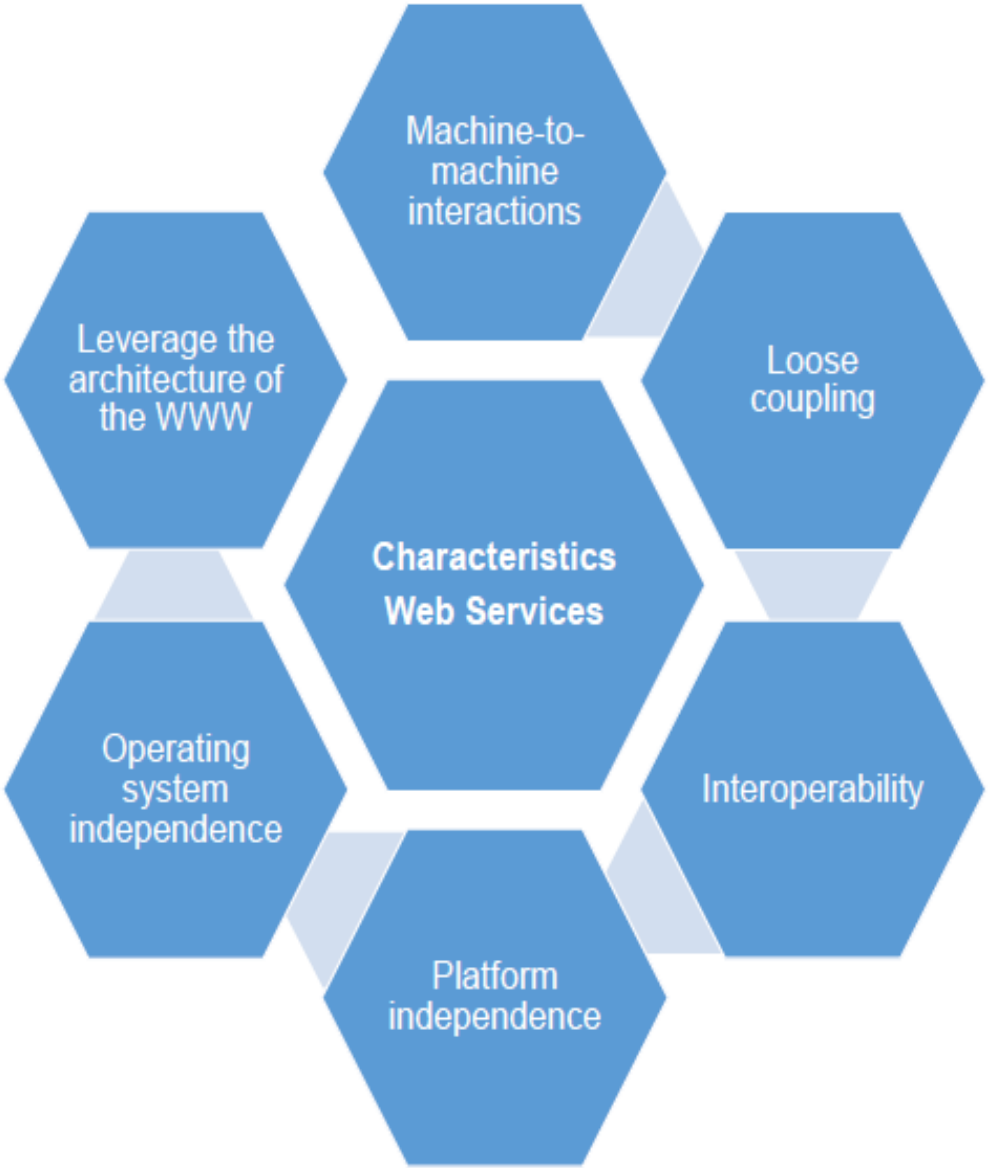
# Web Services SOAP & RESTful

# What is a Web Service?

- A **software system** designed to support interoperable **machine-to-machine** interaction over a **network**.
  - Self-contained, modular, distributed, dynamic
  - Standardized messaging system
  - Can be described, published, located and/or invoked over the network.
  - Language-agnostic
  - Vendor and transport neutral

