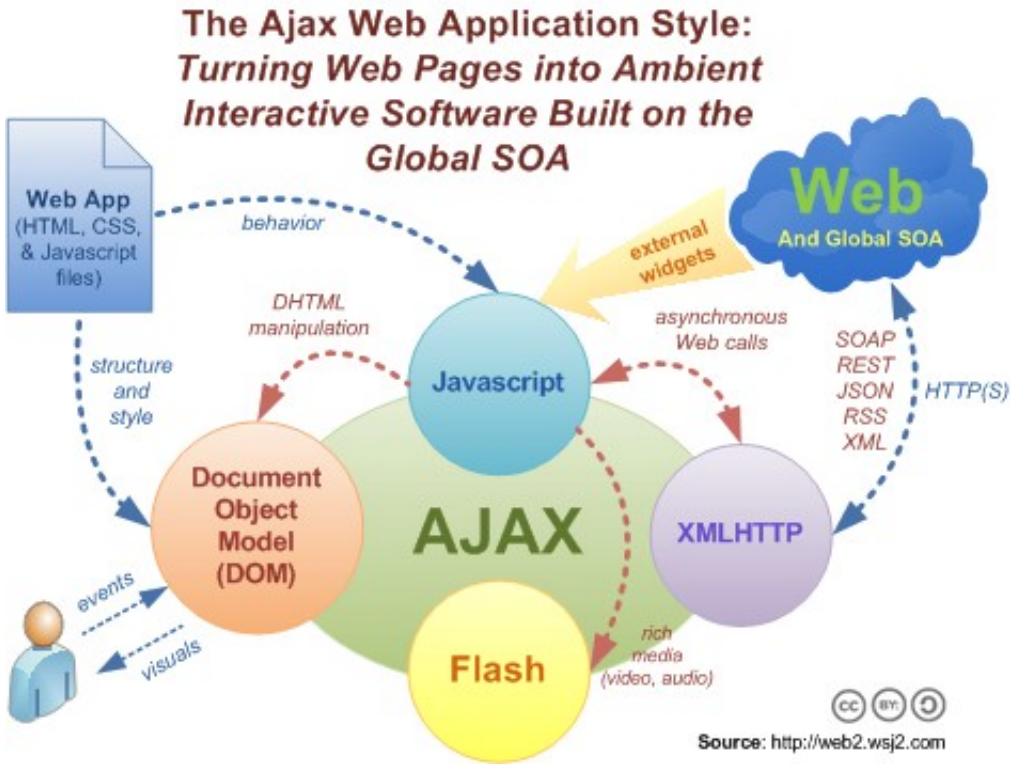
REST a tREST – architektúra a technológia

Autor: Peter Rybár <peter.rybar@fmph.uniba.sk>

V dnešnom IT svete opäť prišla doba, keď sa výrazne menia trendy v použití Webu (World Wide Web). Web ako najúspešnejšia distribučná platforma hypermediálneho obsahu na svete prechádza obdobím zmien, kde sa do popredia sa dostávajú najmä **Bohaté Webové aplikácie** (Rich Internet Applications – RIA, AJAX Web Applications) a **REST Web služby**. Spoločným znakom oboch je **REST**.



REST

Representational **S**tate **T**ransfer – **REST** je štýl softvérovej architektúry určený pre distribuované hypermediálne systémy ako je World Wide Web. Bol zavedený v dizertačnej práci Roya Thomasa Fieldinga "Architectural Styles and the Design of Network-based Software Architectures" (http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm). REST je určený aby uviedol spôsob, ako sa má správať dobre navrhnutá Webová aplikácia alebo služba. Z pohľadu RESTu je Webová aplikácia sieť Web stránok, virtuálnych stavových strojov, cez ktoré užívateľ prechádza prostredníctvom liniek (prenosov stavu), výsledkom čoho je vždy nasledujúca stránka, reprezentujúca nasledujúci stav aplikácie, prenesená k užívateľovi. REST striktne definuje kolekciu princípov sieťovej architektúry, ktoré popisujú ako sú definované a adresované internetové zdroje, ich URL, ich reprezentácie a linky medzi nimi. Tieto princípy pokrývajú hlavné ciele:

1. **Škálovateľnosť interagujúcich komponent** – rôznorodosť typov serverov a klientov
2. **Všeobecnosť rozhrania** – REST tvorí lepšiu základňou pre frameworky Webových služieb než technológie založené na SOAP
3. **Nezávislosť nasadenia komponent** – rôznorodosť implementácií serverov a klientov
4. **Kompatibilita s medziľahlými komponentami** na zníženie latencie, zvýšenie bezpečnosti a enkapsulácie – použitie Webových proxy a brán na enkapsuláciu nevebových systémov

