UNSW SYDNEY | Australia's Global University

COMP9321

Data Services Engineering

Term 1, 2022

Week 4 REST Services Part 1

Jeff Bezos' (Key to Success) Mandate

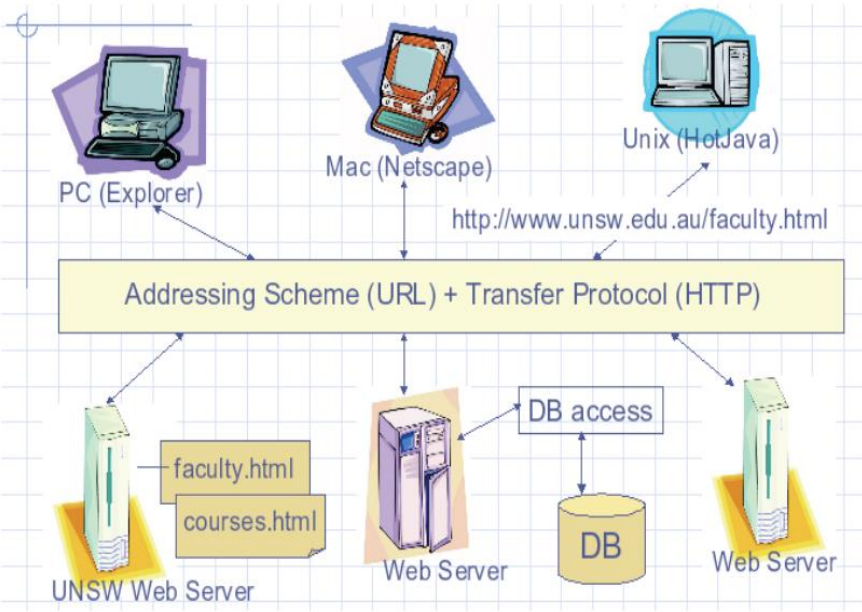
- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols—doesn't matter.
- All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- Anyone who doesn't do this **will be fired.**

Web Essentials

Web = Higher Level Protocol over the Internet

Three basic components of the Web:

- A Uniform Notation Scheme for addressing resources (Uniform Resource Locator - URL)
- A protocol for transporting messages (HyperText Transport Protocol - HTTP)
- A markup language for formatting hypertext documents (HyperText Markup Language – HTML)

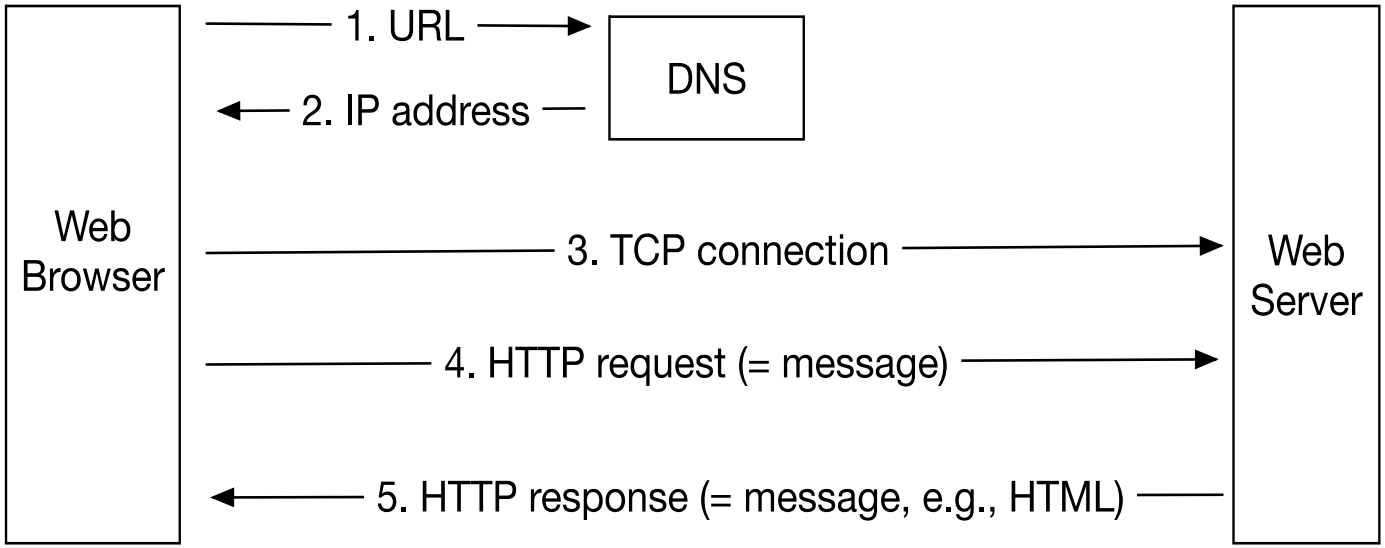


In the end, these building blocks become the essentials of the Web applications as well.

Web Essentials

The building blocks of the Web

A Typical “Web” Interaction (all components together)



The basic “Web” communication model is HTTP request-response (blocked!)

An URL (HTTP scheme)

`http://www.example.org:56789/a/b/c.txt?t=win&s=chess#para5`

authority part:

- after 'http://' through to the next slash - www.example.org:56789
- It consists of either a domain name or an IP address
- optionally followed by a port number (if omitted, port 80 is implied)

path part:

- after the authority through to question mark (?)
- /a/b/c.txt (/ is part of the path, ? is not)
- much like a file path in file system ...

query string part:

- after the path up to a number sign #
- contains a set of name-value pairs, separated by & (e.g., t=win&s=chess)

fragment identifier part:

- after the number sign #, not including #

Web Essentials - HTTP

HTTP Request (from browser to server):

It is composed of `Request Line + Header + (additional data)`

Syntax for the Request Line:

`Request-Method sp Request-URI sp HTTP-version CRLF`

eg, GET `http://www.smh.com.au/index.html` HTTP/1.1

- There must be a newline (CRLF) between the header and the additional data part.
- Common Request methods: GET, POST, HEAD ...
- Request header: `User-Agent`, `Referer`, `Authorization`.
- Additional data (body): parameters (POST), block of data

You can utilise many parts of these HTTP request data to be more effective

HTTP Request Methods (version 1.1)

Method	Description
GET	It the simplest, most used. It simply retrieves the data identified by the URL. If the URL refers to a script (CGI, servlet, and so on), it returns the data produced by the script.
HEAD	It only returns HTTP headers without the document body.
POST	It is like GET. Typically, POST is used in HTML forms. POST is used to transfer a block of data to the server.
OPTIONS	It is used to query a server about the capabilities it provides. Queries can be general or specific to a particular resource.
PUT	It stores the body at the location specified by the URI. It is similar to the PUT function in FTP.
DELETE	It is used to delete a document from the server. The document to be deleted is indicated in the URI section of the request.
TRACE	It is used to tract the path of a request through firewall and multiple proxy servers. TRACE is useful for debugging complex network problems and is similar to the traceroute tool.

Web Essentials - HTTP

HTTP Response (from server to browser):

- Composed of `Status Line + Header + Body`
- Status line: 200 OK, 404 Not Found, etc.
- Header:
  - Content-Type, Content-Language, Content-Length, Cache-control, etc.
- Body:
  - Body contains the requested data
  - Body is in specific MIME format (eg., text/HTML)
  - MIME (Multipurpose Internet Mail Extension): text (plain, HTML), multimedia data, applications such as PDF, PowerPoint, etc.

