tREST

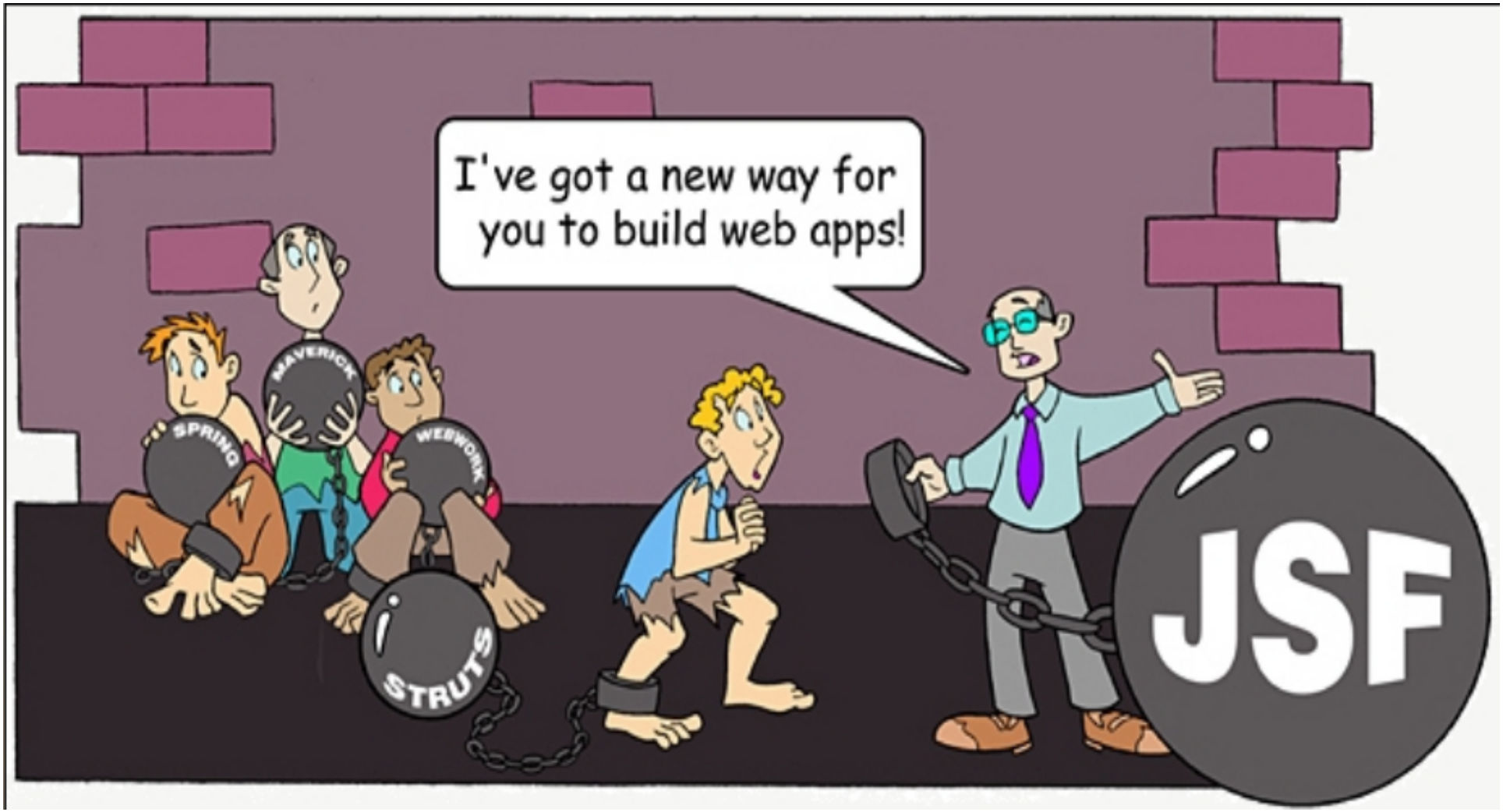
framework pre webové aplikácie a služby

Peter Rybár

Centaur s.r.o.

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/

CENTAUR

Ktorú cestu zvoliť?

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

CENTAUR

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”
  - Prečo? Lebo ho dokonale poznáte!
    - Snaha kupovať ľudí za technológiami
      Guido van Rossum, Anders Hejlsberg, …

tREST

výslovnosť podľa IPA: /**ˈtiːrest**/

Peter Rybár

Centaur s.r.o.

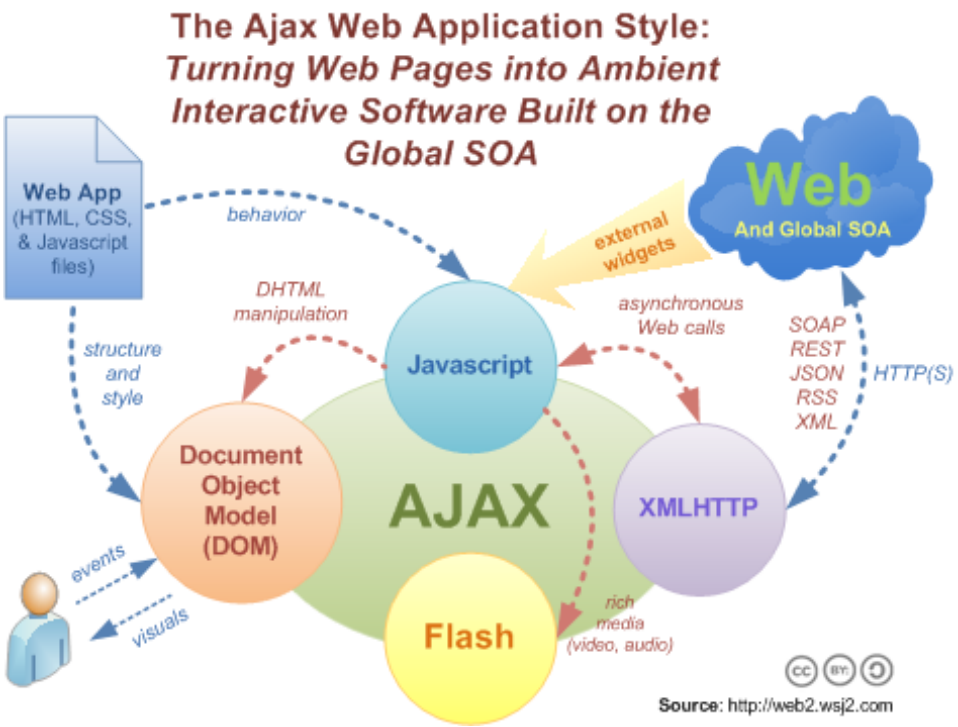
tREST – čo to je?

