



tREST

framework pre
webové aplikácie a služby

Peter Rybár

Centaur s.r.o.

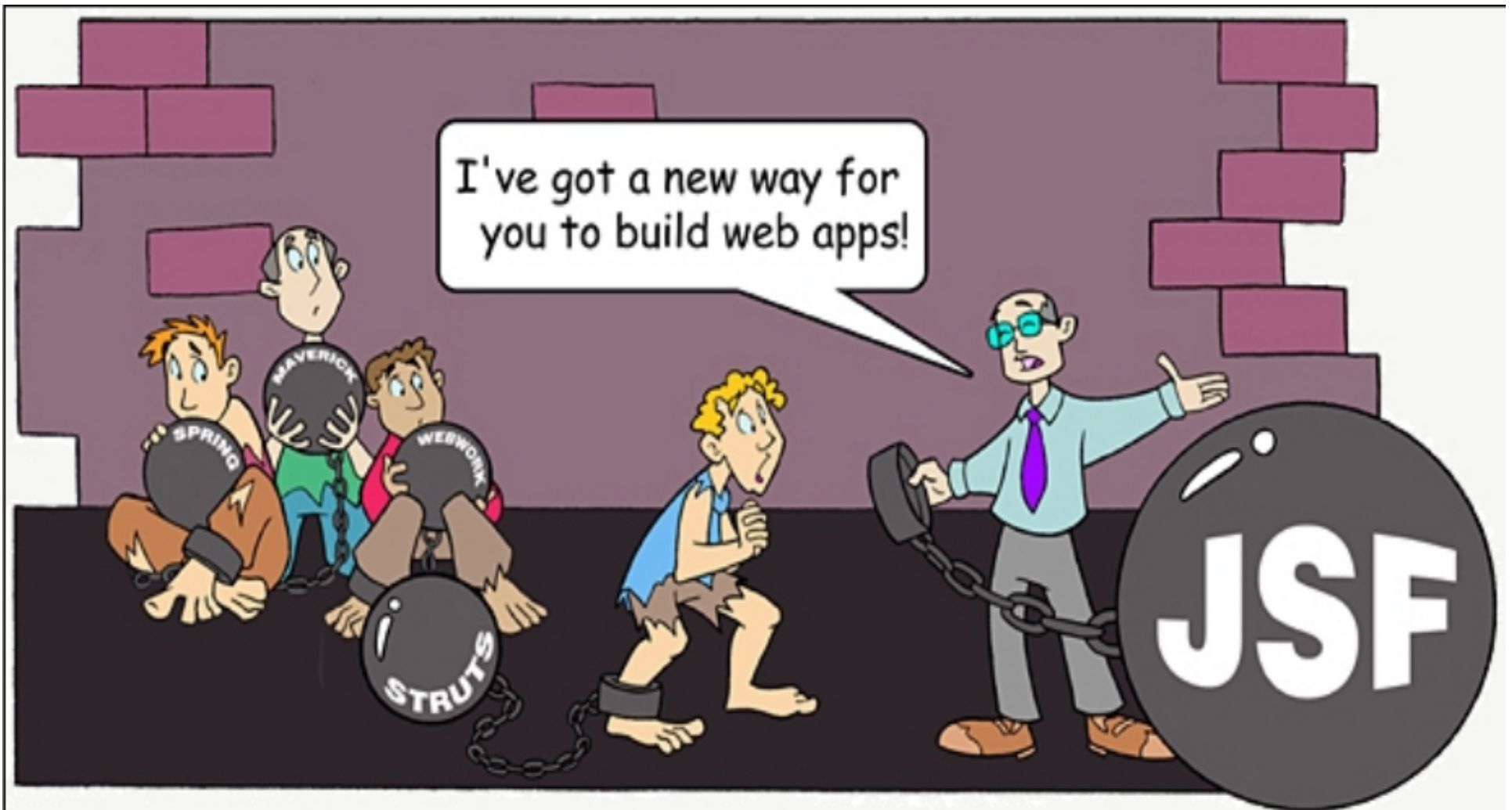


Obsah

- Zhodnotenie situácie v IT
- REST
 - Web, Webové služby, REST, ROA
- tREST
 - Server – Java
 - Klient – Javascript
 - Výkon
 - Budúcnosť a smerovanie

Situácia v korporátnej sfére

- Dominuje technológia a nie architektúra





Situácia na Webe

- Dominuje architektúra **ROA**
- **REST** – štýl softvérovej architektúry pre distribuované hyper-mediálne systémy
- REST Webové aplikácie:
 - Google
 - <http://code.google.com/webtoolkit/>
 - Amazon
 - <http://developer.amazonwebservices.com/>
 - Yahoo!
 - <http://developer.yahoo.com/>



Ktorú cestu zvolit’?

■ Otázka

- ☐ **SEO** – Search engine optimization
- ☐ Funkcionalita
- ☐ Ceny
- ☐ Výkonu
- ☐ Otvorenosti kódu
- ☐ Technickej podpory
- ☐ Zložitosti vývoja
- ☐ **Kvality vývojárov**



Výber riešenia

- Prečo je java web frameworkov viac ako realizovaných projektov?
 - Žiaden framework nevie všetko
 - Rôzna náročnosť vývoja
 - Rôzna väzba na iné technológie
- Ktorý web framework je najlepší?
 - Vlastný
 - Lebo ho dokonale poznáte!
 - Snaha kupovať ľudí za technológiami
Guido van Rossum, Anders Hejlsberg, ...



REST

Peter Rybár

Centaur s.r.o.



WEB

- **World Wide Web** (skrátene **Web**) je systém vzájomne nalinkovaných hypertextových dokumentov prístupných prostredníctvom **HTTP** v sieti Internet
- **World Wide Web** vytvorili Sir **Tim Berners-Lee** and **Robert Cailliau** v roku **1989** v **CERNe** v Ženeve, Švajčiarsko
- Princíp – používateľ Webu (HTTP klient) prezerá web stránky obsahujúce **text, obrázky, videá**, a iné **multimédiá** a naviguje sa medzi nimi pomocou **hyperliniek** v **HTML**



Web – HTTP

- **HTTP** – Hypertext Transfer Protocol
- Transportný protokol pre prenos informácií, hypertextových stránok na internete
- Všetok Web prenos je realizovaný prostredníctvom jednoduchého **HTTP API**:
 - **GET** = "daj mi nejakú informáciu" (Retrieve)
 - **POST** = "tu máš nejakú novú informáciu" (Create)
 - **PUT** = "tu je nejaká update informácia" (Update)
 - **DELETE** = "vymaž nejakú informáciu" (Delete)
- **HTTP API** je **CRUD** (Create, Retrieve, Update, a Delete)



Webové komponenty

■ Firewall:

- Rozhodujú ktoré HTTP správy (messages) môžu von a ktoré môžu dnu
- Uplatňujú **Webovú bezpečnosť**

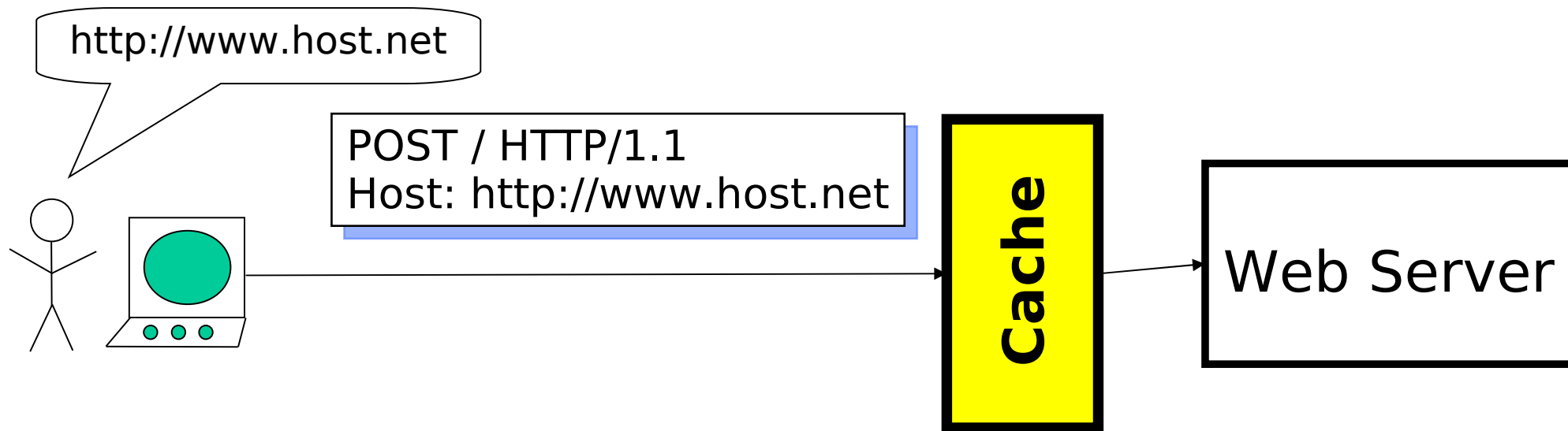
■ Route:

- Rozhodujú kde poslať HTTP správy (messages)
- Riadia **Webovú škálovateľnosť**

■ Caches:

- Rozhodujú či uložená kópia HTTP správy (message) môže byť použitá
- Zvyšujú **Webovú rýchlosť**

Webové komponenty: Cache



- Cache sa rozhoduje, či by mala použiť ešte platnú kópiu HTTP dokumentu, alebo si vyžiadať aktuálnejšiu správu od web servera
- Všetky rozhodnutia sú na základe **HTTP hlavičky**
- **Cache nikdy nepozera do prenášaných dát HTTP správ** (payload)

Webové služby

- Softvérový systém navrhnutý pre komunikáciu Stroj-Stroj prostredníctvom siete **internet**.
- Sada nástrojov, ktoré môžu byť použité rôznym spôsobom na rôzne účely.
- Tri najčastejšie spôsoby použitia sú:
 - **RPC**
 - **SOA**
 - **REST**



Webové služby: RPC

- RPC Webové služby reprezentujú **interface pre volanie vzdialenej funkcie** (metódy)
- Sú **kritizované pre silnú väzbu**, pretože boli implementované *mapovaním služieb priamo do špecifických funkcií a metód daného jazyka*
- Keď bola do XML-RPC zavedená nová funkcionálna, bol uvedený nový štandard známy dnes ako SOAP
- Niektorí vývojári stále preferujú XML-RPC pred SOAP pre jeho **minimalizmus a jednoduchosť použitia**

Webové služby: RPC

Príklad XML-RPC

POST /RPC HTTP/1.0

Host: example.org

Content-Type: text/xml

Content-length: nnn

```
<?xml version="1.0"?>
```

```
<methodCall>
```

```
  <methodName>examples.getStateName</methodName>
```

```
  <params>
```

```
    <param>
```

```
      <value><i4>41</i4></value>
```

```
    </param>
```

```
  </params>
```

```
</methodCall>
```



Webové služby: SOA

- Základnou jednotkou komunikácie je **správa (message)**, skôr než procedúra
- Táto architektúra zvykne byť označovaná ako “**message oriented**”
- Implementácia konceptu **servisne orientovanej architektúry (SOA)**
- Najčastejšie používaný protokol pre SOA je **SOAP Document Literal**
- **Simple Object Access Protocol** neskôr ako **Service Oriented Architecture Protocol**

Webové služby: SOA

Príklad SOAP Document Literal

POST /InStock HTTP/1.1

Host: www.example.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
<soap:Envelope
```

