<?XML?> <SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

Web Services
e
REST
REpresentational State Transfer

2018

Lorenzo Sommaruga

Contenuti ed Obiettivi

Contenuti

- Architettura REST
- Sviluppo di web services in REST e vincoli
- Descrizione di API REST

Obiettivi

- (BI-2) Comprendere lo stile di architettura web di REST
- (BI-2) Comprendere i vincoli di REST
- (BI-2) Comprendere CRUD con HTTP in REST
- (BI-1) Conoscere come descrivere API REST

Cos'è un Web Service?

- Working Def. del W3C
- Ref: URI http://www.w3.org/TR/wsa-reqs/ Web Services Architecture Requirements, W3C Working Group Note 11 February 2004

- Web service

  "A Web service is a software system identified by a URI [RFC 2396], whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols."

Cos'è REST?

- REST = REpresentational State Transfer

- Termine introdotto da Roy T. Fielding 2000 (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

- per descrivere uno stile di sistema distribuito in rete

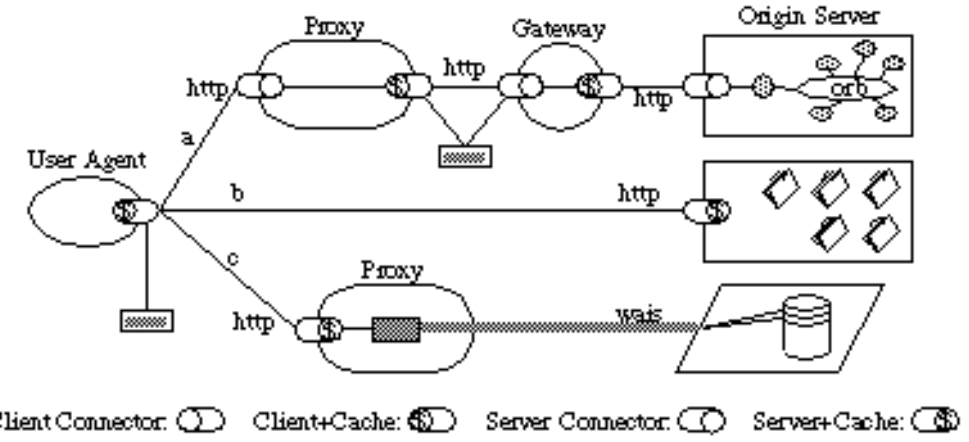

Figure 5-10. Process View of a REST-based Architecture

A user agent is portrayed in the midst of three parallel interactions: a, b, and c. The interactions were not satisfied by the user agent's client connector cache, so each request has been routed to the resource origin according to the properties of each resource identifier and the configuration of the client connector. Request (a) has been sent to a local proxy, which in turn accesses a caching gateway found by DNS lookup, which forwards the request on to be satisfied by an origin server whose internal resources are defined by an encapsulated object request broker architecture. Request (b) is sent directly to an origin server, which is able to satisfy the request from its own cache. Request (c) is sent to a proxy that is capable of directly accessing WAIS, an information service that is separate from the Web architecture, and translating the WAIS response into a format recognized by the generic connector interface. Each component is only aware of the interaction with their own client or server connectors; the overall process topology is an artifact of our view.

Motivazioni per REST

"The motivation for developing REST was to create an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web protocol standards.

REST has been applied to describe the desired Web architecture, help identify existing problems, compare alternative solutions, and ensure that protocol extensions would not violate the core constraints that make the Web successful."

- Roy Fielding

Representational State Transfer

- Il Client richiede una risorsa Web mediante un URI
- Una *rappresentazione* (*representation*) della risorsa viene ritornata (pagina HTML)
- Questa *rappresentazione* (e.g., amazon1234.html) mette l'applicazione client in uno *stato* (*state*)
- Il Client seguendo un link nella pagina HTML accede ad un'altra risorsa (e.g. XML Bible) che ritorna una sua *rappresentazione*
- Questa nuova rappresentazione mette l'applicazione client in un altro *stato*
- L'applicazione client cambia (*transfers*) così stato con ogni rappresentazione di risorsa

→ *Representational State Transfer*

Macchina a stati virtuale

- Comportamento di una applicazione web come macchina a stati virtuale
- Selezione di links = transizione di stati

  che portano alla
- pagina seguente = stato seguente dell'applicazione

REST non è uno Standard!