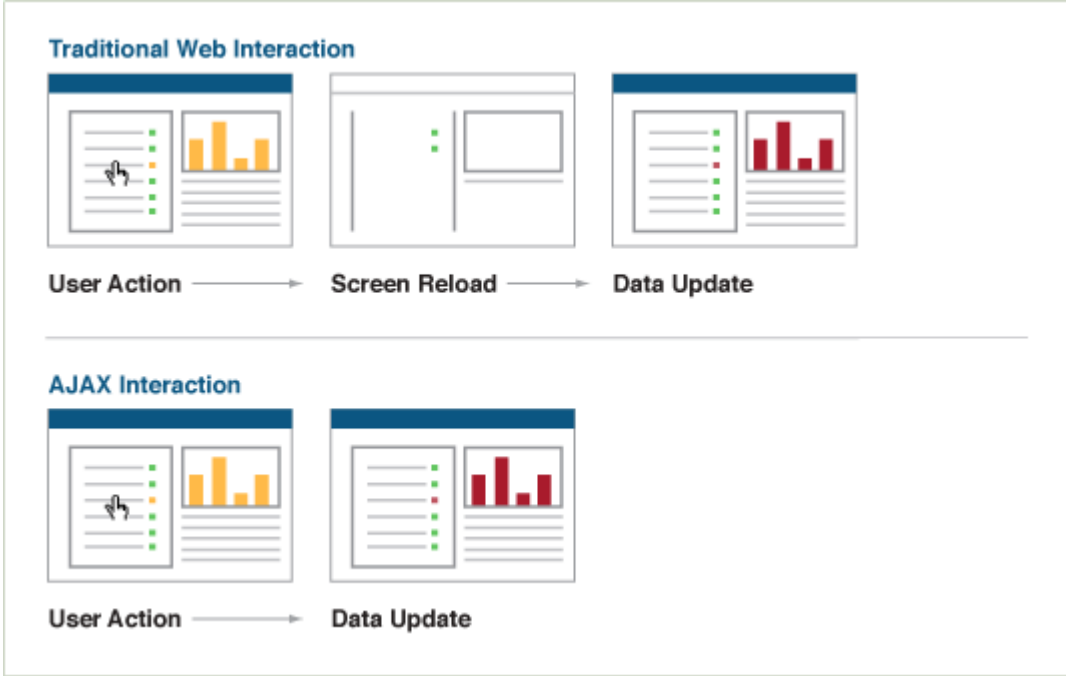
- Web framework
- Softvérový framework navrhnutý na vývoj
  - Webových aplikácií
  - Webových služieb
- Určený pre vývoj architektonickým štýlom **REST** (**Re**presentational **S**tate **T**ransfer)
- Kladie dôraz na:
  - Jednoduchosť vývoja
  - Efektivitu vývoja
  - Výkon

CENTAUR

tREST – čo umožňuje?

- Jednotný spôsob ako vytvárať
  - Tradičné Web aplikácie
  - AJAX Web aplikácie a Webové služby





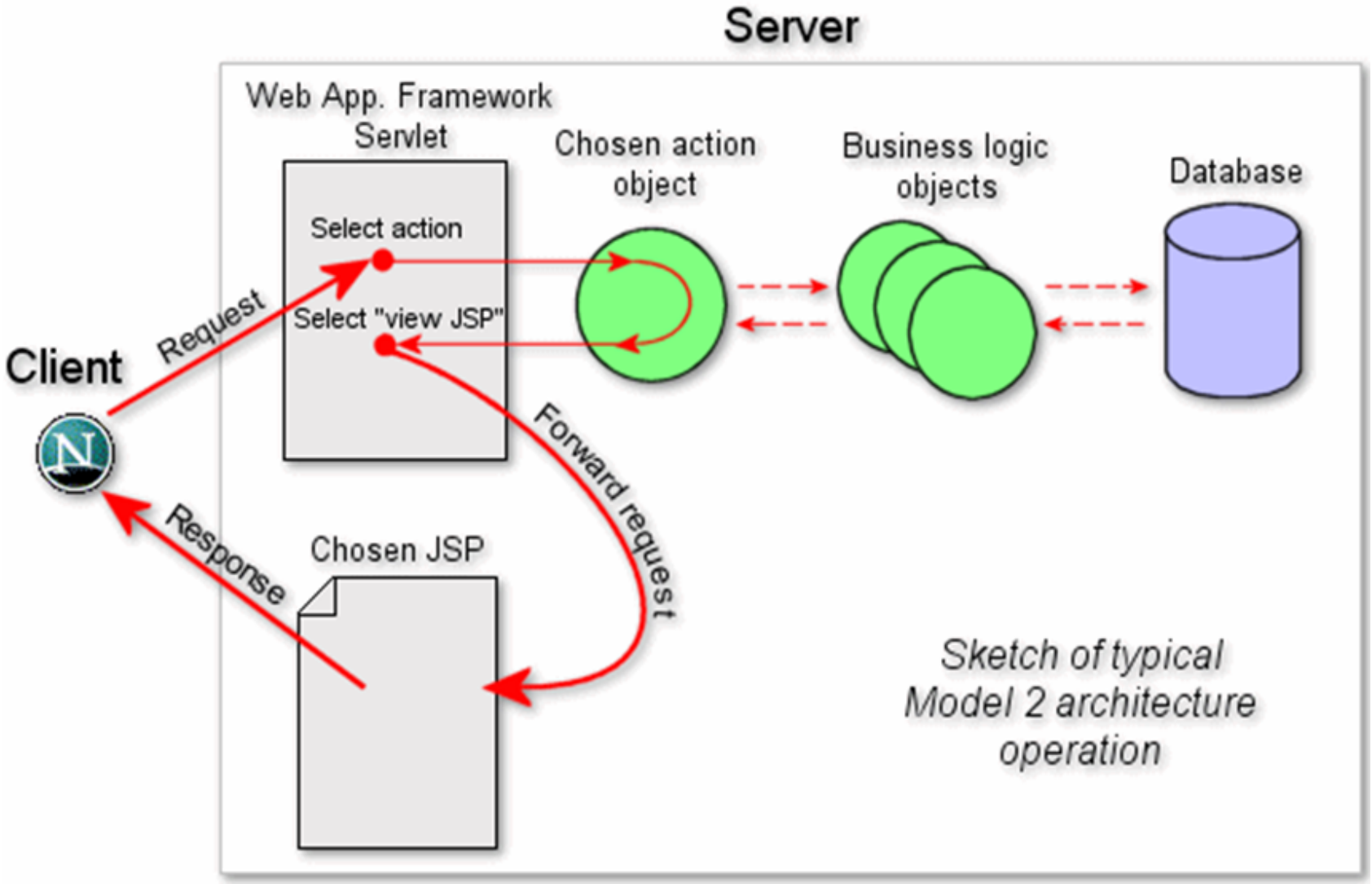
CENTAUR

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

CENTAUR

tREST – schéma architektúry



CENTAUR

tREST – spĺňa požiadavky

- Minimálna doba nábehu vývojára do vývoja vo frameworku – čas rádovo v hodinách
- Horizontálny vývoj aplikácií – bežný 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

CENTAUR

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 ľubovolnou technológiou na strane servera.

CENTAUR

tREST – rozšírenia

- Access controll
- OpenID authentication
- Web flow
- Serializátori
- Validátory
- ...