```
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
    <soap:Body xmlns:m="http://www.example.org/stock">
```

```
      <m:GetStockPrice>
```

```
        <m:StockName>IBM</m:StockName>
```

```
      </m:GetStockPrice>
```

```
    </soap:Body>
```

```
  </soap:Envelope>
```




Webové služby: REST

- **REST** - Representational state transfer
- **Dôraz** sa kladie **na interakciu so stavovými zdrojmi** (resources), než na **správy** (messages) alebo **operácie** (procedures)
- RESTovské Webové služby sa pokúšajú emulovať HTTP a podobné protokoly obmedzením rozhrania na sadu štandardných operácií (ako, GET, PUT, POST, DELETE).



Webové služby: REST

Príklad REST

POST /parts/12345 HTTP/1.0

Host: example.org

Content-Type: text/xml

Content-length: nnn

<?xml version="1.0"?>

<UnitCost currency="USD">0.10</UnitCost>

Web služby: SOAP vs REST

SOAP

The following is a sample SOAP request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /DemoWebServices2.8/service.asmx HTTP/1.1
Host: api.efxnow.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "https://api.efxnow.com/webservices2.3/GetTime"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetTime xmlns="https://api.efxnow.com/webservices2.3" />
  </soap:Body>
</soap:Envelope>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetTimeResponse xmlns="https://api.efxnow.com/webservices2.3">
      <GetTimeResult>string</GetTimeResult>
    </GetTimeResponse>
  </soap:Body>
</soap:Envelope>
```



Web služby: SOAP vs REST

HTTP GET

The following is a sample HTTP GET request and response. The [placeholders](#) shown need to be replaced with actual values.

```
GET /DemoWebServices2.8/service.asmx/GetTime? HTTP/1.1
Host: api.efxnow.com
```

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

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://api.efxnow.com/webservices2.3">string</string>
```

HTTP POST

The following is a sample HTTP POST request and response. The [placeholders](#) shown need to be replaced with actual values.

```
POST /DemoWebServices2.8/service.asmx/GetTime HTTP/1.1
Host: api.efxnow.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length
```

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

<?xml version="1.0" encoding="utf-8"?>
<string xmlns="https://api.efxnow.com/webservices2.3">string</string>
```



REST

- Akronym pre **R**epresentational **S**tate **T**ransfer
- **Štýl softvérovej architektúry** určený pre **distribúované hypermediálne systémy** ako je **World Wide Web**
- Zaviedol ho **Roy Fielding** v jeho Ph.D. práci pre **popis architektonického štýlu sieťou prepojených systémov**
- REST striktne definuje kolekciu princípov sieťovej architektúry, ktorá **vysvetľuje ako sú zdroje (resources) definované a adresované**



REST – Motivácia vzniku

- Zozbierať charakteristiky webu, ktoré ho spravili úspešným
- Následne použiť tieto charakteristiky ako návod pre evolúciu webu

REST – Poslanie

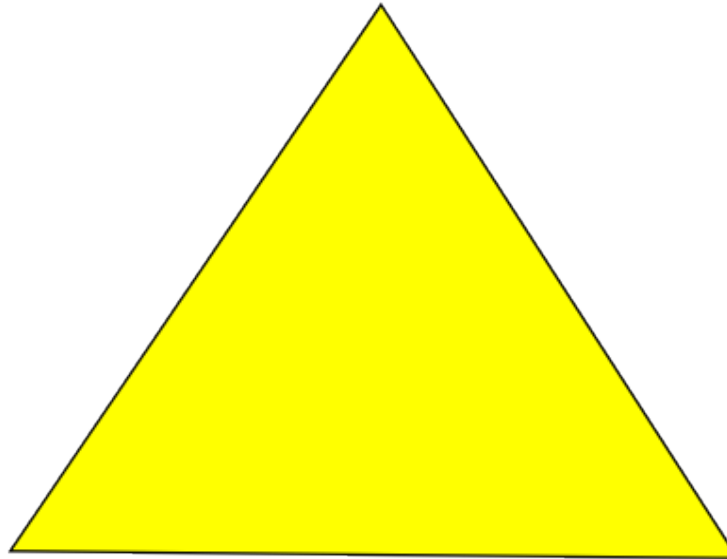
- REST je učený aby uviedol spôsob, **ako sa má správať dobre navrhnutá webová aplikácia**
 - Webová aplikácia je sieť web **stránok, virtuálnych stavových strojov**
 - Užívateľ prechádza cez aplikáciu prostredníctvom **liniek, prenosov stavu**
 - Výsledkom prenosu stavu je nasledujúca **stránka, reprezentujúca** nasledujúci **stav aplikácie**, prenesená k užívateľovi

REST – Restovský trojúhelník

Nouns

(Unconstrained)

eg <http://wikipedia.org/>



Verbs

(Constrained)

eg GET

Content Types

(Constrained)

eg HTML



REST – Prenos stavu

- Klient referencuje **zdroj** pomocou **URL**
- Klientovi je vrátená **reprezentácia** zdroja
- Reprezentácia uvedie klienta do **nového stavu**
- Klient zvolí hyperlinku ďalšieho **zdroja**
- Klientová aplikácia teda robí transfér stavu s každou reprezentáciou zdroja
- **Nová reprezentácia – nový stav!**

REST – Arch. štýl, nie štandard

- **REST nie je štandard!**
- Neuvidíte W3C špecifikáciu RESTu
- Nemali by ste vidieť predávať developerské toolkity od IBM, Microsoft alebo Sunu
- **Prečo?**
- Lebo **REST je “iba” architektonický štýl**
(analógia s klient-server štýlom)
- Môžete RESTu iba rozumieť a dizajnovat' aplikácie podľa neho



REST – Štandardy

- Hoci **REST** nie je štandard, **používa štandardy**:
 - **HTTP** (Transport protocol)
 - **URL** (Resource identifier)
 - **XML/HTML/GIF/JPEG/...** (Resource Representations)
 - **text/xml, text/html, image/gif, image/jpeg, ...** (MIME Types)



REST – Web je REST systém

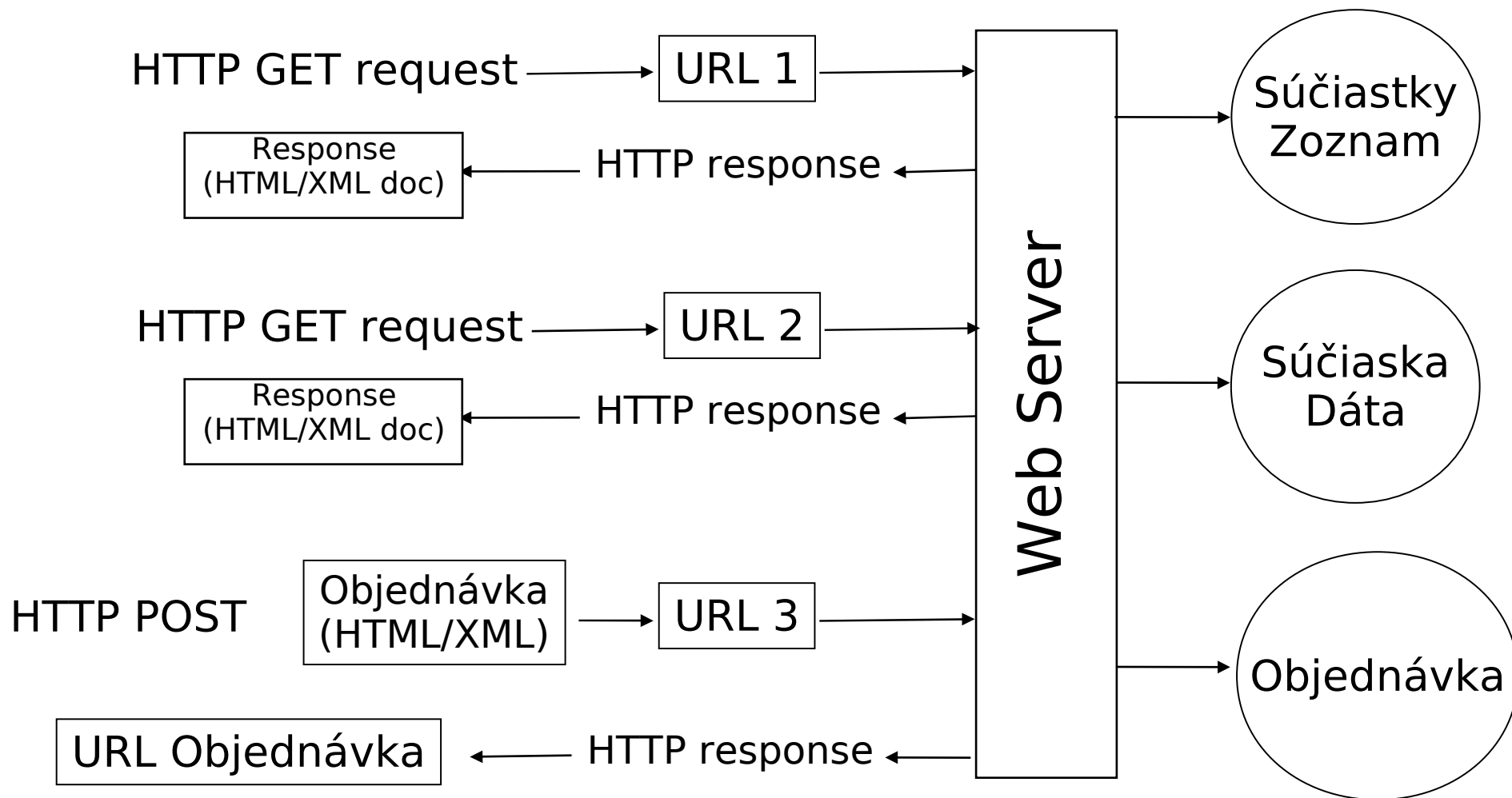
- Webové služby, ktoré používame už roky:
 - book-ordering services
 - search services
 - online dictionary services
- Všetko sú to RESTovské Webové služby
- Používali sme REST, stavali sme RESTovské webové služby a ani sme o tom “nevedeli” !
- **Google – koniec SOAP – už iba REST !**



REST – Príklad

- **Firma Sklad súčiastok s.r.o.**
- Sklad súčiastok – REST Web servis
- Umožniť zákazníkovi:
 - Získať výpis zoznamu súčiastok
 - Získať detailnú informáciu o konkrétnej súčiastke
 - Zaslať objednávku
- Ako bude služba implementovaná?

