Obsoleted

http://goo.gl/6rf5g

Representational State Transfer (REST)

Introduction

Representational State Transfer/ Úvod

Roy Fielding: ***Architectural Styles and the Design of Network-based Software Architectures*** *(2000)*

REST

- architektonický styl pro distrubuovaná hypermedia
- REST definuje množinu architektonických omezení
- aplikace REST na webové technologie (URI, HTTP)

Cíle návrhu

- vysoká škálovatelnost aplikace, aplikačních komponent
- obecné aplikační rozhraní
- možnost použití 'intermediary' komponent (proxy, cache,...), snížení času odezvy, vynucení autentizace, a integraci legacy systémů

Representational State Transfer/ Úvod

Roy Thomas Fielding

- spoluautor HTTP 1.0 (1996) a 1.1 (1999). První verze HTTP (0.9) vznikla v roce 1991
- Apache HTTP Server co-founder
- OpenSolaris board member
- Apache Software Foundation Chairman

Navržená architektonická omezení v aplikaci na HTTP vynucují jeho ideální použití a zaručují škálovatelnost. REST ale není svázán pouze s URI a HTTP, může být aplikován i na jiné distribuované prostředí.

Web je distribuovaný systém hypermedií dostupných přes internet. RESTless aplikace tunelují Web. REST je architektonický styl Webu.

Hypermedia

Non-linear/interactive medium of information. (e.g. hypertext, hyperlink,...)

Hypermedia jsou nadmožinou multimedií.

Web je distrubuované hypermedium.

Hypermedia as the engine of application state (HATEOAS).

e.g. Multimédium je filmový pásek, kombinuje zvuk, obraz, text. je lineární. Hypertext je hypermédium, umožňuje nelineární navigaci pomocí linků.

Hypermedia/ Example/ HTML

HyperText Markup Language (HTML)

```
<html>
   <head>
        <link rel="stylesheet" href="..."/>
   </head>
    ...
   <a rel="next" href="...">next</a>
   ...
</html>
```

Hypermedia/ Example/ XML

eXtensible Markup Language (XML)

```
<persons>
   <person>
       <link rel="self" href="..."/>
   </person>
    ...
   <link rel="next" href="...">next</link>
</persons>
```

Ne všechny reprezentace jsou hypermediální! Co JSON?

Resources

- Prostředek je zdroj specifické informace
- Prostředek může zastupovat jakýkoliv informaci, statická data, stav dynamických komponent,...
- Kolekce je množina prostředků a také prostředek
- Kolekce i prostředky jsou, v REST, vzhledem ke klientovi, beztypové, komunikace probíhá prostřednictvím jejich reprezentace

Příklady: Entita, Relace, Stav, Event, Soubor, Video,...

REST/ Architektonická omezení

- uniform interface
  - *identification of resources*
  - *manipulation of resources through representations*
  - *self-descriptive messages*
  - *hypermedia as the engine of application state*
- client-server
- stateless
- cache
- layered system
- code on demand (optional)

Aplikace splňující všechny tyto omezení je RESTful

REST/ Constraints/ Uniform Interface

IDENTIFICATION OF RESOURCES

- každý prostředek je jednoznačně identifikovatelný, má unikání adresu, e.g. URL
- adresa neurčuje reprezentaci prostředku, s jedinou vyjímkou, file postfix type, e.g. .hml, .txt, .xml.
- je dobrým zvykem odlišit kolekci pomocí slash '/' (analogie k adresářové struktuře filesystémů)

```
e.g.
   http://..../product/3123213
   http://..../books/
```

REST/ Constraints/ Uniform Interface

MANIPULATION OF RESOURCES THROUGH REPRESENTATIONS

- klient pracuje s prostředkem vždy pouze pomocí jeho reprezentace, která nemusí odpovídat tomu jak s prostředkem manipuluje serverová část.
- reprezentace se skládá z serializovaného prostředku a metadat
- reprezentace prostředku, včetně metadat, musí nést kompletní informaci pro zpracování
- mezi klientem a serverem probíhá content media type negotiation

REST/ Constraints/ Uniform Interface

SELF-DESCRIPTIVE MESSAGES

Požadavky i odpovědi obsahují kompletní informaci.