- **tREST-client** – Javascript client side libs

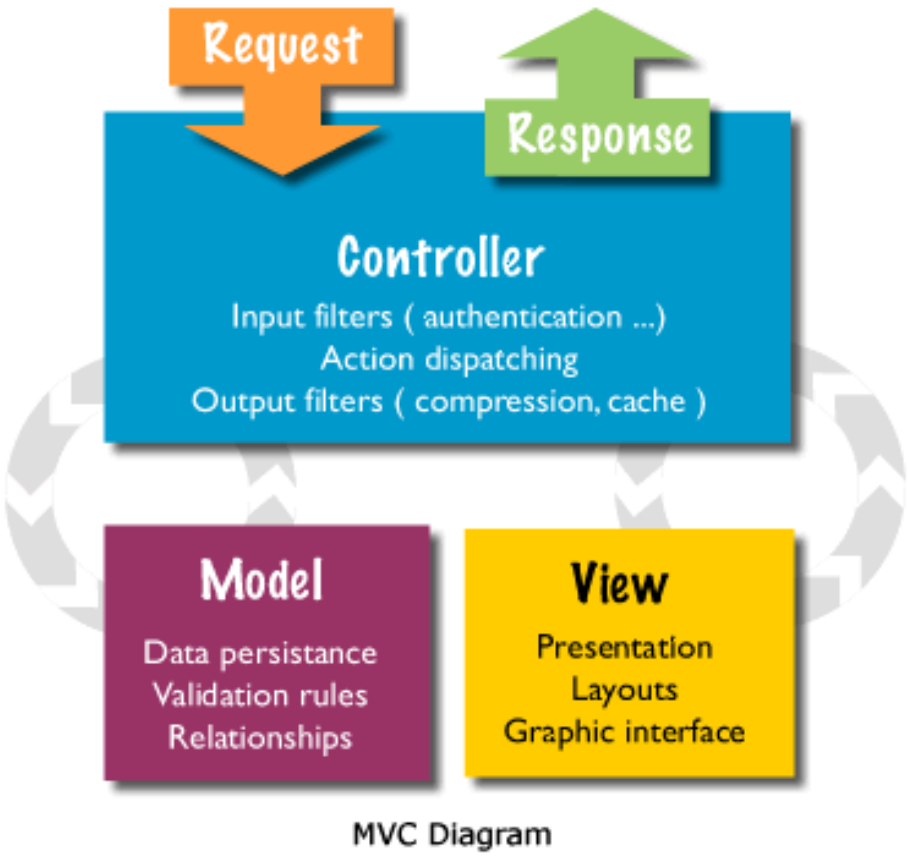
CENTAUR

tREST – server

Peter Rybár
Daniel Buchta

Centaur s.r.o.

tREST – kontróler



MVC Diagram

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

CENTAUR

tREST – mapovanie kontrólera

- Kontroler mapujeme na **URL** od rootu aplikácie
  - http://<server>/**<app_root>**/**<controller_path>**/
- Príklad:
  - http://server.net/**company/depot/**

```
mount("/depot/", new DepotController());
```

CENTAUR

tREST – mapovanie metód

- Metóda kontrólera 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

CENTAUR

tREST – mapovanie vstupov

- HTTP parametre sú mapované na **natívne dátové typy** parametrov metód:
  - Java typy so stringovým konštruktorom a ich polia
  - Java primitívne typy a ich polia
  - Špeciálne agregované typy implementujúce:
    - trest.core.types.InputType
    - trest.core.types.UrlParametersType
    - trest.core.types.RequestParametersType
    - trest.core.types.FileRequestParametersType

tREST – mapovanie vstupov

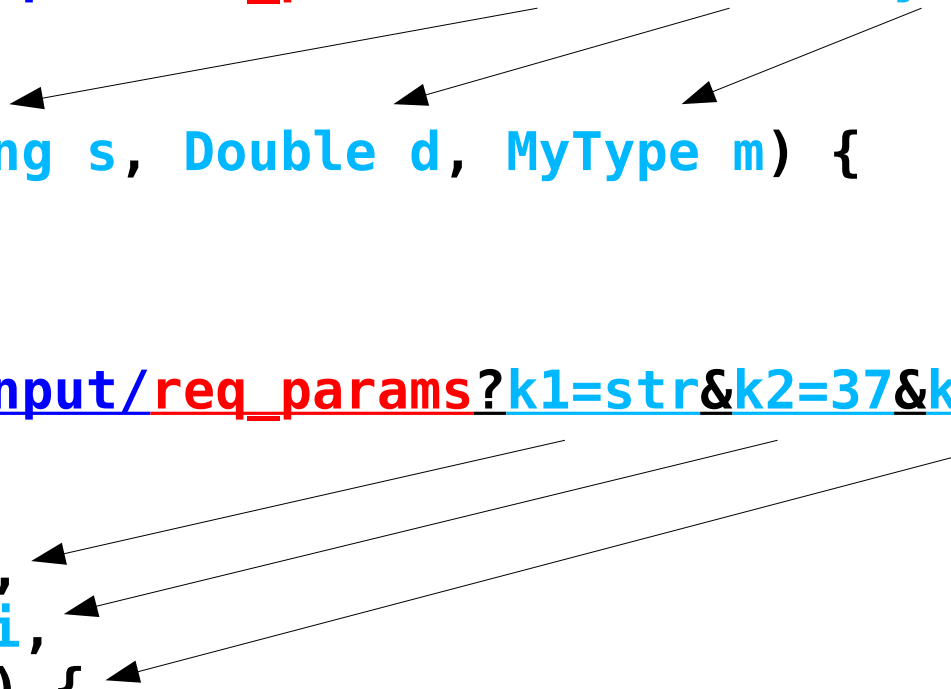
- Vstupy – parametre s verejným stringovým konštruktorom

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

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

```
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) {
}
```

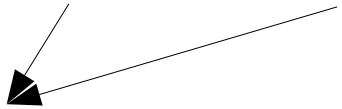


CENTAUR

tREST – mapovanie vstupov

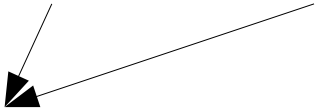
- Vstupy – polia typov s verejným stringovým konštruktorom
  - Iba request parametre – Multiple values

```
http://localhost/tutorial/input/req_params?k1=str1&k1=str2
```



```
public void req_params(@Key("k1") String[] s) {
}
```

```
http://localhost/tutorial/input/req_params?k1=3.14&k1=6.28
```



```
public void req_params(@Key("k1") Double[] d) {
}
```

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.