REST – Príklad



REST – Príklad

- Klient získa zoznam súčiastok na URL:
<http://sklad-suciasatok.sk/suciasatky>
- Klient obdrží dokument:

```
<?xml version="1.0"?>
<p:Parts xmlns:p="http://www.sklad-suciasatok.sk"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part id="00345" xlink:href="http://sklad-suciasatok.sk/suciasatky/00345"/>
  <Part id="00346" xlink:href="http://sklad-suciasatok.sk/suciasatky/00346"/>
  <Part id="00347" xlink:href="http://sklad-suciasatok.sk/suciasatky/00347"/>
  <Part id="00348" xlink:href="http://sklad-suciasatok.sk/suciasatky/00348"/>
</p:Parts>
```

- Zoznam súčiastok má linky na zdroje detailných informácií o súčiastkach

REST – Príklad

- Služba má URL na zdroj informácií o súčiastke na linke:

<http://sklad-suciasatok.sk/suciasatky/00345>

- Klient obdrží dokument:

```
<?xml version="1.0"?>
<p:Part xmlns:p="http://sklad-suciasatok.sk"
        xmlns:xlink="http://www.w3.org/1999/xlink">
  <Part-ID>00345</Part-ID>
  <Name>Widget-A</Name>
  <Description>This part is used within the frap assembly</Description>
  <Specification xlink:href="http://sklad-suciasatok.sk/suciasatky/00345/spec"/>
  <UnitCost currency="USD">0.10</UnitCost>
  <Quantity>10</Quantity>
</p:Part>
```


REST – Príklad

- Pre objednávku klient vytvorí objednávkový dokument podľa objednávacej schémy a odošle ho ako HTTP POST na url:
<http://www.sklad-suciastok.sk/objednavka/>
- Objednávková služba odpovie na HTTP POST s URL na odoslanú objednávku
<http://www.sklad-suciastok.sk/objednavka/1234>
- Teda klient môže kedykoľvek prístupit' k objednávke neskôr kvôli update/edit



REST – Pre a proti

■ Výhody

- Linkovanie a bookmarkovanie – “google friendly URLs”
- Podpora štandardnej sady operácií (CRUD)
- Škálovateľnosť
- Jednoduchá implementácia – HTTP libs
- Slabá väzba komponentov (loose coupling)
- Možná neskorá väzba – HTTP status 302
- Vyššia možnosť znovupoužitia kódu



REST – Pre a proti

■ Nevýhody

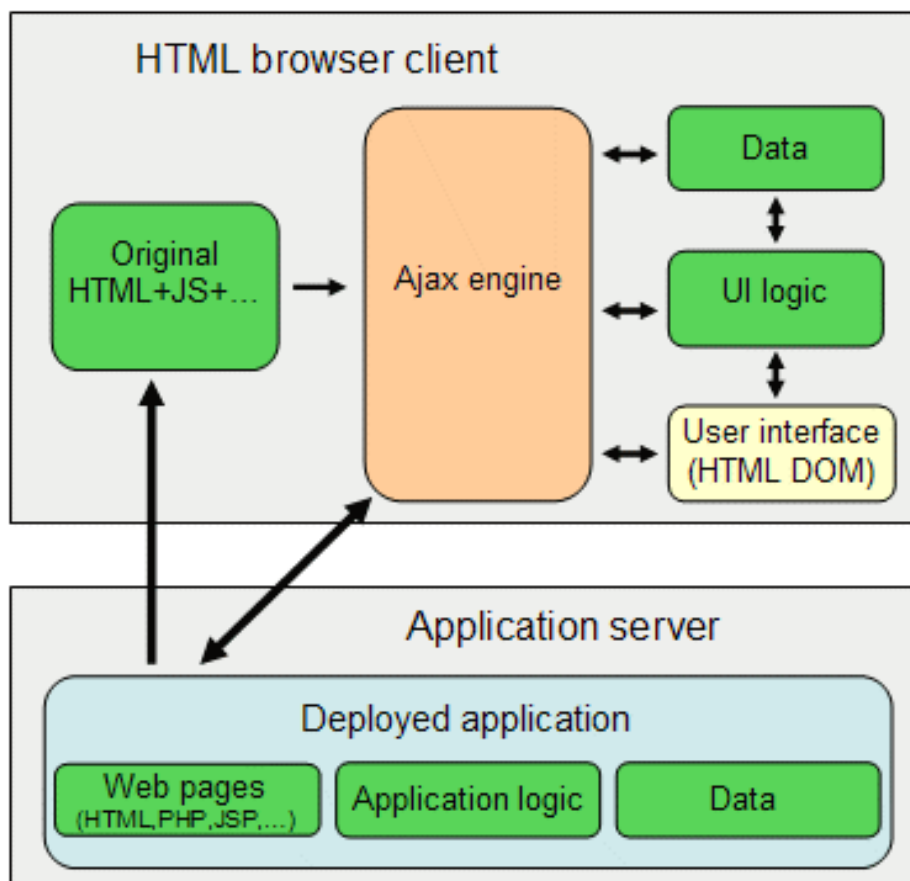
- Viazaný na HTTP
- Veľké množstvo objektov
- Správa URI menných priestorov (namespace) môže byť ťažkopádna – závisí na architektovi



Novodobé Web aplikácie

- **RIA** alebo **Web 2.0**
- **AJAX** - Asynchronous JavaScript and XML
 - Umožňuje asynchrónnu prácu s dátami
 - Nie je potrebné znovuzobrazenie stránky na zobrazenie nových dát
 - Jednoduché zavedenie dynamiky do webu
 - Umožňuje asynchrónnu prácu s dátami
- **Dátový zdroj** pre AJAX sú **REST** web služby!

Novodobé Web aplikácie



- Možné rôzne formáty výmeny dát
 - XML
 - XHTML
 - JSON
 - YAML
- Dbáme na **sémantiku** dát



Web 2.0 vlastnosti/techniky

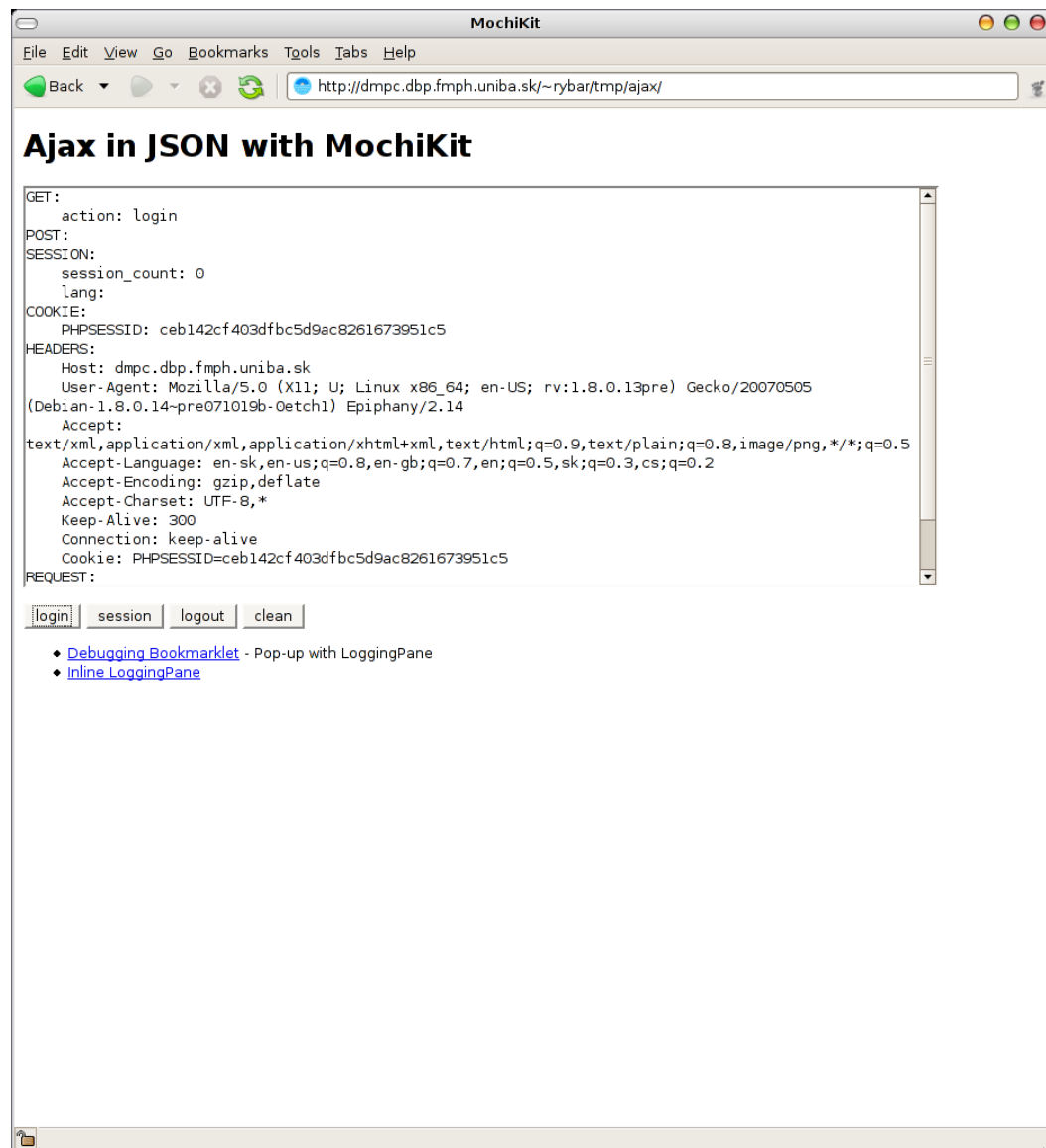
- **Rich Internet Application** techniky sú zvyčajne Ajax-based
- sémanticky valídne XHTML
- mikroformáty Rozširujú web stránky o novú prídavnú sémantiku
- **Cascading Style Sheets** separuje prezentáciu od obsahu



Web 2.0 vlastnosti/techniky

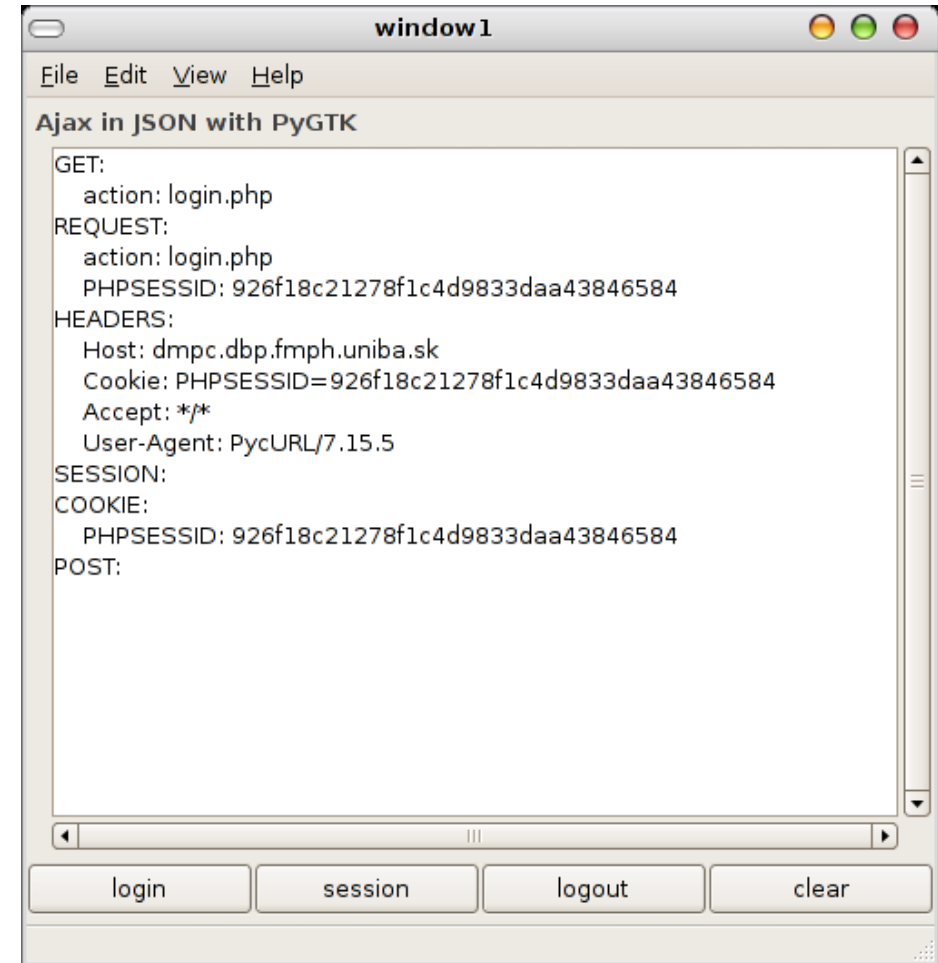
- REST a XML- a/alebo JSON-based API
- syndikácia, agregácia a notifikácia pomocou RSS alebo Atom feedov
- mashups, spájanie kontentu z rôznych zdrojov, klient- and server-side
- mashup ako nadmnožina starších portletov
- weblog-publishing tools
- wiki alebo fóra na podporu užívateľom generovaného obsahu

Weboví a neweboví klienti ku REST web službám



■ webový klient

■ newebový klient





Newwebový RESTovský klient

- Rozdiely od webového klienta:
 - Použitie iného programovacieho jazyka
 - Možnosť práce s dátami uloženými na lokálnom počítači
 - Väčšia diverzia v GUI
 - Možnosť vybrať si jazyk vyššej alebo nižšej úrovne
- Jazyky – Python, Java, Ruby, C++, ...



