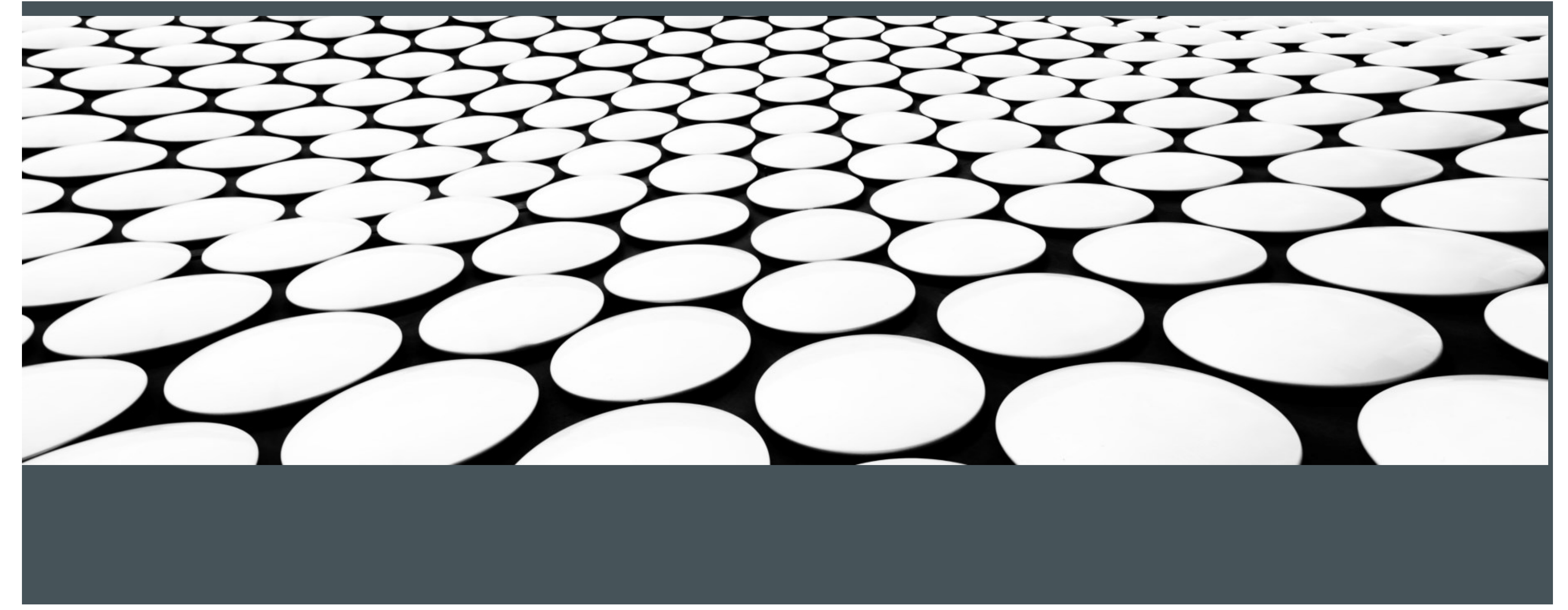
WEB SERVICES: JSON & REST

VINCENZO CALABRÒ



AGENDA

- Introduzione
- JSON
- REST
- Dettagli implementativi
- Approfondimenti

Introduzione

- REpresentational State Transfer (REST)
- Un’alternativa alle architetture software basate su SOAP
- Derivano dalla tesi di dottorato di Roy Fielding (2000)
- Gestiscono un tipo di contenuto variabile
  - Può essere XML
  - Principalmente JSON

JSON

Overview

- JSON o JavaScript Object Notation è un text-based open standard leggero disegnato per lo scambio di dati human-readable
- Convenzioni usate da JSON sono conosciute a programmatori C, C++, Java, Python, Perl etc.
  - Specificato da Douglas Crockford
  - Definito per scambio di dati human-readable
  - Esteso da JavaScript scripting language
  - Facile da analizzare e generare per le macchine
  - Estensione dei file .json
  - JSON Internet Media type è application/json

Utilizzo di JSON

- Usato quando si scrivono applicazioni basate su JavaScript (include estensioni al browser e siti web)
- Usato per serializzare e trasmettere dati strutturati su una connessione di rete
- Primariamente usata per trasmettere dati tra server e applicazioni web
- Web Service e API usano JSON per fornire dati pubblici
- Può essere usato con linguaggi moderni

Caratteristiche di JSON

- Facile da leggere e scrivere
- Formato di scambio basato su testo e molto leggero
- Indipendente dal linguaggio

Sintassi

- La sintassi JSON è considerate come un sottoinsieme della sintassi JavaScript
  - Dati rappresentati come coppie nome/valore
  - Parentesi graffe contengono oggetti e ogni nome è seguito dal : (colon), le coppie nome/valore sono separate da , (comma)
  - Parentesi quadre definiscono array e sono separate da, (comma)

Esempio

```
{
  "book": [
  {
    "id":"01",
    "language": "Java",
    "edition": "third",
    "author": "Herbert Schildt"
  },
  {
    "id":"07",
    "language": "C++",
    "edition": "second"
    "author": "E.Balagurusamy"
  }]
}
```

Sintassi

- JSON supporta due strutture dati
  - Insieme di coppie nome/valore
    - Struttura dati supportata da diversi linguaggi di programmazione
  - Lista ordinate di valori
    - Array, liste, vettori, sequenze…

Datatype

Tipo	Descrizione
Number	Formato Javascript floating-point double-precision
String	Unicode double-quoted con backslash escaping
Boolean	True o false
Array	Una sequenza ordinata di valori
Value	String, number, true o false, null…
Object	Una collezione non ordinate di coppie nome:valore
Whitespace	Usato tra ogni coppia di token
null	Vuoto

Number

- È floating-point double-precision come in JavaScript e dipende dall'implementazione
- Formati ottale ed esadecimale non usati
- Nessun supporto per NaN o Infinity in Number

Tipo	Descrizione
Integer	Digits 1-9, 0 e positive o negative
Fraction	Frazioni: .3, .9
Exponent	Esponenti: e, e+, e-,E, E+, E-

- Sintassi
  - var json-object-name = { string : number_value, ...}
- Esempio
  - var obj = {"marks": 97}

String

- Sequenza di zero o più caratteri Unicode double quoted con backslash escaping
- Character è una stringa di lunghezza 1

Tipo	Descrizione
"	double quotation
\	reverse solidus
/	solidus
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

String

- Sintassi
  - var json-object-name = { string : "string value", ...}

- Esempio
  - var obj = {"name": "Pippo"}

Boolean

- True o false

- Sintassi
  - var json-object-name = { string : true/false, ...}

- Esempio
  - var obj = {"name": "Pippo", "marks": 97, "distinction": true}

Array

- Collezione ordinate di valori
- Inclusi in parentesi quadre
  - Iniziano con '[' e finiscono con ']'
- Valori separati da , (comma)
- Indice dell’array parte da 0 o 1
- Array dovrebbero essere usati quando i nomi delle chiavi sono interi sequenziali

Array

- Sintassi
  - [value, ...]