CENTAUR

tREST – spracovanie výstupov

- Telo HTTP response môže obsahovať buď textový, alebo binárny stream reprezentovaný
  - java.lang.String
  - java.io.Reader
  - Byte[]
  - java.io.InputStream
- Metóda kontrólera vracia ľubovoľný natívny Java objekt – do tela HTTP response zapíše stringová reprezentácia objektu získaná volaním metódy **toString()**

CENTAUR

tREST – výstupy – príklady

```
public Reader reader() {
    Reader reader = new StringReader("Hello. I'm Reader.");
    return reader;
}

public byte[] bytearray() {
    byte[] bs = "Hello. I'm byte[]".getBytes();
    return bs;
}

public MyObject stream() {
    MyObject object = new MyObject("text1", "text2");
    return object;
}
```

CENTAUR

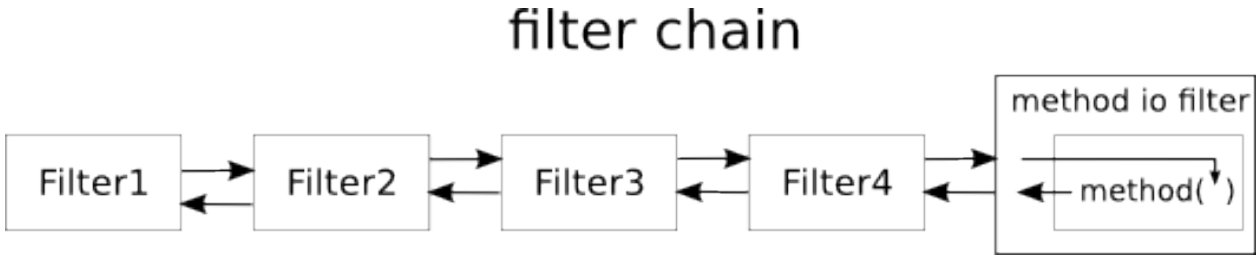
tREST – anotácie

- @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_anotation(@Key("age") int age) {}
}
```

CENTAUR

tREST – I/O filtre




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

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

}
```

CENTAUR

tREST

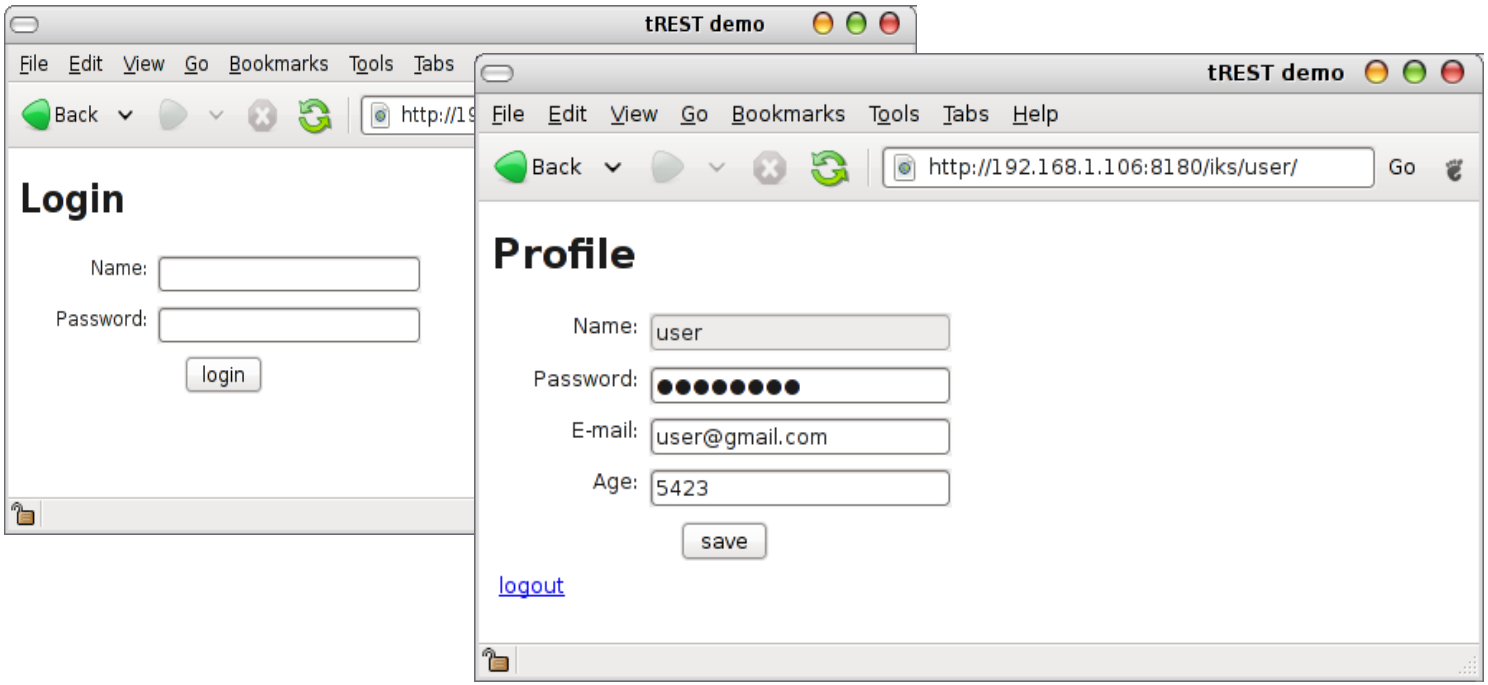
demo aplikácia

Peter Rybár
Daniel Buchta

Centaur s.r.o.

tREST – demo aplikácia

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



CENTAUR

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
```

CENTAUR

tREST – web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
    <display-name>Archetype Created Web Application</display-name>
    <session-config>
        <session-timeout>30</session-timeout><!-- 30 minutes -->
    </session-config>
    <filter>
        <filter-name>ServletFilter</filter-name>
        <filter-class>trest.core.ServletFilter</filter-class>
        <init-param>
            <param-name>Application</param-name>
            <param-value>demo.DemoApplication</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>ServletFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

CENTAUR

tREST – aplikácia a mapovanie kontrólerov

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