Newebový RESTovský klient

- Zhody s webovým klientom:
 - Rovnaký **protokol** na komunikáciu (HTTP) medzi klientom a serverom
 - Rovnaké webové služby na strane servera
 - **Jedna biznis logika** na serveri pre webových aj pre newebových klientov
 - Jednotné REST API pre klient-server komunikáciu
 - Rovnaký formát prijímaných a posielaných dát



tREST

výslovnosť podľa IPA: /'ti:rest/

Peter Rybár

Centaur s.r.o.

tREST – čo to je?

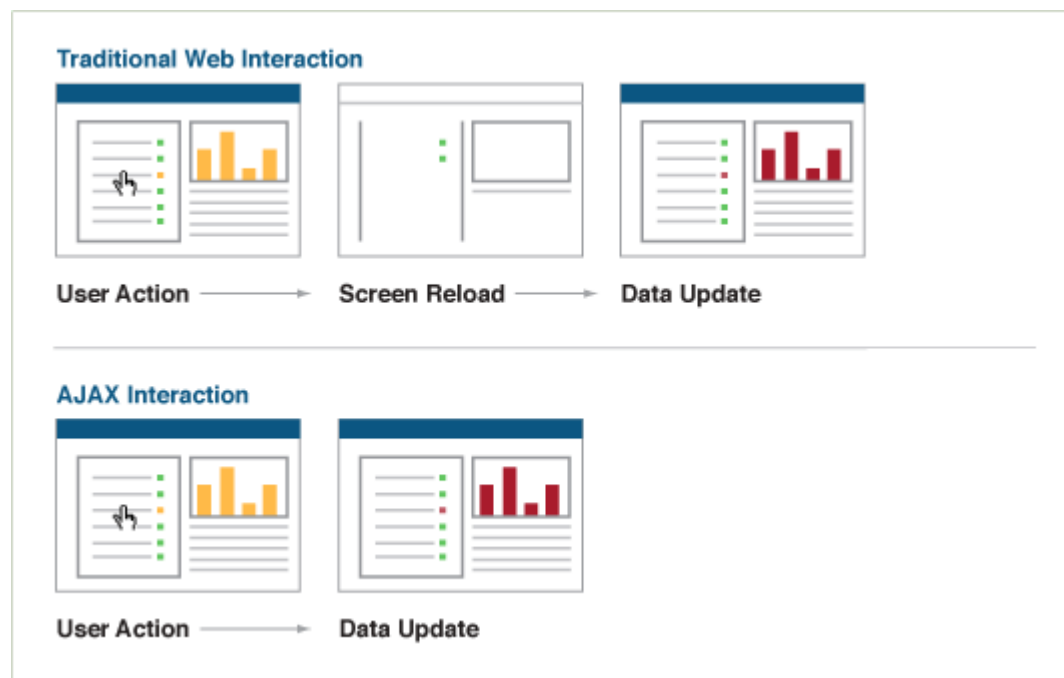
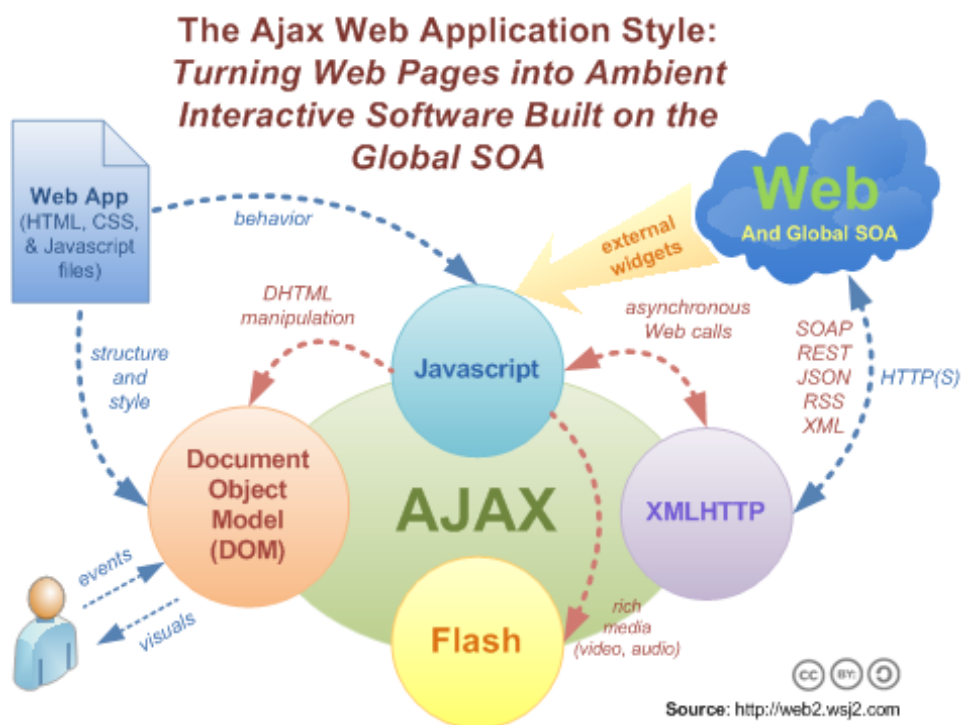
- **Web framework**
- Softvérový framework navrhnutý na vývoj
 - Webových aplikácií
 - Webových služieb
- Určený pre vývoj **architektonickým štýlom REST** (**R**epresentational **S**tate **T**ransfer)
- Kladie dôraz na:
 - Jednoduchosť vývoja
 - Efektivitu vývoja
 - Výkon

tREST – čo umožňuje?

■ Jednotný spôsob ako vytvárať

- Tradičné Web aplikácie

- AJAX Web aplikácie a Webové služby

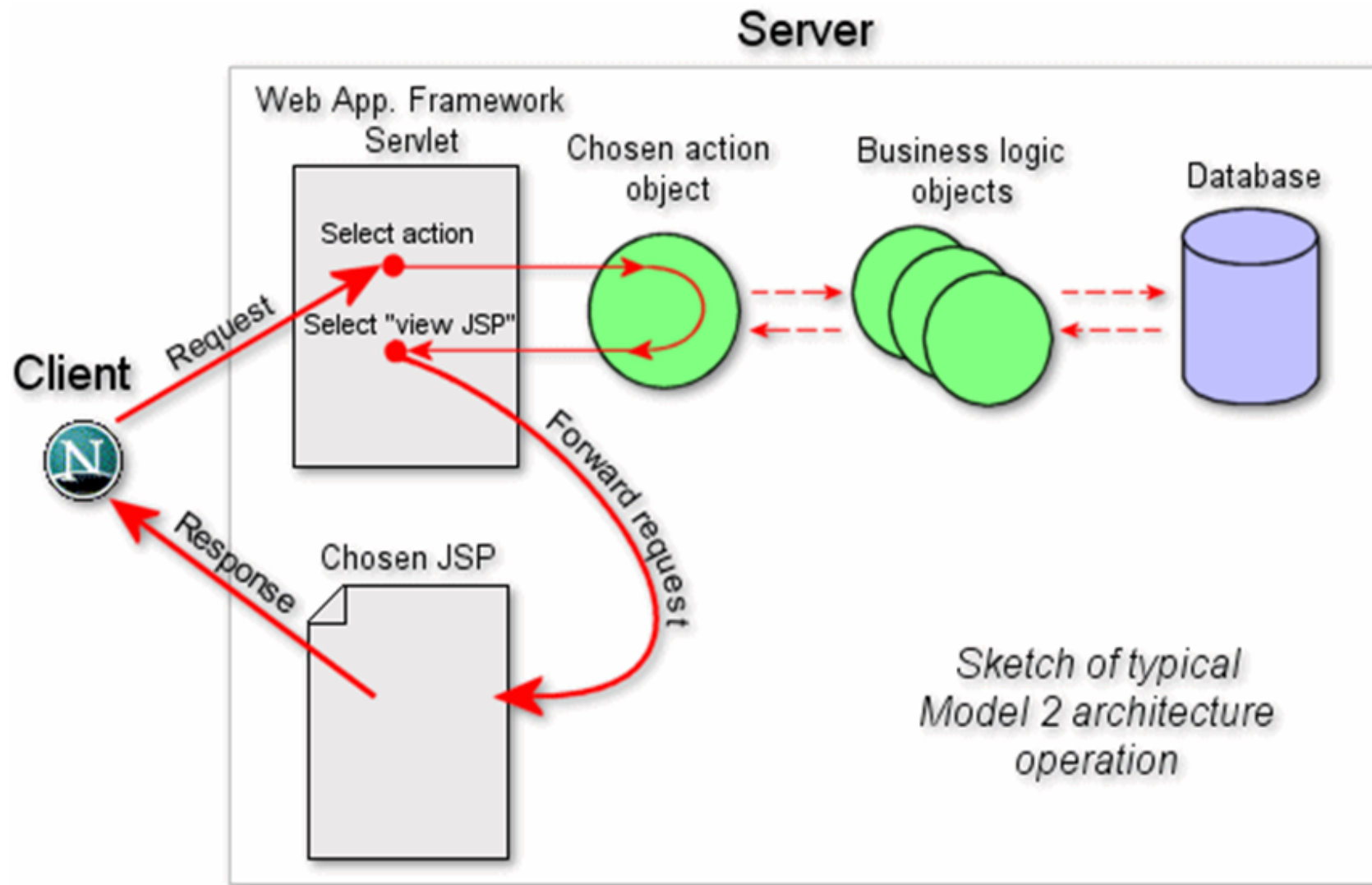




tREST – zameranie

- **Modularita** – modulárna architektúra
- **Extenzibilita** – jednoduchá možnosť integrácie iných technológií
- **Vývoj REST Web aplikácií a služieb** – AJAX, RIA (**R**ich **I**nternet **A**pplications)
- **Oddelenie vývoja**
 - prezentačnej vrstvy
 - serverového backendu
 - aplikačnej logiky

tREST – schéma architektúry





tREST – spĺňa poŹiadavky

- Minimálna doba nábehu vývojára do vývoja vo frameworku – čas rádovo v hodinách
- Horizontálny vývoj aplikácií – beŹný vývojár ovláda iba svoju doménu, nemusí ovládať všetky technológie naprieč aplikáciou, čoho dôsledkom je vyššia kvalitu kódu, efektivita
- Voľná väzba – nezávislosť frameworku od veľkého počtu technológií, stabilita
- Platformová nezávislosť – Java 5 a viac



tREST – dva komponenty

- Framework **pozostáva z dvoch komponentov**

- **Serverový komponent**

- **Kontróler** pre servletový kontajner so sadou rozšírení

- **Klientský komponent**

- **Javascript knižnica** – je možné ju použiť v kombinácii s ľubovoľnou technológiou na strane servera.