- Example:
  - {

```
        "books": [
         { "language":"Java" , "edition":"second" },
         { "language":"C++" , “edition":"fifth" },
         { "language":"C" , “edition":"third" }
        ]
  }
```

Object

- Insieme non ordinate di coppie nome/valore
- Object sono racchiusi tra parentesi graffe
  - Iniziano con '{' e finiscono con '}'
- Ogni nome è seguito da : (colon) e coppie nome/valore sono separate da , (comma)
- Le chiavi devono essere stringhe e dovrebbero essere diverse tra loro
- Object dovrebbero essere usati quando le chiavi sono stringhe arbitrarie

Object

- Sintassi
  - { string : value, ...}

- Esempio
  - {

```
"id": "011A",
"language": "JAVA",
"price": 500,
}
```

Whitespace

- Può essere inserito tra ogni coppia di token
  - Utilizzato per rendere tutto più leggibile

- Sintassi
  - {string:"   ",....}

- Esempio
  - var i= "   sachin";
  - var j = " saurav"

NULL

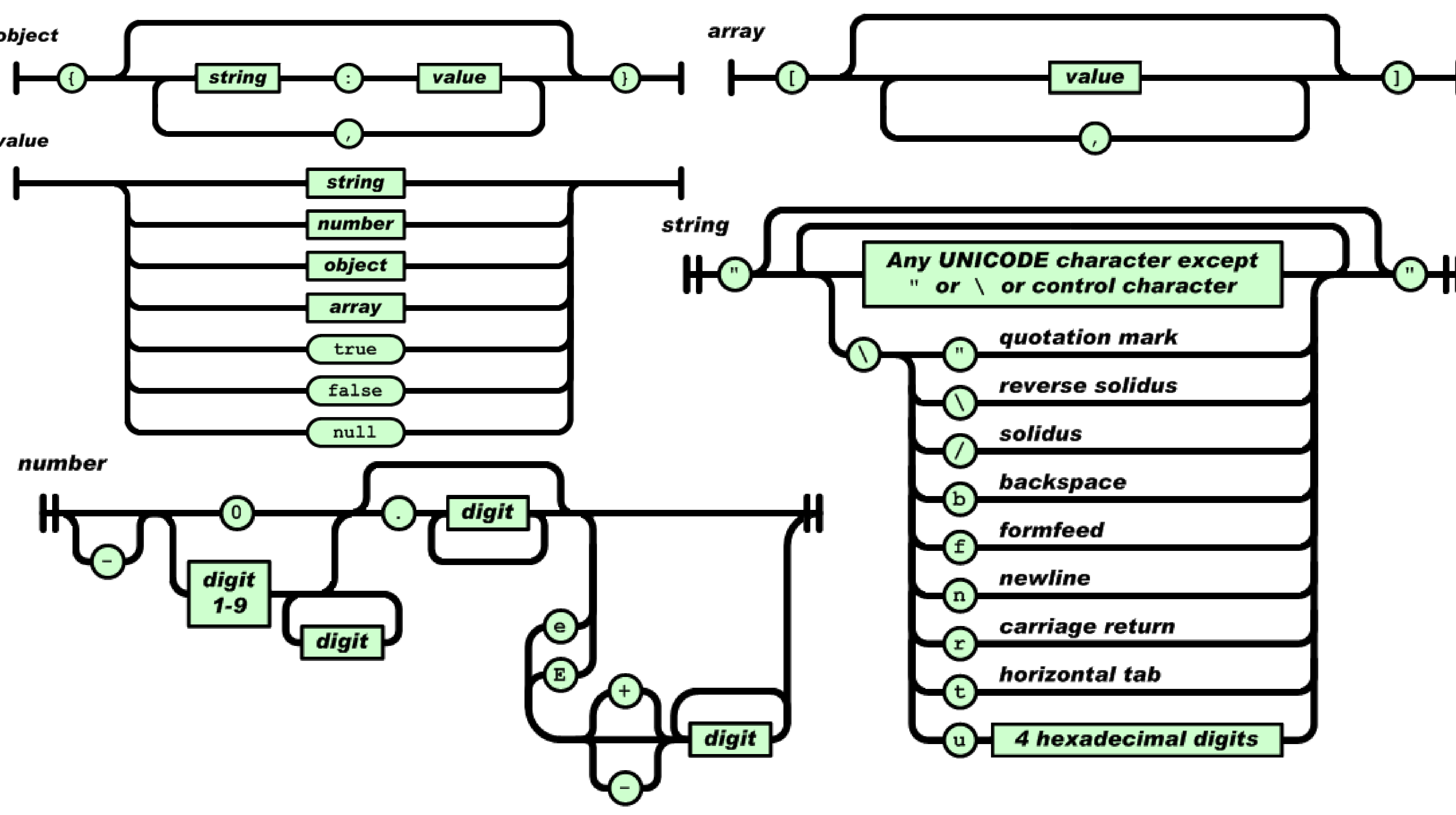
- Tipo vuoti
- Sintassi
  - null
- Esempio
  - var i = null;

```
if(i==1) {
    document.write("<h1>value is 1</h1>");
}
else {
    document.write("<h1>value is null</h1>");
}
```

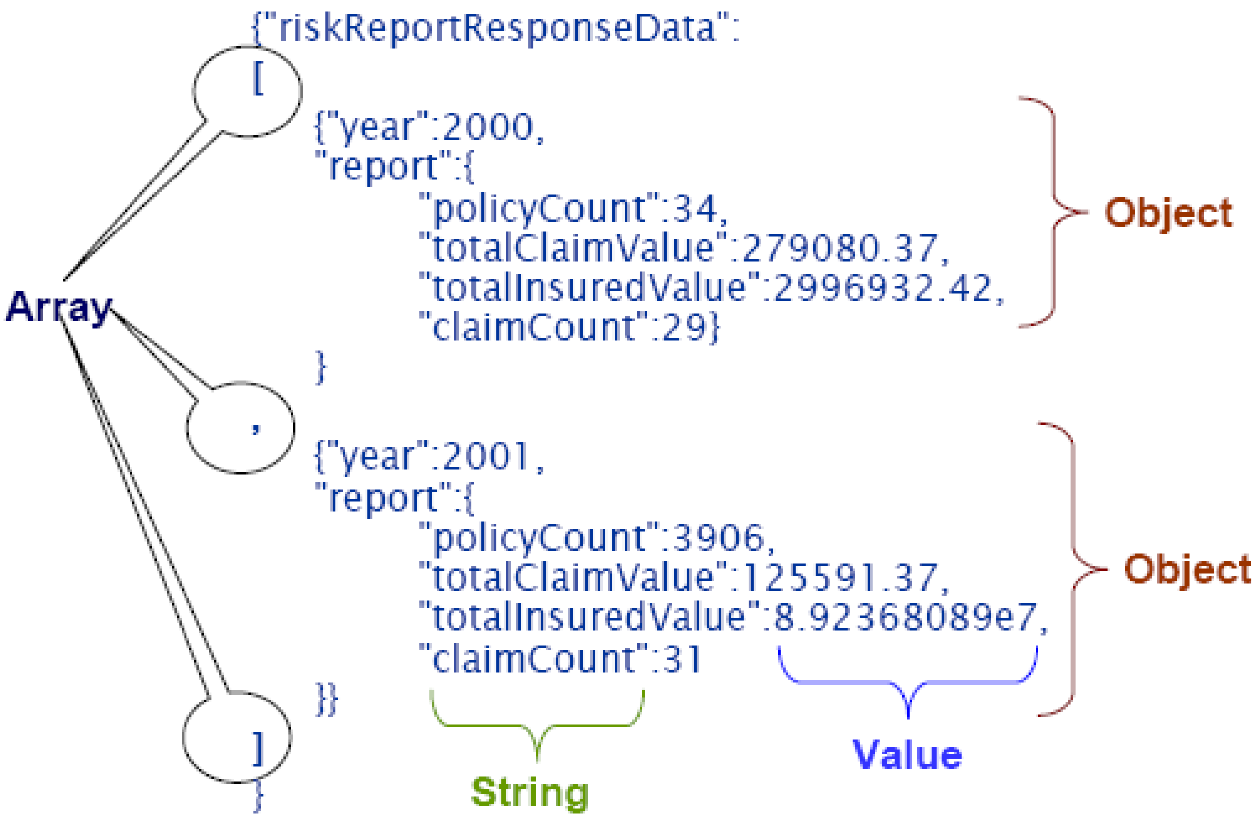
Value

- Include
  - number (integer o floating point)
  - string
  - boolean
  - array
  - object
  - null
- Sintassi
  - String | Number | Object | Array | TRUE | FALSE | NULL
- Esempio
  - var i =1;
  - var j = "sachin";
  - var k = null;