# Characteristics of Web Services



# Why Web Services?

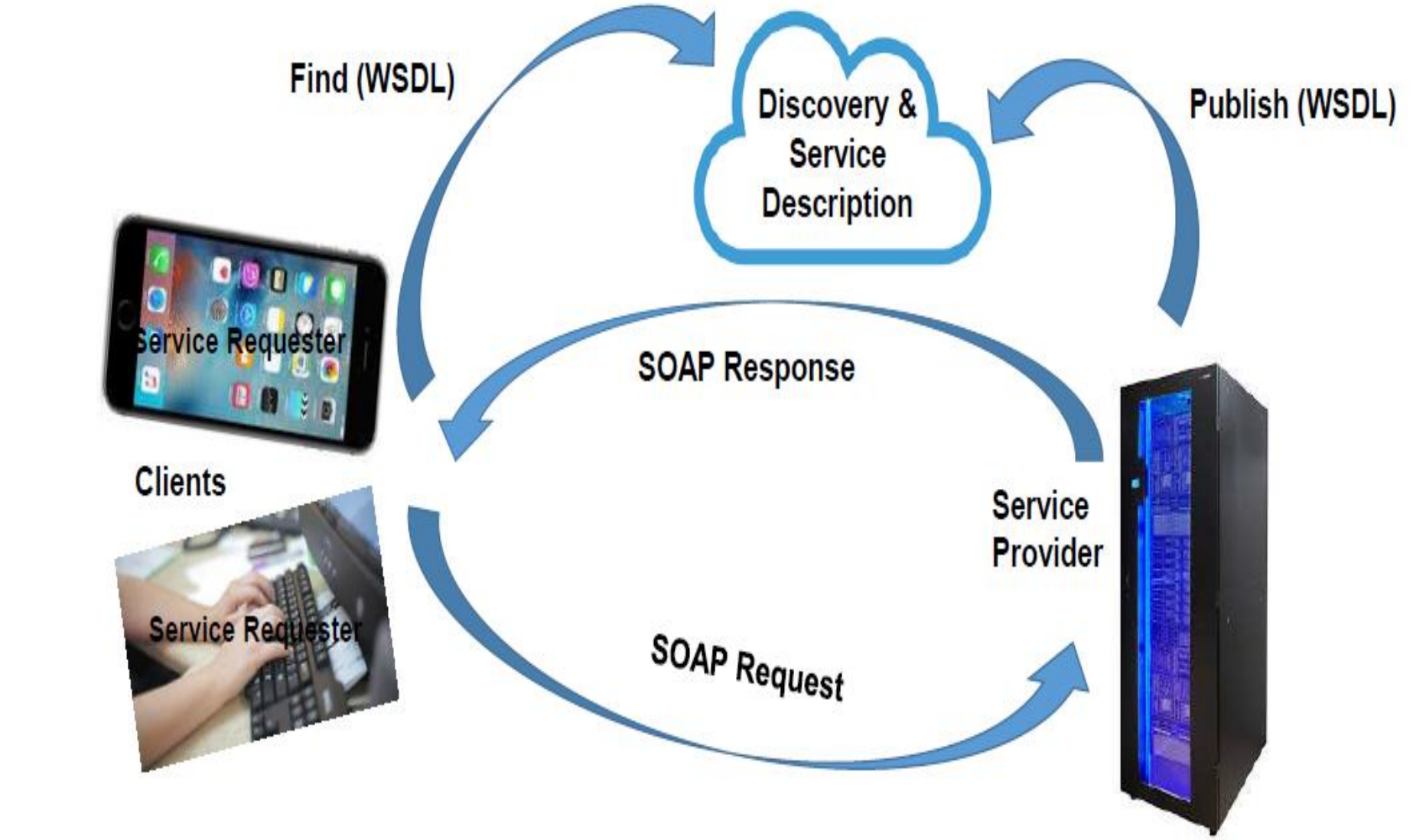
- **Expose the functionality** of existing applications e over the network – without application changes.
- **Loosely Coupled:** Each service exists independently of the other services.
- **Service Reuse:** A function coded once and used over and over
- **Low Cost of Communication:** HTTP over existing internet