Požadavky i odpovědi mají nastaven příslušný Media Type.

- v HTTP se využívají HTTP hlavičky (metadata); e.g. Content-Type pro typ serializace prostředku
- Custom Content-Type; e.g. applicaton/vnd.[something] +json

REST/ Constraints/ Uniform Interface

SELF-DESCRIPTIVE MESSAGES

Obsah požadavků na server i odpovědí má vypovídací hodnotu, lze jednoduše určit,odhadnout, sémantiku vlastního požadavku/odpovědi, i jeho částí.

```
"person" { "firstName": "John", 'lastName': 'Smith', "age": 25,
"address": { "streetAddress": "21 2nd Street", "city": "New York",
"state": "NY", "postalCode": 10021, } }

"interfaces" : [ { "vnet": "/vnets/10.31.145.0", "mac_address":
"00:16:3E:08:00:92", "ip_address": "10.31.145.254", "nic": "eth0"
} ]
```

REST/ Constraints/ Uniform Interface

HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

Existuje pouze několik výchozích prostředků, které mají definovanou adresu. Ideálně server poskytuje pouze jeden entry point. e.g. /api/v1

Reprezentace prostředku obsahuje adresy prostředků, včetně metod (links), které umožňují získání dalších prostředků, nebo manipulaci s prostředkem. Server může tyto informace filtrovat dle oprávnění, stavu aplikace.

REST/ Constraints/ Uniform Interface

HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

Linky obsahují sémantickou informaci.

```
e.g.
<person>
  <name>....</name>
  <link rel="self" href=".../person/321"/>
  <link rel="address" href=".../person/321/address"/>
</person>
```

Klient se řídí, konfiguruje, dle obsahu získané reprezentace.

REST/ Constraints/ Uniform Interface

HYPERMEDIA AS THE ENGINE OF APPLICATION STATE (HATEOAS)

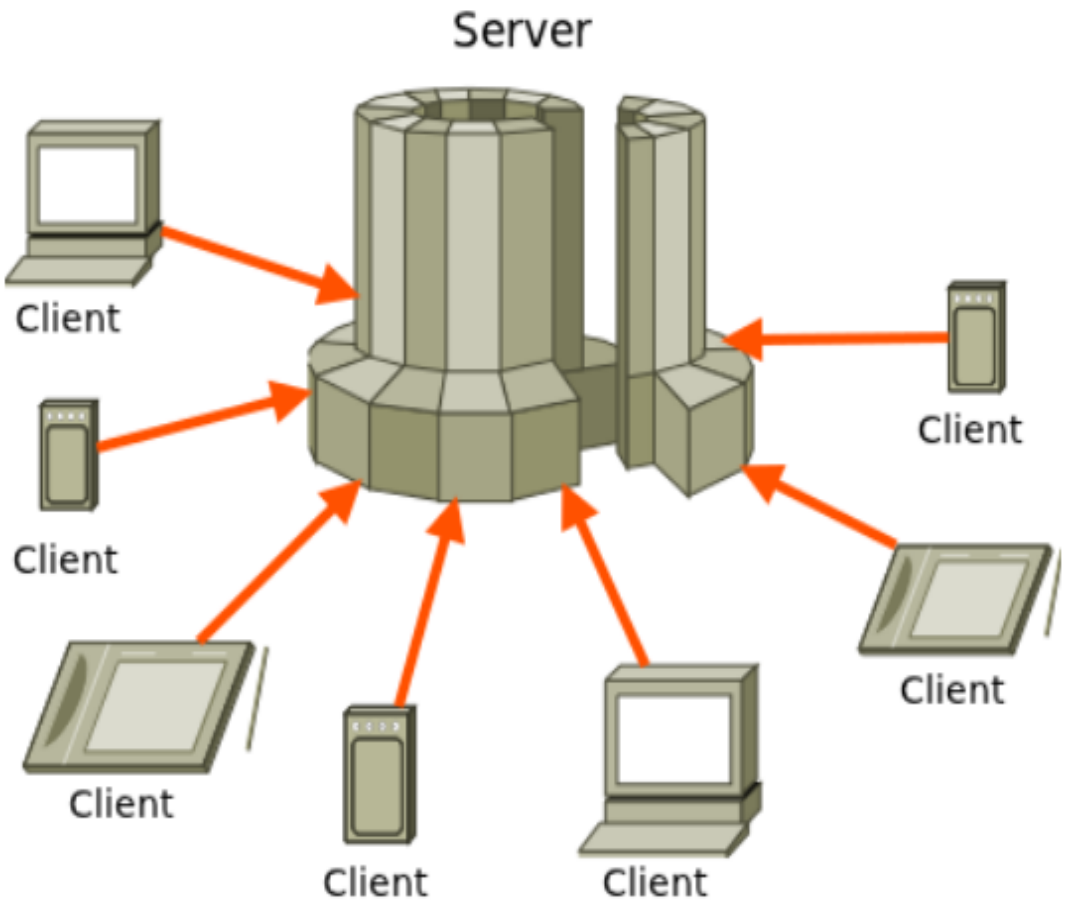
Pokud známe adresu prostředku může klient provést auto discovery na metody které jsou podporovány, povoleny.