Strutture dati



Esempio JSON



JSON Schema

- JSON Schema è una specifica per la definizione dei dati JSON

- Scritta in un IETF draft scaduto nel 2011
  - Descrive il formato dei dati
  - Documentazione semplice, human-readable e machine-readable
  - Validazione strutturale completa
    - Utile per testing automatico
    - Valida dati sottomessi dal client

JSON Schema: Librerie di validazione

- Diversi validatori per diversi linguaggi di programmazione
- JVS il validatore più completo e consistente

Languages	Libraries
C	WJElement (LGPLv3)
Java	json-schema-validator (LGPLv3)
.NET	Json.NET (MIT)
ActionScript 3	Frigga (MIT)
Haskell	aeson-schema (MIT)
Python	Jsonschema
Ruby	autoparse (ASL 2.0); ruby-jsonschema (MIT)
PHP	php-json-schema (MIT). json-schema (Berkeley)
JavaScript	Orderly (BSD); JSV; json-schema; Matic (MIT); Dojo; Persevere (modified BSD or AFL 2.0); schema.js.

JSON Schema: Esempio

- Descrizione catalogo prodotti
  - ```
    {
        "$schema": "http://json-schema.org/draft-04/schema#",
        "title": "Product",
        "description": "A product from Acme's catalog",
        "type": "object",
        "properties": {
            "id": {
                "description": "The unique identifier for a product",
                "type": "integer"
            },
            "name": {
                "description": "Name of the product",
                "type": "string"
            },
            "price": {
                "type": "number",
                "minimum": 0,
                "exclusiveMinimum": true
            }
        },
        "required": ["id", "name", "price"]
    }
    ```
    ```

JSON Schema: Esempio

Keywords	Description
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to give a title to your schema
description	A little description of the schema
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	This is the constraint to be put on the value and represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The length of a string instance is defined as the maximum number of its characters.
minLength	The length of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

Istanza dello schema: Esempio

