RESTful Web Services

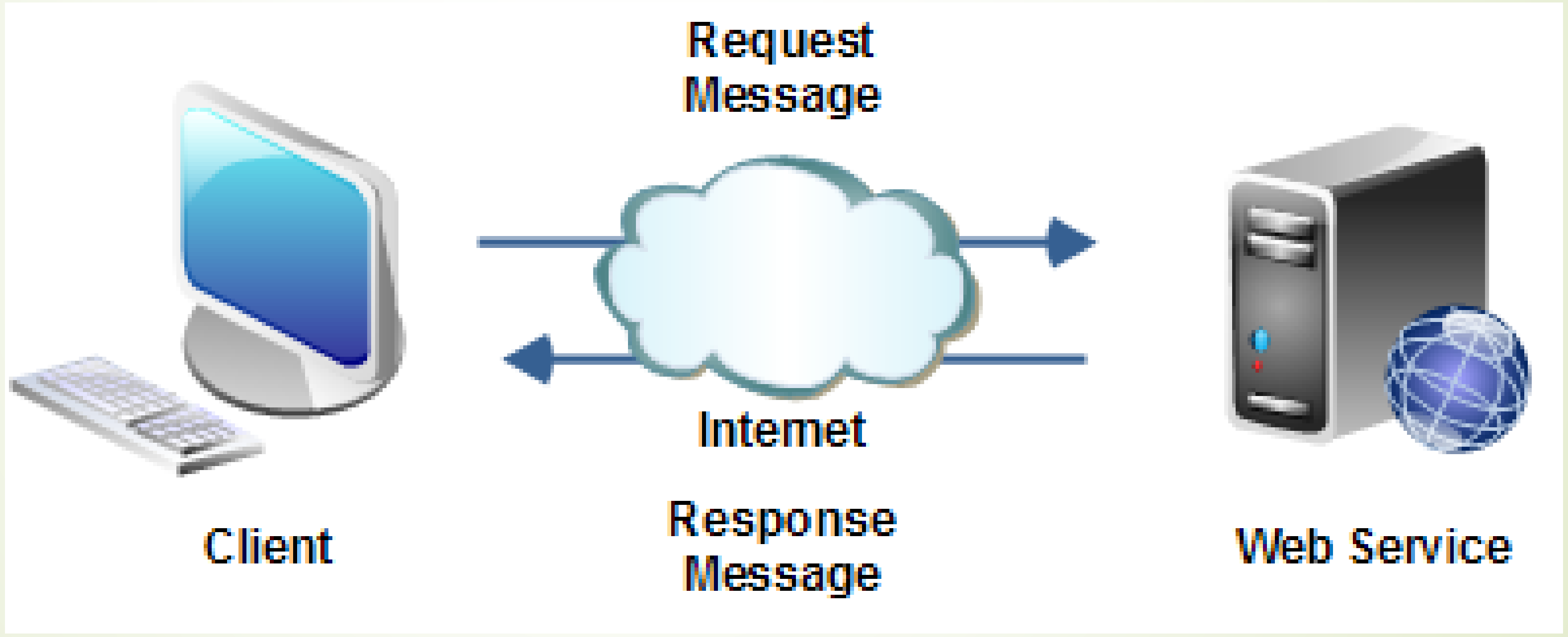
Concepts and Practice

Agenda

- Web Services
- REST Concepts
- Designing RESTful Web Services
- Developing RESTful Web Services using JAX-RS
  - Server Side
  - Client Side
- Wrap-up

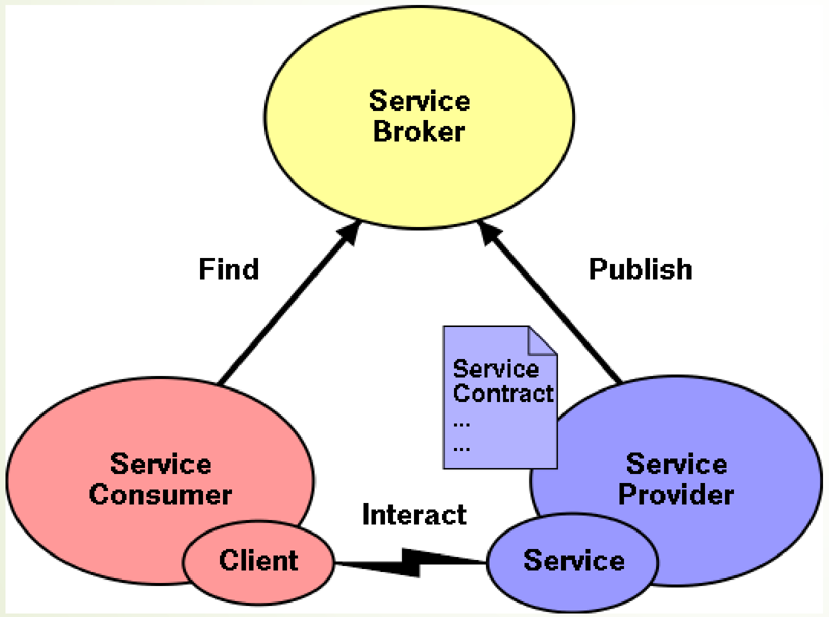
Web Services

- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. (W3C)