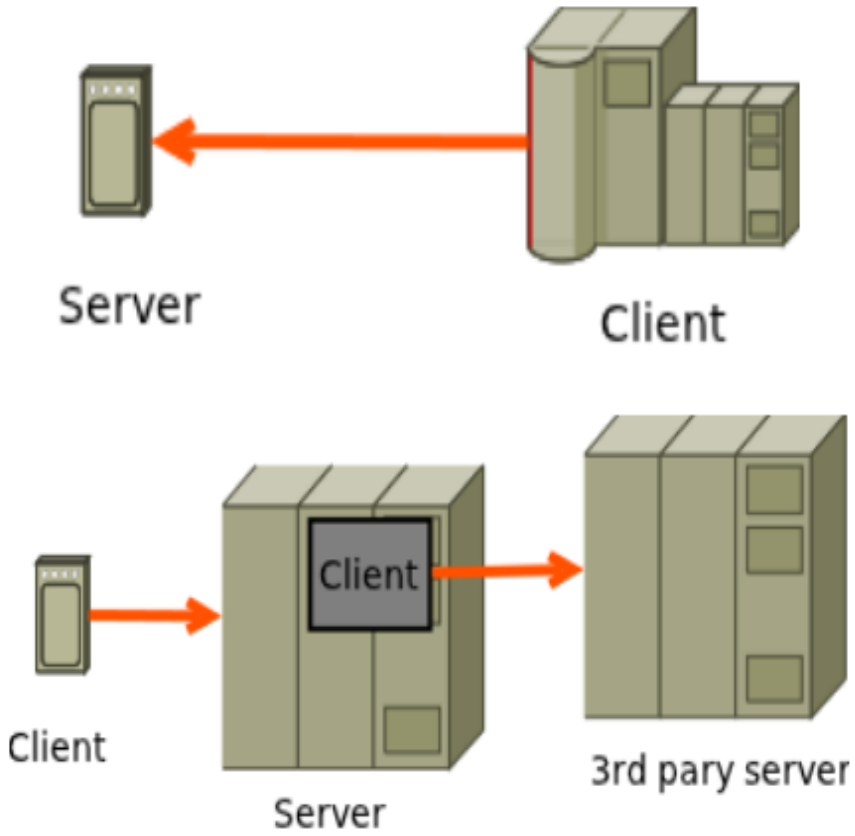
```
e.g.
OPTIONS /person/31231

204 No Content
Allow: OPTIONS,GET,PUT
Content-Length: 0
Date: Sun, 13 Mar 2011 21:11:44 GMT
```

REST/ Constraints/ Client-Server

Architecture for distributed applications (alternative: peer-to-peer)