# SOAP Web Services

# SOAP

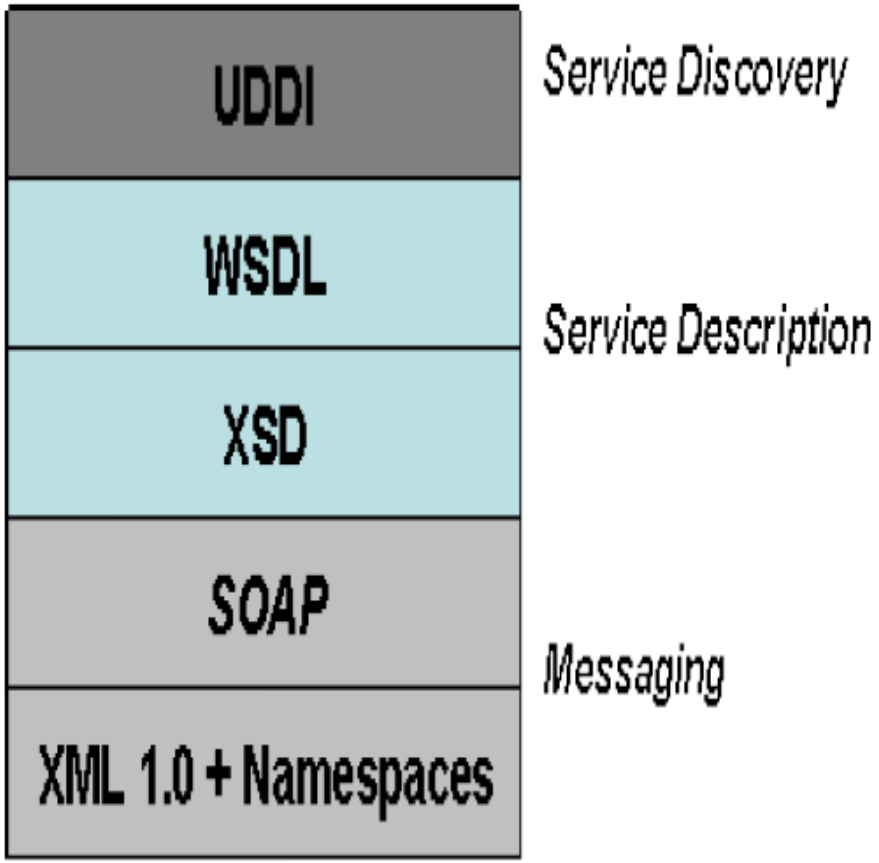
- An industry accepted W3C specification for XML distributed computing infrastructure.
- Acronym for **S**imple **O**bject **A**ccess **P**rotocol (acronym not used now)
- Extends HTTP for XML messaging and provides data transport for Web services
- Enables client applications to connect to remote services and invoke remote methods
- Driven by standard specifications
  - Basic: UDDI, WSDL, SOAP, XML & Namespaces
  - Extended: WS-Security, WS-Policy, WS-I (Interoperability) etc.