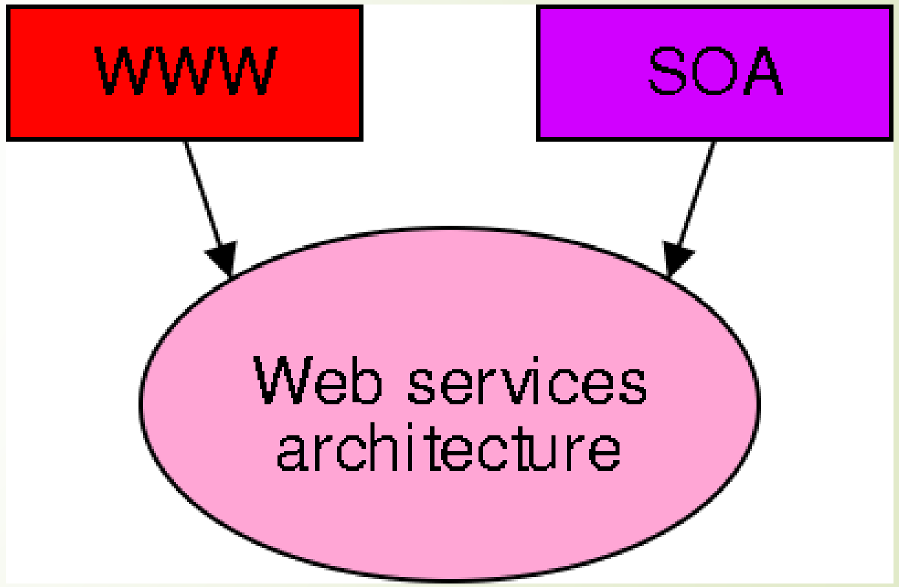


SOA and Web Services Architecture

Service Oriented Architecture

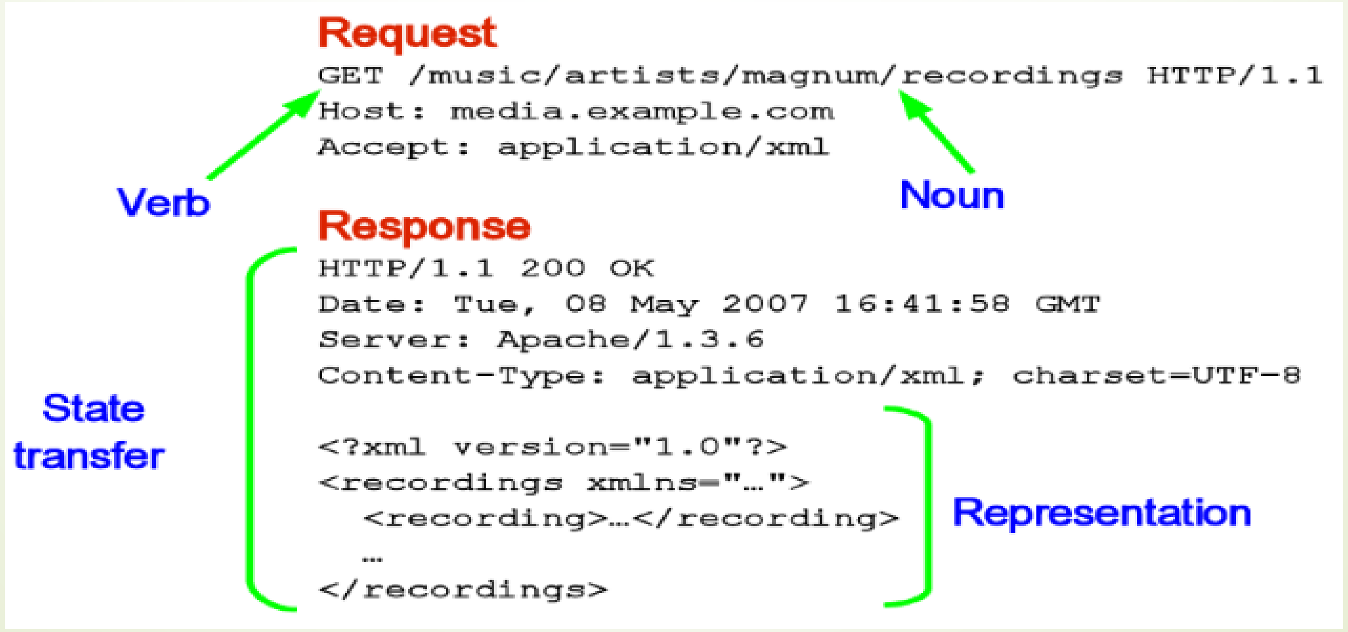


Web Services Architecture



REST Concepts

- REST stands for **RE**presentational **S**tate **T**ransfer
- REST by itself is not an architecture
- REST can be realized as a software architecture when set of guidelines and constraints applied – Thanks to Roy Fielding.



```
GET /music/artists/magnum/recordings HTTP/1.1
Host: media.example.com
Accept: application/xml
```