- **REST è uno stile di architettura**
- **Non c'e' una specifica**
- **Non c'è standard di sviluppo REST**

- **Può essere usato come stile per progettare web services**
- **Anche se non è uno standard REST prescrive l'uso di standards:**
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/… (Rappresentazione di Risorse)
  - text/xml, text/html, image/gif, image/jpeg, etc (Resource Types, MIME Types)

Caratteristiche di REST

- Stile d'interazione pull-based Client-Server: i nodi clienti richiedono le rappresentazioni
- Stateless: ogni richiesta deve contenere tutte le informazioni necessarie per comprendere la richiesta senza contare su info di contesto nel server
- Le risorse del sistema sono nominate con un URI
- Le risorse (representations) sono interconnesse mediante URI che permettono il passaggio da uno stato ad un altro

I sei vincoli di REST

1. Interfaccia uniforme
2. Stateless
3. Cacheable
4. Client-Server
5. Sistema stratificato
6. Codifica on demand (facoltativo)

Ref.: http://www.restapitutorial.com/lessons/whatisrest.html

Vincolo 1 - Interfaccia uniforme 1/2

- Il vincolo di un'interfaccia uniforme **definisce l'interfaccia tra client e server**
- Semplifica e disaccoppia l'architettura, permette ad ogni parte di evolvere in modo indipendente
- **Quattro principi guida dell'interfaccia uniforme:**

Pg1 - Basata sulle risorse identificate da URI

- Le risorse sono concettualmente distinte dalle loro rappresentazioni che vengono restituite al cliente. Ad esempio, il server non invia il proprio database, ma piuttosto, un po' di HTML, XML o JSON che rappresenta alcuni record del database espresso, per esempio, in finlandese e codificato in UTF-8, a seconda dei particolari della richiesta e l'implementazione del server

Pg2 - Manipolazione delle risorse attraverso le rappresentazioni

- Quando un cliente detiene una rappresentazione di una risorsa, compresi eventuali metadati allegati, si dispone di informazioni sufficienti per modificare o eliminare la risorsa sul server, se si ha permesso di farlo

1 - Interfaccia uniforme 2/2

- **Pg3 - Messaggi auto-descrittivi**
  - Ogni messaggio contiene informazioni sufficienti per descrivere come elaborarlo. Per esempio, quale parser invocare è specificato con un tipo di media di Internet (precedentemente conosciuto come un tipo MIME). Le risposte indicano anche in modo esplicito la loro cache-ability.

- **Pg4 - Hypermedia Come Il Motore Di Stato dell' Applicazione (HATEOAS)**
  - L'applicazione deve essere pilotata da collegamenti ipertestuali, che consentono ai clienti di scoprire altre risorse

  - I clienti forniscono lo stato tramite il contenuto del body, i parametri di query-string, request headers e l'URI richiesto (il nome della risorsa). I Servizi forniscono lo stato ai clienti tramite il contenuto del body, i codici di risposta, e response headers. Questo è tecnicamente definito come ipermediale (o collegamenti ipertestuali all'interno di un ipertesto)
  - HATEOAS significa anche che, dove necessario, i collegamenti sono contenuti nel boby restituito (o header) fornendo l'URI per il recupero dell'oggetto stesso o oggetti correlati

Vincolo 2 - Stateless

- **Lo Stato necessario per gestire la richiesta è contenuto all'interno della richiesta** stessa, sia come parte dell'URI, parametri di query-string, boby, o le headers. **L'URI identifica in modo univoco la risorsa e il body contiene lo stato** (o il cambiamento di stato) di quella risorsa. Poi, dopo che il server esegue l'elaborazione, lo stato appropriato, o il pezzo dello stato rilevante, sono comunicati al client tramite response header, stato e body
- Altrimenti in altro modo si usa il concetto di "sessione", che mantiene lo stato tra più richieste HTTP. **In REST, il cliente deve includere tutte le informazioni per il server per soddisfare la richiesta**, come se fosse la prima rinviando lo stato se necessario, se tale stato deve valere su più richieste. Stateless consente una maggiore scalabilità poiché il server non ha bisogno di mantenere, aggiornare o comunicare lo stato della sessione, minimizzando l'uso di memoria del server e la sua complessità. Inoltre, il bilanciamento del carico non deve preoccuparsi di sessioni
- Ogni ciclo di request-response deve rappresentare un'interazione completa del client con il server
- **Differenza tra Stato e Risorsa**: considerare lo stato dell'applicazione come dati che potrebbero variare a seconda del cliente, e per ogni richiesta; la risorsa, o stato della risorsa, invece, è costante in ogni cliente che lo richieda

- Senza stato il server ha -> **scalabilità** (e.g. replicazione e load-balancer)

Vincolo 3 - Cacheable

- Come nel World Wide Web, **i clienti possono memorizzare nella cache le risposte**

- Le **risposte** devono quindi, implicitamente o esplicitamente, **dichiarare** se sono **memorizzabili nella cache o no**, per evitare che i clienti riutilizzino dati non aggiornati o inappropriati in risposta alle ulteriori richieste

- Caching ben gestiti eliminano parzialmente o completamente alcune interazioni client-server, migliorando ulteriormente la scalabilità e le prestazioni

Vincolo 4 - Client-Server

- L'interfaccia uniforme **separa i client dai server**
- Questa **separazione di ambiti (paradigma SoC)** significa, per esempio, che i clienti non si preoccupano di memorizzazione dei dati, che rimane interna a ciascun server, in modo che la portabilità del codice client è migliorata
- I server non sono interessati allo stato di interfaccia utente, in modo che **i server possono essere più semplice e scalabili**
- I **server e client** potranno essere **sostituiti e sviluppati in modo indipendente**, finché l'interfaccia non viene alterata.

