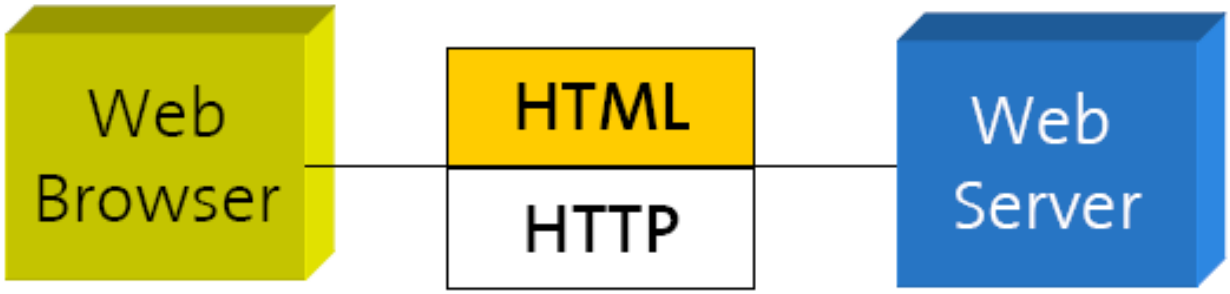
Lezione 10: REST

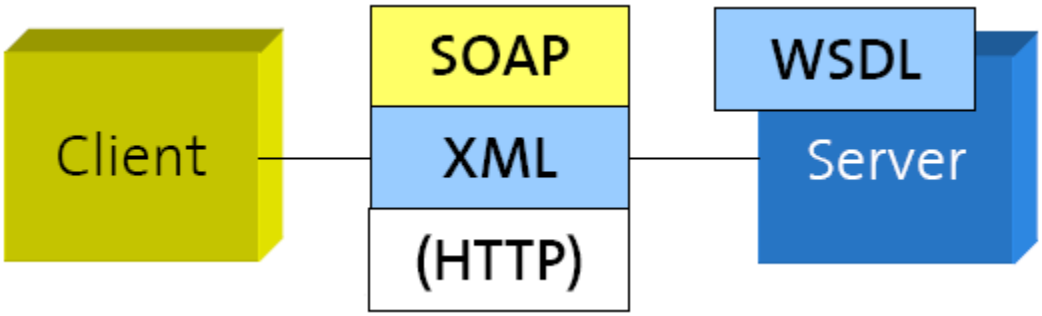
Introduzione

- REpresentational State Transfer (REST)
- Derivano dalla tesi di dottorato di Roy Fielding (2000)
- Gestiscono un tipo di contenuto variabile
  - Può essere XML
  - Principalmente JSON

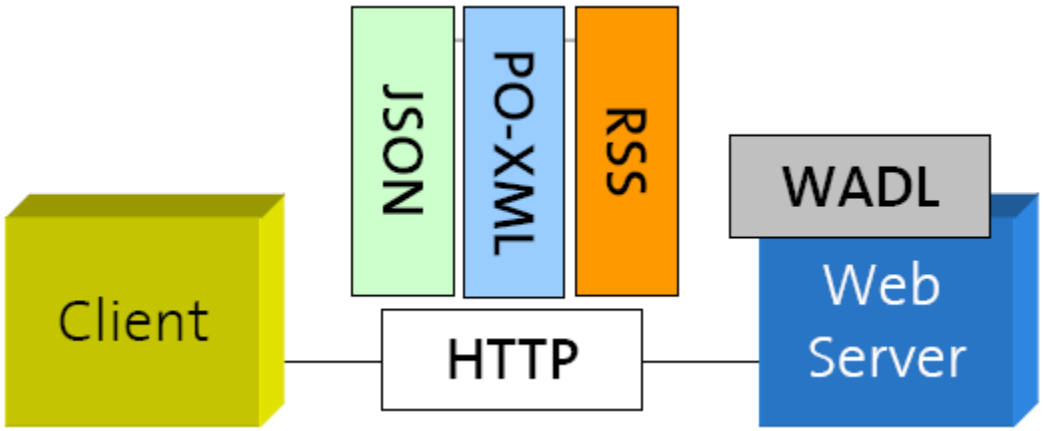
Siti web (1992)



WS-* Web Service (2000)



RESTful Web Service (2007)



REpresentational State Transfer

- Moderna architettura web
  - Descritta attraverso un architectural style
  - «Architectural style is a named, coordinated set of architectural constraints»

- Design architetturale
  - Considera un sistema nella sua interezza, senza vincoli (constraint)
  - Identifica e applica vincoli incrementalmente agli elementi del sistema
    - Differenzia lo spazio del design
    - Permette ai parametri che influenzano il comportamento del sistema di fluire naturalmente, in armonia con il sistema

Elementi architetturali: ROA

- Resource-Oriented Architecture
  - REST è un'astrazione di elementi architetturali in un sistema hypermedia distribuito
  - REST ignora i dettagli implementativi dei componenti e la sintassi dei protocolli
  - REST si focalizza sul ruolo dei componenti, vincoli per interazione con altri componenti e l'interpretazione dei dati

- RESTFul Web Service
  - È una configurazione di URI, HTTP, XML/JSON che lavora come il resto del web

Data element

- La natura e lo stato dei data element è un aspetto essenziale di REST
  - Dovuto alla natura degli hypermedia distribuiti
- Quando un link è selezionato l'informazione fluisce dalla locazione dove è memorizzata a quella dove verrà usata
- Tre possibilità di comunicazione (REST supporta un ibrido delle tre)
  1. Fare rendering del dato dove si trova e mandare un'immagine a formato fisso al ricevente
  2. Incapsulare il dato con l'engine di rendering e inviarlo al ricevente
  3. Mandare il dato all'utente che farà il rendering

Data element

- REST fornisce una versione ibrida delle tre opzioni
  - Comprensione condivisa di dati e metadati
  - Limita lo scope di cosa viene rivelato all'interfaccia standard
- Componenti REST trasferiscono una risorsa usando una delle rappresentazioni standard
  - Rappresentazione selezionata in base ai desideri del ricevente e alla natura della risorsa
  - Conoscenza della rappresentazione originale nascosta dietro le interfacce
  - La rappresentazione consiste di istruzioni in un formato standard relative a un engine di rendering

Data element

- REST fornisce
  - Separation of concern
  - Scalabilità
  - Permette information hiding attraverso interfacce generiche che supportano encapsulation e service evolution
  - Fornisce un set di funzionalità attraverso un engine di feature scaricabile

Data element

Data Element	Modern Web Examples
resource	the intended conceptual target of a hypertext reference
resource identifier	URL, URN
representation	HTML document, JPEG image
representation metadata	media type, last-modified time
resource metadata	source link, alternates, vary
control data	if-modified-since, cache-control

Resource

- Resource
  - Ogni informazione che può avere un nome (ad es., documento, immagine, persone, …)
  - Ogni concetto che può essere target di riferimento ipertestuale
- Ogni resource è una membership function che al tempo t viene associata a un set di entità/valori
  - Valori sono la rappresentazione della risorse e/o gli identificativi
  - Risorse sono vuote, statiche, dinamiche
  - L'unica cosa sempre statica è la semantica della risorsa
  - Il mio libro preferito (risorsa dinamica)
  - Il libro XYZ (risorsa statica)

Resource identifier

- Resource identifier
  - L’autorità di naming deve mantenere la validità semantica del mapping tra identificatore e risorsa
  - L’autore sceglie l’identificatore
  - URI/URN
  - Due URI diverse possono puntare allo stessa risorsa, una stessa URI può ritornare informazioni su più risorse

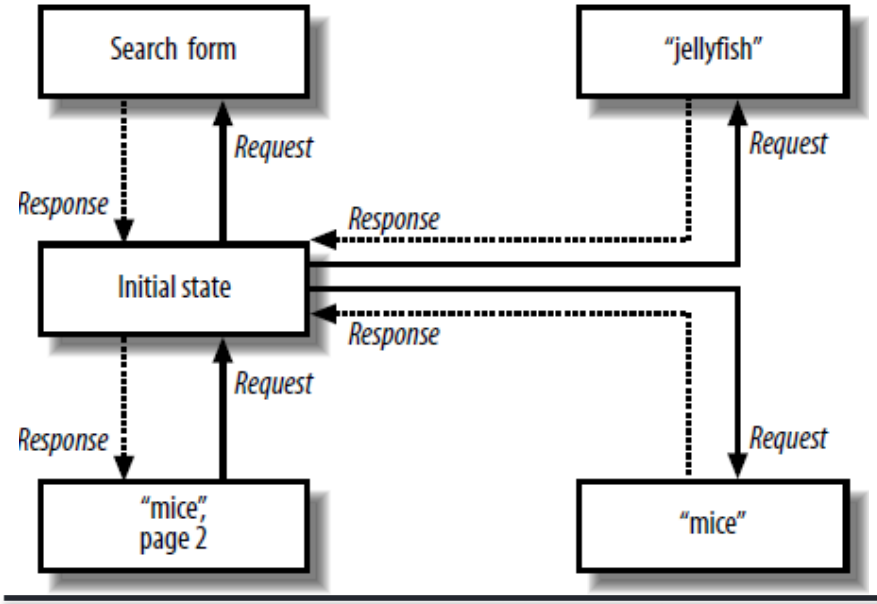
Addressability

- Un'applicazione è indirizzabile se
  - Espone i suoi dati come risorse
  - Espone un URI per ogni informazione di interesse
  - Il file system è indirizzabile (ogni file ha un percorso univoco)
- Uno dei requisiti fondamentali per gli end user
- Addressability permette di supportare bookmarking, linking, emailing…