# SOAP - Architectural Overview



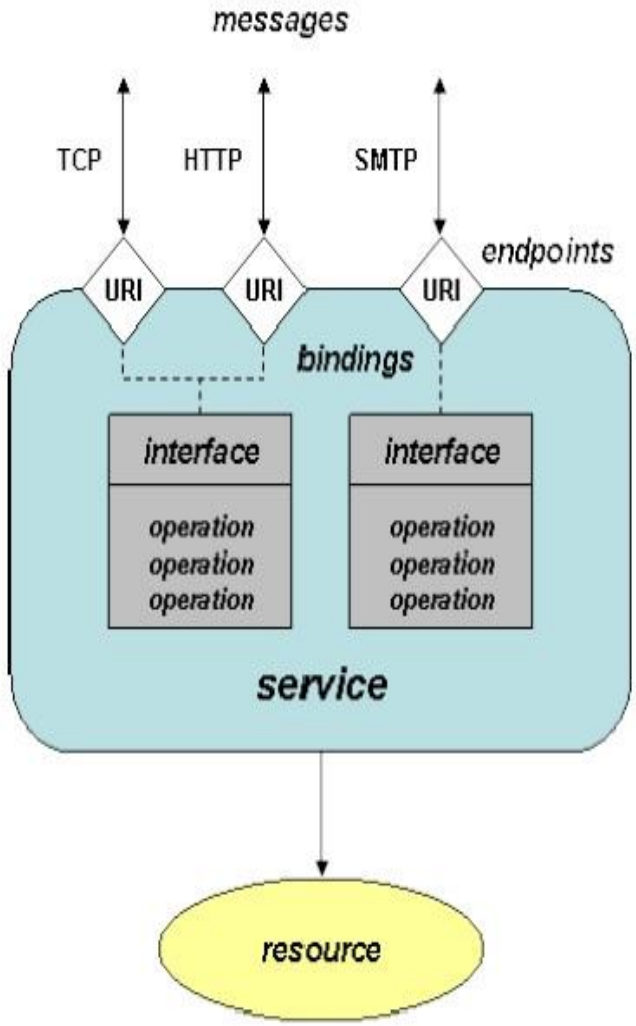
# SOAP: Basic Specifications

- **UDDI** (Universal Description, Discovery, and Integration): Platform-independent way of describing and discovering Web services and Web service providers.
- **WSDL**: Defines services as collections of network endpoints.
- **SOAP**: Simple and lightweight mechanism for exchanging structured and typed information.
  - XML + Namespaces



# Web Services Description Language (WSDL)

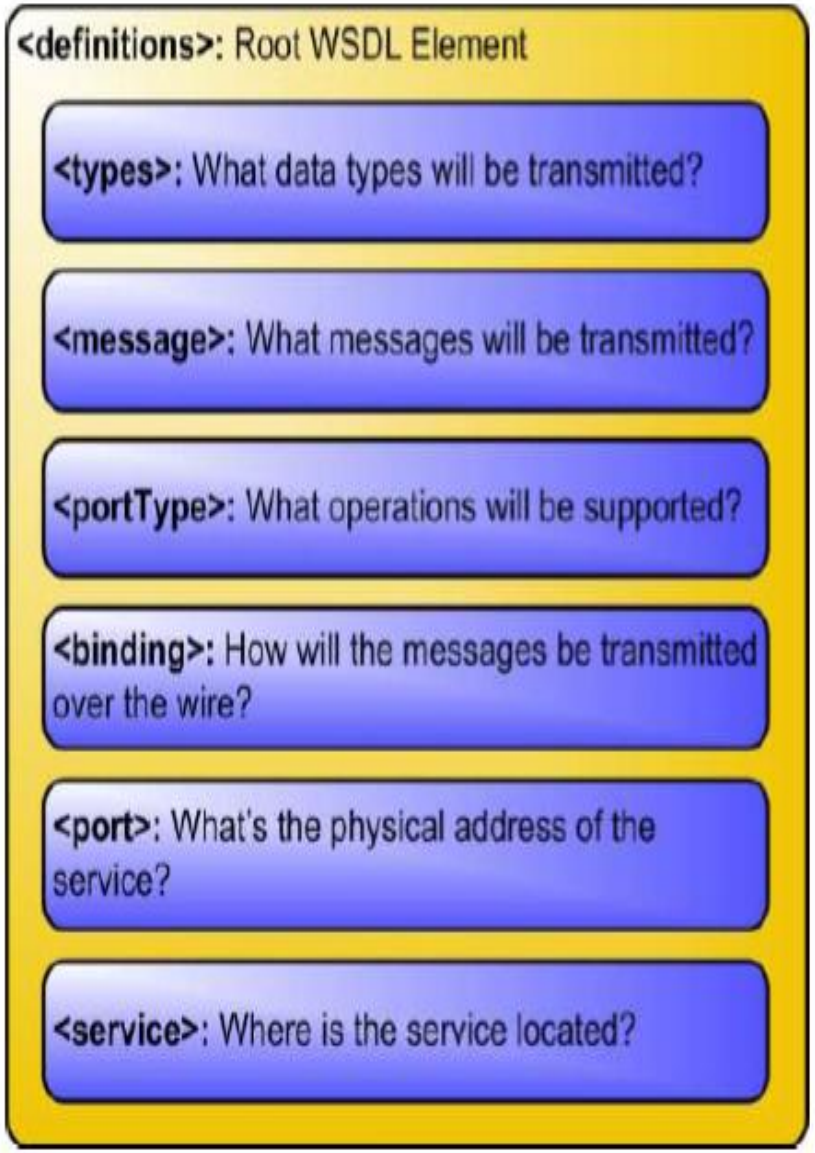
- An XML format for describing information needed to invoke and communicate with a Web Service. It gives the answers to the questions
  - Who?
  - What?
  - Where?
  - Why?
  - How?
- It describes the complete contract for application communication with the Web Service
- A service description has two major components:
  - **Functional Description**: How the Web Service is invoked, where it's invoked. Focuses on the details of the syntax of the message and how to configure the network protocols to deliver the message.
  - **Nonfunctional Description**: Details that are secondary to the message (such as security policy) but instruct the requestor's runtime environment to include additional SOAP headers.