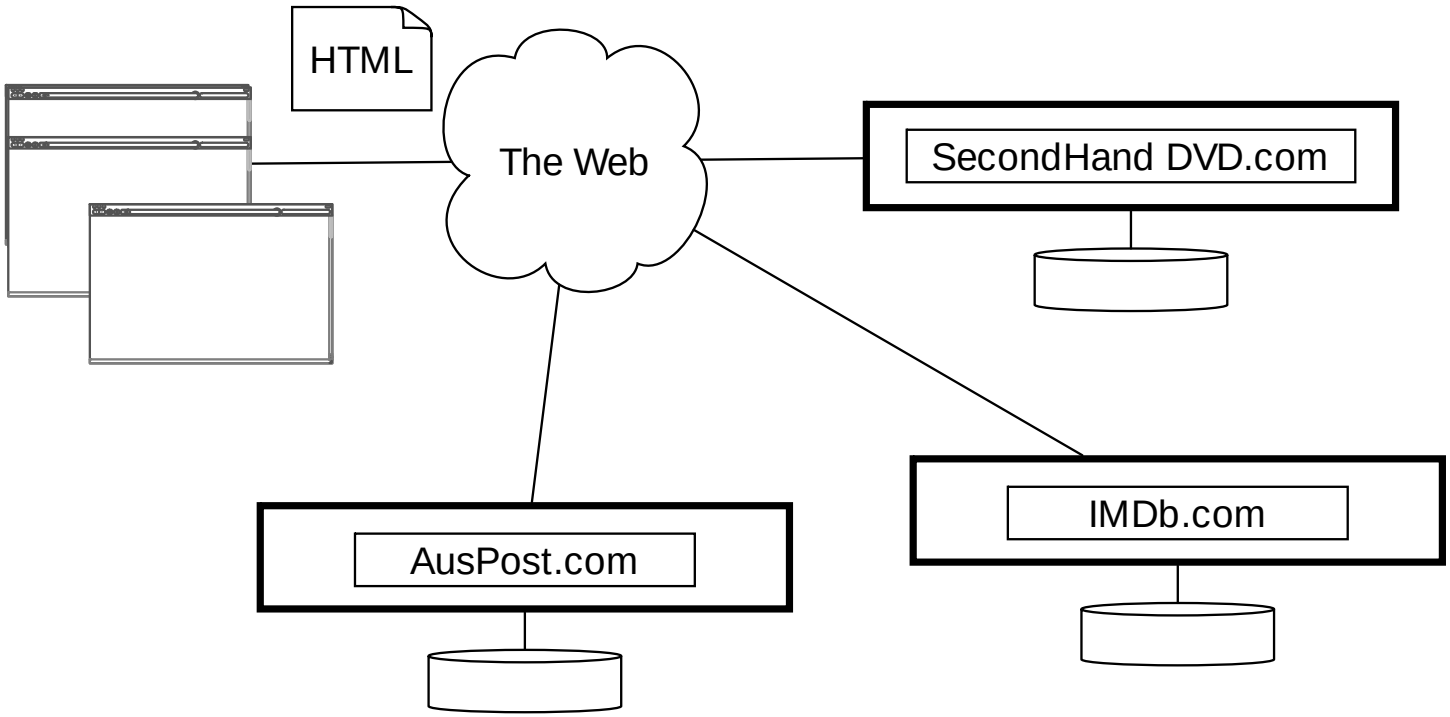
HTTP Status Code: classes/common codes

Digit	Class	Standard Use
1	Informational	Provides information to client before request processing has been completed
2	Success	Request has been successfully processed
3	Redirection	Client needs to use a different resource to fulfill request
4	Client Error	Client's request is not valid
5	Server Error	An error occurred during server processing

Code	Reason Phrase	Usual Meaning
200	OK	Request processed normally
301	Moved Permanently	URI for the requested resource has changed
401	Unauthorised	The resource is password protected, and the user has not yet supplied a valid password
403	Forbidden	The resource is present on the server, but is read protected
404	Not Found	No resource corresponding the URI was found
500	Internal Server Error	Server software detected an internal failure

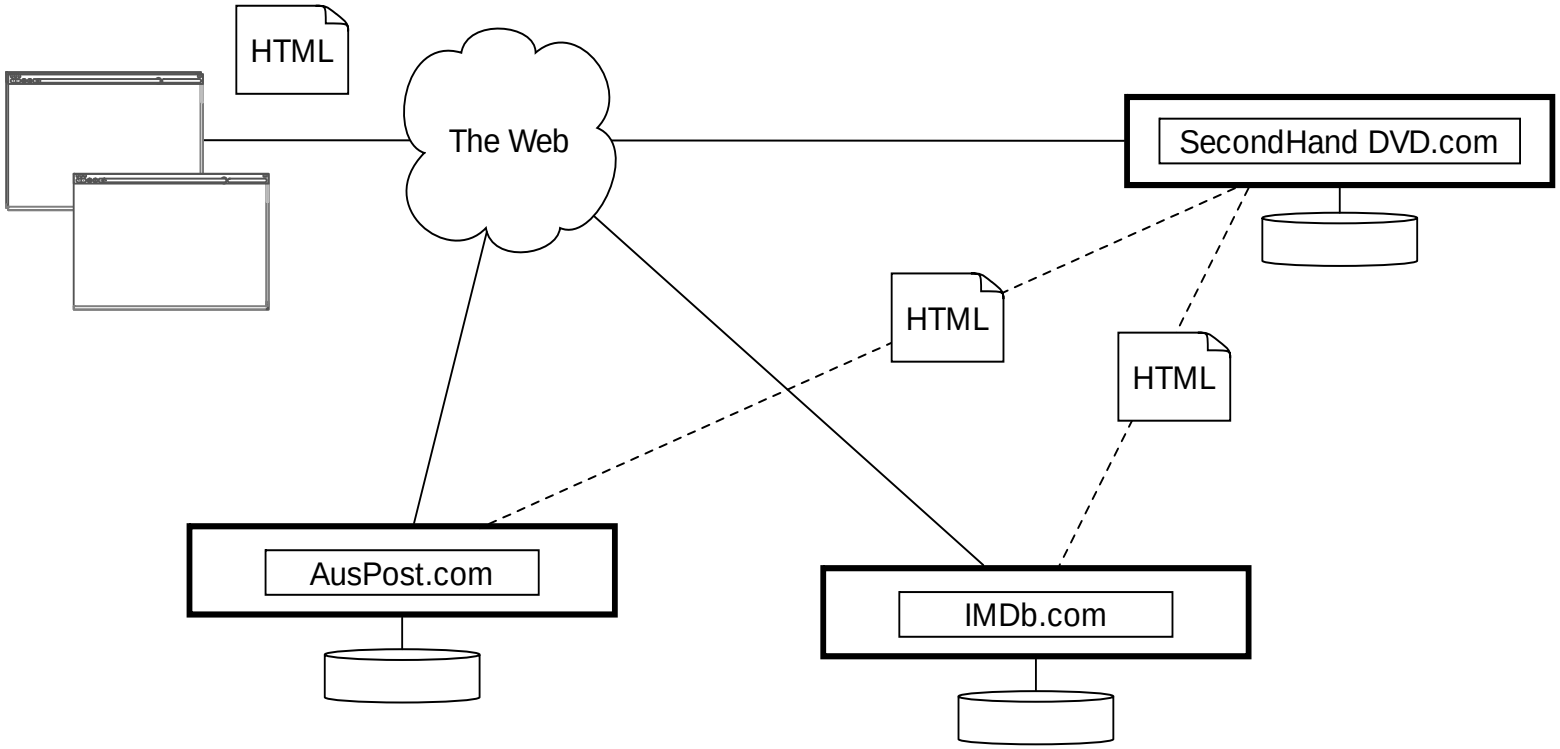
Back to the story of integrating applications

Over the last 20 years, we have built a lot of Web sites!