```
HTTP/1.1 200 OK
Date: Tue, 08 May 2007 16:41:58 GMT
Server: Apache/1.3.6
Content-Type: application/xml; charset=UTF-8

<?xml version="1.0"?>
<recordings xmlns="…">
  <recording>…</recording>
  …
</recordings>
```

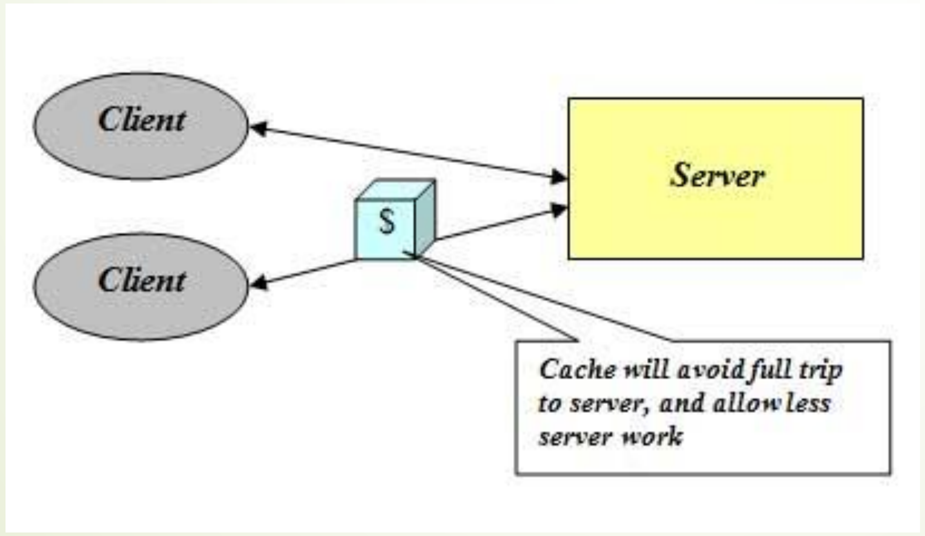
Roy Fielding’s Architectural Constraints

0) The Null style – No constraints at all

1) Client-Server – separation of concern

2) Stateless – No client information in the server side

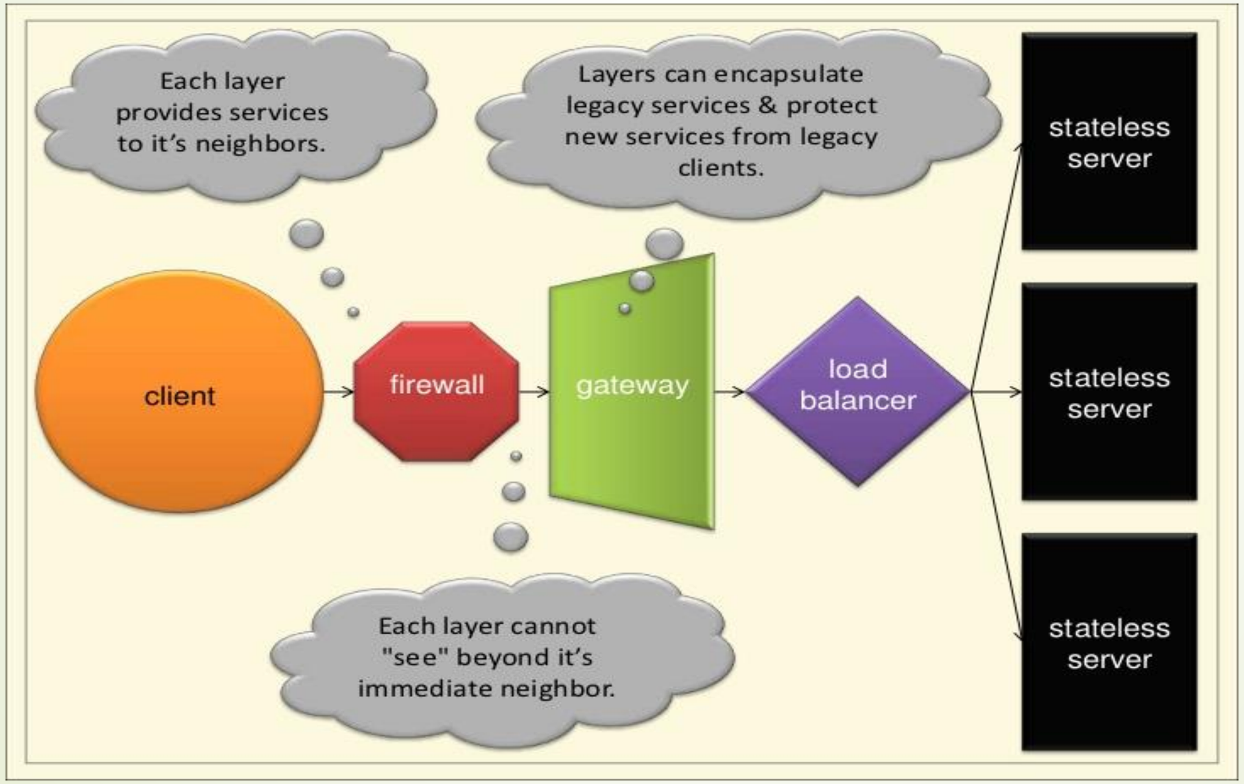
3) Cache – Clients can cache the responses