tREST – server

Peter Rybár

Daniel Buchta

Centaur s.r.o.



tREST – vlastnosti

- Implementuje vlastnosti:
 - **Mapovanie kontrolerov** – Java tried vystavujúcich funkcionality verejných metód
 - Dynamické **mapovanie URL parametrov**
 - Automatické **mapovanie Webových vstupov do natívnych dátových typov**
 - Automatická **serializácia výstupov z natívnych dátových typov**
 - Jednoduché **rozhranie pre implementáciu vlastných rozšírení** (extensions)

tREST – vlastnosti

- Deklaratívny prístup pri použití rozšírení realizovaný **Java anotáciami**
- Sada rozšírení modulárne doplňujúcich funkcionality kontrolera/filtra
- Rozšíriteľná sada validátorov validujúcich syntax a sémantiku **s podporou konverzie do natívnych dátových typov**
- Oddelenie prezentačnej vrstvy od aplikačnej logiky **pri zachovaní slabej väzby** vzhľadom k použitej prezentačnej technológii (JSP, FreeMarker, SiteMesh, ...)



tREST – vlastnosti

- Zabudovaná podpora interceptingu pre podporu aspektovo orientovaného programovania
- Podpora internacionalizácie (i18n) pri validácii dát
- Podpora pre **lineárny workflow**
- Možnosť dynamického pripájania a odpájania kontrolerov
- Role based user access management



tREST – vlastnosti

- Deklaratívny spôsob dokumentácie funkcionality pomocou Java anotácií priamo v kóde
- Webové rozhranie pre prístup k dokumentácii na základe Java reflexie
- Možnosť testovania implementovaných konrólerov, Webových služieb, s využitím vstavaného Webového rozhrania – vhodné najmä pre účely vývoja konrólerov

tREST Services documentation

Complete list of [service groups](#).

Complete list of [services](#) for current service group.

settings

Get map's settings

Service call input

```
get http://localhost:8180/gpsmon/administracia/mapa/data/settings
```

Parameters

Method

get

Header

[add](#)

URL parameters

Request parameters

Invoke

Service call output

Data

Status

200: OK

Header

Server: Apache-Coyote/1.1
Content-Type: text/json;charset=UTF-8
Content-Length: 462
Date: Mon, 26 May 2008 17:38:59 GMT

Payload

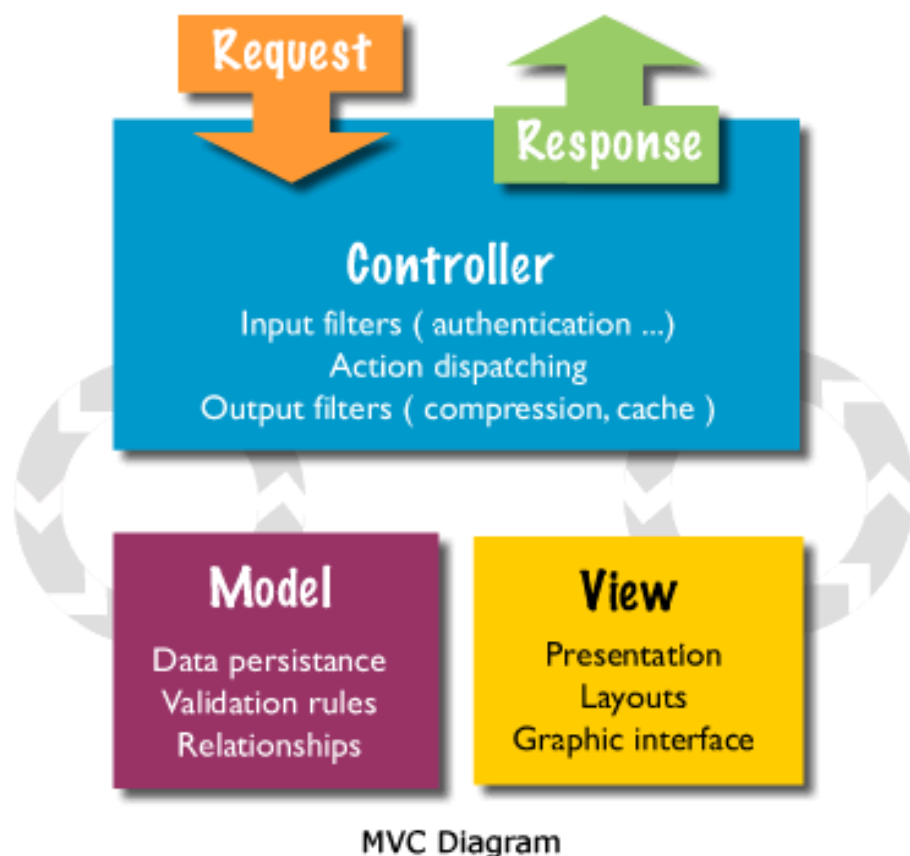
```
{"armaturny_uzol":["true","true","false","false","true","true","false","false","true","true"],"bod_zaujmu":["false"
```



tREST – rozšírenia

- Access controll
 - OpenID authentication
 - Web flow
 - Serializátori
 - Validátory
 - ...
-
- **tREST-client** – Javascript client side libs

tREST – kontrolér



■ Kontrolér

- Základná funkčná jednotka
- Extenduje vždy spoločného predka (*trest.core.Controller*)
- Alebo jeho potomka (napríklad *WebController*)
- *Multiaction* – nie komand
- *Akcje* sú mapované na verejné metódy kontroléra

```
public class PartsDepotController extends WebController {  
  
}
```

tREST – mapovanie metód

- Metóda kontrolera je mapovaná na časť **URL** prislúchajúcu akcii a HTTP metóde
 - http://<server>/<app_root>/<controller_path>/action
- Píklad:
 - http://server.net/company/depot/parts

tREST – mapovanie vstupov

■ Vstup z webu len v textovej podobe

□ **URL parametre:** <uri>/param1/param2/.../paramN

■ Príklad:

http://server.net/company/depot/parts/p1/p2/p3

□ **GET:** <uri>?key1=value1&key2=value2

■ Príklad:

http://server.net/company/depot/parts?k1=v1&k2=v2

□ **POST:** uri encoded v HTTP body

■ key1=value1&key2=value2

tREST – mapovanie vstupov

- Vstupy – parametre s verejným stringovým konštruktorom

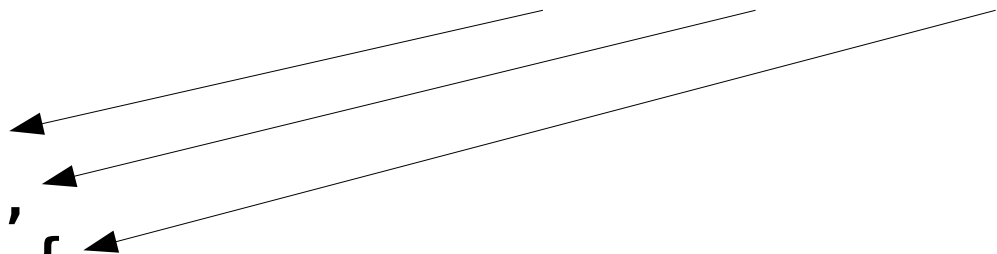
http://localhost/tutorial/input/url_params/str/30.126/xyz

```
public void url_params(String s, Double d, MyType m) {  
}
```

Three arrows point from the URL segments to the method parameters: from 'str' to 'String s', from '30.126' to 'Double d', and from 'xyz' to 'MyType m'.

http://localhost/tutorial/input/req_params?k1=str&k2=37&k3=xyz

```
public void req_params(  
    @Key("k1") String s,  
    @Key("k2") Integer i,  
    @Key("k3") MyType m) {  
}
```

Three arrows point from the query parameters to the method parameters: from 'k1=str' to '@Key("k1") String s', from 'k2=37' to '@Key("k2") Integer i', and from 'k3=xyz' to '@Key("k3") MyType m'.

tREST – anotácie

- `@Doc`, `@DefaultAction`, `@Action`, `@Key`

```
@Doc(value="Documentation for class Annotations")
public class Annotations extends WebController {

    @Doc("Method with method documentation")
    public void method_doc(@Doc("Attribute") String s) {}

    @DefaultAction
    public String default_action() {return "default"}

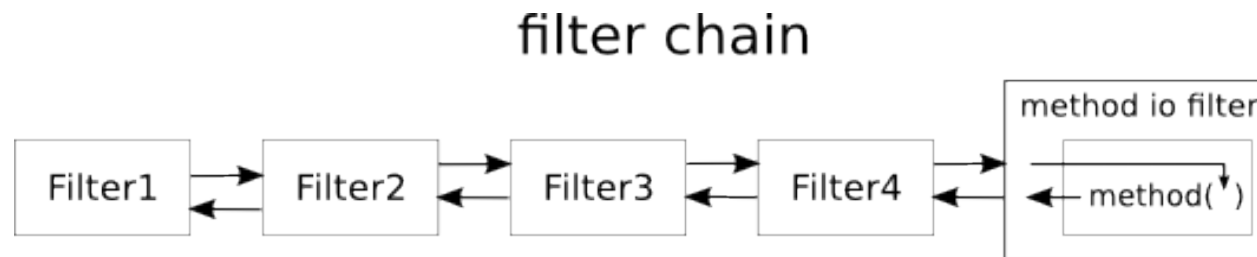
    @Action(name = "product", httpMethod = "DELETE")
    public void deleteProduct(int id) {}

    public void params_anoatation(@Key("age") int age) {}
}
```

tREST – validácia vstupov

- Veľmi dôležitá – **bezpečnosť**
- potrebujeme validovať dva druhy vstupov
 - **jednotlivé vstupy** (webové služby)
 - **celý formulár** (webové aplikácie)
- dve fázy validácie
 - **syntaktická** – prevod textového reťazca na požadovaný typ
 - **sémantická** - napr. kontrola veľkosti čísla a pod.