# Understanding WSDL Structure

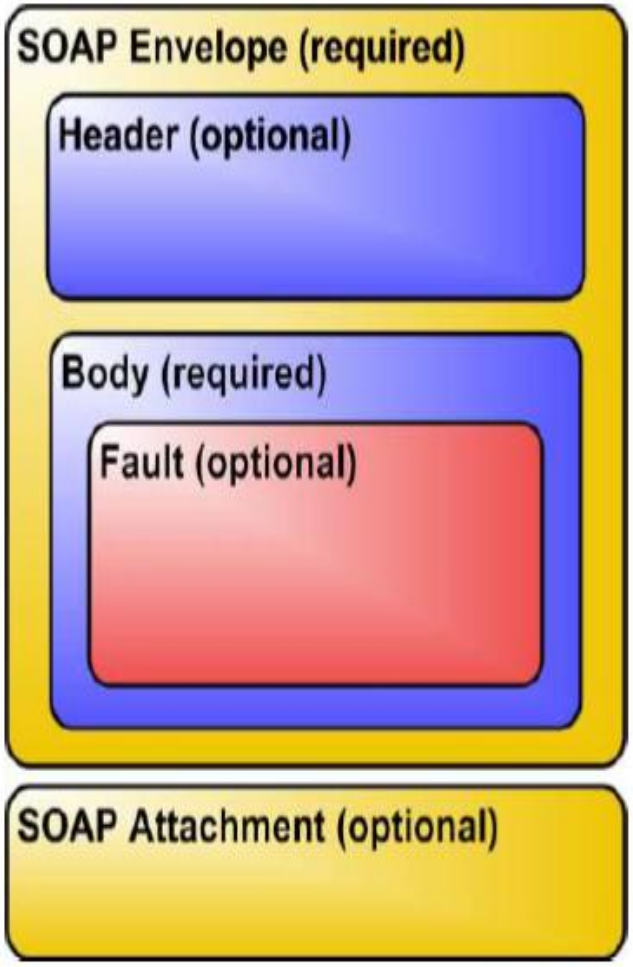
A WSDL Document is a set of definitions with a single root element. Services can be defined using the following XML
elements:

- **Type:** Used to define custom message types (Data Type)
- **Message:** Abstraction of request and response messages that my client and service need to communicate (Methods)
- **PortType:** Contains a set of operations (Interfaces) which organize WSDL messages.
- **Binding**: Binds the portType to a specific protocol (typically SOAP over http) (Encoding Scheme)
- **Port**: provides physical address of the Service.
- **Service:** Gives one or more URLs for the service many URLs



# Understanding SOAP Message

- **XML message format** for exchanging structured, typed data
- Structured of following components:
  - 0 or 1 header elements
  - 1 body element
  - Envelop that wraps it all
- Envelope : Root element of a SOAP message.
- Header
  - Can contain additional payloads of "metadata"
  - Security information, quality of service, etc
- Body:
  - Contains the XML payload
  - Can have optional Fault elements used to describe exceptional situations.
- Attachment: MIME encoded for exchanging binary data.




- Sample SOAP message

# SOAP – Pros & Cons

- SOAP Pros
  - Standard protocol for exchanging information in a decentralized and distributed environment.
  - Platform independent & Vendor neutral.
  - Simple compared to RMI, CORBA, and DCOM etc
  - Decouples the encoding and communications protocol.
  - Anything that can generate XML can communicate through SOAP.
  - Additional Security in addition to HTTP authentication or HTTPS.
  - Supported by most languages and tools.

