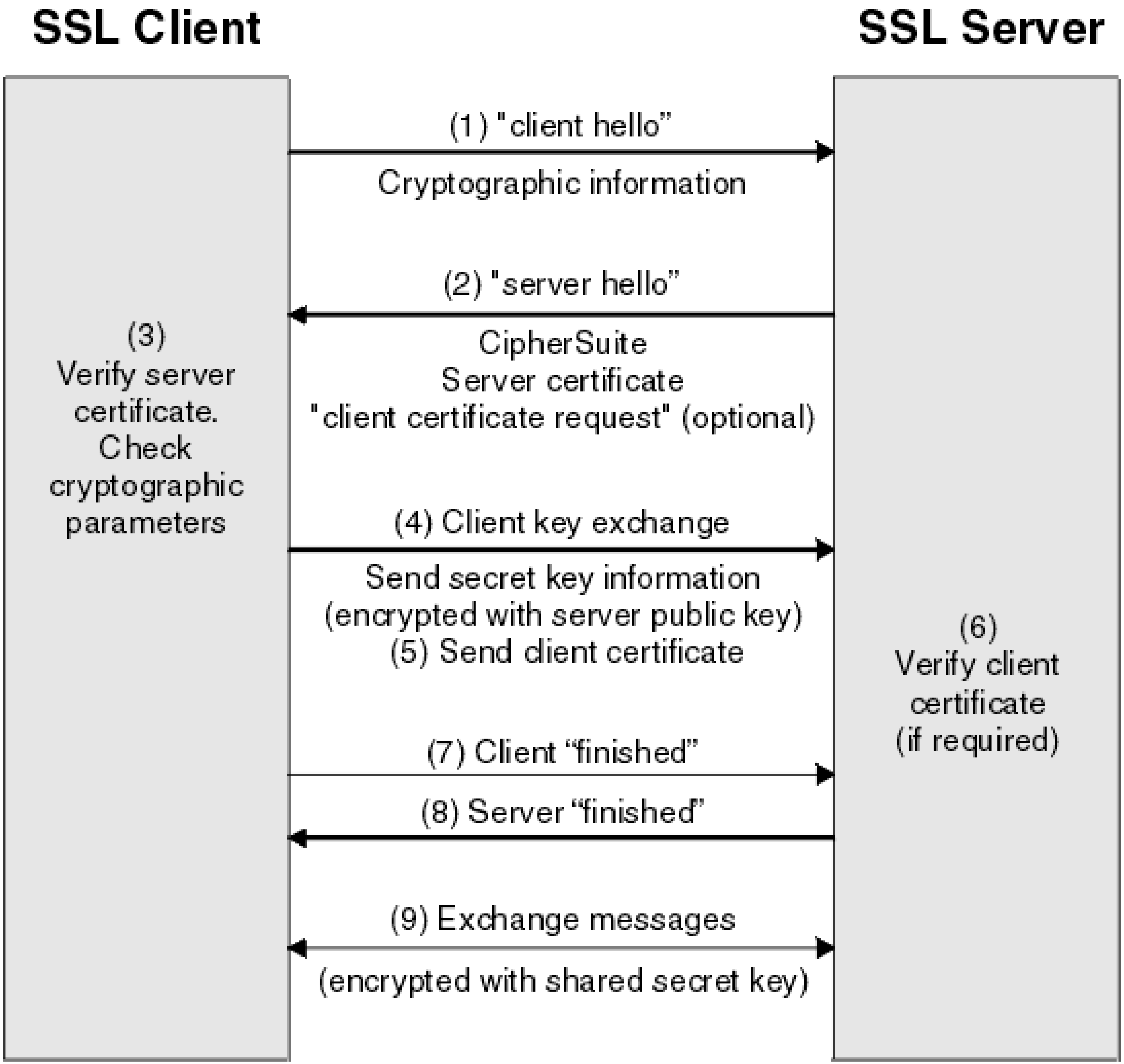
Client Hello
Information that the server needs to communicate with the client using SSL. This includes the SSL version number, cipher settings, session-specific data.

Server Hello
Information that the server needs to communicate with the client using SSL. This includes the SSL version number, cipher settings (public key), session-specific data.

Authentication and Pre-Master Secret
Client authenticates the server certificate. (e.g. Common Name / Date / Issuer) Client (depending on the cipher) creates the pre-master secret for the session, Encrypts with the server's public key and sends the encrypted pre-master secret to the server.

Decryption and Master Secret
Server uses its private key to decrypt the pre-master secret. Both Server and Client perform steps to generate the master secret with the agreed cipher.