```
[
  {
    "id": 2,
    "name": "An ice sculpture",
    "price": 12.50,
  },
  {
    "id": 3,
    "name": "A blue mouse",
    "price": 25.50,
  }
]
```

JSON vs XML

- JSON e XML sono formati human readable e indipendenti dal linguaggio

- Possibile confrontare JSON e XML sulla base di tre fattori principali
  - Verbosità
  - Utilizzo degli array
  - Parsing

JSON vs XML

- Verbose
  - XML è più verboso di JSON
  - Più veloce scrivere JSON per esseri umani

- Utilizzo degli array
  - XML è usato per descrivere dati strutturati che non includono array
  - JSON include array

- Parsing
  - Il metodo eval di JavaScript fa il parsing di JSON
  - Quando applicato a JSON, eval ritorna l'oggetto descritto

Formato di rappresentazione delle risorse: XML vs. JSON

- XML
  - Plain Old XML (PO-XML): XML base
  - SOAP (WS-*)
  - RSS, ATOM
- Sintassi testuale standard per dati semi-strutturati
- Molti tool disponibili
  - XML Schema, DOM, SAX, XPath, XSLT, XQuery
- Chiunque può parsare XML
- Lento e verboso

Formato di rappresentazione delle risorse: XML vs. JSON

- JSON (JavaScript Object Notation)
  - Formato introdotto per applicazioni Web AJAX (comunicazioni browser-web server)
  - Sintassi testuale per la serializzazione di strutture dati non ricorrrenti
  - Supportato da molti linguaggi (non solo JavaScript)
  - Non estendibile
  - “JSON has become the X in AJAX”

JSON vs XML: Esempio

- JSON

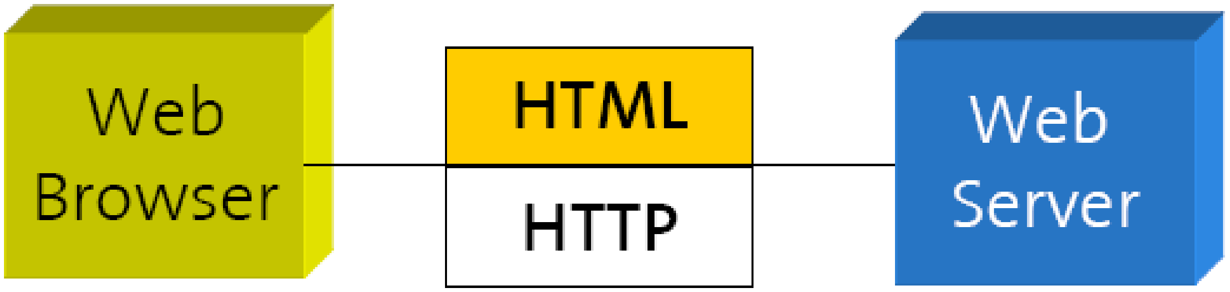
```
{
  "company": Volkswagen,
  "name": "Vento",
  "price": 800000
}
```

- XML

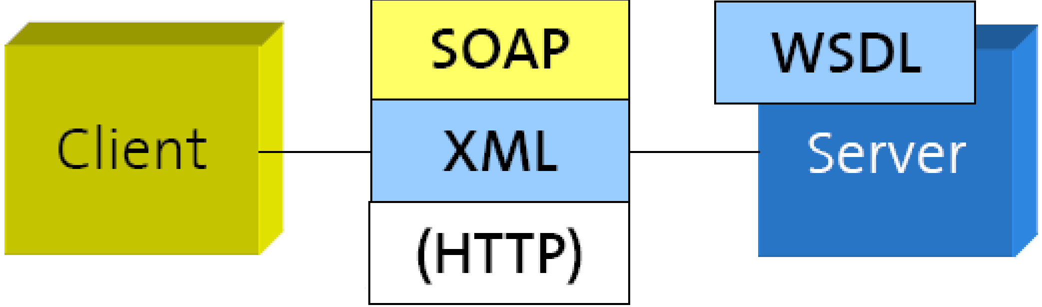
```
<car>
  <company>Volkswagen</company>
  <name>Vento</name>
  <price>800000</price>
</car>
```

REST

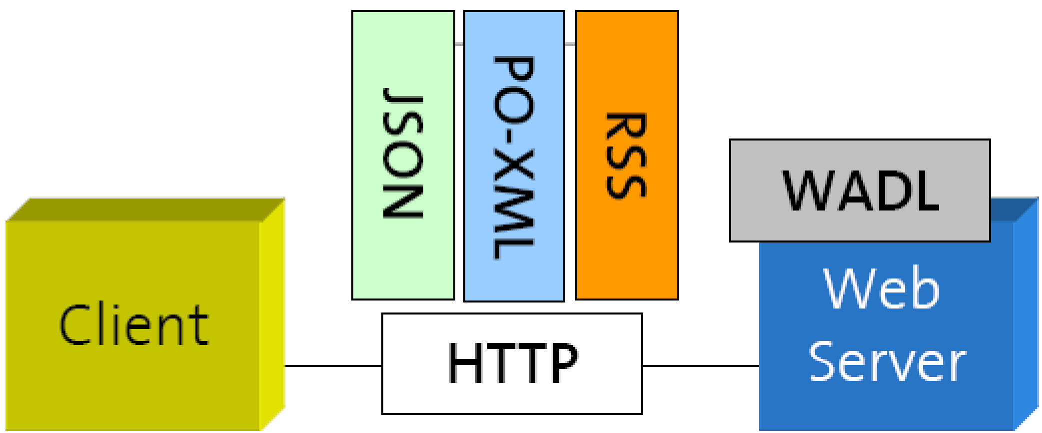
Siti web (1992)



WS-* Web Service (2000)



RESTful Web Service (2007)



Perché web service?

- Risolvono il problema dell’enterprise software standardization
- Enterprise Computing Standards for Interoperability (WS inizia nel 2001)
- Un’architettura a livelli con una varietà di specifiche di messaging, description e discovery
- Produce oggetti dal basso, velocemente, usando specifiche ben fattorizzate e distinte
- Tool nascondono la complessità

Big web service

- Grande complessità (SOAP)

- Processo di standardizzazione problematico
  - Infighting
  - Lack of architectural coherence
  - Fragmentation
  - Design by committee
  - Feature Bloat (Merge of competing specs)
  - Lack of reference implementations
  - Standardization of standards (WS-I)

Big web service

- Quando l'interoperabilità tra web service sarà veramente supportata?
- Dobbiamo davvero comprare dispositivi e servizi basati su XML per ottenere buone performance?

WS-PageCount	
Messaging	232 pages
Metadata	111 pages
Security	230 pages
WS-BPEL	195 pages
XML/XSD	599 pages
Transactions	39 pages

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

Design architetturale

- 7 fasi con vincoli incrementali
  - Null style
  - Client-server
  - Stateless
  - Cache
  - Uniform interface
  - Layered system
  - Code-on-demand

- Maggiori dettagli nel Capitolo 5 della tesi di Roy Felding https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

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

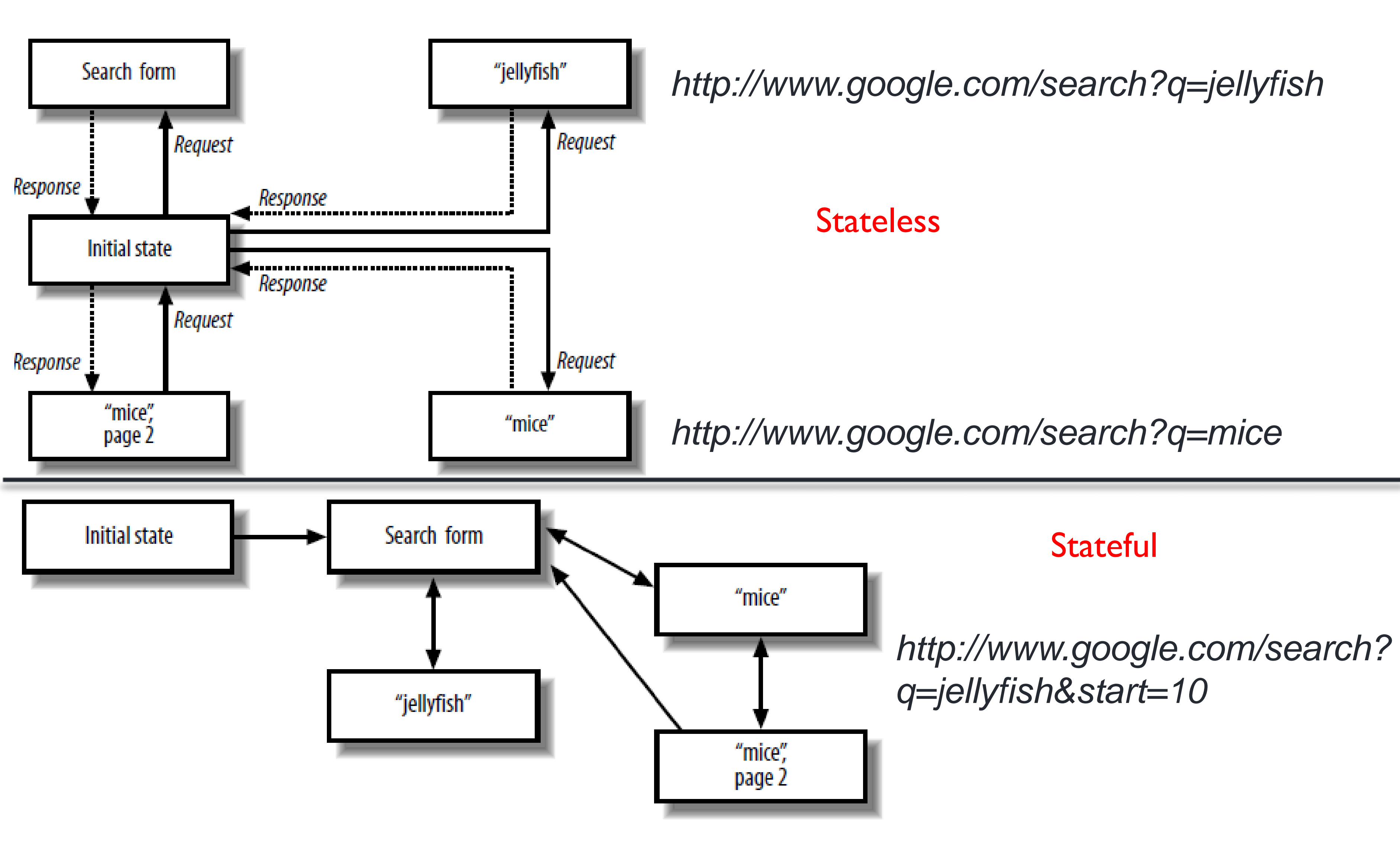
Addressability

- 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



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
  - Document, 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

Data element

- Connector
  - Interfaccia astratta per la comunicazione tra componenti
  - Incapsula le attività per accedere a risorse e trasferire resource representation
  - Comunicazione stateless, richiesta contiene tutte le informazioni per capirne il contenuto

Connector	Modern Web Examples
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI
cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

Data element

- La gestione dei connector è simile a quella delle invocazioni a procedure

- Differenze sono nel passaggio di parametri e nei risultati
  - Input: request control data, resource identifier, optional representation
  - Output: response control data, optional resource metadata, optional representation
  - Processing può essere invocato prima che il dato sia stato interamente ricevuto

- Connector type: client, server, cache, resolver, tunnel

Data element

Component	Modern Web Examples
origin server	Apache httpd, Microsoft IIS
gateway	Squid, CGI, Reverse Proxy
proxy	CERN Proxy, Netscape Proxy, Gauntlet
user agent	Netscape Navigator, Lynx, MOMspider

Data element

- User agent: colui che usa un client connector per iniziare una richiesta
- Origin server: colui che usa un server connector per gestire i namespace di una determinata risorsa
- Intermediary che agiscono sia come client che server
  - Proxy: servizio selezionato dal client che fornisce incapsulamento di interfacce di altri servizi, data translation, miglioramento di performance, protezione di sicurezza
  - Gateway: imposto dalla rete o dal server, fornisce incapsulamento di interfacce di altri servizi, data translation, miglioramento di performance, protezione di sicurezza