- SOAP Cons
  - Complex compared to RESTful Services
  - Higher learning curve
  - Being protocol heavy may lead to performance issues

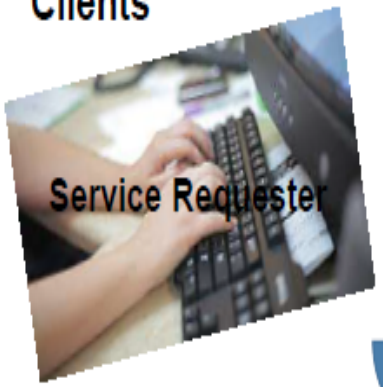
# RESTful Web Services

# RESTful Web Service

- Representational State Transfer
- REST is an architecture all about the Client-Server communication
- Guided by REST constraints (design rules).
- Based on Resource Oriented Architecture
  - A network of web pages where the client progresses through an application by selecting links
  - Requests/responses relate to representations of states of a resource
  - When client traverses link, accesses new resource (i.e., transfers state)
- Uses simple HTTP protocol and service methods:
  - GET: Return data, nothing is changed on server
  - POST: Create, update, or delete data on server
  - PUT: Replace referenced resource(s)
  - DELETE: Delete referenced resource(s)

# RESTful – Architectural Overview


Clients




HTTP Response (XML/JSON)

HTTP Request (XML/JSON)

REST
Web
Server

# RESTful Design Specifications (Constraints)

**Client-Server**

- Separation of concerns - user interface vs data storage
- Client and server are independent from eachother

**Uniform Interface**

- All resources are accessed with a generic interface (HTTP-based) which remains same for all clients.

**Stateless**

- Each request from client to server must contain all of the information
- No client session data or any context stored on the server

**Layered System**

- Allows an architecture to be composed of hierarchical layers
- Each component cannot “see” beyond the immediate layer.

**Cacheable**

- Specify data as cacheable or non cacheable
- HTTP responses must be cacheable by the clients

**Code On-Demand**

- REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

# HTTP Methods for RESTful Services

| HTTP Method | URI | CRUD | Request Stream | Response Stream | Response Code |
|---|---|---|---|---|---|
| **POST** | /customers | Create | Customer without id | customer | 201 / 404 / 409 |
| **GET** | /customers | Read | n/a | Customers collection | 200 / 404 |
| **GET** | /customers/{id} | Read | n/a | Customer | 200 / 404 |
| **PUT** | /customers/{id} | Update | Customer | n/a | 200 / 204 / 404 |
| **DELETE** | /customers/{id} | Delete | n/a | n/a | 200 / 404 |
| **OPTIONS** | /customers/ | Available Methods | n/a | Available Methods | 200 / 204 |

# RESTful Design Considerations

Steps for designing RESTful Web Service

- Identifying resources the service will expose over the network.
- Designing the URI Templates – map URIs to resources
- Applying the Uniform HTTP Interface – options available on each resource for different user groups.
- Security Considerations – Authentication and authorization
- Designing the Resource Representations – XML/JSON.
- Supporting alternate Representations – XML or JSON based on filters
- Providing Resource Metadata – Ability to discover resources and options

# RESTful Design Considerations

RESTful Service Implementation Considerations

- Parse the incoming request to
  - Use URI to identify the resource.
  - Identify URI variables ( and map them to resource variables)
  - HTTP method used in the request (and whether it's allowed for the resource).
  - Read the resource representation
- Authenticate and authorize the user.
- Use all of this information to perform the underlying service logic.
- Generate an appropriate HTTP response, including
  - Proper status code
  - Description
  - Outgoing resource representation in the response entity body

# RESTful – Pros & Cons

- RESTful Pros
  - Simple interface (URI based)
  - Uses HTTP service methods (GET, POST, …)
  - Caching can be leveraged for better performance
  - Small learning curve
  - Simple to test (browser compatible)
  - Less reliance on tools
  - No standard
- RESTful Cons
  - Not yet well integrated into IDE's (but getting better)
  - Security relies on HTTP authentication
  - Less reliance on tools
  - No standard

# Comparing: SOAP vs RESTful

| SOAP | RESTful |
| --- | --- |
| XML based Messaging Protocol | REST is an architectural style |
| Uses WSDL for communication between Consumer and Provider | Uses XMl or JSON to send or receive data |
| SOAP is Service Oriented – Invokes services by calling RPC methods | REST is Resource Oriented - uses (generally) URI and methods like (GET, PUT, POST, DELETE) to expose resources |
| SOAP supports for stateful implementation | REST follows stateless model |
| Transfer is over HTTP as well as other protocols such as SMTP, FTP, etc | REST is over only HTTP |
| SOAP is Distributed Computing style implementation | REST is Web Style (Client Server) Implementation |
| SOAP can be called from JavaScript but difficult to implement. | Easy to call from JavaScript. |