Client
Client
S
Server
Cache will avoid full trip to server, and allow less server work

4) Uniform Interface – a unique address and valid point of access to resources

5) Layered System – to support scalability

6) Code-On-Demand (Optional) – allowing the download of code on demand


Each layer provides services to it's neighbors.
Layers can encapsulate legacy services & protect new services from legacy clients.
stateless server
client
firewall
gateway
load balancer
stateless server
stateless server
Each layer cannot "see" beyond it's immediate neighbor.

REST Core Principle – Uniform Interface

- **Identification of Resources – URI**

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

  - **scheme** is the protocol (either http or https)
  - **host** is DNS name or IP address; followed by optional **port**, which is numeric
    - **host** and **port** represent the location of your resource on the network
  - **path** expression delimited by the "/" – analogous directory list of a file
  - **query string** is a list of parameters - name/value pairs.
    - **?** Separates path from query string, each name./value pair is delimited by **&**
  - **fragment**, delimited by # used to point to a certain place in the document

- **Representation – Data & Meta-data**
  - **Data**: Message body (**Payload**) of the request/response.

```
[data, data, data]
```

  - **Meta-data:** Name-value pairs (**Header** fields) that describe the representation
    - **Content-Type**: Multipurpose Internet Mail Extension (MIME)

```
Content-Type: type/subtype
```

type is the main format family and **subtype** is a category

    - **Content-Length**: Anticipated size of the message body

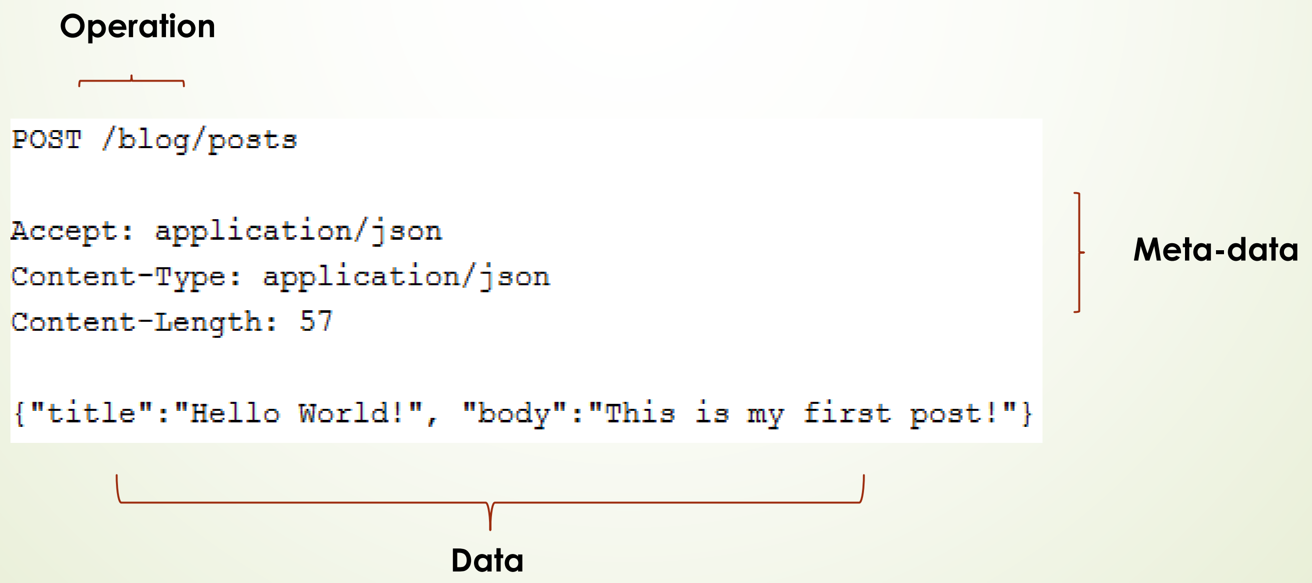
```
Content-Length: <numeric value>
```

    - **Authorization:** Contains credential information

```
Authorization: Basic base64_encode(username:password)

Authorization: AWS AWSAccessKeyId:base64_encode(signature)
```