Statelessness

- Ogni richiesta HTTP è eseguita in isolation
- Il server non usa informazioni da richieste precedenti, il client inserisce tutto ciò che è necessario a soddisfare la richiesta
- Consideriamo statelessness in funzione di addressability
  - Statelessness definisce i possibili stati di un server come risorse
- Nonostante il principio di statelessness, spesso il web è implementato stateful

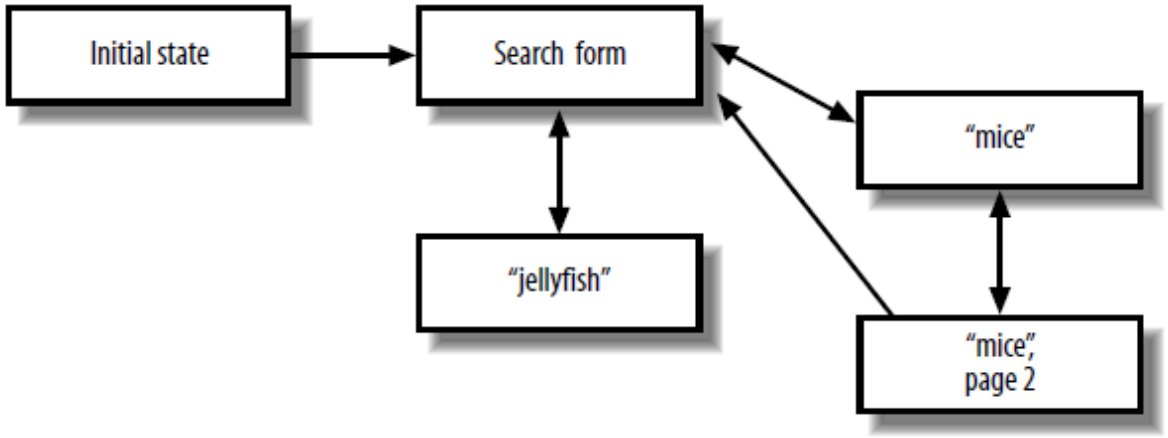
Statelessness



http://www.google.com/search?q=jellyfish

Stateless

http://www.google.com/search?q=mice



Stateful

http://www.google.com/search?q=jellyfish&start=10

Statelessness

- Perché statelessness?
  - Lo stato renderebbe le richieste molto più semplici, ma…
  - … la gestione del client HTTP sarebbe molto più complicata
    - Necessità di mantenere lo stato e la sincronizzazione tra client e server
  - Senza stato si rimuovono gran parte delle condizioni di fallimento (failure condition)
    - Ad es., nessuna necessità di controllare il timeout con una sola richiesta, il server non perde mai traccia di dove si trova il client, il client non sbaglia mai la directory di esecuzione

Statelessness: Server

- Perché statelessness?
  - Facile distribuire applicazioni tra server in load balancing
  - Scaling up fatto semplicemente aggiungendo server senza preoccuparsi dello stato
  - Applicazioni facili da cachare
    - Si guarda al risultato di una singola richiesta

Statelessness: Client

- Perché statelessness?
  - Il client può processare una ricerca fino alla n-esima pagina e ricominciare da dove ha lasciato una settimana dopo
  - Una URI raggiunta dopo ore di lavoro funzionerà alla stessa maniera se usata come prima di una nuova sessione
- Sessione può essere aggiunta a livello applicativo
  - Cookie con stringa univoca che viene associata lato server allo stato

Representation

- Representation
  - Usata per ottenere lo stato di una risorsa
  - Trasferita tra componenti
  - È una sequenza di byte più metadati che la descrivono
    - Metadati: informazioni su un libro: titolo, autori, prezzo
    - Dati: copia elettronica del libro
  - Documenti, file, HTTP message entity…

Representation

- Una risorsa può avere rappresentazioni diverse
  - Diversi linguaggi
  - Diversi formati
  - …
- Il client può scegliere quale rappresentazione
  - Ogni rappresentazione ha un'URI diversa
  - Accept-header
- Il server decide la rappresentazione in base a header della richiesta e metadati addizionali

Representation

- Representation include dati, metadati e metadati per metadati (di solito per verifica di integrità)
  - Representation e resource metadata inclusi nella risposta
- Control data definiscono l'obiettivo del messaggio scambiato
  - Azione richiesta, significato della risposta
  - Ad esempio, modifica la gestione della cache
- Rappresentazione del formato dei dati definito come media type (hypermedia)
  - Messaggio processato in accordo al control data e media type
  - Media type servono per processo automatico e/o rendering per l'utente

Representation

- Il design del media type può impattare sulle performance percepite dall'utente
  - Informazioni importanti messe per prime
  - Visualizzate subito
  - Altre informazioni visualizzate incrementalmente mentre arrivano
  - Ad esempio, rendering di pagine web

Link e connettività

- *Hypermedia as the engine of application state*
  - Link e form all'interno di una pagina web
  - Connettività (supportata da link e form)

- Lo stato di una «sessione» HTTP è gestita dal client come stato applicativo
  - Ad esempio, la query che faccio e la pagina in cui sono in un motore di ricerca
  - Il server fornisce suggerimenti su come andare dal presente al prossimo stato (link a «next», «page 2», «cached page» nella pagina di ricerca)
  - Risorse devono essere collegate tramite link nella loro rappresentazione

Visione architetturale

- Process view
  - Mostra le interazioni tra componenti rivelando il percorso dei dati nel sistema
- Connector view
  - Si concentra sulla comunicazione tra componenti, specialmente i vincoli che definiscono interfacce di risorse generiche
- Data view
  - Rivela lo stato dell'applicazione al fluire dell'informazione

RESTful Services

REpresentational State Transfer

- Nonostante REST sia un architectural style per networked hypermedia application, è usato principalmente per costruire lightweight, maintainable, e scalable Web services
- Un servizio basato su REST è chiamato RESTful service
- REST non dipende da nessun protocollo, anche se quasi tutti i RESTful service usano HTTP

REpresentational State Transfer (Client)

- Visione astratta del client REST
  - Definire informazioni da aggiungere all'HTTP request: HTTP method, URI, HTTP header, documenti da mandare nel body
  - Creare l'HTTP request e mandarla al server HTTP
  - Parsare la risposta (codice, header, entity body) e fornire le informazioni al codice REST

REpresentational State Transfer (Client)

- Il client usa HTTP library per lettura/scrittura di messaggi HTTP
  - Feature obbligatorie: supporto HTTPS/SSL, supporto HTTP method, possibilità di customizzare header e body del messaggio, accesso alla risposta (compreso header e response code), HTTP proxy
  - Feature opzionali: supporto per dati in forma compressa, cache delle risposte, supporto dell'autenticazione HTTP (Basic, Digest, WSSE), supporto per HTTP redirect, capacità di interpretare e creare cookie
- Try It! CURL per fare accesso a servizi REST con una semplice command line

REpresentational State Transfer (Client)

- Il client usa XML parser per interpretare la risposta
  - Document-based model
    - DOM, Tree-style parser
    - Struttura data annidata accessibile e modificabile tramite XPath e CSS
    - Richiede caricamento di tutto il documento in memoria
  - Event-based strategy o pull
    - SAX
    - Trasforma il documento in un flusso di eventi (inizio/fine tag, commenti XML, dichiarazione di entità)
    - Gestisce un evento alla volta e può fermare il parsing su richiesta

REpresentational State Transfer (Client)

- Client REST possono essere sviluppati in molti linguaggi
  - Ruby
  - Python
  - Java
  - C#
  - PHP
  - Javascript
  - E altri: ActionScript, C, C++, Common Lisp, Perl…

REpresentational State Transfer (Server)

- Visione astratta del server REST
  - Accettare HTTP request
  - Parsare l'HTTP request ed estrarre le informazioni importanti
  - Spedire la risposta la risposta al client
- Anche il server può utilizzare HTTP library

REpresentational State Transfer (Server)

- REST accede ai dati usando URI
  - http://www.pippo.com/resource/travels
  - http://www.pippo.com/resource/travels/newyork

- I suoi quattro principi mostrano il successo e la scalabilità del protocollo HTTP
  - Resource Identification attraverso URI
  - Uniform Interface per tutte le risorse
    - GET (Query the state, idempotent, can be cached): ottiene una rappresentazione della risorsa
    - POST (Create a child resource): crea una nuova risorsa
    - PUT (Update, transfer a new state): crea o aggiorna una risorsa
    - DELETE (Delete a resource): cancella una risorsa
  - "Self-Describing" Message attraverso metadati e rappresentazione di risorse multiple
  - Hyperlink per definire le transizioni di stato dell'applicazione e le relazioni tra risorse

REpresentational State Transfer

- Metodi HTTP aggiuntivi della uniform interface
  - HEAD: riceve solo i medati di una risorsa
  - Options: controlla quali metodi sono supportati dalla risorsa
    - Ritorna un HTTP allow
    - Allow: GET, HEAD (risorsa in sola lettura)