```
1  package demo;
2
3  import trest.core.Application;
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
```

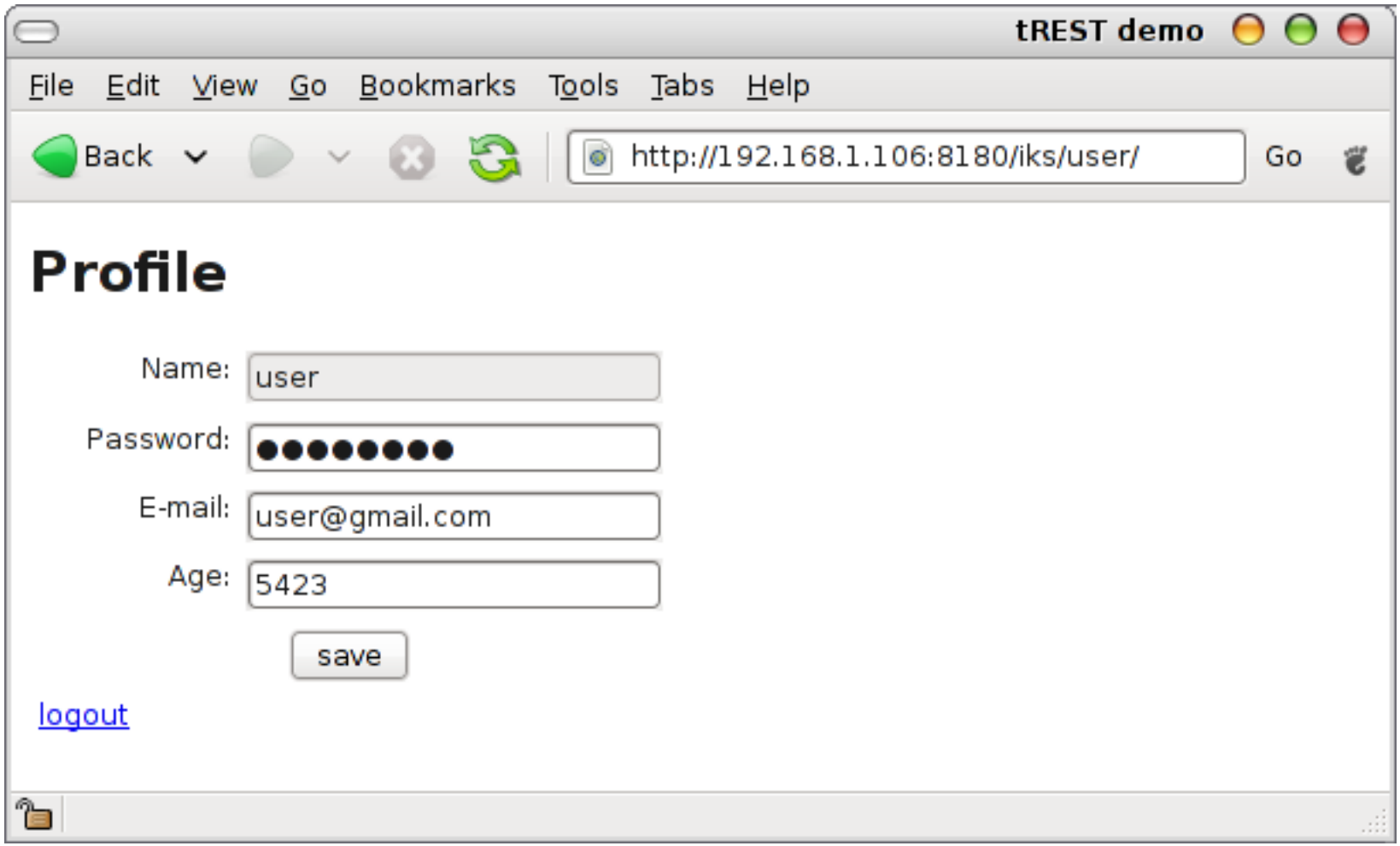
CENTAUR

tREST – mapovanie I/O

```java
package demo;

import trest.core.Controller;
import trest.core.annotations.Key;

public class ProfileController0 extends Controller {

    public String profile() {
        return "profile: ";
    }

    public String profile1(@Key("name") String name) {
        return "profile: " + name;
    }

    public String profile2(Integer p1, @Key("p2") Float p2) {
        return "profile: p1=" + p1 + " 2*p2=" + 2 * p2;
    }

}

```

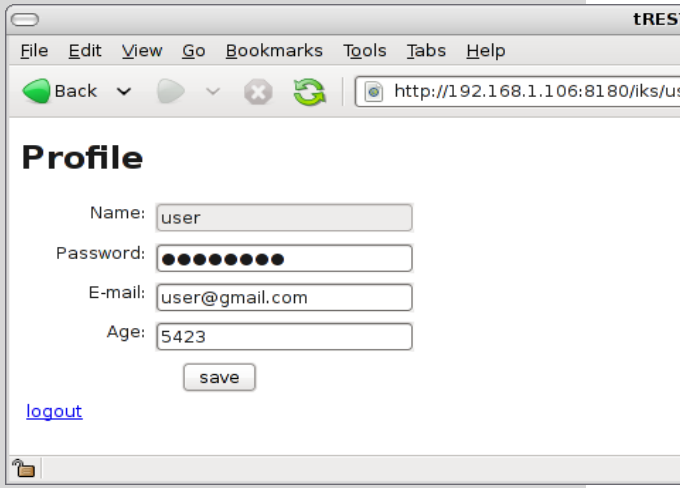
CENTAUR

tREST – demo app, formulár



CENTAUR

```
7
8  public class ProfileController1 extends Controller {
9
10     @Doc("Zobrazi formular pre editaciu profilu")
11     @DefaultAction
12     public String profile(
13             @Doc("meno pouzivatela")  @Key("name")     String name,
14             @Doc("heslo pouzivatela") @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                 + "</body>\n"
49                 + "</html>";
50
51         return html;
52     }
53 }
```



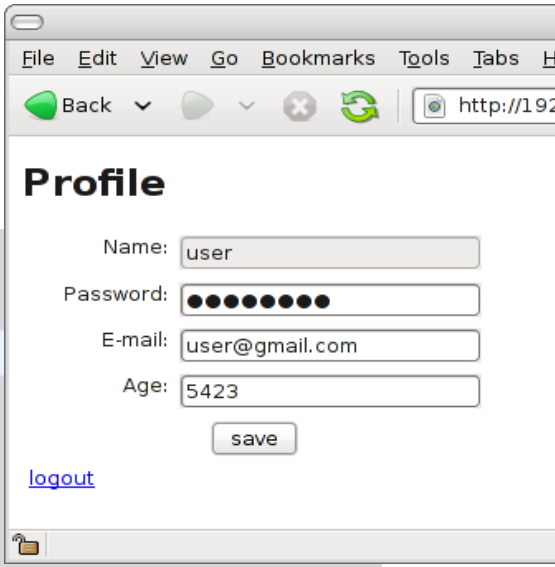
tREST – MVC, čisté riešenie

- Oddelenie
  - 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  }
```

CENTAUR

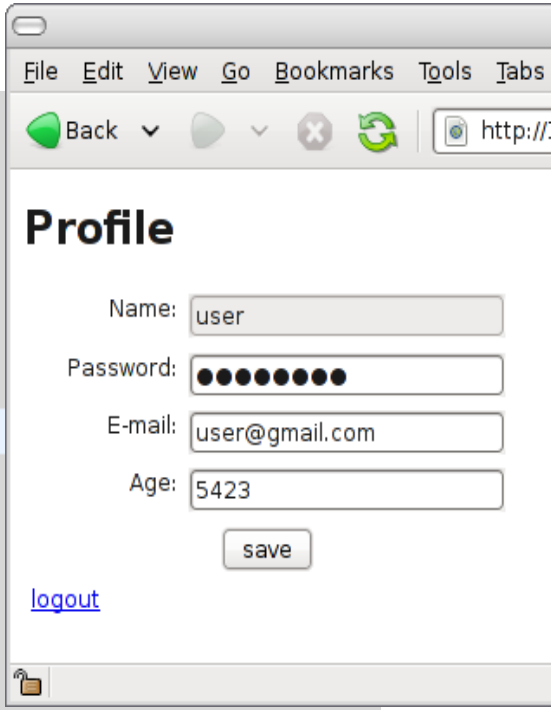
tREST – JSP ako view




```
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>
```

tREST – definovanie formulára

```java
package demo.forms;

import trest.validators.EmailValidator;

public class ProfileForm extends Form {

    @Override
    protected void addValidators() {
        addValidator("name", new ProfileNameValidator());
        addValidator("password", new StringValidator(6, 20));
        addValidator("email", new EmailValidator());
        addValidator("age", new LongValidator());
    }
}

class ProfileNameValidator extends RegexpValidator {
    public ProfileNameValidator() {
        super("\\w{4,}");
        setRegexpErrorMessage("at least 4 alphanumeric characters");
    }
}
```