- **Self-descriptive Messages**
  - **Data & Meta-data:** Containing all the necessary information to complete the task
  - **URI and HTTP Methods:** Definitive set of operations
    - **GET** – Retrieve (analogous to SQL SELECT)
    - **POST** – Create (analogous to SQL INSERT)
    - **PUT** – Update (analogous to SQL UPDATE)
    - **DELETE** – Delete/Cancel (analogous to SQL DELETE)



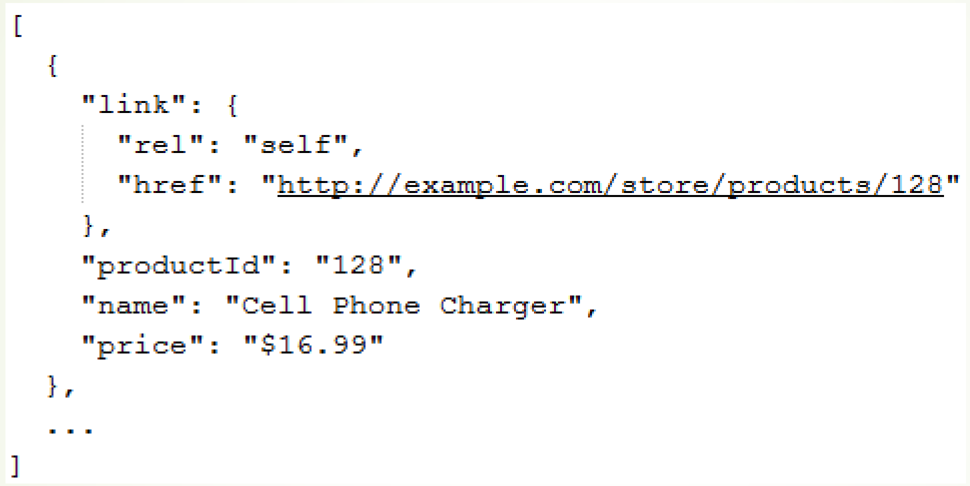
```
POST /blog/posts

Accept: application/json
Content-Type: application/json
Content-Length: 57

{"title":"Hello World!", "body":"This is my first post!"}
```

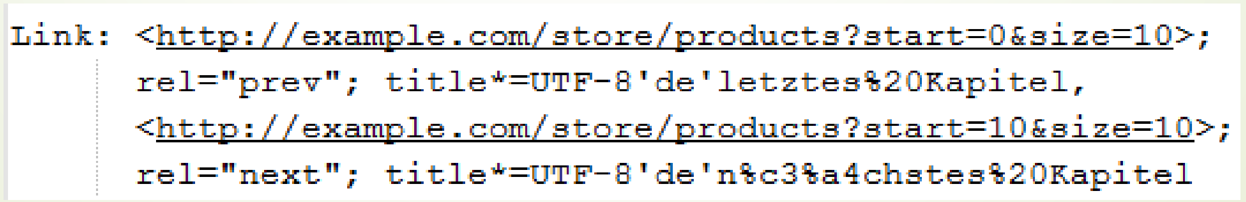
Hypermedia As The Engine Of Application State - HATEOAS

- Resources discoverability (resource itself or related resources) through **hyperlinks**
- Links can be contained in the *Payload (data)*.



```
[
  {
    "link": {
      "rel": "self",
      "href": "http://example.com/store/products/128"
    },
    "productId": "128",
    "name": "Cell Phone Charger",
    "price": "$16.99"
  },
  ...
]
```

- Links can also be contained in *header* (Meta-data).