Safety e idempotenza

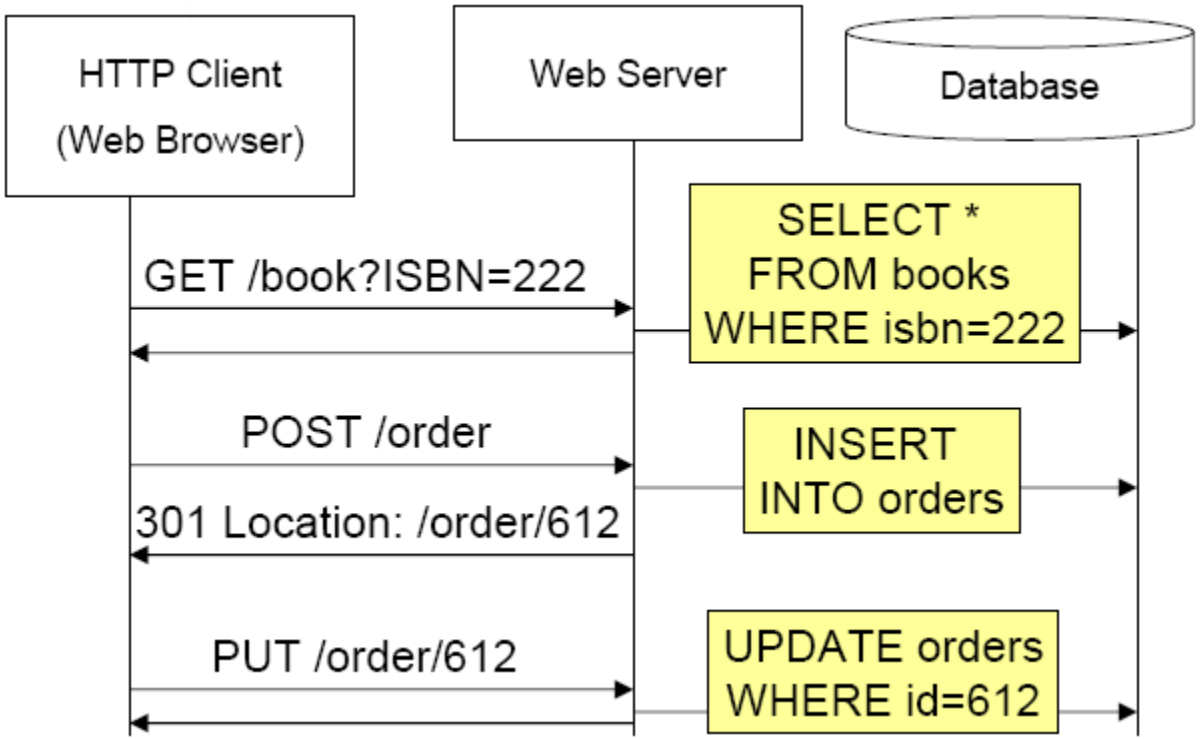
- Safety
  - GET, HEAD
  - Richieste safe, solo lettura, non causano danni catastrofici
  - Side effect: incremento un contatore, loggo la richiesta

- Idempotenza
  - GET, HEAD, PUT, DELETE
  - L'esecuzione di un commando una o più volte ha lo stesso effetto

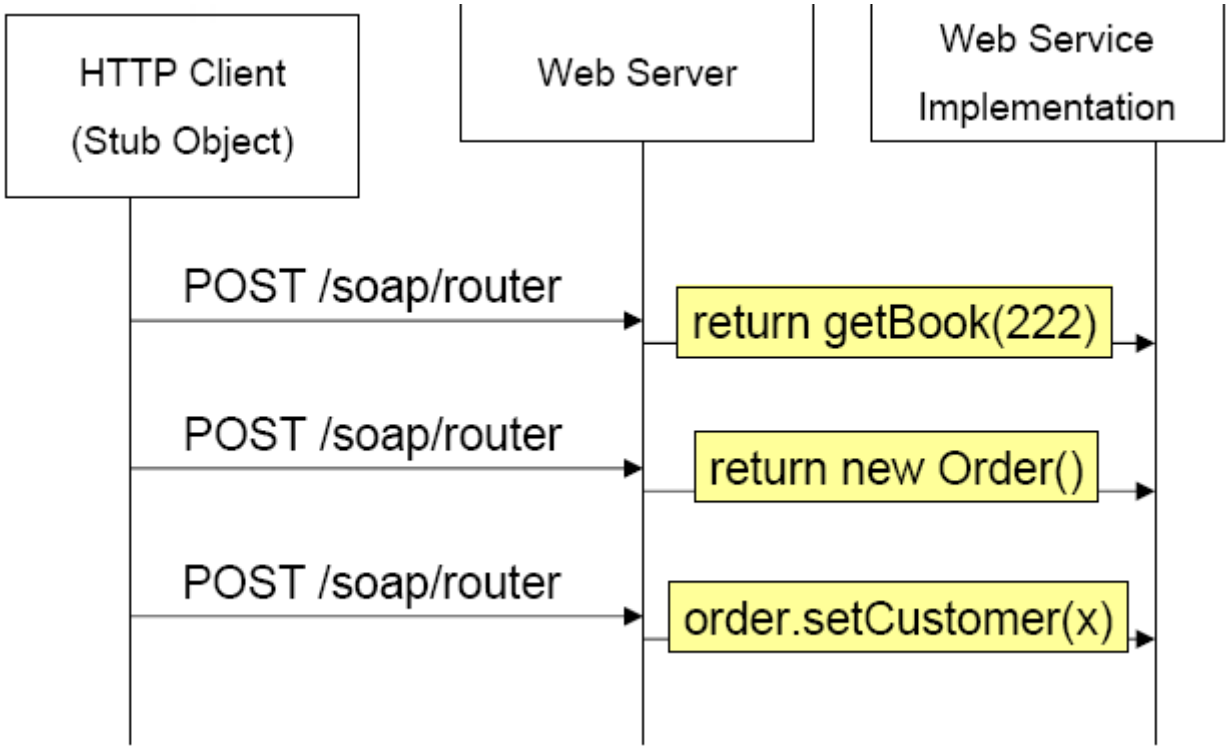
RESTful Web Service

- RESTful Web Service usano risorse
  - Figure, video, pagine web, business information, o qualunque cosa possa essere rappresentata in un computer-based system
- Obiettivo: fornire un hook attraverso cui il client può accedere queste risorse
- Proprietà e feature
  - Representations
  - Messages
  - URIs
  - Uniform interface
  - Stateless
  - Links between resources
  - Caching

RESTful Web Service: Esempio



Web Service: Esempio (da una prospettiva REST)



Representations

- REST si focalizza su risorse e come accedere a risorse
- Prima cosa da fare identificare le risorse e poi rappresentarle
- Possono essere usate diverse rappresentazioni (ad es., JSON/XML)

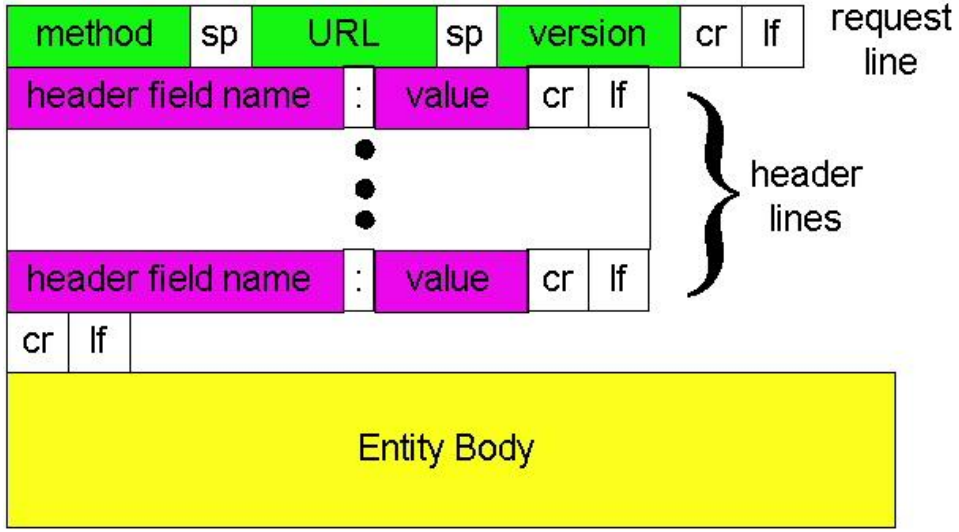
```
{
   "ID": "1",
   "Name": "M Vaqqas",
   "Email": "m.vaqqas@gmail.com",
   "Country": "India"
}
```

```
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

Representations

- Selezione del formato della risposta dipende dal tipo di cliente e da parametri della richiesta
- Una buona rappresentazione ha le seguenti caratteristiche
  - Client e server conoscono il formato della rappresentazione
  - Una rappresentazione rappresenta la risorsa nella sua interezza
    - Se la risorsa ha rappresentazioni parziali può essere divisa in risorse figlie, con vantaggi computazionali e di performance
  - Permette di collegare diverse risorse, inserendo URI or unique ID di altre risorse nella rappresentazione

Messages: Richiesta

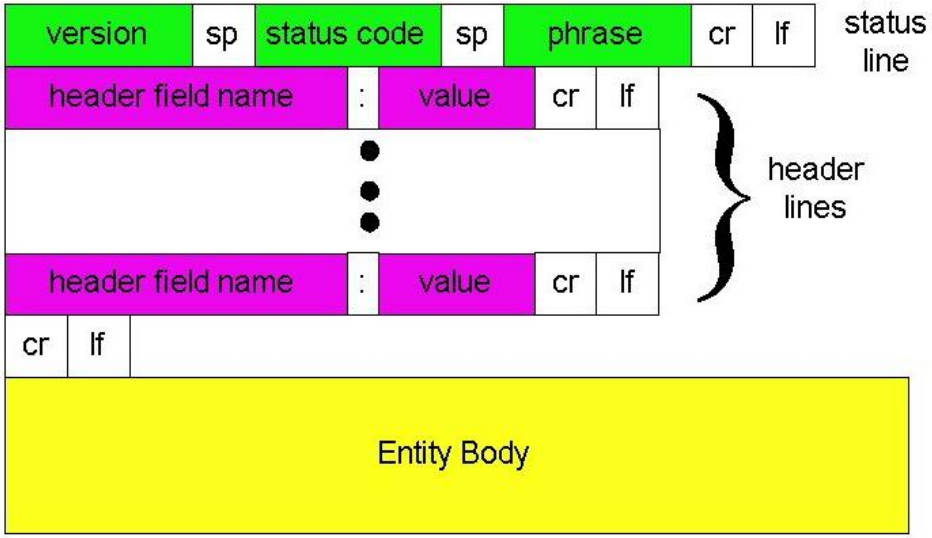


POST http://MyService/Person/
Host: MyService
Content-Type: text/xml; charset=utf-8
Content-Length: 123