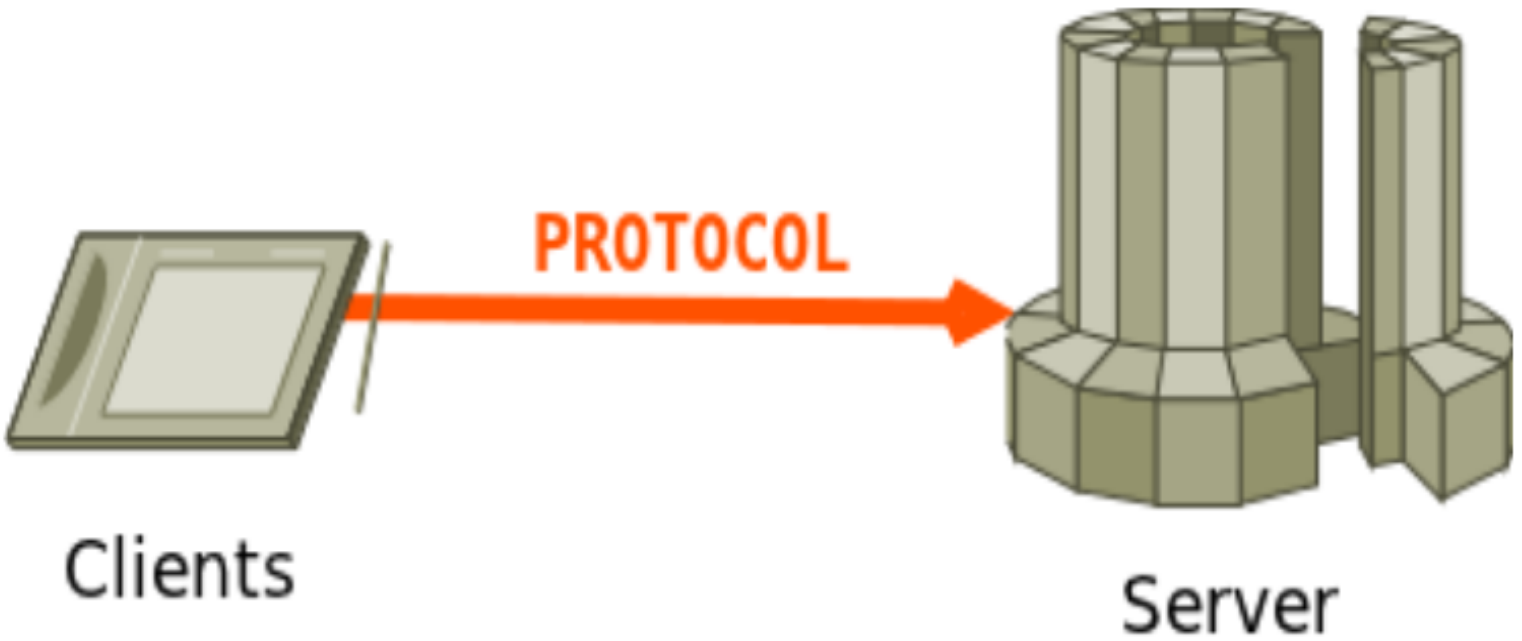




Notes

- **Server can use a client to connect another server**
- **Fat vs Thin Client**

REST/ Constraints/ Client-Server



client-server protocols: **HTTP**, SMTP, Telnet, DNS, SSH, etc.

REST/ Constraints/ Client-Server

Požadavky

- oddělení rozhraní serverové a klientské aplikace
- klientská aplikace využívá služby serveru transparentně (proxies, firewalls, gateways)

Výhody

- přenositelnost klientské aplikace, více klientů
- škálovatelnost serverové aplikace
- transparentní vývoj bez skrytých závislostí

REST/ Constraints/ Stateless

Požadavky

- každý požadavek klienta na server musí obsahovat všechny potřebné informace k jeho zpracování, nelze předpokládat v jakém stavu server je
- klientská aplikace využívá služby serveru transparentně
- pokud klientská aplikace potřebuje 'session informace' spravuje je plně ve své režii

Výhody

- škálovatelnost
- spolehlivost, aplikace se snáze zotaví

REST/ Constraints/ Cache

Server označuje cacheable odpovědi a umožňuje tak zapojení proxies, caches (browser cache).