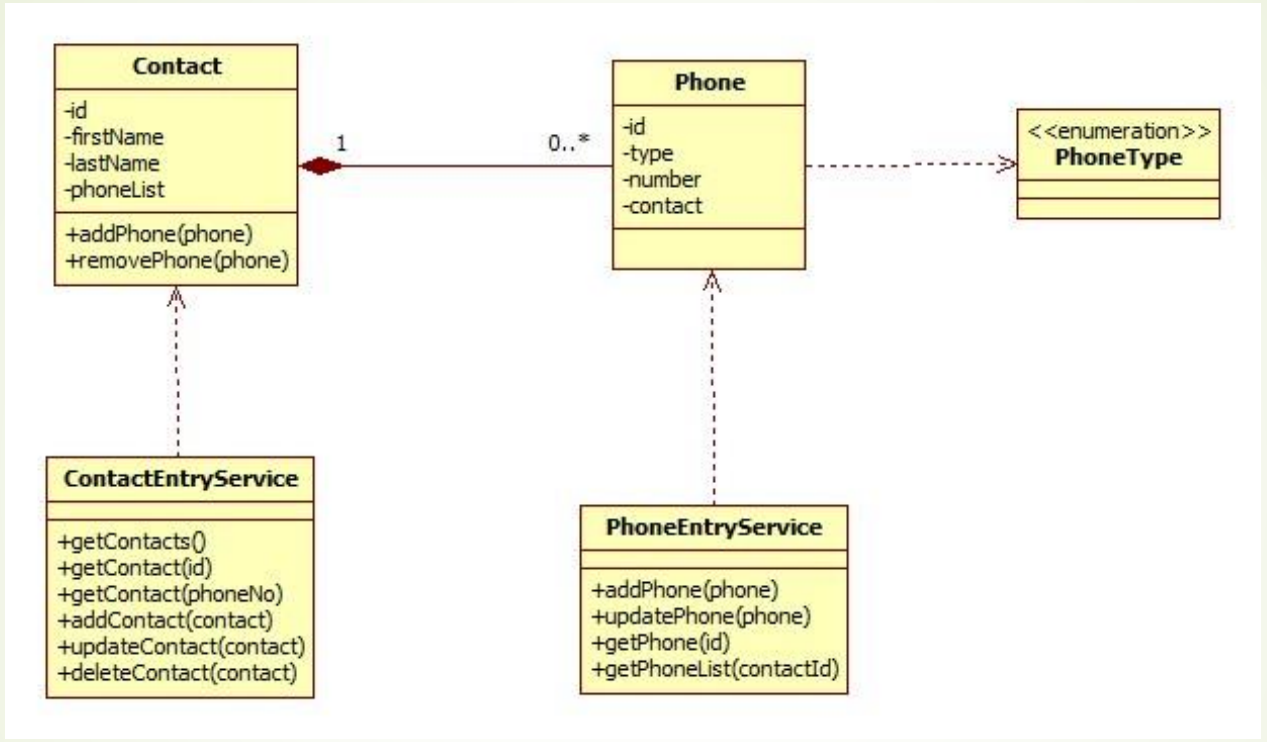


```
Link: <http://example.com/store/products?start=0&size=10>;
      rel="prev"; title*=UTF-8'de'letztes%20Kapitel,
      <http://example.com/store/products?start=10&size=10>;
      rel="next"; title*=UTF-8'de'n%c3%a4chstes%20Kapitel
```

Designing REST Web Services

1) Examine underlying Object Model
2) Identify Resources
3) Model the URIs and define endpoints
4) Define message (data) format
5) Assign HTTP method to each endpoint

1. **Examine underlying Object Model**



2. **Identify Resources**
   - Contact
   - Contact List
   - Phone
   - Phone List

3. Model the URIs and define Endpoints

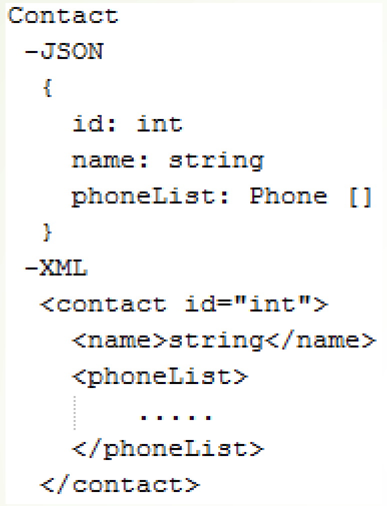
- /contacts
- /contacts/{id}
- /contacts?phoneNo={phoneNo}
- /phones
- /phones/{id}
- /contacts/id/phones

Note:

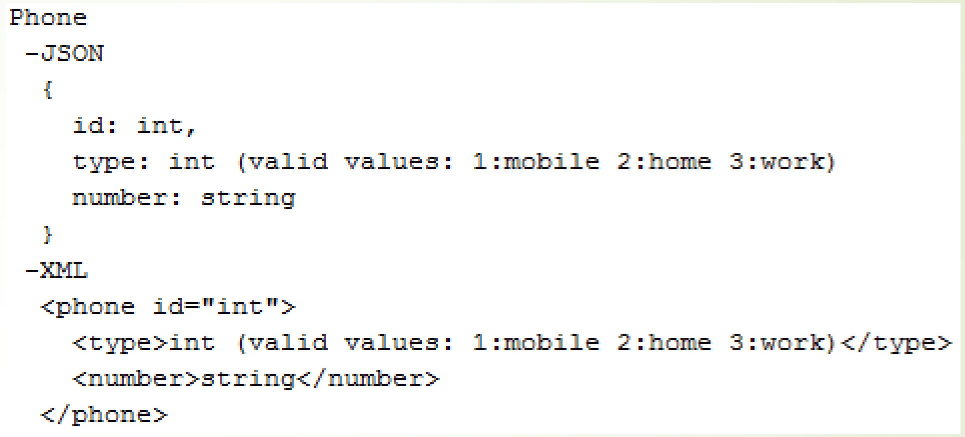
- The nouns in object model have been represented as URIs.
- URI itself doesn't identify operations.
- A combination of HTTP methods and the data format should be used to model operations