Story of Web APIs

Web sites in early 2000 were pretty much 1-Tier system from the application integration standpoint

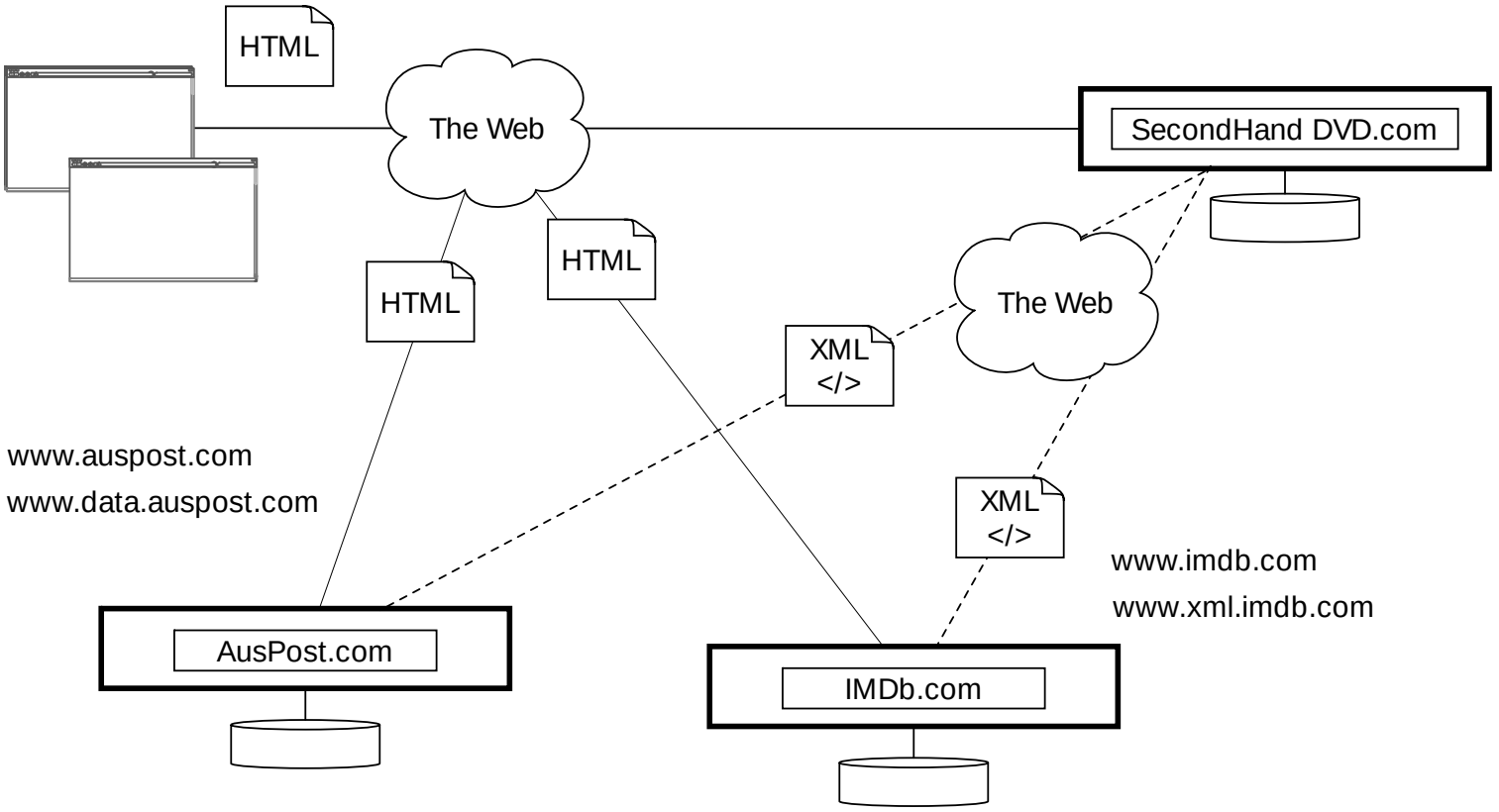


Faking browser clicks (requests) – HTTP/HTML interactions

Brittle … the sources can change without you.

Story of Web APIs

A better version of the idea …



Development of an idea that software-to-software interactions are to be treated differently.

Web Services

A WebService is a Consumer_Machine-to-Provider_Machine collaboration schema that operates over a computer network.

- The data exchanges occur independently of the OS, browser, platform, and programming languages used by the provider and the consumers.
- A provider may expose multiple EndPoints (sets of WebServices), each offering any number of typically related functions.
- WebServices expect the computer network to support standard Web protocols such as XML, HTTP, HTTPS, FTP, and SMTP.
- Example: Weather information, money exchange rates, world news, stock market quotation are examples of applications that can be modelled around the notion of a remote data-services provider surrounded by countless consumers tapping on the server's resources.

Why Web Services

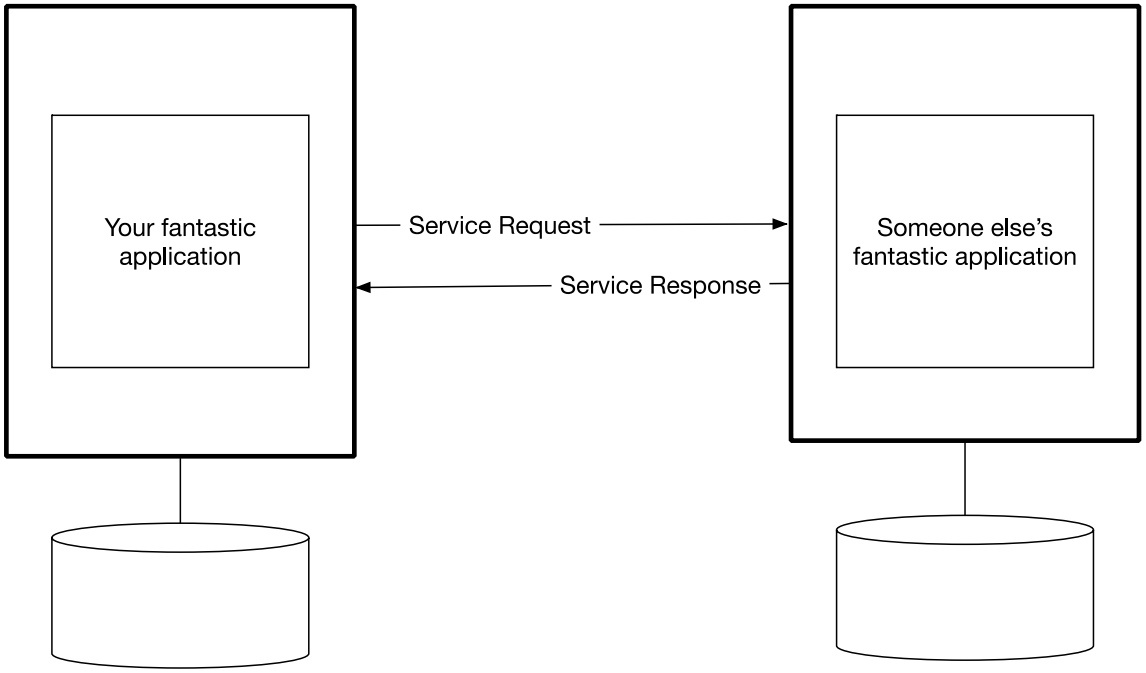
- There are many cases in which the data needed to work with is very extensive, or changes very often and cannot (should not) be hardcoded into the app. Instead, this kind of data should be requested from a reliable external source (for instance, what is the Euro-to-Dollar rate of change right now?)
- Another class of apps requires a very high computing power perhaps not available in the mobile device (think of problems such as finding the shortest/fastest route between two mapped locations, or best air-fare & route selection for a traveller)