tREST – I/O filtre



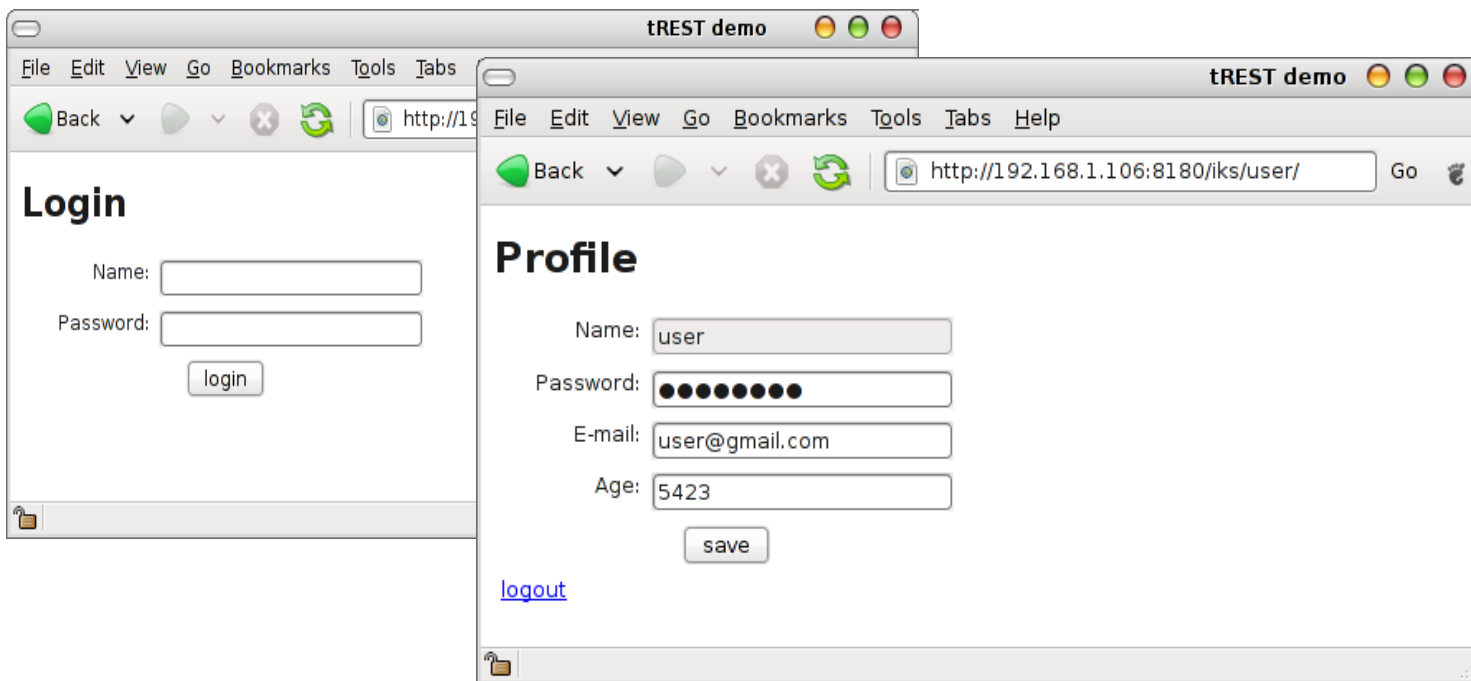
```
@Filter1
@Filter2("attribute")
public class FiltersOrder {

    @Filter3(param = "value")
    @Filter4
    public String method() {
        return "xyz";
    }

}
```

tREST – demo aplikácia

- Stránka s formulárom – profil používateľa
- Validácia vstupných dát
- JSP ako prezentačná vrstva
- RBAC – riadenie prístupu na základe rolí



tREST – štruktúra projektu

```
demo
|-- docs
|-- lib
|   |-- trest.jar
|-- src
|   |-- demo
|   |   |-- DemoApplication.java
|   |   |-- controllers
|   |       |-- ProfileController.java
|-- web
|   |-- META-INF
|   |-- WEB-INF
|   |   |-- web.xml
|   |-- index.html
|-- build.properties
-- build.xml
```

tREST – web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!DOCTYPE web-app PUBLIC
4   "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
5   "http://java.sun.com/dtd/web-app_2_3.dtd" >
6
7 <web-app>
8   <display-name>Archetype Created Web Application</display-name>
9   <session-config>
10     <session-timeout>30</session-timeout><!-- 30 minutes -->
11   </session-config>
12   <filter>
13     <filter-name>ServletFilter</filter-name>
14     <filter-class>trest.core.ServletFilter</filter-class>
15     <init-param>
16       <param-name>Application</param-name>
17       <param-value>demo.DemoApplication</param-value>
18     </init-param>
19   </filter>
20   <filter-mapping>
21     <filter-name>ServletFilter</filter-name>
22     <url-pattern>/*</url-pattern>
23   </filter-mapping>
24 </web-app>
```

tREST – aplikácia a mapovanie kontrolérov

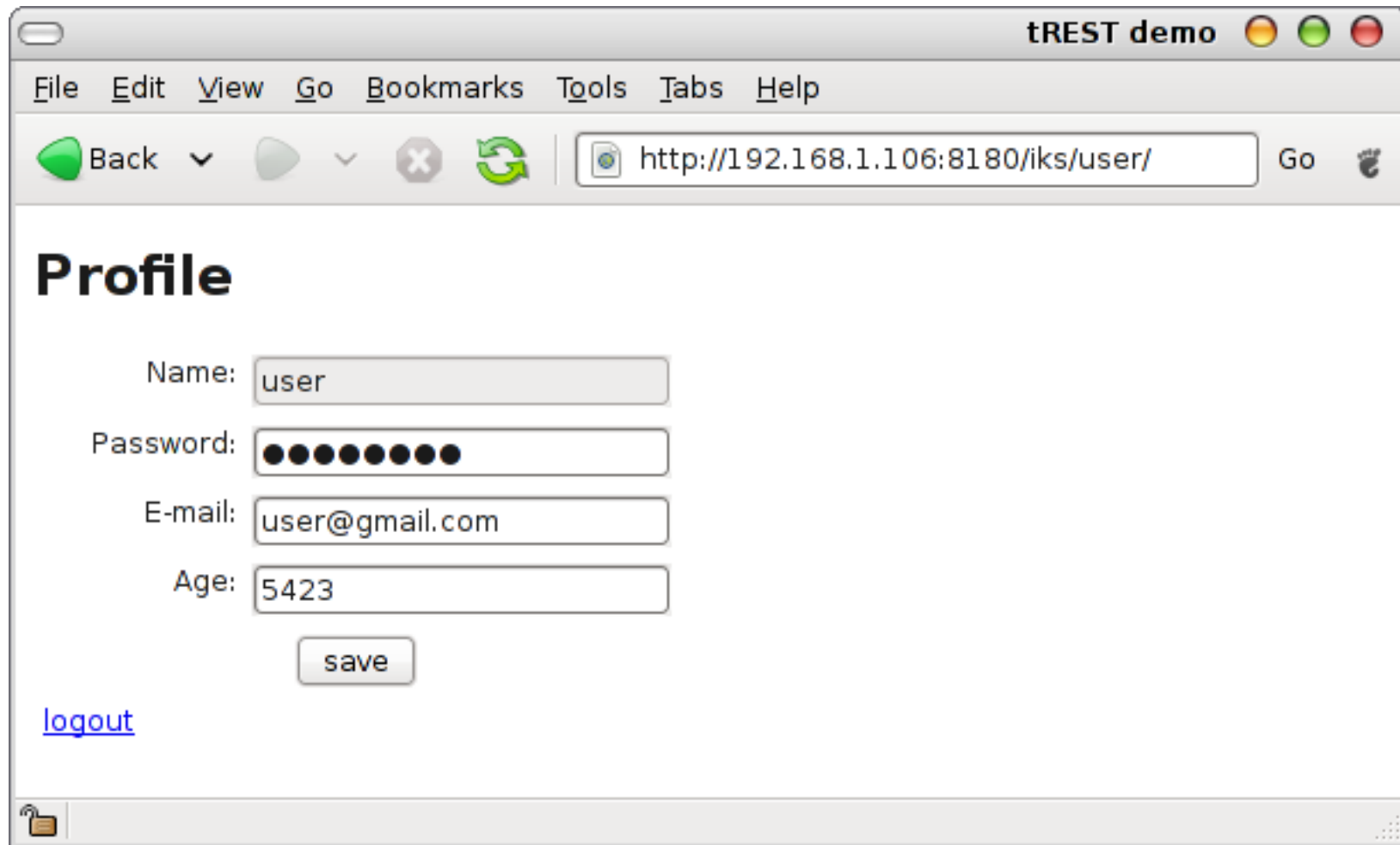
- *trest.core.Application* - reprezentuje tREST aplikáciu
- Jej úlohou je mapovanie kontrolérov

```
1 package demo;
2
3 import trest.core.Application;
4
5
6
7 public class DemoApplication extends Application {
8
9     @Override
10    public void initialize() throws ApplicationException {
11        mount("/user/", new ProfileController());
12    }
13
14 }
15
```

tREST – mapovanie I/O

```
1 package demo;
2
3 import trest.core.Controller;
4 import trest.core.annotations.Key;
5
6 public class ProfileController0 extends Controller {
7
8     public String profile() {
9         return "profile: ";
10    }
11
12    public String profile1(@Key("name") String name) {
13        return "profile: " + name;
14    }
15
16    public String profile2(Integer p1, @Key("p2") Float p2) {
17        return "profile: p1=" + p1 + " 2*p2=" + 2 * p2;
18    }
19
20 }
21
```

tREST – demo app, formulár



The screenshot shows a web browser window titled "tREST demo". The address bar displays the URL "http://192.168.1.106:8180/iks/user/". The main content area is titled "Profile" and contains a form with the following fields:

- Name:
- Password:
- E-mail:
- Age:

Below the form is a "save" button. A blue link labeled "logout" is located at the bottom left of the form area.

```

7
8 public class ProfileController1 extends Controller {
9
10     @Doc("Zobrazí formulář pro editaci profilu")
11     @DefaultAction
12     public String profile(
13         @Doc("meno pouzivatele") @Key("name") String name,
14         @Doc("heslo pouzivatele") @Key("password") String password,
15         @Doc("e-mailova adresa") @Key("email") String email,
16         @Doc("pouzivatelov vek") @Key("age") Long age)
17     {
18         name = name != null ? name : "";
19         password = password != null ? password : "";
20         email = email != null ? email : "";
21
22         String html = "<html>\n"
23             + "<head><title>tREST validation example</title></head>\n"
24             + "<body>\n"
25             + "<h1>Profile</h1>\n"
26             + "<form method=\"post\" action=\"\">\n"
27             + "    <div>\n"
28             + "        <label for=\"name\">Name:</label>\n"
29             + "        <input type=\"text\" id=\"name\" name=\"name\" value=\"" + name + "\" />\n"
30             + "    </div>\n"
31             + "    <div>\n"
32             + "        <label for=\"password\">Password:</label>\n"
33             + "        <input type=\"password\" id=\"password\" name=\"password\" value=\"" + password + "\" />\n"
34             + "    </div>\n"
35             + "    <div>\n"
36             + "        <label for=\"email\">E-mail:</label>\n"
37             + "        <input type=\"text\" id=\"email\" name=\"email\" value=\"" + email + "\" />\n"
38             + "    </div>\n"
39             + "    <div>\n"
40             + "        <label for=\"age\">Age:</label>\n"
41             + "        <input type=\"text\" id=\"age\" name=\"age\" value=\"" + (age != null ? age : "") + "\" />\n"
42             + "    </div>\n"
43             + "    <div>\n"
44             + "        <input type=\"submit\" id=\"submit\" name=\"submit\" value=\"save\" />\n"
45             + "    </div>\n"
46             + "</form>\n"
47             + "<div>\n"
48             + "</div>\n"
49             + "</body>\n"
50             + "</html>";
51
52         return html;
53     }
54 }

```

tREST

File Edit View Go Bookmarks Tools Tabs Help

Back http://192.168.1.106:8180/ks/us

Profile

Name:

Password:

E-mail:

Age:

[logout](#)

tREST – MVC, čisté riešenie

■ Odedelenie

- Model – dátovej reprezentácie
- View – prezentačnej vrstvy
- Controller – riadiacej aplikačnej logiky