Výhody

- performance klienta, snížení latence
- škálovatelnost

Využívá se HTTP hlaviček, Cache-Control, Expires, ETag,...

REST/ Constraints/ Layered System

Architektura aplikace je složena z hiearchie vrstev, každá z vrstev obsahuje množinu komponent a 'nevidí' dále než na sousední vrstvy.

Klient nikdy neví zda komunikuje přímo se serverem nebo prostředníkem (e.g. proxy, browser-cache).

Vylepšuje škálovatelnost, umožňuje load-balacing, vynucení bezpečnostní politiky, transparentní integraci legacy systémů

REST/ Constraints/ Code on Demand (optional)

Klientská aplikace může stahovat scripty, applety, a tím umožnit rozšíření své funkcionality.

Výhodou je snadná rozšiřitelnost klienta, deployment, aktualizace klientů.

Z důvodů snížení visibility, e.g. ne všichni uživatelé mají nainstalovanou javu pro podporu java applets, je tento vzor volitelný.

REST/ Architektonická omezení

- uniform interface
  - *identification of resources*
  - *manipulation of resources through representations*
  - *self-descriptive messages*
  - *hypermedia as the engine of application state*
- client-server
- stateless
- cache
- layered system
- code on demand (optional)

Aplikace splňující všechny tyto omezení je RESTful

REST/ Web Service API

Aplikační rozhraní je definováno s použitím URL, HTTP, a při jeho návrhu jsou aplikována omezující pravidla.

- URL -> unique resource identification
- Internet Media Type (Content-Type) -> resource representation type[s]
- HTTP Methods -> resource CRUD + metadata methods
- custom HTTP headers -> metadata
- HTTP Accept, Vary headers -> media type negotiation

Aplikace REST na aplikační interface není postačující pro splnění všech podmínek.
HTTP API != RESTful

REST/ Web Service API/ CRUD

CREATE -> HTTP POST, URL kolekce
    vytvoří nový prostředek a jeho identifikátor

READ -> HTTP GET, URL
    vrátí reprezentaci prostředku nebo obsahu kolekce prostředků

UPDATE -> HTTP PUT, URL prostředku
    aktualizace nebo vytvoření prostředku s daným identifikátorem

DELETE -> HTTP DELETE, URL prostředku
    odstraní prostředek

REST/ Web Service API/ CRUD

READ METADATA -> HTTP HEAD, URL
    vrací seznam HTTP hlaviček nesoucí metadata

METHOD DISCOVERY -> HTTP OPTIONS, URL
    vrací seznam podporovaných HTTP metod

PARTIAL UPDATE -> HTTP PATCH (RFC proposed)
    aktualizuje pouze část informací, narozdíl od PUT které slouží jako čisté replace

REST/ Richardson's Maturity Model

Základní kategorizace webových aplikací, definovaná by Leonard Richardson, která umožňuje klasifikovat míru adopce REST do čtyř úrovní.

LEVELS

0 - The Swamp of POX

1 - Resources, breaking a large service endpoint down into multiple resources

2 - HTTP Verbs, *uniform inteface*

3 - Hypermedia Controls, *hypermedia as the engine of application state (HATEOAS)*

* - The Zen of REST ;)

HTTP based API Styles/ Overview

- **Remote Procedure Call (*-RPC)**
  - subject: methods, procedures
- **Simple Object Access Protocol (SOAP, SOA Web Services)**
  - subject: service methods
- **REST**
  - subject: representations of an information (just data)

REST vs. XML-RPC

- REST: architektonický styl
- XML-RPC: protokol pro volání vzdálených metod

- REST: přenos reprezentací prostředků
- XML-RPC: volání vzdálených metod

- REST: jednoznačně identifikuje aplikační prostředky a metody
- XML-RPC: publikuje metody, aplikační prostředky jsou schované za jejich invokací

Použití XML-RPC nedokáže samoosobě zaručit škálování aplikace, možnost použití cache, proxy, ..., je to protokol.