```
<?xml version="1.0" encoding="utf-8"?>
<Person>
  <ID>1</ID>
  <Name>M Vaqqas</Name>
  <Email>m.vaqqas@gmail.com</Email>
  <Country>India</Country>
</Person>
```

GET http://www.w3.org/Protocols/rfc2616/rfc2616.html HTTP/1.1
Host: www.w3.org
Accept: text/html,application/xhtml+xml,application/xml; …
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 …
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,hi;q=0.6

Messages: Risposta




```
HTTP/1.1 200 OK
Date: Sat, 23 Aug 2014 18:31:04 GMT
Server: Apache/2
Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT
Accept-Ranges: bytes
Content-Length: 32859
Cache-Control: max-age=21600, must-revalidate
Expires: Sun, 24 Aug 2014 00:31:04 GMT
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns='http://www.w3.org/1999/xhtml'>
 <head>
  <title>Hypertext Transfer Protocol -- HTTP/1.1</title>
 </head>
 <body>
  ...
```

Uniform Resource Identifier

- Standard Internet per naming e identification di risorse (nasce nel 1994, rivisto fino al 2005)
- REST non raccomanda l'uso di "nice" URI
- In molti stack HTTP, URI non possono avere lunghezza arbitraria (4Kb)

Uniform Resource Identifier: Buone Pratiche

- Preferire sostantivi (plurali) a verbi
- Non usare spazi
- Mantenere le URI corte
- Usare uno schema di convenzione conosciuto in modo che il client possa costruirle da programma
- Seguire uno schema di passaggio dei parametri posizionale (invece di usare un encoding key=value&p=v)
- URI postfix possono essere usati per specificare il tipo del contenuto
- Non cambiare URI, usare la redirezione (HTTP status code 300) se c'è bisogno di cambiare gli URI

Addressing Resources: URI

- REST richiede un'URI per ogni risorsa
- RESTful service usa una human readable URI per indirizzare le risorse
  - Gerarchia basata sul concetto di directory
  - URI identifica una risorsa o collezione di risorse
  - L'operazione è determinata dall'HTTP method
  - L'URI non dovrebbe dire nulla riguardo operazione/azione in modo da poter chiamare un'URI con diversi HTTP method per eseguire diverse operazioni

- Lista di persone con dati personali:
  - Protocol://ServiceName/ResourceType/ResourceID
  - Esempio: http://MyService/Persons/1
  - Da evitare: http://MyService/FetcthPerson/1 oppure http://MyService/DeletePerson?id=1

Addressing Resources: URI

- REST permette di usare URI costruite con l'aiuto di query parameter
- Svantaggi
  - Maggior complessità, minor leggibilità
  - Search-engine crawlers e indexers come Google ignorano URI con query parameter
- Tuttavia, i query parameter sono necessari in alcuni casi (fornire parametric a un'operazione)
  - Ad esempio, selezione del formato di rappresentazione http://MyService/Persons/1?format=xml&encoding=UTF8 oppure http://MyService/Persons/1?format=json&encoding=UTF8
  - Includere i parametri nell'URI in una relazione padre figlio non è corretto
    - http://MyService/Persons/1/json/UTF8
  - Query parameter supportano parametri opzionali
- Risultato: utilizzare i query parameters sono per fornire i valori di parametri a un processo

Uniform Interface Principle: Esempio CRUD

CRUD	REST	
CREATE	POST 	Create a sub resource
READ	GET 	Retrieve the current state of the resource
UPDATE	PUT 	Initialize or update the state of a resource at the given URI
DELETE	DELETE 	Clear a resource, after the URI is no longer valid

POST vs PUT

- POST usata per creare risorse subordinate, PUT usata per creare/modificare risorse
- Con la PUT il client decide il nome della risorsa
  - Oggetto S3 è completamente determinato dal suo nome e quello del bucket
- Con la POST il server decide il nome della risorsa
  - In un weblog è più difficile per il client capire quale è il nome da dare a un entry
  - Il client genera una risorsa senza conoscere l'URI, basta quella del padre

POST vs PUT

	PUT to a new resource	PUT to an existing resource	POST
/weblogs	N/A (resource already exists)	No effect	Create a new weblog
/weblogs/myweblog	Create this weblog	Modify this weblog's settings	Create a new weblog entry
/weblogs/myweblog/entries/1	N/A (how would you get this URI?)	Edit this weblog entry	Post a comment to this weblog entry

Statelessness

- RESTful service è stateless, non mantiene lo stato dell'applicazione per il client
- Stateless service è più facile da deployare, mantenere e scalare, fornisce miglior tempo di risposta
- Stateless
  - Request1: GET http://MyService/Persons/1 HTTP/1.1
  - Request2: GET http://MyService/Persons/2 HTTP/1.1
- Stateful design
  - Request1: GET http://MyService/Persons/1 HTTP/1.1
  - Request2: GET http://MyService/NextPerson HTTP/1.1

Link tra risorse

- Una resource representation può contentere link ad altre risorse come in una pagina HTML (che contiene link ad altre pagine HTML)
- Un client richiede una risorsa che contiene altre risorse…
- … invece di scaricarle tutte, le risorse vengono elencate

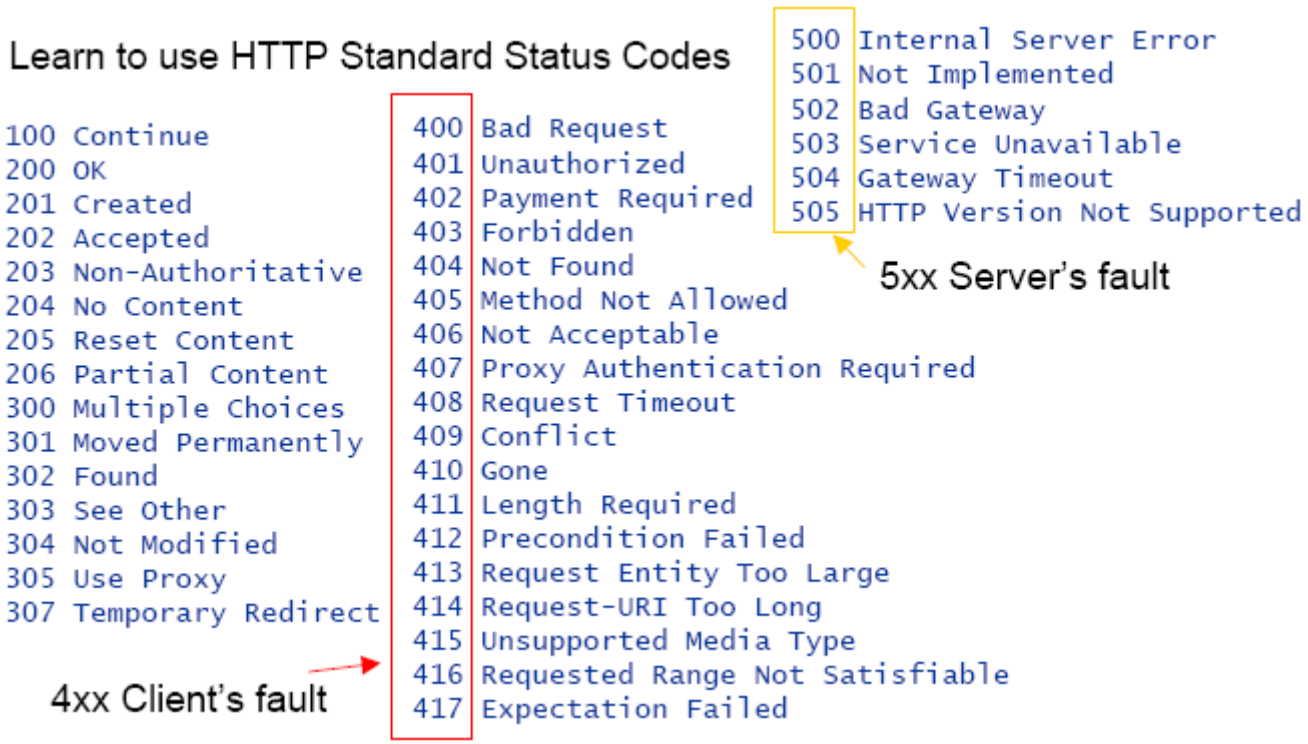
```
<Club>
  <Name>Authors Club</Name>
  <Persons>
    <Person>
      <Name>Mario Rossi</Name>
      <URI>http://MyService/Persons/1</URI>
    </Person>
    <Person>
      <Name>Marco Verdi</Name>
      <URI>http://MyService/Persons/12</URI>
    </Person>
  </Persons>
</Club>
```

Documentazione

- Nonostante sia semplice identificare e navigare attraverso un servizio RESTful, un minimo livello di documentazione è molto utile

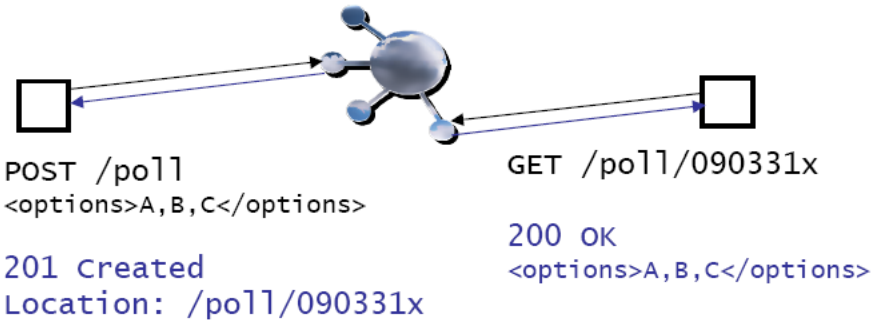
- Informazioni utili
  - Service Name
  - Address
  - Resource, Methods, URI, Description (per ogni resource)

Gestione delle eccezioni



Semplice API per Doodle: Esempio

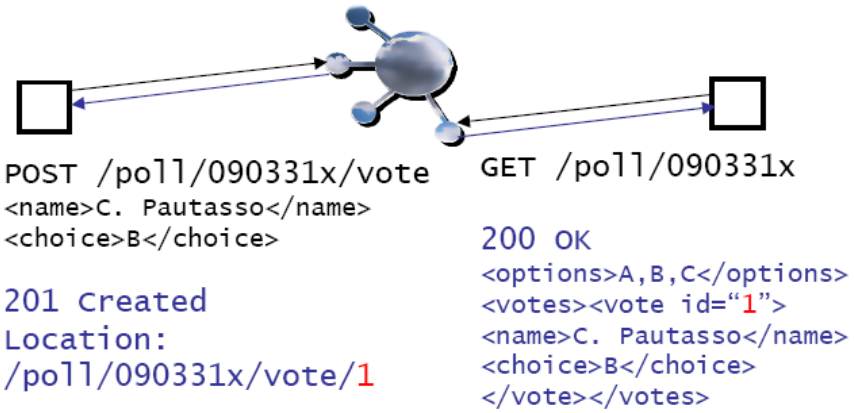
- Creare un sondaggio (trasferire lo stato di un nuovo sondaggio al servizio Doodle)



- Leggere il sondaggio (trasferire lo stato del sondaggio dal servizio Doodle)

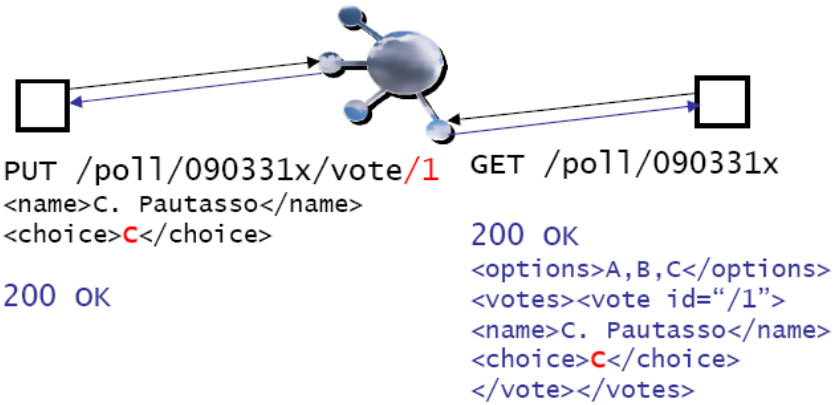
Semplice API per Doodle: Esempio

- Partecipare al sondaggio creando una sub-resource voto



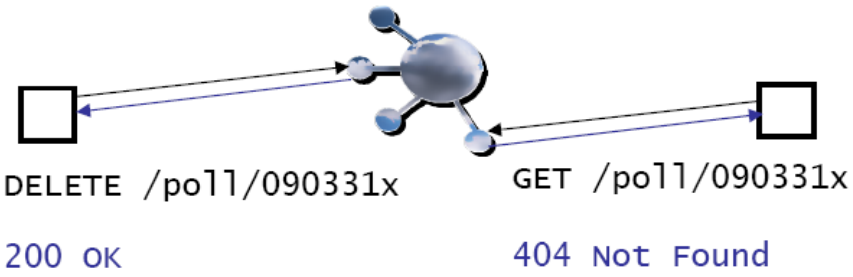
Semplice API per Doodle: Esempio

- Voti esistenti possono essere aggiornati (header di controllo dell'accesso non sono mostrati)



Semplice API per Doodle: Esempio

- Sondaggi possono essere cancellati quando una decisione è stata presa



Approfondimenti

REST Design: Suggerimenti

- Capire GET vs. POST vs. PUT
- Rappresentazioni multiple
  - Negoziazione basata su Content-Type
- Gestione delle eccezioni
  - Idempotent vs. Unsafe

POST vs GET

- GET è un'operazione read-only
  - Può essere ripetuta senza influenzare lo stato delle risorse (idempotent) e può essere cachata
- POST è un'operazione read-write
  - Può cambiare lo stato della risorsa e provocare alcuni side effect lato server
  - Web browser propongono un warning quando si fa refresh di una pagina generata usando POST

POST vs PUT

- Come creare una risorsa (inizializzarne lo stato)?