Why Web Services

Under the Web Service strategy the invoked functions are implemented once (in the server) and called many times (by the remote users).

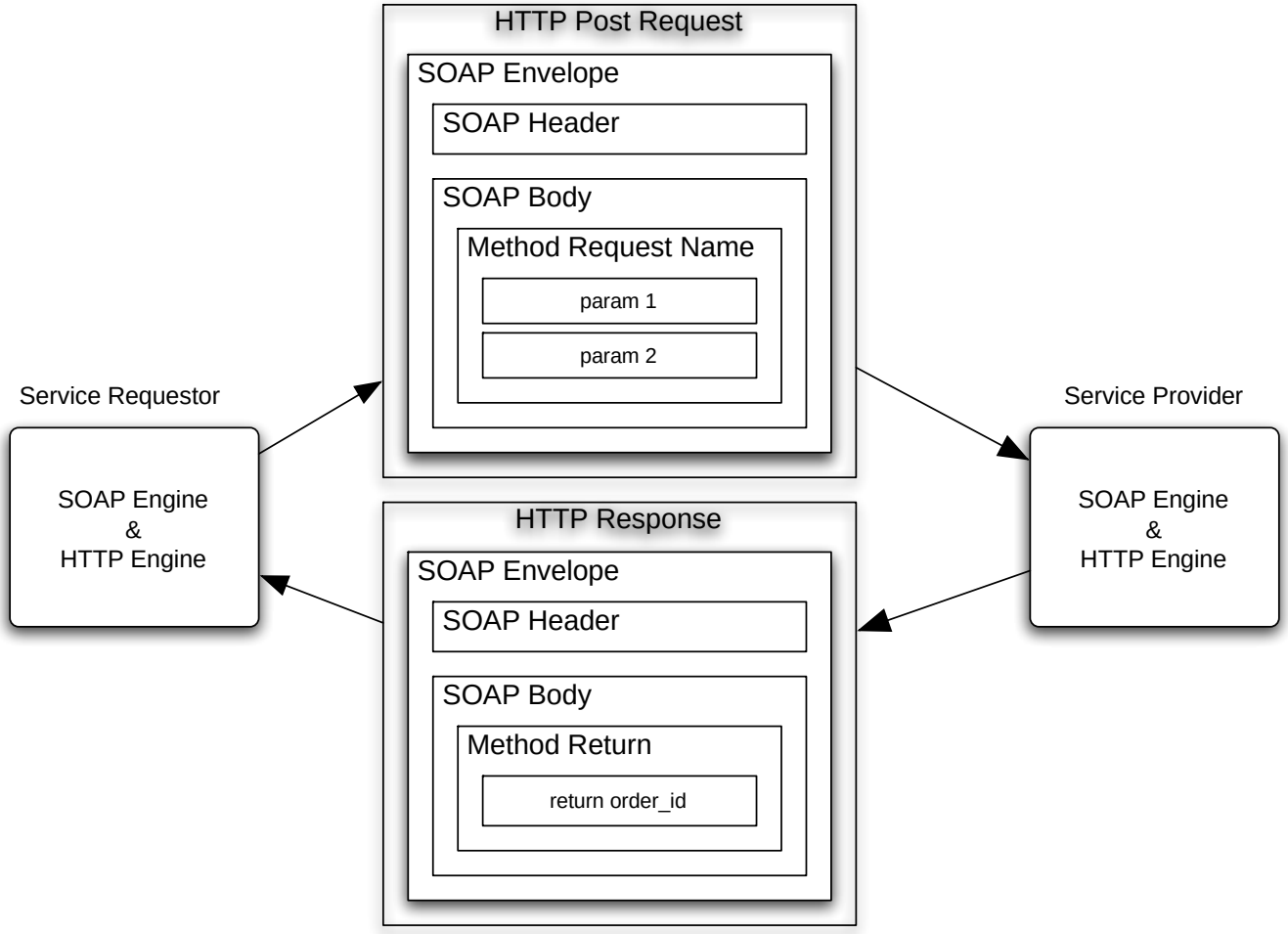
- Some advantages of this organization are:
  - Elimination of redundant code,
  - Ability to transparently make changes on the server to update a particular service function without clients having to be informed.
  - Reduced maintenance and production costs.

API – Application Programming Interface



- The interface is not meant for human interactions – there is another program on the other side → implication of this: you must have a clear contract (e.g., IOU Alice Bob 100)
- These days companies use APIs internally (private APIs) as well as exposing them externally (public APIs)

XML-based APIs …



Early versions of API utilised XML documents → SOAP protocol (W3C standards), or XmlRPC

XML-based APIs …

Couple of SOAP examples:
http://www.dneonline.com/calculator.asmx
http://www.webservicex.net/stockquote.asmx/