REST dosiahol ciele aplikovaním štyroch obmedzení:

1. **Identifikácia zdrojov pomocou URI** – zdroje sú logické objekty identifikované URI
2. **Manipulácia so zdrojmi iba cez ich reprezentáciu** – zdroje nie sú priamo prístupné
3. **Samo-popisné správy** – REST diktuje že HTTP správa musí vyť čo najviac samo-popisná
4. **Hypermedia ako nositeľ stavu aplikácie** – aktuálny stav aplikácie musí byť zachytený v hypertextovom dokumente, reprezentácii

Hoci REST ako štýl softvérovej architektúry nie je štandard, silne podporuje používanie štandardov ako: HTTP, URL, XML/HTML/GIF/JPEG/.., MIME (text/xml, text/html, image/gif, image/jpeg, …).

REST filozofia háji existujúce princípy a protokoly Webu ako dostatočné pre tvorbu **robustných Webových služieb**. To znamená, že vývojári, ktorí rozumejú HTTP dokážu implementovať Webové služby bez potreby použitia iných nástrojov, než aké používajú normálne na vývoj Webových aplikácií.

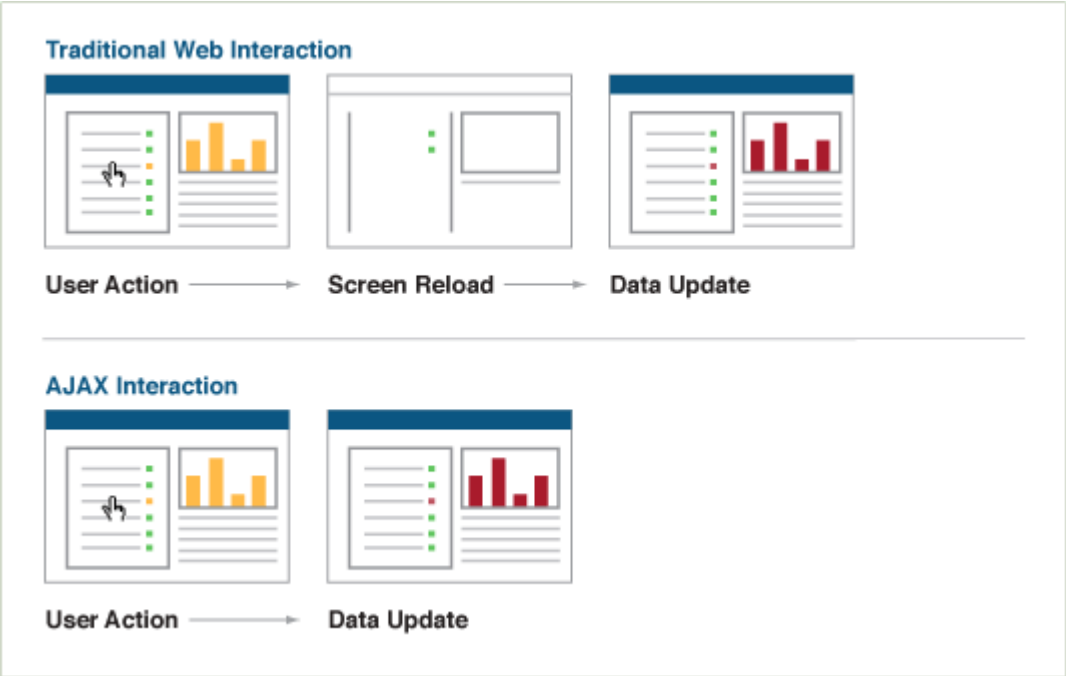
Kľúčom k REST metodológii je písať Webové služby použitím rozhraní, ktoré sú už dobre známe a rozšírené. Ľubovoľná klient, alebo server aplikácia s podporou HTTP dokáže jednoducho volať takéto služby.

Ako sme si už iste všimli, IT spoločnosti pôsobiace v oblasti Webu používajú pre svoje služby REST API už dosť dlho. Teda REST je rozhodne tendenčný spôsob ako vytvárať Webové služby jednoducho a efektívne. Hlavnou výhodou REST Web služieb je:

1. **Ľahkosť** (lightweight) – nevyžadujú žiaden extra XML markup v porovnaní so SOAP
2. **Zrozumiteľnosť pre ľudí** (Human Readable Results)
3. **Jednoduchosť vytvárania** (Easy to build) – nevyžadujú použitie extra nástrojov pre ich tvorbu

Väčšina spomedzi Webových služieb v súčasnosti publikovaných na Internete sú URI-based.