  - Il client sceglie se usare o meno il proxy

Visione architetturale

- Process view
- Connector view
- Data view

Visione architetturale: Process view

- Mostra le relazioni tra componenti e verifica il percorso preso dai dati quando fluiscono attraverso il sistema

Proxy
Gateway
Origin Server
http
http
http
orb
a
User Agent
DNS
b
http
c
Proxy
DNS
http
wais

Client Connector: Client+Cache: Server Connector: Server+Cache:

Visione architetturale: connector view

- Si concentra sui meccanismi di comunicazione tra componenti

- REST supporta diversi protocolli
  - HTTP default, FTP, Gopher, WAIS

- REST limita le interfacce tra protocolli
  - Riduce i vantaggi degli altri protocolli, ma mantiene un’interfaccia singola e generica per la semantica dei connettori

Visione architetturale: data view

- Rivela lo stato dell'applicazione analizzando come l'informazione fluisce attraverso i componenti

- REST vede un'applicazione come una struttura coesa di informazioni e alternative di controllo attraverso cui l'utente esegue un task
  - Ad esempio, ricerca in un vocabolario, visita a un museo virtuale
  - L'applicazione impone requisiti sul sistema sottostante attraverso cui vengono misurate le performance

Visione architetturale: data view

- Interazione tra componenti avviene con scambio di messaggi
  - Messaggi di controllo (piccoli/medi)
    - GET il messaggio più usato
  - Messaggi contenenti la rappresentazione della risorsa (grandi)

- Lo stato dell'applicazione memorizzato nella risposta
  - Nessun bisogno di mantenere lo stato per più di una richiesta
  - Stato dell'applicazione definito da: richieste pendenti, topologia dei componenti connessi, le richieste attive dei componenti, il data flow in risposta alle richieste, processing delle rappresentazioni quando ricevute dallo user agent

Dettagli implementativi

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

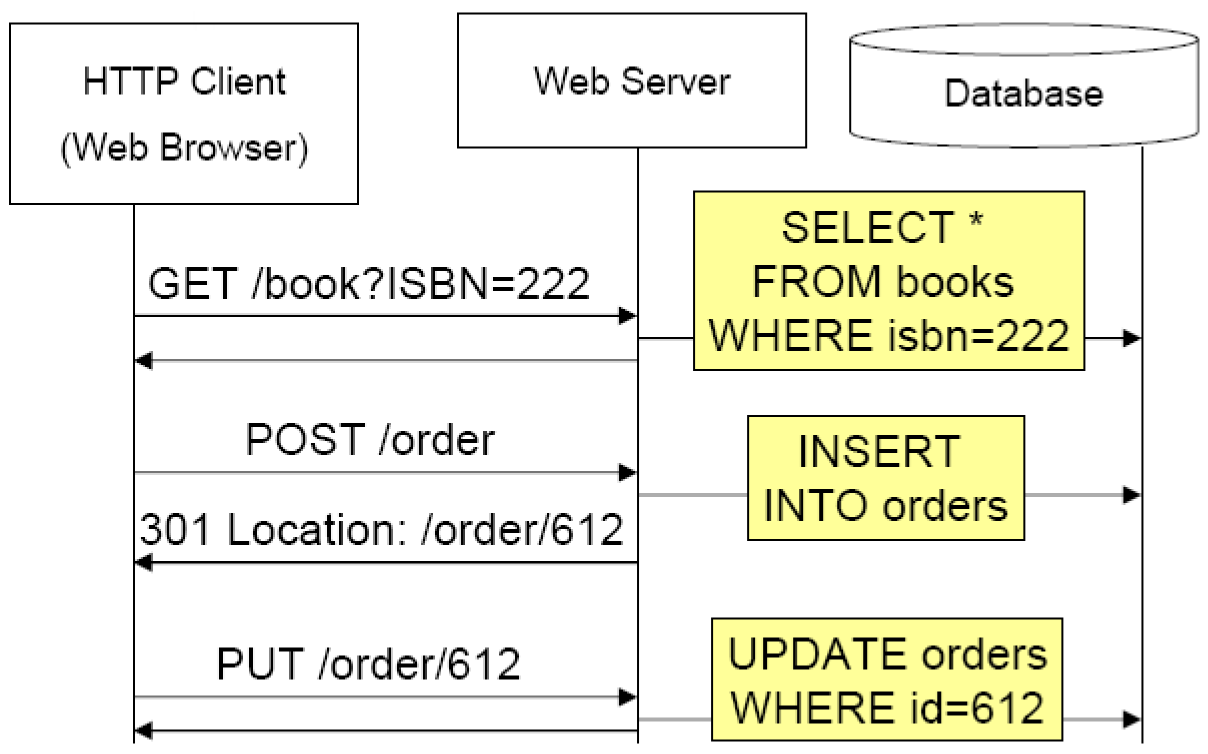
- Metodi HTTP aggiuntivi della uniform interface
  - HEAD: riceve solo i medati di una risorsa
  - Options: controlla quali metodi sono supportati dalla risorsa
    - Ritorna un HTTP allow
    - Allow: GET, HEAD (risorsa in sola lettura)

Safety e idempotenza

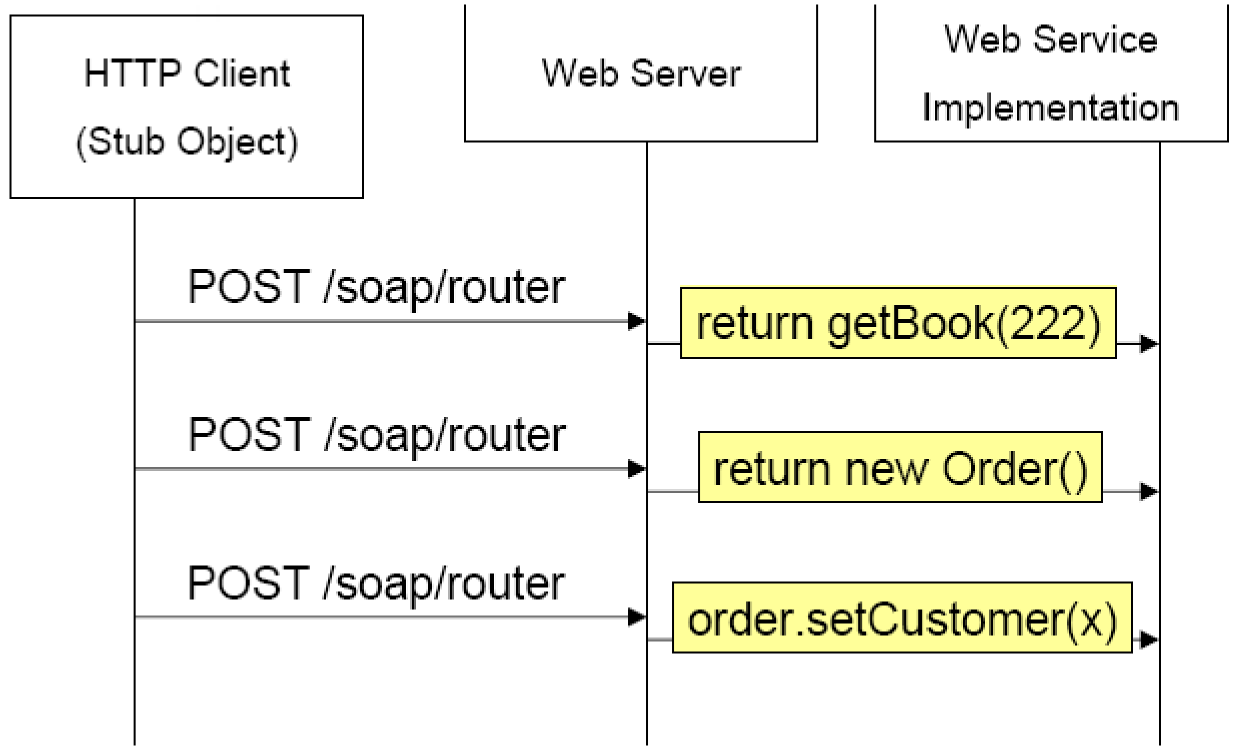
- Safety
  - GET, HEAD
  - Richieste safe, solo lettura, non causano danni catastrofici
  - Side effect: incremento un contatore, loggo la richiesta

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

RESTful Web Service: Esempio



Web Service: Esempio (da una prospettiva REST)



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

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

- Preferire sostantivi a verbi
- Mantenere le URI corte
- 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 se c'è bisogno di cambiare gli URI

High REST vs. Low REST

- High REST
  - Utilizzo di “nice” URI viene raccomandato
  - Utilizzo di quattro verbi: GET, POST, PUT, e DELETE
  - Risposte codificate come Plain Old XML

- Low REST
  - HTTP GET per richieste idempotenti, POST per tutte le altre
  - Risposte in qualsiasi MIME Type (ad esempio, XHTML)

REST: Vantaggi

- Semplicità
  - Interfaccia uniforma immutabile (no problem of breaking client)
- HTTP/POX è ubiquo (attraversa i firewall)
- Interazione Stateless/Sincrona
- Scalabilità
  - "After all the Web works", caching, clustered server farm per QoS

REST: Vantaggi

- Facilità di utilizzo e adozione (infrastruttura leggera)
  - Solo un browser per iniziare, nessun bisogno di comprare un middleware WS-*
- Utilizzato da tutte le applicazioni web 2.0
  - 85% dei client preferiscono Amazon RESTful API
  - Google non supporta più SOAP/WSDL API

REST: Svantaggi

- Confusione (high REST vs. low REST)
  - È davvero composto di soli 4 verbi? (HTTP 1.1. ha 8 verbi: HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, e CONNECT)

- Il mapping della semantica sincrona del REST-style al di sopra dei sistemi di backend esistenti crea disallineamenti nel design (quando sono basati su messaging asincrono o interazioni guidate dagli eventi)

- Non può rilasciare enterprise-style “-ilities” oltre HTTP/SSL

REST: Svantaggi

- Difficile identificare e localizzare le risorse in maniera appropriata in tutte le applicazioni
- Apparente mancanza di standard (oltre che URI, HTTP, XML, MIME, HTML)
- Descrizione semantica/sintattica molto informale (orientata all'utente/uomo)

RESTful Web Service: Design Methodology

- Identificare risorse che possono essere esposte come servizi (ad es., yearly risk report, book catalog, purchase order, open bug, polls and votes)
- Definire “nice” URL per indirizzare i servizi
- Capire il significato di GET, POST, PUT, DELETE su una data URI di una risorsa
- Definire e documentare la rappresentazione delle risorse

RESTful Web Services: Design Methodology

- Modelli di relazione (ad es., contenimento, riferimento, transizione di stato) tra risorse con hyperlink che possono essere seguite per ottenere più dettagli (o effettuare transizioni di stato)

- Implementare e deployare un Web server

- Testare i servizi

Gestione delle eccezioni

Learn to use HTTP Standard Status Codes