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<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>
```

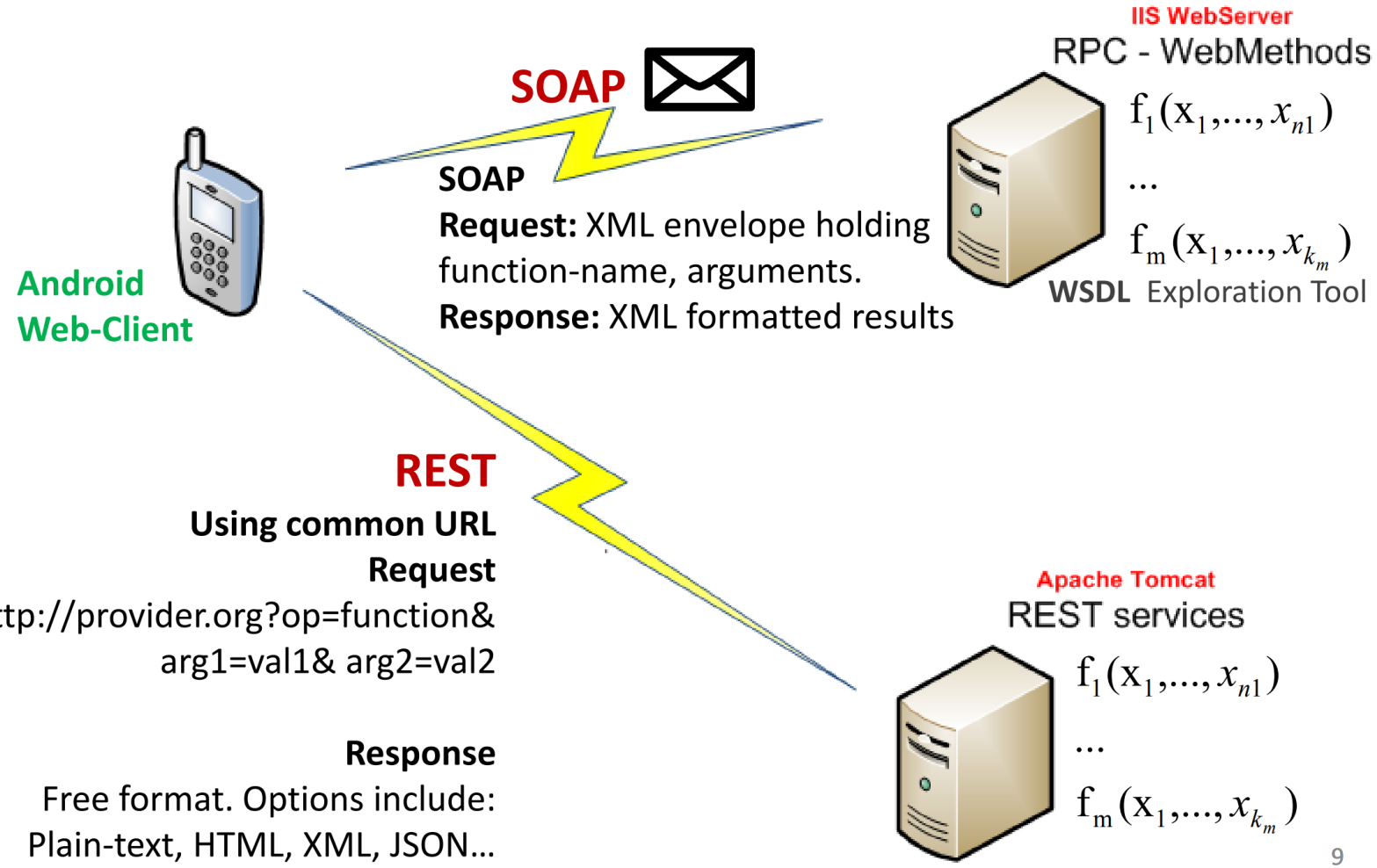
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<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:GetLastTradePriceResponse xmlns:m="Some-URI">
         <Price>34.5</Price>
       </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Use HTTP as the ‘carrier’ of a tightly-constructed messages

http://www.webservicex.net/stockquote.asmx (Web Service Description Language)

SOAP vs. REST


IIS WebServer
RPC - WebMethods
SOAP
SOAP
Request: XML envelope holding function-name, arguments.
Response: XML formatted results
WSDL Exploration Tool
Android Web-Client
REST
Using common URL
Request
http://provider.org?op=function& arg1=val1& arg2=val2
Response
Free format. Options include: Plain-text, HTML, XML, JSON…
Apache Tomcat
REST services

SOAP vs. REST

Although SOAP and REST technologies accomplish the same final goal, that is request and receive a service, there are various differences between them.

- REST users refer to their remote services through a conventional URL that commonly includes the location of the (stateless) server, the service name, the function to be executed and the parameters needed by the function to operate (if any). Data is transported using HTTP/HTTPS.
- SOAP requires some scripting effort to create an XML envelop in which data travels. An additional overhead on the receiving end is needed to extract data from the XML envelope. SOAP accepts a variety of transport mechanisms, among them HTTP, HTTPS, FTP, SMTP, etc.
  - SOAP uses WSDL (WebService Description Language) for exposing the format and operation of the services. REST lacks an equivalent exploratory tool.

What is REST

- REST stands for REpresentational State Transfer.

– REST is web standards based architecture and uses HTTP Protocol for data communication.

– It revolves around resource , where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

– REST was first introduced by Roy Fielding in 2000

- REST architecture

– a REST server simply provides access to resources

– a REST client accesses and presents the resources.

– each resource is identified by URIs/ global IDs

- REST uses various representations to represent a resource like text, JSON and XML.

Now JSON/REST is ‘preferred’ choice

GET /stockquote/DIS HTTP/1.1
Host: www.stockquoteserer.com
Accept: application/json

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: xxx