Encryption with Session Key
Both client and server exchange messages to inform that future messages will be encrypted


SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3)
Verify server
certificate.
Check
cryptographic
parameters
(4) Client key exchange
Send secret key information
(encrypted with server public key)
(5) Send client certificate
(6)
Verify client
certificate
(if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

OpenID

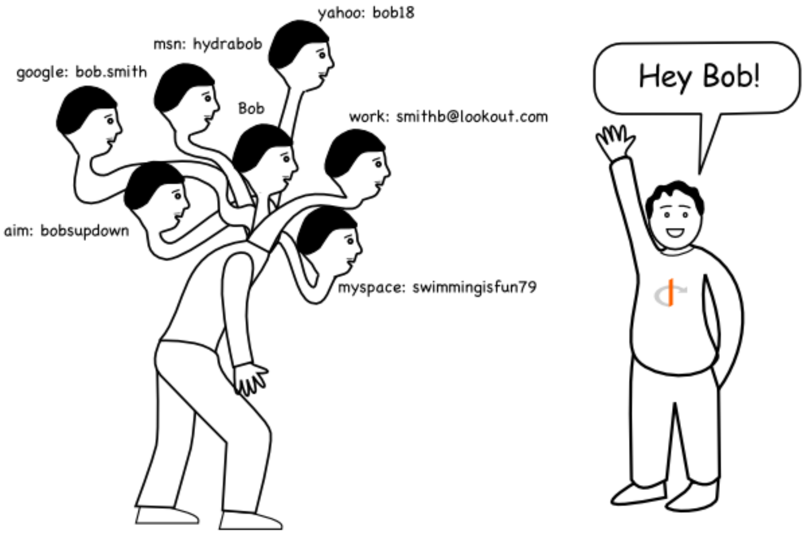
OpenID allows you to use an existing account to sign in to multiple websites, without needing to create new passwords.

You may choose to associate information with your OpenID that can be shared with the websites you visit, such as a name or email address. With OpenID, you control how much of that information is shared with the websites you visit.

With OpenID, your password is only given to your identity provider, and that provider then confirms your identity to the websites you visit. Other than your provider, no website ever sees your password, so you don't need to worry about an unscrupulous or insecure website compromising your identity.

Promoted by the non-profit OpenID Foundation, it allows users to be authenticated by co-operating sites – (WIKIPEDIA)

OpenID also simplifies signing in. With OpenID **you only have to remember one username and one password**



OpenID vs OAuth

OpenID is about authentication (ie. proving who you are), OAuth is about authorization (ie. to grant access to functionality/data/etc.. without having to deal with the original authentication).

First the scenario for OpenID:

- User wants to access his account on example.com
- example.com (the “Relying Party” in OpenID lingo) asks the user for his OpenID
- User enters his OpenID
- example.com redirects the user to his OpenID provider
- User authenticates himself to the OpenID provider
- OpenID provider redirects the user back to example.com
- example.com allows the user to access his account

And now the scenario for OAuth:

- User is on example.com and wants to import his contacts from mycontacts.com
- example.com (the “Consumer” in OAuth lingo) redirects the user to mycontacts.com (the “Service Provider”)
- User authenticates himself to mycontacts.com (which can happen by using OpenID)
- mycontacts.com asks the user whether he wants to authorize example.com to access his contacts
- User makes his choice
- mycontacts.com redirects the user back to example.com
- example.com retrieves the contacts from mycontacts.com
- example.com informs the user that the import was successful

OAuth

Used for delegated **authorization** only -- meaning you are authorizing a third-party service access to use personal data, without giving out a password. Also OAuth "sessions" generally live longer than user sessions. Meaning that OAuth is designed to allow authorization

i.e. Flickr uses OAuth to allow third-party services to post and edit a persons picture on their behalf, without them having to give out their flicker username and password.

OpenID

Used to **authenticate** single sign-on identity. All OpenID is supposed to do is allow an OpenID provider to prove that you say you are. However many sites use identity authentication to provide authorization (however the two can be separated out)

i.e. One shows their passport at the airport to authenticate (or prove) the person's who's name is on the ticket they are using is them.

OpenID Foundation

The OpenID Foundation (OIDF) promotes and enhances the OpenID community and technologies. The OIDF is a non-profit international standards development organization of individual developers, government agencies and companies who wish to promote and protect OpenID. The OpenID Foundation was formed in June 2007

Questions ?

Isuru.wijeratna@ifsworld.com