```
100 Continue
200 OK
201 Created
202 Accepted
203 Non-Authoritative
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
```

4xx Client's fault

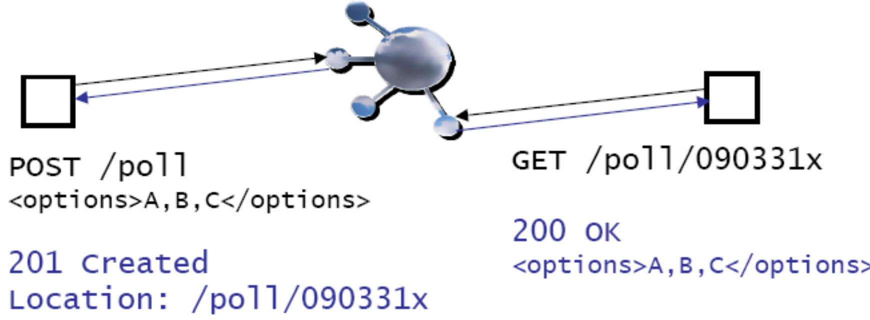
```
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Request Entity Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
```

5xx Server's fault

```
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
```

Semplice API per Doodle: Esempio

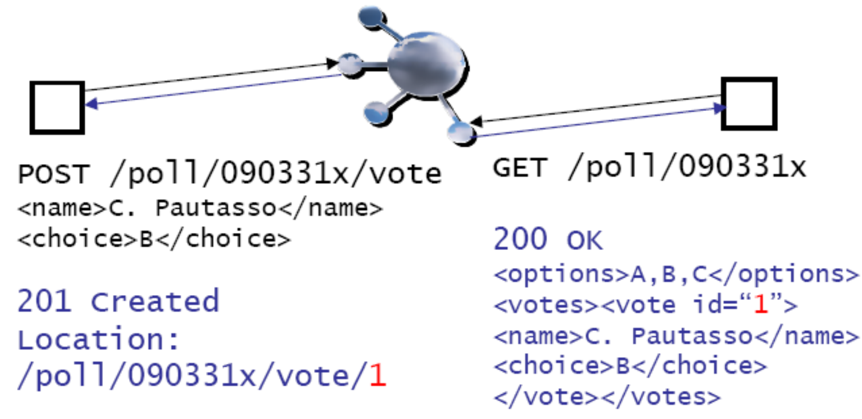
- Creare un sondaggio (trasferire lo stato di un nuovo sondaggio al servizio Doodle)



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

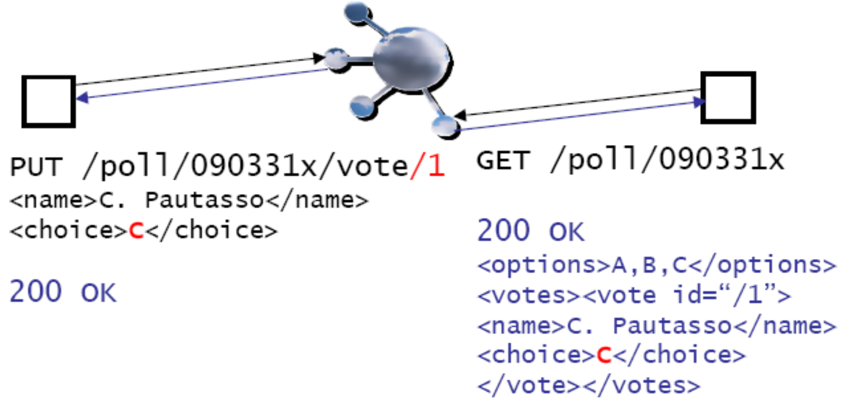
Semplice API per Doodle: Esempio

- Partecipare al sondaggio creando una sub-resource voto



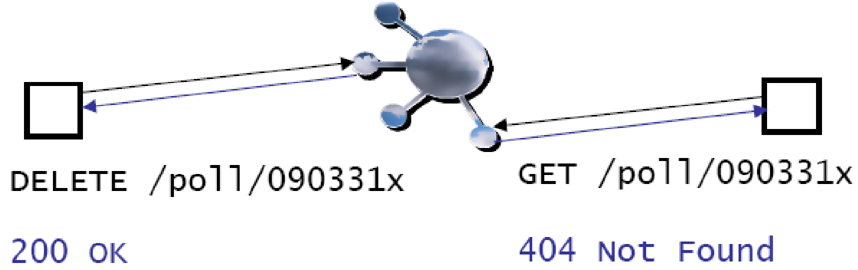
Semplice API per Doodle: Esempio

- 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
```