```
{
   “ticker”: “DIS”,
   “price”: 34.5
}
```

Let’s deconstruct this idea in detail …

What is and Why a RESTful Service

Early XML-based API fell out of favour along with the rise of the number of ‘mobile’ devices (and other ‘non-traditional’ client devices) due to the ‘heavy’ data payload and processing load

REST is an architectural style of building networked systems - a set of architectural constraints in a protocol built in that style.

The protocol in REST is HTTP (the core technology that drives the Web)

Popular form of API ... It is popularised as a guide to build modern distributed applications on the Web – let’s work with the components that the Web itself is built in.

REST itself is not an official standard specification or even a recommendation. It is just a “design guideline” for building a system (or a service in our context) on the Web

REST – REpresentational State Transfer

(of resources)

Roy Fielding coined the term

The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine. REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions and to ensure that protocol extensions would not violate the core constraints that make the web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).

- https://www.youtube.com/watch?v=w5j2KwzzB-0
- http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Resources in REST

In REST, everything starts and ends with resources (the fundamental unit).

What is a Resource?

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. today's weather in Los Angeles), a collection of other resources, a non-virtual object (e.g. a person), and so on. In other words, any concept that might be the target of an author's hypertext reference must fit within the definition of a resource. – Roy Fielding's dissertation.

Resources and Representational States

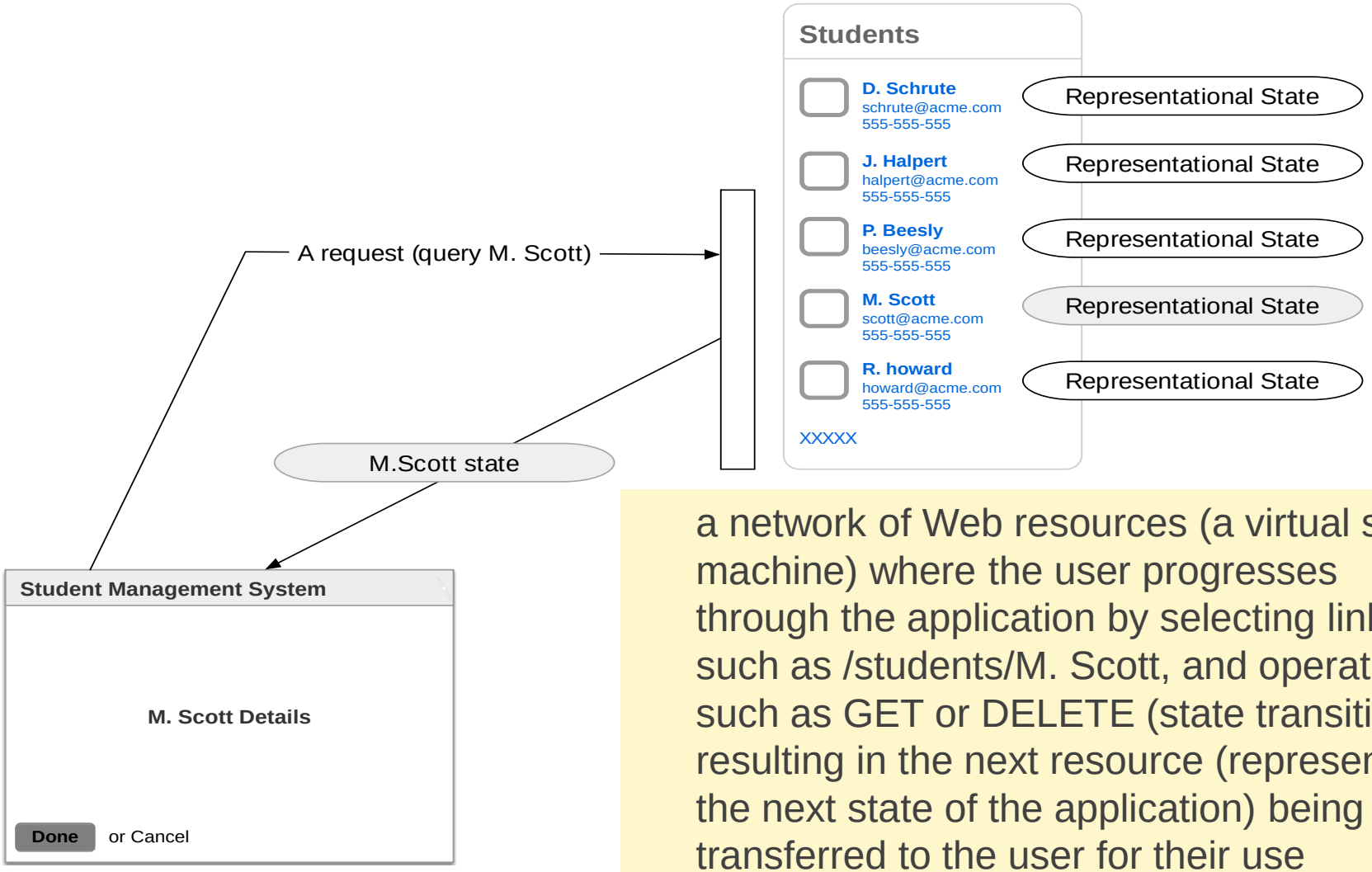
What is a Resource? (more concretely …)

A resource is a thing that:

- is unique (i.e., can be identified uniquely)
- has at least one representation
- has one or more attributes beyond ID
- has a potential schema, or definition
- can provide context (state) – which can change (updated)
- is reachable within the addressable universe
- collections, relationships

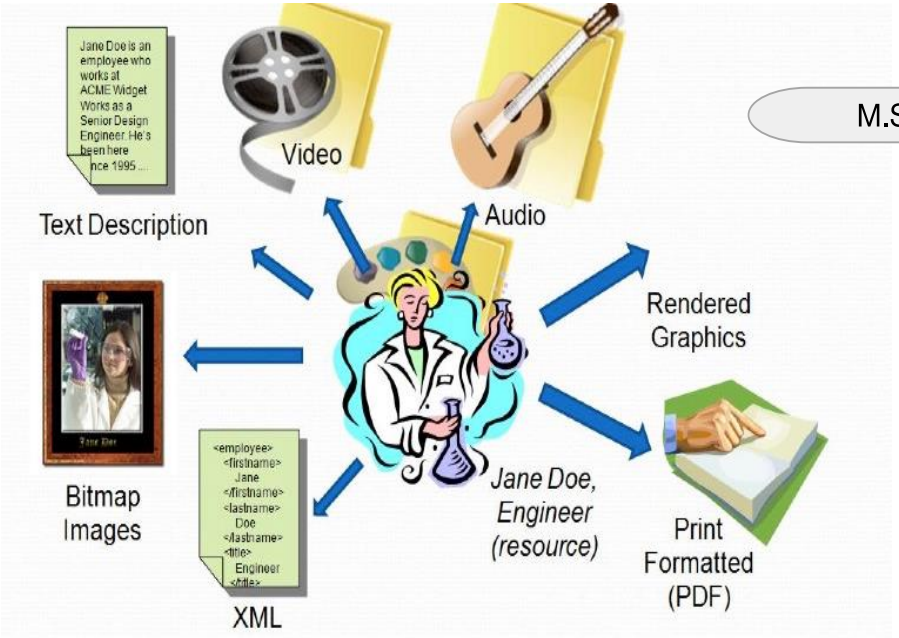
e.g., Students, Courses, Program

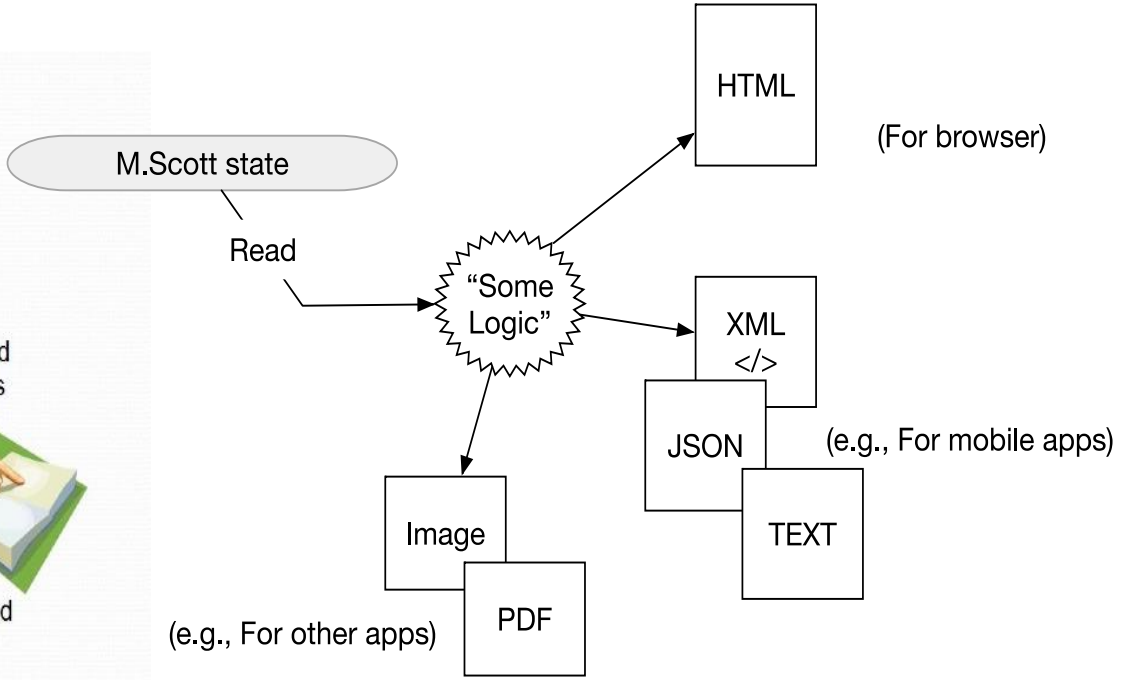
Representational State Transfer



a network of Web resources (a virtual state machine) where the user progresses through the application by selecting links, such as /students/M. Scott, and operations such as GET or DELETE (state transitions), resulting in the next resource (representing the next state of the application) being transferred to the user for their use

Resource's Multiple Representations





Based on the needs of the consumer … (REST principles do not specify "standard data format")

What makes a RESTful Service?

REST is an architectural style of building networked systems - a set of architectural constraints in a protocol built in that style.

A RESTful service/API MUST meet the architectural constraints by following the design guide/principles (i.e., you can say what is and is not RESTful)

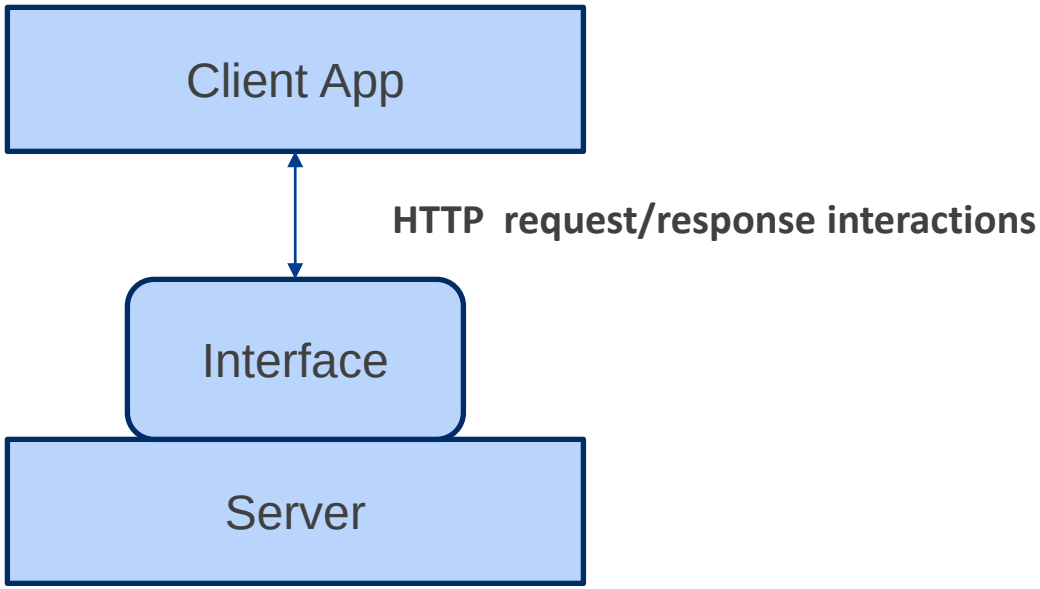
Architectural Constraints of REST

1. Client-Server
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System
6. Code on demand (optional)

If your design satisfies the first five, you can say your API is 'RESTful'

Client-Server

1. **Client-Server**
2. Uniform Interface
3. Statelessness
4. Caching
5. Layered System



The same idea as the classic two-tier architecture, foundation of REST architecture.

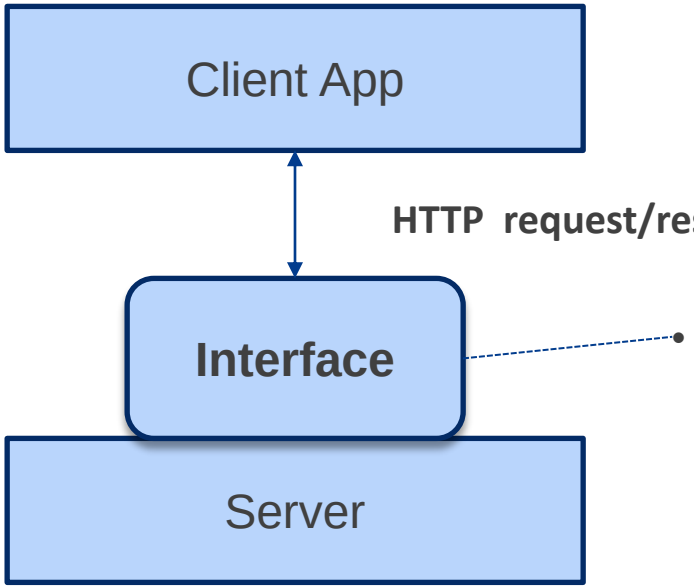
Server-side (business logic layer and beyond) is physically separated from the client app

WHY good thing? – separation of concerns, they can evolve without affecting each other

- Server-side: performance, scaling, data management, data security, etc.
- Client App: user experience, multiple form factor/devices support, etc.

Uniform Interface

1. Client-Server