- Väčšina AJAX Web aplikácií nie je nič len klient v podobe Internetového prehliadača ku URI-based REST Web službám.
- Väčšina najväčších Webových aplikácií na Internete sú technicky na nerozoznanie od URI-based REST Web služieb.



Keby REST nefungoval, potom by nefungoval ani Web a Internet.

REST architektonický štýl teda predstavuje jednotnú bázu pre vývoj na funkcionalitu bohatých AJAX Web aplikácií (RIA), ktoré majú vlastnosti a funkcionalitu tradičných desktopových aplikácií. RIA prenáša spracovanie nevyhnutné pre užívateľské rozhranie na Web klienta, zatiaľ čo ponecháva dáta udržujúce stav aplikácie na strne aplikačného Web servera. Výhodou RIA je, že aplikácia s prvkami správania sa desktopovej aplikácie beží vo Web prehliadači a nevyžaduje inštaláciu softvéru.

Podstatný je aj fakt, že prezentačnú vrstvu RIA bežiacu v prehliadači môžeme kedykoľvek nahradiť rozhraním štandardnej desktopovej aplikácie, ktorá bude ako zdroj dát pre svoj chod využívať tie isté REST Web služby ako RIA (na rozdiel od databázovej konekcie ako sa to robilo za starých čias). Navyše v prípade takejto RDA (Rich Desktop Application) je výhodou, že aplikačná logika je z väčšej časti na serveri, kde sa lepšie implementuje a testuje.

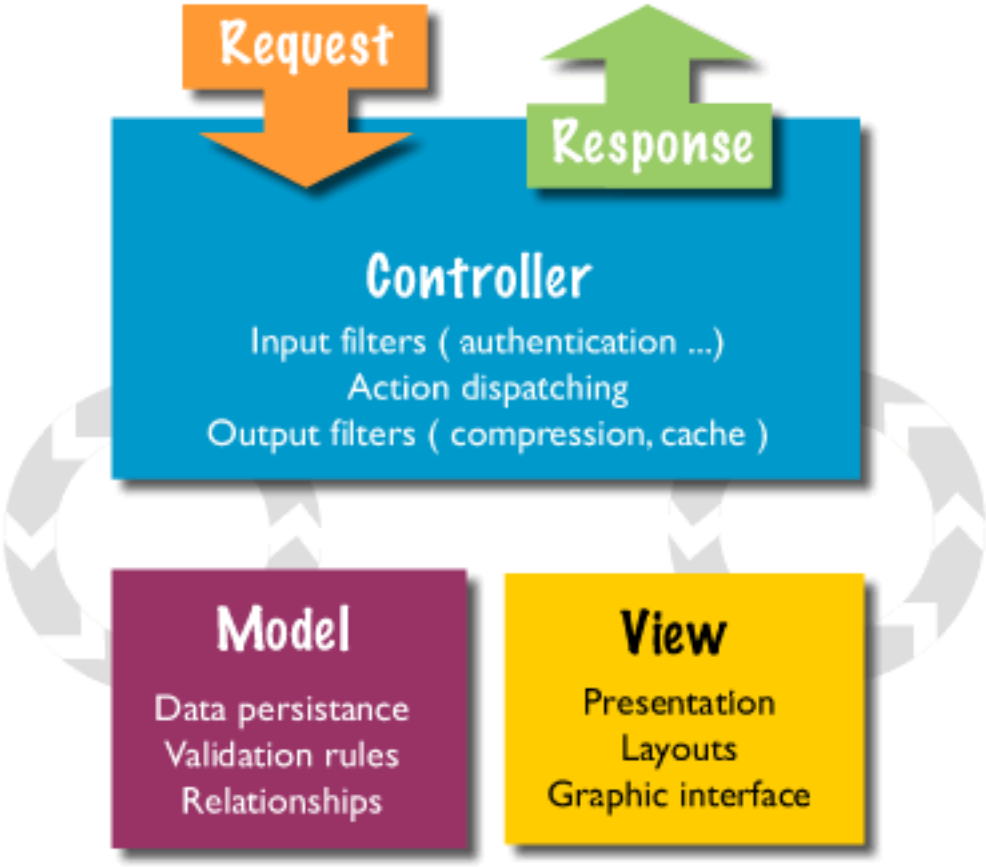
tREST – framework pre tvorbu RIA a Web služieb.