4. Define the message (data) format: JSON/XML

- **Read Format:**

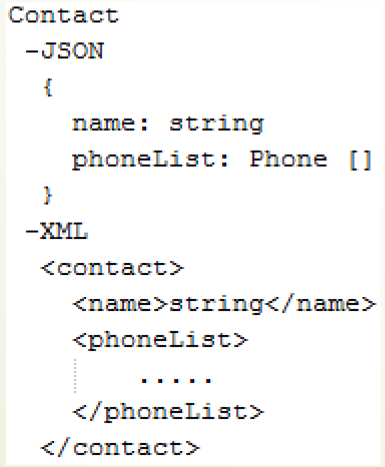


```
Contact
 -JSON
  {
    id: int
    name: string
    phoneList: Phone []
  }
 -XML
  <contact id="int">
    <name>string</name>
    <phoneList>
        .....
    </phoneList>
  </contact>
```

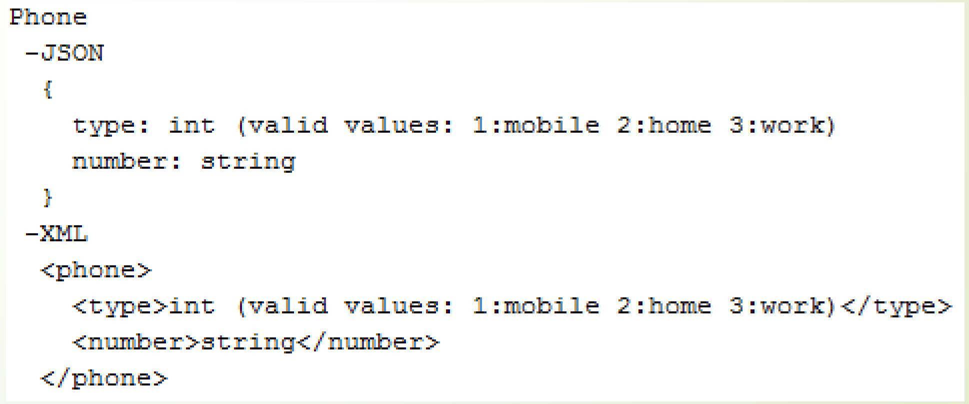


```
Phone
 -JSON
  {
    id: int,
    type: int (valid values: 1:mobile 2:home 3:work)
    number: string
  }
 -XML
  <phone id="int">
    <type>int (valid values: 1:mobile 2:home 3:work)</type>
    <number>string</number>
  </phone>
```