```
41  
42 @Doc("Zobrazenie stranky s uzivatelovym profilom")  
43 @DefaultAction  
44 @View(template = "/templates/profile2.jsp")  
45 public Form profile(@Doc("formular profilom") ProfileForm form) {  
46     return form;  
47 }  
48 }
```

tREST – JSP ako view

```
137 <h1>Profile</h1>
138
139 <form id="profile_form" method="post" action="">
140     <% Form form = (Form) request.getAttribute("data"); %>
141
142     <div>
143         <label for="name">Name:</label>
144
145         <input type="text" id="name" name="name" readonly="readonly"
146             value="<%= form.getValidator("name").getValueString(0) %>" />
147
148         <span id="name_message" class="error">
149             <%= form.getValidator("name").getErrorMessage(0) %>
150         </span>
151     </div>
152
153     <div>
154         <label for="password">Password:</label>
155
156         <input type="password" id="password" name="password"
157             value="<%= form.getValidator("password").getValueString(0) %>" />
158
159         <span id="password_message" class="error">
160             <%= form.getValidator("password").getErrorMessage(0) %>
161         </span>
162     </div>
```

Profile

Name:

Password:

E-mail:

Age:

[logout](#)

tREST – definovanie formulára

```
1 package demo.forms;
2
3 import trest.validators.EmailValidator;
4
5
6
7
8
9 public class ProfileForm extends Form {
10
11     @Override
12     protected void addValidators() {
13         addValidator("name", new ProfileNameValidator());
14         addValidator("password", new StringValidator(6, 20));
15         addValidator("email", new EmailValidator());
16         addValidator("age", new LongValidator());
17     }
18 }
19
20 class ProfileNameValidator extends RegexpValidator {
21     public ProfileNameValidator() {
22         super("\\w{4,}");
23         setRegexpErrorMessage("at least 4 alphanumeric characters");
24     }
25 }
```

Profile

Name:

Password:

E-mail:

Age:

[logout](#)

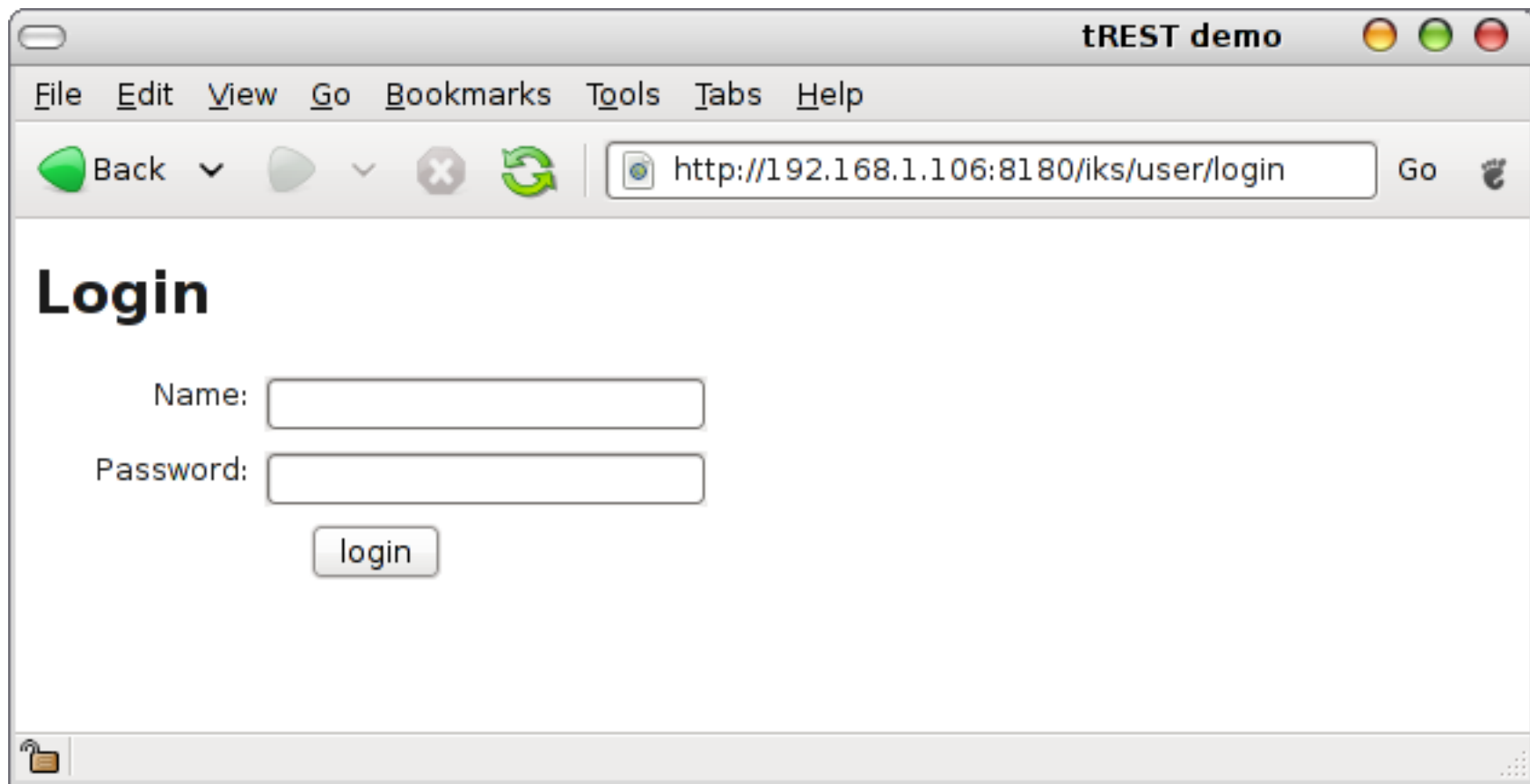
```
1 package demo.forms;
2
3 import java.util.Arrays;
4
5
6
7
8
9 public class ExtendedProfileForm extends ProfileForm {
10
11     public ExtendedProfileForm() {
12         super();
13     }
14
15     public ExtendedProfileForm(Profile profile) {
16         RequestParameters rp = new RequestParameters();
17         rp.add("name", Arrays.asList(profile.getName()));
18         rp.add("password", Arrays.asList(profile.getPassword()));
19         rp.add("email", Arrays.asList(profile.getEmail()));
20         rp.add("age", Arrays.asList(String.valueOf(profile.getAge())));
21
22         try {
23             validate(rp);
24         } catch (FormValidationException e) {
25             // form is not valid
26         }
27     }
28
29     public Profile getProfile() {
30         String username = (String) getValidator("name").getValue();
31         String password = (String) getValidator("password").getValue();
32         Long age = (Long) getValidator("age").getValue();
33         String email = (String) getValidator("email").getValue();
34
35         Profile result = new Profile(username, password, email, age);
36         return result;
37     }
38 }
39
```

```
1 package demo.model;
2
3 public class Profile {
4
5     private final String name;
6
7     private final String password;
8
9     private final String email;
10
11     private final long age;
12
13     public Profile(String name, String password, String email, long age) {
14         this.name = name;
15         this.password = password;
16         this.email = email;
17         this.age = age;
18     }
19
20     public long getAge() {
21         return this.age;
22     }
23
24     public String getEmail() {
25         return this.email;
26     }
27
28     public String getName() {
29         return this.name;
30     }
31
32     public String getPassword() {
33         return this.password;
34     }
35
36 }
37
```

tREST – logika + zabezpečenie

```
1 package demo;
2
3 import java.util.HashMap;
4
19
20 public class ProfileController extends Controller {
21
22     @Doc("Zobrazenie stranky s uzivatelskym profilom")
23     @DefaultAction
24     @Access(roles = "user")
25     @View(template = "/templates/profile.jsp")
26     public Form profile(
27         @Doc("formular s profilom") ExtendedProfileForm form,
28         @Doc("submit atribut") @Key("submit_button") String submit)
29     {
30         if (submit != null) {
31             Profile profile = form.getProfile();
32             saveProfile(profile);
33         } else {
34             User user = AccessManager.getInstance().getLoggedInUser();
35             Profile profile = loadProfile(user);
36             form = new ExtendedProfileForm(profile);
37         }
38
39         return form;
40     }
}
```

tREST – zabezpečenie, login



```

1 package demo;
2
3 import java.util.HashMap;
4
17
18 public class ProfileController3 extends Controller {
19
20     @Doc("Zobrazenie prihlasovacieho formulara")
21     @View(template = "/templates/login.jsp")
22     public Form login(@Doc("prihlasovaci formular") LoginForm loginForm)
23         throws RedirectException
24     {
25         if (loginForm.isValid()) {
26             String username = (String) loginForm.getValidator("name").getValue(0);
27             String password = (String) loginForm.getValidator("password").getValue(0);
28
29             Profile profile = Profiles3.get(username);
30
31             if (profile != null && profile.getPassword().equals(password)) {
32                 AccessManager am = AccessManager.getInstance();
33                 am.setLoggedInUser(new User(profile.getName(), "user"));
34                 am.followLoginReferer();
35                 redirect("profile");
36             }
37         }
38
39         return loginForm;
40     }

```

File Edit View Go Bookmarks Tools Tabs Help

Back <http://192.168.1.106:8180/iks/>

Login

Name:

Password:

```

97<h1>Login</h1>
98
99<form id="login_form" method="post" action="">
100    <% Form form = (Form) request.getAttribute("data"); %>
101
102    <div>
103        <label for="name">Name:</label>
104
105        <input type="text" id="name" name="name"
106            value="<%= form.getValidator("name").getValueString(0) %>" />
107
108        <span id="name_message" class="error">
109            <%= form.getValidator("name").getErrorMessage(0) %>
110        </span>
111    </div>
112
113    <div>
114        <label for="password">Password:</label>
115
116        <input type="password" id="password" name="password"
117            value="<%= form.getValidator("password").getValueString(0) %>" />
118
119        <span id="password_message" class="error">
120            <%= form.getValidator("password").getErrorMessage(0) %>
121        </span>
122    </div>
123
124    <div>
125        <input type="submit" id="next" name="next" value="login" />
126    </div>
127</form>
128

```

File Edit View Go Bookmarks Tools Tabs Help

Back http://192.168.1.106:81

Login

Name:

Password:



tREST – klient

Peter Rybár
Jozef Sivek

Centaur s.r.o.



tREST klient – vlastnosti

- **JavaScript knižnica** - implementuje:
 - Signál-Slot návrhový vzor
 - Objektové API
 - Vizuálnych komponentov (tabuľky, záložky, zoznamy, ...)
 - Formulárových komponentov
 - Formulárových validátorov
 - Podpora AJAX
 - Logovacie API

tREST klient – dedenie

- tREST client dedenie realizuje ako prebratie funkcionality od super typu.