Už pár rokov sme pociťovali vo firme Centaur s.r.o. potrebu frameworku pre jednoduchú, tvorbu bohatých internetových a intranetových aplikácii, spolu s Webovými službami. Pre výber vhodného riešenia sme si definovali požiadavky ktoré by mal spĺňať framework vhodný pre naše potreby:

1. Modularita – modulárna architektúra frameworku
2. Extenzibilita – jednoduchá možnosť integrácie iných technológií
3. Oddelenie vývoja prezentačnej vrstvy od serverového backendu a aplikačnej logiky
4. Horizontálny vývoj aplikácii – aby bežný vývojár nemusel ovládať všetky technológie naprieč aplikáciou, od čoho očakávame vyššiu kvalitu kódu
5. Minimálna doba nábehu do vývoja v danom frameworku – očakávame čas rádovo v hodinách

6. Možnosť vývoja REST Web aplikácií a služieb - RIA
7. Nezávislosť frameworku od veľkého počtu technologii – jednoduchá údržba a stabilita počas dlhého obdobia
8. Framework by mal byť platformovo nezávislý – Java 5 a viac

Žiaľ, hoci existujú Webové frameworky ktoré do značnej miery spĺňajú nami definované požiadavky, nie

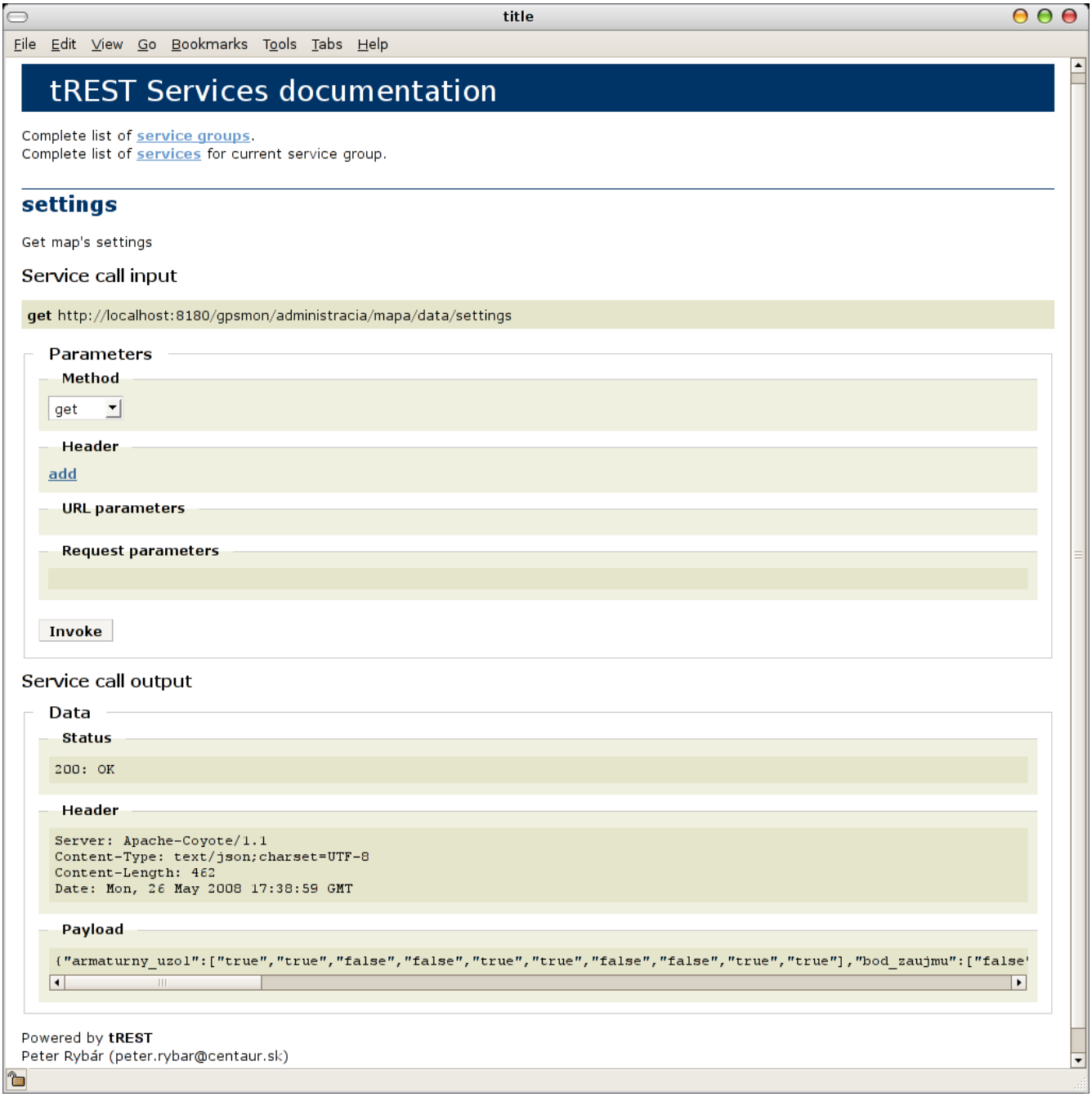


MVC Diagram

sú na platforme Java. Nové myšlienky a technológie sa skôr rodia na platformách ako Python, Ruby, ktoré dávajú programátorom viac priestoru pre realizáciu myslienok na úkor písania kódu (sú to dynamicky typované prototypovacie jazyky). Hoci technologicky sú ďaleko pred súčasnými Webovými riešeniami popredných softvérových korporácií, nie je možné ich použitie z marketingového hľadiska (náš zákazník náš pán), alebo kvôli tomu že ich stav nie je možné často krát označiť za produkčný. Kvôli uvedeným dôvodom sme pristúpili k implementácii vlastného frameworku vyvíjanému pod kódovým označením **tREST**. Rozhodli sme sa pre návrhový vzor MVC, ktorý dáva pri tvorbe Web aplikácie pocit ako pri vývoji desktopovej aplikácie. tREST Web framework pozostáva z dvoch komponent: serverovej a klientskej.

Serverová tREST komponenta je kontróler pre servletový kontajner určený pre efektívny vývoj Webových služieb a Webových aplikácií architektonickým štýlom REST. Implementuje nasledujúce vlastnosti:

1. Mapovanie kontrólerov