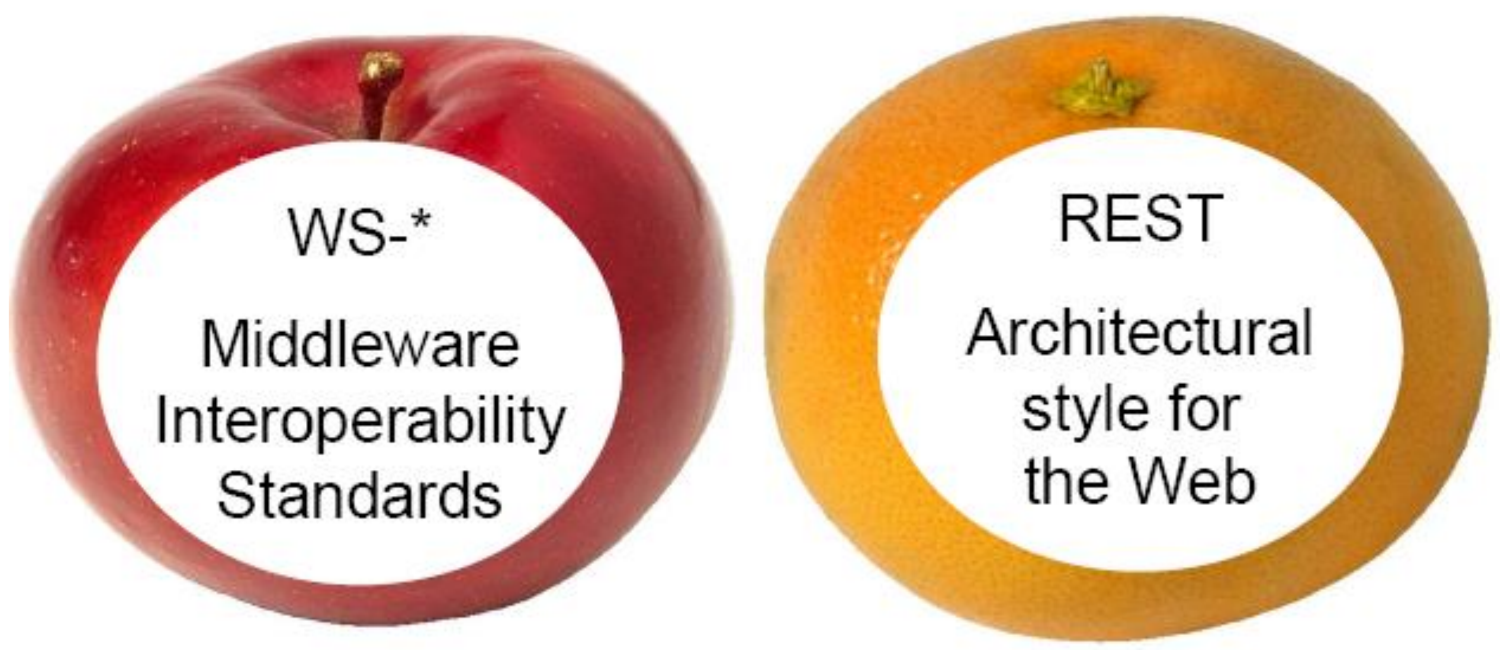
- Richieste unsafe richiedono gestione delle situazioni eccezionali (ad es., state reconciliation)

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

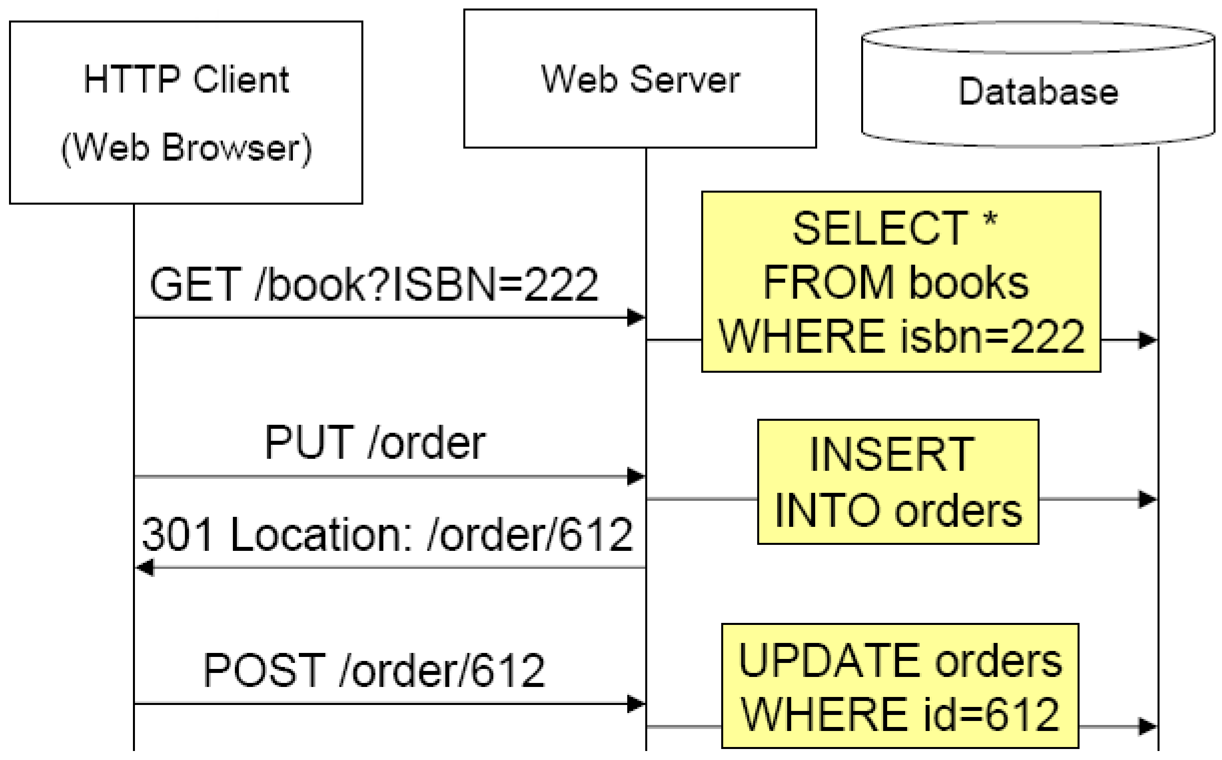
- In alcuni casi le API possono essere ridefinite per usare operazioni idempotenti