CENTAUR

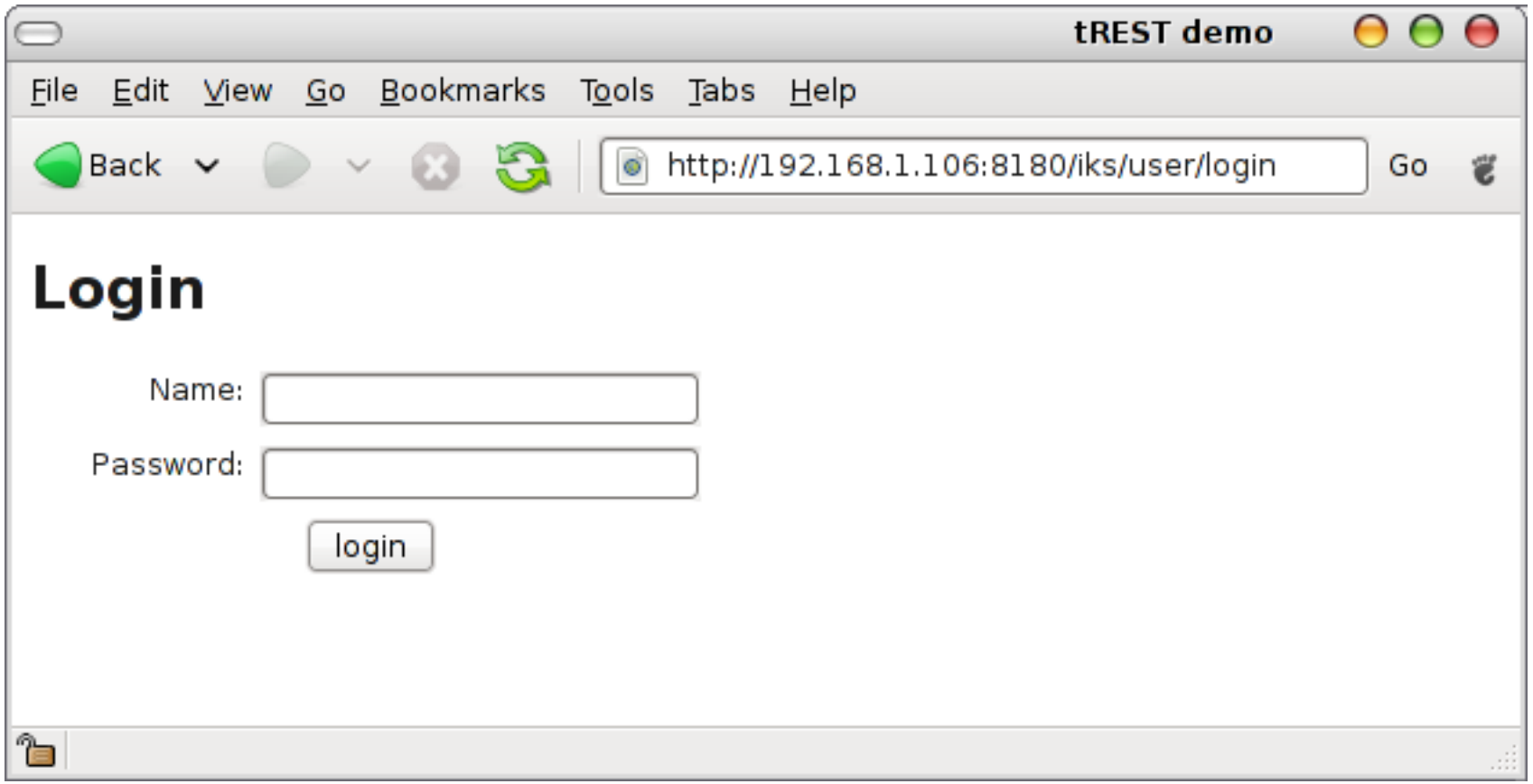
```java
package demo.forms;

import java.util.Arrays;

public class ExtendedProfileForm extends ProfileForm {

    public ExtendedProfileForm() {
        super();
    }

    public ExtendedProfileForm(Profile profile) {
        RequestParameters rp = new RequestParameters();
        rp.add("name", Arrays.asList(profile.getName()));
        rp.add("password", Arrays.asList(profile.getPassword()));
        rp.add("email", Arrays.asList(profile.getEmail()));
        rp.add("age", Arrays.asList(String.valueOf(profile.getAge())));

        try {
            validate(rp);
        } catch (FormValidationException e) {
            // form is not valid
        }
    }

    public Profile getProfile() {
        String username = (String) getValidator("name").getValue(0);
        String password = (String) getValidator("password").getValue(0);
        long age = (Long) getValidator("age").getValue(0);
        String email = (String) getValidator("email").getValue(0);

        Profile result = new Profile(username, password, email, age);
        return result;
    }
}
```

```java
package demo.model;

public class Profile {

    private final String name;

    private final String password;

    private final String email;

    private final long age;

    public Profile(String name, String password, String email, long age) {
        this.name = name;
        this.password = password;
        this.email = email;
        this.age = age;
    }

    public long getAge() {
        return this.age;
    }

    public String getEmail() {
        return this.email;
    }

    public String getName() {
        return this.name;
    }

    public String getPassword() {
        return this.password;
    }

}
```

tREST – logika + zabezpečenie

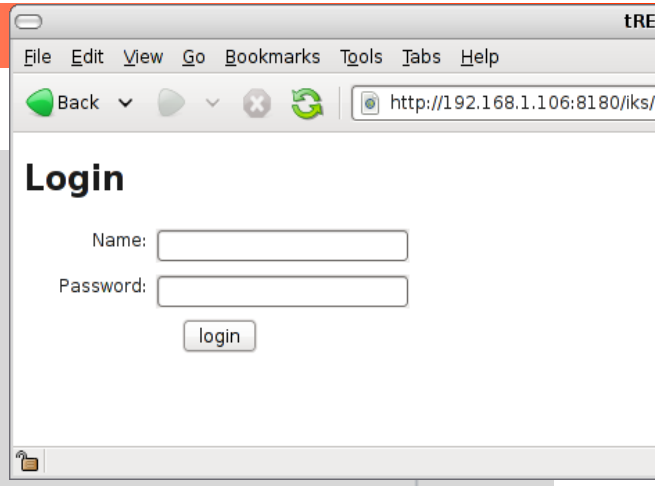
```
1  package demo;
2
3  import java.util.HashMap;
19
20 public class ProfileController extends Controller {
21
22     @Doc("Zobrazenie stranky s uzivatelovym profilom")
23     @DefaultAction
24     @Access(roles = "user")
25     @View(template = "/templates/profile.jsp")
26     public Form profile(
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().getLoggedUser();
35             Profile profile = loadProfile(user);
36             form = new ExtendedProfileForm(profile);
37         }
38
39         return form;
40     }
```