REST/ Design Pattern/ General Rules

- využití toho co HTTP obsahuje
- využití HTTP hlaviček pro popis metadat
- využití Cache-Control, Expire, ETag hlaviček
- HATEOAS, self-discovery
- zavedení vlastních HTTP hlaviček pro popis vlastních metadat (X-*)
- zavedení vlastních reprezentací (e.g. application/vnd.*)
- POST není idenpotetní, GET/PUT/DELETE ano

REST/ Design Pattern/ Authorization

Využívá se HTTP hlavičky Authorization v kombinaci s shared token.

```
e.g.
GET / HTTP/1.1
Host: commondatastorage.googleapis.com
Date: Mon, 15 Feb 2010 11:00:00 GMT
Authorization:GOOG1GOOGTS7C7FUP3AIRVJTE:Y9gBLAEInIlFv5zlAm9ts=
Content-Length: 0
```

REST/ Design Pattern /Paging

QUERY PARAMETERS

uses URL query parameters

e.g.

/cars/?offset=1

/cars/?offset=34&sortField=name&sortDirection=asc

/cars/?offset=12&sort=name:asc,type:desc

/cars/?offset=10&limit=30

Most proxies won't cache resources which include parameters

REST/ Design Pattern /Paging

TEMPLATE PARAMETERS/NEW 'FILTER' RESOURCE
uses part of the URL to carry the paging information

e.g.
/cars/1
/cars/name/asc/32
/cars/name/asc/type/desc/11
/cars/3/30/
/cars/name/2/50/

Nevýhodou je plýtvání adresním prostorem a zavedení množství prostředků které slouží pouze jako jiný pohled, přičemž informace o této relaci je pro ostatní aplikace/komponenty netransparentní.

REST/ Design Pattern /Paging

(CUSTOM) HTTP HEADERS
uses HTTP header (Content-Range for binary files or custom headers)

e.g.
GET /cars/
Item-Range: 10-30;name:asc

GET /cars/
Item-Range: 23-63;name:asc;type:desc

REST/ Example/ Google Storage API

Google Storage Service umožňuje ukládata, spravovat a sdílet data. Data jsou fyzicky uložena 'v cloudu'. Přístup k datům je umožnen přes RESTful API.

Prostředky:
Buckets, Objects, ACL

REST/ Example/ Google Storage API

Identifikátory, URL

výchozí URL, kontejner pro všechny buckety
http://commondatastorage.googleapis.com/ - pouze LIST

Buckets, Objects, ACLs
http[s]://[bucketname]co..googleapis.com/objectname[?acl]

REST/ Example/ Google Storage API

Prostředky

Buckets
kontejner obalující objekty, může mít ACL, ale nelze vnořit

Objects
Object Data - vlastní data, content
Object Metadata - metadata, name=value pairs list (custom HTTP headers)

ACL - Access Control List
seznam oprávnění k danému buckets, objektu
READ/WRITE/FULL_CONTROL

REST/ Example/ Google Storage API

API

- GET Service - seznam buckets [read, kolekce]
- PUT Bucket - vytvoření bucket [create/update]
- GET Bucket - seznam objektu v bucket
- DELETE Bucket - odstranění prázdného bucket
- GET Object - získání objektu [read, prostředek]
- PUT Object - vytvoření objektu s daným id [create/update]
- DELETE Object - odstranění objektu
- HEAD Object - metadata objektu
- POST Object - vytvoření objektu

REST/ Notes

- Amazon has both SOAP and REST interfaces to their web services and 85% of their usage is of the REST interface.
- Yahoo uses REST for all their services including Flickr and del.ici.ous
- Google used to provide only SOAP for all their services, but in 2006 they deprecated in favor of REST
- WAKA - binary token based replacement for HTTP by R.T. Fielding (unfinished)
- SPDY (speedy) - TCP based protocol for transporting web content by Google (Search, GMail)
- Hyperland - a documentary film about hypertext written by Adam Douglas! in 1990! (in four parts on YouTube)
- RESTafarian is REST envagelist/advocate ;)

"Web's major goal was to be a shared information space through which people and machines could communicate." *Tim Berners-Lee*