2. **Uniform Interface**
3. Statelessness
4. Caching
5. Layered System



- UNIFORM INTERFACE
  - Uniform ≈ “Common” in All RESTful interface
  - Contract for communication btw C-S
  - Described using HTTP methods and Media Type only

REST design principles go into how to design the interface to meet the constraint.

On Uniform Interface – Resource URI

Resources are identified by a URI (Uniform Resource Identifier)

A resource has to have at least one URI

Most well-known URI types are URL and URN

- URN (Uniform Resource Name)
  - urn:isbn:0-486-27557-4 (Shakespeare's Romeo and Juliet book)
  - urn:isan:0000-0000-9E59-0000-O-0000-0000-2 (2002 spider man film)

- URL (Uniform Resource Locator)
  - file:///home/tommy/plays/RomeoAndJuliet.pdf
  - http://home/tommy/plays/RomeoAndJuliet.html

Every URI designates exactly one resource

- http://www.example.com/software/releases/1.0.3.tar.gz
- http://www.example.com/software/releases/latest

On Uniform Interface - Addressability

The resources must be addressable

An application is ‘addressable’ if it exposes its data set as resources (i.e., usually a large number of URIs)

- File system on your computer is addressable system
- The cells in a spreadsheet are addressable (cell referencing)

flickr – a good example of “addressable” Web
you can bookmark, use it as link in a program, you can email, etc.

Systems that are not addressable ?

REST advocates identification and addressability of resources as a main feature of the Uniform Interface constraint

On Uniform Interface - Representations

A resource needs a representation for it to be sent to the client

a representation of a resource - some data about the 'current state' of a resource

E.g., On some software project, a list of open bugs can have representations in :–

- an XML file,
- a web page,
- comma-separate-values,
- printer-friendly-format, etc.

when a representation of a resource may also contain metadata about the resource (e.g., books: book itself + metadata such as cover-image, reviews, other related books) - relationships.

Representations can flow the other way too: a client send a new or updated 'representation' of a resource and the server creates/updates the resource.

On Uniform Interface - Representations

And … you shall provide multiple representations …

Deciding between multiple representations

Option 1.

Have a distinct URI for each representation of a resource:

- http://www.example.com/release_doc/104.en (English)
- http://www.example.com/release_doc/104.es (Spanish)
- http://www.example.com/release_doc/104.fr (French)

Looks very “addressable” → good!

On Uniform Interface - Representations

Deciding between multiple representations

Option 2.

Use HTTP HEAD (metadata) for content negotiation:

Expose a single URL: http://www.example.com/release_doc/104

Client HTTP request contains Accept-Language

Content Negotiation (part of HTTP spec)

Other types of request metadata can be set to indicate all kinds of client preferences,

e.g., file format, payment information, authentication credentials, IP address of the client, caching directives, and so on.

Option 1 or 2 are both acceptable REST-based solution ...

On Uniform Interface – Description Syntax

Use pure HTTP methods as main operations on resources

What HTTP Spec says about the following methods:

GET: Retrieve a representation of a resource.

PUT: Create a new resource (new URI) or update a resource (existing URI)

DELETE: Clear a resource, after the URI is no longer valid

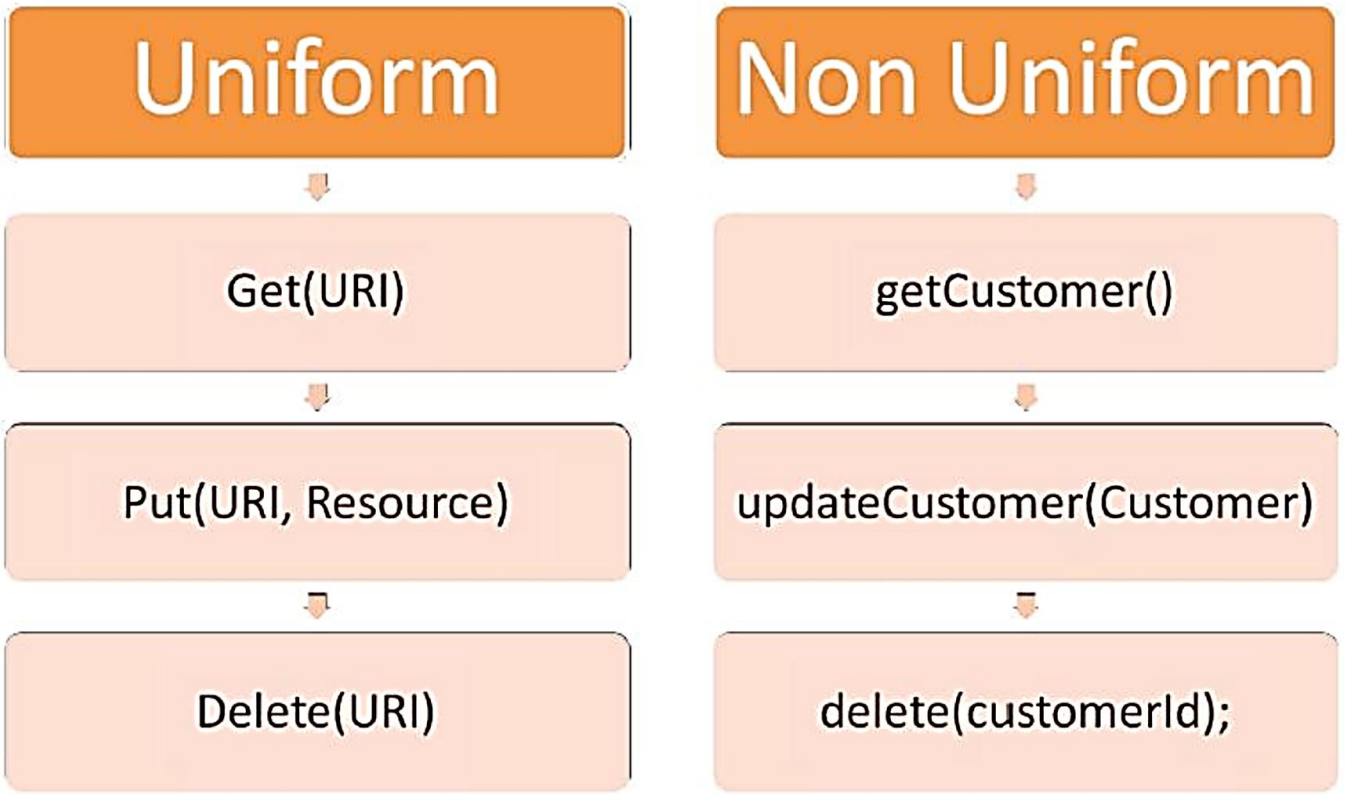
POST*: Modify the state of a resource. POST is a read-write operation and may change the state of the resource and provoke side effects on the server. Web browsers warn you when refreshing a page generated with POST.

HEAD, OPTIONS and PATCH

Similar to the CRUD (Create, Read, Update, Delete) databases operations

*POST: a debate about its exact best-practice usage ...

On Uniform Interface – “Uniform” Interface



On Uniform Interface – Description Syntax