tREST – zabezpečenie, login



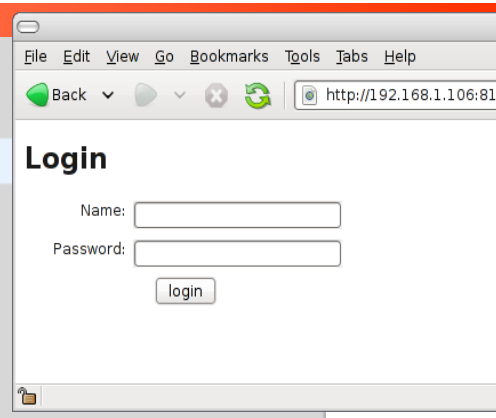
CENTAUR

```java
package demo;

import java.util.HashMap;

public class ProfileController3 extends Controller {

    @Doc("Zobrazenie prihlasovacieho formulara")
    @View(template = "/templates/login.jsp")
    public Form login(@Doc("prihlasovaci formular") LoginForm loginForm)
            throws RedirectException
    {
        if (loginForm.isValid()) {
            String username = (String) loginForm.getValidator("name").getValue(0);
            String password = (String) loginForm.getValidator("password").getValue(0);

            Profile profile = Profiles3.get(username);

            if (profile != null && profile.getPassword().equals(password)) {
                AccessManager am = AccessManager.getInstance();
                am.setLoggedUser(new User(profile.getName(), "user"));
                am.followLoginReferer();
                redirect("profile");
            }
        }

        return loginForm;
    }
```



CENTAUR

```
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
```



JavaScript

tREST (client)

Jozef Sivek

Centaur s.r.o.

JavaScript

- plnohodnotný objektovo orientovaný jazyk
- dynamicky typovaný
- navzdory mnohým implementáciam je veľmi dobre použiteľný (vo verzii ECMAScript edition 3)
- základné konštrukcie a syntax je podobná tej z jazyka C alebo Java

Prečo znovuobjavenie?

JavaScript vo forme akú poznáme je tu už skoro 10 rokov. Tomu však nezodpovedá miera a spôsob jeho použitia posledných 10 rokov.

Príčin je viacero:

- podceňovanie
- nedostatočná alebo zlá dokumentácia
- nepochopenie/neznalosť prototypovania

Jadro veci

Ako chápať JavaScript ako objektový jazyk

JavaScript má objekty.

- Hoci nemá triedy, má konštrukčné funkcie.
- Dedičnosť sa nerealizuje cez dedičnosť tried ale pomocou prototypov.

JavaScript neposkytuje ekvivalentné metódy aké sú známe v jazykoch s triedami a je treba si to plne uvedomiť, inak sa môže stať, že ho budeme viac zneužívať než využívať silu, ktorú nám dáva prototypovanie.

Objekty sú v ňom chápané ako kolekcie párov: klúč-hodnota, teda asociatívne pole, slovník (dictionary) alebo hash mapa.

Výroba objektov

Sú dve cesty ako kreovať objekty

- pomocou literálu

```
var obj = {};
```

- pomocou konštrukčných funkcií

```
var obj = new Object();
//K tomuto spôsobu sa ešte vrátime.
```

Definovanie a pristupovanie k parametrom objektu

Definovať premenné objektu sa dajú priamo v literále alebo kedykoľvek dynamicky priradením hodnoty ku kľúču:

```
> var obj = {"key": "value", "key2": 2.7183};
> obj["key"] //takto môže byť kľúč i rezervované slovo
value
> obj["key2"] = 5;
> obj["key2"]
5

> obj.new_key = function(v){return v;};//bodková notácia
> obj.new_key()
undefined
```

Konštrukčné funkcie

V podstate sú to funkcie tak ako iné:

```
function add(a, b){
   return a + b;
}
```

Ale používame ich spolu s operávorom **new**. Ten spôsobí, že funkcia bude spustená v kontexte novovytvoreného objektu a nám sa vráti tento novovytvorený objekt:

```
function A(param){
   this.value = param;// "this" zosobňuje kontext
}

var a = new A("meno");

> a.value
meno
```

CENTAUR

Prototypovanie

Realizuje zdieľanie funkcionality medzi „rovnakými“ objektami.

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

> var a = new A("Alice");
> a.say_hello()
Alice says hello
```

Jedine v prípade ak funciu voláme `a.say_hello()` alebo `a["say_hello"]()` referencuje „**this**“ aktuálny objekt, v našom prípade „**a**“.

CENTAUR

Bibliografia

- developer.mozilla.org/en/A_re-introduction_to_JavaScript
- www.wikipedia.org

tREST – klient

Peter Rybár
Jozef Sivek

Centaur s.r.o.

tREST klient – vlastnosti

- Implementuje vlastnosti
  - Signál-Slot návrhový vzor
  - Objektové API
    - Vizuálnych komponentov (tabuľky, záložky, zoznamy, …)
    - Formulárových komponentov
    - Formulárových validátorov
  - Podpora AJAX
  - Logovacie API

CENTAUR

tREST klient – dedenie

Žiadne masívne dedenie, orientácia na prebratie existujúcej funkcionality.

```
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);//prebratie z prototypu A

B.prototype.say_hello_upper = function() {
   return this.say_hello().toUpperCase();
}

> var b = new B("Alice");
> b.say_hello_upper()
ALICE SAYS HELLO
```

CENTAUR

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 realizácií logiky „front-end“ aplikácie.
- Koncept spočíva v tom, že objekty (tiež nazívané „widgets“) môžu posielať signály obsahujúce potrebnú informáciu, ktoré sú prijímané funkciami (slotmi).

CENTAUR

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);
```