```
PUT /resource/{id}
```

- Problema: come assicurare che {id} sia univoco?
  - Risorse create da client multipli e concorrenti

```
POST /resource
201 Created
Location: /resource/{id}
```

- Soluzione: lasciare al server il dovere di calcolare l'id univoco

Negoziazione del contenuto

- Negoziare il formato del messaggio non richiede l'invio di più messaggi

```
GET /resource
Accept: text/html, application/xml,
   application/json
```

- Il client elenca una lista di formati (MIME types) che capisce

```
200 OK
Content-Type: application/json
```

- Il server sceglie la più appropriata per la risposta

Negoziazione del contenuto forzata

- URI generica supporta negoziazione del contenuto

```
GET /resource
Accept: text/html, application/xml,
   application/json
```

- URI specifica punta a un formato di rappresentazione specifico usando il postfix

```
GET /resource.html
GET /resource.xml
GET /resource.json
```

- Warning: “best practice” convenzionale (non uno standard)

Idempotent vs Unsafe

- Richieste idempotenti possono essere processate diverse volte senza side effect (lo stato del server non cambia)

```
GET /book
PUT /order/x
DELETE /order/y
```

- Se qualcosa va male (server down, server internal error), la richiesta può essere semplicemente rimandata fino a quando il server non ritorna attivo

Idempotent vs Unsafe

- Richieste unsafe modificano lo stato del server e non possono essere ripetute senza ulteriori effetti

```
Withdraw(200$) //unsafe
Deposit(200$)  //unsafe
```

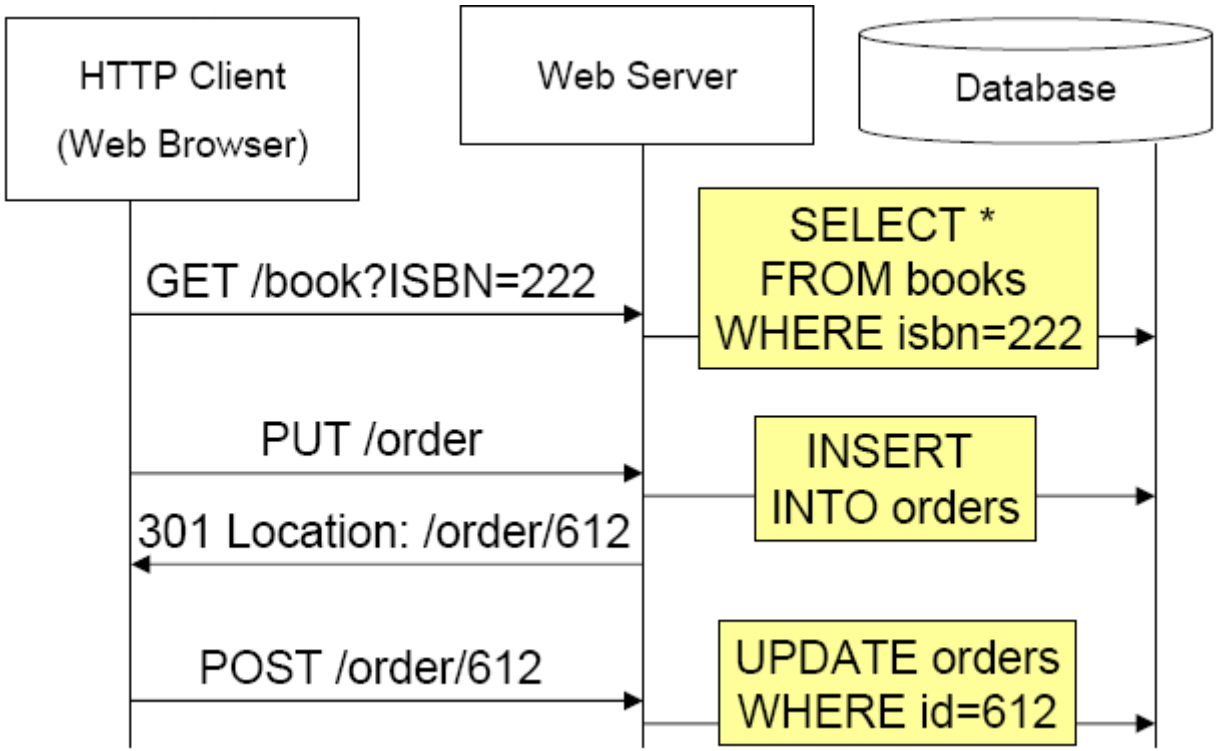
- Richieste unsafe richiedono gestione delle situazioni eccezionali (ad es., state reconciliation)