Vincolo 5 - Sistema stratificato

- **Componenti intermedi** (e.g. proxy, load-balancer) **possono essere messi tra il client e il server** sfruttando l'interfaccia uniforme del web
- Un client non può generalmente dire se è collegato direttamente al server finale, o ad un intermediario lungo il cammino
- **Server intermediari** possono migliorare la **scalabilità** del sistema, consentendo il bilanciamento del carico e fornendo cache condivise
- Gli strati possono anche implementare policy di sicurezza

Vincolo 6 - Codice on Demand (opzionale)

- I server sono in grado di estendere o personalizzare la funzionalità di un client trasferendo logica in modo che esso la possa eseguire temporaneamente. Esempi di questo possono includere componenti compilati (applet Java) e script lato client come JavaScript

Il rispetto di questi vincoli, e quindi la conformità allo stile architettonico REST, consentirà a qualsiasi tipo di sistema ipermediale distribuito di avere proprietà emergenti desiderabili, quali le prestazioni, la scalabilità, la semplicità, modificabilità, la visibilità, la portabilità e affidabilità.

NOTA: L'unico vincolo opzionale dell'architettura REST è il codice su richiesta. Se un servizio viola qualsiasi altro vincolo, non può essere strettamente indicato come RESTful

REST - HTTP access pattern

- **CRUD via HTTP**
- **Idea base è di una avere una Resource, con un URI, una Representation, un insieme standard codificato di Operations**
- **Principali OPERAZIONI:**

Operazione	Descrizione	Metodo HTTP
Create	Crea una nuova risorsa	POST
Read	Trasferisce la risorsa senza effetti collaterali	GET
Update	Modifica il valore di una risorsa	PUT
Delete	Elimina risorsa	DELETE

REST e il Web

- **Il Web è un esempio di sistema REST**

- **Esistono numerosi servizi Web che seguono la filosofia REST, e.g.:**
  - ordinazione libri,
  - servizi di ricerca,
  - dizionari online services,
  - …

Esempio API Google calendar

GET https://www.googleapis.com/calendar/v3/users/me/calendarList?key={YOUR_API_KEY}

GET …/me/calendarList/isin04.supsi%40gmail.com?key={YOUR_API_KEY}

Response

200 OK

{
"kind": "calendar#calendarList",
"etag": "\"p3xxxxxscxxxxxg\"",
"nextSyncToken": "CNjxxxxxxxxxbWxxxxxx29t",
"items": [
{
"kind": "calendar#calendarListEntry",
"etag": "\"12345678939000\"",
"id": "isin04.supsi@gmail.com",
"summary": "Calendario ISIN",
"timeZone": "Europe/Zurich",
"summaryOverride": "Calendario ISIN",
"colorId": "17",
"backgroundColor": "#9a9cff",
"foregroundColor": "#000000",
"accessRole": "reader",
"defaultReminders": [
],
"conferenceProperties": {
"allowedConferenceSolutionTypes": [
"eventHangout"
]
}
}, …

Descrizione di Web services

- Descritti mediante:
  - **WSDL**
  - WRDL (Web Resource Description Language)
    - Tipicamente un servizio consiste di documenti XML, di cui XML schemas ne danno una parziale descrizione
    - Quello che non descrivono sono le transizioni da un documento all'altro
    - cioè il "service's runtime behaviour" ← descritto da WRDL
  - WADL (Web Application Description Language)

Descrizione di Web services in REST
Specifica OpenAPI

- https://www.openapis.org/
- governata da Linux Foundation Nov 2015
- Basata su Swagger Specification
- Specifica OpenAPI gen 2016
- Standardizza come sono descritte le API REST con un formato neutrale (indip. da vendor):
    - Endpoints disponibili (e.g. /users) e operazioni su ogni endpoint (e.g. GET /users, POST /users)
    - Parametri dell' operazione, Input e output
    - Metodi di authentication
    - Info di contatto, license, termini d'uso, …
- Scritta in YAML o JSON

Riferimenti

- RESTful Service Best Practices Recommendations for Creating Web Services, Tutorial per implementare REST: http://www.restapitutorial.com/

- https://www.openapis.org

- https://swagger.io/tools/swagger-ui/