CENTAUR

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

CENTAUR

tREST – klient príklad

Peter Rybár
Jozef Sivek

Centaur s.r.o.

```
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]+$/, "Bad name format");
43            name_validator.signal_valid.connect(
44                function (value) {
45                    document.getElementById("name_message").innerHTML = "";
46                }
47            );
48            name_validator.signal_invalid.connect(
49                function (error_message, value) {
50                    document.getElementById("name_message").innerHTML = error_message;
51                }
52            );
53
54            // password
55            var password_input = new trest.widgets.forms.TextInput("password");
56            var password_validator = new trest.validators.RegexValidator(
57                    /^.{6,20}$/, "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

Výkon a efektivita

Peter Rybár

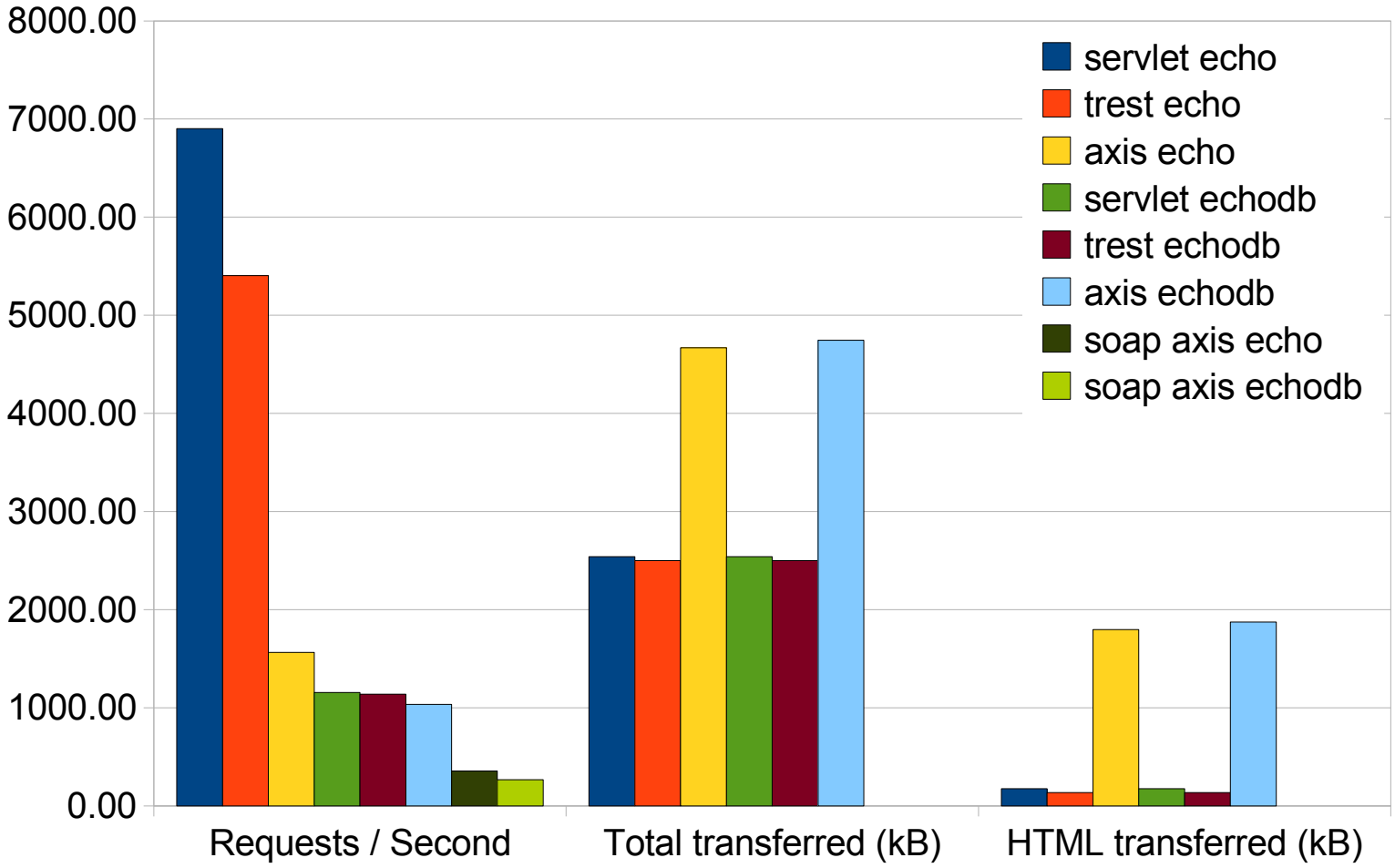
Centaur s.r.o.

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 ťažko hľadajú chyby a ešte ťažšie opravujú
- Znovupoužitie kódu je najlepšie vo forme knižníc
- Špecifickosť kódu je nepriamo úmerná jeho opätovnému použitiu

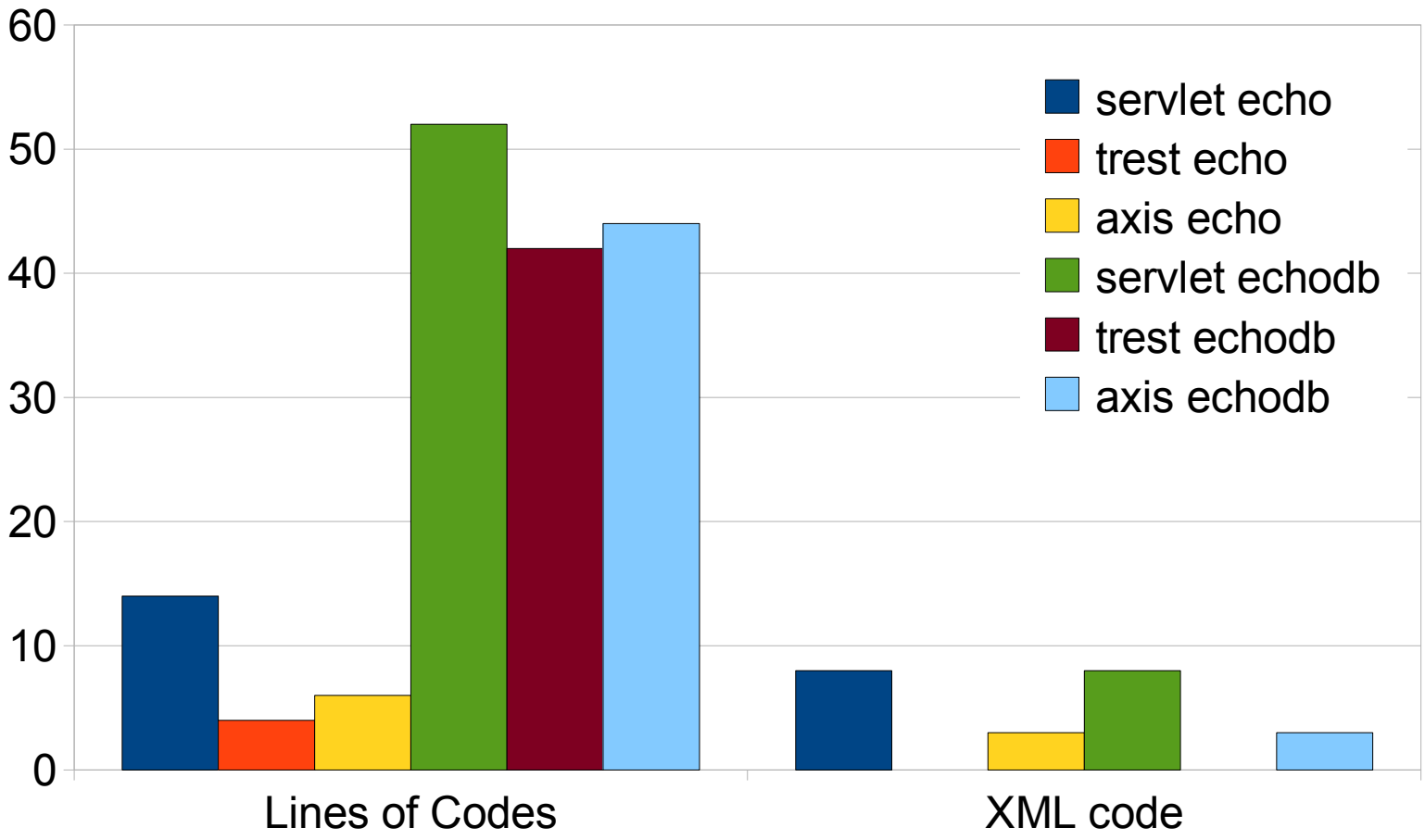
CENTAUR

tREST – výkon, efektivita, testy



CENTAUR

tREST – výkon, efektivita, testy



CENTAUR

Ďakujem

Otázky

Peter Rybár

peter.rybar@centaur.sk

dbuchta@centaur.sk

jsivek@centaur.sk