2. Mapovanie URL parametrov
3. Automatické mapovanie Webových vstupov do natívnych dátových typov
4. Automatická serializácia výstupov z natívnych dátových typov
5. Jednoduché rozhranie pre implementáciu vlastných rozšírení (extensions)
6. Deklaratívny prístup pre pri použití rozšírení realizovaný Java anotáciami
7. Sada rozšírení modulárne doplňujúcich funkcionalitu kontrólera/filtra
8. Rozšíriteľná sada validátorov validujúcich syntax a sémantiku s podporou konverzie do natívnych dátových typov
9. Oddelenie prezentačnej vrstvy od aplikačnej logiky pri zachovaní slabej väzby vzhľadom k použitej prezentačnej technológii (JSP, FreeMarker, SiteMesh, ...)
10. Zabudovaná podpora interceptingu pre podporu aspektovo orientovaného programovania
11. Podpora internacionalizácie (i18n) pri validácii dát
12. Podpora pre lineárny workflow
13. Možnosť dynamického pripájania a odpájania kontólerov (aj cez Web rozhranie)
14. Role based user access management
15. Deklaratívny spôsob dokumentácie funkcionality pomocou Java anotácií priamo v kóde
16. Webové rozhranie pre prístup k dokumentácii na základe Java reflexie
17. Možnosť testovania implementovaných konrólerov, Webových služieb, s využitím vstavaného Webového rozhrania



Klientská tREST komponenta je Javascript knižnica pre efektívnu implementáciu RIA. Je nezávislá na serverovej tREST komponente a je možné ju použiť v kombinácii s ľubovolnou technológiou na strane servera. Implementuje vlastnosti:

1. Objektové API ku vizuálnym komponentom (tabuľky, záložky)
2. Objektové API formulárových komponent a validátorov
3. Podpora AJAX
4. logovacie API

Terajší stav tREST považujeme za produkčný.

Produkčné nasadenie tREST

Spoločnosť Softec a Centaur v súčasnosti realizujú vo frameworku tREST projekt „Monitoring vozidiel“ pre spoločnosť SPP. Framework spĺňa požiadavky projektu, ktorý je svojou povahou ideálny pre nasadenie takejto technológie. Dúfame, že projekt prispeje k odhaleniu ďalších možností a vlastností, ktoré by pomohli doviesť tREST ako technológiu do vyššieho stupňa dokonalosti aby časom nahradil súčasné výbehové technológie.