Given a resource (coffee order): a representation in XML

<order>
*<drink>***latte***</drink>*
</order>

What HTTP Spec says about the input/output of the following methods. What does the following operations mean and what do they return?

POST /starbucks/orders

GET /starbucks/orders/order?id=1234

PUT /starbucks/orders/order?id=1234

DELETE /starbucks/orders/order?id=1234

On Uniform Interface - POST and PUT

POST creates a new resource

- But ... the server decides on that resource's URI and returns the new URI for the resource in the response
- Common examples: creating a new employee, a new order, a new blog posting, etc.

PUT "creates" or "updates" a resource:

- But ... the URI is given in the request input by client
- if existing, the contained entity is considered as a modified version of the resource ...

Generally, POST to create, PUT to update ...

`

On Uniform Interface - PUT and PATCH

PATCH is added later in the HTTP spec to support “partial updates” of a resource:

HTTP spec says: In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the origin server, and the client is requesting that the stored version be replaced. With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version.

PATCH /students/0001 [{ "op": "replace", "path": "/DOB", "value": "12/12/1990" }]

Neither safe nor idempotent ...

On Uniform Interface – Safe & Idempotent

Uniform Interface must be safe and idemponent

- *Being Safe*

Read-only operations ... The operations on a resource do not change any server state. The client can call the operations 10 times, it has no effect on the server state.

(like multiplying a number by 1, e.g., 4x1, 4x1x1, 4x1x1x1, ...)

- *Being Idempotent*

Operations that have the same “effect” whether you apply it once or more than once. An effect here may well be a change of server state. An operation on a resource is idempotent if making one request is the same as making a series of identical requests.

(like multiplying a number by 0, e.g., 4x0, 4x0x0, 4x0x0x0, ...)

On Uniform Interface – Safe & Idempotent

Safety and Idempotency

- GET: safe (and idempotent)
- HEAD and OPTION: safe

- PUT: - idempotent
  - If you create a resource with PUT, and then resend the PUT request, the resource is still there and it has the same properties you gave it when you created it.
    If you update a resource with PUT, you can resend the PUT request and the resource state won't change again
- DELETE: - idempotent
  - If you delete a resource with DELETE, it's gone. You send DELETE again, it is still gone !

What about POST?

On Uniform Interface – Safety & Idempotent

Why Safety and Idempotency matter:

The two properties let a client make reliable HTTP requests over an unreliable network.

Your GET request gets no response? make another one. It's safe.

Your PUT request gets no response, make another one. Even if your earlier one got through, your second PUT will have no side-effect

There are many applications that misuse the HTTP uniform interface. e.g.,

- GET https://api.del.icio.us/posts/delete
- GET www.example.com/registration?new=true&uname=John&passwd=123

Many expose unsafe operations as GET and there are many use of POST operation which is neither safe nor idempotent. Repeating them has consequences ...

On Uniform Interface – Safe & Idempotent

Why Safety and Idempotency matter

Allows the “Uniformity” in REST interface (≈ accepted convention by the community)

The point about REST Uniform Interface is in the ’uniformity’: that every service uses HTTP’s interface the same way.

It means, for example, GET does mean ’read-only’ across the Web no matter which resource you are using it on.

It means, we do not use methods in place of GET like doSearch or getPage or nextNumber.

It is not just having a method called GET in your service, it is about using it the way it was meant to be used.

Useful Resources

- Book: Undisturbed REST: A Guide to Designing the Perfect API
- API design guidance - Best practices for cloud applications | Microsoft Docs

Questions