```
POST /order/x/payment
```

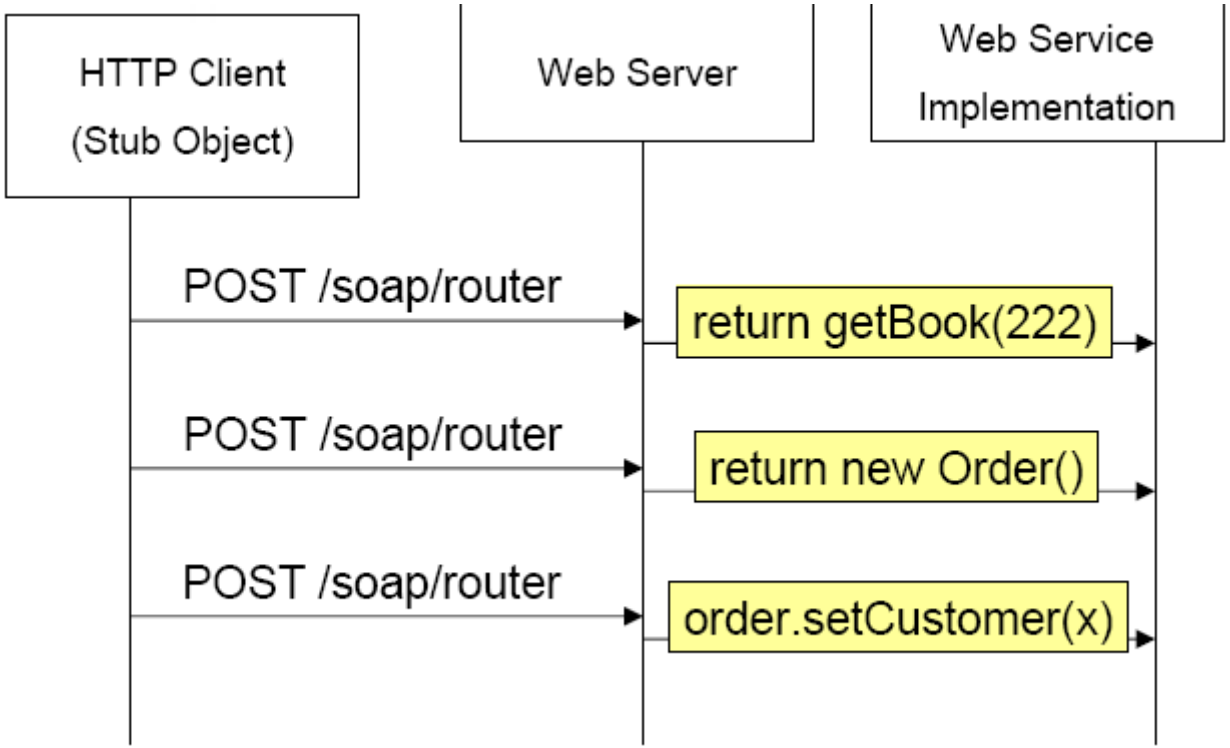
- In alcuni casi le API possono essere ridefinite per usare operazioni idempotenti