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

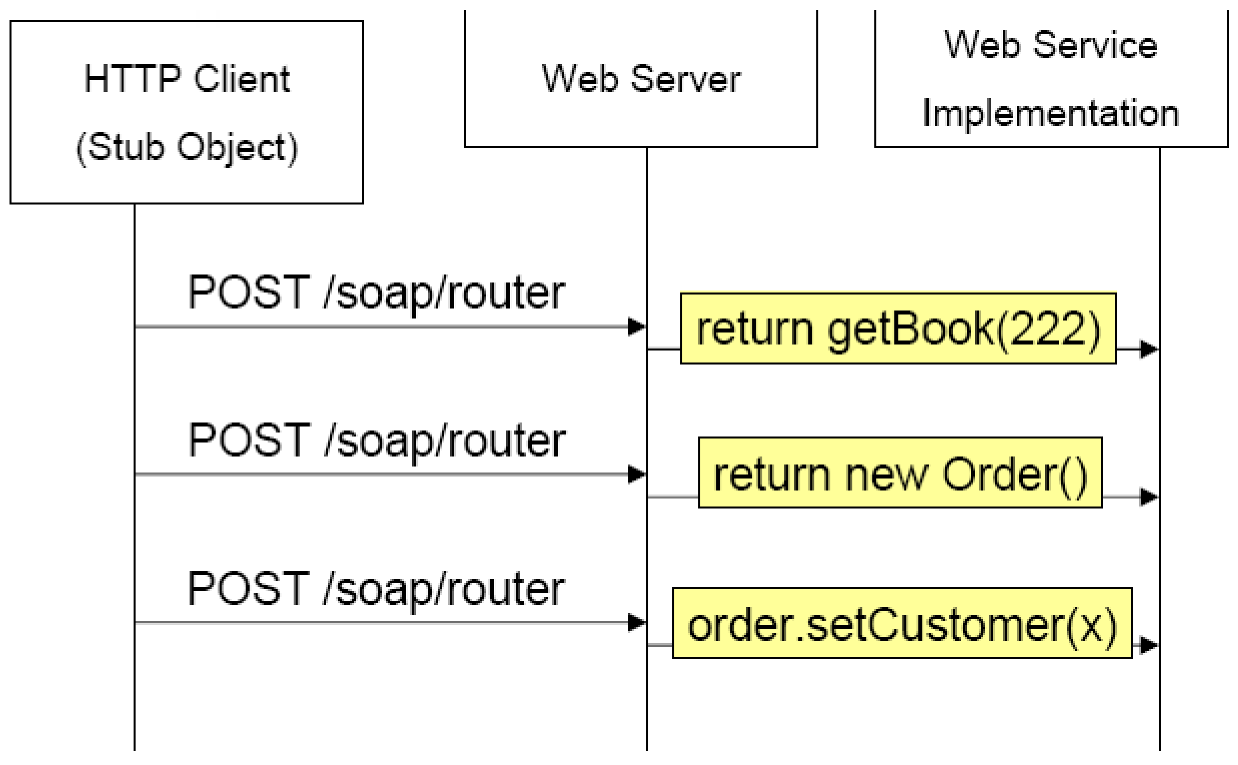
WS-* vs REST



RESTful Web Application: Esempio



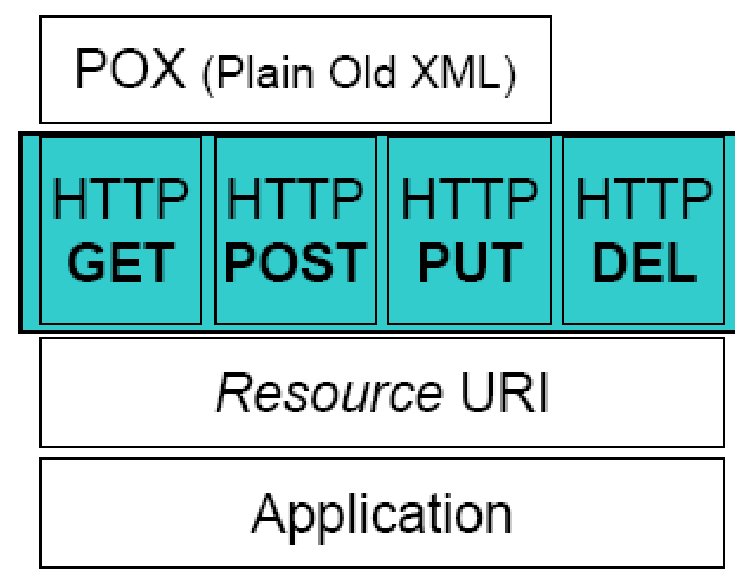
Web Service Example: da una prospettiva REST



REST vs SOAP

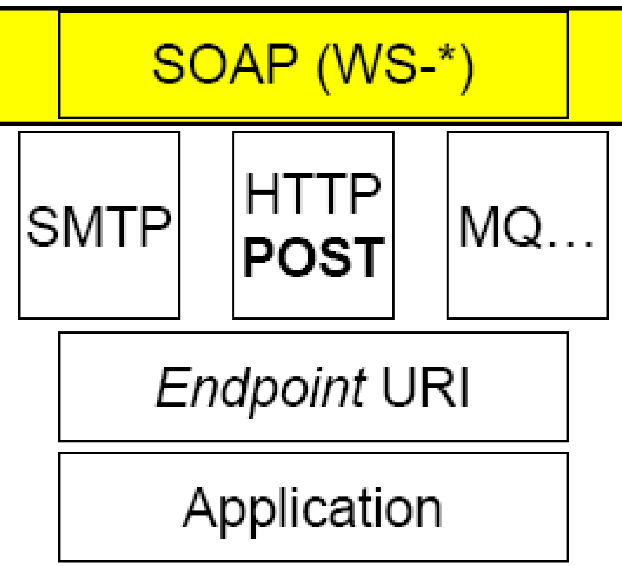
- “The Web is the universe of globally accessible information” (Tim Berners Lee)

  - Applicazioni dovrebbero pubblicare i loro dati sul web (attraverso URI)

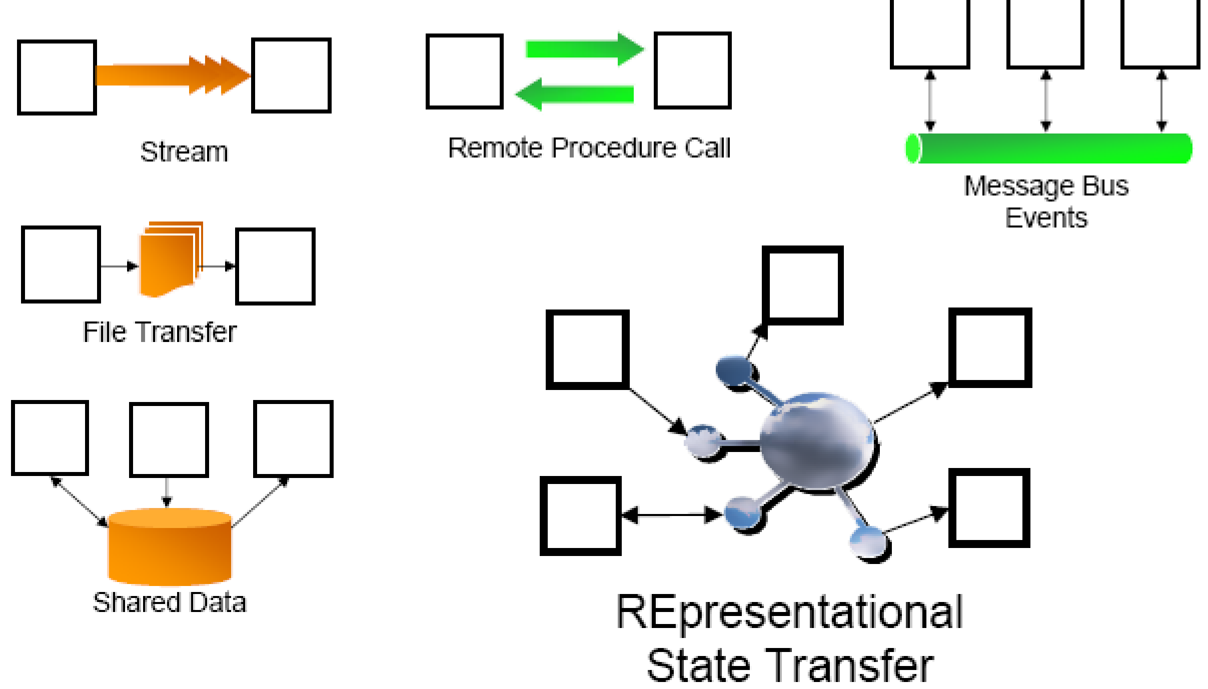


REST vs SOAP

- “The Web is the universal transport for messages”
  - Applicazioni hanno un’occasione per interagire ma rimangono “outside of the Web”



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 descripion

- 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?

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/

QUESITI?

vincenzocalabro.it