- **Create Format:**

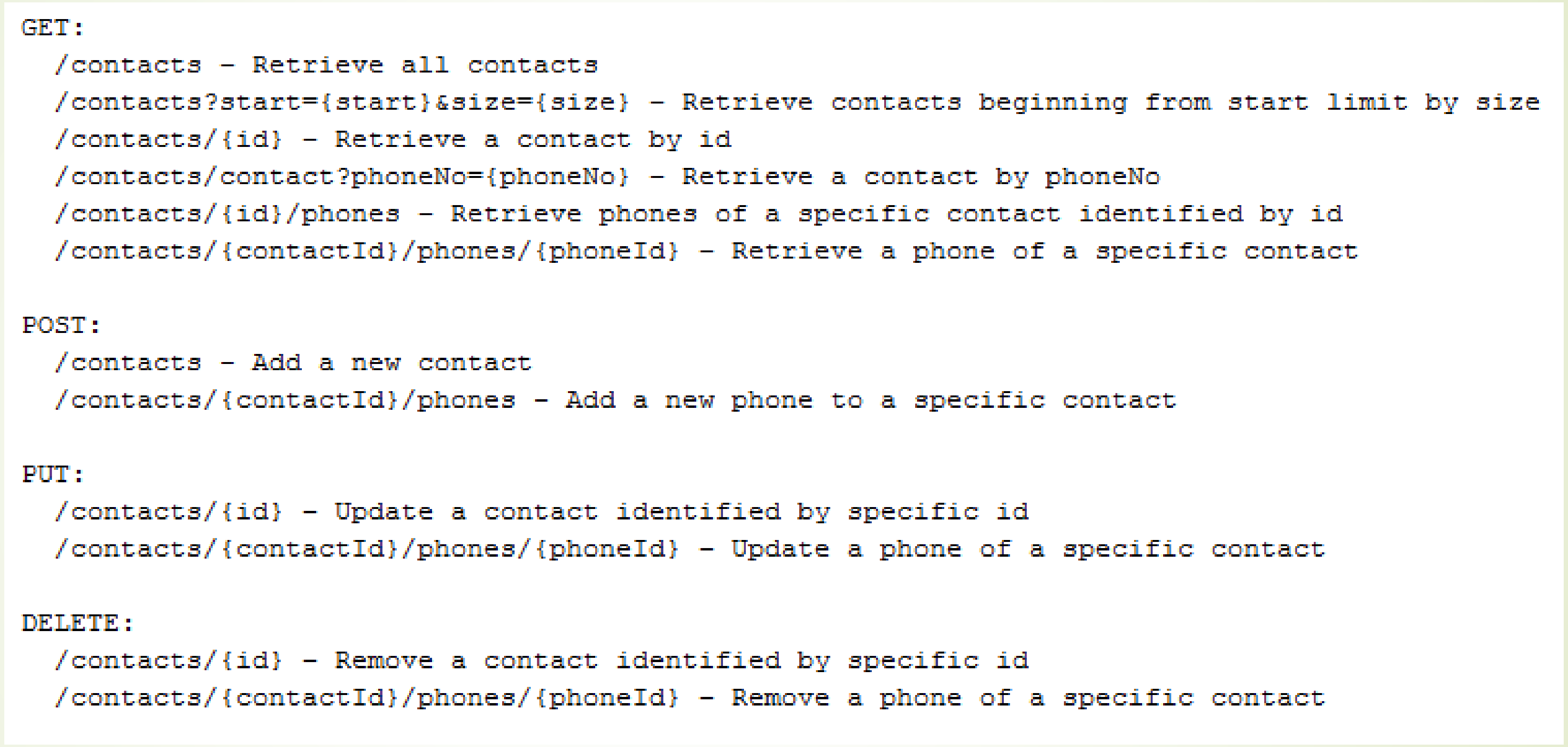


```
Contact
 -JSON
  {
    name: string
    phoneList: Phone []
  }
 -XML
  <contact>
    <name>string</name>
    <phoneList>
        .....
    </phoneList>
  </contact>
```



```
Phone
 -JSON
  {
    type: int (valid values: 1:mobile 2:home 3:work)
    number: string
  }
 -XML
  <phone>
    <type>int (valid values: 1:mobile 2:home 3:work)</type>
    <number>string</number>
  </phone>
```

5. Assign HTTP methods



```
GET:
  /contacts – Retrieve all contacts
  /contacts?start={start}&size={size} – Retrieve contacts beginning from start limit by size
  /contacts/{id} – Retrieve a contact by id
  /contacts/contact?phoneNo={phoneNo} – Retrieve a contact by phoneNo
  /contacts/{id}/phones – Retrieve phones of a specific contact identified by id
  /contacts/{contactId}/phones/{phoneId} – Retrieve a phone of a specific contact

POST:
  /contacts – Add a new contact
  /contacts/{contactId}/phones – Add a new phone to a specific contact

PUT:
  /contacts/{id} – Update a contact identified by specific id
  /contacts/{contactId}/phones/{phoneId} – Update a phone of a specific contact

DELETE:
  /contacts/{id} – Remove a contact identified by specific id
  /contacts/{contactId}/phones/{phoneId} – Remove a phone of a specific contact
```

Server Side Implementation

- **JAX-RS**
  - JSR 311 Specification - Java API for RESTful Web Services
  - Providers: Jersey, RestEasy, RestLet, Apache CXF
- **Spring MVC**
  - Frontend Web Application framework
  - Provides comprehensive support for RESTful Web Services

Client Side Implementation

- **JAX-RS Client API**
  - High-level Client API for accessing any REST resources
  - Supports pluggability of other HTTP Clients such Apache HTTP Client
- **Spring RestTemplate**
  - Spring’s central class for HTTP client side implementation
  - Pluggability of other third-party HTTP clients

References

- Burke, B. (2014). *RESTful Java with JAX-RS 2.0*. Sebastopol, CA: O'Reilly.
- Cisneros, S. (1991). *The house on Mango Street*. New York: Vintage Books.
- Fielding, R. (2000). *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. [online] Roy.gbiv.com. Available at: http://roy.gbiv.com/pubs/dissertation/rest_arch_style.htm [Accessed 12 Feb. 2017].
- Tools.ietf.org. (2014). *RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. [online] Available at: https://tools.ietf.org/html/rfc7231 [Accessed 12 Feb. 2017].