```
B = GetBalance() //safe
B = B + 200$     //local
SetBalance(B)    //safe
```

RESTful Web Application: Esempio

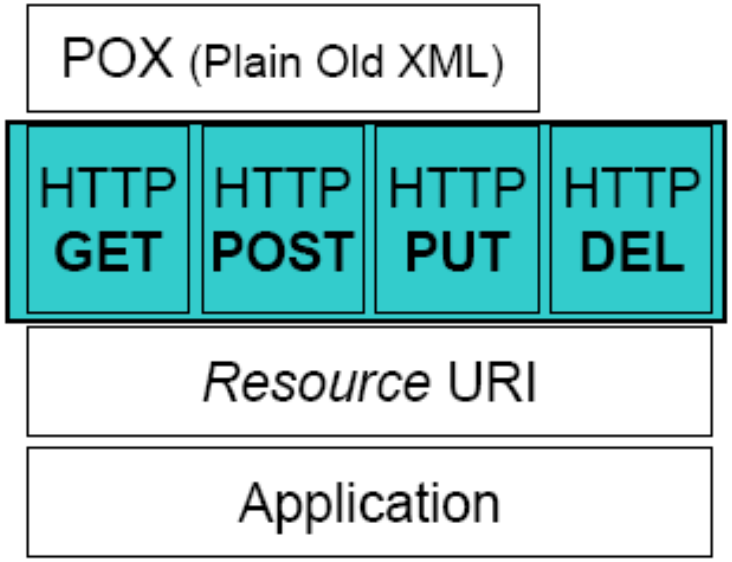


Web Service Example: da una prospettiva REST

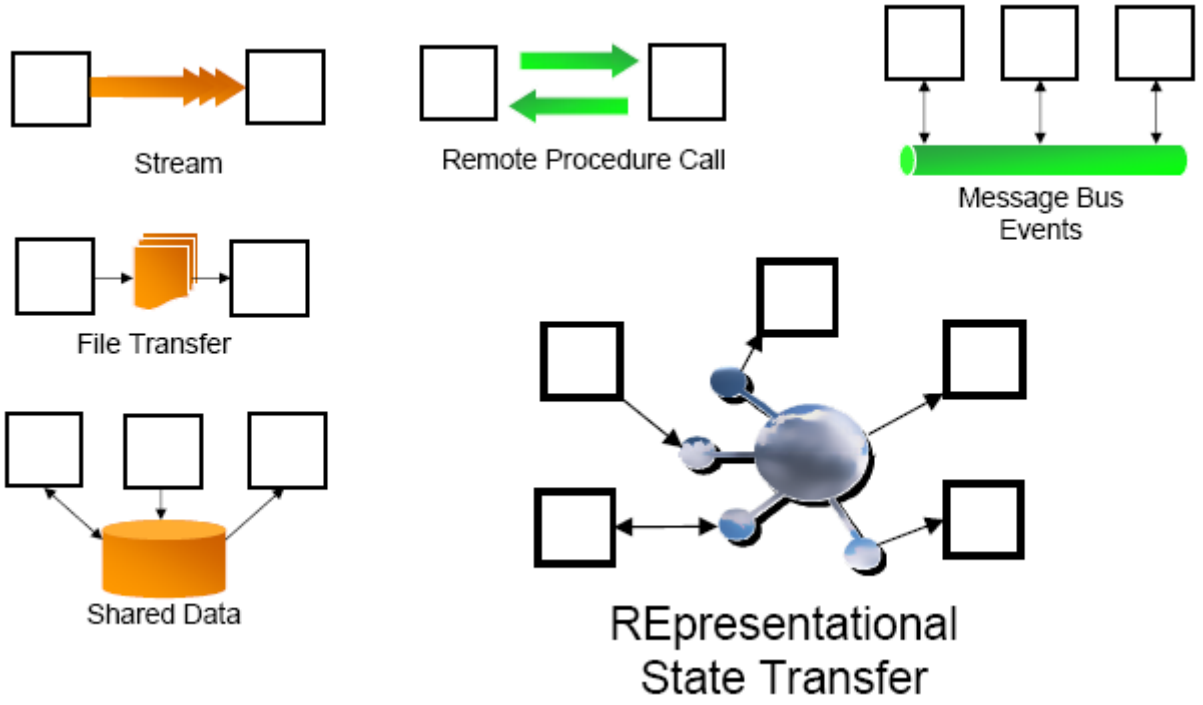


REST come connettore

- "The Web is the universe of globally accessible information" (Tim Berners Lee)
  - Applicazioni dovrebbero pubblicare i loro dati sul web (attraverso URI)



REST come connettore



Stateless o Stateful?

- REST fornisce transizioni di stato esplicite
  - Communicazioni sono stateless
  - Le risorse contengono dati e link che rappresentano transizioni di stato valide
  - Client mantengono lo stato in maniera corretta seguendo i link in una modalità generica

- Tecniche per aggiungere la sessione a HTTP
  - Cookie (HTTP Header)
  - URL Re-writing
  - Hidden Form Field

Service description

- REST si basa su documentazione human readable che definisce le URI delle richieste e le risposte (XML, JSON)
- Interagire con un servizio significa ore e ore di test e debug di URI costruite manualmente come combinazioni di parametri
  - È realmente più semplice costruire URI a mano?
- Perchè abbiamo bisogno di messaggi SOAP fortemente tipati se entrambi i lati sono già d'accordo sul contenuto?
- WADL (Web Application Description Language) proposto Novembre 2006
  - Definisce l'URI da visitare, i dati attesi, i dati ritornati in uscita
- XML Form sono abbastanza?

Service Deployment: Architettura a Microservizi (SOLA LETTURA)

History

- 2011: term coined in a software architecture conference close to Venice
- May 2012: microservice identified as the most proper term
- March 2012: “Microservices: Java, the Unix Way” at 33rd degree by James Lewis
- September 2012: “µService Architecture“ at Baruco (Barcelona Ruby Conference 2012) by Fred George
- Adrian Cockroft in the meantime becomes the pioneer of this style at Netflix, calling it “fine grained SOA”

http://martinfowler.com/articles/microservices.html#footnote-etymology

Principle

- At a logical level, microservice architecture means

  functional system decomposition into manageable and independently deployable components

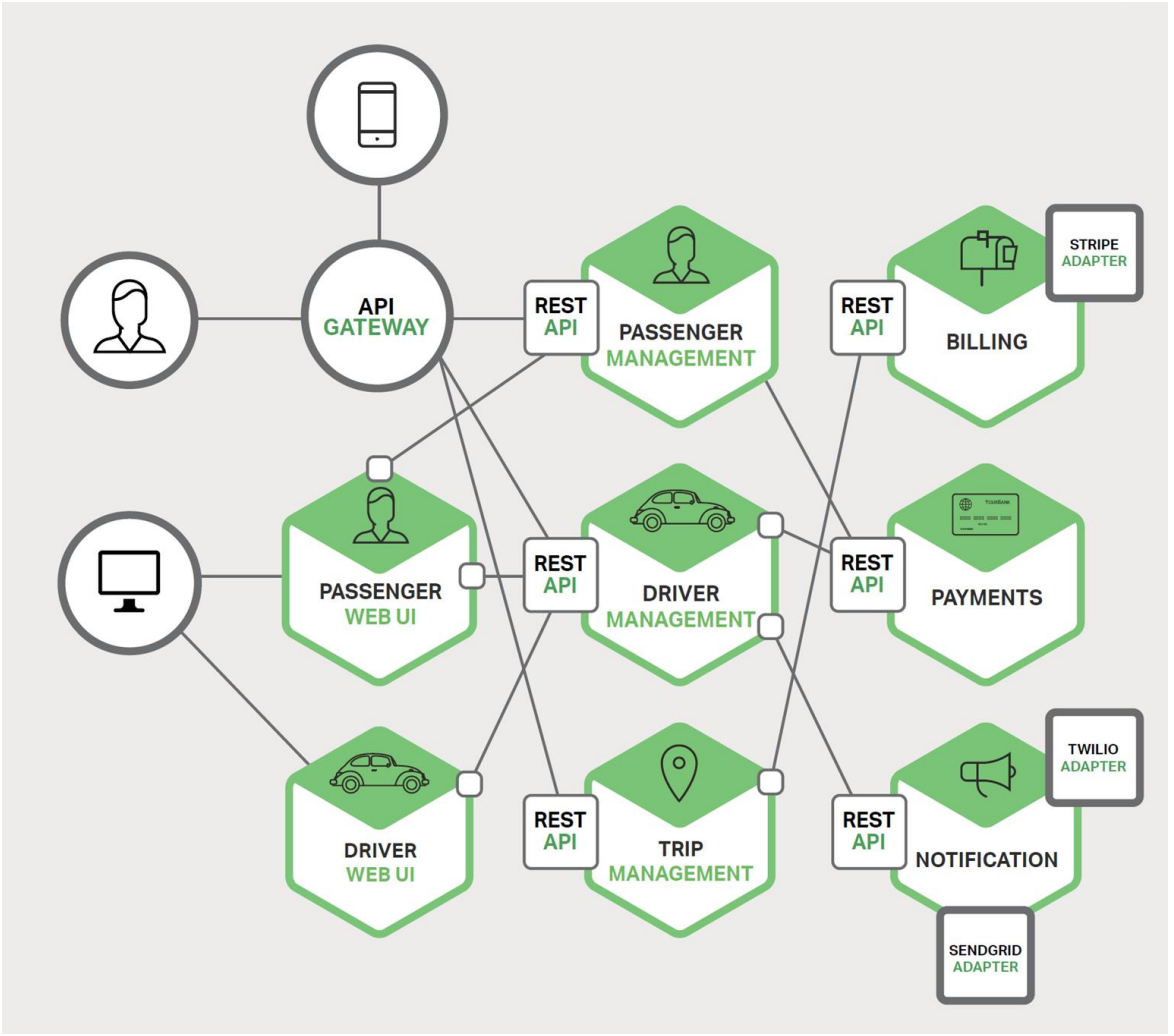
- The term “micro” is referred to the dimension
  - A microservice has to be manageable by a single development team
- Functional system decomposition means vertical decomposition (as opposed to the horizontal approach of a layered system)
- Independent deployment means no shared state or inter-process communication (often through HTTP interfaces REST-like)

Principle

- The idea is to divide the application into small interconnected services
- Each service typically implements a set of features or functionalities like order management, customer management, etc.
- Each microservice is a mini-application with its own hexagonal architecture consisting of business logic and several adaptors
  - Some microservices expose an API that can be used by other microservices, or by client applications