```
function A(name) {this.name = name;}  
A.prototype.say_hello = function(){return this.name + " says hello";}
```

```
function B(name) {  
    A.call(this, name); //niečo ako zavolanie superkonštruktora  
}  
(new trest.Type(B)).extend(A);
```

```
B.prototype.say_hello_reverse = function() {  
    var txt = this.say_hello(), re = "";  
    for(var i = txt.length - 1; i >= 0; i--) re += txt.charAt(i);  
    return re;  
}
```

```
> var b = new B("Alice");  
> b.say_hello()  
olleh syas ecilA
```

tREST klient – Signal-Slot vzor

- **Signal-Slot** návrhový vzor je spôsob ako implementovať **Observer pattern**
- Ponúka väčší potenciál ako „callback“.
- Originálna koncepcia tohto vzoru pochádza z GUI knižnice QT a výborne sa hodí pri realizácii logiky „front-end“ aplikácie.
- Koncept spočíva v tom, že objekty (tiež nazívané "widgets") môžu posielat' signály obsahujúce potrebnú informáciu, ktoré sú prijímané funkciami (slotmi).

tREST klient – Signal-Slot vzor

- **Widget** má schopnosť **emisie signálov**
- **Signal objekt** (**signal_.***) sa stará o pripojenie, odpojenie a notifikáciu **slotov**

```
var input = new trest.widgets.forms.TextInput("input_id");
```

```
var receiver = {  
    slot_fnc: function(value) {  
        alert("new value is: " + value);  
    }  
};
```

```
// slot je metoda objektu  
input.signal_change.connect(receiver, receiver.slot_fnc);
```

```
// slot je funkcia  
input.signal_change.connect(slot_fnc);
```

```

35 <script type="text/javascript">
36     trest.queue_load_event(init);
37
38     function init() {
39         // name
40         var name_input = new trest.widgets.forms.TextInput("name");
41         var name_validator = new trest.validators.RegexValidator(
42             /^[a-zA-Z]+$/, "Bad name format");
43         name_validator.signal_valid.connect(
44             function (value) {
45                 document.getElementById("name_message").innerHTML = "";
46             }
47         );
48         name_validator.signal_invalid.connect(
49             function (error_message, value) {
50                 document.getElementById("name_message").innerHTML = error_message;
51             }
52         );
53
54         // password
55         var password_input = new trest.widgets.forms.TextInput("password");
56         var password_validator = new trest.validators.RegexValidator(
57             /^[0-9a-zA-Z]{6,20}$/, "Bad password format");
58         password_validator.signal_valid.connect(
59             function (value) {
60                 document.getElementById("password_message").innerHTML = "";
61             }
62         );
63         password_validator.signal_invalid.connect(
64             function (error_message, value) {
65                 document.getElementById("password_message").innerHTML =
66                     "Invalid value " + value + " (" + error_message + ")";
67             }
68         );
69
70         // form
71         var form_validator = new trest.validators.FormValidator();
72         // realtime validation
73         form_validator.add_form_object(name_input, name_validator, true);
74         // validate only on submit
75         form_validator.add_form_object(password_input, password_validator);
76
77         document.getElementById("login_form").onsubmit = function() {
78             return form_validator.validate();
79         }
80     }
81 </script>

```



tREST klient – princípy

- Využiť čo najviac silu JavaScript-u ako prototypovacieho objektového jazyka
- Vysoká „reusability“ a konektivita s okolím
- Účelná abstrakcia existujúceho DOM v predpripravených komponentách
- Jednoduchosť
- Nezávislosť na použitej serverovej technológii



tREST

Výkon a efektivita

Peter Rybár

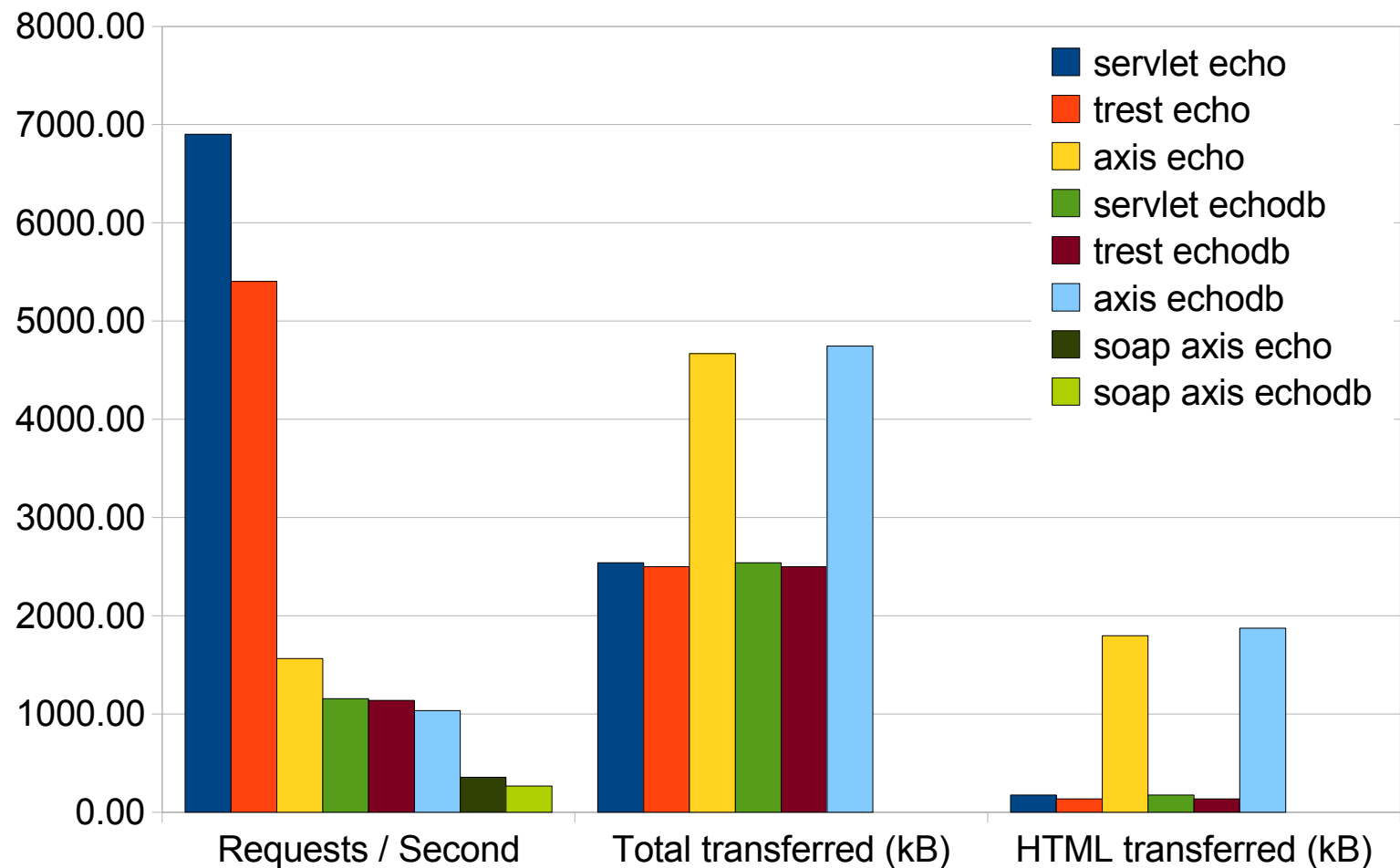
Centaur s.r.o.



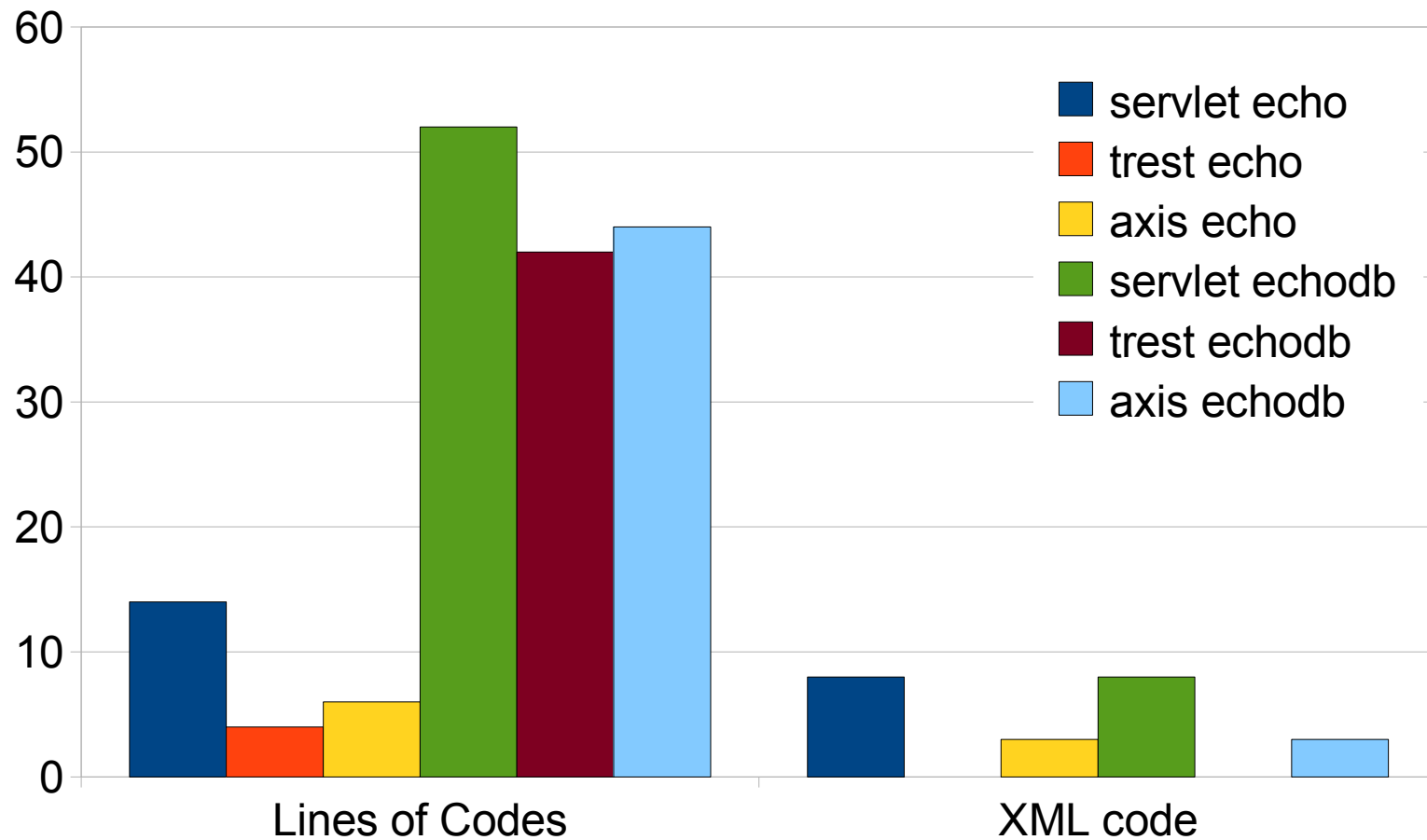
tREST – výkon, efektivita

- Programátor **najlepšie** rozumie kódu
- Kód píšeme v programovacom jazyku
- XML **nie je** programovací jazyk
- Generovaný kód **nie je** optimálny
- V generovanom kóde sa **t'ážko** hľadajú chyby a **ešte t'ážšie** opravujú
- Znovupoužitie kódu je najlepšie vo forme **knižníc**
- Špecifickosť kódu je nepriamo úmerná jeho opätovnému použitiu

tREST – výkon, efektivita, testy



tREST – výkon, efektivita, testy





tREST – výkon, efektivita

- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- Special cases aren't special enough to break the rules.



tREST

budúcnosť

Peter Rybár

Centaur s.r.o.



tREST – budúcnosť

- Tvorba rozšírení
 - Pre potreby web aplikácií
 - Pre potreby web služieb

- Implementácia REST riešení
 - Message bus – integrácia
 - Business process management
 - OpenID



Ďakujem

Otázky

Peter Rybár

peter.rybar@fmph.uniba.sk

peter.rybar@centaur.sk

<http://dmph.dbp.fmph.uniba.sk/~rybar/>