  - Other microservices implements a web interface
- At run time, each instance is, often, a virtual machine or a Docker container

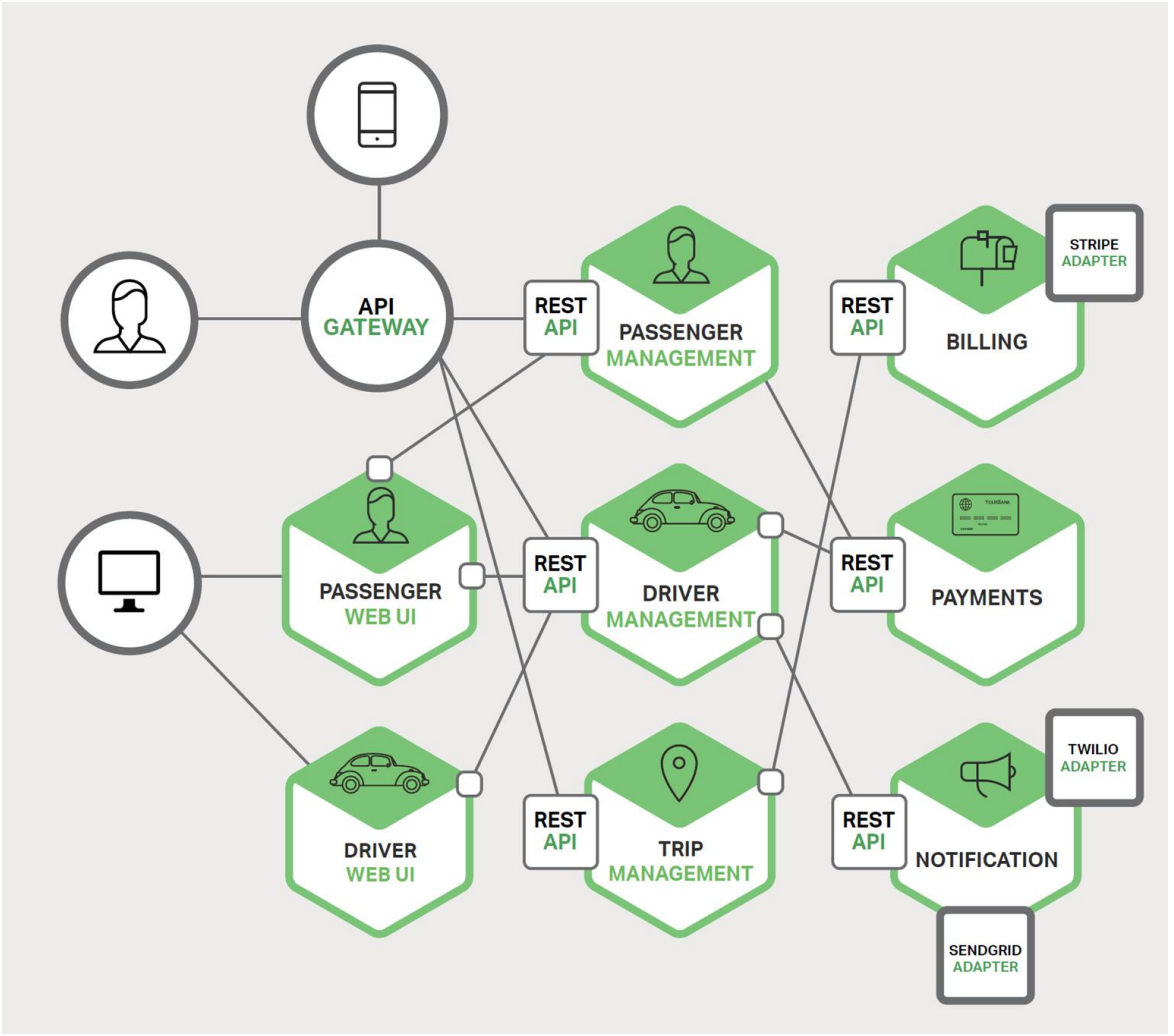
Microservice architecture: Example

- Each functional area is implemented with its own microservice
- The web application is divided into a set of simpler web applications
- It simplifies the deployment by distinguishing operations for specific users, or specialized use cases



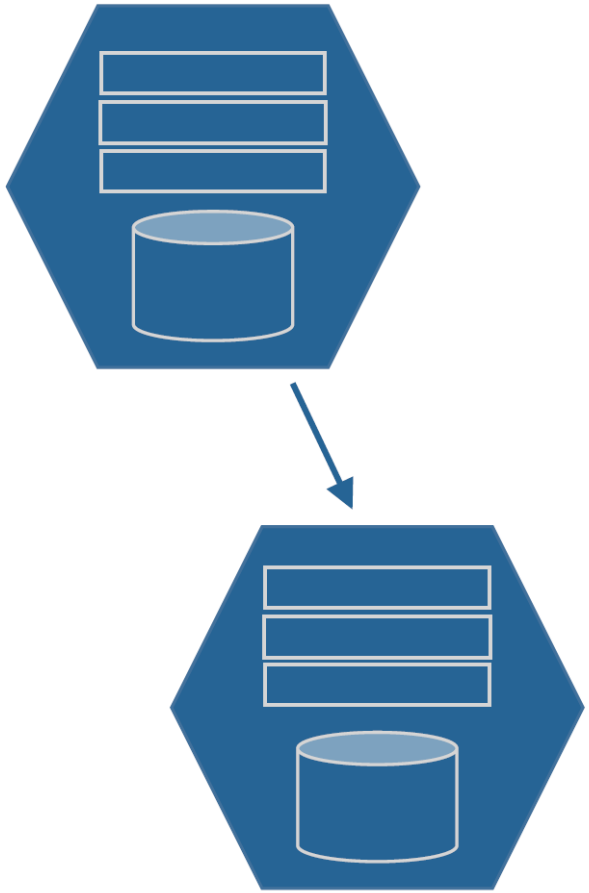
Microservice architecture: Example

- Each backend service exposes an API and several services use the API of other services
- Services can use asynchronous communication based on messages
- Communications are mediated by a known broker like an API Gateway
  - API Gateway is responsible for activities such as load balancing, caching, access control, API metering, and monitoring



More in detail

- Each microservice is functionally complete with
  - Resource representation
  - Data management
- Each microservice manages a resource
  - Client
  - Shop Item
  - Cart
  - Checkout

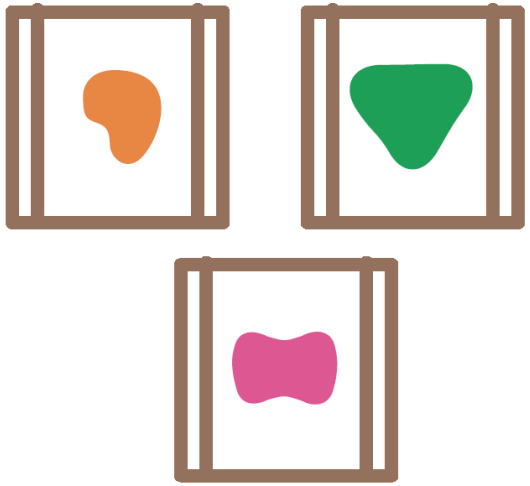


Microservices are called *fun-sized services*, because they are "*still fun to develop and deploy*"

Easy and fun

- A monolithic application has all the functionalities in a single process…

- A microservice architecture has each element/functionality in a separated service...



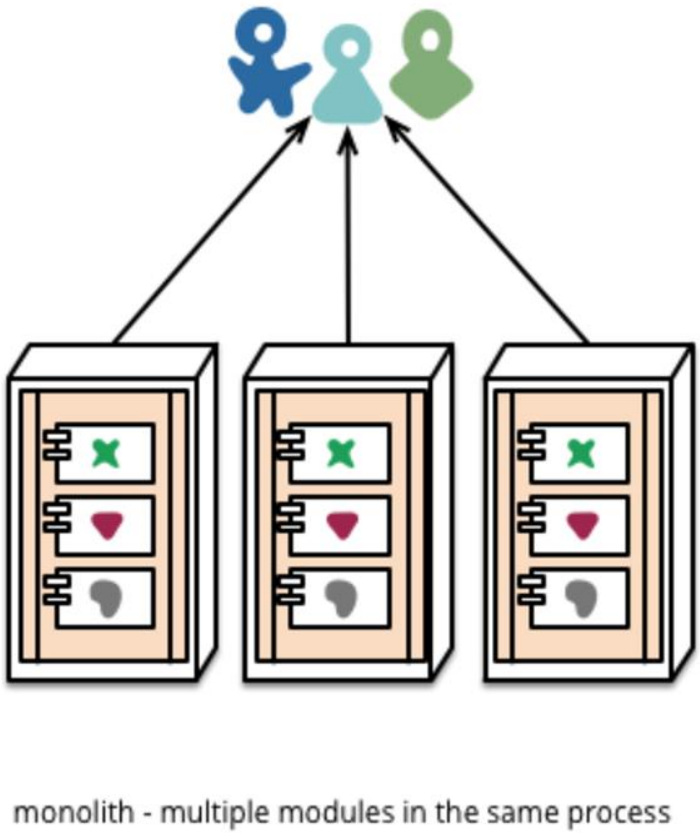
Independent deployment is fundamental

- Allows separation and independent evolution
  - Code base
  - Technological stack
  - Scaling
  - Functionalities

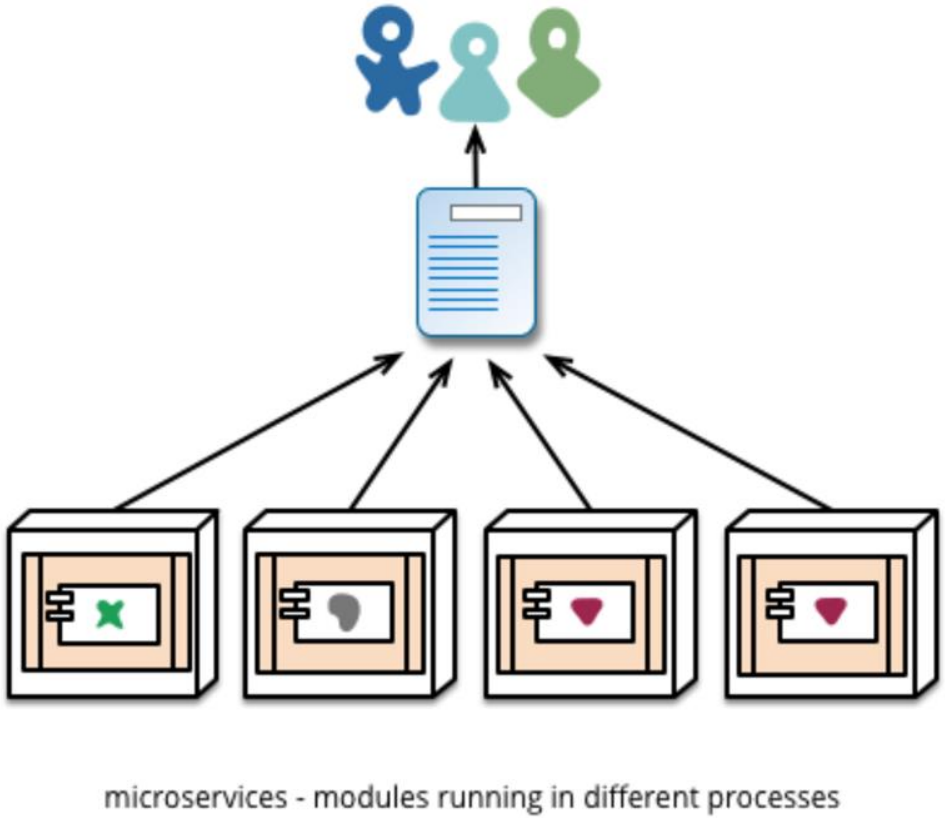
Independent code base

- Each service has its own software repository
  - The code is maintainable for the developers
    - *It fits into their brain*
  - Tools work faster – building, testing, refactoring the code take seconds
  - Service startup takes few seconds
  - There are no accidental cross-dependencies between different code bases

Independent Process



monolith - multiple modules in the same process



microservices - modules running in different processes

Conclusioni

- Service-Oriented Architecture possono essere implementate in diversi modi
- Scegliere l'architettura che più si adatta al lavoro da fare e riconoscere il valore degli open standard
  - Evitare di considerare ogni specifica tecnologia/architettura come LA tecnologia/architettura
- Il passo giusto è stato fatto nello sviluppo delle più recenti specifiche WS-* che rende questa visione realtà

Conclusioni

- La scelta tra SOAP e la famiglia di specifiche WS-*, e REST dipende dalle circostanze e dall'applicazione considerata

- REST rappresenta la soluzione più utilizzata per le applicazioni web 2.0
  - Si integra alla perfezione con il web
  - Ne sfrutta l'open standard e la scalabilità

Riferimenti

- Tesi di Roy Felding https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (Capitolo 5)

- Richardson, Ruby, RESTful Web Services, http://www.crummy.com/writing/RESTful-Web-Services/